Projet : Le compilateur While

L'objectif du projet de compilation est de concevoir un compilateur pour le langage While. Les spécifications de ce langage sont fournies dans un document annexe. Ce compilateur devra traduire le langage While en un langage cible de votre choix et au final générer un programme exécutable.

Les différentes étapes de conception

Les grandes étapes de la conception de votre compilateur seront les suivantes :

1. Décrire la grammaire de While en ANTLR
2. Créer, par l'intermédiaire des fonctionnalités d'ANTLR un arbre de syntaxe abstraite (AST) associé à votre langage. Vous veillerez à spécifier clairement la structure de votre arbre, cela vous sera utile lors des diverses passes d'analyse / génération.
3. En travaillant sur l'AST, il s'agira ensuite d'effectuer l'analyse sémantique du programme, incluant :
    a. la construction de la table des symboles,
    b. la validation du programme (typage, pas de définition multiple…).
4. Toujours à partir de l'AST, il faudra traduire chaque fonction en code 3 adresses. Ce sera à vous de spécifier votre code 3 adresses sachant que While est un langage assez simple qui ne nécessite pas forcément un code 3 adresses aussi complet que pour des langages de programmation plus évolués.
5. Ecrire le backend qui a pour rôle de réécrire les fonctions et le code 3 adresses qui leur est associé dans le langage cible et générer le / les fichiers en résultant.
6. Si le langage cible est un langage compilé (C, C++, java…), il faudra concevoir un script permettant d'enchainer le compilateur While avec le compilateur du langage cible de manière à générer un exécutable.

Bien sûr le compilateur s'appuiera sur l'existence de certaines fonctions lors du passage de l'AST au code 3 adresses ou encore pour la gestion du backend. Il faudra donc concevoir une bibliothèque « runtime » pour votre langage. Cette bibliothèque sera conçue dans le langage cible et inclura toutes les fonctionnalités nécessaires à l'exécution de votre programme ainsi que le pretty printer affichant les valeurs de retour du programme principal (voir document de spécification de While).

Un objectif optionnel est de travailler sur le code 3 adresses de manière à optimiser le code généré.

Informations, conseils et points de vigilance

Parcours de l'AST et analyse sémantique

La documentation de la bibliothèque « runtime » java de ANTLR est consultable à cette adresse : Overview (ANTLR 3 Runtime 3.5.3 API). L'interface correspondant à un nœud de l'AST est fournie ici : Tree (ANTLR 3 Runtime 3.5.3 API).

Il vous est conseillé d'utiliser la notion d'attributs pour faciliter l'analyse sémantique. Ces attributs peuvent être associés à des nœuds de l'AST et vous permettre de transmettre / stocker de l'information lors du parcours de ce dernier. Plusieurs passes peuvent aussi collaborer entre-elles par ce biais : certaines génèrent des attributs, d'autres utilisent les attributs préalablement générés par exemple. Cette association peut être externe (par exemple une table de correspondance entre une référence sur Tree et l'attribut) ou encore par la création de votre propre classe de nœud de l'AST et son utilisation dans le parser (voir Interfacing AST with Java - ANTLR 3 - Confluence (atlassian.net)).

Enfin, les traitements à effectuer sont souvent dépendants de la nature du nœud de l'AST actuellement traité. Il est intéressant ici de réfléchir à un patron de conception de type visiteur adapté à la structure de l'AST fourni par ANTLR. Ce patron de conception, s'il est bien fait, peut vous permettre de très rapidement écrire de nouvelles passes d'analyse sur l'AST. Couplé avec les attributs, cette technique est extrêmement efficace en termes de développement.

Validation du programme While

La validation du programme consiste à prouver, avant la génération du code intermédiaire que le programme est correct. Cette validation consiste principalement à éviter les définitions multiples d'une même fonction et à vérifier que toutes les expressions sont bien typées. Prenez bien soin de regarder les spécifications du langage While ainsi que sa syntaxe. A partir de cette analyse, vous devriez être à même de déterminer tous les endroits où les expressions peuvent être mal typées.

Pour tester cette étape, vous prendrez soin de concevoir des petits exemples pertinents permettant de tester tous les cas de rejet ainsi que les cas d'acceptation. Ces exemples peuvent être conçus avant la réalisation et utilisés ensuite pour tester et valider vos avancements.

Traduction des fonctions en code 3 adresses

Cette étape doit être systématique et la plus simple possible à implémenter. Ici encore, penser à bien spécifier vos schémas de traduction de manière à vous simplifier la vie. A titre d'exemple, l'enchainement de deux commandes « C1 **;** C2 », revient à concaténer le code 3 adresses de « C1 » avec le code 3 adresses de « C2 ». Votre arbre de syntaxe abstraite met (normalement…) en évidence toutes les constructions du langage. Donc pour chaque nœud de l'AST, vous devez être capable de construire le code 3 adresses à partir de la nature de ce nœud et du code 3 adresses généré pour les nœuds fils.

Un point de vigilance se situe sur les fonctions à plusieurs valeurs de retour. Réfléchissez bien à la manière de traiter de ce problème.

Enfin, réfléchissez bien à la manière de traiter des arbres binaires et des fonctionnalités mises à disposition par le langage : **cons**, **list**, **hd**, **tl**…

Langage cible pour le backend

Lorsque vous décrivez votre programme sous la forme de code 3 adresses, vous utilisez des commandes qui effectuent des sauts (jump / goto par exemple). Certains des langages que vous pouvez choisir comme langage cible peuvent ne pas mettre de commande équivalente à goto à disposition du programmeur. Pas de panique, si c'est le cas, il vous suffit de générer un automate dans le langage cible et ce dernier vous permettra de réaliser la traduction sans souci.

Un peu de méthodologie

De l'importance du test

Pour vérifier / valider votre avancement à chaque étape de la conception de votre compilateur, écrivez des tests en langage While. Ces tests doivent être représentatifs du langage et de ce qu'il permet de faire (programmes qui doivent compiler) ainsi que représentatif des erreurs que les programmeurs peuvent commettre (programmes qui ne doivent pas compiler). Ces programmes de test doivent, pour la plupart, être petits afin de vous permettre de tester une fonctionnalité ciblée de votre compilateur. Ces tests peuvent être conçus à priori, sans même avoir encore décrit la grammaire de While en ANTLR…

Ces petits tests peuvent être utilisés / réutilisés à plusieurs moments dans votre cycle de développement : tester votre grammaire et donc l'analyse syntaxique, tester la validation du programme (typage etc…), tester la génération de code 3 adresses, tester le backend…

Il ne faut surtout pas négliger ces petits programmes de tests, ils sont d'une aide précieuse dans votre développement et vous permettront de rapidement valider des fonctionnalités et / ou mettre rapidement en évidence des bugs dans votre compilateur.

Commencer petit

Lors de la conception des différentes phases de génération, commencez petit. Mettez en place des fonctionnalités minimales, testez-les sur les petits programmes de test puis une fois les fonctionnalités validées, programmez des fonctionnalités plus avancées. L'idée maitresse ici est de ne pas concevoir l'intégralité du compilateur à priori pour à la fin de se rendre compte que rien ne fonctionne…

A titre d'exemple, il n'est peut-être pas nécessaire de gérer la compilation des fonctions à plusieurs valeurs de retour dès le départ, les structures de contrôle sont probablement beaucoup plus importantes dans un premier temps… De même, l'introduction de « nop » dans les mots clés du langage est là pour faciliter vos tests, vous pouvez rapidement tester la compilation d'une fonction qui ne fait rien en gérant ce mot clé...

Découpez le projet en tâches, organisez-vous…

La réalisation de ce projet nécessite de concevoir plusieurs outils logiciels, de réaliser des tests etc… Tous ces éléments ne sont pas forcément interdépendants en termes de développement. Essayer de lister les différentes tâches que vous allez avoir à réaliser pour mener à bien ce projet. Explicitez les dépendances entre ces dernières. A partir de ce travail vous pourrez alors facilement paralléliser la réalisation du projet en affectant des tâches à chaque sous équipe.