


4 Analyses par cas

4.1 L'alternative

```
if condition then expression1 else expression2
```

- *condition* est une expression booléenne;
- *expression₁* et *expression₂* sont du même type; 
- la valeur de l'alternative est *expression₁* si *condition* a la valeur *true* et *expression₂* si *condition* a la valeur *false*.

L'alternative est une expression.

```
# if 2 > 1 then
  "higher"
else
  "lower" ;;
- : string = "higher"
```



```
# let absolute_value x =
  if x >= 0 then
    x
  else
    -x ;;
val absolute_value : int -> int = <fun>
```

4.2 Le filtrage

4.2.1 Filtrage "explicite"

```
match ident with
| pattern1 -> expression1
| patterni -> expressioni
| ...
| patternn -> expressionn
```

```
# let f x = match x with
  0 -> 0.
  | 1 -> 1.
  | x -> let x = float_of_int x in x *. x ;;
val f : int -> float = <fun>
```

-  — *expression₁ ... expression_n* doivent toutes être du même type : ce sera le type du résultat;
-  — les motifs *pattern₁, pattern₂, ...* doivent "filtrer" le même type celui de *ident*;
- le résultat de cette **expression** sera *expression_i* si la *forme* de la *valeur ident* correspond au motif *pattern_i*;
- le filtrage doit être *exhaustif* : les motifs doivent recouvrir tous les cas (tout le type de *ident*);
- les différents motifs sont examinés dans l'ordre dans lequel ils sont donnés.

4.2.2 Les motifs

Peuvent être utilisés comme *motifs* (pattern) :

- des valeurs "simples" : 1, true, 'c'
- des identifiants : x
- le motif *universel* : _
- des "unions" de cas, les filtres ou : *pattern₁ | pattern₂ | ...*
- des filtres *gardés* : *ident when condition*, où *condition* est une expression booléenne.
- des motifs structurés (à l'aide de constructeurs : n-uplets, listes...)

Le "règle" se traduit
de 3 manières :
"y -> ..."
"x -> ..."

4.2.3 Warning

Filtrage "exhaustif" : tous les cas doivent être pris en compte.

```
# let even x = match x with
  0 | 2 | 4 | 6 | 8 -> true
  | 1 | 3 | 5 | 7 | 9 -> false ;;
```

Warning 8: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched: 10
val even : int -> bool = <fun>

```
# let even x = match x mod 10 with
  0 | 2 | 4 | 6 | 8 -> true
  | _ -> false;;
val even : int -> bool = <fun>
```