Enregistrements

IAP1

Catherine Dubois

Un type enregistrement = un type produit plus souple que le type des n-uplets

Rappel : un type produit permet de décrire un objet en énumérant chacune de ses caractéristiques

Exemple : une date se décrit en donnant le jour, le mois et l'année.

Les *n*-uplets manquent de souplesse :

- ordre strict imposé aux différentes composantes d'un *n*-uplet :
  la date représentée par le triplet `(25, 12, 2000)` n'est pas équivalente à `(2000, 12, 25)`.
  Si non respect, au mieux le programme échoue, au pire il fournit une réponse erronée.
  Pourtant l'ordre est souvent artificiel !
- nécessité de faire figurer toutes les composantes d'un *n*-uplet dans un filtre

```
#let proj1 (x,y,z) = x;;
val proj1 : 'a * 'b * 'c -> 'a = <fun>

#let proj1 (x,y) = x;;
 val proj1 : 'a * 'b -> 'a = <fun>
```

La solution naturelle à ce problème :
⇒ nommer (*étiqueter*) chacune des composantes (*champ*)

On repère alors une composante par son étiquette et non plus par sa position dans le n-uplet
⇒ plus d'ordre strict sur les composantes

Le n-uplet ainsi étiqueté est appelé *enregistrement*.

Le types des enregistrements est appelé type enregistrement

Un type enregistrement est déclaré (comme un type somme)

Déclaration d'un type enregistrement

```
type nom_du_type =
  { étiq_1 : t_1 ;
    étiq_2 : t_2 ;
        ⋮
    étiq_n : t_n
  };;
```

où $n \geq 1$, `t1`, `t2`,..., `tn` sont des types quelconques.

- L'ordre dans lequel les étiquettes sont écrites n'a pas d'importance.
- Les étiquettes introduites au sein d'un type enregistrement doivent être distinctes deux à deux.
- 2 types enregistrements distincts : étiquettes différentes.

```
#type date = {jour : int; mois : int; année : int};;

#type étudiant =
  { nom : string;
    prénom : string;
    date_de_naissance : date;
    no_de_sécu : string;
    groupeTD : int };;
```

Création et manipulation d'enregistrements

Créer un objet d'un type enregistrement suit une syntaxe calquée sur la déclaration du type : à chaque étiquette, une valeur

```
#let noël_75 = {jour=25; mois=12; année=1975};;
val noël_75 : date = {jour = 25; mois = 12; année = 19
```

On peut permuter les étiquettes :

```
# let noël_75_bis = {année=1975; mois=12; jour=25};;
val noël_75_bis : date = {jour = 25; mois = 12; année
# noël_75 = noël_75_bis;;
- : bool = true
```

Plus généralement, expression enregistrement :

Syntaxe

Un enregistrement est de la forme :

`{ étiq_1 = expr_1 ; étiq_2 = expr_2 ;...; étiq_n = expr_n }`

où $n \geq 1$, `expr_1`, `expr_2`, ..., `exp_n` sont des expressions quelconques

Typage

Le type de l'enregistrement est `t` si `t` est le dernier type enregistrement déclaré possédant les étiquettes `étiq_1`,..., `étiq_n`

`type t = { étiq_1 : t_1 ; étiq_2 : t_2 ; ...; étiq_n : t_n }`

et si pour tout *i*, `expr_i` a le type `t_i`

Évaluation

La valeur de l'enregistrement est

`{ étiq_1 = v_1 ;...; étiq_n = v_n }`

si, pour tout *i*, `v_i` est la valeur de `expr_i`

Exemple : le type des rationnels et la fonction de création d'un rationnel

Convention de représentation : le dénominateur est toujours un nombre positif

```
# type rationnel = { num: int; dén: int};;
type rationnel = { num : int; dén : int; }
# let créer_ratio (p,q) =
  if q = 0
  then failwith "créer_ratio"
  else if q > 0
          then {num = p; dén = q}
          else {num = -p; dén = -q};;
val créer_ratio : int * int -> rationnel = <fun>
# créer_ratio ((-2),(-3));;
- : rationnel = {num = 2; dén = 3}
```

Accès aux composantes d'un enregistrement

- en utilisant *la notation pointée* (idem en C)
- ou en *filtrant* l'enregistrement

1 Notation pointée

Pour désigner le mois d'une date `d` (de type `date`) : `d.mois`

Pour désigner l'année de naissance de l'étudiante `dupont` (de type `étudiant`) : `dupont.date_de_naissance.année`

```
# noël_75.mois;;
- : int = 12

# let dupont = {nom = "dupont" ; prénom = "isabelle" ;
                groupeTD = 1; date_de_naissance = noël_7
                no_de_sécu = "2750359023013 45" }
in dupont.date_de_naissance.année;;
 - : int = 1975

# let est_Noël d =
  d.jour = 25 && d.mois = 12;;
val est_Noël : date -> bool = <fun>
```

Plus généralement,

Syntaxe

Pour accéder à la composante étiquetée par `étiq_i` d'un enregistrement `expr`, on écrit `expr.étiq_i`

Typage

Le type de l'expression `expr.étiq_i` est `t_i` si `t` est le dernier type enregistrement déclaré possédant les étiquettes `étiq_1,...,étiq_n`

```
type t = { étiq_1 :  t_1 ;...étiq_i :  t_i ;...étiq_n
:  t_n }
```

et si `expr` a le type `t`

Évaluation

La valeur de l'expression `expr.étiq_i` est `v_i` si `expr` est un enregistrement dont la valeur est

```
{ étiq_1 = v_1 ; ...étiq_i = v_i ; ...étiq_n = v_n }
```

Un raccourci syntaxique pour créer une valeur de type enregistrement :

$\{enr\ with\ c_i = v\}$

équivalent à
$\{c_1 = enr.c_1\ ; \ldots.\ c_{i-1} = enr.c_{i-1}; c_i = v; c_{i+1} = enr.c_{i+1}\ \ldots\ ; c_k = enr.c_k\}$

Attention, ceci ne sert qu'à créer une valeur de type enregistrement à partir d'une autre.

Pour introduire 2 champs modifiés : $\{enr\ with\ c_i = v; c_j = w\}$

2 Filtrage des enregistrements

`match ...with ...` avec les enregistrements

Filtre d'enregistrement = enregistrement où les valeurs sont remplacées par des motifs

Pour désigner un jour de Noël : `{année=_; mois=12 ;jour=25}` (ordre des étiquettes sans importance)

```
# let est_Noël d =
  match d with
  | {jour=25; mois=12; année=_} -> true
  | _ -> false;;
val est_Noël : date -> bool = <fun>
```

Mieux : on peut omettre le champ année puisque nous n'en avons pas besoin (accès partiel) :

```
let est_Noel d = match d with
  | {mois=12; jour=25} -> true
  | _ -> false;;

let après_2000 d = match d with
  {année=a} -> a>2000;;
```

Le filtre {année=2000} est équivalent au filtre {année=2000;mois=_;jour=_}.

```
# let rec né_en (an, l) = match l with
  [] -> []
| {nom=n; date_de_naissance={année = a}}::r ->
  if an=a then n::(né_en (an, r)) else né_en (an, r);;
val né_en : int * étudiant list -> string list = <fun>
```