LISP on TeX

July 29, 2022

Abstract

A LISP interpreter written only with TeX macros. It works as a style file of LaTeX. LISP on TeX adopts static scoping, dynamic typing, and eager evaluation. We can program easily with LISP on TeX.

Contents

1 Summary

To use LISP on TeX, you should include the `lisp-on-tex` package.

```
\usepackage{lisp-on-tex}
```

If you do it, you can write LISP codes as a argument of `\lispinterp`.

```
\lispintrep{
  (\some \LISP 'codes')
  % example
  (\define (\sum \a \b) (\+ \a \b))
}
```

In LISP on TeX, a symbol is a control sequence; a string is tokens surrounded by quotation marks; and an integer is a TeX's integer using colon prefix.

2 Installation

Put all files into your TEXMF tree.

3 Details

3.1 Class Options

Option Name	Meaning
`noGC`	Never use GC (default)
`markGC`	Using Mark-Sweep GC
`GCopt=...`	Passing option to the GC engine

Currently, LISP on TeX supports Mark-Sweep GC. If you want to use it, you should use `markGC` option. You can also control heap size by using `GCopt=heapsize=n` where n is greater than 3000. The default heap size is 32768. For example, the code

```
\usepackage[markGC, GCopt={heapsize=5000}]{lisp-on-tex}
```

shows that LISP on TeX uses Mark-Sweep GC and the heap size is 5000.

3.2 Syntax

Kinds	Literals	Examples
CONS Cell	‘(‘ *obj* ... ‘.‘ *obj* ‘)‘, ‘(‘ *obj* ... ‘)‘	‘(\+ :1 :2)‘
Integer	‘:‘ *TeX’s integer*	‘:42‘, ‘:"3A‘
String	‘’‘ *TeX’s balanced tokens* ‘’‘	‘’\foo{bar}baz’‘
Symbol	*TeX’s control sequence*	‘\cs‘
Boolean	‘/t‘ or ‘/f‘	
Nil	‘()‘	
Skip	‘@‘ *TeX’s skip*	‘@12pt plus 34cm‘
Dimen	‘!‘ *TeX’s dimen*	‘!56pt‘

3.3 Functions and Special Forms

3.3.1 Definition

3.3.1.1 \define : Define a symbol.

```
% symbol form
(\define \foo :42) % ()
\foo % :42
% function form
(\define (\foo \n) (\* \n :2))
(\foo :3) % :6
```

3.3.1.2 \defineM : Define a mutable symbol

```
% symbol form
(\defineM \foo :42) % ()
\foo % :42
```

3.3.1.3 \setB : Rewrite a mutable symbol.

```
(\setB \foo ’bar’)
\foo % ’bar’
```

3.3.1.4 \defmacro : Define a macro.

```
(\defmacro (\foo \x) (\list (\quote \bar) \x \x \x)) % ()
```

3.3.1.5 \macroexpand : Expand a macro

```
(\macroexpand (\quote (\foo :1))) % (\bar :1 :1 :1)
```

3.3.1.6 \lambda : Create a function.

```
% normal form
((\lambda (\x) (\+ \x :2)) :3) % :5
% list form
((\lambda \x \x) :1 :2) % (:1 :2)
% remain argument form
((\lambda (\x . \y) \y) :1 :2 :3) % (:2 :3)
```

3.3.1.7 \let : Define local symbols.

```
(\define \x 'foo')
(\let ((\x :4) (\y :3)) (\+ \x \y)) % :7
\x % 'foo'
```

3.3.1.8 \letM : Define mutable local symbols.

```
(\letM ((\x 'foo'))
  (\begin (\setB \x 'bar') \x)) % 'bar'
```

3.3.1.9 \letrec : Define local symbols recursively.

```
(\letrec
  ((\oddQ (\lambda (\n)
            (\lispif (\= \n :0) /f (\evenQ (\- \n :1)))))
   (\evenQ (\lambda (\n)
            (\lispif (\= \n :0) /t (\oddQ (\- \n :1))))))
   (\oddQ :42)) % /f
```

3.3.2 Control Flow

3.3.2.1 \lispif : Branch.

```
(\lispif /t 'true' 'false') % 'true'
(\lispif /f 'true' 'false') % 'false'
```

3.3.2.2 \begin : Execute expressions.

```
(\letM ((\x :1)) (\begin (\setB \s 'foo') \x))
% 'foo'
```

3.3.2.3 \callOCC : One-shot continuation.

```
(\defineM \x 'unchanged')
(\callOCC (\lambda (\c)
            (\begin (\c '\foo ')
                    (\setB \x 'changed')))) % '\foo '
\x % 'unchanged'
(\callOCC (\lambda (\c) :42)) % :42
```

3.3.3 String Manipulations

3.3.3.1 \concat : Concatenate tokens.

```
(\concat '$' '\foo ' '{bar}' '$') % '$\foo {bar}$'
```

3.3.3.2 \intTOstring : Convert a integer to TeX's tokens.

```
(\intTOstring :42) % '42'
```

3.3.3.3 \group : Grouping.

```
(\group '\some {tokens}') % '{\some {tokens}}'
```

3.3.3.4 \ungroup : Ungrouping.

```
(\ungroup '{\some {tokens}}') % '\some {tokens}'
```

3.3.3.5 \expand : Expand tokens.

```
\newcommand\foo[1]{I got #1!}
\lispinterp{
  (\expand '\foo{Foo}') % 'I got Foo!'
}
```

3.3.4 Arithmetical Functions

3.3.4.1 \+ : Addition.

```
(\+) % :0
(\+ :1 :2) % :3
(\+ :3 :4 :5) % :12
```

3.3.4.2 \- : Subtraction.

```
(\- :1) % :-1
(\- :3 :2) % :1
(\- :3 :2 :1) % :0
```

3.3.4.3 * : Multiplication.

```
(\*) % :1
(\* :2 :3) % :6
(\* :3 :4 :5) % :60
```

3.3.4.4 \/ : Division.

```
(\/ 2) % :0 (1/2 -> 0)
(\/ 7 2) % :3
```

3.3.4.5 \mod : Modulo.

```
(\mod :42 :23)  % :19
(\mod :3 :2)    % :1
(\mod :3 :-2)   % :1
(\mod :-3 :2)   % :-1
(\mod :-3 :-2)  % :-1
```

3.3.4.6 \>, \<, \geq, \leq : Comparison.

```
(\> :3 :2)   % /t
(\< :2 :3)   % /t
(\geq :3 :2) % /t
(\geq :3 :3) % /t
(\leq :2 :3) % /t
(\leq :3 :3) % /t
```

3.3.4.7 \isZeroQ,\positiveQ,\negativeQ : Some predicates.

```
(\isZeroQ :0)    % /t
(\positiveQ :42) % /t
(\negativeQ :-2) % /t
```

3.3.4.8 \max : Maximum.

```
(\max :-10 :-5 :0 :5 :10) % :10
```

3.3.4.9 \min : Minimum.

```
(\min :-10 :-5 :0 :5 :10) % :-10
```

3.4 Logical functions

3.4.0.1 \and, \or, \not : Logical and, or, not

```
(\and /t /t) % /t
(\and /t /f) % /f
(\or /t /t)  % /t
(\or /t /f)  % /t
(\not /t)    % /f
```

3.5 Traditional LISP Functions and Special Forms

3.5.0.1 \quote : Quote.

```
(\quote :42) % :42
(\quote (\+ :1 :2)) % (\+ :1 :2)
```

3.5.0.2 \cons, \car, \cdr : CONS, CAR, CDR

```
(\cons :42 'foo') % (:42 . 'foo')
(\car (\quote (:1 :2))) % :1
(\cdr (\quote (:1 :2))) % (:2)
```

3.5.0.3 \list : Create a list

```
(\list :1 :2 (\+ :3 :4)) % (:1 :2 :7)
```

3.5.0.4 \length : Get the length of a list.

```
(\length ()) % :0
(\length (\list :1 :2 'three')) % :3
```

3.5.0.5 \map : Map function.

```
(\define (\f \x \y \z) (\+ \x \y \z))
(\map \f (\list :1 :2 :3)
         (\list :4 :5 :6)
         (\list :7 :8 :9)) % (:12 :15 :18)
```

3.5.0.6 \nth : Get the n-th value of a list (starting with 0).

```
(\nth (\list 'foo' 'bar' 'baz') :1) % 'bar'
```

3.5.0.7 \= : Equality.

```
(\= '42' :42) % /f
(\= :23 :23) % /t
(\= (\cons :1 'foo') (\cons :1 'foo')) % /f
(\= 'foo' 'foo') % /t
```

3.5.0.8 \texprint : Convert a object to TeX's tokens and output it to the document

```
(\texprint (\concat '\foo' (\group '42'))) % return () andoutput \foo{42}
(\texprint :42) % output 42
```

3.5.0.9 \print : (For test) output a object as TeX's tokens

```
(\print ()) % output ()
(\print (\quote \foo)) % output \string\foo
(\print :42) % output :42
(\print 'bar') % output 'bar'
```

3.6 Type predicates : \xyzQ

```
(\symbolQ (\quote \cs))
(\stringQ 'foo')
(\intQ :42)
(\booleanQ /f)
(\dimenQ !12pt)
(\skipQ @12pt plus 1in minus 3mm)
(\pairQ (\cons :1 :2))
(\nilQ ())
(\funcQ \+)
```

```
(\closureQ (\lambda () ()))
(\defmacro (\x) ())
(\macroQ \x)
(\listQ ())
(\listQ (\list :1 :2))
(\atomQ :23)
(\atomQ 'bar')
(\procedureQ \+)
(\procedureQ (\lambda () ()))
```

3.7 LaTeX Utils

3.7.0.1 \readLaTeXCounter : Read an integer from LaTeX

```
\setcounter{foo}{42}
\lispinterp{
  (\readLaTeXCounter 'foo') % :42
}
```

3.7.0.2 \message : Wrapper of LaTeX's message

```
(\message 'output') % output "message" to console and return ()
```

3.8 Others

3.8.0.1 \read : Read a LISP expression from stdin

```
(\read) % input :42 and return it
```

3.8.0.2 \fgets : Read a string from stdin.

```
(\fgets) % input \some {tokens} and return '\some {tokens}'
```

4 Additional Packages

4.1 Fixed Point Numbers

The package `lisp-mod-fpnum` adds fixed point numbers to LISP on TeX. Load it by `\usepackage`:

```
\usepackage{lisp-on-tex}
\usepackage{lisp-mod-fpnum}
```

4.1.1 Syntax

Kinds	Literals	Examples
Fixed point number	`+{fpnum::‘ *number* ‘}‘`	`+{fpnum::1.23}`

4.1.2 Functions

4.1.2.1 \fpnumTOstring : Convert a fixed point number to a string.

```
(\fpnumTOstring +{fpnum::1.23}) % '1.23'
```

4.1.2.2 \fpplus : Addition.

```
(\fpplus +{fpnum::1.2} +{fpnum::1.4}) % 2.59999 (arithmetical error)
```

4.1.2.3 \fpminus : Subtraction.

```
(\fpminus +{fpnum::4.2} +{fpnum::2.3}) % 1.9
```

4.1.2.4 \fpmul : Multiplication.

```
(\fpmul +{fpnum::1.2} +{fpnum::1.4}) % 1.67998
```

4.1.2.5 \fplt : Comparison.

```
(\fplt +{fpnum::1.2} +{fpnum::2.3}) % /t
```

4.2 Regular Expressions

The package lisp-mod-l3regex is thin wrapper of l3regex. Load it by \usepackage:

```
\usepackage{lisp-on-tex}
\usepackage{lisp-mod-l3regex}
```

4.2.1 Functions

4.2.1.1 \regMatch, \regMatchResult : Match.

```
(\regMatch 'hoge+' 'hogeeeeeee') % /t
(\regMatchResult '(\w+)\s+is\s+(\w+)\.' 'He is crazy.')
% ('He is crazy.' 'He' 'crazy')
```

4.2.1.2 \regExtract : Extraction.

```
(\regExtract '\w+' 'hello regex world') % ('hello' 'regex' 'world')
```

4.2.1.3 \regReplaceAll, \regReplaceOnce : Replace.

```
(\regReplaceAll '(\w+?)to(\w+?)' '$\1\c{to}\2$' 'AtoB BtoC') % '$A\to B$ $B\to C$'
(\regReplaceOnce 'foo+' '[\0]' 'foooofooooooo') % '[foooo]fooooooo'
```

4.2.1.4 \regSplit : Split.

```
(\regSplit '/' '/path/to/hogehoge') % ('' 'path' 'to' 'hogehoge')
```

CHANGELOG

```
## TODOs ##

* Writing user manual
* Add functions and special forms

## CHANGELOG ##

Oct. 25, 2015 : 2.0
*************************

* Add GC
* Refine some special forms like \define
* Add checking #args for some functions.
* Add thin wrapper of l3regex

Jul. 12, 2014 : 1.3
*************************

* Add one shot continuations.
* Add some arithmetical functions.
* Debug environment.

Jan. 03, 2014 : 1.2
*************************

* Added TUG2013's examples.
* Improved the performance.

Aug. 10, 2013 : 1.1
*************************

* Added \letrec and \expand.
* debug

Mar. 04, 2013 : 1.0
*************************

## Licence ##

Modified BSD (see LICENCE)

************************************************
HAKUTA Shizuya <hak7a3@live.jp>

https://bitbucket.org/hak7a3/lisp-on-tex/
```