Introduction to Functional Programming in *OCaml*

Roberto Di Cosmo, Yann Régis-Gianas, Ralf Treinen

Week 1 - Sequence 5: Recursion

Recursive Functions

- Functions that are defined by calling themselves on smaller arguments
- Natural on recursively defined data structures (see Week 3)
- Example: $fact(n) = \begin{cases} 1 & \text{if } n = 1 \\ n * fact(n-1) & \text{if } n > 1 \end{cases}$

Recursive Definitions in OCaml

- A priori, the use of f in a definition of f refers to the *previous* value of f
- The keyword `rec` changes this, and allows us to define a function by recursion

Recursive Definitions in OCaml I

```
let x = 1;;
# val x : int = 1
let x = x+1;;
# val x : int = 2
x;;
# - : int = 2

let f x = x+1;;
# val f : int -> int = <fun>
let f x = f (f x);;
# val f : int -> int = <fun>
f 1;;
# - : int = 3
```

Recursive Definitions in OCaml II

```
let fact n = if n <=1 then 1 else n*fact(n-1);;
# Characters 37-41:
  let fact n = if n <=1 then 1 else n*fact(n-1);;
                                    ^^^^
Error: Unbound value fact

let rec fact n = if n <=1 then 1 else n*fact(n-1);;
# val fact : int -> int = <fun>

fact 10;;
# - : int = 3628800
```

Mutually Recursive Functions

- Generalization of direct recursion
- Several functions are defined by calling each other on smaller arguments
- Natural on mutual recursive data structures
- Example:
  - n is even if $n = 0$, or $n > 0$ and $n - 1$ is odd
  - n is odd if $n = 1$, or $n > 1$ and $n - 1$ is even

Mutually Recursive Definitions in OCaml I

```
let rec even x = if x=0 then true else odd (x-1);;
# Characters 39-42:
  let rec even x = if x=0 then true else odd (x-1);;
                                          ^^^
Error: Unbound value odd

let rec even x = if x=0 then true else odd (x-1)
and odd x = if x=0 then false else even (x-1);;
# val even : int -> bool = <fun>
val odd : int -> bool = <fun>

even 17;;
# - : bool = false
even 10;;
# - : bool = true
```