Programmation fonctionnelle 2023-2024

Feuille 5 : Application : zones géométriques

Exercice 5.1 [a]

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, la zone correspondant qu plan tout entier est représentée par la variable `everywhere` suivante :

```
# let everywhere = fun point -> true;;
val everywhere : 'a -> bool = <fun>
```

1. Éditer le fichier `zones.ml`.
2. Comprendre le code de la fonction `point_in_zone_p`.
3. Comprendre le code de la fonction `everywhere`.
4. Lancer `utop` et visualiser la zone `everywhere`.
5. Définir la zone vide dans la variable `nowhere`.

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

[a] Pour passer à la partie `zones`, il est nécessaire que le fichier `mycomplex.ml` soit correct.

Exercice 5.2 1. L'appel `translate_zone zone vector` retourne la zone translatée par le vecteur `vector` (représenté comme un point). Comprendre le code donné pour cette fonction.

2. 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.

3. 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.
4. 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).
5. Écrire une fonction `zone_complement zone1` qui retourne la zone correspondant aux points ne se trouvant pas dans la zone `zone1`.
6. Écrire une fonction `zone_difference zone1 zone2` qui retourne la zone correspondant aux points se trouvant dans la zone `zone1` mais pas dans la zone `zone2`.

Exercice 5.3 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 5.4 1. S'inspirer de la fonction `translate_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) (λ_1, λ_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 5.5 Écrire une fonction `make_disk radius point` qui retourne un disque de rayon `radius` centré au point `point`.

Exercice 5.6

```
let test () = fun _ ->
  begin
  let c = make_disk0 1. in
  let c1 = translate_zone c (make_point 1. 0.) in
  assert (point_in_zone_p (make_point 0.0, 0.5) c);
  assert (not (point_in_zone_p (make_point 1.0, 0.5) c));
  assert (point_in_zone_p (make_point 0.5, 0.) (zone_intersection c c1));
  end

let _ = test ()
```

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

Exercice 5.7 1. Regarder le contenu du fichier `visu-zones.ml`.

2. Utiliser la fonction `view_zone zone` pour produire une image PNG et visualiser une partie finie d'une zone.
3. Pour adapter la taille de la zone visualisée, utiliser `view_zone_size zone size`.

Exercice 5.8 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`.