Programmation Fonctionnelle
Cours 8

Michele Pagani

Université Paris Diderot
UFR Informatique
Institut de Recherche en Informatique Fondamentale

pagani@irif.fr

14 novembre 2016

Graphisme

La bibliothèque `graphics.cma`

- une bibliothèque très rudimentaire de fonctionnalités graphiques
  - pour GUI plus avancés voir: `http://lablgtk.forge.ocamlcore.org`

- Le top-level n'a par défaut pas les fonctionnalités graphiques. Il y a plusieurs possibilités :
  - Charger bibliothèque dans interpréteur : `#load "graphics.cma";;`
  - Inclure bibliothèque au moment du lancement interpréteur : `ocaml graphics.cma`
  - Créer une nouvelle instance interpréteur (voir le manuel): `ocamlmktop -o mytop graphics.cma`

- Pour compiler un programme qui utilise le graphisme : `ocamlc` *other options* `graphics.cma` *other files*

La bibliothèque `graphics.cma`

- une bibliothèque très rudimentaire de fonctionnalités graphiques
  - pour GUI plus avancés voir: `http://lablgtk.forge.ocamlcore.org`

- Le top-level n'a par défaut pas les fonctionnalités graphiques. Il y a plusieurs possibilités :
  - Charger bibliothèque dans interpréteur : `#load "graphics.cma";;`
  - Inclure bibliothèque au moment du lancement interpréteur : `ocaml graphics.cma`
  - Créer une nouvelle instance interpréteur (voir le manuel): `ocamlmktop -o mytop graphics.cma`

- Pour compiler un programme qui utilise le graphisme : `ocamlc` *other options* `graphics.cma` *other files*

La bibliothèque `graphics.cma`

- une bibliothèque très rudimentaire de fonctionnalités graphiques
  - pour GUI plus avancés voir:
    `http://lablgtk.forge.ocamlcore.org`
- Le top-level n'a par défaut pas les fonctionnalités graphiques. Il y a plusieurs possibilités :
  - Charger bibliothèque dans interpréteur :
    `#load "graphics.cma";;`
  - Inclure bibliothèque au moment du lancement interpréteur :
    `ocaml graphics.cma`
  - Créer une nouvelle instance interpréteur (voir le manuel):
    `ocamlmktop -o mytop graphics.cma`
- Pour compiler un programme qui utilise le graphisme :
  `ocamlc` *other options* `graphics.cma` *other files*

La bibliothèque `graphics.cma`

- une bibliothèque très rudimentaire de fonctionnalités graphiques
  - pour GUI plus avancés voir:
    `http://lablgtk.forge.ocamlcore.org`

- Le top-level n'a par défaut pas les fonctionnalités graphiques. Il y a plusieurs possibilités :

  - Charger bibliothèque dans interpréteur :
    `#load "graphics.cma";;`
  - Inclure bibliothèque au moment du lancement interpréteur :
    `ocaml graphics.cma`
  - Créer une nouvelle instance interpréteur (voir le manuel):
    `ocamlmktop -o mytop graphics.cma`

- Pour compiler un programme qui utilise le graphisme :
  `ocamlc` *other options* `graphics.cma` *other files*

La bibliothèque `graphics.cma`

- une bibliothèque très rudimentaire de fonctionnalités graphiques
  - pour GUI plus avancés voir: `http://lablgtk.forge.ocamlcore.org`
- Le top-level n'a par défaut pas les fonctionnalités graphiques. Il y a plusieurs possibilités :
  - Charger bibliothèque dans interpréteur : `#load "graphics.cma";;`
  - Inclure bibliothèque au moment du lancement interpréteur : `ocaml graphics.cma`
  - Créer une nouvelle instance interpréteur (voir le manuel): `ocamlmktop -o mytop graphics.cma`
- Pour compiler un programme qui utilise le graphisme : `ocamlc` *other options* `graphics.cma` *other files*

La bibliothèque `graphics.cma`

- une bibliothèque très rudimentaire de fonctionnalités graphiques
  - pour GUI plus avancés voir: `http://lablgtk.forge.ocamlcore.org`
- Le top-level n'a par défaut pas les fonctionnalités graphiques. Il y a plusieurs possibilités :
  - Charger bibliothèque dans interpréteur : `#load "graphics.cma";;`
  - Inclure bibliothèque au moment du lancement interpréteur : `ocaml graphics.cma`
  - Créer une nouvelle instance interpréteur (voir le manuel): `ocamlmktop -o mytop graphics.cma`
- Pour compiler un programme qui utilise le graphisme : `ocamlc` *other options* `graphics.cma` *other files*

La fenêtre graphique

(`open Graphics;;` plus besoin de la notation pointée pour cette bibliothèque)

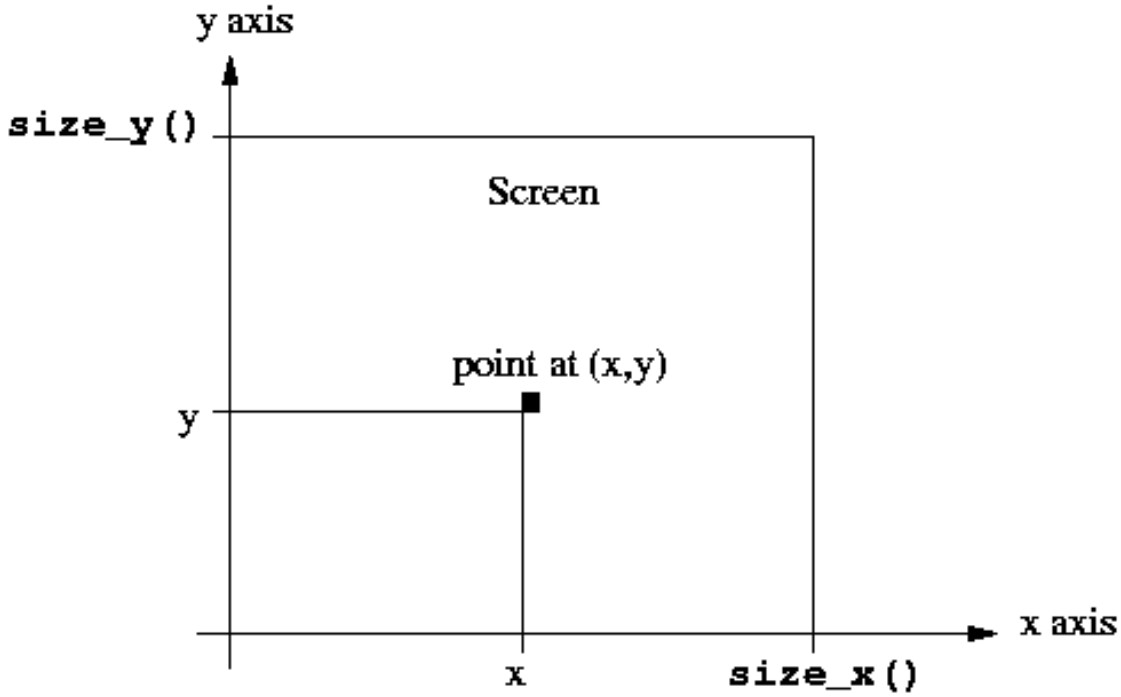
- `open_graph " lxh"`: crée fenêtre graphique, où *l* et *h* sont le nombre de pixels pour la largeur (l) et hauteur (h), e.g.:

  `open_graph " 600x400"`

  - attention espace début argument
    (obligatoire pour UNIX, voir manuel pour Windows)
  - on peut avoir une seule fenêtre graphique

- `close_graph:unit->unit` ferme fenêtre graphique
- `clear_graph:unit->unit` efface contenu fenêtre graphique
- `set_window_title:string->unit` donne un titre à la fenêtre

Coordonnés sur le canevas graphique



L'origine (0,0) est en bas à gauche

Dessiner

- Il y a un curseur, qui au début se trouve à l'origine (0,0).
- `current_point:unit->int*int` renvoie position du curseur
- `moveto x y` positionne curseur en (x, y)
- `plot x y` dessine point à position (x, y) et positionne curseur en (x, y)
- `lineto x y` dessine une ligne de position actuelle curseur à (x, y), et positionne le curseur en (x, y)
- `set_line_width n` sélectionne n pixels comme épaisseur lignes
- `draw_char c` affiche caractère c à position curseur
- `draw_string s` affiche chaîne s à position curseur

Exercice (lignes)

- Définir deux fonctions `horizontal: int -> unit` et `vertical: int -> unit` qui étant donné un entier `n`, dessine sur la fenêtre graphique `n` lignes horizontales (respectivement verticales) équidistantes.

Solution (lignes)

```
1  open Graphics;;
2
3  open_graph " 600x400" ;;
4
5  (* orizontal lines *)
6    let horizontal n =
7      for i = 1 to n do
8        let space = (size_y ()) / n in
9        let y = space*i in
10       moveto 0 y ;
11       lineto (size_x ()) y
12     done
13   ;;
14
15 (* vertical lines *)
16   let vertical n =
17     let space = (size_x ()) / n in
18     for i = 1 to n do
19       let x = space*i in
20       moveto x 0 ;
21       lineto x (size_y ())
22     done
23   ;;
```

Polygones et courbes

- `draw_rect x y l h` dessine un rectangle avec vertex bas gauche en (x, y), largeur l, hauteur h
- `draw_poly_line:(int * int) array->unit` dessine une ligne qui joint les points donnés dans le tableau
- `draw_poly:(int * int) array->unit` dessine le polygone (ligne fermé) qui joint les points donnés dans le tableau
- `draw_circle x y r` dessine un cercle de centre (x, y) et rayon r
- `draw_ellipse x y rx ry` dessine ellipse de centre (x, y), rayon horizontal rx et vertical ry
- `draw_arc x y rx ry a1 a2` dessine arc elliptique de centre (x, y), rayon horizontal rx et vertical ry, entre angles $a1$ et $a2$ (en dégrées)

Couleurs

- `color` un type représentant les couleurs
- constantes prédéfinies de `color`:

  `black`, `white`, `red`, `green`, `blue`, `yellow`, `cyan`, `magenta`

- `rgb r v b` renvoie la couleur (type `color`) avec composantes rouge r, verte v et bleue b.
  Les valeurs légales pour arguments: de 0 à 255.
- `set_color c` sélectionne c comme la couleur courante
- Function `fill_XXX` : remplir le polygone XXX dans la couleur courante:

  `fill_rect`, `fill_poly`, `fill_arc`, `fill_ellipse`, `fill_circle`

Exercice (ellipse colorée)

- Définir une fonction `random_ellipse: int -> int -> Graphics.color list -> unit` qui, étant donnés deux entiers `x`, `y` et une liste de couleurs `liste` dessine une matrice $\texttt{x} \times \texttt{y}$ d'ellipses équidistantes, chacune d'un couleur choisi aléatoirement entre les couleurs de la `liste`.

Solution (ellipse colorée)

```
1  open Graphics;;
2
3  open_graph " 400x600";;
4
5  (* pick a random color *)
6  let random_color list=
7    let _ = Random.self_init () in
8    let l = List.length list in
9    let c = List.nth list (Random.int l) in
10   set_color c
11 ;;
12
13 (* matrix of n x m ellipses with equal distance *)
14 let random_ellipse n m list =
15   let dx = (size_x () / n) in
16   let dy = (size_y () / m) in
17   for x = 0 to n-1 do
18     for y = 0 to m-1 do
19       random_color list ;
20       fill_ellipse (dx/2+dx*x) (dy/2+dy*y) (dx/2-5) (dy/2-5)
21     done
22   done
23 ;;
24
25 let colors = [black;white;red;green;blue;yellow;cyan;magenta]
26 ;;
```

Interaction avec l'utilisateur

Le type event

```
type event =
  Button_down | Button_up | Key_pressed | Mouse_motion
```

- Un événement se produit quand l'utilisateur clique sur un bouton de la souris, déplace la souris ou presse une touche du clavier. Le type `event` contient les formes différentes des événements.

- `wait_next_event:event list -> status`
  prend comme argument une liste *l* d'événements et attend le prochain événement appartenant à la liste *l* (les autres événements seront ignorés). Quand le premier événement se produit une description détaillée est renvoyée, du type `status`.

Le type event

```
type event =
  Button_down | Button_up | Key_pressed | Mouse_motion
```

- Un événement se produit quand l'utilisateur clique sur un bouton de la souris, déplace la souris ou presse une touche du clavier. Le type `event` contient les formes différentes des événements.

- `wait_next_event:event list -> status`
  prend comme argument une liste *l* d'événements et attend le prochain événement appartenant à la liste *l* (les autres événements seront ignorés). Quand le premier événement se produit une description détaillée est renvoyée, du type `status`.

Le type status

```
type status =
{
 mouse_x    : int;   (*coordonnee x de la souris          *)
 mouse_y    : int;   (*coordonnee y de la souris          *)
 button     : bool;  (*bouton de la souris est enfonce ?*)
 keypressed : bool;  (*touche clavier a ete pressee ?    *)
 key        : char;  (*touche pressee clavier si le cas *)
}
```

- Remarque : il n'y a aucune distinction entre les boutons différents de la souris.

Exemple (paint.ml)

```
1  open Graphics;;
2  open_graph " 500x500";;
3  exception Quit;;
4  let rec loop t =
5    let eve = wait_next_event [Mouse_motion;Key_pressed]
6    in
7    if eve.keypressed
8    then
9      match eve.key with
10       | 'b' -> set_color black; loop t
11       | 'r' -> set_color red; loop t
12       | 'g' -> set_color green; loop t
13       | 'q' -> raise Quit
14       | '0'..'9' as x -> loop (int_of_string (String.make 1 x))
15       | _ -> loop t
16   else begin
17     fill_circle (eve.mouse_x-t/2) (eve.mouse_y-t/2) t;
18     loop t
19   end
20 in
21 try loop 5
22 with Quit -> close_graph ();;
```

Exercice (ellipses avec souris)

- Écrire une fonction qui, étant donné en entrée deux entiers `x` et `y`, dessine une grille de `x` colonnes et `y` lignes et permets de placer des ellipses au centre des cases de la grille en utilisant la souris.

Solution (ellipses avec souris)

```
1  open Graphics;;
2  open_graph " 400x600";;
3
4  let main x y =
5    (* draw grid using previous defined fonctions *)
6    vertical x;
7    horizontal y;
8    (*compute radius (i.e. axis/2)*)
9    let ax = size_x()/(2*x) in
10   let ay = size_y()/(2*y) in
11   let rec loop () =
12     (*wait for mouse button*)
13     let eve = wait_next_event [Button_down;Key_pressed] in
14     if eve.keypressed then ()
15     else
16       begin
17         (*compute center coordinates*)
18         let cx = ((eve.mouse_x/(2*ax))*2*ax + ax) in
19         let cy = ((eve.mouse_y/(2*ay))*2*ay + ay) in
20         fill_ellipse cx cy ax ay;
21         loop ()
22       end
23   in loop ()
24 ;;
25
```

Double-tampon

- par défaut, la fenêtre graphique a une mémoire tampon, appelée backing store, synchronisée avec la fenêtre graphique à la fin de l'exécution de chaque instruction de dessin
- pour un affichage plus rapide, on peut désactiver l'affichage atuomatique sur le fenêtre, travailler sur le tampon et afficher son contenu seulement quand le dessin est terminé:
  - `auto_synchronize : bool -> unit`
    `auto_synchronize false` désactive la synchronisation automatique entre la memoire tampon et la fenêtre graphique (par defaut `auto_synchronize true`).
  - `synchronize : unit -> unit`
    `synchronize ()` synchronise manuellement la memoire tampon et la fenêtre graphique

Example (synchronize)

```
1  open Graphics;;
2
3  let main_double () =
4    open_graph " 600x700";
5    set_color black;
6    for i = 0 to 599 do
7      for j = 0 to 699 do
8        plot i j
9      done
10   done
11
12 let main_single () =
13   open_graph " 600x700";
14   auto_synchronize false; (* Manual double buffering, much faster.
15   set_color black;
16   for i = 0 to 599 do
17     for j = 0 to 699 do
18       plot i j
19     done
20   done;
21 synchronize () ; (* We commit the drawing *)
```