TD4

mercredi 5 octobre 2022 09:48

Exercice 4.1 [62] : Lever une exception

Définir une fonction OCaml `trouve` qui prend en arguments un prédicat p et une liste l, et qui :
- renvoie le premier élément de l qui vérifie la propriété p ;
- si aucun élément ne convient, lève une exception que vous aurez définie préalablement.

Que se passe-t-il si le calcul de p lève lui-même une exception, par exemple sur une division par 0 ?

• Rappel : exception Toto ...
raise (Toto (1, ...
try ...
with
| Toto(a,b) -> ...
| _ ->

let rec trouve = fun p l => match l with
| Nil -> raise(Toto(1, "nul germain"))
| Cons(x, l2) -> if p x then x else trouve p l2

si p lève une exep° alors le programme s'arrête et affiche l'exep° p

let rec trouve = fun p l -> match l with
| [] -> raise(Fail("not found"))
| x::l2 -> if p x then x else trouve p l2

si p... alors trouve rend une exep° de p

Exercice 4.2 [63] : Attraper une exception

Utiliser la fonction précédente pour définir une fonction `trouve_bis` qui renvoie :
- une liste contenant uniquement le premier élément de l qui vérifie la propriété p.
- une liste vide si aucun élément ne convient.

Même exercice avec comme type de résultat `'a option` au lieu de `'a list`.
Rappel : le type `'a option` comprend deux constructeurs :
- `None`
- `Some of 'a`

et on peut le voir comme codant des listes avec seulement 0 ou exactement 1 élément.

let trouver_bis = fun p l -> try [trouve p l] Somme(trouve p l) with
| Failure(s) -> [] None
| _ -> ...

type option = None | Some

Exercice 2.4 [52] : Concaténation de deux listes

Propriétés algébriques de la concaténation

- Rappeler la définition de la fonction concaténation.
- Démontrer que $[]$ est un élément neutre, c'est-à-dire : $\forall u,\ [] @ u = u = u @ []$.
- Démontrer que la concaténation est associative, c.-à-d. : $\forall u_1\ u_2\ u_3,\ (u_1 @ u_2) @ u_3 = u_1 @ (u_2 @ u_3)$.
  Indication : u_2 et u_3 étant fixés arbitrairement, procéder par récurrence structurelle sur u_1.

let rec (@) l1 l2 = match l1 with
| [] -> l2
| x::l -> x::(l @ l2)

- $\forall p \quad [] \oplus p = p$ trivial,
  $\forall p \quad p \oplus [] = p$, rec struct

- $\forall p_1, p_2, p_3 \quad p_1 \oplus p_2 \oplus p_3 = p_1 \oplus (p_2 \oplus p_3)$, induc° struct sur p_1

  $p_1 = [] \rightarrow [] \oplus p_2 \oplus p_3 = p_2 \oplus p_3$ et $[] \oplus (p_2 \oplus p_3) = (p_2 \oplus p_3) = p_2 \oplus p_3$

  $p_1 = x :: p \rightarrow x :: p \oplus p_2 \oplus p_3 = [x] \oplus p \oplus p_2 \oplus p_3 = [x] \oplus p \oplus (p_2 \oplus p_3)$ (HR)