Projet de Compilation

Rapport Final

Activité d'Apprentissage S-INFO-012

Membres du groupe:

Altruy Alan
Nosal Victor

Année académique 2022-2023

Faculté des Sciences, Université de Mons

Résumé

Ce rapport présente l'implémentation en python d'un moteur de templates interprétant du code Dumbo, qui a été réalisé dans le cadre du cours de Compilation dispensé par Mme BRUYERE Véronique et Mr DECAN Alexandre en année académique 2022-2023.

Table des matières

Description de la grammaire

La grammaire utilisée dans ce projet est basée sur la grammaire de base du dumbo, que nous avons complétée et modifiée. Voici une liste de ces modifications et ajouts :

- Ajout de la possibilité pour un bloc dumbo d'être vide via la règle : < dumbo_bloc > → {{ }}
- Gestion du « if » via la règle :
  < expression > → if < boolean_expression > do < expressions_list > endif
- Ajout des booléens BOOL « true » et « false », des opérations « and » et « or », mais aussi des comparaisons d'entiers COMP « < », « > », « = » et « != » via les règles :
  < boolean_expression > → < int_term > COMP < int_term >
  | < boolean_term > or < boolean_term >
  | < boolean_term > and < boolean_term >
  | BOOL
  < boolean_term > → < boolean_expression >
  | VARIABLE

- Gestion des entiers INT, des variables entières ainsi que des opérations d'addition ADD_OP, de soustraction ADD_OP, de multiplication MUL_OP et de division MUL_OP via les règles liées à < int_expression > :
  < int_expression > → INT
  | VARIABLE
  | < int_expression > ADD_OP < int_expression >
  | < int_expression > MUL_OP < int_expression >
  < int_term > → VARIABLE
  | < int_expression >
  | (< int_expression >)

- Ajout de la possibilité d'assigner une < int_expression > à une variable via la règle :
  < expression > → VARIABLE := < int_expression >

- Ajout de la possibilité d'assigner une < boolean_expression > à une variable via la règle :
  < expression > → VARIABLE := < boolean_expression >

Analyse sémantique

Discussion

Après nous être occupés des tokens pendant la phase d'analyse lexicale et de la grammaire durant l'analyse synthaxique, nous avons dans un premier temps essayé de traiter l'analyse sémantique uniquement via les règles de calcul de la grammaire. Nous nous sommes alors rendu compte que la caractéristique « bottom-up » du parser ne nous permettrait pas de gérer le cas des boucles for. C'est à ce moment que nous avons décidé de créer l'équivalent d'un arbre syntaxique en sortie du parser. Cet arbre, qui n'en est pas vraiment un et dont les nœuds sont représentés par des tuples, pourrait alors être parcouru de manière « top-down », nous permettant de traiter plus facilement certains cas comme celui des boucles for.

Boucles for et scopes

Nous avons décidé de nous servir d'un dictionnaire comme structure de données permettant de stocker les valeurs des différentes variables. En nous intéressant quelque peu à la notion de « scope », nous nous sommes rendu compte qu'il était nécessaire, avant de traiter une boucle for, de garder en mémoire la valeur, si celle-ci existe, de la variable qui sera modifiée à chaque itération de la boucle. Pour ce faire, le code de la gestion d'une boucle for retient la valeur de la variable en question le temps du traitement de cette boucle.

Problèmes rencontrés

Ce sont sans conteste les boucles for qui ont représenté le plus grand défi de ce projet. Cependant, c'est bien grâce à celles-ci que nous avons finalement implémenté une version du moteur de templates correspondant plus à notre vision théorique de la compilation. Il nous a fallu relativement longtemps pour nous rendre compte que la gestion « bottom-up » du parser ne nous permettrait pas de traiter les boucles for. Nous avons alors implémenté le code nous permettant de parcourir les données de manière « top-down ». La gestion des scopes et des variables s'est aussi naturellement imposée à nous.

Nous nous sommes également rendus compte d'un problème majeur alors que le développement du projet était terminé. En effet, nous avions désactivé le mode 'debug' de PLY, ce qui signifie que les erreurs et avertissements de conflits n'étaient pas signalés. Nous avons donc pensé que la grammaire était sans conflit, alors qu'en réalité ce n'était pas le cas. Ce n'est qu'après un examen minutieux du code et une série de tests approfondis que nous avons identifié ce problème. Cela nous a demandé un peu d'effort pour permettre de proposer une grammaire sans ambiguïté mais nous y sommes parvenus.

Répartition du travail

En ce qui concerne la répartition de la charge de travail, Victor a commencé par construire les fondations des phases d'analyse lexicale et syntaxique. S'en est suivi une période d'ajustement et de complétion de ces phases par les deux membres du groupe. Ensuite, Alan s'est occupé de la majorité des aspects constituant la phase sémantique. Enfin, tous les membres du groupe se sont attelés à la gestion des dernières erreurs ainsi qu'aux tests.

Conclusion

Bien que ce projet de compilation nous ait donné du fil à retordre, notamment de par la complexité de la librairie PLY, celui-ci nous a permis de faire le lien entre les notions théoriques enseignées pendant les cours et l'application pratique du moteur de templates.