News

• PA4 is up (due 5/6)
  - Week *AFTER* midterm

• Midterm: Tue 5/3
  - In class, open book, etc.

Datatypes with many type variables

- Multiple type variables

```plaintext
type ('a,'b) tree =
  Leaf of ('a * 'b)
|  Node of ('a,'b) tree * ('a,'b) tree
```

- Type is instantiated for each use:

  - `Leaf("joe",1) : (string,int) tree`
  - `Leaf("william",2) : (string,int) tree`
  - `Node(...) : (string,int) tree`
  - `Node(Leaf("joe",1),Leaf(3.14, "pi"))`

Polymorphic Data Structures

- Container data structures independent of type!
- Appropriate type is *instantiated* at each *use*:

```plaintext
'a list
  ("a" : 'b) tree
  ("a" : 'b) hashtbl ...
```

- Appropriate type instantiated at use
  - No *unsafe* casting as in C/C++/Java

- Static type checking catches errors early
  - Cannot add int key to string hashtable

- Generics: in Java,C#,VB (borrowed from ML)
Other kinds of polymorphisms

- That was OCaml...

- But what about other kinds of polymorphisms..

Sub-type polymorphism

```java
void f(Shape s)
- Can pass in any sub-type of Shape
```

Parametric polymorphism

```java
void proc_elems(list[T])
- can pass in ANY T
- this is the kind in OCaml!
```

Bounded polymorphism

```java
void proc_elems(list[T]) T extends Printable
- Hey... isn’t this subtype polymorphism?
- No, for example:

```java
bool ShapeEq(T a, T b) T extends Shape
- Can call on
  • (Rect, Rect)
  • (Circle, Circle)
- But not (Rect, Circle)
```

Summary of polymorphism

- Subtype

- Parametric

- Bounded = Parametric + Subtype
- In Java/C#
Back to OCaml

- Polymorphic types allow us to reuse code
- However, not always obvious from staring at code
- But... Types never entered w/ program!

Another In-Class Exercise!

Polymorphic Types

- Polymorphic types are tricky
- Not always obvious from staring at code
- How to ensure correctness?
- Types (almost) never entered w/ program!

Type inference

aka: how in the world does Ocaml figure out all the types ???
Polymorphic Type Inference

- Computing the types of all expressions
  - At compile time: Statically Typed

- Each binding is processed in order
  - Types are computed for each binding
  - For expression and variable bound to
  - Types used for subsequent bindings

- How is this different from values?
  - Values NOT computed statically (e.g. fun values)

Example 1

\[
\begin{align*}
\text{let } x & = 2 + 3; \\
\text{let } y & = \text{string_of_int } x;
\end{align*}
\]

Example 2

\[
\begin{align*}
\text{let } x & = 2 + 3; \\
\text{let } y & = \text{string_of_int } x; \\
\text{let } \text{inc } y & = x + y;
\end{align*}
\]
Example 3

```ocaml
let foo x =
  let (y,z) = x in
  z-y
;;
```

Example 4

```ocaml
let rec cat l =
  match l with
  | [] -> ""
  | h::t -> h^(cat t)

ML doesn't know what the function does, or even that it finishes only its type!
```

Example 4

ML doesn't know what the function does, or even that it finishes only its type!

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

('a -> 'b) -> 'a list -> 'b list
```

Example 5

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

Introduce unknown tyvar: Unify,solve, Remaining tyvar gets a “forall”
```

Wednesday, April 20, 2011
Example 6

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

Example 7

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

```
let foo1 f g x =
  if f x
  then x
  else g x
```

```
let foo2 f g x =
  if f x
  then x
  else foo2 f g (g x)
```