Practical LPeg

By Mitchell
Lua Workshop 2016

Outline

- Introduction
- Real-world LPeg
  - Syntax highlighting
  - Template engine
  - Domain-Specific Language (DSL)
- Q & A

Introduction

- What is LPeg?
- How is it useful?
  - Lua pattern limitations
  - Regular expression-like features
  - Cherries on top
    - Captures
    - Grammars

Lua Patterns Refresher

- Lua identifier

```
"[%a_][%w_]*" -- name, _G, foo123
```

- Lua comment

```
"%-%-[^\n]*" -- comment
```

- Lua single or double quoted string

```
[=[(['"])[^%1]*%1]=] -- 'string', "another"
```

- Limitations
  - No escaped quotes in strings
  - No “identifier OR string” constructions

Basic LPeg

- Lua identifier

```
(lpeg.R('az', 'AZ') + '_') *
(lpeg.R('az', 'AZ', '09') + '_')^0
```

- Lua comment

```
lpeg.P('--') * (lpeg.P(1) - '\n')^0
```

- Lua single or double quoted string

```
P("'") * (1 - S("\\'") + '\\' * P(1))^0 * P("'") +
P('"') * (1 - S('\\"') + '\\' * P(1))^0 * P('"')
```

- A bit more verbose, but more precise

P: Literal
S: Set
R: Range
*: And
+: Or
^: At least
1: Any single
-: Except

Real-World Problem #1

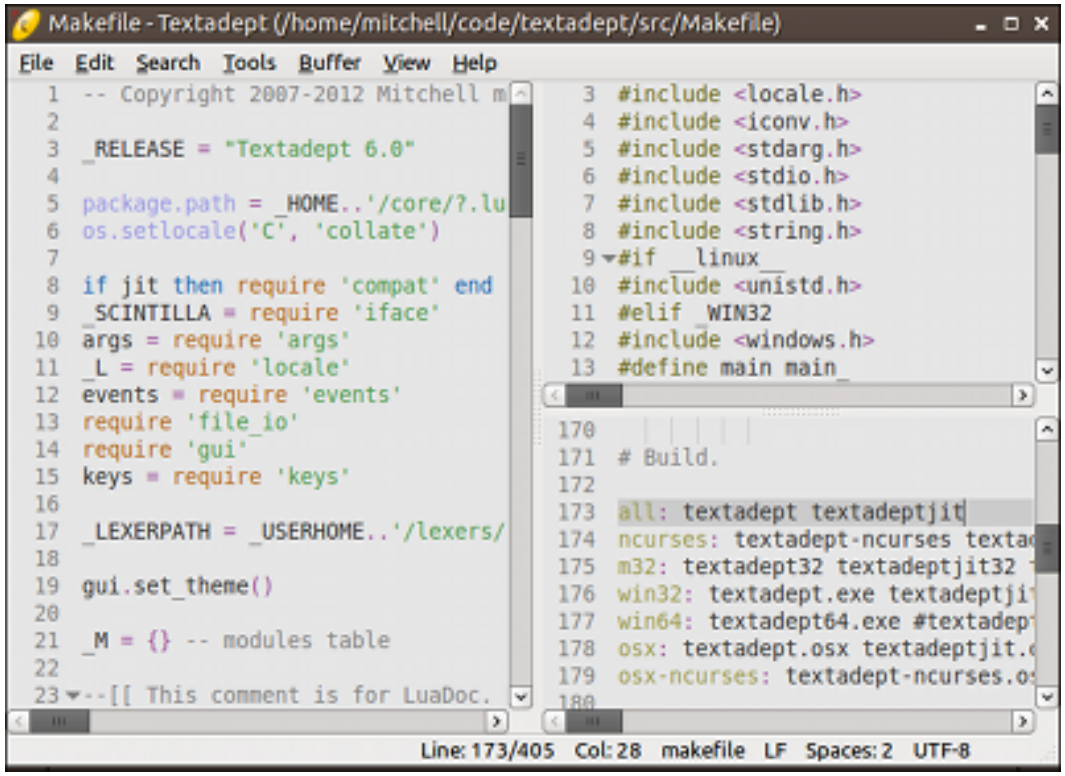
```lua
-- Say hi.
function hello(name)
  print("Hello "..name)
end
```

```lua
-- Say hi.
function hello(name)
  print("Hello "..name)
end
```

```lua
{"comment",    10, -- -- Say hi.
 "whitespace", 11,
 "keyword",    19, -- function
 "whitespace", 20,
 "identifier", 25, -- hello
 "operator",   26, -- (
 "identifier", 30, -- name
 "operator",   31, -- )
 "whitespace", 34,
 "function",   39, -- print
 "operator",   40, -- (
 "string",     48, -- "Hello "
 "operator",   50, -- ..
 "identifier", 54, -- name
 "operator",   55, -- )
 "whitespace", 56,
 "keyword",    59} -- end
```

Syntax Highlighting

- Advanced pattern matching
- Tokens
  - Whitespace
  - Comments
  - Strings
  - Etc.
- Rules
- Grammars



Simple Grammar

```lua
-- Say hi.
function hello(name)
  print("Hello "..name)
end
```

```lua
local P, S, R = lpeg.P, lpeg.S, lpeg.R

local whitespace = S(" \t\r\n\f")^1
local keyword = P("function") + "end"
local function = P("print")
local identifier = (R("az", "AZ") + "_") *
                   (R("az", "AZ", "09") + "_")^0
local string = P('"') * (1 - S('\\"') + "\\" * P(1))^0 * P('"')^-1
local comment = P("--") * (P(1) - "\n")^0
local operator = S("().")

local grammar = (whitespace + keyword + function + identifier
                 string + comment + operator)^1
```

P: Literal
S: Set
R: Range
*: And
+: Or
^: At least
1: Any single
-: Except

Considerations

- Repetition
  - Common patterns
  - Delimited ranges (strings)
  - Keyword lists
- Embedded languages
  - Finite pattern limit

Scintillua

- Syntax highlighting library
- Lexers
  - Single language
  - Embedded languages
- Common LPeg patterns
  - `lexer.space, lexer.nonnewline, lexer.word`
  - `lexer.delimited_range('"')`
  - `lexer.word_match{"function", "end"}`

Sample Lexer for Lua

```
local l = require('lexer')
local M = {_NAME = 'lua'}

local whitespace = l.token(l.WHITESPACE, l.space^1)
local longstring = [...] -- long string pattern
local line_comment = '--' * l.nonnewline^0
local block_comment = '--' * longstring
local comment = l.token(l.COMMENT, block_comment + line_comment)
local sq_str = l.delimited_range("'")
local dq_str = l.delimited_range('"')
local string = l.token(l.STRING, sq_str + dq_str + longstring)

M._rules = {
  {'whitespace', whitespace},
  [...], -- keywords, functions, constants, libraries, identifiers
  {'string', string},
  {'comment', comment},
  [...], -- numbers, labels, operators
} --> compiles to a grammar
```

*: And
+: Or
^: At least

Embedding Lua in HTML

```
local l = require('lexer')
local M = {_NAME = 'elua'}

local html = l.load('html')
local lua = l.load('lua')
local start_tag = l.token('lua_tag', lpeg.P('<?lua') * l.space^1)
local end_tag = l.token('lua_tag', lpeg.P('?>'))
l.embed_lexer(html, lua, start_tag, end_tag)
```

P: Literal	*: And	^: At least

```
<h1><?lua print("Hi!") ?></h1>
```

```
{"element",     5, -- <h1>
 "lua_tag",    10, -- <?lua
 "whitespace", 11,
 "function",   16, -- print
 "operator",   17, -- (
 "string",     22, -- "Hi!"
 "operator",   23, -- )
 "whitespace", 24,
 "lua_tag",    26, -- ?>
 "element",    31} -- </h1>
```

Real-World Problem #2

```
<html>
<head>
  <title>My Webpage</title>
</head>
<body>
<ul id="navigation">
{% for item in navigation %}
  <li>
    <a href="{{ item.href }}">{{ item.caption }}</a>
  </li>
{% endfor %}
</ul>

<h1>My Webpage</h1>
{{ a_variable }}

{# a comment #}
</body>
</html>
```

- Home
- Products
- Contact
- About

My Webpage

13 Oct 2016

Template Engine

- `string.gsub()` on steroids
  - Control structures
  - Data placeholders
    - Optional transformations
  - Environment
- Parse to Abstract Syntax Tree (AST)
- “Walk” the AST & generate output

Processing Loop

```
Matrix:
{% for row in matrix %}
  [{% for value in row %}
    {{ value }},
  {% endfor %}]
{% endfor %}
```

```
{"text", "Matrix:",
 "for", {
   expr = "row in matrix",
   "text", "[",
   "for", {
     expr = "value in row",
     "variable", "value",
     "text", ","
   },
   "text", "]"
 }}
```

+

```
env = {matrix = {{1, 2, 3},
                 {4, 5, 6},
                 {7, 8, 9}}}
```

```
Matrix:
[1,2,3,]
[4,5,6,]
[7,8,9,]
```

Lupa

- Template engine library
  - Clone of Python's Jinja2
  - Text-based output format
    - HTML, XML, CSV, LaTeX, etc.
- 3 delimiter types
  - `{% ... %}` – Statements
  - `{{ ... }}` – Expressions
  - `{# ... #}` – Comments

Sample LPeg Grammar

```
local P, V, C, Ct, Cg = lpeg.P, lpeg.V, lpeg.C, lpeg.Ct, lpeg.Cg
local function node(name, patt) return lpeg.Cc(name) * patt end
```

P: Literal	*: And	^: At least	-: Except
V: Named rule	+: Or	1: Any single	

```
local grammar = Ct(P{
  text = node("text", C((P(1) - "{%" - "{{" - "{#")^1)),
    --> captures {..., "text", "captured text", ...}
  variable = "{{" * node("variable", C((P(1) - "}}")^0)) * "}}",
    --> captures {..., "variable", "expression to eval", ...}
  comment = "{#" * (P(1) - "#}")^0 * "#}",
    --> captures nothing (no node() call); comments ignored
  for_block = P("{% for ") *
              node("for", Ct(Cg((P(1) - "%}")^1, "expression") *
                             " %}" * V("body"))) * "{% endfor %}",
    --> captures {..., "for", {expr = "x in y", ...}, ...}
  body = (V("text") + V("variable") + V("comment") +
          V("for_block"))^1,
  V("body") -- grammar entry point
})
```

Creating and Walking the AST

```lua
local template_text = [[...]]
local ast = lpeg.match(grammar, template_text)

local chunks = {} -- for storing the result
for i = 1, #ast, 2 do
  local node, block = ast[i], ast[i + 1]
  if node == "text" then
    chunks[#chunks + 1] = block
  elseif node == "variable" then
    chunks[#chunks + 1] = eval(block, env)
  elseif node == "for" then
    -- process `block.expression` and add to `chunks`
  end
end
print(table.concat(chunks)) -- the rendered template
```

Error Handling

- Empty renders tell nothing

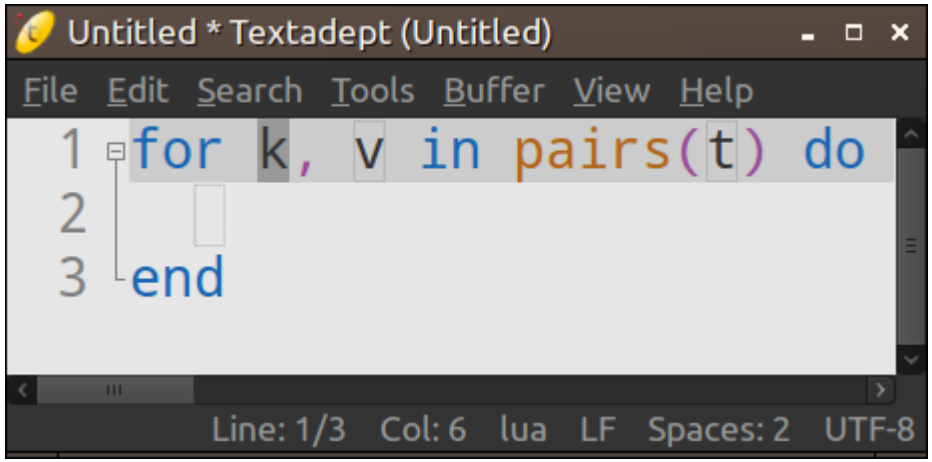
```
local function lpeg_error(errmsg)
  return lpeg.P(function(input, index)
    input = input:sub(1, index)
    local _, line_num = input:gsub("\n", "")
    error(string.format("Parse Error on line %d: %s",
                        line_num + 1, errmsg)
  end)
end

local grammar = lpeg.Ct(lpeg.P{
  ...
  variable = "{{" * node("variable", C((P(1) - "}}")^0)) *
             ("}}" + lpeg_error('"}}" expected')),
    --> Parse Error on line X: "}}" expected
  ...
```

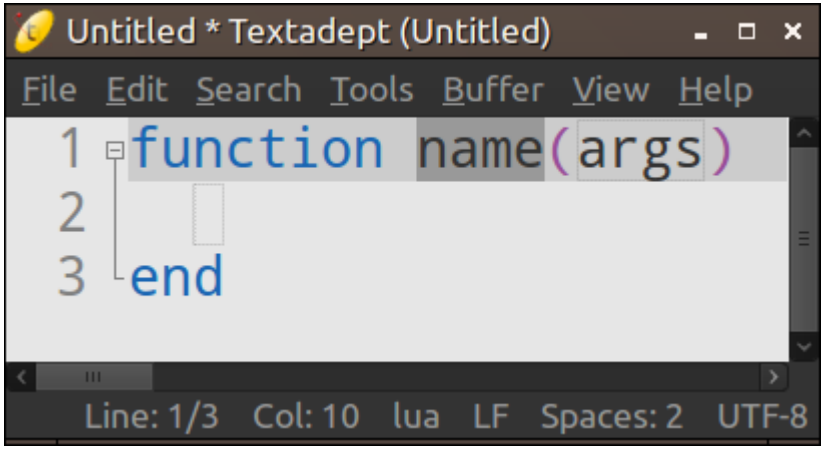
*: And	1: Any single
+: Or	-: Except
^: At least	

Real-World Problem #3

```
snippets.forp = [[
for %1(k), %2(v) in pairs(%3(t)) do
  %0
end]]
```




```
snippets.func = [[
function %1(name)(%2(args))
  %0
end]]
```



Domain-Specific Language

- Similar approach to template engine

```
snippets.func = [[
function %1(name)(%2(args))
  %0
end]]
```

`lpeg.match()` yields:

- `"function "`, `{index = 1, default = "name", position = 10}`
- `"("`, `{index = 2, default = "args", position = 15}`
- `")\n  "`, `{index = 0, position = 23}`
- `"\nend"`

Wrap Up

- LPeg is extremely versatile, robust, and solves real-world problems!
  - Syntax highlighting
  - Template engine
  - Domain-Specific Language

Thank You

- Questions?

Scintillua: http://foicica.com/scintillua

Lupa: http://foicica.com/lupa

Textadept: http://foicica.com/textadept