

PA4 : Midnight Sun

CSE 130: Fall 2010

Programming Languages

Lecture 10: Deconstructing ML

Ranjit Jhala
UC San Diego



Key components of a PL

- Units of computation
- Types
- Memory model

Deconstructing Ocaml

The anatomy of a Programming Language

Units of computation

In OCaml

EXPR

In Ocaml: Expressions

Expressions that evaluate to values

Everything is an expression

- if-then-else
- let-in
- match
- fun x -> x+1
- e1 e2

In Java/Python/C/C++

assignment

field
obj

value : ML ~ obj:pyth
expr : ML ~ ? : pyth

In Java/Python/C/C++

- Store and update commands

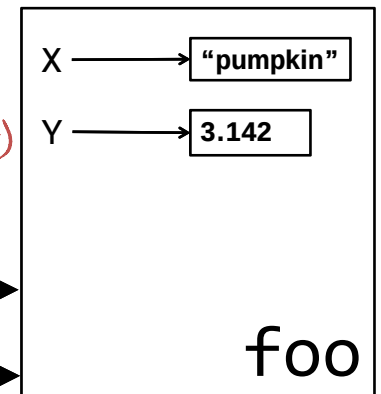
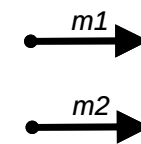
x.set("pumpkin")

```
x := e "umpkin"  
y := 1.571  
y := 2*y
```

y.set(1.57)

- Message sends

```
foo.m1(...)
```



In Prolog

Types

num + string

In Prolog: Facts and Rules

Logical facts + Inference Rules

Mexican(CARNITAS) "Fact"

Food(CARNITAS) "Fact"

~~✗~~ Mexican(X) $\wedge$ Food(X) $\Rightarrow$ Delicious(X) "Rule"

Delicious(CARNITAS) "Fact"

Types = Classification

Of "things" created by programmer

Restrict what can be done with/to things

[3]

(+) 3 5

(3 5)

Thing: int *list* As Arg to (+) As function

In OCaml: Static typing

Types assigned **statically** (at compile time)

- Without computing values
- Rules state when expressions are well-typed

$$\frac{e1 : T1 \rightarrow T2 \quad e2 : T1}{e1 \ e2 : T2}$$

In Python: Dynamic typing

Types assigned when to data computed at run-time (i.e. “dynamically”)

Before an operation is performed, check operands compatible with data

```
def f(x): return x.foo()
```

f(z)

f(p)

f(q)

X + y

plus--(y)

In OCaml: Static typing

How to **reuse code** for different types?

- Parametric types: *'a * 'b -> 'b * 'a*
- Implicit “For all”

Type **inferred** automatically from code

- Less burden on the programmer

In Python: Dynamic typing

More programs accepted by compiler

⇒ More flexible

```
[1, "abc", 1.8, [ "efg", 20]]
```

```
let x = if b then 1 else "abc"  
let y = if b then x+1 else x^"efg"
```

Dynamic vs. Static, OO vs. Func

	Statically typed	Dynamically typed
OO/ <i>imp</i>	<i>Java/C#</i>	<i>SMALLTALK</i> <i>Python</i>
Functional	<i>ML</i>	<i>USE</i> <i>CLOSURE</i> <i>SCHEME</i> <i>ERLANG</i>

Dynamic vs. Static, OO vs. Func

	Statically typed	Dynamically typed
OO	Java	Python, Smalltalk
Functional	Ocaml, <u>Haskell</u>	Lisp/Scheme

Polymorphism

PL that is polymorphic + dynamically typed ?

Every dynamically typed PL is polymorphic

Functions work on **every valid** data

*a * b → b * a*

Explicit polymorphism in statically typed PL

Require general (poly) type at compile time



Memory/Data model

Or, what do variables refer to?

Data model in functional langs

Environment of bindings (phonebook)

x	4 : int
y	64 : int
z	[4;64;68] : int list
x	8 : int

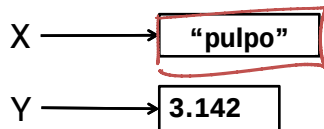
Never change a binding

Add new bindings at the **end**

Data model in Imp/OO langs

Variables are **named cells in memory**

Variables = **names** of objects on the heap



Can **change** them by **assigning** into them

x := "pulpo"

Data model in functional langs

Variables = names in phonebook

Most recent entry looked up during eval

Closures = Function Value

Environment “frozen” inside closure

- function behavior cannot be changed
- easier reasoning
- easier debugging

Data model in Prolog

Variables = **unknowns** to solve for

Mexican(CARNITAS)

Food(CARNITAS)

$\forall X \text{ Mexican}(X) \wedge \text{Food}(X) \Rightarrow \text{Delicious}(X)$

Delicious(Y)?

Q: What is delicious?

A: CARNITAS!

Final words on functional programming

What's the point of all this?

Advantages of functional progs

- Functional programming more concise
“one line of lisp can replace 20 lines of C”
(quote from <http://www.ddj.com/dept/architect/184414500?pgno=3>)
- Recall reverse function:

```
let reverse = fold (::) [];;
```
- How many lines in C, C++?

Better reasoning about Programs

- No “side effects”
 - Same inputs return same outputs

List.map f [—]
- Can safely **reorder** computations
- Can safely **parallelize** computations

Functional $\Rightarrow$ Parallelism $\Rightarrow$ Scale

MapReduce (Google)

- Map + Fold
- Hadoop [Yahoo], Hive [Facebook], Dryad [MS]
- For more, take [Prof. Vahdat's CSE 124](#)

Erlang (FaceBook)

Scala (Twitter)

So what?

- Form the authors: “Inspired by similar primitives in LISP and other languages”
<http://labs.google.com/papers/mapreduce-osdi04-slides/index-auto-0003.html>
- Programmers who only know Java etc. would probably not have come up with this idea

Remember

Say hello to Python



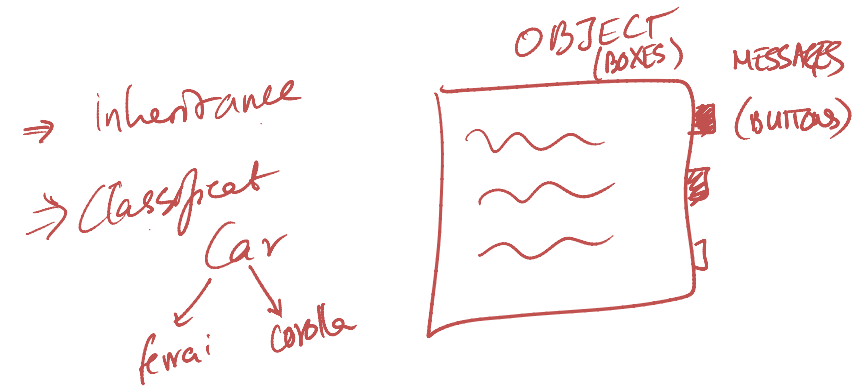
“Free your mind”
-Morpheus

Recap of the course so far

- 4+ weeks of functional with OCaml
- Next: 3 weeks of OO with Python

OO at the highest level

What is OO programming?



OO at the highest level

What is OO programming?

- objects (duh)
- message sends
- dynamic dispatch

Just to Whet Your Appetite

Say we have objects, e.g.

car, duck, pig, cell_phone

X → make_some_noise()

Say we have a message,

make_some_noise

Just to whet your appetite

Each object has own implementation of **make_some_noise** (called “methods”)

- **car**: vroom vroom,
- **pig**: oink oink,
- **duck**: quack quack

Lisp	50's	NO
C++	80	YES
Fortran	50s	NO
Smalltalk	70s	YES
Smalltalk67	67	YES

Can send **make_some_noise** to any object
Depending on the “receiving” object (at run-time)
we'll get a different noise!

Oh btw...

- What's the difference between message and method...
- Message is just the name of the message, method is the implementation
- Message is the “interface” / “prototype” of the method.

Oh btw...

- What's the difference between message and method...

Core Ideas of OO programming

Message

The name of an operation

Method

The implementation of an operation

Dynamic Dispatch

Dynamic type of receiver determines which method is run for given message send

This brings us to Python...

...our vehicle for OO programming

- Fun and Useful PL
- Let's compare with OCaml along some of the dimensions we saw last time

Python

- Python has a very relaxed philosophy
 - if something "can be done" then it is allowed.
- Dynamic types + Everything is an object
 - very flexible
 - very intuitive

OCaml/Python comparison

	ML	Python
PL paradigm		
Basic unit		
Types		
DataModel		

No static types

- **No static type system** to "prohibit" operations.
- No more of that OCaml compiler giving you hard-to-decipher error messages!

No static types: but what instead?

- **Dynamic typing** *Guido Van Rossum*
- At runtime, every "operation" is translated to a method call on the appropriate object. If the object supports the method, then the computation proceeds.
- Duck-typing: if it looks like a duck, quacks like a duck, then it is a duck!

Similarities to ML

Uniform model

Everything is an object, including functions

Pass functions around

Functions are objects!

o-- call --

Supports functional programming

map and fold

Dynamic typing

- This loose, comfortable, free-style, philosophy is at the heart of python.
- But... beware, you can get burned with this flexibility...
- Q: how many times did OCaml complain to you statically about something that was **NOT** a bug?

Other cool things about Python

- A lot of stuff that you may first think is a "language feature" is actually just translated under the hood to a method call...
- Very widely used, supported.
- Has libraries for all sorts of things.

Ok, let's start playing with Python!

- Like Perl, python is a "managed" or "interpreted" language that runs under the python environment, i.e. not compiled to machine code.
- Makes it convenient to rapidly write, check-in and test code!

Let's fire it up!

See lec10.py file for the rest...

Ways to run Python code

- At an interactive Python prompt
 - like "read-eval-print" loop of ML
- As shell scripts
- As stand-alone programs
 - run from the shell.