Programming Languages and Paradigms
COMP 302

Instructor: Jacob Errington
School of Computer Science
McGill University

Polymorphism, recursive types, higher-order functions

Recap: constructors (with fields)

- Constructors can have fields. A field holds some data together with the constructor, e.g.

```
1 type shape =
2   | Circle of float
3   | Square of float
4   | Rect of float * float
5   (* ^ separate fields with * as if a tuple *)
```

Recap: constructors (with fields)

- Constructors can have fields. A field holds some data together with the constructor, e.g.

```
type shape =
  | Circle of float
  | Square of float
  | Rect of float * float
  (* ^ separate fields with * as if a tuple *)
```

- Pattern matching syntax can extract the values of fields from the constructor.

```
let area (s : shape) : float = match s with
  | Circle r -> 3.14 *. r *. r
  | Square c -> c *. c
  | Rect (w, h) -> w *. h
```

This time...

- Polymorphism, aka “generics”
- Lists
- Higher-order functions!

This is going to be a big one!

This time...

- Polymorphism, aka “generics”
- Lists
- Higher-order functions!

This is going to be a big one!

Buckle up.

Polymorphic types

demo

Recap: polymorphism

- Any type involving *type variables* (e.g. `'a`, `'b`) is polymorphic, e.g. `'a -> 'a -> 'a`
- Usually pronounced as greek letters “alpha”, “beta”; but also “tick A”, “tick B”; and also simply “A” and “B”.

Recap: polymorphism

- Any type involving *type variables* (e.g. `'a`, `'b`) is polymorphic, e.g. `'a -> 'a -> 'a`
- Usually pronounced as greek letters “alpha”, “beta”; but also “tick A”, “tick B”; and also simply “A” and “B”.
- OCaml infers the most general (generic) type possible.

Recap: polymorphism

- Any type involving *type variables* (e.g. `'a`, `'b`) is polymorphic, e.g. `'a -> 'a -> 'a`
- Usually pronounced as greek letters “alpha”, “beta”; but also “tick A”, “tick B”; and also simply “A” and “B”.
- OCaml infers the most general (generic) type possible.

```
1 let f x = x (* has type: 'a -> 'a *)
2 let f y x = y (* has type: 'a -> 'b -> 'a *)
3 let f (x, _) = x (* has type : 'a * 'b -> 'a *)
```

Recap: polymorphism

- Any type involving *type variables* (e.g. `'a`, `'b`) is polymorphic, e.g. `'a -> 'a -> 'a`
- Usually pronounced as greek letters “alpha”, “beta”; but also “tick A”, “tick B”; and also simply “A” and “B”.
- OCaml infers the most general (generic) type possible.

```
1 let f x = x (* has type: 'a -> 'a *)
2 let f y x = y (* has type: 'a -> 'b -> 'a *)
3 let f (x, _) = x (* has type : 'a * 'b -> 'a *)
```

- This constrains the possible (reasonable) implementations. Could functions than others than those above have been implemented at those types?

Generic types: option and list; recursive types

demo

Recap: generic types, recursive types

- Generic type marked by a type parameter left of name

```
1 type 'a option = None | Some of 'a
2 type 'a mylist = Nil | Cons of 'a * 'a mylist
```

Recap: generic types, recursive types

- Generic type marked by a type parameter left of name

```
1 type 'a option = None | Some of 'a
2 type 'a mylist = Nil | Cons of 'a * 'a mylist
```

- `'a mylist` is a recursive type; it refers to itself.
  - `Nil` has type `'a mylist` for any type `'a`.
  - `Cons (x, xs)` has type `'a mylist`
    provided that `x : 'a` and `xs : 'a mylist`

Built-in list type and higher-order functions

demo