Labelled Parameters

Ng Zhi An

Annotations for function parameters

Simple example

```
# let f x y = x + y ;;

val f : int -> int -> int = <fun>

# f 1 2 ;;

- : int = 3
```

Simple example

```
# let f ~x ~y = x + y ;;

val f : x:int -> y:int -> int = <fun>

# let x = 1 and y = 2 in f ~x ~y ;;

- : int = 3

# f ~x:1 ~y:2;;

- : int = 3
```

Different names for label and variable

```
# let f ~x:x1 ~y:y1 = x + y ;;

Error: Unbound value x

# let f ~x:x1 ~y:y1 = x1 + y1 ;;

val f : x:int -> y:int -> int = <fun>

# f ~x:1 ~y:2;;

- : int = 3
```

Ordering

```
# let f ~x ~y = x + y ;;

val f : x:int -> y:int -> int = <fun>

# f ~y:2 ~x:1;;

- : int = 3
```

Partial application

```
# let f ~x ~y = x + y ;;

val f : x:int -> y:int -> int = <fun>

# let g = f ~y:2;;

val g : x:int -> int = <fun>

# g ~x:1 ;;

- : int = 3
```

Total application

```
# let f ~x ~y = x + y ;;

val f : x:int -> y:int -> int = <fun>

# f 1 2

- : int = 3
```

Repeated labels

```
# let triple ~x:x1 ~x:x2 ~y = (x1, x2, y);;

val triple : x:'a -> x:'b -> y:'c -> 'a * 'b * 'c = <fun>

# triple ~x:3 ~x:4 ~y:5;;

- : int * int * int = (3, 4, 5)

# triple 3 4 5;;

- : int * int * int = (3, 4, 5)
```

Optional labels

```
# let bump ?(step = 1) x = x + step;;

val bump : ?step:int -> int -> int = <fun>

# bump 2;;

- : int = 3

# bump ~step:3 2;;

- : int = 5
```

Optional labels as option types

```
# let bump ?step x =
    match step with
    | None -> x * 2
    | Some y -> x + y
  ;;
val bump : ?step:int -> int -> int = <fun>
```

Passing through optional argument

```
val bump : ?step:int -> int -> int = <fun>

# let bumpy ?step = bump ?step

# bumpy 3;;

- : int = 4
```

Motivation

Clean, clear, self-documenting function interface (API)

Allow more checking

Flexibility in function application

```
UnixLabels.write :
  File_descr
  -> bytes
  -> int
  -> int
  -> unit
```

```
UnixLabels.write :
  File_descr
  -> buf:bytes
  -> pos:int
  -> len:int
  -> unit
```

Swift

```
func f1(x: Int, y: Int) -> Int { return x + y }

f1(x: 1, y: 2) // 3, f1(1, 2) is an error

func f2(_ x: Int, _ y: Int) -> Int { return x + y }

f2(1, 2) // 3

func f4(x x1: Int, y y1: Int) -> Int { return x1 + y1 }

func f5(x: Int = 1, y: Int) -> Int { return x + y }

f5(y: 2) // 3
```

References

http://caml.inria.fr/pub/docs/manual-ocaml/lablexamples.html

https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/Functions.html