0.1 Architecture du back end

Le back-end est composé de deux parties. La premiere partie dont le rôle est de générer le code intermédiare, et une deuxieme partie dont le rôle est de transformer le code en code machine au format ELF.

0.1.1 Le générateur du code intermédiaire LLVM

Il traduit l'arbre syntaxique du programme en code intermédiaire LLVM.Le générateur utilise le KawaTree pour obtenir les informations nécéssaires à la génération du programme.
Ce dernier est composé de deux sous-parties :

- Le IRCompiler est le composant responsable de la production du code intermédiare. Il parcours l'arbre syntaxique grâce au KawaTree et appelle les objets objets de production du code intermédiaire.s
- La famille de générateurs, un ensemble classes utilisées pour la génération du code intermédiaire. Il s'agit d'une sur-couche à l'API LLVM déja existante.

0.1.2 Structure du IRCompiler

Les noeuds de l'abre syntaxique et le génération de code étant découplée, le *patron visiteur* est adapté pour l'implémentation du IRCompiler

0.1.3 Famille de génération de code

Il s'agit d'une sur-couche aux classes de génération de code déja existantes. Elle est composée d'un ensemble de classes implémentant l'ensemble des spécifications qui suivent. Les spécification pourront être implémentées séparément pour le mode monolithique, et pour le mode partagé :

- **AffectationGenerator** : gère les opérations d'affectation
- **BlockGenerator** : gère la construction des blocs d'instruction
- **CallGenerator** : gère les appels de méthodes et de constructeurs
- **FunctionGenerator** : gère la creation de fonction et de méthodes
- **GlobalVariableGenerator** : gère la creation de variables gloabales néceéssaire à la cohésion du programme
- **NameBuilder** : gère le nommage des elements du code intermédiare
- **PrimitiveBinaryOperation** : gère les opérations binaire
- **PrimitiveCreator** : gère la création des primitives
- **PrimitiveValueConverter** : gere la convertion des valeurs primitieves
- **TypeGenerator** : gère le creation et l'accès au typage LLVM

0.1.4 La chaîne de compilation du code IR

LLVM nous fournit des outils de compilation permettant de traduit du code IR LLVM bien formé en assembleur. Le code assembleur peut ensuite être tra-

duit en code machine. Cette chaine de compilation est composée des éléments suivant :

- **llc**compile le fichier source en un fichier assembleur que l'on peut passer à un linker pour generer un executable
- **clang**compilateur permettant de compiler le code assembleur.

0.2 Fonctionnment dynamic du back end

0.2.1 Fonctionnment dynamic du IRCompiler

Le IRCompiler prendra en entrée un arbre syntaxique représentant la structure du programme à compiler. Le noeuds de l'arbre de cet arbre doivent implémenter les interfaces de la famille d'interface AST. (Voir annexe). Au cours de la production, on decorera l'arbre syntaxique avec de objets LLVM de rendre les noeud réutilisables pour la production de code. La production du code LLVM IR d'un noeud de l'arbre syntaxique, entrainera la production en cascade des noeuds fils de ce noeud. Le IRCompiler est responsable des algorithmes de génération de code IR. Selon le mode de compilation, il implémente différent algoriithme.

- MonolithiqueIRCompiler Compilateur en mode monolithique. Il produit un fichier .ll unique contenant le code intermédiaire représentant toute les classes compilées.
- SharedIRCompiler Compilateur en mode partagé, génére plusieurs fichiers .ll représentant les classes à compilées.

0.2.2 Fonctionnment des générateurs de code IR

Fonctionnement du NameBuilder

L'implémentation des méthodes de cette classe nécéssite la définition de contracts de nommage, permettant d'éviter toutes ambiguté entre les viariables qui seront générées. Ce composant est u

Fonctionnement du TypeGenerator

Ce composant permet de creer et d'obtenir des objets de typages d'un module llvm. Les types primitifs bool, int, char, float, double et string et tableaux représentant respectivement les types llvm i8, i32, i8, float, double, i8*, [nb x type]. On les obtient graçe au à la classe Type de l'api de LLVM.
Le structure principale de la classes est composée d'une référence vers un sous type contenant les attributs de la classe et une référence vers un tableau adHoc. Les méthodes et attributs statics sont quant à eux représentés par des variables globales.La génération d'une classe entrainera la génération de variables globales représentant les indexes des attributs dans la structure. Enfin le nommage de ces variables globales sera géré par le NameBuilder. (Voir l'annexe pour un exemple)

Fonctionnement du PrimitiveCreator

Ce composant permet de creer des objets LLVM représentant des valeurs primitives. Les classes LLVM à utiliser sont : ConstantInt, ConstantFP, ConstantDataArray (Voir l'annexe pour un exemple)

Fonctionnement du PrimitiveConverteur

Ce composant à pour role de creer les opérations de conversion d'un type vers l'autre. Le LLVM IR code ne supporte nativement les opérations binaires intertypées. Il faudra se servir de l'objet IRBuilder pour faciliter l'implémentation de la conversion. (Voir l'annexe pour un exemple)

Fonctionnement du PrimitiveBinaryOperation

Ce composant permet la génération des opérations primitives.L'utilisation d'un IRBuilder est conseillée. (Voir l'annexe pour un exemple)

Fonctionnement du FunctionGenerator

Ce composant permet de declarer et d'avoir accès au composant d'un module. Il utilise Namebuilder pour generer le nom du code de la fonction dans le code IR. La génération d'une méthode non statique ajoute un paramètre supplémentaire représentant l'objet ẗhis. La génération d'un constructeur génére deux fonction llvm. L'une créant et renvoyant une instance de l'objet creer, et une function appelée par la précédente permettant l'initialisation de l'instance. Cette fonction prendra un paramètre supplémentaire représentant l'objet à initialiser. De plus il permet aussi de déclarer et d'obtenir ades fonctions des librairies présentent sur le système comme, strlen, strcpy, strcat..etc (Voir l'annexe pour un exemple)

Fonctionnement du CallGenerator

Ce composant permet de generer les appels de méthodes et de constructeurs. (A completer) (Voir l'annexe pour un exemple)

Fonctionnement du GlobalVariableGenerator

Ce composant permet la génération des variables globales nécéssaire à la sémentique du programme. Il permet de declarer des variables statics, les tables AdHoc, et les indexes de membres. La compilation en mode Monilithique ét partagé divergera à ce niveau au niveau de atributs des variables. (Voir l'annexe pour un exemple)

Fonctionnnement des BlocGenérator

(A completer)