Rapport Projet Compilation - Taif Aabo-Aljaloo et João Fernandes Lopes

Repartition des tâches

- Taif Aabo-Aljaloo : Parser,Lexer,Simplification,analyse des types,extensions,programmes de test.
- João Fernandes Lopes : Parser,Lexer.

Les choix de conception

Pour ce projet nous avons décidé de rendre un compilateur qui respecte les conditions obligatoires. En revanche nous avons décidé de faire quelques extensions pour rendre le compilateur plus complet, par exmples:

Systeme ternaire

Pour utiliser le systeme ternaire il faut utiliser le mot clé `?` et `:` pour séparer les deux valeurs, ainsi que la condition. Exemple:

```
Int(x,1<5?1:0)
```

dans cet exemple la valeur de x sera 1 si la condition est vraie et 0 si elle est fausse. Pour implenter cette extension nous avons ajouté deux tokens `?` et `:` dans le lexer et nous avons ajouté une règle dans le parser qui permet de faire une expression ternaire. De plus l'AST a été modifié pour pouvoir prendre en compte cette extension. Pour l'interpréteur et le simplificateur je traite le systeme ternaire comme un IfThenElse. Enfin pour l'analyse des types on traite le type de retour de la condition booléenne si elle est vraie on prend le type de la première expression sinon on prend le type de la deuxième expression et on verifie que la premiere et la derniere expression ont le meme type. pour un exemple plus complet voir le fichier `programs/my_examples/ternal.stippled`.

boucle while

Comme dans la plupart des langages de programmation moderne nous avons décidé d'ajouter une boucle while. comme dans l'exemple precedent les memes fichiers ont été modifiés pour prendre en compte cette extension. l'extension fonctionne de la façon suivante:

```
 Begin
Int(k);
While (k < 10)
Begin
    Print(k);
    Copy(k, k+1)
End;
End
```

On peut remarquer que la boucle while evalue la condition et si elle est vraie elle execute le bloc de code qui debut par le mot clé `Begin` et fini par le mot clé `End`. Pour un exemple plus complet voir le fichier `programs/my_examples/while.stippled`.

les puissances

Le cas de puissances était assez simple à traiter , il suffit juste de rajouter une nouvelle regle dans les operations unaires et de definir les regles dans le parser et le lexer ainsi que dans le simplificateur et l'analyse des types. Pour un exemple plus complet voir le fichier `programs/my_examples/pow.stippled`.

initialisation des variables lors de la déclaration

Pour l'initialisation des variables lors de la déclaration etait un cas assez simple à traiter, il suffit juste de coniderer une nouvelle regle dans le parser et le lexer qui gere la declaration des variables avec un nom et une valeur. et de plus dans l'interpréteur il suffit juste de coniderer cette regle comme une **Variable_declaration** et **Assignement** en meme temps. voici un exemple:

```
 Begin
Int(x,1);
Print(x);
End
```

Pour un exemple plus complet voir le fichier `programs/my_examples/declare.stippled`.

le cast des variables

Pour le cast des variables est aussi un cas assez simple à traiter, il suffit juste de verifier dans l'analyse des types que les deux variables qui n'ont pas le meme type peuvent etre casté. par exemple si on a une variable de type **Int** et une autre de type **Float** on peut donc faire une operation entre les deux variables et le resultat sera de type **Float**.

pour un exemple plus complet voir le fichier `programs/my_examples/implecit_cast.stippled`.

Les difficultés rencontrées

Nous avons rencontré quelques difficultés lors de la réalisation de ce projet, par exemple:

OCAML

Nous avons eu du mal à comprendre le fonctionnement de OCAML et de ses librairies, meme malgre le fait que nous avons fait pendant plusieurs TDs ainsi que la L2. Ce langage possede une syntaxe assez particuliere et il faut un certain temps pour s'y habituer. ainsi qu'une maniere de penser assez differente de la plupart des langages de programmation moderne.

systeme ternaire

Nous avons eu du mal à implementer le systeme ternaire car au debut on a pense qu'il fallait le declairer comme une operation à trois operandes mais on a vite compris que ce n'etait pas le cas et qu'il fallait le traiter comme un **IfThenElse**. Gerer les priorités des operations etait aussi un peu compliqué.

scope des variables

gerer le partie des blocs et le scope des variables etait aussi un peu compliqué on est resté pendant des heures à essayer de comprendre comment gerer le scope des variables. mais le gros probleme qui freinait notre avancement etait le de ne pas bien connaitre la librairie **List** de OCAML. mais apres avoir compris comment elle fonctionne on a pu avancer plus vite.

Conclusion

En conclusion nous avons pu realiser un compilateur qui respecte les conditions obligatoires et qui prend en compte quelques extensions. Nous avons pu apprendre beaucoup de choses sur le fonctionnement d'un compilateur et sur le langage OCAML. Malgre le fait que nous avons eu quelques difficultés nous avons pu les surmonter et realiser un compilateur qui fonctionne correctement.