ConTEXt

PEG Grammars

John Stewart

September 10, 2014

The PEG module produces typeset Parsing Expression Grammars (PEGs) in a ConTEXt document. The module supports all of the standard PEG syntax, as well as a couple of extensions to better describe certain grammars. A PEG grammar can be described within a `peg` block, started with `\startpeg` and closed with `\stoppeg`. Within the block the PEG can be written in the plain text syntax described in the original PEG paper, with extensions. The output will be typeset such that the productions within the same block are aligned around the left arrow operator.

The following example uses the PEG module to typeset PEG grammar itself:

```
\placepeg[][]{Hierarchical syntax}
{\startpeg
  # Hierarchical syntax
  Grammar <- Spacing Definition+ EndOfFile
  Definition <- Identifier LEFTARROW Expression

  Expression <- Sequence (SLASH Sequence)*
  Sequence <- Prefix*
  Prefix <- (AND / NOT)? Suffix
  Suffix <- Primary (QUESTION / STAR / PLUS)?
  Primary <- Identifier !LEFTARROW /\n
             OPEN Expression CLOSE /\n
             Literal / Class / DOT
\stoppeg}
\placepeg[][]{Lexical syntax}
{\startpeg
  # Lexical syntax
  Identifier <- IdentStart IdentCont* Spacing
  IdentStart <- [a-zA-Z_]
  IdentCont <- IdentStart / [0-9]

  Literal <- ['] (!['] Char)* ['] Spacing /\n
             ["] (!["] Char)* ["] Spacing
  Class <- '[' (!']' Range)* ']' Spacing
  Range <- Char '-' Char / Char
  Char <- '\\' [nrt'"\[\]\\] /\n
          '\\' [0-2][0-7][0-7] /\n
          '\\' [0-7][0-7]? /\n
          !'\\' .

  LEFTARROW <- '<-' Spacing
  SLASH <- '/' Spacing
  AND <- '&' Spacing
  NOT <- '!' Spacing
  QUESTION <- '?' Spacing
  STAR <- '*' Spacing
```

```
  PLUS <- '+' Spacing
  OPEN <- '(' Spacing
  CLOSE <- ')' Spacing
  DOT <- '.' Spacing

  Spacing <- (Space / Comment)*
  Comment <- '#' (!EndOfLine .)* EndOfLine
  Space <- ' ' / '\t' / EndOfLine
  EndOfLine <- '\r\n' / '\n' / '\r'
  EndOfFile <- !.
\stoppeg}
\placefloats
```

$$
\begin{aligned}
\text{Grammar} &\longleftarrow \text{Spacing Definition+ EndOfFile} \\
\text{Definition} &\longleftarrow \text{Identifier LEFTARROW Expression} \\
\text{Expression} &\longleftarrow \text{Sequence (SLASH Sequence)*} \\
\text{Sequence} &\longleftarrow \text{Prefix*} \\
\text{Prefix} &\longleftarrow \text{(AND / NOT)? Suffix} \\
\text{Suffix} &\longleftarrow \text{Primary (QUESTION / STAR / PLUS)?} \\
\text{Primary} &\longleftarrow \text{Identifier !LEFTARROW /} \\
&\qquad \text{OPEN Expression CLOSE /} \\
&\qquad \text{Literal / Class / DOT}
\end{aligned}
$$

Grammar 1 Hierarchical syntax

```
Identifier ⟵ IdentStart IdentCont*  Spacing
IdentStart ⟵ [a − zA − Z_]
IdentCont ⟵ IdentStart / [0 − 9]
Literal ⟵ ['] (!['] Char)*  ['] Spacing /
          ["] (!["] Char)*  ["] Spacing
Class ⟵ “[” (!“]” Range)*  “]” Spacing
Range ⟵ Char “-” Char / Char
Char ⟵ “\\” [nrt'"\[\]\\] /
       “\\” [0 − 2] [0 − 7] [0 − 7] /
       “\\” [0 − 7] [0 − 7]?  /
       !“\\” .
LEFTARROW ⟵ “<-” Spacing
SLASH ⟵ “/” Spacing
AND ⟵ “&” Spacing
NOT ⟵ “!” Spacing
QUESTION ⟵ “?” Spacing
STAR ⟵ “*” Spacing
PLUS ⟵ “+” Spacing
OPEN ⟵ “(” Spacing
CLOSE ⟵ “)” Spacing
DOT ⟵ “.” Spacing
Spacing ⟵ (Space / Comment)*
Comment ⟵ “#” (!EndOfLine .)*  EndOfLine
Space ⟵ “  ” / “\t” / EndOfLine
EndOfLine ⟵ “\r\n” / “\n” / “\r”
EndOfFile ⟵ !.
```

Grammar 2 Lexical syntax

PEG Extensions

Several extensions are available for use with the PEG module. First, the identifiers for non-terminals have been expanded to allow Unicode letter characters, as well as the en-dash (-) character in the non-initial position.

A new non-terminal has been made available which reflects the common practice of mak-

ing non-formal descriptions of the strings which match to certain non-terminals. This is the “description” non-terminal, which can be indicated by wrapping the descriptive text with angle brackets.

```
\startpeg
  Whitespace <- <Any Unicode whitespace character of class WS>
\stoppeg
```

$$\text{Whitespace} \longleftarrow \langle \textit{Any Unicode whitespace character of class WS} \rangle$$

An additional escape sequence is made available for characters. In addition to standard escaped characters, (such as \n), this module introduces the descriptive Unicode escape. The form of this escape is \x(<codepoint>:<description>). The codepoint value should be up to four characters with the character’s code point, and the description is an informal description of the character (such as the Unicode character code point, and the subscript description. This is a pretty-print alternative to the standard PEG code point escape format of \NNN and \NN.

```
\startpeg
  Whitespace <- ' ' / '\x{000A:LINE FEED}'
\stoppeg
```

$$\text{Whitespace} \longleftarrow \text{“ ”} \;/\; \text{“}U^{000A}{}_{\textit{LINE FEED}}\text{”}$$

A special line feed symbol can also be included in your grammar. This symbol is not typeset as part of the grammar itself. It will add a hard line break to your grammar, allowing the definition to continue on the next line aligned to the start of the definition from the previous line. The symbol for a line feed is the literal string \n.

```
\startpeg
  MultilineDefinition <- "return" / \n "function"
\stoppeg
```

$$\text{MultilineDefinition} \longleftarrow \text{“}\texttt{return}\text{”} \;/\; \text{“}\texttt{function}\text{”}$$

```
\writestatus{loading}{Loading the PEG Grammar module}
\registerctxluafile{t-peg.lua}{1.0.0}
\unprotect
\definesystemvariable{peg}
```

Define the namespace and environment for grammars.

```
\definenamespace[peg][
  type=module,
  name=PEG,
  setup=list]
\installsimplecommandhandler \????peg {peg} \????peg
```

```
\installparameterhandler \????peg {peg}
\installdefinehandler \????peg {peg} \????peg
\definefloat[peg][pegs]

\setupheadtext[\s!en][peg=Grammar]
\setupheadtext[\s!en][pegs=Grammars]
\setuplabeltext[\s!en][peg=Grammar ]
```

Define some commands to use in the default setup for rendering PEG constructs.

```
\def\defaultpeggrammarstart{\startformula\startalign[n=3,right,center,left]}
\def\defaultpeggrammarstop{\stopalign\stopformula}
\def\defaultpegdefinitionnt#1{\pegparameter{identifiercommand}{#1}}
\def\defaultpegdefinition#1#2{\NC #1 \NC \longleftarrow \NC #2 \NR }
\def\defaultpegchoice#1#2{#1\;/\;#2}
\def\defaultpegsequence#1#2{#1\;#2}
\def\defaultpeggroup#1{(#1)}
\def\defaultpegprefix#1{\text{#1}}
\def\defaultpegsuffix#1{\unskip\text{#1}}
\def\defaultpegliteral#1{\text{\quotation{\tt #1}}}
\def\defaultpegclass#1{[#1]}
\def\defaultpegsimplerange#1{\text{\tt #1}}
\def\defaultpegrange#1#2{\pegparameter{simplerangecommand}{#1}-\pegparameter{simple
\def\defaultpegidentifier#1{\text{#1}}
\def\defaultpegdescription#1{\langle\text{\em #1}\rangle }
\def\defaultpegcomplexchar#1#2{\text{\em\rm U\high{#1}\low{#2}}}
\def\defaultpegdot{\text{\tt .}}
\def\defaultpeglinefeed{\NR\NC\NC\NC }
```

Define the default setup for PEG typesetting. These environment parameters can be customized to change how grammars are typeset.

```
\setuppeg[
  grammarstartcommand=\defaultpeggrammarstart,
  grammarstopcommand=\defaultpeggrammarstop,
  definitionntcommand=\defaultpegidentifier,
  definitioncommand=\defaultpegdefinition,
  choicecommand=\defaultpegchoice,
  sequencecommand=\defaultpegsequence,
  groupcommand=\defaultpeggroup,
  prefixcommand=\defaultpegprefix,
  suffixcommand=\defaultpegsuffix,
  literalcommand=\defaultpegliteral,
  classcommand=\defaultpegclass,
  simplerangecommand=\defaultpegsimplerange,
  rangecommand=\defaultpegrange,
  descriptioncommand=\defaultpegdescription,
```

```
    identifiercommand=\defaultpegidentifier,
    complexcharcommand=\defaultpegcomplexchar,
    dotcommand=\defaultpegdot,
    linefeedcommand=\defaultpeglinefeed
  ]
```

Define the buffer environment.

```
  \def\startpeg
    {\dostartbuffer
      [peg]
      [startpeg]
      [stoppeg]}
  \def\stoppeg{
    \catcode`\#=12
    \ctxlua{thirddata.peg(buffers.getcontent('peg'))}
  }

  \protect
  \writestatus{loading}{The PEG Grammar module was loaded successfully}
  \endinput
```