Extensions and Combinators in Objective CAML

Dmitri Boulytchev

St.Petersburg State University, Russia

LDTA'11 Tool Challenge, March 26-27, 2009, Saarbrücken

Tool Challenge Solution Overview

- T1-T3 $\times$ L1-L5;
- Parser combinators + syntax extension (`http://oops.math.spbu.ru/projects/ostap`);
- Pretty-printer combinators;
- Extensible types (polymorphic variants).
- Generic monadic transformers;

Files	22
Lines of Code	2838
In Toplevels	973
In Language Components	1074
Commits	14
Man-days	9

Ostap (parser combinator library)

- Direct style monadic parser combinators;
- Syntax extension for OCaml;
- Higher-order parsers, streams with structural subtyping.

```
ostap (
 sample[a][b][c]: a "," b? DELIMITER c*
)

val sample :
  (< getDELIMITER : ('a, 'b, < add : 'c -> 'c; .. > as 'c)
                    Combinators.result;
     look : string -> ('a, 'd, 'c) Combinators.result;
     .. >
     as 'a) ->
  ('a, 'e, 'c) Combinators.parse ->
  ('a, 'f, 'c) Combinators.parse ->
  ('a, 'g, 'c) Combinators.parse ->
  ('a, 'e * 'd * 'f option * 'b * 'g list, 'c)
     Combinators.result
```

Polymorphic Variants

```
val cattle : [< `Cow | `Sheep ] -> string
let cattle = function `Cow   -> "a cow"
                    | `Sheep -> "a sheep"

let _ =
  List.iter
    (Printf.printf "%s\n")
    (List.map cattle [`Cow; `Sheep; `Sheep; `Cow])
```

Polymorphic Variants (continue)

```
val cattle : [< `Cow | `Sheep ] -> string
let cattle = function `Cow   -> "a cow"
                    | `Sheep -> "a sheep"

let _ =
  List.iter
    (Printf.printf "%s\n")
    (List.map cattle [`Cow; `Sheep; `Sheep; `Cow;
                      `Cat; `Dog])
```

Polymorphic Variants (continue)

```
val cattle : [< `Cow | `Sheep ] -> string
let cattle = function `Cow   -> "a cow"
                    | `Sheep -> "a sheep"
val friend : [< `Cat | `Dog ] -> string
let friend = function `Cat -> "a cat"
                    | `Dog -> "a dog"

let _ =
  List.iter
    (Printf.printf "%s\n")
    (List.map (function
               | (`Cow | `Sheep) as x -> cattle x
               | (`Cat | `Dog  ) as x -> friend x)
       [`Cow; `Sheep; `Sheep; `Cow; `Cat; `Dog]
    )
```

Polymorphic Variants (continue)

```
val cattle : ([> `Cow | `Sheep] as 'a -> string) ->
             'a -> string
let cattle other = function `Cow   -> "a cow"
                          | `Sheep -> "a sheep"
                          | x      -> other x
val friend : ([> `Cat | `Dog ] as 'a) -> string ->
             'a -> string
let friend other = function `Cat -> "a cat"
                          | `Dog -> "a dog"
                          | x    -> other x

let rec all x = cattle (friend all) x

let _ =
  List.iter
    (Printf.printf "%s\n")
    (List.map all [`Cow; `Sheep; `Sheep; `Cow;
                   `Cat; `Dog])
```

Polymorphic Variants (continue)

```
val cattle : ([> `Cow | `Sheep] as 'a -> string) ->
             'a -> string
let cattle other = function `Cow   -> "a cow"
                          | `Sheep -> "a sheep"
                          | x      -> other x
val friend : ([> `Cat | `Dog ] as 'a) -> string ->
             'a -> string
let friend other = function `Cat -> "a cat"
                          | `Dog -> "a dog"
                          | x    -> other x

let rec fix f x = f (fix f) x
let all y = fix (fun x -> cattle (friend x)) y

let _ =
  List.iter
    (Printf.printf "%s\n")
    (List.map all [`Cow; `Sheep; `Sheep; `Cow;
                   `Cat; `Dog])
```

Drawbacks

- Often types become “too open”:

```
val all : [> `Cat | `Cow | `Dog | `Sheep ] -> string
let all y = fix (fun x -> cattle (friend x)) y

let _ =
  List.iter
    (Printf.printf "%s\n")
    (List.map all [`Cow; `Sheep; `Sheep; `Cow;
                   `Cat; `Dog; `Pig])
```

- Type-error notification can be verbose.

Thank you!