TP3 – Génération de code intermédiaire

Hivers 2019

Chargé de laboratoire : Félix Brunet (felix.brunet@polymtl.ca)

1. Objectifs

- Se familiariser avec la traduction dirigée par la syntaxe
- Utilisé une forme de code intermédiaire : le code à trois adresses.
- Implémenter des techniques d'optimisations de code intermédiaire.

2. Travail à faire

Une fois l'analyse lexical, syntaxique et sémantique complétée, On peut avoir confiance que le code qui a passé ces étapes est valide. L'analyse lexical à permis de confirmer que tous les mots de notre code sont bien écrits. L'analyse syntaxique vérifie que le texte et les phrases suivent la bonne structure, et l'analyse sémantique à vérifier que les phrases avaient un sens.

La prochaine étape du compilateur est de faire quelque chose avec ce code. Le troisième travail pratique consistera à écrire un visiteur qui traduira du code valide en une représentation intermédiaire, et qui fera certaine optimisation simple sur celui-ci.

La représentation intermédiaire choisis est le code à trois adresse, similaire à celle présenté dans le livre du dragon au chapitre 6. La différence majeure avec le livre est que les labels seront intercalés avec le code plutôt que mis à part. De plus, seule la partie concernant les expressions arithmétique et les flux de contrôles seront utilisé.

Le livre utilise dans ses exemples une approche ou il concatène la chaine de charactère représentant le code intermédiaire. Dans ce Tp, il sera plus simple de « print » le résultat au fur et à mesure. Ainsi, lorsque le livre utilise « gen » ou « label » dans ses exemples de codes, c'est qu'il faut « print » le résultat. Et lorsque le livre utilise la variable « code » d'un nœud enfant, c'est qu'il est temps d'accepter (jjtAccept) l'enfant pour qu'il puisse lui aussi généré tous le code dont il a besoin.

Dans tous le TP, on peut considérer que les tests d'analyse sémantique passent déjà. Il n'est pas nécessaire de prévoir aux erreurs rendu à cette étape de la compilation, grâce aux vérifications des étapes précédentes. Il serait possible de faire l'analyse sémantique et la génération de code intermédiaire en une seul passe, mais cela alourdirait le visiteur inutilement pour le bien de ce Travail Pratique. Une implémentation du visiteur « SemantiqueVisitor » est fournis avec les sources. Il est donc conseillé de tester son générateur de code intermédiaire en fournissant les mêmes données aux deux visiteurs en parallèle, de sorte que si le code tester est invalide, l'analyseur syntaxique le révèlera aussitôt.

2.1 Traduction des expressions arithmétique et des assignations de nombre

Il faut traduire les expressions arithmétiques en code à trois adresses tel que décris dans le livre du dragon à la section 6.4.1.
On doit faire une distinction entre l'assignation de nombre et celle de booléen. La table de symbole des déjà remplis dans le code fournis pour permettre d'avoir l'information de type au moment opportuns.

Pour la traduction des expressions arithmétiques, il est conseillé d'avoir une approche « bottom up », utilisant principalement les valeurs de retour des fonctions « visit ». Chaque nœud d'expressions devrait print le calcul qui le représente, et retourner le nom de la variable qui contient le résultat de toutes les opérations effectuées, pour que le nœud parent puisse utiliser ce résultat dans ses propres opérations. C'est l'approche utilisé dans le livre.

Les nœuds d'expression logique devraient retourner la valeur de leur première enfant pour être transparent face aux expressions arithmétiques. L'analyse sémantique nous assure que les expressions logique et arithmétique ne sont pas mélangées.

2.2 Traduction des expressions logique, des flux de contrôle et des assignations booléenne

Il faut traduire les expressions logiques avec court-circuitage tel que décris à la section 6.6.3 et 6.6.4 du livre. La figure 6.36 et la figure 6.37 sont très utiles pour se faire, mais ne sont pas suffisante en soit, car notre grammaire est un peu différente de celle du livre.
Entre autres, les opérations « && », « || » et « ! » peuvent être utilisé plusieurs fois dans la même production. Par exemple, la chaine « a || b || c » produit un nœud qui a trois enfants.

De plus, les booléens n'existe pas dans le code à trois adresses. Les variables ne peuvent contenir que des nombres. Cela signifie que les expressions logiques doivent être transformé en une série de « goto » et de « if test goto » tel que présenté dans les sections mentionnées plus haut.

Puisque les booléens n'existe pas, lorsque le résultat d'une expression logique est enregistré dans une variable, Il faut modifier le comportement de l'assignation pour qu'elle enregistre le résultat sous forme de 1 ou de 0 dans la variable. De la même manière, il faut modifier la référence de variable dans les expressions booléenne pour que cela produise des goto conditionnelle approprié plutôt que de retournée la valeur.

Pour cette partie, il est conseillé d'avoir une approche « top down » qui passe en paramètre le nom des labels True et false. Ce sont les opérations logiques qui vont faire la majorité des gotos, et il faut qu'ils sachent vers ou aller selon si l'expression est vraie ou non. C'est l'approche utilisé par le livre.

2.3 Élimination de Goto et de Label inutile.

Il est possible de réduire le nombre de label et de goto en utilisant la technique présentée à la section 6.6.5 du livre du dragon. Cette technique doit bien entendu être adapté pour notre grammaire.

La référence de variable fonctionne de la même manière que présenté à la figure 6.39, pour l'utilisation d'opérateur relationnelle.

Cette technique n'éliminera pas tous les goto et label qu'il serait possible d'éliminer à la main. Il n'est

pas attendu que le code généré soit totalement optimisé. Il est normal qu'il y ait du code mort, des labels inutiles et des « goto » redondant, même après l'optimisation.

L'optimisation parfaite du code représenterais une tache beaucoup plus grande que celle demandé dans le TP et demanderais plusieurs techniques successive et complexe d'optimisation qui ne sont pas à l'étude pour ce cours d'introduction.

3. Barème

Le TP est évalué sur 20 points, avec un nombre de points distribué également entre toutes les questions.

Des tests unitaires et des exemples de résultat sont fournis pour aider à la compréhension du Travail. Ces exemples sont à titre indicatif et ne sont pas la seule bonne réponse au travail. Il n'est pas nécessaire que les tests unitaires passent.

4. Remise

Remettez sur Moodle une archive nommée log3210-tp3-*matricule1*-*matricule2*.zip avec uniquement le fichier IntermediateCodeGenVisitor.java.

Le TP **doit** être fait en équipe de 2. En cas de nombre impair d'étudiant dans la classe, il y aura exactement une équipe de 3.

L'échéance pour la remise est le dimanche 23 mars 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 demandé seulement).