Présentation projet Fouine

Robin Jourde & Nicolas Nardino

mai 2021

Dites « Bonjour » à notre amie la fouine !

Table de matières

Un terme polysémique

Un terme polysémique

- un petit animal trop mignon :

Un terme polysémique

- un petit animal trop mignon :

- un sous langage de OCaml

Un terme polysémique

- un petit animal trop mignon :

- un sous langage de OCaml
- un interpréteur de ce sous langage

Un terme polysémique

- un petit animal trop mignon :

- un sous langage de OCaml
- un interpréteur de ce sous langage

Par la suite on n'utilisera le terme que dans ses 2 derniers sens.

Le langage

Grammaire simplifiée :

$$e := let\ (rec)\ p = e_1\ in\ e_2 \mid fun\ p \rightarrow e_1 \mid e_1\ e_2 \mid e_1 := e_2$$
$$\mid\ !e_1 \mid if\ e_1\ then\ e_2\ else\ e_3 \mid match\ e_1\ with\ [\mid p_i -> e_i]$$
$$\mid e_1 :: e_2 \mid try\ e_1\ with\ E\ p -> e_2 \mid raise\ e_1 \mid p$$

p représente un pattern :

$$p := x \mid \_ \mid k \mid p_1,\ p_2 \mid p_1 :: p_2$$

k est une constante

Le langage

Grammaire simplifiée :

$$e := let\ (rec)\ p = e_1\ in\ e_2 \mid fun\ p \rightarrow e_1 \mid e_1\ e_2 \mid e_1 := e_2$$
$$\mid\ !e_1 \mid if\ e_1\ then\ e_2\ else\ e_3 \mid match\ e_1\ with\ [\mid p_i -> e_i]$$
$$\mid e_1 :: e_2 \mid try\ e_1\ with\ E\ p -> e_2 \mid raise\ e_1 \mid p$$

p représente un pattern :

$$p := x \mid \_ \mid k \mid p_1,\ p_2 \mid p_1 :: p_2$$

k est une constante
Et les opérations arithmétiques me diriez vous ? ...

Le langage

Grammaire simplifiée :

$$\begin{aligned} e \ := \ & \mathit{let}\ (\mathit{rec})\ p\ =\ e_1\ \mathit{in}\ e_2 \mid \mathit{fun}\ p\ \rightarrow\ e_1 \mid e_1\ e_2 \mid e_1\ :=\ e_2 \\ & \mid\ !e_1 \mid \mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ e_3 \mid \mathit{match}\ e_1\ \mathit{with}\ [\mid p_i\ ->\ e_i] \\ & \mid e_1\ ::\ e_2 \mid \mathit{try}\ e_1\ \mathit{with}\ E\ p\ ->\ e_2 \mid \mathit{raise}\ e_1 \mid p \end{aligned}$$

p représente un pattern :

$$p\ :=\ x \mid \_ \mid k \mid p_1,\ p_2 \mid p_1\ ::\ p_2$$

k est une constante
Et les opérations arithmétiques me diriez vous ? ...
... Et là vous croyiez que j'allais le dire maintenant mais en fait non, il a falloir attendre ! Suspens ... !!

L'interprète

- `./fouine [options] test1515.ml`

L'interprète

- `./fouine [options] test1515.ml`
- Mais quelles options ??? Demandons à fouine : `./fouine --help`

L'interprète

- `./fouine [options] test1515.ml`
- Mais quelles options ??? Demandons à fouine : `./fouine --help`
- `-showsrc -debug -tree -outval -stdin -cps -outcode -outcode-tree -run -autotest -optim -reduc -prefix -notypes -showtypes -showmoretypes -monotypes -cpstypes`

Un exemple

- fichier de test `not.ml`

```
let x = not (0>1) in if x then prInt 1 else prInt 0
```

- `./fouine -debug -outval -showtypes not.ml`

```
x : bool
- : int
let x = not (( > ) (0) (1)) in (if x then prInt (1)
else prInt (0))
1
1
```

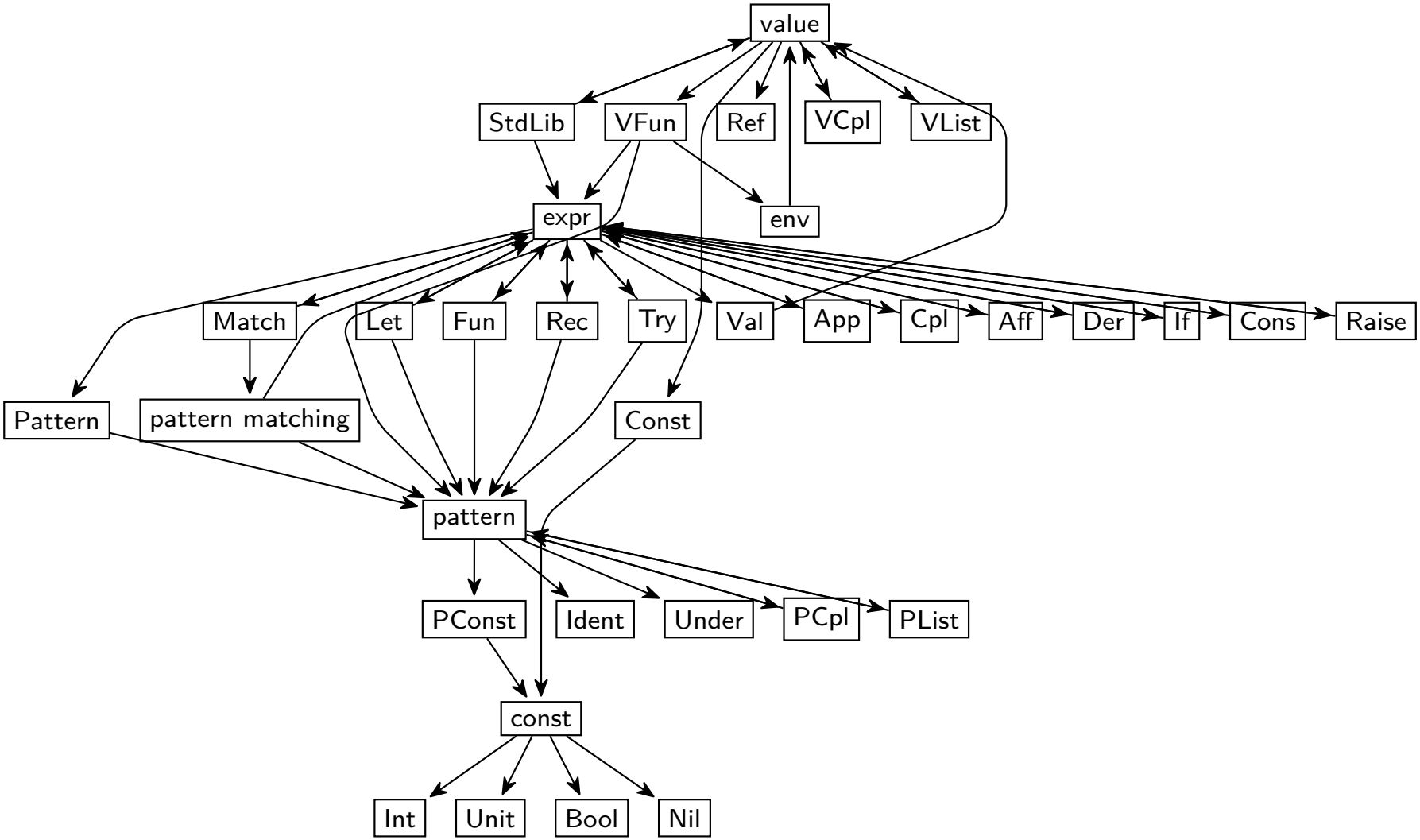
Que fait fouine ?

Étapes :

1. Lire : lexer et parser
2. Typer
3. (Si souhaité) traduction CPS
4. (Si souhaité) réduction
5. Évaluer

Pour tout cela on a besoin de représenter les expressions...

Des constructeurs et des types



La bibliothèque standard

On tient notre promesse !

- `fst , snd`
- `prInt`
- `ref`
- `( + ) , ( - ) , ( * ) , ( / )`
- `( && ) , ( || )`
- `not`
- `( <= ) , ( >= ) , ( < ) , ( > )`
- `( = ) , ( <> )`

Exemple détaillé : prInt

- code de la fonction

```
let _prInt m = function
 | Const (Int i) -> (print_int i ; print_newline () ; &
(Int i) )
 | _ -> raise PrInt_not_int
```

- insertion dans l'environnement

```
("prInt", StdLib _prInt) :: ...
```

- type

```
("prInt", TFun(TInt, TInt) ):: ...
```

C'est fini !

Merci à tous & Bisous !