Parsing complex data formats in LuaTEX with LPEG

Henri Menke

TUG2019: August 9–11, 2019

LPEG

LPEG is a Domain Specific Embedded Language

- Domain: Parsing
- Embedded: Within Lua using operator overloading
- Language: PEG (Parsing Expression Grammar)

Integrated in LuaTEX since the beginning.

Quick Introduction to Lua

All variables are global by default, local variables need the `local` keyword.

```
local x = 1
```

Functions are first class variables

```
function f(...) end    local f = function(...) end
```

Only a single complex data structure, the table

```
local t = { 11, 22, 33, foo = "bar" }
print(t[2], t["foo"], t.foo) -- 22 bar bar
```

If a function argument is a single literal string or table, parentheses can be omitted

```
f("foo")              f"foo"
f({ 11, 22, 33 })     f{ 11, 22, 33 }
```

Ad-hoc parsing

Parse dates of the format 09-08-2019.

```
\newcount\n
\def\isdate#1{\n=0\splitdate#1-\end}
\def\splitdate#1-#2\end{\advance\n by 1
  \ifx\end#1\end\errmessage{field \the\n\space is empty}
  \else\isdigit{#1}\fi
  \ifnum\n>3\errmessage{too many fields}\fi
  \ifx\end#2\end\else\splitdate#2\end\fi}

\def\isdigit#1{\splitdigit#1\end}
\def\splitdigit#1#2\end{%
  \ifnum`#1<`0\else\ifnum`#1>`9
    \errmessage{`#1' is not a digit}
  \fi\fi
  \ifx\end#2\end\else\splitdigit#2\end\fi}
```

Regular expressions

- Starts out innocent. Dates of the format 09-08-2019

```
[0-3][0-9]-[0-1][0-9]-[0-9]{4}
```

- Does not cover all the cases. Explosion of complexity:

```
^(?:(?:31(\/|-|\.)(?:0?[13578]|1[02]))
\1|(?:(?:29|30)(\/|-|\.)(?:0?[1,3-9]
|1[0-2])\2))(?:(?:1[6-9]|[2-9]\d)?\d{
2})$|^(?:29(\/|-|\.)0?2\3(?:(?:(?:1[
6-9]|[2-9]\d)?(?:0[48]|[2468][048]|[
13579][26])|(?:(?:16|[2468][048]|[
3579][26])00))))$|^(?:0?[1-9]|1\d|2[0-
8])(\/|-|\.)(?:(?:0?[1-9])|(?:1[0-2]
))\4(?:(?:1[6-9]|[2-9]\d)?\d{2})$
```

Parsing Expression Grammars

PEG for email (not really)

$$\langle \text{name} \rangle \leftarrow [a-z]+ \ (\text{"."} \ [a-z]+)^*$$

$$\langle \text{host} \rangle \leftarrow [a-z]+ \ \text{"."} \ (\text{"com"}/\text{"org"}/\text{"net"})$$

$$\langle \text{email} \rangle \leftarrow \langle \text{name} \rangle \ \text{"@"} \ \langle \text{host} \rangle$$

Translates almost 1:1 to LPEG

```
local name  = R"az"^1 * (P"." * R"az"^1)^0
local host  = R"az"^1 * P"." * (P"com" + P"org" + P"net")
local email = name * P"@" * host
```

Basic Parsers

- `lpeg.P(string)` Matches `string` exactly

```
lpeg.P("hello") -- matches "hello" but not "world"
```

- `lpeg.P(n)` Matches exactly n characters

```
lpeg.P(1) -- match any single character
lpeg.P(-1) -- match only the end of input
```

- `lpeg.S(string)` Matches any character in `string` (Set)

```
lpeg.S(" \t\r\n") -- match all whitespace
```

- `lpeg.R("xy")` Matches any character between x and y (Range)

```
lpeg.R("09") -- match any digit
lpeg.R("az", "AZ") -- match any ASCII letter
```

Parsing Expressions

Description	PEG	LPEG
Sequence	$e_1 e_2$	`patt1 * patt2`
Ordered choice	$e_1 \mid e_2$	`patt1 + patt2`
Zero or more	$e*$	`patt^0`
One or more	$e+$	`patt^1`
Optional	$e?$	`patt^-1`
And predicate	$\&e$	`#patt`
Not predicate	$!e$	`-patt`
Difference		`patt1 - patt2`

```
P"pizza" * R"09"
-- "pizza4"

P(1) * P":" * R"09"
-- "a:9"
```

Parsing Expressions

Description	PEG	LPEG
Sequence	$e_1 e_2$	`patt1 * patt2`
Ordered choice	$e_1 \mid e_2$	`patt1 + patt2`
Zero or more	e^*	`patt^0`
One or more	$e+$	`patt^1`
Optional	$e?$	`patt^-1`
And predicate	$\&e$	`#patt`
Not predicate	$!e$	`-patt`
Difference		`patt1 - patt2`

```
R"az" + R"09" + S".,;:?!"
-- "a"
-- "9"
-- ";"
-- "+" fails to parse
```

Parsing Expressions

Description	PEG	LPEG
Sequence	$e_1 e_2$	`patt1 * patt2`
Ordered choice	$e_1 \mid e_2$	`patt1 + patt2`
Zero or more	$e*$	`patt^0`
One or more	$e+$	`patt^1`
Optional	$e?$	`patt^-1`
And predicate	$\&e$	`#patt`
Not predicate	$!e$	`-patt`
Difference		`patt1 - patt2`

```
R"az"^0 + R"09"^1
-- "z86", "abcde99", "99"

R"az"^1 + R"09"^1
-- "z86"
-- "abcde99"
-- "99" fails to parse

R"az"^-1 + R"09"^1
-- "z86"
-- "abcde99" fails to parse
-- "99"
```

Parsing Expressions

Description	PEG	LPEG
Sequence	$e_1 e_2$	`patt1 * patt2`
Ordered choice	$e_1 \mid e_2$	`patt1 + patt2`
Zero or more	$e*$	`patt^0`
One or more	$e+$	`patt^1`
Optional	$e?$	`patt^-1`
And predicate	$\&e$	`#patt`
Not predicate	$!e$	`-patt`
Difference		`patt1 - patt2`

```
R"09"^1 * #P";"
-- "86;"
-- "99" fails to parse

P"for" * -(R"az"^1)
-- "for()"
-- "forty" fails to parse
```

Parsing Expressions

Description	PEG	LPEG
Sequence	$e_1 e_2$	`patt1 * patt2`
Ordered choice	$e_1 \mid e_2$	`patt1 + patt2`
Zero or more	e^*	`patt^0`
One or more	$e+$	`patt^1`
Optional	$e?$	`patt^-1`
And predicate	$\&e$	`#patt`
Not predicate	$!e$	`-patt`
Difference		`patt1 - patt2`

```
P"/*" * (1 - P"*/")^0 * P"*/"
-- "/* comment */"

P"helloworld" - P"hell"
-- will never match!
```

Simple Example

```
local lpeg = require"lpeg"
local P, R = lpeg.P, lpeg.R

local input = "cosmic pizza"

local rule = R"az"^1 * P" " * R"az"^1
print(lpeg.match(rule, input) .. " of " .. #input)
```

Output: 13 of 12

Recursive Rules and Grammars

```
local lpeg = require"lpeg"
local P, R, V = lpeg.P, lpeg.R, lpeg.V

local rule = P{"words",
  words = V"word" * P" " * V"word",
  word = R"az"^1,
}
print(rule:match(input) .. " of " .. #input)
```

Output: 13 of 12

Attributes

Operation	Attribute
`lpeg.C(patt)`	The match for `patt`
`lpeg.Ct(patt)`	A table with all captures from `patt`
`lpeg.Cg(patt [, name])`	the values produced by `patt`, optionally tagged with `name`
`lpeg.Cf(patt, func)`	A folding of the captures from `patt`

And a couple of others...

```
local rule = C(R"az"^1)
print(rule:match"pizza")
-- pizza
```

Attributes

Operation	Attribute
`lpeg.C(patt)`	The match for `patt`
`lpeg.Ct(patt)`	A table with all captures from `patt`
`lpeg.Cg(patt [, name])`	the values produced by `patt`, optionally tagged with `name`
`lpeg.Cf(patt, func)`	A folding of the captures from `patt`

And a couple of others...

```
local cell =
   C((1 - P"," - P"\n")^0)
local row =
   Ct(cell * (P"," * cell)^0)
local csv =
   Ct(row * (P"\n" * row)^0)

local t = csv:match[[a,b,c
d,e,f
g,,h]]
```

Attributes

Operation	Attribute
`lpeg.C(patt)`	The match for `patt`
`lpeg.Ct(patt)`	A table with all captures from `patt`
`lpeg.Cg(patt [, name])`	the values produced by `patt`, optionally tagged with `name`
`lpeg.Cf(patt, func)`	A folding of the captures from `patt`

And a couple of others...

```
local key = C(R"az"^1)
local val = C(R"09"^1)

local kv =
    Cg(key * P":" * val) *
        P",^-1
local kvlist =
    Cf(Ct"" * kv^0, rawset)

kvlist:match"foo:1,bar:2"
```

Actually Useful Parsers

```
local lpeg = require"lpeg"
local P, R, S, V = lpeg.P, lpeg.R, lpeg.S, lpeg.V
local number = P{"number",
    number = (V"int" * V"frac"^-1 * V"exp"^-1) / tonumber,
    int = V"sign"^-1 * (R"19" * V"digits" + V"digit"),
    digits = V"digit" * V"digits" + V"digit",
    digit = R"09",
    sign = S"+-",
    frac = P"." * V"digits",
    exp = S"eE" * V"sign"^-1 * V"digits",
}
local x = number:match("+123.456e-78")
print(x .. " " .. type(x))
```

Output: 1.23456e-76 number

Complex Data Formats: JSON

```
-- optional whitespace
local ws = S" \t\n\r"^0

-- match a literal string surrounded by whitespace
local lit = function(str)
    return ws * P(str) * ws
end

-- match a literal string and synthesize an attribute
local attr = function(str,attr)
    return ws * P(str) / function() return attr end * ws
end
```

Complex Data Formats: JSON

```
-- JSON grammar
local json = P{
    "object",

    value =
        V"null_value" +
        V"bool_value" +
        V"string_value" +
        V"real_value" +
        V"array" +
        V"object",
```

Complex Data Formats: JSON

```
null_value =
    attr("null", nil),

bool_value =
    attr("true", true) + attr("false", false),

string_value =
    ws * P'"' * C((P'\\"' + 1 - P'"')^0) * P'"' * ws,

real_value =
    ws * number * ws,
```

Complex Data Formats: JSON

```
    array =
        lit"[" * Ct((V"value" * lit","^-1)^0) * lit"]",

    member_pair =
        Cg(V"string_value" * lit":" * V"value") * lit","^-1,

    object =
        lit"{" * Cf(Ct"" * V"member_pair"^0, rawset) * lit"}"
}
```

Complex Data Formats: JSON

```
local lpeg = require"lpeg"
local C, Cf, Cg, Ct, P, R, S, V =
    lpeg.C, lpeg.Cf, lpeg.Cg, lpeg.Ct, lpeg.P, lpeg.R, lpeg.S, lpeg.V

-- number parsing
local number = P{"number",
    number = (V"int" * V"frac"^-1 * V"exp"^-1) / tonumber,
    int = V"sign"^-1 * (R"19" * V"digits" + V"digit"),
    digits = V"digit" * V"digits" + V"digit",
    digit = R"09",
    sign = S"+-",
    frac = P"." * V"digits",
    exp = S"eE" * V"sign"^-1 * V"digits",
}

-- optional whitespace
local ws = S" \t\n\r"^0

-- match a literal string surrounded by whitespace
local lit = function(str)
    return ws * P(str) * ws
end

-- match a literal string and synthesize an attribute
local attr = function(str,attr)
    return ws * P(str) / function() return attr end * ws
end

-- JSON grammar
local json = P{
    "object",
    value =
        V"null_value" +
        V"bool_value" +
        V"string_value" +
        V"real_value" +
        V"array" +
        V"object",

    null_value =
        attr("null", nil),

    bool_value =
        attr("true", true) + attr("false", false),

    string_value =
        ws * P'"' * C((P'\\"' + 1 - P'"')^0) * P'"' * ws,

    real_value =
        ws * number * ws,

    array =
        lit"[" * Ct((V"value" * lit","^-1)^0) * lit"]",

    member_pair =
        Cg(V"string_value" * lit":" * V"value") * lit","^-1,

    object =
        lit"{" * Cf(Ct"" * V"member_pair"^0, rawset) * lit"}"
}
```

JSON Parser in Action

```
local example = [[{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}]]
local m = json:match(example)
print(m.menu.popup.menuitem[2].value)
```

Output: Open

Thank you!

Questions?