TD #6 — Types somme, filtrage...

Exercice 1. *Inférence de types*

Donnez les **types OCaml** de chacune des expressions ci-dessous :

1. `[1; 2; 3]`
2. `[1, 2, 3]`
3. `[[]; []]`
4. `None`
5. `Some (Some None)`
6. `fun x d -> match x with Some v -> v | None -> d`
7. `function x::_ -> Some x | _ -> None`
8. `function (_,x)::_ -> Some x | _ -> None`
9. `fun () -> 0`
10. `fun 3 -> true`

La fonction définie par cette dernière expression est problématique...

11. Donnez deux exemples d'appel de cette fonction qui illustrent son comportement et ce problème.
12. Pourquoi la fonction de l'avant-dernière expression n'est-elle *pas* problématique ?

Exercice 2. *Définition de type*

1. Définissez un type somme permettant de représenter ou bien un `int` ou bien un `bool`
2. Donnez un exemple de construction d'une valeur de ce type pour chacun des cas.
3. Écrivez une fonction OCaml `ibadd : ib -> ib -> int` qui additionne deux valeurs de type `ib` en traitant `false` comme valant `0` et `true` comme `1`

Exercice 3. *Listes d'association*

On définit informellement une *liste d'association* comme une liste OCaml de type `('a * 'b) list`, où chaque couple `(k, v:'a * 'b)` présent dans la liste « associe » la *valeur* `v` à la *clef* `k`.

1. Écrivez une fonction OCaml :
   `find : 'a -> ('a * 'b) list -> 'b option`
   telle que pour `(x: ('a * 'b) list)`, et `(k : 'a)`, `find k x` s'évalue en `Some v` avec `v` la première (en partant de la tête) valeur telle que `k, v` apparaisse dans `x` si une telle valeur existe, et `None` sinon.

2. Illustrez l'utilisation de `find` avec trois exemples d'appel (et d'évaluation associée) bien choisis.
3. Déterminez le coût temporel pire cas de votre fonction `find`, et exprimez le de façon asymptotique en fonction d'un paramètre pertinent.
4. Écrivez une fonction OCaml :
   `add : 'a -> 'b -> ('a * 'b) list -> ('a * 'b) list`
   telle que pour `(x: ('a * 'b) list)`, `(k: 'a)`, `(v: 'b)`, `add k v x` s'évalue en une liste identique à `x` à laquelle on a ajouté l'association `(k, v)` *en tête*.
5. Écrivez une fonction OCaml :
   `del : 'a -> ('a * 'b) list -> ('a * 'b) list`
   telle que pour `(x: ('a * 'b) list)`, `(k: 'a)`, `del k x` s'évalue en une liste identique à `x` si aucune association faisant intervenir la clef `k` n'y apparaît, et sinon identique à `x` dont on a « supprimé » la première de ces associations (en partant de la tête).

Exercice 4. *Introduction aux exceptions*

OCaml possède un type `exn` de *valeurs exceptionnelles*, ou *exceptions*. Une exception peut être *levée* en lui appliquant la fonction `raise : exn -> 'a`. Le type de `raise` indique *que son évaluation ne terminera jamais* (nous reviendrons là-dessus en cours) : lorsqu'une exception est levée, le flot d'exécution normal du programme est interrompu jusqu'à ce que le programme tout entier termine brutalement, ou que l'exception soit *rattrapée*. Une exception se rattrape avec un filtrage de motif adapté, habituellement en utilisant la construction :
`try en with <Ex1> -> ex1 | <Ex2> -> ex2 ...`
qui s'évalue en `en` si aucune exception ne s'échappe lors de son évaluation, et en `ex1` si une exception correspondant au motif `<Ex1>` est levée (et pas déjà rattrapée) lors de son évaluation, et en `ex2` si une exception correspondant au motif `<Ex2>` (mais pas au motif `<Ex1>`) est…

Soit l'exception prédéfinie `exception Exit` :

1. En quoi s'évalue `try raise Exit with _ -> true` ?
2. En quoi s'évalue `try false with _ -> true` ?
3. Pourquoi l'expression `try false with _ -> None` n'est-elle pas typable ?

Soit l'exception prédéfinie `exception Failure of string` dont le constructeur (unaire) s'appelle avec un argument `string` (« chaîne de caractères » ; une `string` s'écrit entre deux « " ») :

4. Écrivez une fonction `head : 'a list -> 'a` qui s'évalue en la tête de son argument si elle en possède une, et lève une exception `Failure` sinon.
5. Illustrez son comportement avec trois exemples d'appels bien choisis.
6. Quelle alternative à une exception pourrait-on employer pour une telle fonction `head`, en changeant un peu d'approche (et son type) ?