Introduction to Functional Programming in *OCaml*

Roberto Di Cosmo, Yann Régis-Gianas, Ralf Treinen

Week 4 - Sequence 3: Partial Application of Functions









Partial Application

► A function

```
let f1 = fun \times y \rightarrow exp
is the same as
let f2 = function \times \rightarrow (function y \rightarrow exp)
```

► Since £2 can be applied to one argument, yielding a function, we can do the same with £1.

Partial Function Application I

```
let f1 = fun \times y \times z \rightarrow x + y \times z;;
# val f1 : int -> int -> int -> int = <fun>
let f2 = f1 1;;
# val f2 : int -> int -> int = <fun>
let f3 = f2 2;;
# val f3 : int -> int = <fun>
f3 4;;
# - : int = 9
```

What happens at function application

- ► Application of function x -> e to a:
 - ► evaluate e in a context x=a
- ▶ No evaluation ever happens below a function ... → ...

function blocks evaluation I

```
let f1 = fun x y -> (x / 0) + y;;
# val f1 : int -> int -> int = <fun>
let f2 = f1 17;;
# val f2 : int -> int = <fun>
f2 42;;
# Exception: Division_by_zero.
```

Partial Evaluation

- ► Sometimes we can do part of a calculation as soon as we have the first few arguments.
- ▶ Factor out that part of the calculation in order to profit from partial evaluation.

Partial Evaluation I

```
let egal 11 12 = List.sort compare 11 = List.sort compare 12;;
# val egal : 'a list -> 'a list -> bool = <fun>
let f = egal [783; 42; 17];;
# val f : int list -> bool = <fun>
f [17:42:783]::
# - : bool = true
let egalp 11 =
  let l1sorted = List.sort compare l1
  in function 12 -> 11sorted = List.sort compare 12;;
# val egalp : 'a list -> 'a list -> bool = <fun>
```

Partial Evaluation II

```
let f = egalp [783; 42; 17];;
# val f : int list -> bool = <fun>

f [17;42;783];;
# - : bool = true
```