Extending OCaml with copatterns and first-class labels for Free

YANN RÉGIS-GIANAS, PAUL LAFORGUE, Diderot University

ACM Reference format:
Yann Régis-Gianas, Paul Laforgue. 2017. Extending OCaml with copatterns and first-class labels for Free. *ACM J. Comput. Cult. Herit.* 9, 4, Article 00 (March 2017), 3 pages.
DOI: 0000001.0000001

1 SYNTAX

Here we describe the syntax of our extension. This one is conflictless with the actual syntax used in OCaml 4.04. In the following section :

- Programs with a yellow background stand for the original code, while blue ones represent generated code.
- Data constructors indiced with a small cross (such as Stream$^{\times}$) are inaccessible for the programmer.
- Infix symbol (▷) corresponds to the reverse-application operator such that (x ▷ f) is equivalent to f (x).
- Type constructors codata and query are abstracts and automatically imported from module `Pervasives`.

1.1 Codata types

Codata types are introduced with the **type** keyword.

```
type α !stream = {
  Head : α;
  Tail  : α !stream;
}
```

```
type (σ, α) stream =
   | Stream^× : {dispatch : σ.(σ query, α) stream → σ} → (codata, α) stream
   | Head     : (α query, α) stream
   | Tail     : (((codata, α) stream) query, α) stream
let head = function Stream^× {dispatch} → dispatch Head
let tail  = function Stream^× {dispatch} → dispatch Tail
```

```
type (α, β) !product = {
  Fst  : α;
  Snd : β;
}
```

DOI: 0000001.0000001

```
type (σ, α, β) product =
  | Product^× : { dispatch : σ.(σ query, α, β) product → σ } → (codata, α, β) product
  | Fst       : (α query, α, β) product
  | Snd       : (β query, α, β) product
let fst  = function Product^× { dispatch } → dispatch Fst
let snd = function Product^× { dispatch } → dispatch Snd
```

1.2 Copattern matching

```
let zeros = comatch zs : int !stream with
  | zs#Head → 0
  | zs#Tail   → zs
```

```
let zeros =
  let rec zs : (codata, int) stream =
    let dispatch : type σ.(σ query, int) stream → σ = function
      | Head → 0
      | Tail   → zs
    in Stream { dispatch }
  in zs
```

1.3 Unnesting copatterns

Here, we process more or less as in [?].

```
let fibonacci = comatch fib : int !stream with
  | fib#Head → 0
  | fib#Tail#Head → 1
  | fib#Tail#Tail   → zipWith (+) fib fib#Tail
```

```
let fibonacci =
  let rec fib : (codata, int) stream =
    let f1 =
      let dispatch : type σ.(σ query, int) stream → σ = function
        | Head → 1
        | Tail → zipWith (+) fib (fib ▷ tail)
      in Stream^× { dispatch }
    in
    let dispatch : type σ.(σ query, int) stream → σ = function
      | Head → 0
      | Tail → f1
    in Stream^× { dispatch }
  in fib
```

1.4 BNF

REFERENCES

[] Anton Setzer, Andreas Abel, Brigitte Pientka, and David Thibodeau. 2014. Unnesting of copatterns. In *International Conference on Rewriting Techniques and Applications*. Springer, 31–45.

Received February 2007; revised March 2009; accepted June 2009