Motifs inutilisés : les cas suivants un motif qui filtre n'importe quelle valeur n'ont aucune utilité.

```
# let f x = match x with
      0 -> 18
    | y -> y * y
    | 1 -> 24 ;;
Warning 11: this match case is unused.
val f : int -> int = <fun>
```

4.2.4 Filtrage et fonctions anonymes

Fonctions anonymes à un paramètre

let *f x* = *expression* ≡ let *f* = function *x* -> *expression*

```
# let succ = function x -> x + 1 ;;
val succ : int -> int = <fun>
```

let *ident_function* = function *pattern*$_1$ -> *expression*$_1$ | *pattern*$_i$ -> *expression*$_i$ | ... | *pattern*$_n$ -> *expression*$_n$;;

```
# let sign = function
      0             -> 0
    | y when y > 0  -> 1
    | _             -> -1 ;;
val sign : int -> int = <fun>
```

Fonctions anonymes à plusieurs paramètres

let *f x y* = *expression* ≡ let *f* = function *x* -> function *y* -> *expression*

```
# let average = function a -> function b -> (a + b) / 2 ;;
val average : int -> int -> int = <fun>
```

4.2.5 Produit cartésien et Filtrage de plusieurs valeurs

Produit cartésien

```
# let one = (1, "one") and two = (2., "two") ;;
val one : int * string = (1, "one")
val two : float * string = (2, "two")
# let pair = (one, two) ;;
val pair : (int * string) * (float * string) = ((1, "one"), (2, "two"))
```

Filtrage de plusieurs valeurs

```
# let implication = function
      (true, x) -> x
    | _         -> true ;;
val implication : bool * bool -> bool = <fun>

# let implication a b = match (a, b) with
     (true, x) -> x
    | _        -> true;;
val implication : bool -> bool -> bool = <fun>
```

Peut s'écrire : # 1 2
(1, 2)