News

- Programming Assignment 4 is up, due Monday Feb 16, 5pm
- Long assignment, start early
- Midterm Thursday Feb 12
  - covers only ocaml part of the course, assignments 1 through 3.

Recap: Inferring types

- Introduce unknown type vars
- Figure out equalities that must hold, and solve these equalities

Example 5

```ocaml
let rec map f l =
  match l with
  | [] -> []
  | h::t -> (f h)::(map f t)
```

Example 5

```latex
\begin{align*}
\text{let rec } & \quad \text{map } f \ l = \\
& \quad \text{match } l \text{ with } \\
& \quad \quad \text{[]} \rightarrow \text{[]} \\
& \quad \quad | h::t \rightarrow (f \ h)::(\text{map } f \ t)
\end{align*}
```
Inferring types with ‘a

• Introduce unknown type vars

• Figure out equalities that must hold, and solve these equalities

• Remaining types vars get a forall and thus become the ‘a, ‘b, etc.

Example 6

```ocaml
let compose (f, g) x = f (g x)
```

Example 7

```ocaml
let rec fold f cur l = match l with
  [] -> cur
| h::t -> fold f (f h cur) t
```
Example 7

let rec fold f cur l =
  match l with
  | [] -> cur
  | h::t -> fold f (f h cur) t

Deconstructing OCaml

What makes up a language

Key components of a lang

- Units of computation
- Types
- Memory model
<table>
<thead>
<tr>
<th>In OCaml</th>
<th>In OCaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Expressions that evaluate to values</td>
<td></td>
</tr>
<tr>
<td>• Everything is an expression</td>
<td></td>
</tr>
<tr>
<td>• int, bool, real</td>
<td></td>
</tr>
<tr>
<td>• if-then-else</td>
<td></td>
</tr>
<tr>
<td>• let-in</td>
<td></td>
</tr>
<tr>
<td>• match</td>
<td></td>
</tr>
<tr>
<td>• fn x -&gt; x+1</td>
<td></td>
</tr>
<tr>
<td>• e1 e2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>In Java/Python</th>
<th>In Java/Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Store and update commands</td>
<td></td>
</tr>
<tr>
<td>• Message sends</td>
<td></td>
</tr>
</tbody>
</table>
In Prolog

• Logical facts
• Inference rules

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

Types

• Used to classify things created by the programmer
• Classification used to check what can be done with/to those things
**In OCaml: Static typing**

- Types are assigned statically at compile time

- Without computing values

- Rules state when expressions are type-correct

\[
\begin{align*}
e1 : T_1 &\rightarrow T_2 \\
e2 : T_1 &\rightarrow T_2 \\
e1 \ e2 &: T_2
\end{align*}
\]

**In Python: Dynamic typing**

- Types assigned to values/objects as they are computed, ie: dynamically

- Before an operation is performed, check that operands are compatible with operation

\[
[1, \text{"abc"}, 1.8, \text{["efg","abc"]}] \\
\text{let } x = \text{if } b \text{ then 1 else "abc"} \\
\text{let } y = \text{if } b \text{ then } x + 1 \text{ else } x ^ \text{"efg"}
\]

**In OCaml: Static typing**

- How can one reuse code for different types?
  - parametric types: ‘a * ‘b -> ‘b * ‘a
  - implicit forall

- Type “discovered” (inferred) automatically from code
  - less burden on the programmer
### 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</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Functional</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Polymorphism

- Name a language that is dynamically typed, but not polymorphic

- Every dynamically typed language is polymorphic
  - functions just simply work on any datatype that can be operated on at runtime

- Only need explicit polymorphism in statically typed languages to assign at compile time a suitably general polymorphic type

### Memory/Data model

aka: what do variables refer to?
Data model in functional langs

- Environment of bindings (phonebook)
  - Never change a binding
    - add new bindings at the end of the phonebook

Data model in OO langs

- Variables are cells in memory
- Can change them by assigning into them
- Variables point to objects on the heap
- \( x = x + 10 \)

Data model in Prolog

- Variables in Prolog are unknowns to solve for
  - Mexican(CARNITAS)
  - Food(CARNITAS)
  - \( \forall X \text{ Mexican}(X) \land \text{Food}(X) \Rightarrow \text{Delicious}(X) \)
  - Delicious(Y)?

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 in OCaml:
  
  `let reverse = fold (::) [];`

- How many lines in C, C++?

Don’t be fooled

- Some of the programming assignments made you do certain things using fold in order to force you to think about it, even though using fold was not the easiest way to do it.

- But there are many cases where map and fold make life A LOT EASIER.
Can better reason about progs

- No side effects. Call a function twice with same params, produces same value
- As a result, computations can be reordered more easily
- They can also be parallelized more easily

So what?

- Form the authors: “Inspired by similar primitives in LISP and other languages”
- The point is this: programmers who only know Java/C/C++ would probably not have come up with this idea
- Many other similar examples in industry

Remember

- The next time you use google, think of how functional programming has inspired some of the technical ideas behind their engine
- And of course:
  “Free your mind”
  -Morpheus