Rapport d'Activité n°3 - Projet Compilation

Lucie BOUCHER, Rémi BOURDAIS, Lola MONTIGNIER

09 Janvier 2022

1 Mise en Contexte

Ce rapport est le 3ème et dernier rapport d'activité de la première partie du projet compilation. Le 2ème rapport traitait la création d'un AST et des tables des symboles (TDS) associées à n'importe quel programme codé avec le langage de programmation CHAOS.

Dans cette 3ème partie, nous avons amélioré la génération des TDS afin de réaliser par la suite des contrôles sémantiques.

Dans ce rapport, vous retrouverez les améliorations de la génération des TDS réalisées, une liste des contrôles sémantiques réalisés. Ensuite, la partie sur les tests réalisés sur nos contrôles sémantiques sera présentée puis une partie sur la gestion de ce projet sera détaillée.

2 Amélioration de la TDS

Depuis le dernier jalon, nous avons ajouté les champs *isUsed* et *initialized* et des fonctions à nos classes VarType (qui gèrent les variables et types) et ProcFonc (qui gèrent les procédures et fonctions) dans les tables des symboles. Ces champs et fonctions permettent par la suite de gérer des erreurs et/ou warnings.

Nous avons modifié l'affichage des TDS afin de les rendre plus lisibles (séparation des variables et types lors de l'affichage) et d'afficher les nouveaux champs.

De plus nous générons plus de TDS que précédemment, car nous n'avions pas encore généré de TDS pour les boucles While et For et les If Then Else.

```
| TABLE ID=0
| Nom de la fonction=programme
| Fonctions:
| Nature     | Name          | Type      | Nb Args | Types Arguments      | Est Utilisé
| fonction   | somme         | null      | 1       | int,                 | true

| TABLE ID=1
| Nom de la fonction=somme
| ID du pere=0
| Variables:
| Nature | Name          | Type      | Deplacement  | Initialisation | Est Utilisé    | Dimensions
| Var    | x             | int       | 1            | true           | true           | 1
| Fonctions:
| Nature     | Name          | Type      | Nb Args | Types Arguments      | Est Utilisé
| fonction   | Then block 1  | Then      | 0       |                      | true
| fonction   | Else block 1  | Else      | 0       |                      | true
```

Figure 1: Nouveau visuel des tables des symboles pour le programme somme.exp

3 Analyse sémantique

3.1 Choix des contrôles sémantiques

3.1.1 Gestion des erreurs

L'analyse sémantique permet de détecter différentes erreurs :

- Contrôles des déclarations :

Contrôle	Réalisé par
Utilisation d'un type de structure inexistant	Rémi x
Accès à un champ d'une structure inexistant	Lucie x
Accès à un type/variable/fonction non déclaré	Rémi x
Multiple déclaration de type/variable/fonction	Lucie x

- Contrôles des affectations/accès :

Contrôle	Réalisé par
Mauvais placement de l'élément break	Lola x
Affection d'un mauvais type à une variable d'un autre type	Lucie x
Borne min d'une boucle for strictement supérieure à la borne max	Lucie x
Accessibilité d'éléments d'une liste avec un rang donné	Lola x

- Contrôles des appels de fonctions :

Contrôle	Réalisé par
Mauvais types d'arguments lors de l'appel d'une fonction	Rémi x
Mauvais nombre d'arguments lors de l'appel d'une fonction	Rémi x

- Contrôles des Opérations :

Contrôle	Réalisé par
Mauvais types utilisés sur les opérations unaires et binaires (booléennes et arithmétiques)	Rémi x
Division par 0	Lola x

3.1.2 Gestion des warnings

L'analyse sémantique permet de détecter différents warnings :

- Contrôles des Opérations :

Contrôle	Réalisé par
Suggestion de simplification de calcul	Lola

- Contrôles des déclarations :

Contrôle	Réalisé par
Vérification utilité des blocs IfThen et While	Lola x
Type/Variable/Fonction déclaré(e) mais jamais utilisé(e)	Lucie x

- Contrôles des affectations/accès :

Contrôle	Réalisé par
Indication de code inaccessible suite à un break	Lola x
Borne min d'une boucle égale à la borne max	Lucie x

3.2 Implémentation des contrôles sémantiques

Chaque contrôle sémantique est défini dans une classe différente. Puis , les contrôles sémantiques sont appelés dans le fichier tdsVisitor.java dans les fonctions ayant besoin d'être contrôler. Par exemple, le contrôle sémantique qui vérifie qu'une division par 0 n'est pas effectué, défini dans le classe Division.java, est appelé lorsqu'on visite l'instance Div.

3.3 Test des contrôles sémantiques

Afin de tester l'efficacité des différents contrôles sémantiques implémentés, nous avons réalisé toute une batterie de tests sur ces derniers. A chaque nouveau contrôle sémantique implémenté, un programme test était réalisé afin de vérifier son fonctionnement.
De plus, nous avons également réalisé un programme réunissant tous les contrôles sémantiques implémentés

```
let

    type rec = { val : int }
    type rec_arr = array of rec
    type i = array of int
    type i = int /*Double Déclaration de type */
    var  table := rec_arr[2] of nil
    var i := 0 /* Variable inutilisée */

    function printboard() =
        (for i := 5 to 5  /* Borne Min == Borne Max */
            do 25/0)  /* Division par 0 */

    function nombrePremier(d : int) =
        (for i := 7 to 6 /* Borne Min > Borne Max */
            do print(d))

    function nombrePremier(c:int) = /*Double Déclaration*/
        (for i := 5 to 6*"a" /* Mauvais type opération */
            do print(c))

in

    break; /* Break hors boucle */
    for i := 0 to 1 do
        table[0] := (rec{val = 42}) ;

    table[0].val := 51*10+3/7-9; /* Simplification Calcul */
    table[1].val := "toto"; /* Erreur de type */
    table[0].valeur := 48; /* Erreur champ structure inexistant */
    table[0].val := N; /* Variable non déclarée */
    table[2].val := 49; /* Erreur de débordement accés liste*/

    table.val := 45; /* Erreur type de structure inexistant */
    nombrePremier1(); /* Fonction non déclarée */

    if "verifie" = "verifie" | 3>1  & 1=1   then /* If toujours vrai */
        print(1);

    printboard();
    printboard(2,3); /* Fonction avec mauvais nombre d'arguments */
    nombrePremier("test"); /* Fonction avec mauvais type d'arguments */

    if -"operation" then /* Mauvais type opération unaire */
        print(1);

    if 1+2*3 | 1<2 then /* Mauvais type opération booléenne */
        print(1);

    while  3<>3 | "vrai"="faux"  do /* while toujours vrai */
        break;
        print(1); /* Code inaccessible */
        print(3) /* Code inaccessible */
end
```

Figure 2: Programme en CHAOS regroupant tous les contrôles sémantiques

```
Ligne 6:4 : DoubleDeclarationException : Type i deja déclarée dans le même bloc
Ligne 11:18 : BoucleForWarning : Borne min egale à borne max (attendu une borne min strictement inférieure à une borne max)
Ligne 12:18 : DivisionException : Division par 0 (attendu un diviseur différent de 0)
Ligne 15:18 : BoucleForException : Borne min supérieure à borne max (attendu une borne min strictement inférieure à une borne max)
Ligne 18:4 : DoubleDeclarationException : fonction nombrePremier deja déclarée dans le même bloc
Ligne 19:23 : ExpressionException : Type incorrect dans l'expression : (attendu : int)
Ligne 24:4 : BreakError : le break est mal placé
Ligne 28:20 : SimplificationWarning : vous pourriez remplacer le calcul cote droit du symbole := par 501 plutot
Ligne 29:4 : ExpressionException : Type incorrect dans l'expression (attendu : int dans le membre droit de l'expression)
Ligne 30:4 : DeclarationException : Variable rec.valeur non déclarée
Ligne 31:20 : DeclarationException : Variable N non déclarée
Ligne 32:4 : OutOfRangeError : le rang demandé dépasse la taille de la liste à laquelle on veut accéder
Ligne 34:4 : DeclarationException : Variable rec_arr.val non déclarée
Ligne 35:4 : DeclarationException : Fonction nombrePremier1 non déclarée
Ligne 37:7 : ifInutileWarning : une condition if est utilisée alors qu'elle est toujours vraie
Ligne 41:4 : FonctionException : Nombre de paramètres incorrect (attendu : 0 trouvé : 2)
Ligne 42:4 : FonctionException : Type du 1er paramètre incorrect (attendu : int)
Ligne 44:7 : ExpressionException : Type incorrect dans l'expression (attendu : bool)
Ligne 47:7 : ExpressionException : Type incorrect dans l'expression (attendu : bool)
Ligne 50:11 : whileInutileWarning : une condition while est utilisée alors qu'elle est toujours fausse
Ligne 51:8 : WhileCodeInutileWarning avec break: la boucle n'exécute pas le code
Ligne 5:0 : UtilisationWarning dans programme : Type i non utilisé(e)
Ligne 8:13 : UtilisationWarning dans programme : Var i non utilisé(e)
```

Figure 3: Les contrôles sémantiques affichés lors de l'exécution du programme

4 Gestion de Projet

Pour la gestion de projet, nous avons désigné un chef de projet : Rémi Bourdais. Une séance de travail par semaine était organisée afin de pouvoir garder un rythme soutenu pour accomplir toutes les tâches qui nous avaient été fixées. Un groupe messenger a également été créé afin de communiquer quand nous étions en distanciel. Une matrice SWOT et des diagrammes de gantt prévisionnel et réel ont été également réalisés afin d'appliquer les méthodes qui nous ont été enseignées lors de notre première année à TELECOM Nancy.

4.1 Matrice SWOT

Pour la gestion de projet, nous avons réalisé une matrice SWOT afin d'identifier nos atouts et nos point faibles. Ainsi, nous avons pu nous rendre compte qu'il était important de consacrer des plages de travail à 3 pour avancer tous ensemble.

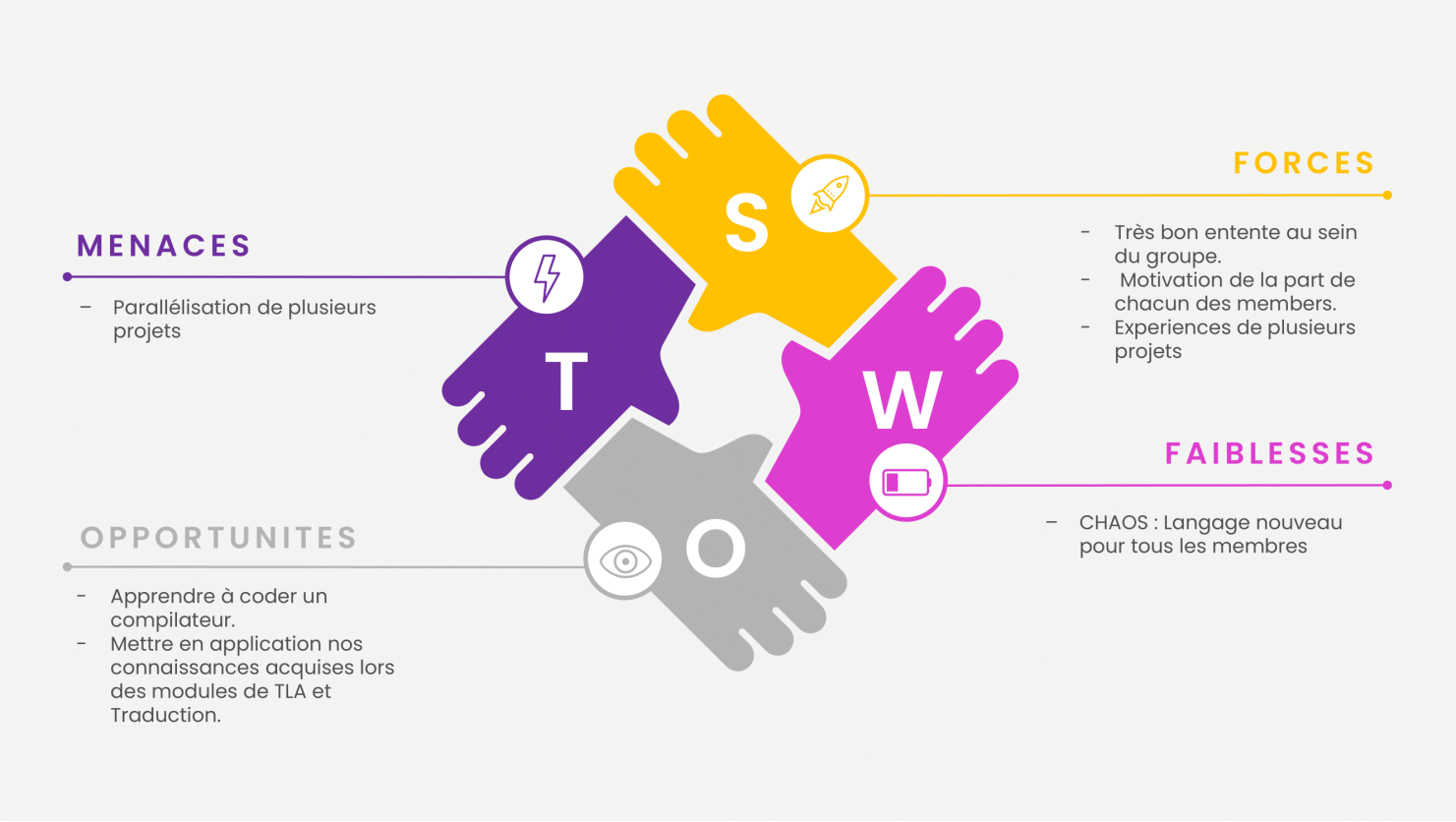

Figure 4: Matrice SWOT

4.2 Diagrammes de Gantt

4.2.1 Diagramme de Gantt prévisionnel

Dans un premier temps, nous avons réalisé un diagramme de Gantt réel afin de pouvoir organiser notre travail en jalons entre les différentes dates clés du projet.

Figure 5: Diagramme de Gantt prévisionnel

4.2.2 Diagramme de Gantt réel

La comparaison des deux diagrammes permet de mettre en avant le fait que nous avons bien respecté le diagramme de Gantt prévisionnel. Cependant, au début du projet nous ne connaissions pas exactement les dates des différents rendus, nous avons donc du adapter légèrement le diagramme réel.

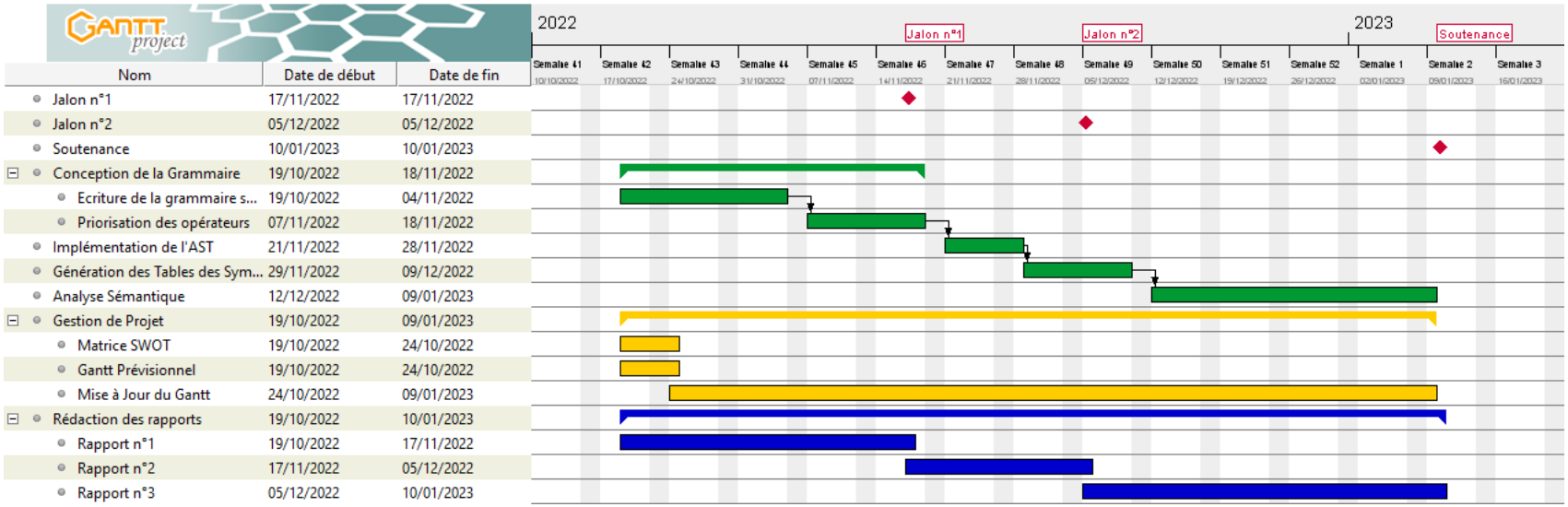

Figure 6: Diagramme de Gantt réel

5 Conclusion

Ce projet nous aura permis de construire un compilateur de l'écriture de la grammaire à l'analyse sémantique en passant par la construction d'arbres lexicaux et syntaxiques pour des programmes écris dans le langage de programmation CHAOS. La cohésion du groupe aura permis d'avancer efficacement lors des séances de travail. Dans l'ensemble, nous n'avons pas rencontré de difficultés particulières, nous avons même trouvé beaucoup de satisfaction dans la réalisation de ce projet.