A Multi-Lingual Approach to CS 2

Robert M. Keller
Computer Science Department
Harvey Mudd College

or …

What to do
until the Perfect Language comes along?

Virtual→Real mapping

- ‘CS1’ → HMC CS 5
- ‘CS2’ → HMC CS 60 (or 65*)

- 65 = new version for AP freshmen, combining 60 with elements of 5.

Goals for HMC ‘CS 2’

- Expose students to a broad view of CS, to help them make a choice of major.
- Introduce, or increase depth of understanding, in important concepts of CS.
- Provide motivation and practice for solving more complex problems than most students have encountered thus far.

Some Additional Goals

- Emphasize abstractions
- Highlight connections among apparently diverse concepts, e.g.
  - Object-Oriented vs. Functional Programming
  - Logic vs. Computation
- Big-O complexity, uncomputability reductions

Components of an HMC Degree

- 30%: Core (Math, Physics, Chemistry, Biology, Engineering, CS)
- 30%: Humanities and Social Sciences
- 40%: Major
  - Foundation and Kernel
  - Electives (3 or more)
  - Capstone (1 year of CS Clinic)
- Integrative Experience (consideration of societal impact, etc.)

Assumed Background (from ‘CS1’)

- Imperative and functional programming (using Python).
- Arrays, recursion.
- Some concept of machine model (using toy machine language).
- Some theory (finite-state machines, halting problem).

(Overall HMC Vision)
Foundation Courses

- CS 1: Taken by all students at the college (unless advanced placed)
- CS 2: Taken by all majors, and others with related major or peripheral interest.
- Math 55: Discrete mathematics
- CS 70: Program Development and Data Structures (C++ programming, SVN, etc.)
- CS 81: Computability and Logic

(Overall Vision)
Kernel Courses required for major

- CS 105: Computer Systems (C programming, architecture, OS calls, processes and threads)
- CS 121: Software development (process, as well as UML, patterns, etc.)
- CS 131: Programming languages (semantics, etc. uses ML)

Computer Science

- is largely a science of abstractions:
  - Information representation
  - Algorithm representation
  - Complexity → Computability
  - Communication

Problems, Abstractions, Languages

- Solving a problem is often facilitated by introducing abstractions.
- Abstractions often reusable for other problems as well.
- Any given abstraction is supported in various languages to varying degrees.

Problems, Abstractions, Languages

- Language designs are driven by the abstractions that the languages are designed to support.

- Lack of support for an abstraction may encourage awkward code or libraries with a high entrance barrier.

Abstractions in HMC CS 2

- Sequences as information structures
- Functions (including higher-order functions)
- Objects, inheritance
- Events
- Relations, predicate logic, constraints
- States & transitions
- Grammars
- Computability

Abstraction 1: Functional Programming

- FP is indisputable as a foundation for CS
  - Church, Gödel, Kleene in 1930's
- Some FP seen in CS 1 using Python
- More in CS 2 using Scheme
- No assignment statements allowed
  - Recursion
  - Mutual recursion
  - Anonymous functions (Lambda expressions)
    Higher-order functions

  needed to get anything non-trivial done

Sample FP Problem (week 2): KWIC index

Input:

```
"It is easier to fight for one's principles than to live up to them."
"The only normal people are the ones you don't know very well."
"Love thy neighbor as thyself, but choose your neighborhood."
"I either want less corruption, or more chance to participate in it."
```

KWIC Output:

```
          The only normal people | are the ones you don't know ve : 1
               Love thy neighbor | as thyself, but choose your ne : 2
 Love thy neighbor as thyself, | but choose your neighborhood.  : 2
  want less corruption, or more | chance to participate in it.   : 3
e thy neighbor as thyself, but | choose your neighborhood.      : 2
             I either want less | corruption, or more chance to  : 3
( ... abridged for brevity ... )
                          Love | thy neighbor as thyself, but c : 2
          Love thy neighbor as | thyself, but choose your neigh : 2
 one's principles than to live | up to them.                   : 0
le are the ones you don't know | very well.                    : 1
                      I either | want less corruption, or more  : 3
e the ones you don't know very | well.                         : 1
nly normal people are the ones | you don't know very well.     : 1
eighbor as thyself, but choose | your neighborhood.            : 2
```

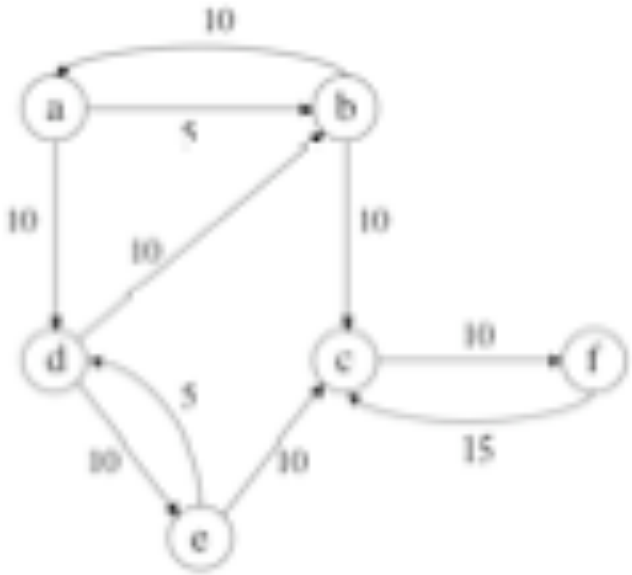
Language: Scheme

- Relatively low entrance barrier
- Good interactive environment
- Clean, elegant, data structuring

- (Later on): Programs as data exercises

Sample FP Problem (week 3): Dijkstra's Shortest Path Algorithm

```
(define graph1 '(
  (a (b 5)   (d 10))
  (b (a 10)  (c 10))
  (c (f 10))
  (d (b 10)  (e 10))
  (e (c 10)  (d 5))
  (f (c 15))
  ))
```



Abstraction 2: Object-Oriented Programming

- Objects are important in contemporary software engineering.
- Especially important in GUI-based applications and visual games.
- We use Java.
- But first …

Implementing Scheme-like Data Structures in Java (week 4)

- Purposes:
  - connect to previous concepts
  - functional model in non-functional language
- “Open-List” model:
  - Linked-list data structure
  - Immutable lists
  - Methods: first, rest, cons, isEmpty
    - both static and instance versions
  - Minimal object orientation at this point

Example Problem: Unicalc (week 5)

- Unit calculator
- Use Open List API in part
- Represent input language using grammar
- Input

  convert 1 meter/(sec sec) to foot/sec/hour

- Ouput

  11811.237000878842

Unicalc Language Grammar

(tokenizer provided)

S → def V E | # E | E

E → P + E | P - E | P

P → U * P | U / P | U

U → (E) | (- E) | Q

Q → D V* | V+

D → a numeric value (Double)

Finally, Mutable Objects

- Example: Maze solving API (week 6)
- Queue and deque implementations

- Spampede 1-player video game (week 7)
  - Applet
  - Event-handling
  - Inheritance

Abstraction 3: Logic Programming

- Database querying using logic
- We use Prolog.
- Constraint problem solving
- Back-tracking
- Difficult to accomplish in ordinary languages

Example Problems

- Movie database querying (week 8)
- Generalized “24” puzzle (week 9)

  Sudo-graph solver (generalizes sudoku)

  Logic puzzle solver

  Generalized (n > 3) Towers of Hanoi solver

Movie Database Problems

Problem: Complete the definitions of Prolog predicates that respond to the following queries, made to the Prolog database providing the following predicates:

movie([Title, Year], Director, [... categories])

actress(Name, [Birth City, Birth State], Birthyear)

actor(Name, [Birth City, Birth State], Birthyear)

plays(Player, Role, [Title, Year])

playedNotDirected(Director) iff Director directed some movies, but played in at least one movie he/she did not direct.

“24” Problem

- Create a program that will accept:
  - A list of numbers
  - A list of operators (e.g. +, *, -)
  - A target
- That computes all possible expressions that evaluate to the target, using each number exactly once.

Week 10 Problem

- Using Prolog's Definite-Clause Grammar framework, implement a compiler for a toy language and machine.

Abstract Machines

- NFA to DFA conversion
- Non-regular language determination (Myhill-Nerode equivalence classes)
- (week 11) Language recognition problems using JFLAP (compare DFA to Turing machines) or, in the future, PFLAP.

More Theory

- Algorithm analysis
- Halting problem
- Reductions from halting problem:
  - Blank-tape halting problem
  - Existential halting problem

.

.

.

Algorithm Analysis and Big-O Complexity (week 12)

- Pencil-and-paper exercises
- Analyzing a shortest path program
- Analyzing bignum multiplication

Conclusions

- Abstractions are helpful to bridge the gap between problem solution and language capability.
- Languages don’t (yet) support abstractions uniformly.
- Language choices should be driven by abstractions, rather than the other way around.

Related to Software Engineering

- Dependency Inversion Principle (Robert C. Martin):
  - Details should depend on abstractions; Abstractions should not depend on details, but at most on other abstractions.
  - High-level modules should not depend on low-level modules; both should depend on abstractions.
  - Specify the interface first, then implement.

Web References

- http://www.cs.hmc.edu/courses/2008/spring/cs60/