RAPPORT DE FIN DE PROJET
Compilateur de WHILE en C++

Nous attestons que ce travail est original, qu'il indique de façon appropriée tous les emprunts, et qu'il fait référence de façon appropriée à chaque source utilisée.

GICQUEL Antoine, CHAPDELAINE Julie, DAGUIN Cléo,
LAAFOUDI Mohammed Amine,TESTIER Thomas, CHIHABY Yasser

ESIR 2 - SI
26/01/2020

ECOLE SUPERIEURE D'INGENIEURS DE RENNES

UNIVERSITÉ DE RENNES 1

INTRODUCTION

Durant notre cours de compilation, nous avons réalisé un compilateur complet par groupe de six personnes. Le langage source était du code WHILE et nous avions le choix du langage cible. Ainsi, pour notre projet, nous avons décidé d'utiliser le langage C++ comme langage cible. Par ailleurs, nous avons adopté une méthodologie agile consistant en trois sprints ayant des objectifs précis. Nous allons expliquer au travers de ce document la réalisation et les aboutissants de chacun des sprints.

Bibliographie

- "Compilateurs : principes, techniques et outils" par Aho, R. Sethi et J. Ullman
- "Implementing Domain-Specific Languages with XText and XTend" par Lorenzo Bettini
- "Programming language pragmatics" par M. Scott

Tables des matières

Sprint #1 : Pretty-Printer

Objectif

Le premier sprint avait pour objectif principal de s'approprier l'outil XText en réalisant un pretty-printer WHILE. Ce programme permet d'indenter correctement un fichier source WHILE selon les conventions préalablement paramétrées par l'utilisateur.

Approche

Avant de démarrer la phase de développement, nous avons préféré commencer par se documenter afin de mieux comprendre le fonctionnement et les possibilités offertes par XText. Nous avons donc étudié la documentation de référence XText : "Implementing Domain-Specific Languages" de Lorenzo Bettini.

Nous avons ensuite divisé le développement du pretty-printer en deux phases :

- Analyse lexicale et syntaxique à l'aide du fichier de langage dédié du langage WHILE (fichier Domain-specific language).
- Parcours de l'arbre de syntaxe abstraite issu de l'interprétation du DSL par XText.

Réalisation

Pour la suite du développement nous avons fait le choix d'utiliser le langage XTend qui fait partie intégrante du projet XText. Ce langage est particulièrement adapté à nos problématiques de compilation, notamment avec ses fonctionnalités suivantes :

- Dynamic dispatch : sélection de l'implémentation de la méthode durant l'exécution selon le type de l'objet.
- Extension methods : permet d'alléger l'écriture du code en ajoutant des méthodes à des classes existantes sans les modifier.
- Intercompatibilité avec Java.

XText génère un arbre de syntaxe abstraite constitué d'objets générés à partir du

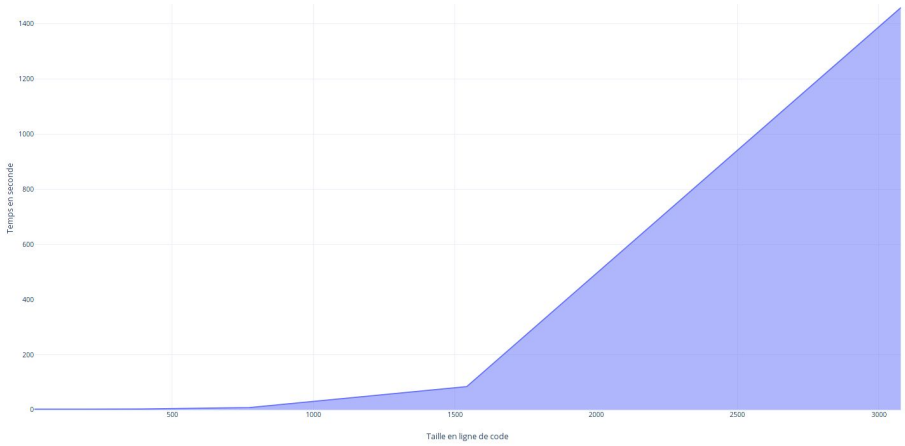
DSL. Nous avons donc pu aisément parcourir l'arbre de syntaxe abstraite en entendant chacune des classes issues du DSL avec l'extension des méthodes. Nous avons créé des méthodes avec le même nom adaptées à l'indentation des éléments syntaxiques puis parcouru l'ASA récursivement en appelant ces méthodes avec le dynamic dispatch.

Procédure de test

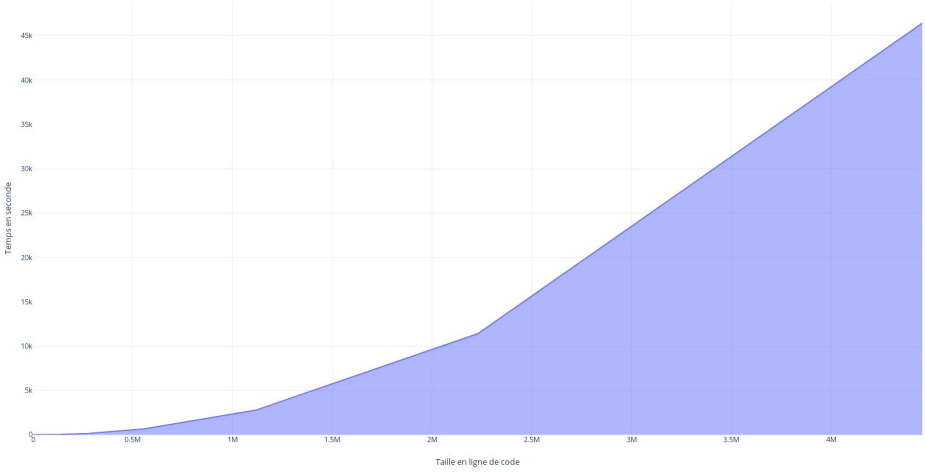
Notre procédure de test était un script shell qui prenait un code WHILE mal indenté puis comparait le résultat du pretty-printer avec un oracle de test afin de valider le test pour chacun des éléments syntaxiques.

Nous avons également fait des tests de complexités :

- Complexité verticale



- Complexité horizontale :



Sprint #2 : Génération de code intermédiaire

Objectif

Ce sprint est étroitement lié au troisième puisqu'ils concernent tous les deux directement le coeur du projet : le compilateur WHILE. Ce sprint aura également été notre premier pas vers la compilation plus concrète.

Ce second sprint est tourné vers l'analyse syntaxique et sémantique du code source afin de réaliser ces deux tâches :

- Vérification sémantique du code source : XText permet uniquement de vérifier si les sources sont syntaxiquement correctes. Néanmoins il existe des erreurs de programmation nécessitant davantage d'analyse.
- Création d'un code intermédiaire : Ce code pourra par la suite être optimisé afin de générer tout type de code. Dans le cadre du projet, nous ferons simplement une génération naïve vers du code C++ durant le sprint #3.

Réalisation

La première étape de ce sprint a donc été d'identifier les problèmes sémantiquement faux mais syntaxiquement corrects dans un programme WHILE. En effet, nous nous sommes aperçus pendant la réalisation du premier sprint sur le pretty-printer qu'il était possible par exemple d'avoir deux fonctions avec le même nom dans un seul fichier WHILE. Ce qui n'est en réalité pas possible. Il nous a donc fallu modifier le validator XText afin de l'adapter à nos besoins. Le validator est un fichier généré automatiquement par XText permettant d'ajouter des règles sémantiques. Il est appelé automatiquement à chaque modification de fichier et nous a ainsi permis de vérifier que chaque fonction a un nom différent ou encore lors d'une affectation multiple, que le nombre de variables à gauche soit bien le même que le nombre de retours de fonctions ou d'affectations à droite.

Concernant la génération de code intermédiaire, nous avons fait les choix techniques suivants :

- Utilisation de pile : Nous utilisons une pile pour toutes les affectations de variables ainsi que pour le passage des paramètres et la récupération des valeurs de retours de fonction. De plus, nous savions qu'un mécanisme de pile est facilement implémentable pour un langage cible.
- Utilisation de GOTO : Nous avons fait le choix d'utiliser des GOTO dans notre représentation intermédiaire alors que nous aurions pu utiliser les structures de données haut niveau pour représenter les embranchements conditionnels. Il n'y a pas de réelle raison à cela mais ce choix ne sera pas préjudiciable car C++ possède les mécanismes d'étiquette et de saut.

La représentation intermédiaire est essentiellement constituée d'une table des symboles et d'une suite de quadruplets. La table des symboles contient une abstraction des variables, symboles, étiquettes et fonctions nécessaires au code prochainement généré. Les quadruplets quant à eux permettent de représenter les codes trois adresses.

La génération de cette représentation intermédiaire se fait de façon similaire au parcours du code du pretty-printer. C'est à dire que nous allons parcourir l'arbre de syntaxe abstraite de la même manière mais en faisant davantage de distinction entre les expressions (si elles sont booléennes ou non). De ce fait, toutes les expressions héritent grâce aux paramètres effectifs les étiquettes si son expression est évaluée comme vraie ou fausse. Au terme de cette construction, nous obtenons une liste de quadruplets et une table de symbole pour chacune des fonctions du code source WHILE.

Sprint #3 : Génération de code exécutable

Objectif

Ce dernier sprint avait pour but de générer du code C++ à partir de la représentation intermédiaire précédemment créée. Nous avons conçu une génération naïve de code cible, il est donc normal de générer un code qui soit lourd dans sa conception et gourmand en usage de variable.

Il a également fallu concevoir et développer la libraire WHILE en C++. Celle-ci comporte donc une partie contenant les arbres binaires selon la sémantique du langage WHILE ainsi qu'une partie bootstrap permettant de lancer le programme et la bonne lecture des paramètres d'entrée.

Réalisation

Les arbres binaires sont des structures immuables, il est donc judicieux d'utiliser les pointeurs pour gérer les fils de l'arbre. La valeur de chacun des noeuds est enregistrée dans une chaîne de caractères. Afin de simplifier grandement l'usage et la mémoire des arbres binaires, les objets sont tous passés par pointeur intelligent. En effet, la librairie standard possède des pointeurs intelligents qui comptent le nombre de références et se détruisent lorsqu'ils ne sont plus référencés par aucun objet.

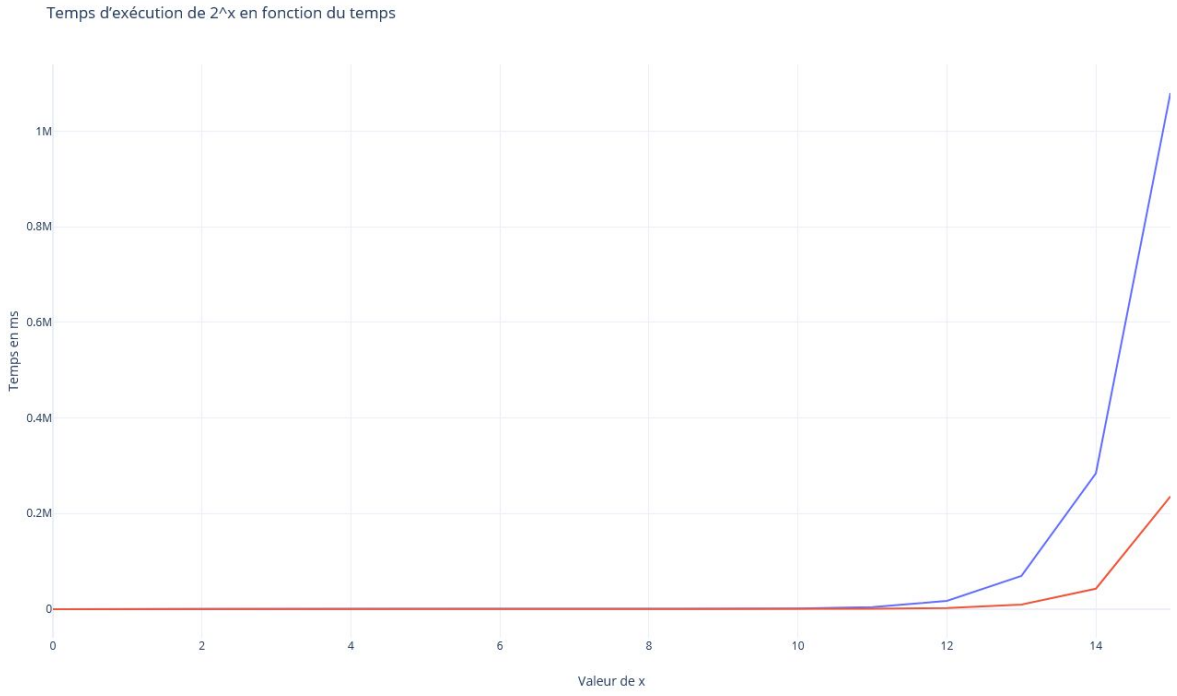
Le code source WHILE doit contenir une fonction nommée main. Cette fonction sera la fonction appelée par la partie bootstrap de la librairie WHILE en C++. La lecture des paramètres ne peut être faite avec des regex car un arbre binaire sous forme textuelle peut être récursif. Nous avons donc utilisé une librairie permettant d'écrire une rapide grammaire et la possibilité d'évaluer l'arbre de syntaxe abstraite.

De ce fait, la librairie C++ est contenue dans plusieurs fichiers headers et devient facile à inclure lors de la compilation. Le code C++ généré par le compilateur contient une ligne *include* pour notre libraire, les signatures des fonctions générées puis la fonction *main* appelant le *bootstrap* de la librairie et enfin le code des fonctions compilées.

Mesures de performance

Nous avons réalisé un test de fuite mémoire sur notre librairie à l'aide de l'outil *Valgrind*. Celui-ci n'a détecté aucune fuite et toutes les allocations sont bien désallouées.

Dans un second temps nous avons fait des mesures de performance sur un programme écrit en WHILE puis compilé. Lorsque nous compilons notre code C++ en code binaire, nous avons pu appliquer une phase d'optimisation du code généré qui améliore grandement les performances. Le graphique ci-dessous illustre le temps d'exécution du code WHILE pow 2 i pour i allant de 0 à 15. La courbe bleu est celle du programme sans optimisation et la courbe rouge est celle avec l'optimisation. On peut constater que les performances sont nettement meilleures.



De plus, les arbres binaires sont immuables donc nous aurions pu créer des arbres *nil* et *cons(nil, nil)* partagés par tous les autres arbres binaires. Cela aurait optimisé la consommation de mémoire et le nombre d'instances créées.

BILAN DE GESTION DE PROJET

Étapes du développement

Pour la réalisation de ce projet nous avons adopté une méthodologie agile. Ce projet a en effet été divisé en 3 sprints et durant la totalité du projet nous avions des échanges quotidiens avec le “client” afin de lui faire part de l’évolution du projet.

Pour commencer, nous avons procédé à une répartition des tâches et à la mise en place d’un git afin de faciliter la collaboration entre développeurs et d'être au courant des modifications apportées tout au long du projet. Nous nous sommes par la suite documentés afin d’avoir une meilleure compréhension de XText, du langage WHILE et de leurs fonctionnements.
Cela nous a permis d’entamer le sprint #1 dans les meilleures conditions. Cependant lors de ce sprint nous nous sommes rendu compte que la répartition des tâches pouvait être améliorée. C’est ce que nous avons fait pour les sprints suivants, cela nous a donc permis de gagner en efficacité et par conséquent en temps. Grâce à cela nous avons même pu effectuer quelques optimisations.

Rapport d’activité individuel

CHAPDELAINE Julie

Tout au long de ce projet, j’ai eu le rôle de développeur. N’ayant pas fait le module de Théorie des Langages, j’ai eu beaucoup de mal au début du projet à appréhender XText ainsi que son fonctionnement, j’ai donc passé plus de temps que les autres à me documenter. Cependant, lors du second sprint, je comprenais déjà mieux son fonctionnement, et j’ai donc pu aider les autres, notamment sur la génération d’erreurs. Lors du troisième sprint, j’ai pu aider également sur la création de la librairie WHILE en C++, et enfin à la rédaction de ce rapport.

DAGUIN Cléo

Mon rôle était principalement tourné vers la documentation. Au cours de ce

projet j'ai mis en place le -help du pretty-printer et du compilateur mais j'ai également construit le diaporama pour chaque présentation au "client" et mis en place le rapport de fin de projet. J'ai également aidé au développement dans une moindre mesure en développant quelques fonctions de la librairie en c++ ou en concevant une partie du schéma de traduction par exemple, généralement en binôme avec Julie et sur la demande d'Antoine.

CHIHABY Yasser

Dans un premier temps je me suis documenté afin de mieux maîtriser XText, XTend et mieux comprendre le langage WHILE. Ainsi j'ai pu, avec Amine et Thomas, réaliser des tests tout au long du projet afin de tester chaque fonctionnalité et partie du projet. J'ai aussi aidé à certaines parties qui comprennent de la rédaction ou du développement. En effet J'ai pu aider avec Amine à la réalisation de la Librairie WHILE. Enfin, j'ai aussi participé à la rédaction de ce rapport.

GICQUEL Antoine : Chef de projet et expert technique

Mes missions au sein de projet furent multiples. En effet en tant que chef de projet, je m'occupais de répartir les tâches entre les autres membres et de faire attention au respect des délais durant les différents sprints. De plus, j'étais le principal interlocuteur avec le client et présentait l'avancement du développement pendant les séances de TD. Également en tant qu'expert technique, j'ai été le principal contributeur technique du projet et j'aidais les membres ayant besoin d'aide.

LAAFOUDI Mohammed Amine

Pour ce projet de compilation, j'ai eu plusieurs tâches. Dans un premier temps, j'étais assigné à la conception et à la documentation, afin de récolter les différentes informations qui vont servir tout au long du projet (par exemple le choix des outils). Yasser et moi avons également participé à la réalisation de la librairie WHILE. Enfin j'ai pu, avec Yasser et Thomas, réaliser des tests afin de vérifier le bon fonctionnement du projet. J'ai également participé à la rédaction de ce rapport.

TESTIER Thomas : Responsable des tests

Avec Yasser et Amine, nous nous sommes occupés des tests. Nous avons donc durant tout le projet réalisé de nombreux oracles de tests afin de vérifier l'avancement du projet. Nous nous sommes séparés les tâches afin de pouvoir tester chaque fonctionnalité et chaque particularité de ce projet, pour finir sur des tests bien plus globaux comprenant des programmes plus complexes.

J'ai cependant pu aider mes autres camarades de façon ponctuelle lors de la rédaction ou du codage de certains points demandant plus de temps et de réflexion.

Conclusion

Ce projet s'est révélé très enrichissant dans la mesure où il a consisté en une approche pragmatique du métier d'ingénieur. En effet, la prise d'initiative, le respect des délais et le travail en équipe seront des aspects essentiels de notre futur métier. Nous avons appliqué la plupart des consignes de la méthodologie Agile qui a porté ses fruits. En effet, le compilateur est fini dans les délais prévus avec toutes les exigences demandées par le client.
Finalement, travailler en équipe sur le projet du compilateur nous a permis de monter en compétences sur tous les aspects du projet.