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 **implementation** [for the two above],
  - [the TeX implementation of the
    “implementation layer”].

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"

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"

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 in general

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 in general

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
```

$\mapsto$

```
%
\int_compare:nNnTF
{1}={0}
{1a}{0z}
```

expl3 in general

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

```
\cs_new:Nn  \__module_function'name:nn {
   <do sth. about #1 & #2 >
}
```

expl3 in general

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

```
\cs_new:Nn  \__module_function'name:nn {
   <do sth. about #1 & #2 >
}

\cs_generate_variant:Nn  \__module_function'name:nn
       {Vx}  % -->  \__module_function'name:Vx
```

Partt 2

l3expan

l3expan

```
\cs_new:Nn  \__module_function'name:nn {
   <do sth. about #1 & #2 >
}
```

l3expan

```
\cs_new:Nn  \__module_function'name:nn {
   <do sth. about #1 & #2 >
}

> \exp_args:NVx=undefined.
```

l3expan

```
\cs_new:Nn  \__module_function'name:nn {
   <do sth. about #1 & #2 >
}

> \exp_args:NVx=undefined.

\cs_generate_variant:Nn  \__module_function'name:nn
       {Vx}  % -->  \__module_function'name:Vx
```

l3expan

```
\cs_new:Nn  \__module_function'name:nn {
   <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=\protected\long macro:
->\exp_args:NVx \__module_function'name:nn .
```

l3expan

```
\cs_new:Nn  \__module_function'name:nn {
   <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=\protected\long macro:
->\exp_args:NVx \__module_function'name:nn .

> \exp_args:NVx=\protected\long macro:
->\::V \::x \::: .
```

l3expan

`\expandafter`,
and why doesn't it work with a <balanced text>

```
\expandafter \def \expandafter \foo \expandafter
      {\bar <…more stuff>}
```

l3expan

\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} }
```

l3expan

`\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} }

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

l3expan

`\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} }

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

…and other `\::⍰`'s.

```
\::N \::n
\::o \::x \::f
\::c \::V \::v
```

l3expan

LATEX 2ε style:

```
\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 } %
        "{⟨arg.2-ed⟩}{⟨arg.3_⟩}"
```

```
\expandafter \def \expandafter
\aux@macro \expandafter{%
  \the \arg@i@int %  remember [...]
}
\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
```

l3expan

LaTeX 2ε style:

```
\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 } %
        "{⟨arg.2-ed⟩}{⟨arg.3_⟩}"
```

```
\expandafter \def \expandafter
\aux@macro \expandafter{%
  \the \arg@i@int %  remember [...]
}
\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
```

expl3 style:

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

\__mod_foo:Vxf
\arg_i_int  {⟨arg.2⟩}
 { \__⟨condition⟩:TF  {⟨arg.3T⟩}{⟨arg.3F⟩}  }
```

l3expan

Do we really always need to generate a variant?

l3expan

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 \::: .
```

l3expan

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 \::: .
```

so –

```
\::V \::x \::f \:::  \__mod_foo:nnn ⟨the “raw” args.⟩
```

l3expan

The \::⍰ macros of l3expan

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

l3expan

The `\::⍰` macros of l3expan

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

But they

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

l3expan

The `\::⍰` macros of l3expan

- bring programming in TEX an abstraction level up
- shorten the code by replacing recurring schemas with variants or `\::⍰`'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

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 l3expan

a General Meta-Scenario

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

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

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

GM-Scenarios, a proper extension to l3expan

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 l3expan

Monster: a small & simple sample

GM-Scenarios, a proper extension to l3expan

Monster: a small & simple sample

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

GM-Scenarios, a proper extension to l3expan

Monster: a small & simple sample

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

\__⁝⁝_prepare'τ⦃ς⦄:w
\::H \::i
\⁝⁝_prepare'FSM'♮:w
\¨F#1 \¨I \¨F#1 \¨in'F: \¨BĐ \¨I
\q__⁝⁝_FSM'craw῀start 1\__⁝⁝_τ῀yield:w
 \::: {}.
```

GM-Scenarios, a proper extension to l3expan

Monster: a small & simple sample

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

\__⁝⁝_prepare'τ⟅ς⟆:w
\::H \::i
\⁝⁝_prepare'FSM'♮:w
\¨F♯1 \¨I \¨F♯1 \¨in'F: \¨BĐ \¨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

GM-Scenarios, a proper extension to l3expan

Monster unleashed: argument substitutions & references

```
\⁝⁝  Hi  ♮11Đ      :
  \nointerlineskip
  \smash{\box 0}
```

GM-Scenarios, a proper extension to l3expan

Monster unleashed: argument substitutions & references

```
\⁝⁝  Hi   ω 1 ⟦ 1Đ≕1 ⟧ 1  41  213 :
  \nointerlineskip
  \smash{\box 0}
  1  \prevdepth
  2  {\hrule width \hsize height }
  3  \relax
  4  \showtokens
  ω
```

GM-Scenarios, a proper extension to l3expan

Monster unleashed: argument substitutions & references

```
\⁝⁝  Hi  ω 1 ⟦ 1Đ≕⁜1 ⟧ 1  41  2R :
  \nointerlineskip
  \smash{\box 0}
  1  \prevdepth
  2  {\hrule width \hsize height \⁜1 \relax}
  3  {}
  4  \showtokens
  ω
```

GM-Scenarios, a proper extension to l3expan

Monster unleashed: argument substitutions & references

```
\⁝⁝  Hi  ω 1 ⟦ 1Đ≕⊧1 ⟧ 1  41  2※13 :
  \nointerlineskip
  \smash{\box 0}
  1  \prevdepth
  2  {\hrule width \hsize height }
  3  \relax
  4  \showtokens
  ω
```

GM-Scenarios, a proper extension to l3expan

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