An overview of Common Lisp

Robert Strandh
Université Bordeaux 1

currently on sabbatical at the
University of Auckland CS department

Contents of talk

- what is/isn't Lisp and Common Lisp
- some history
- implementations
- features
- examples
- community resources
- questions

What is Lisp

Lisp is a family of languages that has been around since 1958.

It is the fourth most widely used (in terms of SLOC) programming language in the Debian GNU/Linux distribution, after C, C++, and Shell, with 3% of the code (this is because of Emacs Lisp).

Members of the family: Common Lisp, Interlisp, MacLisp, NIL, ZetaLisp, (Scheme, Dylan).

What is Common Lisp

Common Lisp is a standardized member of the Lisp family of languages.

It is a multi-paradigm language (functional, imperative, object-oriented).

It is currently one of the few (with Emacs Lisp) widely-used dialects of Lisp.

Some Lisp history

- first implementation in 1958
- LISP 1.5 in 1962
- MacLisp (PDP10 and Multics) 1970s
- Interlisp, ZetaLisp (Lisp Machine) also 1970s
- Common Lisp 1980s, ANSI standard in 1994

Common Lisp implementations

- Commercial: Allegro, LispWorks, MCL, Scieneering, Corman, etc.
- Free (native compilers): SBCL, CMUCL, OpenMCL
- Free (bytecode): ABCL (JVM), CLISP (own)

Features

What are the features that make Common Lisp worthwhile?

To me, it is a combination of the following:

Features

interactivity (dynamic redefinitions), first-class symbols, arbitrary-precision integers, exact rational arithmetic, well-integrated complex numbers, generalized references, multiple values, first-class functions, anonymous functions, macros, multiple inheritance, multiple dispatch, generic functions, method combination, (first-class) classes and meta classes, (first-class) packages, built-in programmable parser (read), built-in programmable unparser, reader macros, compiler macros, optional argument, keyword arguments, meta-object protocol, special (dynamically scoped) variables, named blocks, nonlocal goto (catch/throw), conditions, restarts, the `loop` macro, the `format` function, type declarations, compiler available at run-time, extensive list processing features.

Interactivity

Essential to avoid having to restart the application when modifications are made.

Demo:

1. redefining a class that has instances around.
2. in Gsharp, modify position of clef

Rational and bignum arithmetic

Both are fully integrated with the system built-in operations

Demo: factorial, etc

First-class and anonymous functions

Allow functions to take other functions as parameters

Demo: iota (from APL), mapcar, remove-if-not

Macros example (MIDI I/O)

```
(define-midi-message program-change-message (voice-message)
  :status-min #xc0 :status-max #xcf
  :slots ((program :initarg :program))
  :filler (setf program next-byte)
  :length 1
  :writer (write-bytes program))
```

Essentially, macros make it possible to create DSLs by extending the base language.

Multiple dispatch gives the right answer

As James Noble said

Generic functions and methods

Make things like the visitor pattern unnecessary:

Demo: Generic function on existing classes

Method combinations

make it easier to modify existing code while preserving protocols

Demo: memoization

Programmable reader example

Demo: intervals

Optional arguments

Make it easier for programs to evolve while maintaining backward compatibility.

Demo: add a parameter to an existing function

Keyword arguments

Make it easier to read calls to functions with many parameters.

Example: creating a window in CLX:

```
(create-window :parent root
               :x 10 :y 20 :width 500 :height 800
               :bit-gravity :south-east)
```

Special variables

Demo: dynamically bind a special variable

Conditions and restarts

Are sort of like exceptions in Java or C++, but do not unwind the stack.

Conditions and restarts allow two parts of a program to communicate: one that detects an error and the other that knows what to do about it.

This is essential for debugging.

Type declarations

Useful for type checking for storage conservation

Demo: bit vectors

The format function

Handles things like looping, conditionals, plurals, roman numerals, etc.

Demo: the bottle song

Freely-available development tools

I use (GNU/Linux):

- SBCL
- SLIME (Superior Lisp Interaction Mode for Emacs) with GNU Emacs
- McCLIM (free implementation of the Common Lisp Interface Manager)
- McCLIM tools such as Clouseau (inspector), listener, etc.
- Common Lisp HyperSpec (the standard in HTML format)

Mac and Windows development tools

Windows: probably best to use a commercial Lisp such as LispWorks or Allegro

Mac: Maybe OpenMCL instead of SBCL

Community resources

- the Common Lisp HyperSpec
- comp.lang.lisp newsgroup
- #lisp irc channel on irc.freenode.net
- Cliki (www.cliki.net), a CL Wiki
- planet.lisp.org

Questions?

Functional programming

Question: But functional programming is not always the best solution, right?

Answer: Right, and Common Lisp is *not* a functional language, but a multi-paradigm language, so you can use it to program functionally, or using some other paradigm.

Why not widely used?

Question: If Common Lisp is so good, why is it not used more

Answer: The question suggests that things that are good are widely used. The contrary is true. The internal-combustion engine is not particularly good compared to alternatives.

Why not widely used?

Question: If Common Lisp is so good, why is it not used more

Answer: Industry is essentially incapable of determining what tools are good and what tools are bad.

Only atoms and lists

Question: I have heard that Lisp only has two datatypes, atoms and lists. Is that true?

Yes, but "atom" covers symbols, structures, vectors, arrays, class instances, functions, packages, hash tables, etc.

Greenspun's tenth

Greenspun's Tenth Rule of Programming:

any sufficiently complicated C or Fortran program contains an ad hoc informally-specified bug-ridden slow implementation of half of Common Lisp.