Code fantôme : exemple

Calculer les nombres de Fibonacci avec une fonction récursive en $O(n)$:

```
let rec aux (a b n: int): int
  requires { 0 <= n }
  requires {                                                    }
  ensures  {
                                                                }
  variant  { n }
= if n = 0 then a else aux b (a+b) (n-1)

let fib_rec (n: int): int
  requires { 0 <= n }
  ensures  { result = fib n }
= aux 0 1 n

(* fib_rec 5 = aux 0 1 5 = aux 1 1 4 = aux 1 2 3 =
               aux 2 3 2 = aux 3 5 1 = aux 5 8 0 = 5 *)
```

Code fantôme : exemple

Calculer les nombres de Fibonacci avec une fonction récursive en $O(n)$:

```
let rec aux (a b n: int): int
  requires { 0 <= n }
  requires { exists k. 0 <= k /\ a = fib k /\ b = fib (k+1) }
  ensures  { exists k. 0 <= k /\ a = fib k /\ b = fib (k+1) /\
                                          result = fib (k+n) }
  variant  { n }
= if n = 0 then a else aux b (a+b) (n-1)

let fib_rec (n: int): int
  requires { 0 <= n }
  ensures  { result = fib n }
= aux 0 1 n

(* fib_rec 5 = aux 0 1 5 = aux 1 1 4 = aux 1 2 3 =
               aux 2 3 2 = aux 3 5 1 = aux 5 8 0 = 5 *)
```

Code fantôme : exemple

À la place des existentiels on peut utiliser un paramètre fantôme :

```
let rec aux (a b n: int) (ghost k: int): int
  requires { 0 <= n }
  requires { 0 <= k /\ a = fib k /\ b = fib (k+1) }
  ensures  { result = fib (k+n) }
  variant  { n }
= if n = 0 then a else aux b (a+b) (n-1) (k+1)

let fib_rec (n: int): int
  requires { 0 <= n }
  ensures  { result = fib n }
= aux 0 1 n 0
```

L'esprit du code fantôme

Le code fantôme sert à la spécification et à la preuve

$\Rightarrow$ le principe de non interférence :

On doit pouvoir éliminer le code fantôme
sans modifier le résultat du programme.

Par conséquence :

- le code normal ne lit pas des données fantômes

  si k est fantôme, alors $(k+1)$ l'est aussi

- le code fantôme ne modifie pas des données normales

  si r est une référence normale, alors $r \leftarrow \texttt{ghost}\ k$ est interdit

- le code fantôme ne change pas le flot de contrôle du code normale

  si c est fantôme, `if` c `then ...` et `while` c `do ...` le sont aussi

- le code fantôme termine

  `while true do skip done ; assert false` est prouvable

Code fantôme dans WHYML

Peuvent être déclarés fantômes :

- paramètres des fonctions

```
val aux (a b n: int) (ghost k: int): int
```

- champs des enregistrements et des variants

```
type queue 'a = { head: list 'a;  (* get from head *)
                  tail: list 'a;  (* add to tail *)
            ghost elts: list 'a;  (* logical view *) }
invariant { elts = head ++ reverse tail }
```

- variables et fonctions

```
let ghost x = qu.elts in ...
let ghost rev_elts qu = qu.tail ++ reverse qu.head
```

- expressions de programme

```
let x = ghost qu.elts in ...
```