

The l3tl-analysis package: analysing token lists^{*}

The L^AT_EX3 Project[†]

Released 2017/03/18

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.

<code>\tl_show_analysis:N</code>	<code>\tl_show_analysis:n {⟨token list⟩}</code>
<code>\tl_show_analysis:n</code>	

Displays to the terminal the detailed decomposition of the `⟨token list⟩` 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

<code>\s__tl</code>	The format used to store token lists internally uses the scan mark <code>\s__tl</code> as a delimiter.
---------------------	--

<code>__tl_analysis_map_inline:nn</code>	<code>__tl_analysis_map_inline:nn {⟨token list⟩} {⟨inline function⟩}</code>
---	--

Applies the `⟨inline function⟩` to each individual `⟨token⟩` in the `⟨token list⟩`. The `⟨inline function⟩` receives three arguments:

- `⟨tokens⟩`, which both o-expand and x-expand to the `⟨token⟩`. The detailed form of `⟨token⟩` may change in later releases.
- `⟨catcode⟩`, a capital hexadecimal digit which denotes the category code of the `⟨token⟩` (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).
- `⟨char code⟩`, a decimal representation of the character code of the token, `-1` if it is a control sequence (with `⟨catcode⟩` 0).

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

^{*}This file describes v7019, last revised 2017/03/18.

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

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 $\langle token \rangle$ (even begin-group and end-group character tokens) in a way amenable to manipulating tokens individually. The best we can do is to find $\langle tokens \rangle$ which both *o*-expand and *x*-expand to the given $\langle token \rangle$. 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

$\langle tokens \rangle \backslash s_t1 \langle catcode \rangle \langle char\ code \rangle \backslash s_t1$

The $\langle tokens \rangle$ *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 $\langle catcode \rangle$ is given as a single hexadecimal digit, 0 for control sequences. The $\langle char\ code \rangle$ is given as a decimal number, -1 for control sequences.

Using delimited arguments lets us build the $\langle tokens \rangle$ progressively when doing an encoding conversion in `l3str-convert`. On the other hand, the delimiter $\backslash s_t1$ may not appear unbraced in $\langle tokens \rangle$. This is not a problem because we are careful to wrap control sequences in braces (as an argument to $\backslash exp_not:n$) when converting from a general token list to the internal format.

The current rule for converting a $\langle token \rangle$ to a balanced set of $\langle tokens \rangle$ which both *o*-expands and *x*-expands to it is the following.

- A control sequence $\backslash cs$ becomes $\backslash exp_not:n \{ \backslash cs \} \backslash s_t1 0 -1 \backslash s_t1$.
- A begin-group character $\{$ becomes $\backslash exp_after:wN \{ \backslash if_false: \} \backslash fi: \backslash s_t1 1 \langle char\ code \rangle \backslash s_t1$.
- An end-group character $\}$ becomes $\backslash if_false: \{ \backslash fi: \} \backslash s_t1 2 \langle char\ code \rangle \backslash s_t1$.
- A character with any other category code becomes $\backslash exp_not:n \{ \langle character \rangle \} \backslash s_t1 \langle hex\ catcode \rangle \langle char\ code \rangle \backslash s_t1$.

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

exp commands:

$\backslash exp_after:wN$ 2
 $\backslash exp_not:n$ 2, 2

F

fi commands:

$\backslash fi:$ 2, 2

I if commands: \if_false: 2, 2 Q quark internal commands: \s__tl . 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2 S \ShowTokens 1	T str internal commands: __str_analysis_map_inline:nn 1 tl commands: \tl_show_analysis:N 1 \tl_show_analysis:n 1, 1 tl internal commands: __tl_analysis_from_str_map_ inline:nn 1 __tl_analysis_map_inline:nn .. 1, 1
--	---