```
let print_int_list = do_list print_int;;
print_int_list [0;2;1;0;1;2;1;0];;

let list_max = function
  | [] -> None
  | a::q -> Some (it_list max a q);;

type 'a l_systeme = {
  axiome: 'a list;
  morphisme: 'a -> 'a list
};;

let l_systeme rang ls =
  let mot = ref ls.axiome in
  for i=1 to rang do mot := flat_map ls.morphisme !mot done;
  !mot;;

let thue_morse = {
  axiome=[0];
  morphisme= (function
       0 -> [0;1]
     | 1 -> [1;0]
     | _ ->failwith"thue_morse")};;

l_systeme 5 thue_morse;;

(* 2.6 Il est clairement nécessaire que m_0 soit préfixe de m_1.
   C'est suffisant par récurrence:
   Si m_{n-1} est préfixe de m_n, m_n s'écrit m_{n-1}u et donc
   m_n est préfixe de m_{n+1} = f(m_n) = f(m_{n-1} u) = f(m_{n-1}) f(u) = m_n f(u).
*)

let rec prefixe la lb = match la,lb with
  | [],_ -> true
  | _,[] -> false
  | ta::qa, tb::qb -> ta = tb && prefixe qa qb;;

let mot_infini ls =
  prefixe ls.axiome (l_systeme 1 ls);;

(* [borne_inf; borne_sup] est l'intervalle des lettres qui apparaissent dans la suite des m_i,
   mais pour lesquelle l'image n'a pas encore été testée
   (i.e. leur image peut contenir de nouvelles lettres.).
   [0; borne_inf[ sont les lettres qui ont déjà été testées. *)
let taille_alphabet ls =
  let borne_inf = ref 0 in
  let m1 = l_systeme 1 ls in
  let borne_sup = ref (match list_max ls.axiome, list_max m1 with
    |Some a, Some b -> max a b
    |Some a, None -> a
    |None,_ -> failwith "axiome vide") in
  while !borne_inf <= !borne_sup do
    (match list_max (ls.morphisme !borne_inf) with
      | None -> ()
      | Some i -> borne_sup := max i !borne_sup);
    incr borne_inf;
  done;
  !borne_sup +1;;

taille_alphabet {
  morphisme =
    (function
       | 0 -> [0;1;2]
       | 1 -> [0]
       | 2 -> [3]
       | 3 -> [1]
       | _ -> failwith "bug");
  axiome=[0]};;

taille_alphabet {
  morphisme =
    (function
       | 0 -> [0;1;2]
       | 1 -> [0]
       | 2 -> [1]
       | 3 -> [1]
       | _ -> failwith "bug");
  axiome=[0]};;

let ecrit i = match i with
  | 0 -> "zero"
  | 1 -> "un"
  | 2 -> "deux"
  | 3 -> "trois"
  | 4 -> "quatre"
```

```
  | 5 -> "cinq"
  | 6 -> "six"
  | 7 -> "sept"
  | 8 -> "huit"
  | 9 -> "neuf"
  | _ -> invalid_arg "ecrit";;
let bizarre i =
  let i = ecrit i in
  let result = ref [] in
  for j = string_length i -1 downto 0 do
    result := (int_of_char i.[j] - 97) mod 10 :: !result;
  done;
  !result;;

taille_alphabet {morphisme=bizarre; axiome=[0]};;

let ecrit i = match i with
  | 0 -> "zero"
  | 1 -> "un"
  | 2 -> "deux"
  | 3 -> "drei"
  | 4 -> "vier"
  | 5 -> "funf"
  | 6 -> "six"
  | 7 -> "seven"
  | 8 -> "ocho"
  | 9 -> "nueve"
  | _ -> invalid_arg "ecrit";;
let bizarre i =
  let i = ecrit i in
  let result = ref [] in
  for j = string_length i -1 downto 0 do
    result := (int_of_char i.[j] - 97) mod 10 :: !result;
  done;
  !result;;
taille_alphabet {morphisme=bizarre; axiome=[0]};;
map (fun i -> i,bizarre i) [0;1;2;3;4;5;6;7;8;9];;


let lettres_mortelles ls =
  let ta = taille_alphabet ls in
  let images = init_vect ta (fun i -> flat_map ls.morphisme [i]) in
  let a_eliminer = ref [] in
  for i=0 to ta-1 do
    if images.(i) = [] then a_eliminer := i :: !a_eliminer;
  done;
  let resultat = ref [] in
  while !a_eliminer <> [] do
    let a = hd !a_eliminer in
    resultat := a :: !resultat;
    a_eliminer := tl !a_eliminer;
    for i=0 to ta-1 do
      if images.(i) <> [] then
        (images.(i) <- subtract images.(i) [a];
         if images.(i) = [] then a_eliminer := i :: !a_eliminer);
    done;
  done;
  !resultat;;

lettres_mortelles
 {
  morphisme =
    (function
       | 0 -> [0;1;2]
       | 1 -> []
       | 2 -> [3;0]
       | 3 -> [1]
       | _ -> failwith "bug");
  axiome=[0]};;

let lettres_mortelles ls =
  let ta = taille_alphabet ls in
  let nb = make_matrix ta ta 0 in
  let longueurs = make_vect ta 0 in
  let a_eliminer = ref [] in
  for beta=0 to ta-1 do
    let image = flat_map ls.morphisme [beta] in
    do_list (fun alpha -> nb.(alpha).(beta) <- nb.(alpha).(beta) +1) image;
    longueurs.(beta) <- list_length image;
    if longueurs.(beta) = 0 then a_eliminer := beta :: !a_eliminer;
  done;
  let resultat = ref [] in
  while !a_eliminer <> [] do
    let a = hd !a_eliminer in
    resultat := a :: !resultat;
```

```
    a_eliminer := tl !a_eliminer;
    for beta = 0 to ta-1 do
      if longueurs.(beta) > 0 then (
      longueurs.(beta) <- longueurs.(beta) - nb.(a).(beta);
      if longueurs.(beta) = 0 then a_eliminer := beta :: !a_eliminer)
    done;
  done;
  !resultat;;

lettres_mortelles
 {
  morphisme =
    (function
       | 0 -> [0;1;2]
       | 1 -> []
       | 2 -> [3;0]
       | 3 -> [1]
       | _ -> failwith "bug");
  axiome=[0]};;

(* ********************************************************************** *)
#open "graphics";;
open_graph"";;

type etat = {x:float; y:float; dir:float};;
type lettre =
  | A (* Avance *)
  | V (* Vole *)
  | G (* Gauche *)
  | D (* Droite *)
  | S (* Sauvegarde *)
  | R (* Restauration *)
;;

let trace pas angle etat mot =
  let pile = stack__new () in
  let rec une_lettre etat = function
    | A ->
        let new_etat = {
          x = etat.x +. pas *. cos etat.dir;
          y = etat.y +. pas *. sin etat.dir;
          dir = etat.dir} in
        moveto (int_of_float etat.x) (int_of_float etat.y);
        lineto (int_of_float new_etat.x) (int_of_float new_etat.y);
        new_etat
    | V ->
        let new_etat = {
          x = etat.x +. pas *. cos etat.dir;
          y = etat.y +. pas *. sin etat.dir;
          dir = etat.dir} in
        moveto (int_of_float new_etat.x) (int_of_float new_etat.y);
        new_etat
    | G -> {x=etat.x; y=etat.y; dir = etat.dir +. angle}
    | D -> {x=etat.x; y=etat.y; dir = etat.dir -. angle}
    | S -> stack__push etat pile; etat
    | R -> stack__pop pile in
  it_list une_lettre etat mot;;

let pi = acos (-1.);;

let main pas angle mot =
  clear_graph ();
  let x = size_x () / 2 in
  let y = size_y () / 2 in
  moveto x y;
  trace pas angle {x=float_of_int x; y=float_of_int y; dir=pi/.2.} mot;;

main 40. (pi/.4.) [
  S;
    S;D;D;A;R;A;S;A;D;D;A;R;D;D;A;
  R;D;D;V;V;
  S;
    G;G;A;A;D;D;D;A;A;A;G;G;G;A;A;
  R;V;V;V;
  A;G;G;A;G;G;A;D;D;A;D;D;A;];;

let dragon = {
    morphisme = (function
(*     | A -> [G;A;D;D;V;G;S;G;G;G;A;R] *)
(*     | A -> [G;A;D;A;V;D;V;G;S;G;G;A;D;A;R] *)
    | A -> [G;A;D;V;D;A;G;V;S;G;G;A;D;V;G;A;R]
    | V -> [V;V]
    | G -> [G]
    | D -> [D]
    | S -> [S]
```

```
    | R -> [R]);
  axiome = [A]};;
main 2. (pi/.2.) (l_systeme 8 dragon);;
let ignore _ = ();; (* juste pour éviter un warning *)
for i=0 to 20 do
  ignore (main 2. (pi/.2.+.(float_of_int (i-10) /. 10.)) (l_systeme 8 dragon));
done;;

let ile = {
  morphisme = (function
    | A -> [A;G;A;D;A;D;A;A;G;A;G;A;D;A]
    | V -> [V]
    | G -> [G]
    | D -> [D]
    | S -> [S]
    | R -> [R]);
  axiome = [A;D;A;D;A;D;A;D]
};;
main 160. (pi/.2.) (l_systeme 0 ile);;
main 40. (pi/.2.) (l_systeme 1 ile);;
main 10. (pi/.2.) (l_systeme 2 ile);;
main 0.5 (pi/.2.) (l_systeme 5 ile);;

let von_koch = {
  morphisme = (function
    | A -> [A;G;A;D;D;A;G;A]
    | V -> [V]
    | G -> [G]
    | D -> [D]
    | S -> [S]
    | R -> [R]);
  axiome = [A;D;D;A;D;D;A]};;

main 1. (pi/.3.) (l_systeme 6 von_koch);;

let arbre1 = {
  morphisme = (function
    | A -> [A;S;G;A;R;A;S;D;A;R;S;A;R]
    | V -> [V]
    | G -> [G]
    | D -> [D]
    | S -> [S]
    | R -> [R]);
  axiome = [A]};;
main 120. (pi/.7.) (l_systeme 1 arbre1);;
main 60. (pi/.7.) (l_systeme 2 arbre1);;
main 7. (pi/.7.) (l_systeme 5 arbre1);;

let arbre2 = {
  morphisme = (function
    | A -> [A;S;S;G;A;R;D;A;R]
    | V -> [V]
    | G -> [G]
    | D -> [D]
    | S -> [S]
    | R -> [R]);
  axiome = [A]};;
main 100. (pi/.8.) (l_systeme 1 arbre2);;
main 50. (pi/.8.) (l_systeme 2 arbre2);;
main 25. (pi/.8.) (l_systeme 3 arbre2);;
main 20. (pi/.8.) (l_systeme 8 arbre2);;



type lettre =
  | A (* Avance *)
  | G (* Gauche *)
  | D (* Droite *)
  | S (* Sauvegarde *)
  | R (* Restauration *)
  | B (* Bourgeon *)
  | T (* Tige *)
;;

let trace pas angle etat mot =
  let pile = stack__new () in
  let rec une_lettre etat = function
    | A ->
        let new_etat = {
          x = etat.x +. pas *. cos etat.dir;
          y = etat.y +. pas *. sin etat.dir;
          dir = etat.dir} in
        moveto (int_of_float etat.x) (int_of_float etat.y);
        lineto (int_of_float new_etat.x) (int_of_float new_etat.y);
        new_etat
```

```
    | G -> {x=etat.x; y=etat.y; dir = etat.dir +. angle}
    | D -> {x=etat.x; y=etat.y; dir = etat.dir -. angle}
    | S -> stack__push etat pile; etat
    | R -> stack__pop pile
    | B|T -> etat in
  it_list une_lettre etat mot;;

let main pas angle mot =
  clear_graph ();
  let x = size_x () / 2 in
  let y = size_y () / 2 in
  moveto x y;
  trace pas angle {x=float_of_int x; y=float_of_int y; dir=pi/.2.} mot;;

type l_systeme = {morphisme: lettre -> lettre list; axiome: lettre list};;
let l_systeme rang ls =
  let mot = ref ls.axiome in
  for i=1 to rang do mot := flat_map ls.morphisme !mot done;
  !mot;;

let arbre3 = {
  morphisme = (function
    | B -> [T;S;G;B;R;T;S;D;D;B;R;B]
    | T -> [T;A]
    | A -> [A]
    | G -> [G]
    | D -> [D]
    | S -> [S]
    | R -> [R]);
  axiome = [B]};;

main 10. (pi/.7.) (l_systeme 8 foo);;
;;
```