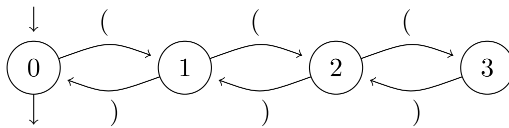
TP 2 – Automates et mots – corrigé

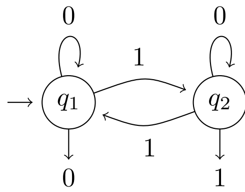
Jean-Baptiste Rouquier

4 et 11 octobre 2005

Voici l'automate de la question 2.3 :



et celui de la question 2.6 :



```
type automate = {
  initial: int;
  finaux: int list;
  arcs: (char * int) list vect};;

let accepte automate mot =
  let etat = ref automate.initial in
  try
    for i=0 to pred (string_length mot) do
      etat := assoc mot.[i] automate.arcs.(!etat);
    done;
    mem !etat automate.finaux
  with Not_found -> false;;

(* Pour ceux qui tiennent à l'écrire en fonctionnel: *)
let accepte automate mot =
  let len = string_length mot in
  let rec calcul etat pos = (* mange une lettre et s'appelle récursivement *)
    if pos = len
    then mem etat automate.finaux
    else
      if mem_assoc mot.[pos] automate.arcs.(etat)
      then calcul (assoc mot.[pos] automate.arcs.(etat)) (succ pos)
      else false in
  calcul automate.initial 0

let bien_parenthese_max_3_imbrications = {
  initial = 0;
  finaux = [0];
  arcs =
    [|
      [`(`,1]; (* niveau d'imbrication 0 *)
      [`(`,2;`)`,0]; (* niveau 1 *)
      [`(`,3;`)`,1]; (* niveau 2 *)
      [       `)`,2]; (* niveau 3 *)
    |]
};;

accepte bien_parenthese_max_3_imbrications "(()(())())";; (* true *)
accepte bien_parenthese_max_3_imbrications "(())()()";;   (* true *)
accepte bien_parenthese_max_3_imbrications "(((())))";;   (* false *)
accepte bien_parenthese_max_3_imbrications "())";;        (* false *)

let n_niveaux n =
  let arcs = init_vect (succ n) (fun i -> [`(`,succ i; `)`,pred i]) in
  arcs.(0) <- [`(`,1];
  arcs.(n) <- [`)`, pred n];
  {initial = 0;
   finaux = [0];
   arcs=arcs};;

accepte (n_niveaux 3) "(()((()))())";; (* true *)
accepte (n_niveaux 2) "(()((()))())";; (* false *)
accepte (n_niveaux 5) "((((()))))";;   (* true *)

type automate = {
  initial: int;
  sortie: int -> char;
  arcs: (char * int) list vect}

let calcul auto mot =
  let etat = ref auto.initial in
  for i=0 to pred (string_length mot) do
    etat := assoc mot.[i] auto.arcs.(!etat);
  done;
  auto.sortie !etat;;

let thue_morse = {
  initial = 0;
  sortie = (function 0 -> `0` | 1 -> `1` | _ -> failwith "thue_morse");
  arcs = [|[`0`,0;`1`,1]; [`0`,1;`1`,0]|]
};;

(* prévu pour une architecture 32 bits *)
let base_2 n =
  let result = create_string 31 in
  for i=0 to 30 do
    result.[30-i] <-
      (match n land (1 lsl i) with
        | 0 -> `0`
        | _ -> `1`)
  done;
  result;;
```

```
let begin_thue_morse n =
  let result = create_string n in
  for i=0 to pred n do
    result.[i] <- calcul thue_morse (base_2 i)
  done;
  result;;

begin_thue_morse 32;;
(* "01101001100101101001011001101001" *)


let print_char_list = do_list print_char;;

let shuffle u v =
  let len_u = string_length u in
  let len_v = string_length v in
  let rec un_char prefixe i j =
    if i < len_u then un_char (u.[i] :: prefixe) (succ i) j;
    if j < len_v then un_char (v.[j] :: prefixe) i (succ j);
    if i=len_u && j=len_v then (print_char_list (rev prefixe); print_newline ()) in
  un_char [] 0 0;;

shuffle "abc" "xyz";;


(* 4i, 1m, 1p, 4s, soit (4+1+1+4)! / (4! * 1! * 1! * 4!) = 6300 *)

let iteri f vect = for i=0 to vect_length vect -1 do f i vect.(i) done

let rec melanges prefixe chars =
  let nb_lettres = ref 0 in
  iteri
    (fun i (char,nb) ->
       if nb > 0 then (
         nb_lettres := nb + !nb_lettres;
         chars.(i) <- (char,nb-1);
         melanges (make_string 1 char ^ prefixe) chars;
         chars.(i) <- char,nb))
    chars;
  if !nb_lettres = 0 then (print_string prefixe; print_newline ())

(* Pour ceux qui veulent absolument l'écrire en fonctionnel
(on a en plus évité de générer, pour une liste l, la liste des couples
(élément e de l, l privée de e) car cela a un coût quadratique): *)
let rec melanges prefixe liste1 liste2 =
  let rec traite_un_char liste autre_liste = match liste with
    | [] -> ()
    | (char,nb) as elt :: suite_liste ->
        if nb=1
        then (
          melanges (make_string 1 char ^ prefixe) suite_liste autre_liste;
          traite_un_char suite_liste (elt::autre_liste))
        else (
          melanges (make_string 1 char ^ prefixe) ((char,nb-1) :: suite_liste) autre_liste;
          traite_un_char suite_liste (elt::autre_liste)) in
  match liste1,liste2 with
    | [],[] -> print_string prefixe; print_newline ()
    | [],_ -> traite_un_char liste2 []
    | _,[] -> traite_un_char liste1 []
    | _ ->
      traite_un_char liste1 liste2;
      traite_un_char liste2 liste1;;

let char_list_of_string mot =
  let result = ref [] in
  for i= pred (string_length mot) downto 0 do
    result := mot.[i] :: !result
  done;
  !result;;

char_list_of_string "bonjour";;

let compte liste =
  let rec fusionne = function
      [] -> []
    | [elt] -> [elt]
    | (char1,nb1) as elt1 :: ((char2,nb2) as elt2) :: q ->
        if char1 = char2
        then fusionne ((char1,nb1+nb2) :: q)
        else elt1 :: fusionne (elt2::q) in
  fusionne (map (fun c -> c,1) liste);;

compte [4;4;4;5;5;6;7;7;7;7];;
```

```
let anagrammes mot =
  let mot = char_list_of_string mot in
  let mot = sort__sort (prefix <) mot in
  let mot = compte mot in
(*   melanges "" mot [];; *)
  melanges "" (vect_of_list mot);;

anagrammes "aabb";;
anagrammes "hell";;
anagrammes "mississipi";;
```