Compare LISP with Object-Oriented Programming Languages

PENGFEI WANG
Department of Computer Science, Furman University, Greenville, SC

Introduction

- LISP is a family of functional programming languages specified in 1958. It is the second-oldest high-level programming language in which are still used today. It is very different from Object-Oriented Languages and doesn’t have much popularity as Object-Oriented Languages do, but, besides its wide use in Artificial Intelligence, it is also a good general purpose programming language as well.
- This paper is going to discuss the differences between LISP and OOP Languages and focus mainly on LISP.
- “LISP” stands for List Processor.

Attention: Abbreviations

In this article, FP and OOP are abbreviations of Functional Programming and Object-Oriented Programming respectively.

LISP: “Hello, world!”

This is SCL 1.2.3, an implementation of ANSI Common LISP. More information about SCL is available at http://www.scl.org/. SCL is free software, provided as is, with absolutely no warranty. It is mostly in the public domain, some portions are provided under OSS-style licenses. See the ‘RED’ and ‘OPTIME’ files in the distribution for more information.

WARNING: the Windows port is fragile, particularly for multithreaded code. Unfortunately, the development team currently lacks the time and resources to maintain this platform edition. (Contact TH, my name is Common LISP?)

Hello, world! [DELL]

Figure 1: LISP is concise. Source: RedMonk

Advantages of LISP

1. Variables are Immutable
   1. Easy to test: functions CANNOT change any values of symbols. Programmers don’t have to worry about changing in values of the variables. If error occurs, they only have to trace the wrong return value from one function to another. This feature significantly decreases the amount of work on debugging.
   2. No deadlock can occur: LISP programs can be run in parallel without deadlock. It actually doesn’t need a “lock” on variables because it will never change its value (typically). On multi-thread programs, LISP is more efficient than OOP, because it can be run in parallel. Even if the program is single-thread, the compiler can make the program run in parallel on CPU.

2. Closure
   - Based on higher-order functions, LISP can actually do what OOP languages hardly do: closure. Closure is needed when a function tries to access a captured variable from outside.
   - When a function i.e. lambda (x) refers to a variable outside of it (i.e. n), this function is called closure. The function need to be defined in the same environment with its free variable (i.e. n).
   - Actually some OOP languages start to implement closure based on the idea from functional languages, but LISP’s closure is far more concise than those (e.g. C++/11).

3. Extendable Language: Macro
   - LISP can be extended, amended, or macro, which changes the behavior of a compiler during compiling. By macro, you can simulate any grammar you like, for example the whole JAVA. Theoretically, with macro, LISP is equivalent to Lambda Calculus.
   - The program is going to execute simple-reader-extensions whenever it sees a ‘;’. The substitution (macro expansion) is done during compilation.

Fast & Concise
- <ANSI Common LISP> claims that LISP is “a language for writing fast programs and a language for writing programs fast”.
  - Yes, LISP is concise. Actually, it is one of the most concise language being widely used up till now. Also, its run-time is proved to be faster than others, even comparing to C++. However, in order to make LISP fast, the program and the compiler must obey to strict optimization and FP rules.

Disadvantages of LISP

1. Hard to learn
   - LISP was originally created as a mathematical notation for computer programs. LISP programmers have to learn Lambda Calculus.
   - Therefore, it is nearly impossible for newbies to master LISP, but it is relatively much easier to master an OOL.
   - Furthermore, due to FP and OOP’s differences, it is highly difficult for programmers to transfer from OOP languages (e.g. JAVA) to FP languages (e.g. LISP).

2. Small Libraries
   - Because of the hardness of learning and transferring from OOP to FP, the programmer community is very small.
   - Popularity: JAVA vs LISP
     - Google Search: See Figure 3
     - Amazon Books: 27486 (JAVA) vs 1286 (LISP)
     - Libraries: 4024 (JAVA) vs 978 (LISP)

   - This is essential, because productivity matters in companies. Programmers have to write functions by themselves, which are already existed in JAVA API.

3. Case Study: Bubble Sort
   - Data Set: 1000 Numbers List; Run 100 times
   - Environment: Windows 10; CPU @ 2.90GHz (8 CPUs)
   - Compiler: JAVA 1.8.0; Steel Bank Common LISP
   - In order to save space, JAVA source code do not show up here. The algorithm comes from WikiBooks.

   | 01: | Bubble sort in Common LISP |
   | 02: | (defun bubble-sort (lst) |
   | 03: | (loop repeat (1- (length lst)) do |
   | 04: | (loop for i on lst while (rest i) do |
   | 05: | (when (> (first i) (second i)) |
   | 06: | (rotatef (first i) (second i))) |
   | 07: | ) |

   | Common LISP | Java |
   | Lines | 6 | 14 |
   | Characters | 142 | 282 |
   | Run-Time | 208ms | 16ms |

   Note: Comments are not counted.

   - Analysis: We can see that Common LISP is so concise but its speed is not ideal, comparing to the experiment in Figure 2 (< Lisp as an Alternative to Java). Why? This is a good example of why optimization is necessary.
   1. Accessing an array in JAVA is O(1), but accessing a list in LISP is O(n).
   2. This Common LISP program uses a destructive function called “rotatef”. Using imperative style in LISP (e.g. rotatef), instead of FP style, will make it super slow.

How to optimize?
1. Optimize the Algorithm for FP languages.
2. Choose a better dialect of LISP with better optimization. For example, newLISP can access a list by O(1).

Actually, Common Lisp, as one of the most popular dialect of LISP, is not purely functional. Although programmers are recommended to write in functional way, the language itself is multi-paradigm, which also allows object-oriented programming (which is often procedural as well).