Replacing Regular Expresions with Parsers

Introduction to Treetop and Polygot

Marcin Raczkowski

RuPy, Poznań 2009

Outline

1. Introduction
   - About RegExps and PEGs
   - Problems with reguar expressions
2. Working with Parsing Expression Grammars
   - Matching and Validation
   - Search and Replace
   - Recursion Handling

Usefull links

- http://github.com/swistak/minitop
- http://treetop.rubyforge.org/
- http://rubyconf2007.confreaks.com/d1t1p5_treetop.html

Regular Expressions and Parsing Expression Grammars

- Both are derived from formal language theory
- Parsing Expression Grammars are generalization of Regular Expressions
- Perl 6 rules - nested and named regular expressions
- Limited context sensitivity

Regular Expressions and Parsing Expression Grammars

- Both are derived from formal language theory
- Parsing Expression Grammars are generalization of Regular Expressions
- Perl 6 rules - nested and named regular expressions
- Limited context sensitivity

Regular Expressions and Parsing Expression Grammars

- Both are derived from formal language theory
- Parsing Expression Grammars are generalization of Regular Expressions
- Perl 6 rules - nested and named regular expressions
- Limited context sensitivity

Regular Expressions and Parsing Expression Grammars

- Both are derived from formal language theory
- Parsing Expression Grammars are generalization of Regular Expressions
- Perl 6 rules - nested and named regular expressions
- Limited context sensitivity

Regular Expressions and Parsing Expression Grammars

- Pairs of parenthesis
- Matching html tags
- Comments and String literals in most programming languages
- Recursion and nested rules

Regular Expressions and Parsing Expression Grammars

- Pairs of parenthesis
- Matching html tags
- Comments and String literals in most programming languages
- Recursion and nested rules

Regular Expressions and Parsing Expression Grammars

- Pairs of parenthesis
- Matching html tags
- Comments and String literals in most programming languages
- Recursion and nested rules

Regular Expressions and Parsing Expression Grammars

- Pairs of parenthesis
- Matching html tags
- Comments and String literals in most programming languages
- Recursion and nested rules

```
1  grammar MiniC
2    rule code
3      (string_literal / s_comment / c_comment / .)+ <Code>
4    end
5    rule string_literal
6      sql / dql
7    end
8    rule dql
9      '"' (ec / (!'"' .))* '"' <StringLiteral>
10   end
11   rule sql
12     "'" (ec / (!"'" .))* "'" <StringLiteral>
13   end
14   rule ec
15     '\\' .
16   end
17   rule c_comment
18     '/*' (!'*/' . )* '*/' <Comment>
19   end
20   rule s_comment
21     '//' (!"\n" .)* <Comment>
22   end
23 end
```

Matching And Validation

```
1  parser = Treetop.load('minic.tt').new
2  file = File.read('test.c').gsub(/\\\n/m, "")
3  if tree = parser.parse(file)
4    puts "YES! Finally!"
5    first_comment = tree.all_elements.
6      detect{|e| e.node_types.include?("Comment") }
7
8    puts first_comment.interval
9  else
10   puts "argh, not again"
11 end
```

Search And Replace

```
1  require 'base'
2  class Treetop::Runtime::SyntaxNode
3    def gsub(node, with=nil, &block)
4      if terminal?
5       text_value
6      elsif node_types.include?(node)
7        with || block.call(self)
8      else
9        elements.map{|e| e.gsub(node, with, &block)}.join("")
10     end
11   end
12 end
13 parser = Treetop.load('minic.tt').new
14 tree = parser.parse(File.read('test.c').gsub(/\\\n/m, ""))
15 puts tree.gsub("Comment", '')
16 tree = parser.parse('a␣"123456789"␣b')
17 puts tree.gsub("StringLiteral"){|n| "t("+n.text_value+")"}
```

Recursion Handling

```
1  grammar List
2    rule list
3      atom more_atoms:(',' atom)* {
4        def atoms; [atom] + more_atoms.elements.map{|m| m.atom}; end
5      }
6    end
7    rule atom
8      '(' list ')' / number
9    end
10   rule number
11     ('-'? [1-9] [0-9]* / '0')
12   end
13 end
```

Recursion Handling

```
1  class Treetop::Runtime::SyntaxNode
2   include Enumerable
3  end
4  parser = Treetop.load_from_string(list_grammar).new
5  if tree = parser.parse('(11,(12,13,14),(15,16))')
6    reverser = lambda{|n|
7      n.atoms.map{|m|
8        m.gsub("list", &reverser)
9      }.reverse.join(",")
10   }
11   puts tree.gsub("list", &reverser) end
```

Summary

- Parsing Expression Grammars are generalization of Regular Expressions.
- PEGs can be used in areas that RegExps can not be used.

- Outlook
  - Minitop
  - better reflection.