Parametric Polymorphism

Ziyan Maraikar

August 1, 2014

Lecture Outline

Generalising functions

- ★ Sometimes functions we write should be able to work with multiple types, e.g. swap the values in a tuple, sorting and searching

[1]unlike void* in C

Generalising functions

- ★ Sometimes functions we write should be able to work with multiple types, e.g. swap the values in a tuple, sorting and searching
- ★ Polymorphism is a feature that lets us accomplish this in a type-safe fashion[1].

[1] unlike void* in C

Generalising functions

- ★ Sometimes functions we write should be able to work with multiple types, e.g. swap the values in a tuple, sorting and searching
- ★ Polymorphism is a feature that lets us accomplish this in a type-safe fashion[1].
- ★ Type definitions can also be made polymorphic for added flexibility.

[1]unlike void* in C

Polymorphic functions

What is the type of this function?

```
let id x = x
```

Polymorphic functions

What is the type of this function?

```
let id x = x
val id : 'a -> 'a = <fun>
```

The type variable `'a` (pronounced α) can denote any particular parameter and result type.

Polymorphic functions

What is the type of this function?

```
let id x = x
val id : 'a -> 'a = <fun>
```

The type variable `'a` (pronounced α) can denote any particular parameter and result type.

```
id 10 ;;
- : int = 10
id "hello" ;;
- : string = "hello"
```

Type variables

- When we apply the `id` function to a specific type of parameter α is *instantiated* to that type.

Type variables

- ★ When we apply the `id` function to a specific type of parameter α is *instantiated* to that type.
- ★ For example in the expression `id 10`, we instantiate $\alpha =$`int`.

Type variables

- When we apply the `id` function to a specific type of parameter α is *instantiated* to that type.
- For example in the expression `id 10`, we instantiate $\alpha =$`int`.
- The type of `id` $\alpha \to \alpha$ tells us that the type of the parameter is equal to the type of the result.

Exercise

1. Ocaml's pervasives contains two functions `fst` and `snd` to extract the first and second elements of a tuple. What are its types?

Exercise

1. Ocaml's pervasives contains two functions `fst` and `snd` to extract the first and second elements of a tuple. What are its types?
2. Write a function to swap the first and second elements of a tuple. What is its type?

Lecture Outline

Polymorphic records

User defined types can also be made polymorphic by introducing a type variable.

```
type 'a item = { name:string; quantity:'a }
```

Polymorphic records

User defined types can also be made polymorphic by introducing a type variable.

```
type 'a item = { name:string; quantity:'a }

let it1 = { name="flour"; quantity=110.10 } ;;
val flour : float item
let it2 = { name="coconuts"; quantity=28 } ;;
val it2 : int item
```

Option type

The option type defined in Ocaml pervasives is useful to denote that there is no result for a function,

```
let 'a option =
| Some 'a
| None
```

Option type

The option type defined in Ocaml pervasives is useful to denote that there is no result for a function,

```
let 'a option =
| Some 'a
| None
```

For example, suppose we wanted to find the maximum of a list of numbers. What do we reurn if the list is empty?