

# Présentation projet Fouine

Robin Jourde & Nicolas Nardino

mai 2021

# Dites « Bonjour » à notre amie la fouine !

```
0000 000 0 0 000 0 0 0000
0 0 0 0 0 0 00 0 0
000 0 0 0 0 0 0 0 000
0 0 0 0 0 0 0 00 0
0 000 000 000 0 0 0000 (JN)
```

```
'iAg##N6/. -\e%##8Xc,
,8000000000A`~X0000000000#;
;000000000000#000000000000+
B000000000000000000000000g
Q000000000000000000000000Q
3000000000000000000000000e
`g00000000000000000000000N`
u000000000000000000000000j
-j000000000000000000000j-
|D0000000000000000d=
.7Q00000000Bi-
:m00009:
-6e.
```

```
`=xvCv!':`czzc|;,
.OvsFfU6Qe!,,``~;LE8a6deyk,
=yDfz\,r|: :'?yRCoDR
-OQ' `raQk
wj 0|
Q= 'e$E: o00z ~0!
?g' |8p8B, `CQaQ| ;#'
,0c` ``:w! `~{: '=g,
@;pu^` zge8B- 'tu@;`
Q\l!:C9c- ~?+.~`e/~ Q
dai= ,vp9yo6i~ p"
.Q:;e :z :m lz
`gs `#' %-`D :g
`#z 'm :E B ,0
!B! .a` z|%" vf
iQ, "0` :#A; .Q,
|Q, j7 -@d! `R=
'0: fi -00~ `Ov
`#7 9+ ~@{ ue
`Uk %: dm ,Q`
~W\ ' #~ Cd kv
vd' e `Q. \Q! \@,
-de ~a` L% |:g `9lK/
`#$ \l c9:7,D w| `O\
OD` :d =Q8`!6 '0` `d|
lQ` Q, .@C =j '0d' `B-
,@! |K fj f\ '0+g; mF
pm `Q! cEB' rN ze-!|B
,@' |ka+~,;{CU `B/ +Kya/6
t6 B,d>Lw!QDa xd !dp|`
ma '!=LFQ'!L0du#~w Q:
```

# Table de matières

- (1) Fouine, qu'est-ce que c'est ?
- (2) Fouine, comment ça marche ?
- (3) Conclusion

(1) Fouine, qu'est-ce que c'est ?

(2) Fouine, comment ça marche ?

(3) Conclusion

# 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 :

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

$p$  représente un pattern :

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

$k$  est une constante

# Le langage

Grammaire simplifiée :

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

$p$  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 ? ...

# Le langage

Grammaire simplifiée :

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

$p$  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
```



(1) Fouine, qu'est-ce que c'est ?

(2) Fouine, comment ça marche ?

(3) Conclusion

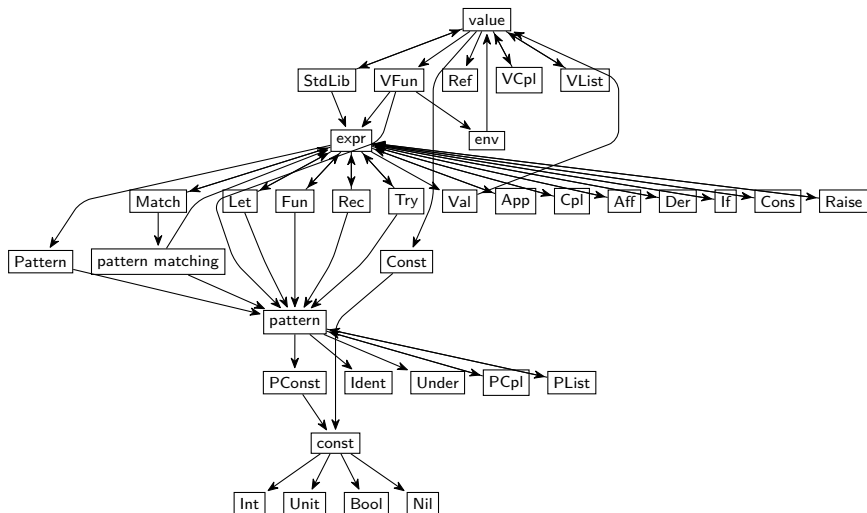
# Que fait fouine ?

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 !

## Conclusion

[illegible]