Introduction to Functional Programming in *OCaml*

Roberto Di Cosmo, Yann Régis-Gianas, Ralf Treinen

Week 6 - Sequence 2: Case study: A module for dictionaries

Interface for modularity

- In this case study, we will see that **information hiding improves modularity**.

A module for dictionaries I

```
module type DictSig = sig
  type ('key, 'value) t
  val empty : ('key, 'value) t
  val add : ('key, 'value) t -> 'key -> 'value -> ('key, 'value) t
  exception NotFound
  val lookup : ('key, 'value) t -> 'key -> 'value
end;;
# module type DictSig =
  sig
    type ('key, 'value) t
    val empty : ('key, 'value) t
    val add :
      ('key, 'value) t ->
      'key -> 'value -> ('key, 'value) t
    exception NotFound
```

A module for dictionaries II

```
  val lookup : ('key, 'value) t -> 'key -> 'value
end
```

A module for dictionaries III

```
module Dict : DictSig = struct
  type ('key, 'value) t = ('key * 'value) list
  let empty = []
  let add d k v = (k, v) :: d
  exception NotFound
  let rec lookup d k =
    match d with
      | (k', v) :: d' when k = k' -> v
      | _ :: d -> lookup d k
      | [] -> raise NotFound
end;;
# module Dict : DictSig
```

A module for dictionaries IV

```
(* The client *)
module ForceArchive = struct
  let force = Dict.empty
  let force = Dict.add force "luke" 10
  let force = Dict.add force "yoda" 100
  let force = Dict.add force "darth" 1000
  let force_of_luke = Dict.lookup force "luke"
  let force_of_r2d2 = Dict.lookup force "r2d2"
end;;
# Exception: Dict.NotFound.
```

A module for dictionaries V

```
module Dict : DictSig = struct
  type ('key, 'value) t =
    | Empty
    | Node of ('key, 'value) t * 'key * 'value * ('key, 'value) t

  let empty = Empty

  let rec add d k v =
    match d with
    | Empty -> Node (Empty, k, v, Empty)
    | Node (l, k', v', r) ->
      if k = k' then Node (l, k, v, r)
      else if k < k' then Node (add l k v, k', v', r)
      else Node (l, k', v', add r k v)

  exception NotFound
```

A module for dictionaries VI

```
  let rec lookup d k =
    match d with
      | Empty ->
        raise NotFound
      | Node (l, k', v', r) ->
        if k = k' then v'
        else if k < k' then lookup l k
        else lookup r k

end;;
# module Dict : DictSig
```

A module for dictionaries VII

```
(* The same client *)
module ForceArchive = struct
  let force = Dict.empty
  let force = Dict.add force "luke" 10
  let force = Dict.add force "yoda" 100
  let force = Dict.add force "darth" 1000
  let force_of_luke = Dict.lookup force "luke"
  let force_of_r2d2 = Dict.lookup force "r2d2"
end;;
# Exception: Dict.NotFound.
```

Weaknesses of this architecture

- A more informative exception would be `exception NotFound of 'key`.
- Yet, exceptions cannot be polymorphic in *OCaml*...
- Here we are forced to use the default polymorphic comparison on keys.
- Sometimes other comparisons are needed.
- In the client, the reference to the module `Dict` is hardcoded.
- Delaying this choice would make the client more reusable.

Forthcoming **functors** will solve these issues.