

## Feuille 4 : Application : zones géométriques

**Exercice 4.1** Dans un plan muni d'un repère orthonormé, on associe au point  $P$  de coordonnées  $(x, y)$  son affixe, le nombre complexe  $z = x + iy$ .

Pour représenter un point du plan, on utilise le type

```
type mycomplex = C of float * float
```

1. Télécharger et désarchivez `zones.zip` dans le répertoire habituel.
2. Se placer dans le répertoire `Zones` ainsi obtenu.
3. Ouvrir le fichier `mycomplex.ml`.
4. Écrire l'accessor `realpart` qui permet de récupérer la partie réelle d'un complexe.
5. Écrire l'accessor `imagpart` qui permet de récupérer la partie imaginaire d'un complexe.
6. Écrire le constructeur `make_complex` qui fabrique un complexe à partir de sa partie réelle et de sa partie imaginaire
7. Définir une variable `c_origin` contenant le point de coordonnées  $(0, 0)$ .
8. Définir une variable `p12` contenant le point  $(1, 2)$ .
9. Implémenter les opérations `c_sum c1 c2`, `c_dif c1 c2`, `c_mul c1 c2`, `c_abs c`, `c_scal lambda c`, `c_exp c` qui permettent respectivement de calculer la valeur absolue d'un complexe, la somme, la différence, le produit de deux complexes, la multiplication d'un complexe par un scalaire (float), l'exponentielle complexe.

**Exercice 4.2** Une transformation  $F$  du plan transforme tout point  $P$  en son image  $P' = F(P)$ . On peut décrire la transformation  $f$  par la fonction  $f$  qui appliquée à  $z$  l'affixe de  $P$  donne  $z'$  l'affixe de  $P'$ .

$$\begin{aligned} f: \mathbb{C} &\rightarrow \mathbb{C} \\ z &\mapsto z' = f(z) \end{aligned}$$

Soit  $O$  le point origine du plan, c'est-à-dire le point d'affixe 0.

1. Soit la translation de vecteur  $\vec{u}$  d'affixe  $z_u$ . Donner l'affixe  $z'$  du transformé d'un point d'affixe  $z$  par cette translation.
2. Donner l'affixe  $z'$  du transformé d'un point d'affixe  $z$  par la rotation de centre  $O$  et d'angle  $\theta$ .
3. Donner l'affixe  $z'$  du transformé d'un point d'affixe  $z$  par la rotation de centre  $C$  d'affixe  $z_c$  et d'angle  $\theta$ ,

**Exercice 4.3** <sup>a</sup>

Dans leur article "Haskell vs. Ada vs. C++ vs. Awk vs. ... An Experiment in Software Prototyping Productivity" <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.368.1058&rep=rep1&type=pdf> de 1994, Paul Hudak et Mark P. Jones rapportent une expérience rare de comparaison entre différents langages de programmation. Lelogiciel implémenté manipule des zones géométriques. Nous

allons étudier une version simplifiée (mais pas tant que ça) de ce logiciel.

Une zone géométrique est un ensemble de points du plan. On représente une telle zone par sa *fonction caractéristique*, c'est-à-dire par la fonction booléenne qui prend un point en argument, et retourne `true` si le point appartient à la zone et `false` sinon. Ainsi, une zone vide est représentée par la variable `nowhere` suivante :

```
# let nowhere = fun point -> false;;
val nowhere : 'a -> bool = <fun>
```

1. Définir une variable `everywhere` représentant la zone correspondant au plan tout entier.
2. Comprendre le code de la fonction `point_in_zone_p`.

Exemples :

```
# point_in_zone_p c_origin nowhere;;
- : bool = false
# point_in_zone_p c_origin everywhere;;
- : bool = true
```

```
# let point_in_zone_p point zone = zone point;;
val point_in_zone_p : 'a -> ('a -> 'b) -> 'b = <fun>
```

Pour forcer les types :

```
type zone = mycomplex -> bool

let point_in_zone (point : mycomplex) (zone : zone) =
  zone point;;
val point_in_zone : mycomplex -> zone -> bool = <fun>
```

<sup>a</sup>Pour passer à la partie `zones`, il est nécessaire que le fichier `mycomplex.ml` soit correct.

- Exercice 4.4**
1. Écrire une fonction `make_rectangle width height` qui retourne la zone rectangulaire définie par les points  $(0, 0)$ ,  $(width, 0)$ ,  $(width, height)$ ,  $(0, height)$ , c'est-à-dire la fonction qui prend un point en argument et retourne `true` si cet argument est dans le rectangle (ou sur sa bordure), et `false` sinon.
  2. L'appel `zone_intersection zone1 zone2` retourne la zone correspondant aux points se trouvant à la fois dans les zones `zone1` et `zone2`. Comprendre le code donné pour cette fonction.
  3. Écrire une fonction `zone_union zone1 zone2` qui retourne l'union des zones `zone1` et `zone2` (c'est-à-dire l'ensemble des points qui appartiennent à l'une ou l'autre des zones).
  4. Écrire une fonction `zone_complement zone1` qui retourne la zone correspondant aux points ne se trouvant pas dans la zone `zone1`.

- Exercice 4.5**
1. Comprendre le code de la fonction `make_disk0 radius` qui retourne un disque de rayon `radius` centré au point  $(0, 0)$ .
  2. Dans un repère orthonormé, dessiner la zone définie par l'expression suivante.  
`zone_union (make_rectangle 2. 4.) (make_disk0 3.)`

- Exercice 4.6**
1. S'inspirer de la fonction `move_zone` pour rajouter une fonction `scale_zone0`. Cette nouvelle fonction aura deux paramètres, le premier est la zone à transformer, et le deuxième est le point

(complexe)  $(\lambda_1, \lambda_2)$ . La fonction applique à la zone la transformation

$$(x, y) \mapsto (\lambda_1 x, \lambda_2 y).$$

2. Écrire une fonction `scale_zone zone coeff point` qui effectue la mise à l'échelle par rapport au point `point` et non pas par rapport à l'origine.

**Exercice 4.7** Écrire une fonction `make_disk radius point` qui retourne un disque de rayon `radius` centré au point `point`.

#### Exercice 4.8

```
let test =
  let c = make_disk0 1. in
  let c1 = move_zone c (C(1., 0.)) in
  assert (point_in_zone_p (C(0.0, 0.5)) c);
  assert (not (point_in_zone_p (C(1.0, 0.5)) c));
  assert (point_in_zone_p (C(0.5, 0.)) (zone_intersection c c1));;
```

En vous inspirant du jeu de tests ci-dessus, écrire un jeu de test pour toutes les fonctions.

**Exercice 4.9** 1. Regarder le contenu du fichier `visu-zones.ml`.

2. Inclure ce fichier à l'aide de la requête `#use "visu-zones.ml"`.
3. Utiliser la fonction `view_zone zone` pour produire une image PNG et visualiser une partie finie d'une zone.
4. Pour adapter la taille de la zone visualisée, utiliser `view_zone_size zone size`.

**Exercice 4.10** 1. Écrire une fonction `rotate_zone0 zone angle` qui effectue une rotation d'angle `angle` autour de l'origine.

2. Écrire une fonction `rotate_zone zone angle point` qui effectue une rotation d'angle `angle` autour du point `point`.