Deconstructing Ocaml

The anatomy of a Programming Language

Key components of a PL

• Units of computation

• Types

• Memory model
In OCaml

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++

• Store and update commands

 assignment

\begin{align*}
\text{field} & \quad \text{obj} \\
\text{value} : \text{ML} \sim \text{obj}:\text{py} & \quad \text{expr} : \text{ML} \sim \text{?s:py}
\end{align*}

In Java/Python/C/C++

• Message sends

\begin{align*}
\text{foo.m1(...)} & \quad \text{m1} \\
\text{foo.m2(...)} & \quad \text{m2}
\end{align*}
In Prolog

Logical facts + Inference Rules

Mexican(CARNITAS)  “Fact”
Food(CARNITAS)  “Fact”
Mexican(X) ∧ Food(X) ⇒ Delicious(X)  “Rule”
Delicious(CARNITAS)  “Fact”

Types = Classification

Of “things” created by programmer

Restrict what can be done with/to things

Types = Classification

Types

num + str
In OCaml: Static typing

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

- Without computing values

- Rules state when expressions are well-typed

\[
\frac{e_1 : T_1 \to T_2 \quad e_2 : T_1}{e_1 \ e_2 : T_2}
\]

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

\[
\text{def } f(x) : \text{ return } x.\text{foo()}
\]

More programs accepted by compiler ⇒ More flexible

\[
[1, \text{"abc"}, 1.8, [\text{"efg"}, 20]]
\]

let x = if b then 1 else "abc"
let y = if b then x+1 else x^"efg"
## Dynamic vs. Static, OO vs. Func

<table>
<thead>
<tr>
<th></th>
<th>Statically typed</th>
<th>Dynamically typed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OO/IMP</strong></td>
<td>Java/C#</td>
<td>SMALLTALK</td>
</tr>
<tr>
<td>Functional</td>
<td>ML</td>
<td>USE CLOSURE SCHM ERLANG</td>
</tr>
</tbody>
</table>

## Polymorphism

PL that is polymorphic + dynamically typed?

*Every* dynamically typed PL is polymorphic

Functions work on *every valid* data

![Example](eq:polymorphism_example)

Explicit polymorphism in statically typed PL

Require general (poly) type at compile time

## Memory/Data model

Or, what do variables refer to?

---

Ocaml, Haskell

Lisp/Scheme
Data model in functional langs

Environment of bindings (phonebook)

<table>
<thead>
<tr>
<th>x</th>
<th>4 : int</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>64 : int</td>
</tr>
<tr>
<td>z</td>
<td>[4;64;68] : int list</td>
</tr>
<tr>
<td>x</td>
<td>8 : int</td>
</tr>
</tbody>
</table>

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

\[ X \rightarrow \text{“pulpo”} \]
\[ Y \rightarrow 3.142 \]

Can change them by assigning into them

\[ x := \text{“pulpo”} \]

Data model in Prolog

Variables = unknowns to solve for

\[ \forall X \text{ Mexican}(X) \land \text{Food}(X) \Rightarrow \text{Delicious}(X) \]

Q: What is delicious?
A: CARNITAS!
Final words on functional programming

Advantages of functional progs

- Functional programming more concise
  “one line of lisp can replace 20 lines of C”

- 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 [1..]
  ```

- Can safely reorder computations

- Can safely parallelize computations
**Functional ⇒ Parallelism ⇒ 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”
  

- Programmers who only know Java etc. would probably not have come up with this idea

---

**Remember**

- “Free your mind”
  - Morpheus

---

**Say hello to Python**
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?

- objects (duh)
- message sends
- dynamic dispatch

Just to Whet Your Appetite

Say we have objects, e.g.
car, duck, pig, cell_phone

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

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...

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

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.
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

### OCaml/Python comparison

<table>
<thead>
<tr>
<th></th>
<th>ML</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL paradigm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic unit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Types</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DataModel</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 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

### 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**
  - 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!

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?

Similarities to ML

Uniform model
Everything is an object, including functions

Pass functions around
Functions are objects!

Supports functional programming
map and fold

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!

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.

Let’s fire it up!

See lec10.py file for the rest...