Un langage de programmation impératif.

**1. Types de bases et expressions types.**

Il arrive à déterminer le type de base par lui-même. Les différentes bases sont : float, int, string, list, char, bool.

1) Bool.

Il peut prendre deux valeurs distinctes, *true* (vrai) et *false* (faux).

On note également trois opérateurs qui sont : **et** (&&), **||** (ou) et **not** (non).

Forme fonction :

- **not** : bool -> bool

- **&&** : bool -> bool -> bool

- **||** : bool -> bool -> bool

(e1 || e2) && (not (e1 && e2))

En maths, les fonctions basiques sont de la forme : x |-> f(x).

Expression fonctionnelle :

Fun (e1 e2) -> (e1 || e2) && (not (e1 && e2)), de type : bool -> bool -> bool.

Donc pour on obtient pour les valeurs a et b suivantes :

Fun (a b) -> (a || b) && (not (a && b)), de type : bool –> bool -> bool (a = true, b = false)

-> (true || false) && (not (true && false))

-> True && (not false)

-> True

Let xor = Fonct. (a b) -> (a || b) && (not (a && b))

Xor (true false) -> true

Let xor (a b) = (a || b) && (not (a && b))

Let xor (a : bool) (b : bool) : bool = (a || b) && (not (a && b))

Fun (x -> x) \* true -> true

Fun (x -> x) \* 2 -> 2

Fun (x -> x) : ‘a -> ‘a

Syntaxe :

If e1 then e2 else e3

(If true then e1 else e2) -> e1

(If false then e1 else e2) -> e2

Let if-mood (a, b, c) = (if a then b else c)

Bool -> ‘a -> ‘a -> ‘a

Let xor (a : bool) (b : bool) : bool =>

If a then

If b then

False

Else

True

Else

If b then

True

Else

False

Let xor (a bool) (b bool) bool =>

If a then

Not b

Else

B

Xor (true false) -> If true then

Not (false)

Else

False

-> Not false

-> True

Tuple :

(True, false) : bool \* bool | (True, 1) : bool \* int

Let paire (a : ‘a) (b : ‘b) : ‘a \* ‘b = (a, b)

‘a -> ‘b -> (‘a \* ‘b)

**2. Filtrage :**

Syntaxe : match a with

|v1 -> e1

|v2 -> e2

|… ->

Let xor (a : bool) (b : bool) : bool =>

Match (a, b) with

| True, true -> false

| True, false -> true

| False, true -> true

| False, false -> false

List xor (a : bool) (b : bool) : bool =>

Match (a, b) with

| True, false -> true

| False, true -> true

Let xor (a : bool) (b : bool) : bool =>

Match (a, b) with

| True, false -> true

| True, true -> false

| False, r -> r

Let id x =>

Match x with

| Y -> Y

Let x = True in, x : bool

Let x = true in

X || (not x) -> True

Let x : True in

Not x -> False

Let x = True in

Let y = False in (x = True)

X || y

E1

-> True

Let fst (z : bool \* bool) : bool :

Match z with

| (a, b) -> a

Let fst (z : bool \* bool) : bool :

Let (a, b) = z in

A

Fst (true, false)

-> let (a, b) = (true, false) in

a

-> true