LOG3210 - Éléments de langages et compilateurs
TP2 : Analyseur sémantique

Doriane Olewicki – Chargée de cours

Chargé(e)s de laboratoire:
Quentin Guidée
Raphaël Tremblay
Sara Beddouch

Hiver 2023

1 Objectifs

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

2 Mise en contexte

Après l'analyse lexicale 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, celui-ci n'est pas suffisant pour s'assurer qu'un programme est valide. Il faut à présent vérifier le sens (la sémantique) de l'arbre (AST) généré.

L'analyseur sémantique effectue cette vérification et permet au compilateur de s'assurer que les variables sont déclarées avant d'être utilisées, que les valeurs attribuées aux variables sont du type attendu, que le nombre et le type des arguments d'un appel de fonction correspondent à la déclaration de celle-ci, etc.. De plus, l'analyse sémantique permet de faire des analyses sur le code, ce qui peut également permettre de créer un interpréteur (comme celui de Python, par exemple), d'ajouter de la coloration syntaxique et de l'auto-complétion dans une interface de développement, etc.

3 Travail à faire

Dans le TP2, vous devrez implémenter un analyseur sémantique pour un langage fourni. Il doit extraire des métriques et vérifier la validité des programmes. Si un programme ne respecte pas la sémantique du langage, votre analyseur doit échouer la compilation et afficher un message d'erreur.

La grammaire est similaire à celle du TP1, mais simplifiée pour les fonctions et inclut une déclaration de variable au début du programme avec son type (bool ou num). Consultez *Langage.jjt* pour plus d'informations.

Pour vous aider, des commentaires ont été laissés dans le visiteur, ainsi qu'une énumération pour les types et une structure de donnée (DataStruct) permettant de transmettre des données plus facilement de nœud en nœud. Enfin, la table de Symboles est déjà initialisée (en tant que HashMap) en haut du visiteur.

3.1 Calcul de métriques du programme

Votre visiteur doit calculer les métriques suivantes par programme (fichier de test) :

- Nombre de variables (identifiant différent, incluant les noms de fonction);
- Nombre de boucles while;
- Nombre de conditions if;
- Nombre de fonctions;
- Nombre d'opérations (toutes les opérations de la grammaire).

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

{VAR:0 , WHILE:0 , IF:0 , FUNC:0 , OP:0}

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

3.2 Vérification de la déclaration unique

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

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

3.3 Vérification du type 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 utilisées avec des opérations booléennes (&&,||,!);
- les types des variables de chaque côté d'une comparaison diffèrent.

Si une erreur est liée à un type non valide, lancez *"Invalid type in expression"*.
Si une erreur est liée à l'utilisation d'un identifiant non défini, lancez *"Invalid type in expression"*.

3.4 Vérification du type 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. Vous devez gérer ceci, quelque soit l'expression de droite, basique (nombre, booléen ou identifiant) ou complexe (avec des opérations).

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

3.5 Vérification du type booléen dans les expressions de conditions

La condition d'un "if" et d'un "while" doit être une expression ("true", "false", une comparaison entre des variables, etc.) ou une variable booléenne.

Si une erreur est détectée, lancez *"Invalid type in condition"* si l'expression est invalide.

3.6 Vérification du type de retour d'une fonction

Le type de valeur renvoyé par la fonction doit correspondre au type déclaré par la signature de fonction.

Si une erreur est détectée, lancez *"Return type does not match function type"*.

4 Fonctions utiles

- `node.childrenAccept(visitor, data);`
  Chaque enfant du nœud "node" est visité par "visitor" (souvent "this" pour dire qu'on utilise le même visiteur) et les données "data" sont transmises par "node" à ses enfants. C.-à-d. qu'on applique la fonction dans le visiteur "visitor" sur les enfants dans l'ordre. Si un enfant modifie l'objet "data", la version modifiée est transmise à l'enfant suivant. La valeur de retour à cette fonction est "data", c'est pourquoi utiliser "data" est utile.

- `node.jjtGetChild(i)`
  Pointeur vers l'enfant numéro "i" (La numérotation commence par 0).

- `node.jjtAccept(visitor, data)`
  Le nœud "node" est visité par "visitor" (souvent "this" pour dire qu'on utilise le même visiteur) et les données "data" sont transmises à "node" par le nœud courant. Souvent, on fera `jjtAccept` sur `jjtGetChild` pour visiter un parent en particulier. Valeur de retour est la valeur de retour de la fonction visit du visiteur qui est utilisée.

- `node.jjtGetNumChildren()`
  renvoie le nombre d'enfants de "node".

- `noe.getValue()` or `noe.getOps()`
  Certains nœuds ont une fonction getValue (par exemple pour le nœud Identifier, cela permet de récupérer le nom de l'identifiant). Référez-vous au dossier `src/analyser/ast` pour voir les nœuds spéciaux en question.

On vous invite à parcourir le fichier `gen-src/analyser.ast/SimpleNode` pour voir toutes les fonctions utilisable sur les nœuds, et le dossier `src/analyser/ast` pour les nœuds particuliers.

5 Barème

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

- Métriques : 3 points;
- Déclaration multiple : 2 points;
- Type booléen dans les expressions de condition : 3 points;
- Type des opérandes dans les expressions : 6 points;
- Type des assignations : 3 points.
- Type des fonctions : 3 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.

6 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 les commentaires concernant le projet.

L'échéance pour la remise est le **dimanche 19 février 2023 à 23 h 59**.

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).

Les remises individuelles ne sont pas autorisées! Si vous ne trouvez pas de binômes, veuillez contacter votre chargé (minimum 5 jours avant la remise).

Si vous avez des questions, veuillez nous contacter sur discord.