Emacs Lisp

PENG

Created: 20260815, Version: 20260831

Contents

1 Introduction

1.1 Lisp History

Lisp (List Processing Language) was first developed in the late 1950s.
Common Lisp (cl-lib), Scheme (Guile)

2 Lisp Data Types

Definition 2.1 (object)**.** *A Lisp object is a piece of data.*

Definition 2.2 (primitive types)**.** *integer, float, cons, symbol, string, vector, hash-table, subr, byte-code function, record, buffer*

Remark 2.3. *The primitive type of each object is* ***implicit*** *in the object itself.*

2.1 Printed Representation and Read Syntax

Definition 2.4 (printed representation and read syntax)**.** `prin1` *and* `read`

Remark 2.5. *In most cases, an object's printed representation is also a read syntax for the object.*

Example 2.6 (objects have no read syntax)**.** `#<type varname>` *cannot be read.*

```
(current-buffer) => #<buffer elisp.tex>
```

An expression is primarily a Lisp object.

Interactive evaluation process: read the text ⇒ produce a Lisp object ⇒ evaluate that object. Evaluation and reading are separate activities.

2.2 Special Read Syntax

Definition 2.7 (special hash notations)**.**

- `#<>` *: objects have no read syntax*
- `#'` *: shortcut for function*

2.3 Comments

Definition 2.8 (comment)**.** `;` *continues to the end of line.*

The Lisp reader discards comments.

2.4 Lisp Programming Types

Two categories of types:

- Lisp programming types:
  - integer
  - floating-point
  - character
  - symbol
  - sequence
  - array
  - string
  - vector
  - char-table
  - bool-vector
  - hash table
  - function
  - macro
  - primitive function
  - closure
  - record
  - type descriptors
  - type specifiers
  - autoload
  - finalizer
- editing (unique to Emacs Lisp)

2.4.1 Integer Type

Definition 2.9 (integer)**.**

- *fixnums : small integers, minimum range: 30 bits*
- *bignums : large integers, arbitrary precision*

Example 2.10 (compare)**.**

```
(eql 1 1.) ; or
(= 1 1.)   ; for all numbers
(eq 1 1.)  ; for fixnums
```

Read syntax for integer: `1. => 1`

2.4.2 Floating-Point Type

3 Variables

3.1 Scoping Rules for Variable Bindings

3.1.1 Lexical Binding

Definition 3.1 (lexical binding)**.** *Any reference to the variable must be located textually within the binding construct.*

Lisp evaluator get the value of a variable process: look first in the lexical environment ⇒ if not then look in the dynamic value cell.

Definition 3.2 (closure)**.** *A closure is created when you define a named or anonymous function with lexical binding enabled.*

Example 3.3.

```
(let ((x 0))
  (defun var ()
    (setq x (1+ x))))

(var) => 0
(var) => 1
(var) => 2
```

4 Regular Expression

5 Automatic Indentation of code

Automatic indentation [1]:

- What is the right indentation of a line?
- When to reindent a line?

Indentation function:

- parsing forward from some safe starting point until the position of interest
- parsing backward from the position of interest

Generic indentation engine:

- CC-mode indentation code
- SMIE
- Tree-sitter

5.1 Simple Minded Indentation Engine

- operator precedence grammar
- unable to detect syntax error
- most programming languages cannot be parsed correctly by SMIE

Definition 5.1 (smie-setup)**.**

```
(smie-setup grammar rules-function :forward-token :backward-token)
```

5.2 Tree-sitter

Parser-based indentation

6 Language Server Protocol

7 Dynamic Module

8 Canvas

References

[1] Bill Lewis, Richard M Stallman, and Dan LaLiberte. *GNU Emacs Lisp reference manual*. Free Software Foundation, 1998. ISBN: 1882114728.