

# Symbolic Computation and Program Manipulation in Lisp

黄 澗石 (Jianshi Huang)

November 27, 2011

- What are the most important principles of good program design
- What defines the expressing power of a programming language (not Turing completeness!)

- What makes Lisp special

# My answer is

- Symbolic Computation
  - Treating expression as value
- Program manipulation
  - Treating code as data, and data as code
- And because it's so easy to do!

# Features already in the language spec

- lexical closures
- symbols
- macros

- defmacro
- define-symbol-macro
- define-compiler-macro
- macrolet and symbol-macrolet

- define-compiler-macro
- &environment

- macrolet
- symbol-macrolet

# We need even more generalization

- I've (and you will) found similar code transformation in many places
- I want ultimate decoupling power
- I want to write general algorithms, but specialized when used (think about C++'s template)

# More Optimization Opportunities

- constant inputs
- domain specific optimization
- compiler assistant

# More Program Manipulation

- pattern matching based code transformation
  - e.g. simplification
- code walker (expand macros)
- function-lambda-expression and inlining
- partial evaluation
- program analysis often needed

- Automatic Programming course from Prof. Novak@UTAustin
  - <http://www.cs.utexas.edu/~novak/cs394p.html>
- Adventures in Advanced Symbolic Programming from Prof. Sussman@MIT
  - <http://groups.csail.mit.edu/mac/users/gjs/6.945/>
  - Unfortunately no lecture notes available (I've asked for them though)
- Prof. Sussman's presentation at StrangeLoop 2011
  - We Really Don't Know How To Compute!  
<http://www.infoq.com/presentations/We-Really-Dont-Know-How-To-Compute>
- Lambda the Ultimate
  - <http://lambda-the-ultimate.org>
- A Hacker's Introduction to Partial Evaluation
  - <http://wry.me/misc/peval.html>