Introduction to Functional Languages

Brandon Marken
Team P-Phunck
Functional Programming: What is it?

• A programming paradigm which treats computation as the evaluation of mathematical functions.
• Produce no side effects.
• Examples: LISP, Scheme, Haskell, Prolog.
History

- Lambda Calculus developed by Alonzo Church at Princeton University in 1936.
- Published in the book *The Calculi of Lambda Conversion* in 1941.
- Provides a method for defining nameless functions.
- Used as the basis for LISP and other functional languages.
History II

• LISP developed at MIT by John McCarthy in 1958
• Designed to solve the “Advice Taker” problem.
• Attempt to teach computers common sense in accomplishing their tasks.
• Current Dialects are Scheme and Clisp
Backus's Comments

- Turing Award Lecture “Can Programming Be Liberated From the Von Neumann Style? A Functional Style and Its Algebra of Programs”
- Argues that modern (circa 1977) programming languages are becoming too bloated and complex.
- Imperative Languages lack useful mathematical properties
- Oblique references to ada.
- Functional Languages produce more reliable and elegant programs.
- Led him to develop the FP (Functional Programming) language.
Basics: Predicate Logic

• Let A and B be $A \land B$

• Let A or B be $A \lor B$

• Let not A be $\neg A$

• Denote a set of propositional statements by
  $$(p_1 \rightarrow e_1, \ldots, p_n \rightarrow e_n)$$

• And T = True and F - False
Lambda Expressions

- Separates defining and naming functions.
- Example: \( \text{Cube}(x) = x \times x \times x \) \( \text{Cube}(2) = 8 \)
- In Lambda Notation
  \[
  \lambda(x) x \times x \times x
  \]
  \[
  (\lambda(x) x \times x \times x)(2) = 8
  \]
- Inadequate for recursive functions
S-Expressions

- Symbolic Expressions.
- Developed to implement recursive functions symbolically.
- Formed by using the characters . ) ( and atomic symbols
- Strings of letters represent atomic symbols.
- Atomic Symbols are S-expressions.
- If A and B are S-expressions so is \((A \cdot B)\)
- Arbitrary length of S-expressions given by
  \[
  (m_1 \cdot (m_2 \cdot (\cdots (m_n \cdot \text{NIL}) \cdots )))
  \]
Shorthand

• (m) stands for \((m \cdot \text{NIL})\)
• We can also substitute \((m_1, \ldots, m_n)\)
• For \((m_1 \cdot (\ldots (m_n \cdot \text{NIL}) \ldots ))\)
M-Expressions

• Functions of S expressions
• Lower case letters for function names.
• Uses the characters [] ;
• S expressions stand for themselves
Elementary M-Functions

- atom\[X\] returns true if X is an atom.
- Eq\[X;Y\] equality operator.
- Only defined if both X and Y are atomic.
- Car\[X\] is defined only if X is non atomic.
- Returns first atom in a given list.
  \[
  \text{car} \left[ (X \cdot A) \right] = X
  \]
- cdr\[X\] defined only is X is non atomic.
- Gives the remaining elements after the first has been removed
Elementary M-Functions II

\[ \text{cdr} \left[ (X \cdot A) \right] = A \]

- Cons[X;Y] defined for any X and Y
- List CONStructor.

\[ \text{cons} [X;A] = (X \cdot A) \]
LISP

• Uses S expressions to represent lists.
• List with elements a b c and d specified as (a b c d)
• Nested lists are specified with the syntax (a b (de) (fg(hi)))
• Function calls done with (function n_1 n_2 .... n_n)
• Example (+ 1 3) adds 1 and 3
Lisp II

• Functions defined using a modified lambda notation.
• (function_name(LAMBDA(arg_1 .....arg_n) expression))
• Example
• (cube(LAMBDA(x) x*x*x)) cubes the result.
LISP III

- Factorial example
- (defun factorial (n)
  - (if (<= n 1)
    - 1
    - (* n (factorial (- n 1))))
- Note similarity to S function factorial example.
References

• http://www-groups.dcs.st-and.ac.uk/~history/Biographies/Church.html


• McCarthy, John “Recursive Functions of Symbolic Expressions” ACM (1960) 184-195