

LOG3210 - Élément de langages et compilateur

TP3 : Génération de code intermédiaire

Ettore Merlo – Professeur
Doriane Olewicki – Chargée de laboratoire

Hiver 2020

1 Objectifs

- Se familiariser avec la traduction dirigée par la syntaxe
- Utiliser 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 a 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 a vérifié 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 choisie est le code à trois adresses, similaire à celle présentée 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 sera utilisée.

Le livre utilise dans ses exemples une approche où il concatène la chaîne de caractère 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 générer 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 rendus 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 seule passe, mais cela alourdirait le visiteur inutilement pour le bien de ce Travail Pratique. Une implémentation du visiteur `SemantiqueVisitor` est fournie 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 révélera aussitôt.

Vous devrez, pour votre part, implémenter deux visiteurs, `IntermediateCodeGenVisitor` et ensuite `IntermediateCodeGenFallVisitor`, répondant aux prochaines sections.

2.1 Code intermédiaire initial

Il faut traduire les expressions de la grammaire en code à trois adresses tel que décrits dans le document de référence **SDD pour la generation de code intermediaire (v3.b)**.

Vous devez traduire les expressions :

- Arithmétique (expression sur des nombres avec des opérantes);
- Booléennes (expressions sur des nombres ou booléen avec des opérantes);
- Conditionnelles (if et while).

Dans le document de référence, vous trouverez le SDD à appliquer pour ce TP.

Vous devez aussi implémenté la **traduction des switches**. Pour cet exercice, vous devez implémenter l'écriture du code intermédiaire pour l'expression switch. Pour cela, suivez les consignes des slides du cours 7.

Voici un exemple de la représentation machine :

Listing 1: Switch statement

```
switch(NUM0) {  
  case NUM1 : stmt1  
  case NUM2 : stmt2  
  default  : stmt3  
}
```

Listing 2: Code intermédiaire

```
goto _L0  
_L2  
//stmt1 traduction  
goto _L1  
_L3  
//stmt2 traduction  
goto _L1  
_L0  
if NUM0 == NUM1 goto _L2  
if NUM0 == NUM2 goto _L3  
_L1
```

Cette première implémentation est à faire dans le visiteur `IntermediateCodeGenVisitor`.

2.2 Code intermédiaire avec fall-through

Dans un deuxième temps, vous devrez étendre votre code pour couvrir le fall-through (optimisation du code intermédiaire pour les booléens). Le code est aussi dans le document de référence.

Cette réduction est aussi appelé élimination des GOTO redondants.

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. Le livre du dragon vous fournira pas mal d'explication sur les différentes choses à implémenter, mais **je vous suggère de vous référez principalement au document "SDD pour la generation du code intermediaire (v3)"** sur moodle, section 3, "style fall through".

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.

Cette première implémentation est à faire dans le visiteur `IntermediateCodeGenFallVisitor`. Vous pouvez copier votre code de `IntermediateCodeGenVisitor` et repartir de là où utiliser l'héritage Java. Travaillez bien dans un deuxième fichier, en gardant le premier intact !

3 Barème

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

- Code intermédiaire pour les expressions arithmétiques : 4 points
- Code intermédiaire pour les expressions booléennes : 4 points
- Code intermédiaire pour les statements conditionnelles : 4 points
- Code intermédiaire pour les switches : 3 points
- Code intermédiaire avec Fall-Through : 5 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.

4 Remise

Le devoir doit être fait en **binôme**. Remettez sur Moodle une archive nommée *log3210-tp3-matricule1-matricule2.zip* avec uniquement :

- le fichier `IntermediateCodeGenVisitor.java`;
- le fichier `IntermediateCodeGenFallVisitor.java`;
- un fichier `README.md` contenant vos commentaires concernant le projet si nécessaire (facultatif).

L'échéance pour la remise est le **29 Février 2020 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, fichiers à rendre).

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