École Polytech de Clermont-Ferrand
Campus des Cézeaux 2 avenue Blaise Pascal
TSA 60206 - CS 60026
63178 AUBIERE Cedex

MINES ParisTech
Centre de **R**echerche en **I**nformatique
35 rue St Honoré
77300 FONTAINEBLEAU

Rapport de stage
4 ème année

Filière : Génie Mathématique et Modélisation.

Participation au projet FEEVER sur le développement du langage musical Faust

Présenté par : **Zouhair MORTABIT**

Responsable Mines : **Pierre JOUVELOT**

Date de soutenance : 31 Août 2016 .

Remerciements

Je tiens à remercier tout d'abord mon tuteur de stage, M. Pierre JOUVELOT, Maître de recherche au Centre de recherche en informatique (CRI) de MINES ParisTech (Ecole nationale supérieure des mines de Paris), ainsi que M. Emilio JESÚS GALLEGO ARIAS et M. Yann ORLAREY, pour l'intérêt porté à mon projet, le suivi et les conseils procurés tout au long de mon stage.

Je tiens également à faire part de ma reconnaissance à l'ensemble des employés du site de Fontainbleau, ingénieurs, chercheurs et autres, pour leur accueil chaleureux et pour avoir été toujours présents pour répondre à mes interrogations et me fournir toutes les explications nécessaires à la compréhension du travail que je devais accomplir.

Résumé

Dans notre monde numérique, via lecteurs MP3, diffusion de radios ou cinémas Surround Sound, la musique, et plus généralement le traitement audio, affecte tout un chacun, où qu'il soit et de façon toujours plus personnalisée. Le projet FEEVER vise à fournir les technologies qui permettront que le monde numérique puisse chanter, en naviguant sur Internet, ou en écoutant une émission de radio en déplacement. Parmi les tâches que le projet FEEVER propose, l'une consiste à fournir un compilateur Faust intégrable, portable, multi-plateforme, efficace et de qualité industrielle. Faust (Functional Audio Stream) est un langage de programmation spécifiquement conçu pour les applications musicales. Il permet de développer rapidement des applications efficaces et fiables pour les principales plateformes audios. La version actuelle traduit un programme Faust qui décrit un processeur de signaux sous la forme d'une fonction qui prend un ensemble de signaux en entrée et produit un ensemble de signaux en sortis sous la forme d'une classe C++.

L'objectif de mon stage a été de regarder la manière dont travaille le compilateur Faust, écrit en C++, et de générer une structure représentative pour un autre langage qui est développé au CRI (centre de recherche en informatique) appelé Wagner. Il s'agit d'un langage de traitement du signal qui peut être vu comme un simple langage synchrone fonctionnel, permettant un codage naturel de filtres et d'oscillateurs numériques de guide d'ondes. Il fallait que ce compilateur respecte pour son développement les critères convenus avec le responsable de stage, ainsi de permettre de manipuler chaque type d'expression du langage Wagner séparément tout en rendant le compilateur plus optimisé.

Mots clés : Informatique, musique, compilateur, Faust, Wagner, C++.

Abstract

In our digital world, from portable MP3 players to radio streaming to Surround Sound-equipped movie theaters, music and more generally audio processing is a life-enhancing practice that impacts everyone everywhere in evermore personalized manners. The FEEVER project aims at providing the key technologies that will make the whole digital world sing, be it when web surfing at home or listening to a radio stream on the go. One of the tasks the FEEVER project intends to make is to provide an industrial-strength, efficient, multi-rate, multi-platform, portable, easily integrable faust compiler.
FAUST (Functional Audio Stream) is a functional programming language for implementing signal processing algorithms in the form of libraries, audio plug-ins, or standalone applications. The current version translates a Faust program describing a processor of signals in the form of a function that takes a set of signals as inputs and produces a set of signals in the output as a C++ class.

The objective of my intership was to look at the way how works the Faust compiler wrote in C++, and to generate code for a language under developing here in the CRI (centre de recherche en informatique) called Wagner. So this language of signal processing can be seen as an intermediate language for Faust.
One needed that this compiler respects, for its development, the norms agreed upon with the person in charge of this internship, thus to make it possible to work separately with each kind of expression of the Wagner language while making the compiler more optimized.

Keywords : Computer science, music, compiler, Faust, Wagner, C++.

Table des matières

Chapitre 1

Introduction

Dans le cadre du cursus d'ingénieur à Polytech de Clermont-Ferrand, j'ai eu le plaisir d'effectuer mon stage de quatrième année au sein du Centre de recherche en informatique (CRI MINES ParisTech) à Fontainebleau du 30 mai au 25 août 2016.
L'École nationale supérieure des mines de Paris connue aussi sous le nom de MINES Paritech est l'une des 210 écoles d'ingénieurs habilitées à délivrer un diplôme d'ingénieur en France.
Dans le cadre de mon stage, j'ai effectué un travail en collaboration avec les membres du centre qui travaillent dans le cadre du projet " FEEVER " sur un langage musical qui s'appelle " Faust ", dans le but de créer un compilateur pour générer une représentation intermédiaire pour un autre langage (Wagner) qui est développé dans le cadre du même projet.
Ce qui a été la source de ma motivation pour ce stage, c'est l'opportunité de travailler au sein d'un grand centre de recherche et développement informatique et de livrer un produit dans un cadre scientifique. En plus de constituer pour moi une première expérience en informatique, ce stage m'a aidé à développer de nombreuses compétences relationnelles. En effet, j'ai été amené à effectuer des réunions et présentations et à être en dialogue permanent avec les employés du centre tout au long de ce travail.
J'ai également amélioré mon autonomie et ma faculté d'adaptation en apprenant rapidement nouvelles notions de compilation et en réfléchissant à des solutions d'implémentations adaptées au besoin .
Dans une première partie de ce rapport, je présenterai le centre et le cadre professionnel dans lequel j'ai travaillé, puis j'aborderai dans une deuxième partie le sujet du projet de développement, en rappelant d'abord le contexte qui l'a motivé puis les fonctionnalités demandées.
Enfin, dans une dernière partie, j'évoquerai les difficultés que j'ai rencontrées ainsi que les bénéfices que j'ai pu tirer de ce stage.

Chapitre 1

Présentation de l'établissement d'accueil

1.1 L'École national supérieur des mines de Paris

L'École nationale supérieure des mines de Paris, aussi connue sous le nom de MINES ParisTech ou École des mines de Paris, est une grande école française et l'une des 210 écoles d'ingénieurs habilitées à délivrer un diplôme d'ingénieur en France.
MINES ParisTech forme depuis sa création en 1783 des ingénieurs de très haut niveau capables de résoudre des problèmes complexes dans des champs très variés.

Première école d'ingénieurs en France par son volume de recherche contractuelle, MINES ParisTech dispense une importante activité de recherche orientée vers l'industrie. Ses domaines de recherche s'étendent de l'énergétique aux matériaux, en passant par les mathématiques appliquées, les géosciences et les sciences économiques et sociales. L'école d'ingénieurs développe également la création de chaires d'enseignement et de recherche sur des thèmes émergents.

MINES ParisTech est membre fondateur de ParisTech qui rassemble 12 des plus grandes écoles d'ingénieurs et de management parisiennes. L'Ecole est également membre de la COMUE Paris Sciences et Lettres qui rassemble 16 institutions d'enseignement supérieur et de recherche prestigieuses, situées au cœur de Paris.

L'École a été fondée en 1783, à l'époque où l'exploitation des mines était l'industrie de haute technologie par excellence et concentrait les problèmes de sécurité des personnels et de planification économique, voire les enjeux géopolitiques (l'accès aux matières premières rares ou stratégiques). Tout naturellement, les compétences de l'École ont suivi le développement de l'industrie et l'École étudie, développe et enseigne aujourd'hui l'ensemble des techniques utiles aux ingénieurs, y compris les sciences économiques et sociales.

Ainsi, MINES ParisTech dispense des formations d'excellence par la recherche marquée culture ingénieur, alliant haut niveau scientifique et pragmatisme économique, avec :

150 ingénieurs civils diplômés ;
200 diplômés des 13 Mastères spécialisés ;
20 ingénieurs du Corps des mines ;
100 nouveaux docteurs diplômés ;
150 stagiaires en formation continue ;
par an.

Elle assure une excellence académique au niveau de la recherche scientifique

2 Prix Nobel ;
233 enseignants-chercheurs ;
18 centres de recherche ;
100 thèses soutenues par an ;
400 articles ou livres publiés chaque année ;
la création de disciplines nouvelles : géostatistique, morphologie mathématique, systèmes plats.

1.2 Le Centre de recherche en informatique (CRI)

MINES Paristech, première école en France par son volume de recherche contractuelle, dispense une importante activité de recherche orientée vers l'industrie. La recherche est structurée autour de cinq grandes thématiques, correspondant à cinq départements d'enseignement et de recherche. Chaque département est organisé en centres de recherche, qui incluent les mathématiques appliquées et l'informatique.

Le Centre de recherche en informatique(CRI) situé à 35, rue Saint Honoré 77305 Fontainebleau, fait partie des 18 centre de recherche de l'école. Il a comme membres :
Un directeur ;
Un directeur adjoint ;
Une secrétaire ;
6 enseignements chercheurs ;
3 ingénieures de recherche ;
5 doctorants ;
1 post-docs ;

Le centre se consacre à l'étude des :

«Langages utilisés par les technologies de l'information (langages de programmation, de description de données, d'interrogation ou semi-formels voire naturels) et

développe des techniques d'analyse sémantique et de transformation automatiques destinées à répondre aux besoins industriels (performance, coût de développement, time-to-market) et aux besoins administratifs et sociétaux (partage d'information cohérente, normalisation des données, accès à l'information, sauvegarde du patrimoine)» [1].

Ainsi, certains doctorats dans le laboratoire sont en partenariat avec des entreprises, ce qui constitue une véritable expérience professionnelle et permet au doctorant d'acquérir non seulement des compétences scientifiques dans des domaines multidisciplinaires mais aussi de développer sa connaissance du monde économique.

1.3 Le projet FEEVER

« Le projet FEEVER, cofinancé par l'ANR et labellisé par le pôle de compétitivité Imaginove, vise a développer une solution globale et ubiquitaire pour le traitement du signal audionumérique sur la plupart des plates-formes, y compris le Web.
Dans le future un ingénieur pourrait concevoir un algorithme de réverbération et en poster l'implémentation sur Internet. Un auditeur pourrait l'utiliser en naviguant sur le site correspondant, ce qui aurait pour effet de télécharger et activer automatiquement le code de traitement du son correspondant dans son navigateur, ou même de réorienter son flux audio vers ce site pour être traité à distance avant écoute. Si ce site peut être commandé à distance, cela permettrait d'envisager des installations encore plus riches sur le plan acoustique, dans lesquelles plusieurs plateformes collaboreraient pour fournir à l'auditeur un environnement d'écoute dédié ; quelques touches sur une tablette, et les paramètres audio seraient adaptés à sa situation, par exemple en quittant sa maison pour s'installer dans l'environnement plus bruyant d'une voiture.
FEEVER a pour ambition de faire de cette vision une réalité. Certes, les défis abondent : les solutions doivent être (1) portables, pour offrir les gains qu'un modèle "programmer une fois, déployer partout" promet, (2) facilement programmable, pour diminuer l'écart entre spécification et implémentation, (3) adaptable à de multiples plateformes, pour une intégration fluide dans les environnements des auditeurs, (4) efficace en temps, puisque les traitements audio demandent de fortes puissances de calcul, et en énergie, ne serait-ce que pour les applications mobiles, et (5) sûre, puisque les activités effectuées sur le poste client ne doivent pas nuire à l'intégrité du système.
FEEVER s'appuie notamment sur la technologie Faust et son eco-système, développé

1. Site du CRI

depuis plus de dix ans à Grame un centre de recherche en informatique musicale situé à Lyon. Le projet va permettre des avancés scientifiques notamment vis à vis des modèles de calcul et des techniques de compilation utilisés. Faust, déjà enseigné dans plusieurs institutions renommées, et les technologies dérivées de FEEVER devraient conduire à de nouveaux modes d'enseignement plus interactifs et motivants, celant plus avant les collaborations interdisciplinaires entre les communautés scientifique et artistique.» [2]

2. Site du projet FEEVER

Chapitre 2

Projet réalisé

2.1 Le langage Faust

FAUST (Functional AUdio STream) est un langage de programmation fonctionnel développé par le GRAME (centre national de création musicale - Lyon - France) depuis 2004, conçu pour la synthèse de son en temps réel et le traitement du signal. Il vise à simplifier l'écriture de fonctionnalités de DSP (Digital Signal Processing) en fournissant une syntaxe simple mais expressive, qui convienne particulièrement bien aux ingénieurs du son et aux développeurs audio.

2.2 Concepts

FAUST combine un style fonctionnel et une syntaxe en block-diagram (schéma fonctionnel). Les signaux sont représentés comme des fonctions du temps, les processeurs de signaux comme des fonctions d'ordre supérieur qui traitent les signaux, et l'on utilise des opérateurs de composition pour combiner les processeurs. Grâce à cette syntaxe, le compilateur travaille sur une fonction mathématique décrivant le programme, et est capable de le représenter graphiquement sous la forme d'un block-diagram qui illustre son fonctionnement.

La version actuelle du compilateur permet de générer le code d'un programme FAUST dans divers langages tels que C, C++, Java ou Javascript. Cela permet la génération d'un code fortement optimisé. Sachant que l'efficacité du code résultant est un objectif principal du langage. Ainsi, FAUST est disponible sur un grand nombre de plateformes, n'a aucune dépendance et est facilement intégrable. Il fournit aussi une abstraction pour la création d'applications standalone avec interface utilisateur.

Un block-diagram (schéma fonctionnel) est une représentation d'un système en diagramme composé de blocs et de lignes qui relient les blocs entre eux. Ce type de diagramme est très utilisé en traitement du signal et permet de décrire des circuits de façon transparente : les blocs sont des processeurs de flux et les lignes relient les entrées et sorties des différents blocs.

La construction des block-diagrams propose une approche algébrique qui se base sur des opérateurs de composition : **séquentiel, parallèle et récursif**. Ces opérateurs permettent de chaîner les blocs avec une gestion fine des entrées et des sorties. Il devient très facile à partir de là de décrire des block-diagrams complexes de façon textuelle.

FAUST est basé sur cette syntaxe de block-diagram, et sur la programmation fonctionnelle. Un programme FAUST définit un processeur de signal, potentiellement composé de plusieurs sous-processeurs assemblés au moyen des opérateurs sus-cités. On peut faire un parallèle entre les processeurs et des fonctions d'ordre supérieur, les signaux correspondant à des fonctions du temps. Mais, en réalité, le programme travaille sur des flots de valeurs (les entrées du programme) et calcule ses valeurs de sortie à l'aide d'opérateurs appliqués sur les valeurs d'entrées et des valeurs précédemment calculées.

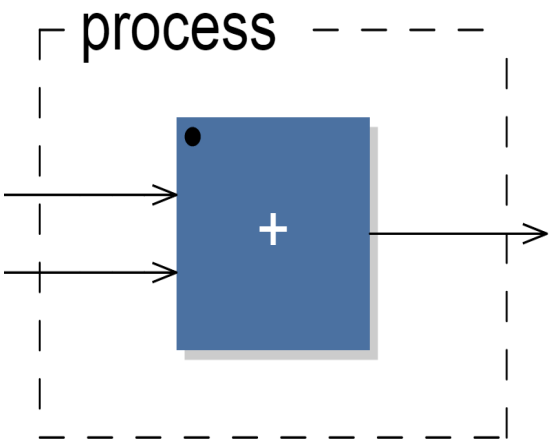
Le langage intègre certaines fonctionnalités pratiques pour le programmeur comme le fait de mélanger l'interface utilisateur avec le code de traitement. FAUST permet de créer des widgets pour contrôler différentes valeurs (par exemple le volume) qui seront eux aussi vus comme des processeurs émettant une valeur.

2.2.1 Exemples

Listing 1 : MyFaustProgram.dsp

```
process = +;
```

L'exemple ci-dessus est un exemple simple de code Faust qui calcule la somme de deux entrées.
le signal de sortie (Output signal) est $y(t) = x_1(t) + x_2(t)$
Avec x_i un signal d'entrée (Input signal) pour $i \in [1, 2]$

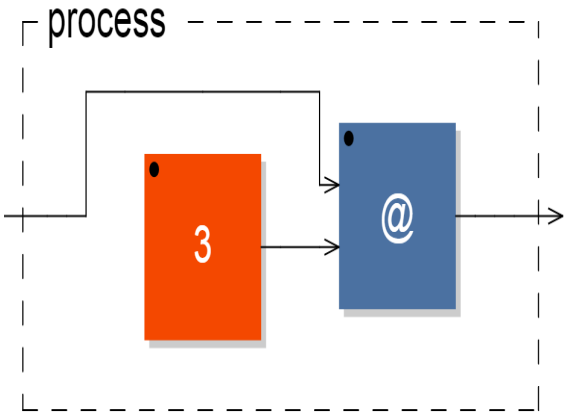


Le block diagram associé au process du Listing 1 .

Listing 2 : MyFaustProgram.dsp

```
process = @(3) ;
```

Cet exemple de code Faust retarde le premier signal d'entré par 3 .
le signal de sortie (Outout signal) est : $y(t) = x(t-3)$, avec x le signal d'entrée (Input signal)



Le block diagram associé au process du Listing 2 .

2.3 Le compilateur Faust

2.3.1 Intérêt d'un nouveau programme compilateur pour Faust

Dans le cadre de l'étude d'une extension multi-rate de Faust il est envisagé d'utiliser une nouvelle option pour la compilation d'un programme en Faust, qui génère une représentation en langage Wagner. Wagner peut être considérer comme un simple langage synchrone fonctionnel, permettant un codage naturel de filtres et d'oscillateurs numériques de guide d'ondes.

J'ai donc été amené à établir les spécifications fonctionnelles grâce à l'aide de mon responsable de stage, Mr Pierre Jouvelot, chercheur au centre et ainsi que celle de Monsieur Emilio Jesús Gallego Arias. Dans un premier temps, le premier travail que j'ai effectué a consisté à dialoguer avec les responsables afin de recueillir leurs besoins, puis à me renseigner sur la manière dont le compilateur travail et les structures de données utilisées.
La progression du développement du programme s'est faite par itération. En effet, je restais en contact régulier et permanent avec mon responsable de stage, qui m'indiquait à chaque fois les modifications et les ajouts qu'il fallait que j'apporte au programme en cours de développement.

2.3.2 Les outils utilisés

Linux
Dés le début de mon travail, mon responsable de stage m'a conseillé d'utiliser Linux comme système d'exploitation, parce qu'il est plus pratique pour le langage Faust et il permet une communication facile avec la machine à travers le terminal.
C++
Le compilateur de Faust était écrit en C++, donc j'ai continué à coder en C++.
Git
Comme mentionné plus haut, le stage était dans le cadre d'un projet, donc l'utilisation d'un logiciel de gestion de versions est devenu indispensable puisqu'on travaille à plusieurs sur un même projet et donc sur le même code source. À cet effet j'ai utilisé Git comme logiciel de gestion de version. Le Git est capable de suivre l'évolution d'un fichier texte ligne de code par ligne de code.

Git
Logiciel de gestion de version très puissant et récent, il a été créé par Linus Torvalds, qui est entre autres l'homme à l'origine de Linux. Il se distingue par sa rapidité et

sa gestion des branches qui permettent de développer en parallèle de nouvelles fonctionnalités.
Git est plus agréable à utiliser sous Linux et sensiblement plus rapide.
Une des particularités de Git, c'est l'existence de sites web collaboratifs basés sur Git, comme GitHub. GitHub, par exemple, est très connu et utilisé par de nombreux projets. C'est une sorte de réseau social pour développeurs : on peut regarder tous les projets évoluer et décider de participer à l'un d'entre eux si cela parait intéressant. On peut aussi y créer des propres projets : c'est gratuit pour les projets open source et il existe une version payante pour ceux qui l'utilisent pour des projets propriétaires.

Git propose les commandes suivantes :

git init : crée un nouveau dépôt.
git status : indique les fichiers modifiés récemment.
git add : ajoute de nouveaux objets modifiés dans la base des objets pour chaque fichier modifié depuis le dernier commit. Les objets précédents restent inchangés.
git commit : ajoute les dernières modifications à l'historique des modifications.
git push : envoi des commits au serveur utilisé (GitHub par exemple).

2.3.3 Les arbres syntaxiques

Un compilateur est un programme qui est chargé de traduire un programme écrit dans un langage dans un autre langage. Le langage du programme de départ est appelé le langage source ; le langage du programme résultat est appelé le langage cible (source language et target language, en anglais). le plus souvent, le langage source est un langage dit de haut niveau, avec des structures de contrôle et de données complexes, alors que le langage cible est du langage machine, directement exécutable par un processeur. Il y a cependant des exceptions ; par exemple, certains compilateurs traduisent des programmes d'un langage de haut niveau vers un autre. En pratique, le compilateur lit le programme et le transforme en un exécutable, c'est à dire quelque chose que la machine peut exécuter directement, disons une suite d'entiers que le processeur de la machine comprend comme des instructions.

Il y a trois étapes importantes pendant la compilation :

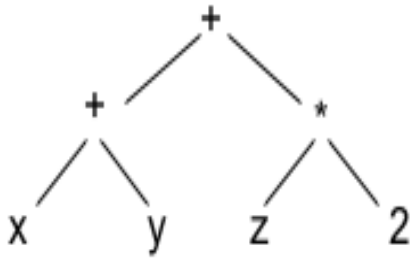
• l'analyse lexicale, qui découpe le code source en petits morceaux appelés jetons (tokens) ;
• l'analyse syntaxique, qui implique l'analyse de la séquence de jeton pour identifier la structure syntaxique du programme ;

• la génération de code, qui consiste à traduire les phrases produites par l'analyseur syntaxique dans le langage cible. (pour chaque construction qui peut apparaître dans une phrase, le générateur de code contient une manière de la traduire).

Un compilateur est en général un programme trop gros pour qu'il soit possible de le maîtriser d'un coup en peut de temps. L'objectif du travail était de changer la manière dont le compilateur génère le code pour qu'il donne une représentation intermédiaire des signaux afin d'utiliser langage Wagner, autrement dit, de travailler sur la partie de la génération du code sans toucher l'analyseur lexical .
le compilateur est divisé usuellement en parties distinctes qui jouent chacune un rôle particulier ; ce fait m'a facilité la tache, en ayant des morceaux plus facile à différer.

Pour bien comprendre, prenons un programme qui contient l'affectation d'un entier 123 à une variable x. Le compilateur va lire les caractères, les transformer en un entier machine, puis produire le code qui range l'entier dans l' adresse allouée à la variable x. Dès lors, à l'exécution le programme compilé se contente de ranger l'entier machine dans x, soit grosso-modo une seule instruction machine.

Exemple :
Bien évidement, des expressions plus complexes que 123 peuvent être utilisées, comme "$x+y+2*2$". Celle-ci sont représentés sous forme arborescente. (cf.figure ci-dessous)



L'arbre syntaxique de l'expression algébrique $(x + y) + (z * 2)$.

2.4 Compilateur Faust/Wagner

Comme mentionné avant, mon travail était de rajouter une partie de code dans le compilateur, pour qu'il y ait un autre moyen de compiler un programme Faust en obtenant une représentation intermédiaire pour le langage "Wagner". Tout au début, j'ai commencé par créer une sous classe que j'appelais "WagnerCompiler "

de la classe "compiler" déjà existait dans le code source. En parallèle, j'ai modifié le fichier "main.cpp" pour ajouter une nouvelle commande de compilation pour le langage Faust afin d'avoir le résultat souhaité en Wagner.

Commande	Langage cible
faust NomFichier.dsp	C++
faust -d NomFichier.dsp	FIR
faust -w NomFichier.dsp	Wagner

Commandes de compilation et les langages cibles associés .

Remarque 1. La branche de développement faust2 permet de générer du code dans différents langages cibles. Dans cette branche, un langage intermédiaire nommé FIR (Faust Imperative Representation) a été défini pour décrire les calculs réalisés sur les échantillons de manière générique. Le langage FIR peut être ensuite traduit en différents langages cible : C, C++, Java, JavaScript ou LLVM.

Ce programme, que j'ai conçu est implémenté parcourt l'ensemble des nœuds de l'arbre syntaxique d'un signal Faust de la classe Tree, de façon récursive. Une fonction générale est utilisée avec des conditions multiples "else if" qui appelle d'autres fonctions, selon le type du noeud rencontré. Cette fonction renvoie systématiquement le retour généré par le sous-arbre qu'elle analyse : cette façon de faire permet de faire les contrôles de type.

Par exemple, pour l'affectation d'une variable, nous lançons l'analyse sur le membre droit, et nous exécutons une fonction sémantique de contrôle de type, selon ce que l'analyse a renvoyé et le type du membre gauche. Si l'analyse détecte une erreur sémantique durant ce contrôle ou durant l'analyse du membre droit, une exception est lancée puis affichée, et le programme passe au sous-arbre suivant.

2.5 Types d'expressions pour le langage Wagner

Le programme contient une fonction générale qui manipule des chaînes de caractères(la bibliothèque String) pour l'impression des signaux. Dans un premier temps, pour chaque type de noeud rencontré, la fonction renvoie le signal sous forme d'une chaîne de caractère. Pendant une discussion usuelle sur l'état d'avancement de mon travail avec mon responsable de stage et Mr Emilio JESÚS GALLEGO ARIAS, post-doc au CRI, nous avons abordé le sujet des différents choix possibles pour but de permettre une plus grande pérennité du compilateur à développer. Suite à ce dialogue, l'idée de gérer les types d'expression s'est imposé, il s'agira de créer une classe pour chaque type d'expression en langage Wagner. Comme ça, la fonction générale renvoie un objet de ces classes crés.

Principalement la classe WagnerCompiler contient la fonction générale "ToWagnerExp" et la fonction "CompileMultiSignal". La première s'appelle d'une manière récursive pour permettre de descendre dans les sousarbres et faire les contrôles sémantiques de type ; elle utilise le " else if " pour le traitement des nœuds que nous pourrons rencontrer dans une expression. Pour la deuxième, -CompileMultiSignal-, elle permet d'afficher les résultats sur le terminal en, appelant la première. Je l'ai utilisée aussi pour tester les tables de hachage (Hashe table) que je citerai dans un paragraphe plus tard.

La classe wExp est une classe abstraite dont héritent les sous classes pour chaque expression Wagner.

```
class wExp {

public :

wExp() { };

virtual ostream  print(ostream  fout) const = 0;

};
```

De plus, pour chaque type d'expression de Wagner, existe une classe correspondante qui est une sous classe de la classe abstraite "wExp" .
Dans la fonctions générale

wExp* ToWagnerExp (Tree sig)

Se trouve le traitement de tous les noeuds qu'on peut rencontrer. Dans chaque cas, je fais appelle au constructeur de la sous classe correspondante .
Concernant les opérations arithmétiques, il s'agit toujours d'un noeud du nom de l'opération, avec deux sous-arbres pour chacun de ses membres. "binopn" est un tableau de caractères qui contient toutes les opérations qu'on peut utiliser.

```
const char *binopn[ ] = {
"+", "-", "*", "/", ""«", "»",
">", "<", ">=", "<=", "==", "!=",
"", "|", ""
};
```

Exemple :

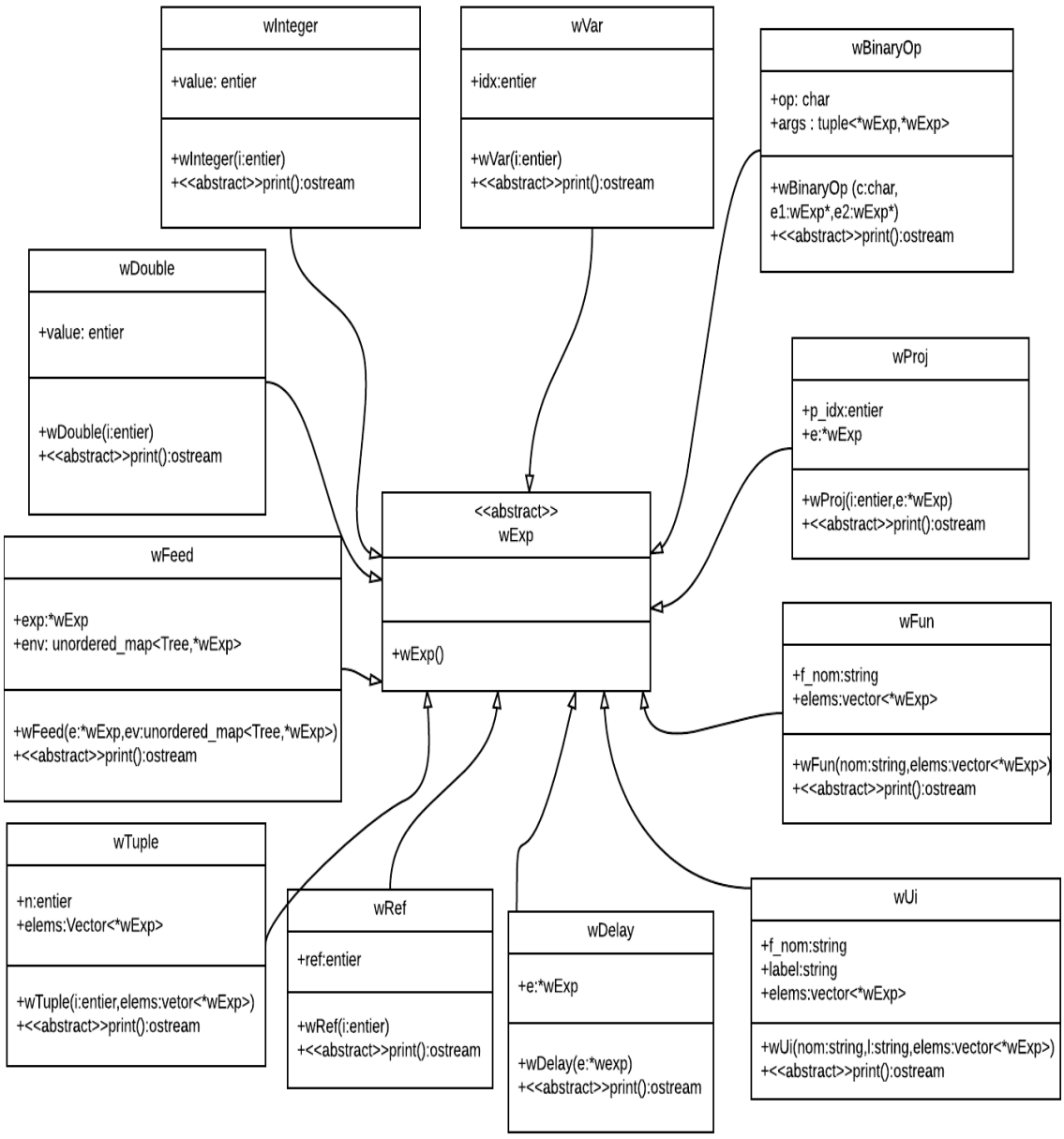
Parcours d'un noeud entier :

```
if(IsSigInt(sig,i))
{
return new wInteger(i) ;
}
```

Parcours d'un noeud opération binaire :

```
if(IsSigBinOp(sig, binopn[i], x, y))
{
return new wBinaryOp(ToWagner(x),ToWagner(y),binopn[i]) ;
}
```

Les classes d'expression de Wagner sont présentées sur le diagramme ci-dessous



2.6 Optimisation par table de hachage

Il existe souvent des modifications du code généré qui permettent d'obtenir un programme équivalent mais plus rapide. Pour cette raison, l'optimiseur examine le code généré pour le remplacer par du meilleur code. Il y a trois raisons principales pour ne pas produire directement du code optimisé dans le générateur de code. D'une part, cela permet au générateur de code de rester plus simple. D'autre part, l'optimisation peut être très coûteuse en temps et en espace mémoire, et cela permet de sauter cette étape et d'obtenir plus rapidement un programme à tester qui ne tournera qu'une seule fois ; le gain de temps obtenu en sautant l'optimisation compense largement le (modeste) gain de temps qu'on aurait obtenu en faisant tourner une seule fois le programme optimisé. Finalement, il est plus difficile de mettre au point un programme optimisé : certaines instructions du programme source ont pu être combinées ou réordonnées, certaines variables ont pu disparaître.
Une table de hachage est une structure de donnée reposant sur des tableaux et qui fournissent des algorithmes de recherche très performants. Chaque élément a une partie qui s'appelle la clé, elle permet de désigner le contenu de cet élément de manière non ambigüe. L'autre partie est la valeur associée

Dans cet esprit, j'ai été amené à utilisé des tables de hachage pour stocker les éléments (des signaux) dans un tableau afin d'optimiser la recherche d'un élément donné sans le crée une autre fois. Premièrement, on introduit des objets de la classe unorderd_map <Tree,wExp*> dont les clés sont des objets de type Tree, et dont les valeurs sont des pointeurs de la classe wExp. L'utilisation des pointeurs permet de gérer les collisions afin d'éviter le fait d'avoir deux clés différentes qui ont la même valeur. Ensuite, chaque fois qu'un signal récursif on crée une structure de type unordered_map <Tree,wExp*> où sont mis les éléments utilisés par ce signal récursif. Donc finalement la table de hachage utilisée est un tableau de pointeurs de type unordered_map<Tree,wExp*>.
À chaque itération, on vérifie si le signal est déjà dans le tableau, en utilisant des itérateurs. Si c'est le cas, la fonction " wExp* ToWagnerExp (Tree sig) " renvoie la valeur du signal déjà stocké. Sinon le traitement normal de la fonction est effectué et à la fin la nouvelle valeur crées est inséré dans le tableau initial.

Exemples d'exécution :
1) Pour le programme suivant : process = + : abs ;
Le résultat est :

process = + : abs ;
process has 2 inputs, and 1 outputs

output signals are
****
Number of elements in the deque : 1
Number of elements in the top map inside the deque : 5
0xa2d5560 :IN0
0xa2d55b8 :IN1
0xa2d57e0 :(P[0xa2d5560]+P[0xa2d55b8])
0xa2d5920 :abs(P[0xa2d57e0])
0xa2d5988 :P[0xa2d5920]
****
Program is :

P[0xa2d5988]

2) Pour le programme suivant : process = (+,+) ~*;
Le résultat est :

process = (+,+) ~*;
process has 3 inputs, and 2 outputs
output signals are
****
Number of elements in the deque : 1
Number of elements in the top map inside the deque : 7
0xa3ff658 :0
0xa408b50 : Feed = P[0xa408ae8]
Feed Table

0xa4076d8 : IN0

0xa407730 : IN1

0xa407788 : IN2

0xa407a20 : REF[1]

0xa407a88 : Proj0 P[0xa407a20]

0xa407ae0 : mem(P[0xa407a88])

0xa407b38 : Proj1 P[0xa407a20]

0xa407b90 : mem(P[0xa407b38])

0xa407cf8 : (P[0xa407ae0]*P[0xa407b90])

0xa407f88 : (P[0xa407730]+P[0xa407788])

0xa408a08 : (P[0xa407cf8]+P[0xa4076d8])

0xa408ae8 : (P[0xa408a08],P[0xa407f88])

0xa408b98 :Proj0 P[0xa408b50]
0xa408be0 :P[0xa408b98]@P[0xa3ff658]
0xa408c38 :Proj1 P[0xa408b50]
0xa408c90 :P[0xa408c38]@P[0xa3ff658]
0xa408d30 :(P[0xa408be0],P[0xa408c90])
****
Program is :

P[0xa408d30]

Chapitre 3

Difficultés et bénéfices tirés du stage

La principale difficulté a été de comprendre la manière dont le compilateur travail, qui met en jeu des concepts variés et complexes (Structures de données, le parcours récursif...) à l'aide du langage C++ qui est assez riche .

Il a également été difficile, mais formateur, d'imaginer des solutions optimisées d'implémentation qui permettent de résoudre le problème posé, comme le stockage en conteneurs, des signaux utilisés auparavant et la créations de nouveaux conteneurs qui manipulent des pointeurs pour chaque signal récursif. Le fait d'utilisé un seul tableau de pointeur était un avantage mais la recherche de ces éléments n'était pas immédiate.
Un cahier des charges qui n'est pas fixe et parfois évolutif.
D'autre part j'ai pris du temps pour me familiariser avec l'utilisation du logiciel Git que je trouve réserver aux développeurs ayant un minimum d'expérience. Comprendre sa configuration et son fonctionnement m'a perturbée au début, surtout la notion de la branche qui fait partie du cœur même de Git et qui gère toutes les modifications que nous faisons au fil du temps .

Conclusion et Perspectives :

Effectuer mon stage de deuxième année au sein du centre de recherche en informatique Mines ParisTech s'est globalement révélé très instructif et très formateur pour moi. En, effet cela m'a notamment poussé à répondre aux attentes d'un grand centre de recherche d'une école prestigieuse, ce qui m'a incité à produire un travail sérieux et des résultats performants.

Bien que j'ai rencontré certaines difficultés durant ce stage liées à diverses raisons, dont la découverte d'un nouveau langage de programmation fonctionnel, la compréhension du compilateur et du traitement récursif utilisé par celui-ci alors qu'il n'y a pas de documentation sur la manière dont il travail ou la difficulté de manipuler et de choisir les conteneurs convenables de la bibliothèque STD en C++ qui doit aboutir à des spécifications optimisées précises et à une implémentation satisfaisante.
Ce dernier point a été pour moi à la fois une difficulté à surmonter et un effort à réaliser qui s'est révélé très bénéfique. J'ai en effet acquis grâce à cela une véritable capacité de me documenter sur les outils performants de la bibliothèque STD et des réflexes de décomposition d'un problème en sous-problèmes à résoudre par les moyens du développement informatique.
Le programme qui a été développée a répondu au besoin prédéfinis au début.
Par ailleurs, ce stage ne m'a pas seulement profité sur le plan technique. En effet, grâce aux différentes réunions que j'ai dû effectuer, avec notamment mon responsable de stage et différents membres du projet, j'ai pu développer mon sens relationnel et acquérir de meilleures compétences de communication. J'ai aussi pu développer le sens de l'analyse et de réflexion qui doit avoir un informaticien et un scientifique en générale.
Pour conclure, je suis très satisfait des apports dont j'ai pu bénéficier grâce à ce stage, tant sur le plan technique que relationnel, qui me seront sans doute utiles dans mes expériences professionnelles futures, et qui ont conforté mon envie de m'orienter vers la recherche scientifique.

Bibliographie :

Compilers Principles, Technique, and Tools, Alfred V.Aho, Ravi Sethi, Jeffrey D.Ullman

Structure et interprétation des programmes informatiques, Harold Abelson/Gerald Jay Sussman avec Julie Sussman

Documentation du logiciel Faust, A Faust Tutorial, Etienne Gaudrain, Yann Orlarey

Documentation du logiciel Faust, Faust Quick Reference, Grame Centre National de Création Musical

http ://www.cplusplus.com/reference/deque/deque/
Titre : < deque >
Date de consultation : 09/08/2016

http ://www.cplusplus.com/reference/unordered_map/unordered_map/
Titre : <unordered_map>
Date de consultation : 07/08/2016

https ://openclassrooms.com/courses/gerez-vos-codes-source-avec-git
Titre : Gérez vos codes source avec Git
Date de consultation : 15/07/2016

https ://github.com/
Titre : Read the guide
Date : 15/04/2015
Date de consultation : 17/07/2016