Correction de fonctions récursives

Florian Bourse

Pour chacune des questions, on cherche à montrer la correction d'une fonction récursive dont le code est donné en OCaml. On supposera que les limites de la machine ne sont jamais atteintes (pas de dépassement de capacité pour les entiers, et pas de dépassement de la taille de la pile).

Question 1.

```
let rec factorielle n =  
  (* Précondition : n est un entier positif ou nul *)  
  (* Postcondition : factorielle n vaut n! *)  
  if n < 2 then 1  
  else n * factorielle (n-1)
```

Question 2.

```
let binomial k n =  
  (* Précondition : n est un entier positif, k est un entier entre 0 et n *)  
  (* Postcondition : binomial k n vaut (n choose k) soit (k parmi n) *)  
  (factorielle k * factorielle (n - k)) / factorielle n
```

Question 3.

```
let rec binomial k n =  
  if k < 0 || k > n then 0  
  else binomial k (n-1) + binomial (k-1) (n-1)
```

Question 4.

```
let rec binomial k n =  
  if k > n / 2 then binomial (n - k) n  
  else if k < 0 then 0  
  else if k = 0 then 1  
  else (n * binomial (k-1) (n-1)) / k
```

Question 5.

```
let factorielle =  
  let factorielle_aux acc n =  
    if n < 2 then acc  
    else factorielle_aux (n * acc) (n-1)  
  in  
  factorielle_aux 1
```

Question 6.

```
let rec log2 n =
  (* Précondition : n est un nombre entier positif strictement *)
  (* Postcondition : log2 n est le plus grand entier k tel que 2^k <= n *)
  if n = 1 then 0
  else 1 + log2 (n / 2)
```

Question 7.

```
let rec log2 =
  let rec log2_aux acc n =
    if n = 1 then acc
    else log2_aux (acc + 1) (n / 2)
  in
  log2_aux 0
```

Question 8.

```
let rec pgcd a b =
  (* Précondition : a et b sont des entiers positifs ou nuls *)
  (* Postcondition : pgcd a b est le plus grand diviseur commun à a et b *)
  if a = 0 then b
  else pgcd (b mod a) a
```

Question 9.

```
let rec nombre_triplets n =
  (* Précondition : n est un nombre entier positif ou nul *)
  (* Postcondition : nombre_triplets n est le nombre de triplets *)
  (* (i, j, k) tels que 0 <= i < j < k < n *)
  if n = 0 then 0
  else
    let rec nombre_couples n =
      if n = 0 then 0
      else n - 1 + nombre_couples (n-1)
    in
    nombre_couples (n-1) + nombre_triplets (n-1)
```

Question 10. Proposer une fonction plus efficace que la précédente pour résoudre le même problème.