LOG3210 - Élément de langages et compilateur
TP2 : Analyseur sémantique

Ettore Merlo – Professeur
Doriane Olewicki – Chargée de laboratoire

Hiver 2020

1 Objectifs

- Se familiariser avec les visiteurs et l'AST de JavaCC
- Faire de l'analyse sémantique.

2 Travail à faire

Après l'analyse lexical et l'analyse syntaxique, il est temps de se pencher sur l'analyse sémantique. Le premier TP a permis de créer un parseur qui vérifie que la syntaxe du langage était respectée. Cependant, l'analyse de la structure n'est pas suffisant pour s'assurer qu'un programme est valide. Il faut aussi vérifier que ce qui est écrit à un sens.

L'analyse sémantique permet au compilateur de s'assurer que les variables sont déclarées avant d'être utilisées, que les valeurs mises dans les variables sont du type attendu, que le nombre et le type des arguments de l'appel d'une fonction correspond à la déclaration de celle-ci, et de beaucoup d'autres choses similaires. De plus, l'analyse sémantique permet de faire des analyses sur le code, ce qui permettrait, en outre, de créer un interpréteur (comme celui de Python, par exemple), de mettre de la coloration syntaxique et de l'auto-complétions automatique dans un interface de développement, etc.

Dans le TP2, il vous faudra écrire un visiteur qui fera de l'analyse sémantique sur un langage fournis. Vous devrez, dans un premier temps faire un visiteur qui extraira quelques métriques de programme. Puis, vous devrez vérifier que les programmes qui passent par votre compilateur sont valides et faire échouer la compilation dans le cas contraire. Le même visiteur devras donc faire les vérifications présentées plus bas et lancer le message d'erreur si le programme ne respecte pas la sémantique du langage.

La grammaire est similaire a celle du TP1, mais rajoute un bloc de déclaration de variable au début du programme avec le type (bool ou num) de ceux-ci. Consultez le fichier `Template.jjt` (la grammaire) pour identifier les spécificités du langage.

Pour vous aider, des commentaires ont été laisser dans le visiteur, ainsi qu'une énumération pour les types, une fonction de comparaison des types et une structure de donnée (DataStruct) permettant de transmettre des données plus facilement de nœuds en nœuds.

De plus, la table de Symbole est déjà initialisé (en tant que HashMap) en haut du visiteur.

2.1 Calcul de métriques programme

Votre visiteur doit calculer les métrique suivante par programme (fichier de test) :

- Nombre de variables (identifiant différent);
- Nombre de boucles while;

- Nombre de conditions if;
- Nombre d'opérations (toutes les opérations de la grammaire).

L'output de ce visiteur doit avoir le format suivant :

{VAR : x , WHILE : y , IF : z , OP : w}

Attention, cet output doit être renvoyé sur `m.writer` dans le cas où il n'y a **PAS** d'erreur de compilation (les sections suivants).

2.2 Vérification de la déclaration unique

Vérifier que une variable n'est déclarée que une fois.

Si une erreur est détectée, lancé *"Identifier ... has multiple declarations"*.

2.3 Vérification du types des assignations

L'analyseur doit vérifier que le type de l'expression à droite d'une assignation soit le même que le type qui a été déclaré au début du programme pour l'identifiant de gauche.

Si une erreur est détectée, lancé *"Invalid type in assignation of Identifier ..."*.

2.4 Vérification du type booléen dans les expressions de conditions

la condition d'un "if" et d'un "while" doit être une expression ("true", "false" ou comparaison entre variable par exemple) ou une variable de type booléen. Lancez l'erreur *"Invalid type in condition."* si l'expression est invalide.

2.5 Vérifications du types des opérandes dans les expressions

Le langage du TP est fortement typé et n'a pas de conversion implicite. Le visiteur doit donc détecter les **erreurs** suivantes :

- Des valeurs booléennes sont utilisées dans des opérations mathématiques (addition, multiplication, négation, comparaison autre que "==" et "!=");
- des valeurs numériques sont utilisés avec des opérations booléennes (&&,||,!);
- les types des variables de chaque côté d'une comparaison diffèrent.

La première erreur rencontrée est celle renvoyée.

On doit vérifier que le type des variables est conforme au type déclaré au début du programme en utilisant la table de symbole.

Lorsque les types ne sont pas valides, le message d'erreur est *"Invalid type in expression."*.

3 Barème

Le TP est évalué sur 20 points, les points étant distribué comme suit :

- Métriques : 6 points;
- Déclaration multiple : 2 points;
- Type des assignations : 3 points;

- Type booléen dans les expressions de condition : 3 points;
- Type des opérandes dans les expressions : 6 points.

L'ensemble des tests donnés sous Ant doivent passer au vert pour que le laboratoire soit réussi. La qualité du code sera aussi vérifiée et prise en compte pour la correction.

Les tests se trouvent dans le dossier `test-suite/SemantiqueTest` où vous pouvez parcourir les dossiers `data` pour les programmes et `expected` pour les résultats attendus. Le test `test_base.le` devrait être vert. Vous pouvez changer ce test pour faire vos propres tests (dans les deux dossiers `data` et `expected`) mais les autres fichiers ne devraient pas être changé.

4 Remise

Le devoir doit être fait en **binôme**. Remettez sur Moodle une archive nommée *log3210-tp2-matricule1-matricule2.zip* avec uniquement le fichier `SemantiqueVisitor.java` ainsi qu'un fichier `README.md` contenant tous commentaires concernant le projet.

L'échéance pour la remise est le **samedi 15 Février 2019 à 23 h 55**.

Une pénalité de 10 points (50%) s'appliquera par jour de retard. Une pénalité de 4 points (20%) s'appliquera si la remise n'est pas conforme aux exigences (nom du fichier de remise, fichier `SemantiqueVisitor.java` seulement).

Si vous avez des questions, veuillez me contacter sur moodle ou sur mon courriel : doriane.olewicki@gmail.com.