Rapport de projet :

Makhloufi Khalil TP3

I- Notice d'utilisation :

Le dossier du projet est constitué de deux fichiers source : intervalles.ml et arbres_intervalles.ml.

1- intervalles.ml :

Ce premier implémente la notion d'intervalles en Ocaml, il nous permet de construire un module modélisant cette notion, ainsi que les opérations nécessaires pour les manipuler.

Le module prend en charge nativement les deux types `EntierOrdreNaturel` et `Character`, représentant respectivement `int` et `char`, si d'autres types veulent être utilisés pour la construction d'intervalles ils doivent nécessairement être codés et implémenter `TypeOrdonne`.

Pour se servir du module, nous utiliserons la ligne de code suivante :

```
module Nom_du_module = MakeIntervalles(type_souhaité)
```

Nous pourrons ainsi utiliser toutes les opérations de la bibliothèque en préfixant chaque opération par le nom du module.Nous retrouvons parmi les opérations du module les suivantes :

- *inter :*

```
val inter : element -> bool -> element -> bool -> inter option
```

Permet de construire un intervalle, prends en entrée 4 valeurs, 2 du type donné en entrée lors de la construction qui représentant les bornes de ce dernier, et deux booléens qui indiquent si oui ou non l'intervalle est fermé à gauche ou à droite.

- *comp :*

```
val comp : inter -> inter -> int
```

Permet de comparer deux intervalles selon l'ordre du type donné en entrée, renvoie 1 si le premier est plus grand que le deuxième, 0 si ils sont égaux, et -1 dans le cas échéant.

- *est_dans_intervalle :*

```
val est_dans_intervalle : element -> inter -> bool
```

Indique si une valeur est présente dans un intervalle

- *sont_disjoints :*

```
val sont_disjoints : inter -> inter -> bool
```

Indique si deux intervalles donnés en entrée sont disjoints

- *union :*

```
val union : inter -> inter -> (inter * inter, inter) Either.t
```

Calcule l'union de deux intervalles.

- *difference:*

```
val difference : inter -> inter -> (inter * inter, inter) Either.t
```

Calcule la différence entre deux intervalles (intervalle 1 privé de l'intervalle 2)

- *est_valide_inter:*

```
val est_valide_inter : unit -> unit
```

Série de calculs qui vérifient le bon fonctionnement de la bibliothèque.

2-arbres_intervalles.ml :

Cette deuxième bibliothèque implémente la notion d'arbres d'intervalles, elle se base sur une structure d'arbres bicolores.

Ce module utilise la bibliothèque Intervalles décrite précédemment, et donc prends en charge primitivement les mêmes type que Intervalles.

Pour se servir de la bibliothèque, nous utiliserons la ligne de code suivante :

```
Module Nom_du_module=MakeArbresIntervalles(type_souhaité)
```

Les opérations prises en charge pas le module sont les suivantes :

- *est_dans_arbre_intervalles :*

```
val est_dans_arbre_intervalle : element -> ab -> bool
```

. Indique si un élément est dans un intervalle de l'arbre donné en entrée.

- *ajout:*

```
val ajout : inter -> ab -> ab
```

Ajoute un intervalle dans un arbre bicolore tout deux donnés en entrées

- *retrait :*

```
val retrait : inter -> ab -> ab
```

Supprime un intervalle d'un arbre bicolore tout deux donnés en entrée.

- *valide_arbre_inter:*

```
val valide_arbre_inter : unit -> unit
```

Série de calculs qui vérifient le bon fonctionnement de la bibliothèque.

III - Difficultés rencontrées et explication des choix d'implémentation:

Lors de la réalisation du projet, j'ai rencontré plusieurs difficultés, parmi les premières était l'utilisation des foncteurs pour écrire mes bibliothèques, outil très pratique mais assez compliqué à comprendre aux premiers abords.

L'écriture des deux fonctions ensemblistes d'union et de différence était assez compliquée aussi, vu le nombre de cas à traiter, surtout ayant essayé de renvoyer le résultat le plus propre possible, par exemple le cas : [1,4] U [5,6] -> [1,6] qui a nécessité l'ajout d'une fonction pour les types `add_one : t -> t` qui incrémente de 1 les valeurs.

Les fonctions de vérification des bibliothèques étaient plutôt compliquées aussi, elles nécessitaient l'écriture de fonction de génération aléatoire pour chaque type pris en charge.

J'ai aussi rencontré pas mal se soucis quant à l'utilisation du module `Either`, au début juste pour m'en servir, mais par la suite pour récupérer les informations retournées par `union` et `difference`, faire du pattern matching avec des valeurs de type `Either` nécessite de faire plusieurs tests, et je me retrouvais souvent avec l'erreur ' `Option is None`' suite à l'appel de Option.get.