Replacing Regular Expresions with Parsers

Introduction to Treetop and Polygot

Marcin Raczkowski

RuPy, Poznań 2009

Outline

Usefull links

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

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

Recursion Handling

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

Recursion Handling

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

Summary

- Parsing Expression Grammars are generalization of Regular Expressions
- They can be used when writing Regular Expression top parse file is hard, or impossible.
- PEGs are useful when you need to handle recursion or nested elements.

- Outlook
  - Work on minitop.
  - Improving treetop with better reflection capabilities.