CSci 4223
Principles of Programming Languages

Lecture 10

Prof. Clarkson
Spring 2013

Review
• Features learned: functions, tuples, lists, let expressions, options, records, datatypes, case expressions, type synonyms, pattern matching, exceptions, type variables, higher-order and anonymous functions
• Today:
  – Course motivation (finally)
  – Type constructors

Logistics
• HW3 delayed
• HW and exam solutions available
  – Come to Prof. Clarkson’s office hours. Bring your marked-up solutions. Be prepared to discuss mistakes before seeing solutions.
• Regular recitation section? Poll on Piazza
• 6223 (MS) students: speak to Prof. Clarkson after class today about additional work required of grad students in course
• Return Midterm 1 at end of class

COURSE MOTIVATION

Course motivation
(Did you think I forgot? 😊)
• Why learn languages that are quite different from Java, C, C++?
• Why learn the fundamental concepts that appear in all/most languages?
• Why focus on functional programming?
• Why use SML and Ruby in particular?

...what follows: several observations. Some may work more or less well for each individual.

Observation 1
Programming languages are like cars and shoes.
Programming languages are like cultures.
Programming languages are like works of art.
What is the best kind of car?

What is the best kind of shoes?

Cars and Shoes

Cars are used for rather different things:

- Winning the INDY 500
- Taking kids to soccer practice
- Off-roading
- Hauling a mattress
- Getting the wind in your hair
- Staying dry in the rain

Shoes:

- Playing basketball
- Going to a formal
- Going to the beach

Learning to work on cars

- To learn how cars work, it might make sense to start with a classic design rather than the latest model
- A popular car may not be a good car for learning how cars work
- Even better analogy: beginning pilots don't fly 747's.

- A good mechanic might have a specialty (2004 Honda Civics), but also understands how all cars work.
- They don't get hung up on the syntax (e.g., upholstery color)

Are all cars the same?

- They all have steering wheels, brakes, windows, headlights, ...
- Makes it easy to rent cars
- But it can be uncomfortable to "learn" a new car
- So are all PLs the same?

Cultures

- People are people
- But cultural diversity is rich and enriching
  - Travel, learn new customs, languages, values, foods, ...
  - Your life is improved by educating yourself about other cultures, maybe adopting some of their practices yourself
  - e.g., travel in the ML culture for awhile, bring back "home" to Java some new experiences that make you a better programmer

Art

Shakespeare's Hamlet:

- Beautiful work of art
- Teaches deep, eternal truths
- Source of well-known sayings

But:

- Strange syntax
- Not modern
- Not popular
- Won't land you a summer job

Yet familiarity with Shakespeare is part of being an (English-)educated person!

Familiarity with beautifully designed PLs is part of being an educated computer scientist.
Observation 2

Programming languages are tools.

Are all tools the same?

Yes:
- Any input-output behavior implementable in language X is implementable in language Y [see Turing completeness, CSci 3313]
- Java, ML, and a language with one loop and three infinitely-large integers are “the same”

No:
- Some computations easy to express in one language, but awkward to express in another
- Some language features built-in, others have to be “coded up”

…choose the right tool for the job!

Observation 3

Semantics do matter. Really!

Practical questions

- What libraries are available for reuse?
- What can get me a summer internship?
- What does my boss tell me to do?
- What is the de facto industry standard?
- What do I already know?

Why semantics (and idioms)?

This course focuses as much as it can on semantics (and idioms)

- Correct reasoning about programs, interfaces, and compilers requires a precise knowledge of semantics
  - Not “I feel that conditional expressions might work like this”
  - Not “I like curly braces more than parentheses”
  - Much of software development is designing precise interfaces; what a PL means is a really good example

- Idioms make you a better programmer
  - Best to see in multiple settings, including where they shine
  - You’ll understand (insert your favorite language here) better even if I never show you equivalent idioms in that language
Observation 4

Functional languages predict the future.

Functional Programming

Okay, so why is 75% of course with functional language where:
- Mutation is discouraged
- One-of types via constructs like datatypes
- Higher-order functions are very convenient

Because:
1. These features are invaluable for correct, elegant, efficient software (great way to think about computation)
2. Functional languages have always been ahead of their time
3. Functional languages well-suited to where computing is going

Most of course is on (1), so a few minutes on (2) and (3) ...

Ahead of their time

All of these were dismissed as "beautiful, worthless, slow things PL professors make you learn in school"
- Garbage collection (Java didn't exist in 1995, LISP did)
- Generics (List<T> in Java didn't exist in 2003, 'a list in ML [1996] did)
- Higher-order functions (C# 3.0 [2007], Java 8 [2013?], Lisp [1958])
- Recursion (a big fight in 1960 about this – I'm told 😊)
- Type inference (C# 3.0)
- MapReduce (everybody)

What will the next "discovery" be?
- Maybe pattern-matching?
- "To conquer" vs. "to assimilate"

Observation 5

Functional languages are actually used in the real world.

Recent Surge

- F#, C# 3.0, LINQ (Microsoft)
- Scala (Twitter, LinkedIn, FourSquare)
- Java 8
- Haskell (dozens of small companies/teams)
- Erlang (distributed systems, Facebook chat)
- OCaml (JaneStreet)
- ...

Full disclosure: SML is showing its age, but OCaml and FP are very similar

Why a surge?

Some guesses:
- Concise, elegant, productive programming
- Javascript, Python, Ruby helped break the Java/C/C++ hegemony
- Avoiding mutation is the best way to make concurrent and parallel programming easier
**Is this real programming?**

- "Use of languages in this course seems silly"  
  - Precisely because lecture and homework focus on interesting language constructs, not on writing big programs
- "Real" programming needs file I/O, string operations, floating point, graphics, project managers, testing frameworks, threads, build systems, ...  
  - Functional languages have all that and more
- Note: If we used Java as an example language instead of ML, you'd develop mistaken idea that Java is silly!

**Observation 6**

SML and Ruby are exemplars.

**Why SML and Ruby?**

- ML: pattern matching, type inference, parametric polymorphism, abstract types and modules
- Ruby: classes but not types, very OOP, mixins

Really wish we had more time:

- Racket: dynamic typing, "good" macros, minimalist syntax, eval
- Haskell: laziness, purity, type classes, monads
- Prolog: unification and backtracking
- and much more...

**Orthogonal language features**

<table>
<thead>
<tr>
<th></th>
<th>Dynamically typed</th>
<th>Statically typed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional</td>
<td>—</td>
<td>SML</td>
</tr>
<tr>
<td>Object oriented</td>
<td>Ruby</td>
<td>Java</td>
</tr>
</tbody>
</table>

Could fill in missing cell with Racket or Scheme

**Summary**

- No such thing as a "best" PL
- There are good general design principles for PLs
- A good language is a relevant, crisp interface for writing software
- Software leaders should know PL semantics and idioms
- Learning PLs is not about syntactic tricks for small programs
- Functional languages have been on the leading edge for decades  
  - Ideas get absorbed by the mainstream, but very slowly  
  - Meanwhile, use the ideas to be a better C/java/PHP hacker

**INFIX OPERATORS**
Cons is infix, not prefix

- Why do we get to write cons inside as 1::2::3 instead of outside as ::(1, ::(2, ::(3, nil)))?
- Answer: not because infix is "baked into" ML. Actually because of one missing piece of list definition.

Same for append operator @

Can make your own infix operators

fun plus (x, y) = x + y
infix plus
val seven = 3 plus 4

(List is a silly example; we'll see a really good one when we talk about function composition)

List is not a type

[] : 'a list
:: : 'a * 'a list -> 'a list
map : ('a -> 'b) * 'a list -> 'b list

- Types: 'a list, int list, (int->int) list, etc.
- Not a type: list
- So what is list?

Stacks as lists

datatype 'a stack = S of 'a list
val empty_stack = S []
fun push (x, S s) = S (x::s)
fun top (S []) = raise Empty
| top (S (x::_)) = x
fun pop (S []) = raise Empty
| pop (S (_::s)) = S s
Maps as lists

datatype (''k,'v) map = M of (''k * 'v) list
exception NotFound;
val empty_map = M []
fun put (k, v, M m) = M ((k,v)::m)
fun exists (k, M []) = false
| exists (k, M ((k', v)::m)) = if k=k'
    then true
    else exists(k, M m)
fun get (k, M []) = raise NotFound
| get (k, M ((k',v)::m)) = if k=k'
    then v
    else get(k, M m)