Lisp Machines and the Analysis of Their High-Level Language Computer Architecture

Dominic Dabish and Evan Bradley

Oakland University

{*dadabish*, *edbradley*}*@oakland.edu*

April 2016

Overview

Early history of Lisp

- Lambda Calculus introduced in 1930s by Alonzo Church
- Fortran in 1957
- No programming languages optimized for artificial intelligence
- Lisp designed in 1958 by John McCarthy
- Lisp code implemented on IBM 170 months after

Lisp machines

- Lisp machines released in mid-1970s, became popular in 1980s
- Manufactured by Symbolics, Lisp Machines, Inc., Xerox, TI
- Offered GUIs, advanced programmability, flexibility
- Noncompetitive hardware
- Eventually became outperformed by general-purpose computers
- Vendors went bankrupt in 1990s

Functions in Lisp

Consider a simple function `g`, which takes one argument `x`.

Example (Function in mathematics)

```
g(x)
```

In Lisp, this is written in the following manner:

Example (Function in Lisp)

```
(g x)
```

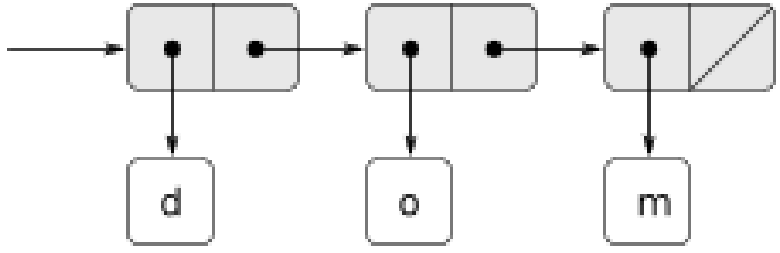
Lists

Consider a simple array of characters in Java:

Example (Array in Java)

```
char[] myArray = ['d', 'o', 'm'];
```

In Lisp, we can write the elements as symbols, and our data structure is a `list`.



Example (List in Lisp)

```
'(d o m)
```

List-manipulating functions

`car` and `cdr` are primitive operations for lists.

- `car` extracts the first element from the list
- `cdr` extracts the rest of the list

Example (car)

```
(car '(d o m)) => 'd
```

Example (cdr)

```
(cdr '(d o m)) => '(o m)
```

Defining a recursive function

Let us define a factorial function in Lisp using recursion:

Example (Factorial function in Lisp)

```
(defun factorial(n)
    (if (= n 0)
        1
        (* n (factorial (- n 1))
    )
)
```

Evaluating a recursive function

Consider the following function call:

Example (Factorial function call in Lisp)

```
(factorial 5)
```

Here is its evaluation path:

Function call	**Evaluation 1**	**Eval 2**	**Eval 3**
`(factorial 5)`	`(* 5 (factorial 4))`	`(* 5 24)`	`120`
`(factorial 4)`	`(* 4 (factorial 3))`	`(* 4 6)`	`24`
`(factorial 3)`	`(* 3 (factorial 2))`	`(* 3 2)`	`6`
`(factorial 2)`	`(* 2 (factorial 1))`	`(* 2 1)`	`2`
`(factorial 1)`	`(* 1 (factorial 0))`	`(* 1 1)`	`1`

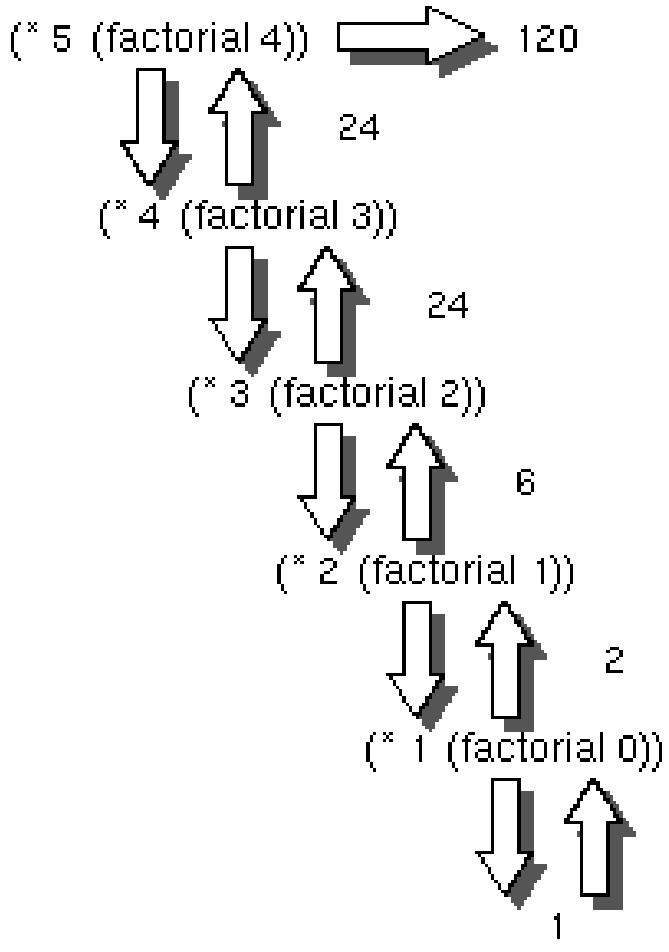
Table: Complete recursive evaluation

Recursive evaluation example explained

Is the argument `n` is equal to 0 (the base case)?

- **YES**: the function evaluates to 1 and terminates.
- **NO**: the function will multiply the argument by the factorial with `n-1` as the argument.

This creates a chain, and evaluates to the factorial of `n`.



Problems in execution

Lisp doesn't run very efficiently on general-purpose hardware. Problems include:

- Evaluating functions
- Environment management
- List representation
- Heap management

Evaluating functions

Lisp has a lot of function calls, and accessing variables is challenging.

- They can be either atoms or other lists.
- There are an unknown number of arguments.
- Functions can be passed as arguments.

Environment management

The environment consists of the current value inside every variable.

- Changes on every function call.
- There can often be many environments.
- Previous environments may need to be accessed: called the *funarg* problem.

List representation

Everything in Lisp is a list, even the program itself.

- Often represented as linked-lists.
- Slow to traverse by general-purpose hardware.
- Variables are untyped, have to be checked before use.

Heap management

Lisp uses a heap to store all variables.

- The heap is managed by Lisp, not the programmer.
- The heap manager needs to ensure computer memory isn't exhausted.
- The hardest part is garbage collection, or reclamation of unused data.

Solutions in Lisp machines

Lisp machines were built with solutions to these problems. Came in three categories:

- *Class-M* machines: Standard computers with special microcode.
- *Class-S* machines: Computers with multiple processors that were specialized for certain tasks.
- *Class-P* machines: Built with multiple processors that were identical and would run code concurrently.

Function evaluation

Function evaluation was largely solved by tagging variable types.

- All three classes allowed for code to be compiled so that the types of arguments were tagged.
- Specific cases were made for up to six arguments.
- Class-P machines would evaluate arguments concurrently.

Managing the environment

Deep binding

- Linked list of name-value pairs
- Searched for every access.
- Has to search whole list in worst-case.

Shallow binding

- Table of name-value pairs.
- Faster function calling.
- Holds less variables; old values overwritten.

Representing linked lists

Many solutions.

- *Vector-coded*: Assumes the list contains no lists, which saves on evaluation time.
- *Structure-coded*: Attaches numbers to each element that allow it to be accessed like an array.
- *Tree-coded*: The list is encoded in a binary tree.
- Numerous other solutions.

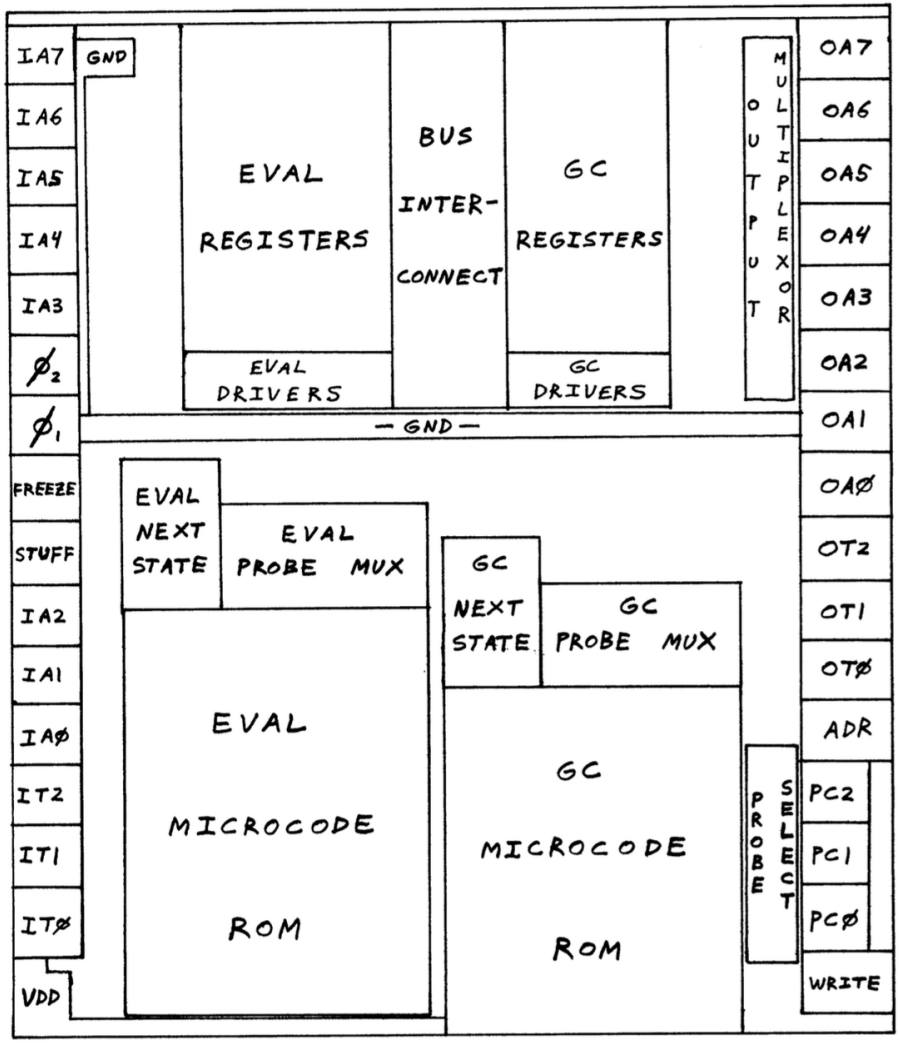
Managing the heap

Garbage collection was the primary concern in heap management design. Two primary solutions:

- *Reference counting*: The number of references for each variable are stored.
- *Marking*: More common; marks data that is fit for reclamation.

Example processor

- Runs Scheme, a simplified version of Lisp (used in SICP).
- Interpreter designed in the hardware with microcode.
- No ALU: the processor only deals with lists.



Legacy of Lisp machines

LISP is now the second oldest programming language in present widespread use (after FORTRAN). Lisp owes its survival to the fact that its programs are lists, which is actually a disadvantage.

- John McCarthy, 1979

- Lisp machines dominated in AI (their most popular domain)
- Worked very well for domain-specific programming
- Introduced new features: Dynamic creation of new objects, dynamically sized-lists, garbage collector
- On-the-fly changes - no need to recompile

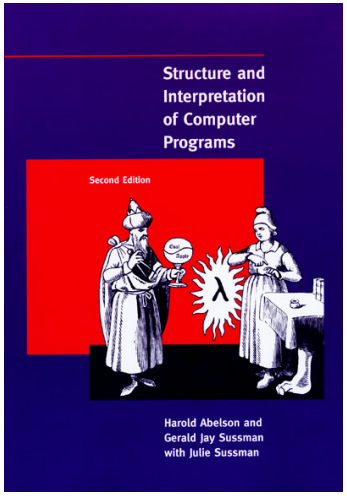
Legacy of Lisp machines

Lisp continues to be a popular tool in

- education
- research
- general programming

Lisp and Lisp machines influenced... Haskell, JavaScript, Lua, Mathematica, ML, Nim, Perl, Python, R, Ruby, Scala, Smalltalk

Structure & Interpretation of Computer Programs is a classic programming textbook, and uses a Lisp variant to teach programming in universities nationwide, including Oakland University.



Legacy of Lisp machines

Though Lisp machines are no longer a commercial enterprise,

- Lisp has continued interest and usage
- Lisp machines serve an important place in history
- Many with interest in retro-computing have recreational interest in Lisp machines

... even many years after the last manufacturers ceased to exist

Figure: These *retro keyboards*, that once were used with then-modern Lisp machines, are today collected by hobbyists.

Thank you!

Questions?