```
(* corrigé du TD Automate 1 *)
(* travail sur des automates déterministes *)
(* par Loïc Le Loarer *)

(* Question 1 : le type automate *)
type ('a,'b) automate = { alphabet : 'b list ; init : 'a list ;
 term : 'a list ; trans : ('a * 'b * 'a) list};;

(* l'automate A1 *)
let auto1 = { alphabet = [`a`;`b`]; init = [0] ; term = [3] ;
              trans = [(0,`a`,1);(0,`b`,0);(1,`b`,2);(1,`a`,1);
                       (2,`b`,0);(3,`a`,1);(3,`b`,2);(2,`a`,3)] };;
(* l'automate A2 *)
let auto2 = { alphabet = [`a`;`b`]; init = [0] ; term = [6] ;
              trans = [(0,`a`,1);(0,`b`,3);(3,`b`,6);(2,`a`,0);
                       (6,`b`,5);(4,`a`,5);(1,`a`,3)] };;
(* l'automate A3 *)
let auto3 = { alphabet = [`a`;`b`]; init = [0] ; term = [6] ;
              trans = [(0,`a`,1);(0,`b`,4);(1,`b`,4);(2,`a`,0);
                       (2,`b`,4);(2,`b`,5);(3,`a`,4);
                       (5,`b`,3);(3,`b`,6);(6,`a`,1)] };;

(* élimine les doublons d'une liste *)
(* version récursive terminale mais qui inverse l'ordre des arguments *)
let sans_doublon l =
    let rec f l = function
        [] -> l
      | e::r when mem e l -> f l r
      | e::r -> f (e::l) r
   in f [] l;;
sans_doublon [1;2;3;4;5;1;2;3;4;6;6;7;8;7];;
(* et en version non terminale mais plus simple *)
let rec sans_doublon = function
  | [] -> []
  | e :: r when mem e r -> sans_doublon r
  | e :: r -> e :: (sans_doublon r);;
(* ou encore plus simple mais plus difficile à lire sans doute *)
let sans_doublon =
  it_list (fun r e -> if mem e r then r else e :: r) [];;

(* Question 2 : liste des états *)
let etats a =
  sans_doublon ((map (function i,_,_ -> i) a.trans)
                @ (map (function _,_,i -> i) a.trans));;

etats auto1;;

(* Question 3 : affiche un automate de manière lisible *)
(* ici, on imprime que les transitions *)
let affiche a print_etats print_lettre  =
  do_list (function (i,c,f) -> print_etats i; print_string " vers ";
    print_etats f; print_string " par ";
    print_lettre c; print_string "\n") a.trans;;

affiche auto1 print_int print_char;;

(* Question 4 : test si un automate est déterministe *)
let isdeterm a =
  let rec aux l = function
      [] -> true
    | (i,s,_)::_ when mem (i,s) l -> false
    |   (i,s,_)::r -> aux ((i,s)::l) r
```



```
  in aux [] a.trans;;
isdeterm auto1;;

(* la fonction string_to_list *)
let string_to_list s =
  let n = string_length s in
  let rec aux = function
      | i when i = n -> []
      | i -> s.[i] :: (aux (i+1)) in
  aux 0;;

(* Question 5 : la fonction image *)
let image a e s =
  let rec aux = function
      | [] -> raise Not_found
      | (i,c,f) :: _ when i = e & c = s -> f
      | _ :: r -> aux r in
  aux a.trans;;

image auto1 3 `a`;;

(* Question 6 : la fonction reconnait *)
let reconnait a l =
  let rec aux e = function
      | [] -> mem e a.term
      | c :: r -> aux (image a e c) r in
  try
    aux (hd a.init) l
  with Not_found -> false;;
(* mais aussi avec une fonction bien pratique de la librairie *)
let reconnait a l =
  try
    mem (it_list (image a) (hd a.init) l) a.term
  with Not_found -> false;;

reconnait auto1 (string_to_list "abbaabaaaba");;

(* insere un élement dans une liste triée *)
let rec insere x = function
    [] -> [x]
  | e::_ as l when x = e -> l
  | e::_ as l when x < e -> x::l
  | e::r -> e::(insere x r);;

(* Question 7 : donne la liste des états accessibles *)
let rec accessibles trans q' =
  let rec auxi = function
      | [] -> q'
      | (i,_,f) :: r ->
          if mem i q'
          then insere f (auxi r)
          else auxi r in
  let q'' = auxi trans in
  if q' = q'' then q' else accessibles trans q'';;
(* ou *)
let rec accessibles trans q' =
    let q'' = it_list
          (fun l (i,_,f) ->
            if mem i q' then insere f l
            else l)
          q' trans in
    if q' = q'' then q' else accessibles trans q'';;
```

```
accessibles auto3.trans auto3.init;;

(* Question 8 : donne la liste des états coaccessibles *)
let rec coaccessibles trans q' =
  let rec auxi = function
    | [] -> q'
    | (i,_,f) :: r ->
        if mem f q'
        then insere i (auxi r)
        else auxi r in
  let q'' = auxi trans in
  if q' = q'' then q' else coaccessibles trans q'';;
(* ou *)
let rec coaccessibles trans q' =
    let q'' = it_list
         (fun l (i,_,f) ->
           if mem f q' then insere i l
           else l)
        q' trans in
    if q' = q'' then q' else coaccessibles trans q'';;

coaccessibles auto3.trans auto3.term;;

(* Question 9 : émonde un automate *)
let emonde a =
  let etatsrestants = intersect (accessibles a.trans a.init)
      (coaccessibles a.trans a.term) in
  {trans = it_list
      (fun l ((i,_,f) as t) ->
        if mem i etatsrestants && mem f etatsrestants then t :: l
        else l)
      [] a.trans;
    init = intersect a.init etatsrestants;
    term = intersect a.term etatsrestants;
    alphabet = a.alphabet }
    ;;
let auto2' = emonde auto2;;
etats auto2';;

(* Question 10 *)
let auto3' = emonde auto3;;
etats auto3';;
```