Rapport d'activité final

Corrections de l'AST

Lors de la dernière réunion avec Mme. Collin, nous avons remarqué que les négations ne se construisaient pas correctement dans l'AST. Nous avons donc commencé par modifier la construction afin de rectifier sa structure.

De même nous nous sommes rendus compte que les accès aux tableaux et aux type Record (nommés LValue) n'étaient pas gérés correctement. En effet, ils fonctionnaient lorsque qu'ils n'étaient composés que d'un accès (par exemple dict.key ou array[1]) mais à partir de deux ou plus (par exemple dict.key[0] ou array[2][1]), la structure du noeud correspondant dans l'AST ne convenait pas. Nous l'avons donc changé afin qu'ils s'affichent correctement (en cascade) dans l'AST.

Table des symboles et contrôles sémantiques

Une fois l'AST définitivement terminé. Il était temps de passer à l'analyse sémantique des programmes qui comprend la création et le remplissage des tables des symboles et les contrôles sémantiques qu'elles rendent possibles.

Parcours de l'AST

Afin d'effectuer l'analyse sémantique, nous avions besoin de parcourir l'AST. Au départ nous l'effectuions en même temps que la création du Graph mais nous nous sommes rapidement rendu compte que ce n'était pas pratique : en plus de rendre le code plutôt illisible, chaque erreur empêchait la création du Graph ce qui rendait le débogage compliqué.

Nous avons donc décidé de réutiliser le fichier .dot afin de créer un nouvel arbre à parcourir pour procéder à l'analyse sémantique.

Finalement, nous avons opté pour la solution la plus simple, à savoir créer une nouvelle instance de AstVisitor.

Remplissage de la TDS

Au début du parcours, on crée une instance de SymboleTableList qui contiendra toutes les TDS et une instance de SymboleTable qui correspond à la zone statique. On y ajoute les fonctions de bases du langage telles que "print" (pour des raisons de lisibilité elles n'apparaissent pas lors de l'affichage des TDS mais elles sont bien enregistrées).

Au cours du parcours de l'AST, à chaque fois qu'une variable doit être ajoutée à la TDS, on ajoute à la SymboleTable courante une instance de VariableTableEntry qui la représente à travers son nom et son type. A chaque fois qu'une fonction doit être ajoutée à la TDS, on ajoute une instance de FunctionTableEntry à la SymboleTable courante, puis on crée une nouvelle SymboleTable qui connait sa TDS mère, on y ajoute une VariableTableEntry pour chaque argument de la fonction puis elle devient la TDS courante le temps de parcourir le corps de la fonction et enfin sa mère devient à nouveau la TDS courante.

A chaque TDS sont attribués un identifiant unique et un numéro de région en fonction de leur TDS mère. Cette arborescence permet de gérer la portée des variables et fonctions définies dans les différentes parties du programme et donc de permettre ou non l'accès à certaines ou la définitions avec un certain nom, comme vu dans la partie suivante.

Contrôles sémantiques

Nous souhaitions créer une classe par contrôle sémantique mais au vu de la similutde entre la plupart des contrôles qui se résument à comparer deux types, nous avons procédé autrement : la plupart des noeuds de l'AST implémente une méthode getType qui permet de récupérer récursivement le type de l'expression contenue dans le sous-arbre. Nous avons aussi créé une classe TypeFactory qui s'occupe de garder en mémoire les types déclarés dans le programme.

Les différents contrôles sémantiques consistent en :

- la comparaison des types des deux membres d'une assignation
- la vérification que les différents opérateurs arithmétiques et booléens sont utilisés avec des types acceptés
- la vérification, à la déclaration d'un type (resp. d'une variable, resp. d'une fonction) qu'il n'existe pas déjà un type (resp. une variable, resp. une fonction) du même nom
- la vérification qu'un type, une variable ou une fonction utilisée à été déclarée au préalable
- la vérification qu'une fonction est utilisée avec les bons paramètres (nombre et types)
- la vérification que les champs accédés d'un dictionnaire existent
- la vérification que les éléments accédés d'un tableau existent
- la vérification de l'absence de division par 0

Ces contrôles sont effectués dans les différents contextes et conditionnent la création des entrées dans les TDS. Si une erreur sémantique est détectée, l'analyse continue mais si l'erreur a lieu sur une déclaration aucune entrée n'est ajoutée à la TDS.

Tout au long de l'analyse, la détection d'une erreur sémantique affiche un message correspondant de la forme "[SEM] ....." sur la sortie standard.

A la fin de l'analyse, l'ensemble des TDS est affichée sur la sortie standard.

Tests

Nous avions déjà, lors des étapes précédentes, créé des programmes avec des erreurs sémantiques. Ils nous ont servi de base et nous en avons créé de nouveaux mais nous avons surtout tenu à jour un fichier "test_sem.tig" qui regroupe tous les types d'erreurs gérés avec un exemple pour chaque et en commentaire l'erreur correspondante.

Ci-dessous, le fichier "test_sem.tig" suivi de ce que l'analyse sémantique affiche sur la sortie standard à sa lecture.

test_sem.tig

```
let
    var i:int := 1
    var s:string := "s"
```

```
    type arr = array of int
    var arr1:arr := arr[2] of 0
    type dict = {key:string, value:int}
    var dict1:dict := dict{key="foo", value=42}
    function foo(arg1:int, arg2:int):int = arg1
    var a:int := "je suis une string"               /* int expected, string
provided */
    var a2:string := i                              /* string expected, int
provided */
    var a3:string := foo(2, 3)                      /* string expected, int
provided */
    var b := 1 - "1"                                /* operator - can't be
used with int and string */
    var b2 := i * s                                 /* operator * can't be
used with int and string */
    var c := 1/0                                    /* can't divide by 0 */
    var d := nonDec                                 /* var nonDec is not
defined */
    var d2 := nonDec1 + nonDec2                     /* var nonDec1 is not
defined + var nonDec2 is not defined */
    var e := s > i                                  /* operator > can't be
used with string and int */
    var f := s & s                                  /* operator & can't be
used with string and string */
    var g := arr1[12]                               /* index out of bounds
*/
    var h := dict1.kye                              /* type dict1 has no
attribute "kye" */
    var s:string := "S"                             /* var s already
declared */
    function foo(arg1:int, arg2:int):int = arg2     /* function foo already
declared */
    var j:tyNoDec := 2                              /* type tyNoDec is not
defined */
    function print():int = 2                        /* function print
already declared */
    type int = string                               /* type int already
declared */
in
    nonDec3 := 2;                                   /* var nonDec3 is not
defined */
    i := nonDec4;                                   /* var nonDec4 is not
defined */
    i := nonDec5 + nonDec6;                         /* var nonDec5 is not
defined + var nonDec6 is not defined */
    nonDec7;
    i := "";                                        /* int expected, string
provided */
    s := foo(2, 2);                                 /* string expected, int
provided */
    bar(2);                                         /* function bar not
defined */
    foo(2);                                         /* function foo
expected 2 args. but 1 was given */
```

```
    foo("1", "2");                                  /* function foo
expected type int for argument 1 but string was given (+ arg 2) */
    print(foo(2, 2))                                /* function print
expected type string for argument 1 but int was given */
end
```

stdout :

```
[SEM] Type mismatch : int was expected but string was provided
[SEM] Type mismatch : string was expected but int was provided
[SEM] Type mismatch : string was expected but int was provided
[SEM] Operator - can't be used with types int and string
[SEM] Operator * can't be used with types int and string
[SEM] Division by 0
[SEM] Variable nonDec is not defined
[SEM] Variable nonDec1 is not defined
[SEM] Variable nonDec2 is not defined
[SEM] Operator > can't be used with types string and int
[SEM] Operator & can't be used with types string and string
[SEM] Index 12 out of bounds while accessing arr1
[SEM] Type dict1 has no attribute kye
[SEM] Variable s already declared
[SEM] Function foo already declared
[SEM] Type tyNoDec is not defined
[SEM] Function print already declared
[SEM] Type int already declared
[SEM] Variable nonDec3 is not defined
[SEM] Variable nonDec4 is not defined
[SEM] Variable nonDec5 is not defined
[SEM] Variable nonDec6 is not defined
[SEM] Type mismatch : int was expected but string was provided
[SEM] Type mismatch : string was expected but int was provided
[SEM] Function bar is not defined
[SEM] Function foo expected 2 arguments but 1 were given
[SEM] Function foo expected type int for argument 1 but type string was
given
[SEM] Function foo expected type int for argument 2 but type string was
given
[SEM] Function print expected type string for argument 1 but type int was
given
```

Gestion de projet

Le rapport d'activité précédent est disponible dans le dossier "gestion_projet" du git, avec des liens vers les différents éléments présentés.

Tout au long du projet, nous nous sommes organisés en planifiant des séances de travail communes en présentiel ou distanciel via Discord. Ces séances, nous ont permis une bonne répartition des tâches entre les membres du groupes et ont facilité la communication et la résolution de problèmes.

Initialement, nous avions prévu pour cette partie de faire chacun la correction de l'AST, la TDS ou les contrôles sémantiques, cependant toutes ses parties étant intrinsèquement liées, nous avons opté pour des séances de travail communes avec des objectifs à plus court termes qui ont aussi permis à chacun de travailler globalement sur tous les aspects et d'avancer de manière moins saccadée.

Ainsi, voici retrospectivement les éléments sur lesquels chacun a principalement travaillé :

Louis	Coralie	Louis-Vincent
Parcours de l'AST	Gestion des types	Ajout des symboles dans la TDS
Création des structures pour la TDS	Ajout des symboles dans la TDS	Contrôles sémantiques
Contrôles sémantiques	Contrôles sémantiques	Rédaction du rappport

Finalement voici les nombre d'heures approximatif de travail des membres du groupe :

Louis	Coralie	Louis-Vincent
78	83	83