Le filtre à motifs

Étendons encore les expressions de mini-ML par les *filtres* et les *motifs* :

- **match** e **with** $p_1 \to e_1 \mid \ldots \mid p_n \to e_n$
- où les p_i sont des motifs.

Les motifs sont définis récursivement par les cas suivants :

• variable	f, g, h (fonctions) et x, y, z (autres).
• unité ou constante	$()$ ou 0 ou **true** etc.
• n-uplet	$p_1, \ldots, p_n$
• parenthèse	(p)
• joker	$\_$

Remarque Les motifs irréfutables sont des motifs particuliers.

Le filtre à motifs (suite)

Typiquement, on se sert des filtres pour faire des définitions par cas de fonctions (mais pas uniquement), comme en mathématiques. Par exemple :

```
let rec fib n =
  match n with
    0 -> 1
  | 1 -> 1
  | _ -> fib(n-1) + fib(n-2)
```

Comme en mathématiques, les cas sont ordonnés par l'écriture et la définition précédente se lit donc : « Si la valeur de `n` *a la forme* de `0` alors `fib(n)` est la valeur de `1`, si la valeur de `n` *a la forme* de `1` alors `fib(n)` est la valeur de `1`, sinon `fib(n)` vaut la valeur de `fib(n-1) + fib(n-2)` ».

Le filtre à motifs (suite)

Que signifie la relation « v a la forme de p » (ou « p filtre v ») ?

- Une constante a la forme de la constante identique dans un motif.
- L'unité () a la forme de () dans un motif.
- Un n-uplet a la forme d'un n-uplet dans un motif.
- **Toute valeur a la forme d'une variable de motif ou du joker « _ ».**

Remarques

- Les motifs ne filtrent aucune fermeture, c.-à-d. que e dans **match** e **with** ne doit pas s'évaluer en une fermeture.
- Dans le cas des constantes et (), la relation se confond avec l'égalité.

Sémantique du filtrage

Le *filtrage* est l'évaluation d'un filtre. Informellement, l'évaluation de

$$\textbf{match } e \textbf{ with } p_1 \to e_1 \mid \ldots \mid p_n \to e_n$$

commence par celle de e en v.

Ensuite v est confrontée aux différents motifs p_i dans l'ordre d'écriture. Si p_i est le premier motif à filtrer v, alors la valeur du filtre est la valeur de e_i.

Par exemple, voici la définition curryfiée de la disjonction logique :

```
let or = fun (x,y) ->
  match (x,y) with
   (false, false) -> false
  | _ -> true
```

Les types sommes et le filtrage des leurs valeurs

Les définitions de types sommes

Les types sommes s'apparentent aux énumérations et aux union du langage C. Par exemple, les valeurs booléennes peuvent être définies ainsi :

```
type booléen = Vrai | Faux
let v = Vrai and f = Faux
```

Les constructeurs ont pour première lettre une majuscule.

Le filtrage permet d'examiner les valeurs d'un type somme :

```
let int_of_booléen = fun b ->
  match b with
     Vrai -> 1
   | Faux -> 0
```

Les types sommes et le filtrage de leurs valeurs (suite)

Les constructeurs peuvent aussi transporter de l'information, dont l'interprétation complète alors celle du constructeur lui-même. Par exemple, un jeu de cartes peut être défini par

```
type carte = Carte of ordinaire | Joker
and ordinaire = couleur * figure
and couleur = Coeur | Carreau | Pique | Trefle
and figure = As | Roi | Reine | Valet | Simple of int
```

Le jeu de cartes

Définissons des cartes et des fonction construisant des cartes :

```
# let valet_de_pique = Carte (Pique,Valet);;
val valet_de_pique : carte = Carte (Pique,Valet)
# let carte f c = Carte (c,f);;
val carte : figure → couleur → carte = ⟨fun⟩
# let roi = carte Roi;;
val roi : couleur → carte = ⟨fun⟩
```

Filtrage des cartes

```
let valeur c = match c with
  Carte (As)        -> 14
| Carte (Roi)       -> 13
| Carte (Reine)     -> 12
| Carte (Valet)     -> 11
| Carte (Simple k) -> k
| Joker             -> 0
```

Un motif peut capturer plusieurs cas :

```
let est_petite c = match c with
  Carte (Simple _) -> true
| _ -> false
```

Filtres incomplets

Rappel Lorsqu'une valeur v est filtrée par $p_1 \to e_1 \mid \ldots \mid p_n \to e_n$, l'expression e_i associée au premier motif p_i qui filtre v est évaluée dans l'environnement courant étendu par les liaisons éventuelles créées par le filtrage de v par p_i.

Le filtre est *incomplet* s'il existe au moins une valeur qui n'est filtrée par aucun motif. Dans ce cas, un message d'avertissement est indiqué à la compilation :

```
# let simple c = match c with Carte (_,Simple k) -> k;;
Characters 15-51
Warning: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched: Joker
val simple : carte → int = ⟨fun⟩
```

Il est préférable d'éviter les définitions incomplètes.

Les motifs non linéaires

Une variable ne peut être liée deux fois dans le même motif.

```
# fun (x,y) -> match (x,y) with (Carte z, z) -> true;;
Characters 41-42
This variable is bound several times in this matching.
```

Un tel motif est dit *non linéaire*.