1 Reconnaissance par un AFD

```
type etat = {final: bool; transitions: (char*int) list};;

type AFD = AFD of etat vect;;

let reconnait (AFD v) w =
    let delta q c = assoc c v.(q).transitions
    in let rec delta_barre q = function
           | [] -> q
           | t::r -> delta_barre (delta q t) r
in
try
    let dernier = delta_barre 0 w
    in v.(dernier).final
with Not_found -> false;;
```

2 Type des graphes

```
type graphe = G of ((int list) vect);;
```

3 Exploration en profondeur

```
let explore_en_profondeur (G graphe) depart =
        let n = vect_length graphe in
        let marque = make_vect n false in
        let rec visite i =
              if not marque.(i) then
                 begin
                    marque.(i) <- true;
                    do_list visite graphe.(i);
                  end
         in visite depart; marque;;
```

4 Exploration en largeur

```
let explore_en_largeur (G graphe) depart =
        let rec reunit = function
               | [] -> []
               | t::r -> union t (reunit r)
        in
        let ajoute_niveau liste =
               let nouveau = reunit (map (function i -> graphe.(i)) liste) in
               union liste nouveau
        in let rec itere l1 l2 =
               if list_length l1 = list_length l2
                   then l1
                   else itere l2 (ajoute_niveau l2)
        in itere [] [depart];;
```

5 Exploration à l'aide de piles

```
let explore (G graphe) depart = (* exploration en profondeur si pile LIFO,
                                   en largeur si pile FIFO *)
        let n = vect_length graphe in
        let marque = make_vect n false in
        let (empile,depile,non_vide) = new_pile () in
        let rec empile_liste = function
             | [] -> ()
             | t::r -> empile t; empile_liste r
        in
        empile depart;
        while non_vide () do
             let x = depile () in
             if not marque.(x) then
                begin
                  marque.(x) <- true;
                  empile_liste graphe.(x)
                 end
        done;
        marque;;
```

6 Emonder un graphe

```
let emonde (G graphe) coacc =
        (* emonde un graphe en fonction d'un ensemble coacc de sommets acceptants
        (sous forme de tableau de booléens) et d'un sommet initial 0. Modifie
        l'ensemble coacc pour ne garder que les sommets du graphe émondé *)
        let n = vect_length graphe in
        let marque = make_vect n false in
        let rec un_vrai = function (* cherche un true dans une liste de booléens *)
                | [] -> false
                | t :: r -> t || (un_vrai r)
        in let rec visite i = (* visite et retourne le booléen coaccessible *)
              if not marque.(i) then begin
                    marque.(i) <- true;
                    let l = map visite graphe.(i) in
                    coacc.(i) <- coacc.(i) || (un_vrai l)
                  end;
               coacc.(i)
         in
         coacc.(0) <- visite 0; (* pas très utile *)
         for i = 0 to (n-1) do coacc.(i) <- coacc.(i) && marque.(i) done;;
                                        (* accessible et coaccessible *)
```

7 Reconnaissance par un AFND

```
type etat = {final: bool; transitions: (char*int) list};;

type AFND = AFND of etat vect;;

let reconnait (AFND v) w =
    let rec reunit = function (* reunit une liste de listes *)
          | [] -> []
          | t::r -> union t (reunit r)
    and cherche_tous a  = function
                     (* recherche toutes les associations *)
          | [] -> []
          | (x,y)::r -> if x=a then y::(cherche_tous a r)
                else cherche_tous a r
    and cherche_bon = function (* recherche un état final *)
          |  [] -> false
          |  t::r -> v.(t).final || cherche_bon r
```

```
    in
    let delta partie c =
        let f = function
                  x -> cherche_tous c v.(x).transitions in
        reunit (map f partie)
    in let rec delta_barre partie = function
           | [] -> partie
           | t::r -> delta_barre (delta partie t) r
in
    let dernier = delta_barre [0] w
    in cherche_bon dernier;;
```

8 Reconnaissance par un AFNDε

```
type etat = {final: bool; transitions: (char*int) list; instantanees: int list};;

type AFNDe = AFNDe of etat vect;;

let reconnait (AFNDe v) w =
    let rec reunit = function (* reunit une liste de listes *)
          | [] -> []
          | t::r -> union t (reunit r)
    and cherche_tous a  = function (* recherche toutes les associations *)
          | [] -> []
          | (x,y)::r -> if x=a then y::(cherche_tous a r)
                else cherche_tous a r
    and cloture l =
       let rec ajoute_instantanees l =
              reunit (map (function x -> v.(x).instantanees)) l
       and aux l1 l2 =
           if list_length l1 = list_length l2 then l1
           else aux l2 (ajoute_instantanees l2)
       in aux [] l
    and cherche_bon = function (* recherche un état final *)
```

```
        |  [] -> false
        |  t::r -> v.(t).final || cherche_bon r
    in
    let delta partie c =
        let f = function
                  x -> cherche_tous c v.(x).transitions in
        cloture (reunit (map f partie))
    in let rec delta_barre partie = function
           | [] -> partie
           | t::r -> delta_barre (delta partie t) r
in
    let dernier = delta_barre (cloture [0]) w
    in cherche_bon dernier;;
```