Grzegorz “myamlak” Murzynowski
parcat.eu

A Night at the Opera (in three acts)

or: the expl3 language
through the eyes of a neophyte

incl.: Dr. Frank’n’Furter:
“What have I done??”

BachoTEX April 29, 2015

1. expl3 in general
2. l3expan as a TeX pearls necklace
3. GMOA, a proper extension to l3expan

Act 1

expl3 in general

expl3 in general

Some general assumptions and conventions of the expl3 language.

expl3 in general

Some general assumptions and conventions of the expl3 language.

or: “Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb”

expl3 in general

Some general assumptions and conventions of the expl3 language.

or: “Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb”

...and: “Who’s afraid of Virginia Woolf”.

expl3 in general

Some general assumptions and conventions of the expl3 language.

or: “Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb”

...and: “Who’s afraid of Virginia Woolf”.

[answer: the programmers]

expl3 in general

- “_” and “:” made letters, catcode 11;

expl3 in general

- "_" and ":" made letters, catcode 11;
- blanks made ignored, catcode 9;

expl3 in general

- “_” and “:” made letters, catcode 11;
- blanks made ignored, catcode 9;
- “~” made whitespace (char 0x20 catcode 10).

expl3 in general

don't worry with blanks, they're ignored!

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

expl3 in general

don't worry with blanks, they're ignored!

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

but such code shouldn't appear in an expl3 env.
instead,

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

expl3 in general: “function”

- privacy,
- module (namespace),
- signature

```
\cs_if_eq:NNF

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

\__exp_arg_next:Nnn
--> \cs_new:Npn \__exp_arg_next:Nnn #1#2#3 { #2 \::: { #3 #1 }
      }
```

expl3 in general

Is the signature somehow checked?

expl3 in general

Is the signature somehow checked?
only by some defining functions:

```
\cs_new:Nn \__module_function_name:nNNn
{…#1 …#3… #4…
#5 % ! Illegal parameter number
…}
```

expl3 in general

Is the signature somehow checked?
only by some defining functions:

```
\cs_new:Nn \__module_function_name:nNNn
{…#1 …#3… #4…
#5 % ! Illegal parameter number
…}
```

but mostly just a convention, e.g.:

```
\hbox:n \bgroup <contents of the box>\egroup
```

expl3 in general: defining “functions” and their “variants”

```
\cs_generate_variant:Nn \cs_new:Npn { Npo } % --> \cs_new:Npo
\cs_generate_variant:Nn \cs_new_nopar:Npn { cpo } % -->
      \cs_new_nopar:cpo
```

expl3 in general: a “variable” [a datum]

- the “scope”: l/g/c,
- privacy,
- module (namespace),
- type

expl3 in general

introducing and setting a datum

```
\<type>_new:N
\<type>_const:Nn
```

expl3 in general

introducing and setting a datum

```
\<type>_new:N
\<type>_const:Nn
```

NO `\bool_const:Nn`.

expl3 in general

introducing and setting a datum

```
  \<type>_new:N
  \<type>_const:Nn
```

NO `\bool_const:Nn`.
"What would it be for if it's either True or False?"

expl3 in general

introducing and setting a datum

```
  \<type>_new:N
  \<type>_const:Nn
```

NO `\bool_const:Nn`.
"What would it be for if it's either True or False?"
my answer: "Why would you need a `\c__columns_number_int` constant if its either One, or Two, or Three?"

expl3 in general

many things I do not use actually (yet):

- handling of the boxes,
- “coffins”

expl3 in general

many things I do not use actually (yet):

- handling of the boxes,
- “coffins”

apparently no handling of `\kerns`.

expl3 in general

What I admire:

- clarity of code,
- “naturalness” of the names,
- almost complete lack of exceptions (like Esperanto),

expl3 in general

What I admire:

- clarity of code,
- “naturalness” of the names,
- almost complete lack of exceptions (like Esperanto),
- many beautiful tricks, especially about expansion of arguments
  (discussed in Act 2)

expl3 in general: what I don't like

- trying to separate the "executables" from "data" (not quite working IMO and a step back against The Foundations)

- the "Hide your TEX so I can forget you have it at all" approach

expl3 in general: what I don't like

- trying to separate the "executables" from "data"
(not quite working IMO and a step back against The Foundations)

- the
"Hide your TEX so I can forget you have it at all"
approach [cf. marquis de Sade's "Speech to women"].

expl3 in general: what I don't like

- trying to separate the "executables" from "data"
(not quite working IMO and a step back against The
Foundations)
"just don't like".

- the
"Hide your TEX so I can forget you have it at all"
approach [cf. marquis de Sade's "Speech to women"].
"objection".

expl3 in general: conclusion: I recommend

despite of some (meta-)reservations,
I would recommend expl3 to TeX programmers:

- decrease of bugs number thanks to
  - iterators,
  - signatures,
  - types &c;
- better readability:
  "It looks almost like a normal programming language!"

expl3 in general: conclusion: I recommend

despite of some (meta-)reservations,
I would recommend expl3 to TeX programmers:

- decrease of bugs number thanks to
  - iterators,
  - signatures,
  - types &c;
- better readability:
  "It looks almost like a normal programming language!"

Thanks be to the LaTeX 3 Team.

Act 2

l3expan as a TEX pearls necklace

l3expan as a TEX pearls necklace

assume we are given

- #1 a number specification: a `\count` register, a `\numexpr` or an explicit number specification in TEX,
- #2 an arbitrary balanced text;
- #3 in two versions in an expandable $\langle\text{condition}\rangle$al

l3expan as a TEX pearls necklace

assume we are given

- #1 a number specification: a `\count` register, a `\numexpr` or an explicit number specification in TEX,
- #2 an arbitrary balanced text;
- #3 in two versions in an expandable ⟨condition⟩al

and we wish to apply them to some macro `\@@mod@foo`, but

- #1 as its computed ("rendered") value,
- #2 fully expanded with `\edef`,
- #3 either its True or False version depending on the ⟨condition⟩

l3expan as a TeX pearls necklace

in LaTeX 2$_\varepsilon$ we'd write (I did, actually):

```
\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
}
```

... »»

l3expan as a TEX pearls necklace

```
\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_⟩}"
```

… »»

l3expan as a TEX pearls necklace

```
\expandafter \def \expandafter
\aux@macro \expandafter{%
  \the \arg@i@int %  remember it can be a numexpr, we don't know
        the num.of tokens
}
\l@aux@args@toks
 \expandafter\expandafter\expandafter
 {\expandafter \aux@macro
   \the\l@aux@args@toks } % "{⟨val.of arg.1⟩}{⟨arg.2-ed⟩}{⟨arg.3__⟩}"
%
%  and, finally,
\expandafter \mod_foo:nnn \the \l@aux@args@toks
```

about 30 lines of code

l3expan as a TEX pearls necklace

while in expl3:

```
\::V \::x \::f \:::
\__mod_foo:nnn
\arg_i_int  {⟨arg.2⟩}
 { \__⟨condition⟩:TF  {⟨arg.3T⟩}{⟨arg.3F⟩}  }
```

l3expan as a TEX pearls necklace

while in expl3:

```
\::V \::x \::f \:::
\__mod_foo:nnn
\arg_i_int  {⟨arg.2⟩}
 { \__⟨condition⟩:TF  {⟨arg.3T⟩}{⟨arg.3F⟩}  }
```

or, if we'll use this often:

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

and at the point of use:

```
\__mod_foo:Vxf ⟨the args.⟩
```

l3expan as a TEX pearls necklace

pearl 0, the biggest and blackest:
it's so much shorter and clearer.

l3expan as a TEX pearls necklace

pearl 0, the biggest and blackest:
it's so much shorter and clearer.

pearl 1:
it's expandable:

```
\cs_new:Npn \__exp_arg_next:Nnn
 #1#2#3 { #2 \::: { #3 #1 } }
…
\cs_new:Npn \::: #1 {#1}
…
\cs_new:Npn \::p #1 \::: #2#3# { #1 \::: {#2#3} }
```

»»

l3expan as a TeX pearls necklace: it's expandable!

```
\cs_new:Npn \::c #1 \::: #2#3
  { \exp_after:wN \__exp_arg_next:Nnn
 \cs:w #3 \cs_end: {#1} {#2} }
\cs_new:Npn \::o #1 \::: #2#3
  { \exp_after:wN \__exp_arg_next:nnn \exp_after:wN {#3} {#1}
      {#2} }
…
```

l3expan as a TEX pearls necklace: it's expandable!

```
\cs_new:Npn \::c #1 \::: #2#3
  { \exp_after:wN \__exp_arg_next:Nnn
 \cs:w #3 \cs_end: {#1} {#2} }
\cs_new:Npn \::o #1 \::: #2#3
  { \exp_after:wN \__exp_arg_next:nnn \exp_after:wN {#3} {#1}
      {#2} }
…
```

except the `x`, of course.

l3expan as a TEX pearls necklace: it's expandable!

```
\cs_new:Npn \::c #1 \::: #2#3
  { \exp_after:wN \__exp_arg_next:Nnn
 \cs:w #3 \cs_end: {#1} {#2} }
\cs_new:Npn \::o #1 \::: #2#3
  { \exp_after:wN \__exp_arg_next:nnn \exp_after:wN {#3} {#1}
      {#2} }
...
```

except the `x`, of course.

pearl 2:
`\romannumeral` for the "full" or "AFAP" expansion: »»

l3expan as a TeX pearls necklace: \romannumeral for the "AFAP" expansion

```
\cs_new:Npn \::f #1 \::: #2#3
  {
    \exp_after:wN \__exp_arg_next:nnn
      \exp_after:wN { \tex_romannumeral:D -`0 #3 }
      {#1} {#2}
  }
```

Act 3

GMOA, a proper extension to l3expan

GMOA, a proper extension to l3expan

a General Manipulation Of Arguments, GMOA

GMOA, a proper extension to l3expan

a General Manipulation Of Arguments, GMOA

Sound with l3expan's "the \::□'s":

```
\⁝⁝  ro :
{ \int_compare:nNnTF {\value{page}}={7}
    {\::ꓤ \ż } {\::ꓤ \bu }  }
{ \the \inputlineno }
```

GMOA, a proper extension to l3expan

a General Manipulation Of Arguments, GMOA

Sound with l3expan's "the \::□'s":

```
\⁝⁝  ro :
{ \int_compare:nNnTF {\value{page}}={7}
    {\::ꓤ \ż } {\::ꓤ \bu }  }
{ \the \inputlineno }
```

the intermediate l3expan-like translation:

```
> \__⁝⁝_prepare'τ⟦ς⟧:w \::r \::o \__⁝⁝_τ᪶yield:w \::: {}.
```

...and the result:

```
> {\bu }{62}.
```

GMOA, a proper extension to l3expan

A proper extension to l3expan:

- applying multiple operators to one argument
- arbitrary reordering, Finite Sequence Manipulation, FSM
- arbitrary (re)grouping, Braced Dyck-language Sequence Manipulation, BDSM

GMOA, a proper extension to l3expan: l3expan plus multiple operators

```
\⁝⁝  (ro) :
{ \int_compare:nNnTF {\value{page}}={7}
    {\::ᴙ \ż } {\::ᴙ \bu }  }
{ \the \inputlineno }
```

GMOA, a proper extension to l3expan: l3expan plus multiple operators

```
\⁝⁝  (ro) :
{ \int_compare:nNnTF {\value{page}}={7}
    {\::ꓤ \ż } {\::ꓤ \bu }  }
{ \the \inputlineno }
```

the intermediate l3expan-like translation:

```
> \__⁝⁝_prepare'τ⟦ς⟧:w \::r∗ \::o \__⁝⁝_τ∽yield:w \::: {}.
```

...and the result:

```
> {bubołak}{\the \inputlineno }.
```

GMOA, a proper extension to l3expan: ...plus FSM

...plus Finite Sequence Manipulation, FSM:

```
\⁝⁝  |4| 1ro 2₂4r :
{ \int_compare:nNnTF {\value{page}}={7}
    {\::ᴙ \ż } {\::ᴙ \bu }  }
{ \the \inputlineno }
\ERROR_undefined_CS:
{ \use:V \g_tex_year_int }
```

or just

```
\⁝⁝      1ro 2₂4r : …
```

GMOA, a proper extension to l3expan: …plus FSM

…plus Finite Sequence Manipulation, FSM:

```
\⁝⁝  |4| 1ro 2₂4r :
{ \int_compare:nNnTF {\value{page}}={7}
    {\::ꓤ \ż } {\::ꓤ \bu }  }
{ \the \inputlineno }
\ERROR_undefined_CS:
{ \use:V \g_tex_year_int }
```

or just

```
\⁝⁝      1ro 2₂4r : …
```

the intermediate l3expan-like translation:

```
> \__⁝⁝_prepare'τ⟦ς⟧:w \⁝⁝_prepare'FSM:w \¨F№1 \::r∗ \::o∗ \¨i
      \¨F№2 \¨I \¨F№2 \¨i \¨F№4 \::r∗ \¨i
      \q__⁝⁝_FSM'craw῝start 4\__⁝⁝_τ῝yield:w \::: {}.
```

…and the result:

```
> {bubołak}\the \inputlineno {\the \inputlineno }{2015}.
```

GMOA, a proper extension to l3expan: ...plus BDSM

...plus Braced Dyck-language Sequence Manipulation

```
\⁝⁝  1ro {{{{2}2}4r}} :
{ \int_compare:nNnTF {\value{page}}={7}
    {\::ᴚ \ż } {\::ᴚ \bu }  }
{ \the \inputlineno }
```

GMOA, a proper extension to l3expan: ...plus BDSM

...plus Braced Dyck-language Sequence Manipulation

```
\⁝⁝  1ro {{{{2}2}4r}} :
{ \int_compare:nNnTF {\value{page}}={7}
    {\::ꓤ \ż } {\::ꓤ \bu }  }
{ \the \inputlineno }
```

the intermediate l3expan-like translation:

```
> \__⁝⁝_prepare'τ⟦ς⟧:w \⁝⁝_prepare'FSM:w \¨F№1 \::r∗ \::o∗ \¨i
      \¨Bɛ \¨Bɛ \¨Bɛ \¨Bɛ \¨B№2 \¨BI \¨Bɔ \::o∗ \¨Bς \¨Bɔ \¨B№2
      \¨BI \¨Bɔ \¨Bς \¨Bɔ \¨B№4 \::r∗ \¨
Bi \¨Bɔ \¨Bς \¨Bɔ \::i \q__⁝⁝_FSM'craw῀start 4\__⁝⁝_τ῀yield:w
      \::: {}.
```

...and the result:

```
> {bubołak}{{{{62}\the \inputlineno }{2015}}}.
```

GMOA, a proper extension to l3expan: it's a DFA… almost

It's a DFA,

GMOA, a proper extension to l3expan: it's a DFA... almost

It's a DFA,
and expanded in one `\expandafter` in a sense.

GMOA, a proper extension to l3expan: it's a DFA… almost

It's a DFA,
and expanded in one `\expandafter` in a sense.

But a DFA cannot recognize a Dyck language!

GMOA, a proper extension to l3expan: it's a DFA... almost

It's a DFA,
and expanded in one `\expandafter` in a sense.

But a DFA cannot recognize a Dyck language!

mystery unveiled:
the outermost group is first picked by TEX's argument skanner and a special symbol is put at the end.

GMOA, a proper extension to l3expan: real life uses

```
\⁝⁝   12{425678}  13{435679}  :
\DeclareOption
{oneside}{twoside} % 23
\PassOptionsToClass % 4
{report} % 5
\ʢ_pdef:Npn % 6 |\protected| doesn't stop |\the|,
      |\expandafter|, |\romannumeral|…
\__ins'_page'oddity'count:  % 7 ^^A <<<<<oneside
\c_one  \c@page
 % 8 (we don't rely on implementation.)   9
```

GMOA, a proper extension to l3expan: real life uses

```
\cs_new_protected:Npn
\__ins'_toˈmule⁝
#1 : % no matter how many arguments (and how preprocessed), we
     group
    % them all and pass in one pair of braces.
{ \⁝⁝  I {#1} : \__ins'_toˈmule:n }
```

GMOA, a proper extension to l3expan: real life uses

```
\cs_new_protected:Npn
\__ins'_to'mule⁝
#1 : % no matter how many arguments (and how preprocessed), we
      group
     % them all and pass in one pair of braces.
{ \⁝⁝  I {#1} : \__ins'_to'mule:n }
```

then uses:

```
\__ins'_to'mule⁝  NN :
\tl_new:N \g_ins'mule_curr'lang_tl
```

…

```
\__ins'_to'mule⁝  Hn  :
\⚳_new'def:Nn
\__ins'mule_style'descriptor:n
{…
…}
```