

The l3tl-analysis package: analysing token lists*

The L^AT_EX3 Project[†]

Released 2014/05/06

1 l3tl-analysis documentation

This module mostly provides internal functions for use in the l3regex module. However, it provides as a side-effect a user debugging function, very similar to the `\ShowTokens` macro from the `ted` package.

`\tl_show_analysis:N`
`\tl_show_analysis:n`

`\tl_show_analysis:n` $\{\langle token\ list\rangle\}$

Displays to the terminal the detailed decomposition of the $\langle token\ list\rangle$ into tokens, showing the category code of each character token, the meaning of control sequences and active characters, and the value of registers.

1.1 Internal functions

`\s__tl`

The format used to store token lists internally uses the scan mark `\s__tl` as a delimiter.

`__tl_analysis_map_inline:nn`

`__tl_analysis_map_inline:nn` $\{\langle token\ list\rangle\}$ $\{\langle inline\ function\rangle\}$

Applies the $\langle inline\ function\rangle$ to each individual $\langle token\rangle$ in the $\langle token\ list\rangle$. The $\langle inline\ function\rangle$ receives three arguments:

- $\langle tokens\rangle$, which both o-expand and x-expand to the $\langle token\rangle$. The detailed form of $\langle token\rangle$ may change in later releases.
- $\langle catcode\rangle$, a capital hexadecimal digit which denotes the category code of the $\langle token\rangle$ (0: control sequence, 1: begin-group, 2: end-group, 3: math shift, 4: alignment tab, 6: parameter, 7: superscript, 8: subscript, A: space, B: letter, C:other, D:active).
- $\langle char\ code\rangle$, a decimal representation of the character code of the token, -1 if it is a control sequence (with $\langle catcode\rangle$ 0).

*This file describes v4745, last revised 2014/05/06.

[†]E-mail: latex-team@latex-project.org

For optimizations in `l3regex` (when matching control sequences), it may be useful to provide a `_t1_analysis_from_str_map_inline:nn` function, perhaps named `_str_analysis_map_inline:nn`.

1.2 Internal format

The task of the `l3tl-analysis` module is to convert token lists to an internal format which allows us to extract all the relevant information about individual tokens (category code, character code), as well as reconstruct the token list quickly. This internal format is used in `l3regex` where we need to support arbitrary tokens, and it is used in conversion functions in `l3str-convert`, where we wish to support clusters of characters instead of single tokens.

We thus need a way to encode any *⟨token⟩* (even begin-group and end-group character tokens) in a way amenable to manipulating tokens individually. The best we can do is to find *⟨tokens⟩* which both `o`-expand and `x`-expand to the given *⟨token⟩*. Collecting more information about the category code and character code is also useful for regular expressions, since most regexes are catcode-agnostic. The internal format thus takes the form of a succession of items of the form

⟨tokens⟩ `\s__t1` *⟨catcode⟩* *⟨char code⟩* `\s__t1`

The *⟨tokens⟩* `o`- and `x`-expand to the original token in the token list or to the cluster of tokens corresponding to one Unicode character in the given encoding (for `l3str-convert`). The *⟨catcode⟩* is given as a single hexadecimal digit, 0 for control sequences. The *⟨char code⟩* is given as a decimal number, `-1` for control sequences.

Using delimited arguments lets us build the *⟨tokens⟩* progressively when doing an encoding conversion in `l3str-convert`. On the other hand, the delimiter `\s__t1` may not appear unbraced in *⟨tokens⟩*. This is not a problem because we are careful to wrap control sequences in braces (as an argument to `\exp_not:n`) when converting from a general token list to the internal format.

The current rule for converting a *⟨token⟩* to a balanced set of *⟨tokens⟩* which both `o`-expands and `x`-expands to it is the following.

- A control sequence `\cs` becomes `\exp_not:n { \cs } \s__t1 0 -1 \s__t1`.
- A begin-group character `{` becomes `\exp_after:wN { \if_false: } \fi: \s__t1 1 \s__t1`.
- An end-group character `}` becomes `\if_false: { \fi: } \s__t1 2 \s__t1`.
- A character with any other category code becomes `\exp_not:n {⟨character⟩} \s__t1 \s__t1`.

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

E		S	
exp commands:		<code>\ShowTokens</code>	<i>1</i>
<code>\exp_after:wN</code>	<i>2</i>	str commands:	
<code>\exp_not:n</code>	<i>2, 2</i>	<code>_str_analysis_map_inline:nn</code>	<i>2</i>
F		T	
fi commands:		tl commands:	
<code>\fi</code> :	<i>2, 2</i>	<code>\s_tl</code>	<i>1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2</i>
I		<code>_tl_analysis_from_str_map_-</code> <code>inline:nn</code>	<i>2</i>
if commands:		<code>_tl_analysis_map_inline:nn</code>	<i>1, 1</i>
<code>\if_false</code> :	<i>2, 2</i>	<code>\tl_show_analysis:N</code>	<i>1</i>
		<code>\tl_show_analysis:n</code>	<i>1, 1</i>