Inspection of Lisp - Names, Bindings, Scopes and Types

Mahmut Bulut - 14501026
Computer Engineering Dept., Yıldız Technical University

March 30, 2015

Naming in Lisp

In Lisp symbols are just a name. Naming makes new relations internally. Symbols are datatypes in Lisp. Also symbols are unique identifiers that are identical to other symbols with the same name without restricton with case-sensitivity.

On the other hand Lisp also names functions as variables. This is a distinction with other programming languages.

```
=> (setq first 'the-bender)
-> THE-BENDER

=> (first (list 3 2 1))
-> 3

=> first
-> THE-BENDER
```

At first and the last one *first* is used as variable. But at second first is used as function. In Lisp a value can more than one name. This makes aliasing easy and useful.[1]

Binding in Lisp

Lisp has lexical binding which means that a variable declared with *let* statement will be lexically bound at compile time and cannot be reach outside of lexical scope of *let*.[2] In other words this can be called as static binding.

```
(let ((data 1))     ; data is lexically bound.
  (+ data 3))
=> 4

(defun getdata ()
  data)             ; data is used free in this function.

(let ((data 1))     ; data is lexically bound.
  (getdata))
```

[1] http://psg.com/~dlamkins/sl/chapter03-05.html
[2] https://www.gnu.org/software/emacs/manual/html_node/elisp/Lexical-Binding.html

```
10 ;error-->Symbol's value as variable is void: data
```

In addition to lexical binding there is also dynamic binding. In dynamic binding variables lives in one global namespace and can be accessed amongst each other. [3] Dynamic binding makes easier to code greater programs with ease but decrease security with all open namespace, symbol and variable definitions.

Scope in Lisp

There are several ways to determine the scope:

- Place of the reference in the expression.
- Kind of reference translation takes place.
- Location of reference
- Declaring a variable in local or global scope
- Environment which solves variable bindings within its context.

There are three types of environment in Lisp[4]; global, dynamic and lexical. **Global** environment is like its name global across a program if a declaration made in it, it will be known through the program. **Dynamic** environment is generator like environment. It binds at constructive expressions like *let* block. **Lexical** environment defines a lexical scope within a expression and scope will last as long as this expression lasts.

Types in Lisp

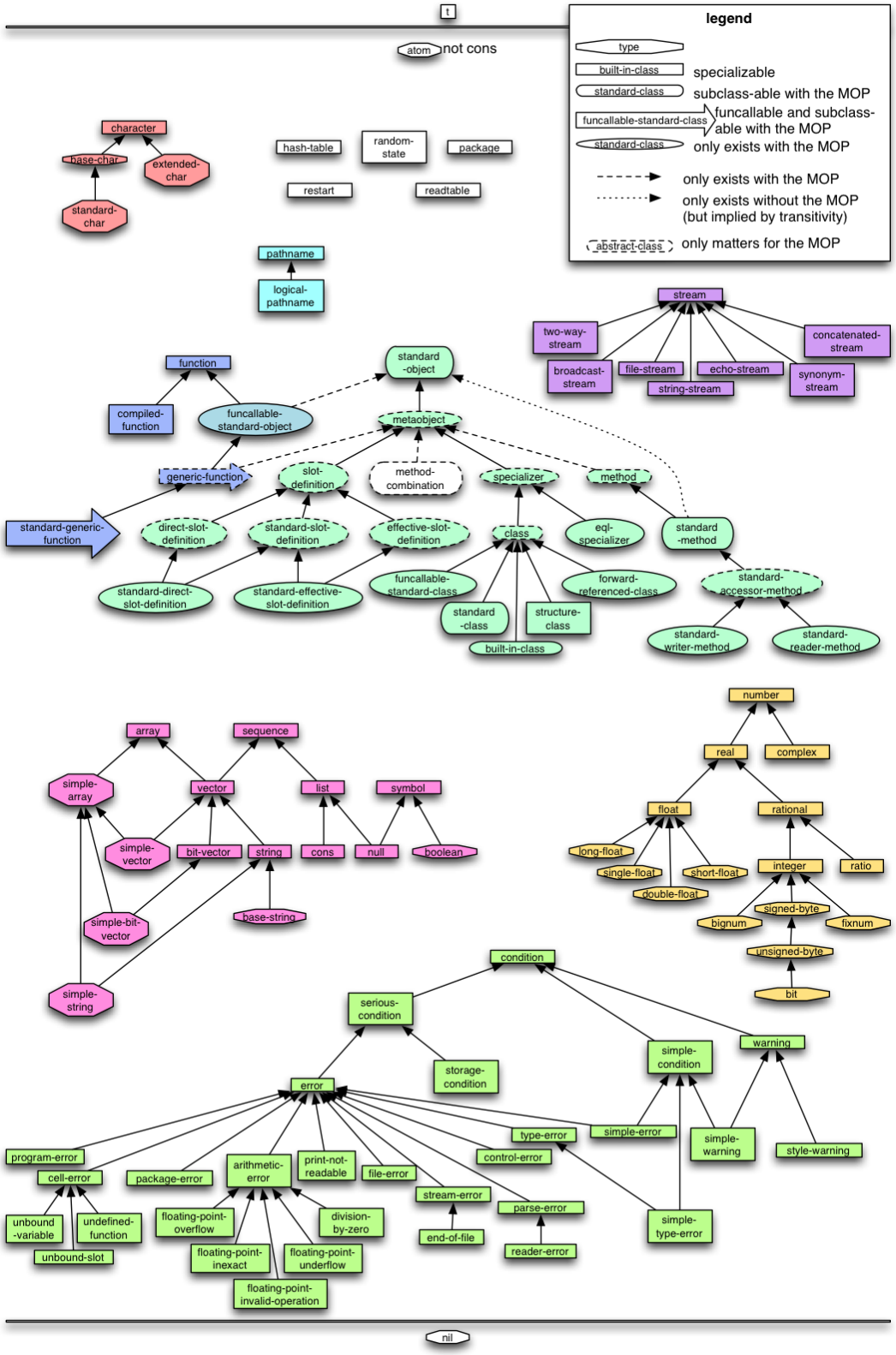
Data types are generally set of Lisp objects. But lisp has two main types:

- atom
- list

Lists contains atoms or lists of elements. Atoms are seperated in lists from each other with whitespace and they cannot contain anything else except itself[5]. With this seperation expression can be made with combination of lists ad atoms[6]. These types can be included in global scope and it makes easier to infer or coerce between each other. There is already a function named **coerce** in Comman Lisp.[7] In addition to it empty data type, which contains no data object are defined by *nil* type. Every type can be created from its relative types and coerce with them. You can see Common Lisp type hierarchy in Fig. 1.

[3] http://emacswiki.org/emacs/DynamicBindingVsLexicalBinding#toc2
[4] http://en.wikipedia.org/wiki/Common_Lisp#Kinds_of_environment
[5] http://graham.main.nc.us/~bhammel/graham/lisp.html
[6] https://www.gnu.org/software/emacs/manual/html_node/eintr/Lisp-Atoms.html
[7] https://www.cs.cmu.edu/Groups/AI/html/cltl/clm/node52.html

Figure 1: Common Lisp Type Hierarchy



List of Figures