Projet type de données

Projet réalisé dans le cadre de l'UE types de données par

- Couderc-Lafont Enguerran
- Bains Anaïs (DLMI)
- Lantrade Morgan (DLMI)

Sommaire

Typages

Nous avons réalisé la fonction tp_fundefn, qui vérifie les déclarations de paramètres et de variables locales et génère un environnement qui permet de typer le corps de la fonction (statement).

Les difficultés rencontrées :

- Gérer les exceptions
- Tester nos fonctions au fur et à mesure

Les optimisations possibles :

- Certaines fonctions auxiliaires ne sont optimisées niveau complexité car nous avons privilégié de travailler sur la suite du projet.

Pour tester nos fonctions de typage, nous avons crée un fichier test_typing.ml où l'on test au cas par cas les différentes erreurs possibles.

Exécution de l'assembleur

Tableau d'instructions

Nous avons réalisé la fonction exec_instr avec, tout d'abord, des jumps relatifs et des variables indexées mais par la suite nous l'avons modifié suite aux consignes de la partie : **génération de l'assembleur**.

Les difficultés rencontrées :

- Une fois les fonctions implémentées, nous ne pouvions pas encore les tester.

Génération de l'assembleur

Nous avons réalisé dans **gen.ml** la fonction gen_instr qui retourne une liste d'instructions à jumps rélatifs et variables nommées.

Par la suite dans **instrs.ml**, nous avons traduis cette liste d'instructions en une liste d'instructions à jumps absolus et variables indexées.

Pour ce faire, nous avons tout d'abord transformé les variables nommées par des variables indexées à l'aide d'une liste d'association **(instr_var*int list)**, où l'entier correspond à l'IIndex. Enfin nous avons transformé les jumps relatifs en absolue à l'aide de **List.mapi** .

Tableau de variables

Dans **instrs.ml**, nous avons initialisé le tableau de variables avec une fois de plus la liste d'association, en s'aidant de sa longueur.

Dans un second temps, nous modifions celui-ci à l'aide de la liste de paramètres après avoir vérifier que le nombre d'arguments correspont aux nombres de paramètres de la fonction.

Les difficultés rencontrées

Lorsqu'un résultat était incorrect, il était très difficile de trouver où se situait l'erreur. Nous avons utilisé la fonction **trace** de ocaml, mais l'affichage dans la console, étant lourd, nous a fait perdre du temps.

Les filtrages **IfThenElse**, **Cond** et **While** ont été particulièrement périlleux à determiner avec le peu d'exemples fournis.

Conclusion

Nous avons crée et testé la fonction run_test dans **use.ml** car n'ayant pas de fichier **interf.ml** compilable, nous ne pouvions pas utilisé le fichier **comp.ml**.

Nous l'avons testé sur plusieurs fichiers test et avons eu des résultats cohérents. En particulier sur le fichier "test_final.c", contenant tous les filtrages que l'on puisse rencontrer.

Nous avons, par soucis de temps, décidé de ne pas s'occuper du graphe de flots.

Difficultés générales

L'environnement de travail(dans une simple terminal), le nombre de fichiers et notre connaissance en Ocaml, ne nous a pas permis de travailler dans des conditions idéales.

La dépendance des différentes parties, nous a empêché de diviser le travail convenablement et avons passé la majeure partie à coder à trois.

Optimisations possibles

Les fonctions auxiliaires ne sont pas toujours optimisées et nous n'avons pas réussi à pleinement utiliser les fonctions propre à Ocaml telles que **List.fold_right** ou **List.map**

Rendu

Le projet zippé rendu à déja été compilé via **Makefile**.

Exemple de test dans **/Test**

- Code

```
int testFinal (bool parserFini, bool grapheFini, bool typageFini) {
  int gen;
  int exe;
  int res;
  gen=3;
  exe=1;
  /* res = 5 */
  res= (gen > exe ? 5:25);
  while (typageFini != parserFini){
    /* res +=3 */
    res = res+gen;
    /* res= 8 puis 10 puis 12*/
    if (res==12) {
      parserFini=true;
      gen=3;
    }else{
      res=res-1;
    }
  }
  /* 24*2-6*1 = 18 */
  return ( (res*2) - 6*( grapheFini ? 0:1) ) ;
}
```

- Appel dans **use.ml**

```
let params = [0;0;1];;
let filename = "Tests/test_final.c";;
let test= run_test (parse filename) params;;
```

- Resultat

```
# # # # # # # # # # #   * * * * * *   # # # # #
#         val params : int list = [0; 0; 1]
# val filename : string = "Tests/test_final.c"
# val test : int = 18
# * * * * * * * * * * * * * * * * * * * * * * *
```