```
(* corrigé du TD Automate 2 *)
(* travail sur des automates non-déterministes *)
(* par Loïc Le Loarer *)

(* 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`,0);(0,`b`,0);(0,`a`,1);
                        (1,`b`,2);(2,`a`,3)] };;

(* 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);;

(* liste des états sans la fonction sans_doublon *)
let etats a =
  it_list (fun l (i,_,j) -> insere i (insere j l)) []  a.trans ;;

etats auto1;;

(* 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 1 : la fonction image *)
let image a e s =
  let rec aux = function
    | [] -> []
    | (i,c,f) :: r when mem i e & c = s -> insere f (aux r)
    | _ :: r -> aux r in
  aux a.trans;;
(* ou *)
let image a e s =
  it_list
    (fun im (i,c,f) ->
      if mem i e & c = s then insere f im else im)
    [] a.trans ;;

image auto1 [0] `a`;;

(* Question 2 : la fonction reconnait *)
let reconnait a l =
  (intersect
```



```
     (it_list (fun etat lettre -> image a etat lettre) a.init l)
     a.term) <> [] ;;

reconnait auto1 (string_to_list "abbaabaaaba");;

(* Question 3 : 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'';;

(* Question 4 : 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'';;

(* Question 5 : 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 }
    ;;

(* Question 6 : le type automate_e *)
type ('a,'b) automate_e =
    { alphabet : 'b list ; init : 'a list ;
      term : 'a list ; trans : ('a * 'b * 'a) list;
      trans_e : ('a * 'a) list };;
```

Automates non–déterministes


```
(* l'automate A1 *)
let auto1 = { alphabet = [`a`;`b`]; init = [0] ; term = [3] ;
              trans_e = [3,0] ;
              trans = [ (0,`a`,0);(0,`b`,0);(0,`a`,1);
                        (1,`b`,2);(2,`a`,3)] };;

(* Question 7 : la fonction image_e *)
(* d'abord, une aide : la cloture d'une liste d'états *)
(* algorithme : on augmente la cloture (etats) tant que l'on peut, et on obtient
   la cloture *)
let rec cloture a etats =
  let etats' = it_list
      (fun l (e,e') -> if mem e etats then insere e' l else l)
      etats a.trans_e in
  if etats = etats' then etats else cloture a etats' ;;

(* l'image : c'est la cloture de l'image de la cloture *)
let image a e s =
  it_list
    (fun im (i,c,f) ->
      if mem i e & c = s then insere f im else im)
    [] a.trans ;;

let image_e a e c =
  cloture a (image a (cloture a e) c) ;;

(* Question 8 : la fonction ote *)

let ote_e a =
  let rec aux trans =
    let auxi (e,e') =
      it_list (fun l (i,c,f) -> if i=e' then insere (e,c,f) l else l)
        (it_list
           (fun l (i,c,f) -> if f=e then insere (i,c,e') l else l)
           trans trans)
        trans in
    let trans' = flat_map auxi a.trans_e in
    if trans'=trans then trans else aux trans' in
  { alphabet = a.alphabet; init = a.init; term = a.term ;
    trans_e = [];
    trans = aux a.trans } ;;
let auto1' = ote_e auto1;;
```