Grzegorz “myamlak” Murzynowski
parcat.eu

GM-Scenarios two years later

A complete madness.
Turing-complete. (Or not?)

BachoTEX April 29, 2017
Grzegorz “myamlak” Murzynowski
parcat.eu

GM-Scenarios two years later

A complete madness.
Turing-complete. (Or not?)

or
how did I
from the l3expan spirit
conceive and bear a monster.

BachoTEX April 29, 2017
1. expl3 in general

2. l3expan

3. GM-Scenarios, a proper extension to l3expan
Partt 1

expl3 in general
expl3 in general

\LaTeX3 in general (AFAIUwMHM)
expl3 in general

\LaTeX3 in general (AFAIUwMHM)

- The “three” layers:
  - the very document,
  - document design
    [specs. of margins, fonts, columns, headings &c.],
  - the \texttt{implementation} [for the two above],
  - [the TeX implementation of the
    “implementation layer”].
\LaTeX{}3 in general (AFAIUwMHM)

- The “three” layers:
  - the very document,
  - document design
    [specs. of margins, fonts, columns, headings &c.],
  - the implementation [for the two above],
  - [the \TeX{} implementation of the
    “implementation layer”].

- expl3 – “a normal programming language”
\LaTeX{}3 in general (AFAIUwMHM)

- The “three” layers:
  - the very document,
  - document design
    [specs. of margins, fonts, columns, headings &c.],
  - the implementation [for the two above],
  - [the \TeX{} implementation of the
    “implementation layer”].

- expl3 – “a normal programming language”
  “...almost \TeX{}-independent”
expl3 in general

\LaTeX3 in general (AFAIUwMHM)

- The “three” layers:
  - the very document,
  - document design
    [specs. of margins, fonts, columns, headings &c.],
  - the implementation [for the two above],
  - [the \TeX implementation of the “implementation layer”].

- expl3 – “a normal programming language”
  “…almost \TeX-independent”

Almost.
expl3 – “a normal programming language” concepts
expl3 in general

expl3 – “a normal programming language” concepts:

- the characters allowed in names
  - “_” and “:” re-catcoded to 11, “letter”
- naming conventions, name spaces
- functions

- data types, variables & constants
- scope of the variables and of the resp. assignments
- blanks ignored
- iterators, loops, mappings
- consistent brace syntax (no “open if” problem)
expl3 – “a normal programming language” concepts:

- the characters allowed in names
  “_” and “:” re-catcoded to 11, “letter”
- naming conventions, name spaces
- functions
- function variants
- data types, variables & constants
- scope of the variables and of the resp. assignments
- blanks ignored
- iterators, loops, mappings
- consistent brace syntax (no “open if” problem)
expl3 in general

“blanks ignored”

\ifnum 1=0
  1a
\else
  0z
\fi
expl3 in general

“blanks ignored”

\ifnum 1=0
  1a
\else
  0z
\fi

\int_compare:nNnTF
  {1}={0}
  {1a}{0z}
expl3 in general

Defining “functions” and –
here comes l3expan –
their “variants”

\cs_new:Nn \__module_function\textquotesingle{name:nn} {
\begin{verbatim}
  \<do sth. about #1 & #2 >
\end{verbatim}
}

expl3 in general

Defining “functions” and – here comes \texttt{l3expan} – their “variants”

\verb|\cs_new:Nn \__module_function\textquotesingle{name}\text{nn} {|
\verb|  \texttt{<do sth. about #1 \\ & #2 >}}|
\verb|}|\verb|

\verb|\cs_generate_variant:Nn \__module_function\textquotesingle{name}\text{nn} |
\verb|{Vx} \% \texttt{--> \__module_function\textquotesingle{name}\text{Vx}}|
Part 2

I3expan
\cs_new:Nn \__module_function\textquote Single Quote \_name\textquote Single Quote {nn {
    \textless \text{do sth. about #1 & #2} \textgreater 
}
\cs_new:Nn \__module_function'name:nn {
  <do sth. about #1 & #2 >
}

> \exp_args:NVx=undefined.
\cs_new:Nn \__module_function\texttt{name:nn} {
    \texttt{<do sth. about #1 & #2 >}
}

\texttt{\exp_args:NVx=undefined.}

\cs_generate_variant:Nn \__module_function\texttt{name:nn} {Vx}  % --> \__module_function\texttt{name:Vx}
\cs_new:Nn \_\_module_function\_name:nn { 
  \textit{<do sth. about \#1 \& \#2 >} 
}

> \exp_args:NVx=undefined.

\cs_generate_variant:Nn \_\_module_function\_name:nn {Vx} % --> \_\_module_function\_name:Vx

> \_\_module_function\_name:Vx=\texttt{protected\_long macro:} 
-\texttt{\exp_args:NVx \_\_module_function\_name:nn} .
\cs_new:Nn \_module_function\textbackslash name:n {  
  \<do sth. about #1 & #2 >
}

> \exp_args:NVx=undefined.

\cs_generate_variant:Nn \_module_function\textbackslash name:n {Vx} % --> \_module_function\textbackslash name:Vx

> \_module_function\textbackslash name:Vx=\protected\long macro:  
  ->\exp_args:NVx \_module_function\textbackslash name:nn .

> \exp_args:NVx=\protected\long macro:  
  ->\::V \::x \::: .
\expandafter, and why doesn't it work with a \textit{<balanced text>}

\expandafter \def \expandafter \foo \expandafter
\{\bar \textit{<\ldots more stuff>}}
\expandafter, 
and why doesn't it work with a <balanced text>

\expandafter \def \expandafter \foo \expandafter 
{\bar <...more stuff>}

and hence \::o

\cs_new:Npn \::o #1 \::: #2#3 { 
\exp_after:wN \__exp_arg_next:nnn 
\exp_after:wN {#3} {#1} {#2} }
\expandafter, and why doesn't it work with a \textit{balanced text}?

\expandafter \def \expandafter \foo \expandafter {\bar ...
...more stuff} and hence \::o

\cs_new:Npn \::o \::: #1 \::: #2\#3 {
  \exp_after:wN \__exp_arg_next:nnn
  \exp_after:wN {#3} {#1} {#2} }

\cs_new:Npn \__exp_arg_next:nnn #1#2#3 {
  #2 \::: { #3 {#1} } }

...and other \::\textgreek{a} \::\textgreek{b}

\cs_new:Npn \::\textgreek{a} \::\textgreek{b} \::\textgreek{c} \::\textgreek{v}
\expandafter, and why doesn't it work with a \texttt{<balanced text>}

\expandafter \def \expandafter \foo \expandafter
{\bar <...more stuff>}

and hence \texttt{::o}

\texttt{\cs_new:Npn ::o #1 ::: #2#3 { \exp_after:wN __exp_arg_next:nnn \exp_after:wN {#3} {#1} {#2} }}

\texttt{\cs_new:Npn __exp_arg_next:nnn #1#2#3 { \#2 :::: { #3 {#1} } }}

...and other \texttt{::[]}’s.

\texttt{::N ::n}
\texttt{::o ::x ::f}
\texttt{::c ::V ::v}
\texttt{\LaTeX \! 2\varepsilon \ style:}

\begin{verbatim}
\newtoks \l@aux@args@toks
\newtoks \l@auxA@toks
\newtoks \l@auxB@toks
\% \l@auxA@toks = {{⟨arg.3T⟩}}
\l@auxB@toks = {{⟨arg.3F⟩}}
\% \edef \aux@macro {\if<condition> \the \l@auxA@toks \else \the \l@auxB@toks \fi}
\l@aux@args@toks \expandafter {\aux@macro} \%
"{⟨arg.3_⟩}"
\% \edef \aux@macro {⟨arg.2⟩}
\l@aux@args@toks \expandafter \expandafter \expandafter \expandafter\aux@macro \the \l@aux@args@toks
% \% and, finally,
\expandafter \mod_foo:nnn \the \l@aux@args@toks
\end{verbatim}
\textbf{\LaTeX\ 2ε style:}

\begin{verbatim}
\newtoks \l@aux@args@toks
\newtoks \l@auxA@toks
\newtoks \l@auxB@toks
%
\l@auxA@toks = {{⟨arg.3T⟩}}
\l@auxB@toks = {{⟨arg.3F⟩}}
%
edef \aux@macro {
  \if<condition>
    \the\l@auxA@toks
  \else
    \the\l@auxB@toks
  \fi
}
\l@aux@args@toks \expandafter \{\aux@macro \}
% "⟨arg.3_⟩"
%
edef \aux@macro {⟨arg.2⟩}
\l@aux@args@toks \expandafter \expandafter \expandafter \{\aux@macro \the\l@aux@args@toks \}
% "⟨arg.2-ed⟩⟨arg.3_⟩"
%
\expandafter \def \expandafter \aux@macro {\the \arg@i@int % remember [...]
\l@aux@args@toks \expandafter \expandafter \expandafter \expandafter \expandafter \expandafter \{\expandafter \aux@macro \the\l@aux@args@toks \}
% [...]
% and, finally,
\expandafter \mod_foo:nnn \the \l@aux@args@toks
\end{verbatim}

\textbf{expl3 style:}

\begin{verbatim}
\cs_generate_variant:Nn __mod_foo:nnn {Vxf}
\__mod_foo:Vxf
\arg_i_int {⟨arg.2⟩}
{ \__⟨condition⟩:TF {⟨arg.3T⟩}{⟨arg.3F⟩} }
\end{verbatim}
Do we really always need to generate a variant?
Do we really always need to generate a variant?

```
\cs_generate_variant:Nn \__mod_foo:nnn {Vxf}
```

> `\exp_args:NVxf=\protected\long macro:
  ->\::V \::x \::f \::: .`
Do we really always need to generate a variant?

\cs_generate_variant:Nn \_mod_foo:nnn {Vxf}

> \exp_args:NVxf=\protect\long macro:
  \rightarrow \V \x \f .

so –

\V \x \f . \_mod_foo:nnn \langle the “raw” args. \rangle
The \texttt{\textbackslash :tn} macros of l3expan

- bring programming in \TeX\ an abstraction level up
- shorten the code by replacing recurring schemas with variants or \texttt{\textbackslash :tn}'s
- and thus decrease the chance of a bug.
The \texttt{:\texttt{::}} macros of \texttt{l3expan}

- bring programming in \TeX\ an abstraction level up
- shorten the code by replacing recurring schemas with variants or \texttt{:\texttt{::}}'s
- and thus decrease the chance of a bug.

But they

- apply just one “elementary operation” to an argument
- act only on separate arguments.
The \l3expan\ macros of l3expan
▶ bring programming in \TeX\ an abstraction level up
▶ shorten the code by replacing recurring schemas with variants or \l3expan\'s
▶ and thus decrease the chance of a bug.

But they
▶ apply just one “elementary operation” to an argument
▶ act only on separate arguments.

“Typical” examples [in my \TeX\ life], not handled by l3expan:
▶ reverse the order of two arguments
▶ double an argument
▶ hit an argument with exactly two \expandafter\’s
Partt 3

GM-Scenarios, a proper extension to l3expan
a GM-Scenario
GM-Scenarios, a proper extension to l3expan

a General Meta-Scenario
GM-Scenarios, a proper extension to l3expan

a General Meta-Scenario

\::V \::x \::f \::: \__mod_foo:nnn <"raw" args.>
GM-Scenarios, a proper extension to l3expan

a General Meta-Scenario

\::V \::x \::f \::: \__mod_foo:nnn <“raw” args.>

V x f : \__mod_foo:nnn <“raw” args.>
GM-Scenarios, a proper extension to \texttt{l3expan}

\begin{quote}
\textit{a General Meta-Scenario}
\end{quote}

\begin{verbatim}
\:::V \:::x \:::f \:::\::: \__mod_foo:nnn \textless \textquote{raw} \text{ args.}\>
V x f : \__mod_foo:nnn \textless \textquote{raw} \text{ args.}\>
\#: I V x f : \__mod_foo:nnn \textless \textquote{raw} \text{ args.}\>
\end{verbatim}
GM-Scenarios, a proper extension to I3expan

Monster
GM-Scenarios, a proper extension to l3expan

Monster

Reverse order of two arguments, double an argument
GM-Scenarios, a proper extension to l3expan

Monster

Reverse order of two arguments, double an argument $\mapsto$ arbitrary permutation with repetitions
GM-Scenarios, a proper extension to l3expan

Monster

Reverse order of two arguments, double an argument $\mapsto$

arbitrary permutation with repetitions

Applying multiple ops. to one argument, within a
permutation or outside
GM-Scenarios, a proper extension to l3expan

Monster

Reverse order of two arguments, double an argument $\mapsto$

arbitrary permutation with repetitions

Applying multiple ops. to one argument, within a

permutation or outside

Various styles of declaring a permutation
GM-Scenarios, a proper extension to \textsc{l3expan}

Monster: a small & simple sample
GM-Scenarios, a proper extension to l3expan

Monster: a small & simple sample

\onslide <2> {
\Hi \HMD: \
\nointerlineskip
\smash{\box 0}
\prevdepth
}
GM-Scenarios, a proper extension to \l3expan

Monster: a small & simple sample

\onslide <2> { 
\::: Hi ⋮ 11O : 
\nointerlineskip 
\smash{\box 0} 
\prevdepth 
\}

\__\__\_prepare\ˈτ{c}w 
\:::H \:::i 
\:::\::_prepare\ˈFSM\ˈw 
\"F#1 \"I \"F#1 \"in\ˈF: \"BØ \"I 
\q\__\__\_FSM\ˈcraw\ˈstart 1\__\__\_τ\ˈyield:w 
\::: {}.
GM-Scenarios, a proper extension to l3expan

Monster: a small & simple sample

\onslide <2> { 
::: Hi ♮11☉ :
\nointerlineskip
\smash{\box 0}
\prevdepth 
}

::: \_prepareˈτ\{ς\}:w
:::H :::i
:::_prepareˈFSM\#w
"F\#1 "I "F\#1 "in′F: "BØ "I
\q\_\_\_FSM′craw"start 1\_\_\_\_τ"yield:w
::: {}. 

\nointerlineskip \smash {\box 0}
\prevdepth 4.234219pt.
GM-Scenarios, a proper extension to l3expan

Monster unleashed: argument substitutions & references
Monster unleashed: argument substitutions & references

\::: Hi ♮110 ::
\\nointerlineskip
\smash{\box 0}
GM-Scenarios, a proper extension to l3expan

Monster unleashed: argument substitutions & references

\[ Hi \omega 1 \ [ 1\Omega=\omega ] 1 41 213 : \]
\begin{verbatim}
\nointerlineskip
\smash{\box 0}
1 \prevdepth
2 {\hrule width \hsize height }
3 \relax
4 \showtokens
\omega
\end{verbatim}
Monster unleashed: argument substitutions & references

\[\text{Hi} \quad \omega \quad 1 \quad \lbrack \quad 1 \theta = \# 1 \quad \rbrack \quad 1 \quad 41 \quad 2R : \]
\text{nointerlineskip}
\text{smash{\text{box 0}}}
1 \text{ prevdepth}
2 \text{ \hrule width \hsize height \#1 \relax}
3 \text{ }$
4 \text{ showtokens}$
GM-Scenarios, a proper extension to l3expan

Monster unleashed: argument substitutions & references

\[\text{Hi} \omega 1 [1\Omega=\#1] 1 41 2^{*13} :\]
\texttt{\noindent \smash{\box 0}}
1 \texttt{\prevdepth}
2 \texttt{\hrule width \hspace{height}}
3 \texttt{\relax}
4 \texttt{\showtokens}
\omega
Monster tamed (soon in the future):

- easily convertible into a preprocessor [on the docStrip level]
  ...at least about what’s expandable
  ...including the FSM/BDSM parts
  (i.e., the permutation-with-repetition-and-bracing),
- and with only the “official” Unicode characters
  [i.e., no PUA of my design, and no special font with them necessary],
- and (possibly) with only ASCII characters.
GM-Scenarios, a proper extension to l3expan

Symbols of my invention