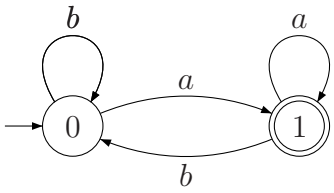
1 Simulation d'automates

1.

```
let execute auto mot=
  let n=string_length mot in
  let s = ref auto.initial in
  for i=0 to n-1 do s := auto.arcs !s mot.[i]
  done;
  auto.vals !s
;;
```

La référence `s` désigne l'état courant lors de le lecture du mot d'entrée `mot`. On part avec l'état initial et pour chaque lettre du mot, on change d'état grâce à la fonction de transition.

2. L'automate déterministe recherché est le suivant :




```
let auto1 = { initial = 0;
              arcs = ( fun
                       | 0 'b' -> 0
                       | 0 'a' -> 1
                       | 1 'b' -> 0
                       | 1 'a' -> 1
                       | _  _  -> failwith "transition non définie");
              vals = fun x -> x; };;
```

3.

```
let auto_dragon = {
         initial = 0;
         arcs = ( fun
                  | 0 '0' -> 0
                  | 0 '1' -> 1
                  | 1 '0' -> 2
                  | 1 '1' -> 3
                  | 2 '0' -> 2
                  | 2 '1' -> 2
                  | 3 '0' -> 3
                  | 3 '1' -> 3
                  | _  _  -> failwith "transition non définie");
         vals = fun
                  | 0 -> 0
                  | 1 -> 0
                  | 2 -> 0
```

```
                | 3 -> 1
                | _ -> failwith "valeur non définie"};;
```

4. La solution proposée est récursive. On utilise le fait que pour un entier n dont l'écriture binaire est $\overline{a_k \cdots a_1 a_0}^2$, a_0 est le reste de la division de de n par 2 (c'est à dire $n \mod 2$) et $\overline{a_k \cdots a_1}^2$ est l'écriture binaire de la partie entière de $n/2$.

```
let rec base2 = function
          0 -> "0"
        | 1 -> "1"
        | n -> (if (0=n mod 2) then "0" else "1") ^ (base2 (n/2));;
```

5.

```
let nieme_dragon n = execute auto_dragon (base2 n);;
```

6.

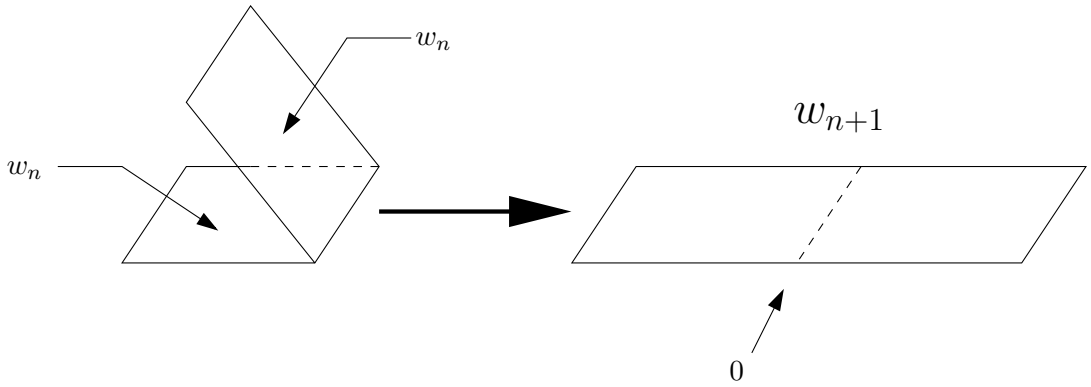
```
let dragon n =
  clear_graph() ;
  let dx = ref 0                  (* direction          *)
  and dy = ref 1                  (*           courante *)

  and x = ref 300                 (* position          *)
  and y = ref 300                 (*          courante *)

  and l = 4 in                    (* longueur d'un pas *)

  moveto !x !y;                   (* on se place au point de départ *)
  x := !x + !dx * l;
  y := !y + !dy * l;              (* on trace un trait *)
  lineto !x !y;

  for i=1 to n do
    match (nieme_dragon i) with
     |0 -> let temp = !dx in      (* virage à gauche *)
           dx := - !dy;
           dy := temp;
           x := !x + !dx * l;
           y := !y + !dy * l;
           lineto !x !y;
     |1 -> let temp = !dx in      (* virage à droite *)
           dx := !dy;
           dy := -temp;
           x := !x + !dx * l;
           y := !y + !dy * l;
           lineto !x !y;
     |_ -> failwith "cas impossible"
  done;;
```

2 Pliage d'une feuille de papier

7. A l'aide d'une bande de papier, on trouve :

$$\begin{aligned} w_3 &= 0010011 \\ w_4 &= 001001100011011 \end{aligned}$$

8. Pour obtenir w_{n+1}, il faut plier la bande en 2 puis effectuer les pliages de w_n sur la demi-bande obtenue. Lorsqu'on déplie la demi-bande, on obtient w_n, un 0 pour le pliage du milieu et enfin l'image miroir de $\overline{w_n}$, où $\overline{w_n}$ est l'inverse bit à bit de w_n.



Exemple :

$$\begin{aligned} w_3 &= 0010011 \\ \overline{w_3} &= 1101100 \\ \text{miroir}(\overline{w_3}) &= 0011011 \\ w_4 &= 001001100011011 \end{aligned}$$

Pour définir `mot_dragon`, il faut programmer la fonction `inverse` qui calcule l'inverse bit à bit d'un mot et la fonction `miroir` qui calcule l'image miroir d'un mot.

```
let inverse s =
  let n = string_length s in
  let m = create_string n in
  for i=0 to n-1 do
    match s.[i] with
      '0' -> m.[i] <- '1'
    |'1' -> m.[i] <- '0'
    | _  -> failwith "cas impossible"
  done;
  m;;

let miroir s =
  let n = string_length s in
  let m = create_string n in
```

```
  for i=0 to n-1 do
    m.[i] <- s.[n-1-i]
  done;
  m;;

let rec mot_dragon n =
  if n=1 then "0"
  else let m=(mot_dragon (n-1)) in m^"0"^(miroir (inverse m));;
```

Dans les fonctions `inverse` et `miroir`, il est important de **créer** une nouvelle chaîne pour initialiser `m`. Si on se contente d'initialiser `m` avec `let m=s in`, les résultats sont faussés car une modification de `m` entraîne une modification de `s` (les deux variables pointent vers la même chaîne de caractères).
Exemple :

```
#let s="david";;
s : string = "david"
#let m=s;;
m : string = "david"
#m.[0]<-'D';;
- : unit = ()
#m;;
- : string = "David"
#s;;
- : string = "David"
```

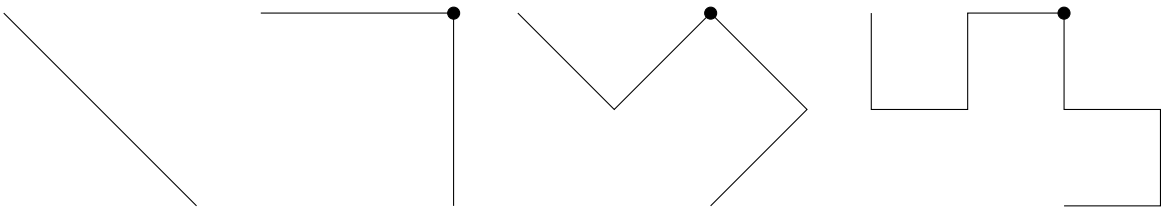
9.

```
let dragon n =
  clear_graph() ;
  let dx = ref 0
  and dy = ref 1
  and x = ref 300
  and y = ref 300
  and l = 4 in
  moveto !x !y;
  x := !x + !dx * l;
  y := !y + !dy * l;
  lineto !x !y;
  let m = mot_dragon n in
  let q = string_length m in
  for i=0 to (q-1) do         (* le seul changement se trouve ici *)
    match m.[i] with
     |'0' -> let temp = !dx in
           dx := - !dy;
           dy := temp;
           x := !x + !dx * l;
           y := !y + !dy * l;
           lineto !x !y;
```

```
    |'1' -> let temp = !dx in
           dx := !dy;
           dy := -temp;
           x := !x + !dx * l;
           y := !y + !dy * l;
           lineto !x !y;
    |_ -> failwith "cas impossible"
  done;;
```

3 Fonctions récursives

10. On peut vérifier sur les petites valeurs de n que la courbe vérifie bien la récurrence indiquée.

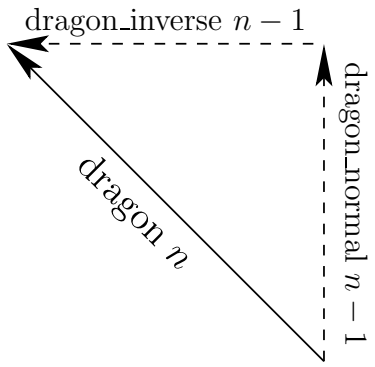


dragon 0 dragon 1 dragon 2 dragon 3

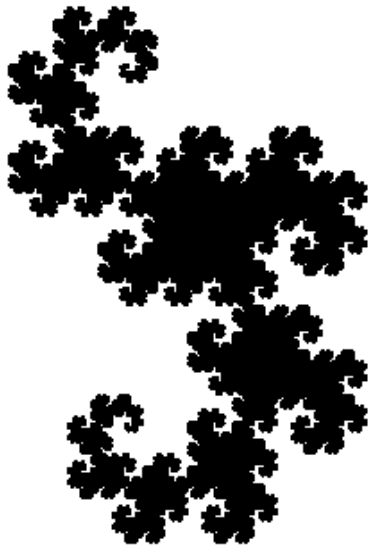
La commande `dragon_rec n (x1,y1) (x2,y2)` trace la courbe du dragon d'ordre n entre les points `(x1,y1)` et `(x2,y2)`. Pour améliorer la précision des dessins, on manipule les coordonnées courantes avec un type `float`.

```
let rec dragon_rec n (x1,y1) (x2,y2) =
  moveto (int_of_float x1) (int_of_float y1);
  if n=0 then lineto (int_of_float x2) (int_of_float y2)
  else begin
         let (x3,y3)=((x1+.x2+.y2-.y1)/.2.,(x1-.x2+.y1+.y2)/.2.) in
         dragon_rec (n-1) (x1,y1) (x3,y3);
         dragon_rec (n-1) (x2,y2) (x3,y3);
       end
  ;;

let dragon n =
  clear_graph();
  dragon_rec n (200.,150.) (200.,350.);;
```

11. Pour tracer la courbe sans lever le crayon, il faut deux fonctions : la première trace la courbe dragon classique et la deuxième trace le courbe dragon à l'envers.




```
let rec dragon_normal n (x1,y1) (x2,y2) =
  if n=0 then lineto (int_of_float x2) (int_of_float y2)
  else begin
         let (x3,y3)=((x1+.x2+.y2-.y1) /. 2. , (x1-.x2+.y1+.y2) /. 2.) in
         dragon_normal (n-1) (x1,y1) (x3,y3);
         dragon_inverse (n-1) (x3,y3) (x2,y2);
       end
 and dragon_inverse n (x1,y1) (x2,y2) =
  if n=0 then lineto (int_of_float x2) (int_of_float y2)
  else begin
         let (x3,y3)=((x1+.x2-.y2+.y1) /. 2. , (x2-.x1+.y1+.y2) /. 2.) in
         dragon_normal (n-1) (x1,y1) (x3,y3);
         dragon_inverse (n-1) (x3,y3) (x2,y2);
       end
;;

let dragon n =
  clear_graph();
  moveto 200 150;
  dragon_normal n (200.,150.) (200.,350.);;
```



dragon 20