Utilisation de VERROU en OCAML

Le type abstrait VERROU est implémenté en OCAML par le module `Mutex` de la librairie `Thread.Posix`. Pour charger ce module sur `utop` on tape donc `#require "threads.posix";;`. Afin de compiler un programme OCAML utilisant ce module on pourra utiliser la ligne de commande ci-dessous.

```
ocamlc -thread unix.cma threads.cma main.ml -o main
```

Les verrous sont des objets de type `Mutex.t`. On les manipule à travers les trois opérations suivantes :

- la fonction `Mutex.create : unit -> Mutex.t` qui crée un verrou ;
- la fonction `Mutex.lock : Mutex.t -> unit` qui permet de verrouiller un verrou ;
- la fonction `Mutex.unlock : Mutex.t -> unit` qui permet de déverrouiller un verrou.

Une fois un verrou `v` créé, on garantit l'exclusion mutuelle entre les sections de codes délimitées par un appel à `Mutex.lock v` et un appel à `Mutex.unlock v`.

Exemple 2. On donne ci-dessous deux façons de coder la cas de deux fils d'exécution devant réaliser la même tâche, à savoir incrémenter un compteur partagé un nombre donné de fois. Si le nombre d'incrémentation souhaité est le même pour les deux fils, ceux-ci peuvent partager la même structure pour leurs entrées/sorties, qui contient le nombre d'incrémentations et un champ mutable pour le compteur partagé. On aboutit alors au code de gauche. Dans le cas contraire, les deux fils ont des entrées différentes, et donc nécessairement des structures différentes pour leurs entrées/sorties. Dans ce cas on ne peut pas utiliser le champ mutable comme compteur commun, car ce seront alors deux compteurs différents. Ainsi on passe plutôt aux deux structures la même référence de type entier. On aboutit alors au code de droite.

```
type args =
  {
    nbt : int; (* nombre de tours, >= 0 *)
    mutable cpt : int;
  }

let verrou = Mutex.create ()

(** Ajoute [a.nbt] fois 1 à [a.cpt] *)
let add_one (a : args) : unit =
  for i = 1 to a.nbt do
    Mutex.lock verrou;
    a.cpt <- a.cpt + 1;
    Mutex.unlock verrou
  done

let main (n : int) : unit =
  let a = {nbt = n; cpt = 0} in
  let f1 = Thread.create add_one a in
  let f2 = Thread.create add_one a in
  Thread.join f1;
  Thread.join f2;
  let res = a.cpt in
```

```
type args =
  {
    nbt : int; (* nombre de tours, >= 0 *)
    cpt_ref : int ref;
  }

let verrou = Mutex.create ()

(** Ajoute [a.nbt] fois 1 à [a.cpt] *)
let add_one (a : args) : unit =
  for i = 1 to a.nbt do
    Mutex.lock verrou;
    a.cpt_ref :=  !(a.cpt_ref) + 1;
    Mutex.unlock verrou
  done

let main (n : int) : unit =
  let cpt = ref 0 (* compteur partagé *)
  ↪  in
  let a1 = {nbt = n; cpt_ref = cpt} in
  let a2 = {nbt = 2 * n; cpt_ref = cpt} in
  let f1 = Thread.create add_one a1 in
  let f2 = Thread.create add_one a2 in
  Thread.join f1;
  Thread.join f2;
  let res = !cpt in
```