Corrigé du DS1

Cases atteintes par un cavalier en p coups

Partie A – Préliminaires

A.1

a

```
# let print_bool = function true -> print_char `V` | _ -> print_string "F";;
print_bool : bool -> unit = <fun>
```

b

```
# let affiche_bool tab = let l = (vect_length tab - 1) in
        for i = 0 to l do
                print_string "|";
                for j = 0 to l do
                        print_bool tab.(i).(j); print_char `|`;
                        done;
                print_newline ();
        done;;
affiche_bool : bool vect vect -> unit = <fun>
```

A.2

```
# let rec filtre predicat = function
        | [] -> []
        | (a :: q) when (predicat a) -> a :: filtre predicat q
        | _ :: q -> filtre predicat q;;
filtre : ('a -> bool) -> 'a list -> 'a list = <fun>
```

A.3

```
# let interv n i = (0 <= i) && (i <= n - 1);;
interv : int -> int -> bool = <fun>

# let seuil n (i, j) = (interv n i) && (interv n j);;
seuil : int -> int * int -> bool = <fun>

# let filtre_echiquier n = filtre (seuil n);;
filtre_echiquier : int -> (int * int) list -> (int * int) list = <fun>

# let deplace n case = filtre_echiquier n (deplace_temp case);;
deplace : int -> int * int -> (int * int) list = <fun>
```

Partie B – Traitement fonctionnel récursif

a

```
# let fusion_simple e1 e2 (i, j) = (e1 (i, j)) or (e2 (i, j));;
fusion_simple : ('a * 'b -> bool) -> ('a * 'b -> bool) -> 'a * 'b -> bool =
 <fun>
```

b

```
(* Version non récursive terminale *)

# let rec fusion_non_term = function
        | [] -> failwith "Pas de fusion possible"
        | [e] -> e
        | e :: q -> fusion_simple e (fusion_non_term q);;
fusion_non_term : ('a * 'b -> bool) list -> 'a * 'b -> bool = <fun>

(* Version récursive terminale *)

# let fusion = let rec fusion_term accu = function
        | [] -> accu
        | e :: q -> fusion_term (fusion_simple accu e) q
in
        fusion_term (function (i, j) -> false) ;;
fusion : ('_a * '_b -> bool) list -> '_a * '_b -> bool = <fun>
```

B.1

```
# let etend e case = e case or not (filtre e) (deplace_n_declare case) = [];;
etend : (int * int -> bool) -> int * int -> bool = <fun>
```

B.2

```
(* Version avec récursivité croisée *)

# let rec liste_acces p = function
        | [] -> []
        | a :: q -> (accessibles1 a p) :: (liste_acces p q)
and
accessibles1 (i0, j0) = function
        | 0 -> (function (i, j) -> i = i0 && j = j0)
        | p -> fusion (liste_acces (p - 1) (deplace_n_declare (i0, j0)));;
liste_acces : int -> (int * int) list -> (int * int -> bool) list = <fun>
accessibles1 : int * int -> int -> int * int -> bool = <fun>

(* Version récursive terminale *)

# let rec accessibles_rec echiquier (i0, j0) = function
        | 0 -> fusion_simple echiquier (function (i, j) -> i = i0 && j = j0)
        | p -> accessibles_rec (etend echiquier) (i0, j0) (p - 1);;
accessibles_rec :
 (int * int -> bool) -> int * int -> int -> int * int -> bool = <fun>

# let accessibles1bis case = accessibles_rec (function x -> x = case) case;;
accessibles1bis : int * int -> int -> int * int -> bool = <fun>
```

Partie C – Traitement impératif et récursif

C.1

```
# let init n = make_matrix n n false;;
init : int -> bool vect vect = <fun>
```

C.2

```
# let valeur config case = config.(fst case).(snd case);;
valeur : 'a vect vect -> int * int -> 'a = <fun>
```

```
(* ou *)

# let valeur_bis config (i, j) = config.(i).(j);;
valeur_bis : 'a vect vect -> int * int -> 'a = <fun>
```

C.3

```
# let transmis case config = let n = vect_length config in
        not (filtre (valeur config) (deplace n case) = []);;
transmis : int * int -> bool vect vect -> bool = <fun>
```

C.4

```
# let atteintes config = let n = vect_length config in let temp = init n in
        for i = 0 to (n - 1) do
                for j = 0 to (n - 1) do
                        temp.(i).(j) <- config.(i).(j) or transmis (i, j) config
                done
        done;
        temp;;
atteintes : bool vect vect -> bool vect vect = <fun>
```

C.5

```
# let rec accessibles echiquier case = function
        | 0 -> echiquier.(fst case).(snd case) <- true; echiquier
        | p -> atteintes (accessibles echiquier case (p - 1));;
accessibles : bool vect vect -> int * int -> int -> bool vect vect = <fun>

# let accessibles2 n = accessibles (init n);;
accessibles2 : int -> int * int -> int -> bool vect vect = <fun>
```

Partie D – Traitement purement impératif

D.1

```
# let echiquier_rempli echiquier =
        let n = vect_length echiquier in
                let temp = make_matrix n n true in
                        echiquier = temp;;
echiquier_rempli : bool vect vect -> bool = <fun>

(* Autre version, bien plus lourde *)

# let echiquier_rempli_lourd echiquier =
        let temp = ref true and n = vect_length echiquier in
         for i = 0 to (n - 1) do
                for j = 0 to (n - 1) do
                        temp := !temp && echiquier.(i).(j)
                done
        done;
        !temp;;
echiquier_rempli_lourd : bool vect vect -> bool = <fun>
```

D.2

```
# let accessibles3 n case p =
        let temp = ref (init n) and i = ref 0 in
        !temp.(fst case).(snd case) <- true;
                while
```

```
                    (not (echiquier_rempli !temp)) && !i < p do
                            temp := atteintes !temp; i := !i + 1 done;
        !temp;;
accessibles3 : int -> int * int -> int -> bool vect vect = <fun>
```

Partie E – Problèmes connexes

E.1

```
# let pcoups n case p = let temp = init n in
        if p = 0 then accessibles3 n case p
        else
        begin
        for i = 0 to (n - 1) do
                for j = 0 to (n - 1) do
                        temp.(i).(j) <- (accessibles3 n case p).(i).(j)
                                && not (accessibles3 n case (p - 1)).(i).(j) ;
                done
        done;
        temp;
        end;;
pcoups : int -> int * int -> int -> bool vect vect = <fun>
```

E.2

```
# let coups_suivants config = let n = vect_length config in let temp = init n in
        for i = 0 to (n - 1) do for j = 0 to (n - 1) do
                temp.(i).(j)  <- transmis (i, j) config done done; temp;;
coups_suivants : bool vect vect -> bool vect vect = <fun>

# let rec positions_possibles echiquier case = function
        | 0 -> echiquier.(fst case).(snd case) <- true; echiquier
        | p -> coups_suivants (positions_possibles echiquier case (p - 1));;
positions_possibles : bool vect vect -> int * int -> int -> bool vect vect =
 <fun>

# let apres_p_coups n = positions_possibles (init n);;
result_pos : int -> int * int -> int -> bool vect vect = <fun>
```

```
(* Résultats *)

(* Fonction de conversion de la première vers la seconde modélisation *)

# let convert echiquier_fonctionnel n = let temp = make_matrix n n false in
        for i = 0 to (n - 1) do
                for j = 0 to (n - 1) do
                        temp.(i).(j) <- echiquier_fonctionnel (i, j)
                done
        done;
        temp;;
convert : (int * int -> bool) -> int -> bool vect vect = <fun>

(* Programme pour comparer le temps d'exécution des algorithmes *)

# let algo_temps algo n case p = let temp = sys__time () in
        affiche_bool (algo n case p);
        sys__time () -. temp;;
algo_temps : ('a -> 'b -> 'c -> bool vect vect) -> 'a -> 'b -> 'c -> float =
 <fun>

(* Tests correspondant aux différentes réponses *)

# let test1 = algo_temps (fun n case p -> convert (accessibles1 case p) n);;
test1 : int -> int * int -> int -> float = <fun>

# let test1bis = algo_temps (fun n case p -> convert (accessibles1bis case p) n);;
test1bis : int -> int * int -> int -> float = <fun>

# let test2 = algo_temps accessibles2;;
test2 : int -> int * int -> int -> float = <fun>

# let test3 = algo_temps accessibles3;;
test3 : int -> int * int -> int -> float = <fun>

(* Données globales *)

# let coups = 4 and pos = (7, 7);;
coups : int = 4
pos : int * int = 7, 7

(* Confrontation des algorithmes proposés *)

# test1 n_global pos coups;;
 |F|F|V|F|V|F|V|F|V|F|V|F|
 |F|V|F|V|V|V|V|V|V|V|V|V|
 |V|F|V|V|V|V|V|V|V|V|V|V|
 |F|V|V|V|V|V|V|V|V|V|V|V|
 |V|V|V|V|V|V|V|V|V|V|V|V|
 |F|V|V|V|V|V|V|V|V|V|V|V|
 |V|V|V|V|V|V|V|V|V|V|V|V|
 |F|V|V|V|V|V|V|V|V|V|V|V|
 |V|V|V|V|V|V|V|V|V|V|V|V|
 |F|V|V|V|V|V|V|V|V|V|V|V|
 |V|V|V|V|V|V|V|V|V|V|V|V|
 |F|V|V|V|V|V|V|V|V|V|V|V|
- : float = 0.032
```

```
# test1bis n_global pos coups;;
|F|F|V|F|V|F|V|F|V|F|V|F|
|F|V|F|V|V|V|V|V|V|V|V|V|
|V|F|V|V|V|V|V|V|V|V|V|V|
|F|V|V|V|V|V|V|V|V|V|V|V|
|V|V|V|V|V|V|V|V|V|V|V|V|
|F|V|V|V|V|V|V|V|V|V|V|V|
|V|V|V|V|V|V|V|V|V|V|V|V|
|F|V|V|V|V|V|V|V|V|V|V|V|
|V|V|V|V|V|V|V|V|V|V|V|V|
|F|V|V|V|V|V|V|V|V|V|V|V|
|V|V|V|V|V|V|V|V|V|V|V|V|
|F|V|V|V|V|V|V|V|V|V|V|V|
- : float = 0.125

# test2 n_global pos coups;;
|F|F|V|F|V|F|V|F|V|F|V|F|
|F|V|F|V|V|V|V|V|V|V|V|V|
|V|F|V|V|V|V|V|V|V|V|V|V|
|F|V|V|V|V|V|V|V|V|V|V|V|
|V|V|V|V|V|V|V|V|V|V|V|V|
|F|V|V|V|V|V|V|V|V|V|V|V|
|V|V|V|V|V|V|V|V|V|V|V|V|
|F|V|V|V|V|V|V|V|V|V|V|V|
|V|V|V|V|V|V|V|V|V|V|V|V|
|F|V|V|V|V|V|V|V|V|V|V|V|
|V|V|V|V|V|V|V|V|V|V|V|V|
|F|V|V|V|V|V|V|V|V|V|V|V|
- : float = 0.0

# test3 n_global pos coups;;
|F|F|V|F|V|F|V|F|V|F|V|F|
|F|V|F|V|V|V|V|V|V|V|V|V|
|V|F|V|V|V|V|V|V|V|V|V|V|
|F|V|V|V|V|V|V|V|V|V|V|V|
|V|V|V|V|V|V|V|V|V|V|V|V|
|F|V|V|V|V|V|V|V|V|V|V|V|
|V|V|V|V|V|V|V|V|V|V|V|V|
|F|V|V|V|V|V|V|V|V|V|V|V|
|V|V|V|V|V|V|V|V|V|V|V|V|
|F|V|V|V|V|V|V|V|V|V|V|V|
|V|V|V|V|V|V|V|V|V|V|V|V|
|F|V|V|V|V|V|V|V|V|V|V|V|
- : float = 0.0

(* Test de pcoups *)

# for i = 0 to 4 do affiche_bool (pcoups 8 (7, 7) i); print_newline () done;;
|F|F|F|F|F|F|F|F|
|F|F|F|F|F|F|F|F|
|F|F|F|F|F|F|F|F|
|F|F|F|F|F|F|F|F|
|F|F|F|F|F|F|F|F|
|F|F|F|F|F|F|F|F|
|F|F|F|F|F|F|F|F|
|F|F|F|F|F|F|F|V|
|F|F|F|F|F|F|F|F|
|F|F|F|F|F|F|F|F|
|F|F|F|F|F|F|F|F|
|F|F|F|F|F|F|F|F|
|F|F|F|F|F|F|F|F|
|F|F|F|F|F|F|V|F|
|F|F|F|F|F|V|F|F|
|F|F|F|F|F|F|F|F|

|F|F|F|F|F|F|F|F|
|F|F|F|F|F|F|F|F|
|F|F|F|F|F|F|F|F|
|F|F|F|F|F|V|F|V|
|F|F|F|F|V|F|V|F|
|F|F|F|V|F|F|F|V|
|F|F|F|F|V|F|F|F|
|F|F|F|V|F|V|F|F|

|F|F|F|F|F|F|F|F|
|F|F|F|F|V|F|V|F|
|F|F|F|V|F|V|F|V|
|F|F|V|F|V|F|V|F|
|F|V|F|V|F|V|F|V|
|F|F|V|F|V|F|F|F|
|F|V|F|V|F|F|F|V|
|F|F|V|F|V|F|V|F|

|F|F|V|F|V|F|V|F|
|F|V|F|V|F|V|F|V|
|V|F|V|F|V|F|V|F|
|F|V|F|V|F|F|F|F|
|V|F|V|F|F|F|F|F|
|F|V|F|F|F|V|F|F|
|V|F|V|F|F|F|V|F|
|F|V|F|F|F|F|F|F|

- : unit = ()

(* Test de apres_p_coups *)

# affiche_bool (apres_p_coups 12 (7, 7) 9);;
|F|V|F|V|F|V|F|V|F|V|F|V|
|V|F|V|F|V|F|V|F|V|F|V|F|
|F|V|F|V|F|V|F|V|F|V|F|V|
|V|F|V|F|V|F|V|F|V|F|V|F|
|F|V|F|V|F|V|F|V|F|V|F|V|
|V|F|V|F|V|F|V|F|V|F|V|F|
|F|V|F|V|F|V|F|V|F|V|F|V|
|V|F|V|F|V|F|V|F|V|F|V|F|
|F|V|F|V|F|V|F|V|F|V|F|V|
|V|F|V|F|V|F|V|F|V|F|V|F|
|F|V|F|V|F|V|F|V|F|V|F|V|
|V|F|V|F|V|F|V|F|V|F|V|F|
- : unit = ()
```