

luaTeX

A user's perspective

Aditya Mahajan

July 29, 2009

What is luaTeX

`www.lua+tex.org` says

- luaTeX is an extended version of pdfTeX using lua as an **embedded scripting language**
- Main objective is to provide an **open and configurable** variant of TeX which is backward compatible with pdfTeX



Translation

Programming in TeX
made easy



How to use lua in ConTeXt

- Call lua from TeX

- ▷ `\ctxlua{...}`
- ▷ `\startluacode ... \stopluacode`

- Call TeX from lua

- ▷ `tex.print(...)`
- ▷ `context(...)`
- ▷ `context.csname(...)` **experimental**



Simple things
are simple



Example: Pick a random option

Encode your Name and Surname as a

```
\startluacode
  local a = {'null-terminated', 'dollar-terminated', 'Pascal'}
  context('%s string', a[math.random(1,3)])
\stopluacode
```

Vyatcheslav Yatskovsky on ntg-context



Example: Convert decimal to binary

Perform logical AND, OR, and XOR of the following pair of hexadecimal numbers:

```
\startluacode
  local n = math.random(10,255)
  local m = math.random(10,255)
  context("%X, %X", n, m)
\stopluacode
```

Vyatcheslav Yatskovsky on ntg-context



You can parse
input without
exploding your brain



Example: Parsing

`\molecule{...}` should type the chemical symbol "correctly"

```
\unexpanded\def\molecule%  
  {\bgroup  
  \catcode`\_=\active \uccode`\~=\`\_ \uppercase{\let~\chemlow}%  
  \catcode`\^=\active \uccode`\~=\`\^ \uppercase{\let~\chemhigh}%  
  \dostepwiserecurse {65}{90}{1}  
    {\catcode \recurselevel = \active  
    \uccode`\~=\recurselevel  
    \uppercase{\edef~{\noexpand\finishchem  
      \rawcharacter{\recurselevel}}}}}%  
  \uccode`\~=\`F \uppercase{\def~{\finishchem F\fluortrue}}}%  
  \catcode`\-=\active \uccode`\~=\`\- \uppercase{\def~{--}}}%  
  \domolecule }  
  
|||||
```

Snippet from mhchem.sty by Martin Hensel

```
\def\ce#1{\mhchem@ce@xiii{\mhchem@ce@viii#1 \mhchem@END\mhchem@ENDEND}}
\def\mhchem@ce@viii#1 #2\mhchem@ENDEND{%
  \ifx\mhchem@END#2%
    \ifx\@empty#1\@empty \else
      \mhchem@ce@x#1\mhchem@END\mhchem@ENDEND%
    \fi
  \else%
    \mhchem@ce@x#1\mhchem@END\mhchem@ENDEND%
    \space\mhchem@ce@viii#2\mhchem@ENDEND%
  \fi}
... ..
\def\mhchem@ce@x#1#2\mhchem@ENDEND { ... }
\def\mhchem@ce@xi#1#2\mhchem@ENDEND{ ... }
\def\mhchem@ce@xii#1\mhchem@END { ... }
\def\mhchem@ce@xiii#1{ ... }
```

LuaTeX has an
inbuilt parser
lpeg



Adapted example from Wolfgang Schuster

```
local lowercase = lpeg.R("az")
local uppercase = lpeg.R("AZ")
... ..
local leftbrace = lpeg.P("{")
local rightbrace = lpeg.P("}")
local nobrace = 1 - (leftbrace + rightbrace)
local nested = lpeg.P { leftbrace * (cname + sign + nobrace
                        + lpeg.V(1))^0 * rightbrace }
... ..
local content = lpeg.Cs(cname + nested + sign + any)
local subscript = lpeg.P("_")
local superscript = lpeg.P("^")
... ..
... ..
```



Adapted example from Wolfgang Schuster

```
local lowhigh      = lpeg.Cc("\\lohi{%s}{%s}")
      * subscript  * content * superscript * content / format
local highlow      = lpeg.Cc("\\hilo{%s}{%s}")
      * superscript * content * subscript   * content / format
local low          = ... ..
local high         = ... ..
      ... ..
local parser       = lpeg.Cs((cname + lowhigh + highlow +
                              low + high + sign + any)^0)
function chemicals.molecule(str) return parser:match(str) end
```

and then

```
\def\molecule#1{\ctxlua{commands.molecule(\!!bs#1!!es)}}
```



Example: Parsing math operators

- Does an operator have subscripts and superscripts?
- Get subscripts and superscripts from

▷ `\command`

▷ `\command` $_$ `{...}`

▷ `\command` $^$ `{...}`

▷ `\command` $_$ `{...}`

▷ `\command` $^$ `{...}` $_$ `{...}`

▷ `\command` $_$ `{...}` $^$ `{...}`

▷ `\command` `\limit` ...

▷ `\command` `\nolimit` ...



Snippet from mathtools.sty by Morton Høgholm

```
\newcommand*\smashoperator[2][lr]{
  \def\MT_smp_use:NNNNN {\@nameuse{MT_smp_smash_#1:NNNNN}}
  \toks@{#2}
  \expandafter\MT_smp_get_args:wwwNnNn
    \the\toks@\@nil\@nil\@nil\@nil\@nil\@nil\@nil
}
... ..
\def\MT_smp_mathop:n {\mathop}
\def\MT_smp_limits: {\limits}
... ..
\MH_new_boolean:n {smp_one}
\MH_new_boolean:n {smp_two}
... ..
```



Snippet from mathtools.sty by Morton Høgholm

```
\def\MT_smap_get_args:wwwNnNn #1#2#3#4#5#6#7\@@nil{%  
  \begingroup  
    \def\MT_smap_arg_A: {#1} \def\MT_smap_arg_B: {#2}  
    \def\MT_smap_arg_C: {#3} \def\MT_smap_arg_D: {#4}  
    \def\MT_smap_arg_E: {#5} \def\MT_smap_arg_F: {#6}  
    \def\MT_smap_arg_G: {#7}  
    ...    ...    ...    ...    ...    ...  
    ...    ...    ...    ...    ...    ...
```



Snippet from mathtools.sty by Morton Høgholm

```
... ..  
\if_meaning:NN \MT_smp_arg_A: \MT_smp_mathop:n  
  \if_meaning:NN \MT_smp_arg_C:\MT_smp_limits:  
    \def\MT_smp_final_arg_A:{#1{#2}}%  
    \if_meaning:NN \MT_smp_arg_D: \@nnil \else:  
      \MH_set_boolean_T:n {smp_one}  
      \MH_let:NwN \MT_smp_final_arg_B: \MT_smp_arg_D:  
      \MH_let:NwN \MT_smp_final_arg_C: \MT_smp_arg_E:  
      \if_meaning:NN \MT_smp_arg_F: \@nnil \else:  
        \MH_set_boolean_T:n {smp_two}  
        \MH_let:NwN \MT_smp_final_arg_D: \MT_smp_arg_F:  
        \edef\MT_smp_final_arg_E:  
          {\xandafter\MT_smp_remove_nil_vi:N \MT_smp_arg_G: }  
      \fi:  
... ..
```

Contrast this with the previous lua parser

```
local lowhigh      = lpeg.Cc("\\lohi{%s}{%s}")
                    * subscript * content * superscript * content / format
local highlow      = lpeg.Cc("\\hilo{%s}{%s}")
                    * superscript * content * subscript * content / format
local low          = lpeg.Cc("\\low{%s}")
                    * subscript * content / format
local high         = lpeg.Cc("\\high{%s}")
                    * superscript * content / format
... ..
... ..
local parser       = lpeg.Cs((cname + lowhigh + highlow
                             + low + high + sign + any)^0)
```



Example: Parsing – Calculator math

```
\usemodule[calcmath]  
...  
\calcmath{2/(sqrt(pi)) int(0,∞, exp(-x^2)) dx = 1}  
...
```

$$\frac{2}{\sqrt{\pi}} \int_{-\infty}^{\infty} e^{-x^2} dx = 1$$

Proof of concept: x-calcmath module



You can do
arithmetic without
using an abacus



Divide two numbers: pgfmath library > 100 LOC

```
\pgfmathdeclarefunction{divide}{2}{%
  \begingroup%
    \pgfmath@x=#1pt\relax%
    \pgfmath@y=#2pt\relax%
    \let\pgfmath@sign=\pgfmath@empty%
    \ifdim\pgfmath@y=0pt\relax%
      \pgfmath@error{You've asked me to divide `#1' by `#2', %
        but I cannot divide any number by `#2'}%
    \fi%
    \afterassignment\pgfmath@xa%
    \expandafter\c@pgfmath@counta\the\pgfmath@y\relax%
    % If y is an integer, use TeX arithmetic.
    \ifdim\pgfmath@xa=0pt\relax%
      \divide\pgfmath@x by\c@pgfmath@counta\relax%
      \edef\pgfmathresult{\pgfmath@tonumber{\pgfmath@x}}%
      \let\pgfmath@next=\pgfmathdivide@@@%
    \else%
```



Divide two numbers: pgfmath library > 100 LOC

```
% Simple long division.
\ifdim\pgfmath@x<0pt\relax%
  \def\pgfmath@sign{-}%
  \pgfmath@x=-\pgfmath@x%
\fi%
\ifdim\pgfmath@y<0pt\relax ... .. \fi%
\pgfmath@ya=\pgfmath@y%
\c@pgfmath@counta=0\relax%
\ifdim\pgfmath@x>\pgfmath@ya%
  \ifdim\pgfmath@ya<1638.4pt\relax%
    \pgfmathmultiply@dimenbyten\pgfmath@ya%
    \ifdim\pgfmath@ya>\pgfmath@x%
      \pgfmathdivide@dimenbyten\pgfmath@ya%
      \c@pgfmath@counta=0\relax%
    \else%
      ... .. Repeat four times ... ..
    \fi\fi\fi\fi\fi\fi\fi\fi\fi
```



Divide two numbers: pgfmath library > 100 LOC

```
% If y < 1pt use reciprocal function.
\ifdim\pgfmath@y<1pt\relax%
  \ifdim\pgfmath@y<.00007pt\relax%
    \pgfmath@error{The result of `#1/#2' is too big for me}{}%
  \fi%
  \pgfmathreciprocal@{\pgfmath@tonumber{\pgfmath@y}}%
  \pgfmath@x=\pgfmathresult\pgfmath@x%
  \edef\pgfmathresult{\pgfmath@tonumber{\pgfmath@x}}%
  \let\pgfmath@next=\pgfmathdivide@@@%
\else%
  \pgfmath@y=\pgfmath@ya%
  \def\pgfmathresult{}%
  \let\pgfmath@next=\pgfmathdivide@@@%
\fi%
\fi%
\pgfmath@next%
```

```
}
```



Trigonometric functions : pgfmath library

```
\pgfmathdeclarefunction{sin}{1}{%
  % Let #1 = a.b
  % Then  $\sin(\#1) \approx (1-b)*\sin(a) + b*\sin(a+1)$ 
  \begingroup%
    \expandafter\pgfmath@x#1pt\relax%
    % ... Get x to be between 0 and 180 ...
    % Now  $0 \leq x < 179$ . So split x into integer and decimal...
    \afterassignment\pgfmath@xa%
    \expandafter\c@pgfmath@counta\the\pgfmath@x\relax%
    % ...if #1 is an integer, don't do anything fancy.
    \ifdim\pgfmath@xa=0pt%
      \expandafter\pgfmath@x\csname pgfmath@cos@
        \the\c@pgfmath@counta\endcsname pt\relax%
    \else
      ... ..
      \expandafter\advance\expandafter\pgfmath@x
        \csname pgfmath@cos@\the\c@pgfmath@counta\endcsname\pgfmath@xa%
    \fi%
  \endgroup}
```

Trigonometric functions : pgfmath library

```
\def\pgfmath@def#1#2#3{\expandafter\def\csname pgfmath@#1@#2\endcsname{#3}}
```

```
... ..
```

```
\pgfmath@def{cos}{0}{1.00000}
```

```
\pgfmath@def{cos}{1}{0.99985}
```

```
\pgfmath@def{cos}{2}{0.99939}
```

```
\pgfmath@def{cos}{3}{0.99863}
```

```
... ..
```

```
... ..
```

```
\pgfmath@def{cos}{178}{-0.99939}
```

```
\pgfmath@def{cos}{179}{-0.99985}
```

```
\pgfmath@def{cos}{180}{-1.00000}
```

```
\pgfmath@def{cos}{181}{-0.99985}
```

```
... ..
```

```
... ..
```

Similar tables for atan and acos

```
... ..
```

```
... ..
```



Arithmetic in luaTeX

```
\pgfmathdeclarefunction{divide}{2}%  
  {\def\pgfmathresult{\cxtlua{context(#1/#2)}}}
```

```
\pgfmathdeclarefunction{sin}{1}%  
  {\def\pgfmathresult{\cxtlua{context(math.sin(#1*2*pi/360))}}}
```



You can write loops
without worrying
about expansion



Simple loops

How can you typeset

(+)	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Question on ntg-context



Natural tables in ConTeXt

```
\setupTABLE[each][each][width=2em,height=2em,align={middle,middle}]  
\setupTABLE[r][1][background=color,backgroundcolor=gray]  
\setupTABLE[c][1][background=color,backgroundcolor=gray]
```

```
\bTABLE  
  \bTR \bTD $(+)$ \eTD \bTD 1 \eTD \bTD 2 \eTD ... .. \eTR  
  \bTR \bTD 1 \eTD \bTD 2 \eTD \bTD 3 \eTD ... .. \eTR  
  \bTR \bTD 2 \eTD \bTD 3 \eTD \bTD 4 \eTD ... .. \eTR  
  ... ..  
  ... ..  
\eTABLE
```



Write a simple loop

```
start_table
start_table_row
  table_element("(+)")
  for y in [1..6] do
    table_element(y)
stop_table_row
for x in [1..6] do
  start_table_row
  table_element(x)
  for y in [1..6] do
    table_element(x+y)
  end
stop_table_row
end
stop_table
```

Simple loops can be difficult in TeX

```
\bTABLE
  \bTR
    \bTD $(+)$ \eTD
  \dorecurese{6}
    {\bTD \recurselevel \eTD}
  \eTR
\dorecurese{6}
  {\bTR
    \bTD \recurselevel \eTD
    \edef\firstrecurselevel{\recurselevel}
  \dorecurese{6}
    {\bTD \the\numexpr\firstrecurselevel+\recurselevel \eTD}
  \eTR}
\eTABLE
```



Expansion,
expansion, expansion



Sprinkle `\expandafter` according to taste

```
\bTABLE
\bTR
  \bTD $(+)$ \eTD
  \dorecurse{6}
    {\expandafter \bTD \recurselevel \eTD}
  \eTR
\dorecurse{6}
  {\bTR
    \edef\firstrecurselevel{\recurselevel}
    \expandafter\bTD \recurselevel \eTD
  \dorecurse{6}
    {\expandafter\bTD
      \the\numexpr\firstrecurselevel+\recurselevel\relax
      \eTD}
  \eTR}
```

ConTeXt Lua Document – cld files

```
context.bTABLE()
context.bTR()
  context.bTD(); context("$+$"); context.eTD();
  for y = 1,6 do
    context.bTD(); context(y); context.eTD() ;
  end
context.eTR()
for x = 1,6 do
  context.bTR()
  context.bTD(); context(x); context.eTD() ;
  for y = 1,6 do
    context.bTD(); context(x+y); context.eTD() ;
  end
  context.eTR()
end
context.eTABLE()
```

Metapost : Labels in loops

Draw a grid of points with labels

(0,5) (1,5) (2,5) (3,5) (4,5) (5,5)

(0,4) (1,4) (2,4) (3,4) (4,4) (5,4)

(0,3) (1,3) (2,3) (3,3) (4,3) (5,3)

(0,2) (1,2) (2,2) (3,2) (4,2) (5,2)

(0,1) (1,1) (2,1) (3,1) (4,1) (5,1)

(0,0) (1,0) (2,0) (3,0) (4,0) (5,0)



Labels in loops – luaTeX to the rescue

```
\startluacode
context.startMPcode()
context("numeric u ; u = 2cm ;")
for x = 0,5 do
  for y = 0,5 do
    context("drawdot(" .. x .. "*u," ..
              y .. "*u"); \n" ) ;
    context("label.bot(" ..
              "btex (" .. x .. "," .. y .. ") etex," ..
              "(" .. x .. "*u," .. y .. "*u)) ;")
  end
end
context.stopMPcode()
\stopluacode
```



You can do everything in TeX. After all, TeX is a Turing machine ... and programming in TeX is as convenient as programming a Turing machine.

Reinhard Kotucha on texhax, quoting Klaus Lagally



You can do everything in TeX. After all, TeX is a Turing machine ... and programming in TeX is as convenient as programming a Turing machine.

Reinhard Kotucha on texhax, quoting Klaus Lagally

luaTeX is changing this!

