**Les Classes et Fonctions LLVM**

1. Les Classes:

a) LLVMContext:

http://llvm.org/doxygen/classllvm\_1\_1LLVMContext.html

C'est une classe importante pour utiliser LLVM dans un contexte chaîné. Il gère les données "globales" de l'infrastructure principale de LLVM, y compris le type et les tables de constantes. LLVMContext ne fournit aucune garantie de verrouillage.

b) Module:

http://llvm.org/doxygen/classllvm\_1\_1Module.html

La class Module représente la structure de plus haut niveau présent dans les programmes LLVM. La class module garde les traces des fonctions, des variables globales, et la table des symboles.

Module::FunctionListType &getFunctionList() : Returns the list of Functions.

Module::GlobalListType &getGlobalList() : Returns the list of GlobalVariables

SymbolTable \*getSymbolTable() : Return a reference to the SymbolTable

c) JITEventListener:

http://llvm.org/doxygen/classllvm\_1\_1JITEventListener.html

Cette interface est utilise par le JIT pour notifier au client les différents événements lors de la compilation. Par exemple on pourrait avoir des implémentations pour profilers et debuggers qui ont besoin de savoir ou la fonction a été émise.

virtual void llvm::JITEventListener::NotifyFreeingMachineCode(void \*OldPtr)[inline, virtual]

NotifyFreeingMachineCode est appelé a l'intérieur de freeMachineCodeForFunction(), après que le mapping global soit enlever, mais avant que le code machine retourne vers l’allocateur. OldPtr est l'adresse du code machine et sera le même que le paramètre Code d'un précédent appelle a NotifyFunctionEmitted. La fonction passé a NotifyFunctionEmitted peut avoir été détruit le temps que NotifyFreeingMachineCode soit appelé.

virtual void llvm::JITEventListener::NotifyFunctionEmitted (const Function &F, void \*Code, size\_t Size, const EmittedFunctionDetails & Details)[inline, virtual]

NotifyFunctionEmitted est appelé après qu’une fonction a été émise en mémoire avec succès. La fonction est toujours attache à sa MachineFunction.

d) IRBuilder:

http://llvm.org/doxygen/classllvm\_1\_1IRBuilder.html

IRBuilder - propose une API pour créer des instructions et les insérer dans un block aussi bien à la fin qu’a un endroit donné de ce block.

e) Function:

http://llvm.org/doxygen/classllvm\_1\_1Function.html

La class Function représente une procédure simple en LLVM. C'est l'une des classes les plus complexes de la llvm parce qu’elle doit garder la trace d'un large éventail de données: liste de BasicBlocks, liste d'arguments, une table des symboles. Une fonction est une GloabalValue et une Constant.

Function(const FunctionType \*Ty, LinkageTypes Linkage, const std::string &N = "", Module\* Parent = 0)

Constructeur utilisé quand nous avons besoin de créer une nouvelle fonction a ajouter au programme. Le constructeur doit spécifier le type de la fonction à créer et quel type de linkage la fonction doit avoir. L’argument FunctionType spécifie les arguments et la valeur de retour pour la fonction. La même valeur de FonctionType peut être utilisée pour créer plusieurs fonctions. L'argument Parent spécifie le module dans lequel la fonction est définies. Si l'argument est passé la fonction sera automatiquement insérer dans la liste des fonctions du module.

bool isDeclaration()

Retourne si la fonction a ou n'a pas de corps définis. Si la fonction est externe, elle n'a pas de corps et doit donc être résolue au moment du linkage avec une fonction d'une autre "unite de compilation".

Function::BasicBlockListType &getBasicBlockList()

Retourne la liste des BasicBlocks.

Function::ArgumentListType &getArgumentList()

Retourne la liste des arguments.

BasicBlock &getEntryBlock()

Retourne le block d'entree.

SymbolTable \*getSymbolTable()

Retourne un pointeur sur la table des symboles.

Function::viewCFG && Function::viewCFGOnly() ::= ouvre une fenêtre avec le graphe de la fonction

inst\_iterator ::= iterator qui permet d' avoir les instructions d une fonction sans passer par BasicBlock

f) Type:

http://llvm.org/doxygen/classllvm\_1\_1Type.html

Type est une superclasse de toutes les classes types.

\* bool isIntegerTy() const: Returns true for any integer type.

\* bool isFloatingPointTy(): Return true if this is one of the five floating point types.

\* bool isAbstract(): Return true if the type is abstract (contains an OpaqueType anywhere in its definition).

\* bool isSized(): Return true if the type has known size. Things that don't have a size are abstract types, labels and void.

g) FunctionType:

http://llvm.org/doxygen/classllvm\_1\_1FunctionType.html

Hérite de Type. Cette classe sert pour indique la valeur de retour d’une fonction.

h) PointerType:

http://llvm.org/doxygen/classllvm\_1\_1PointerType.html

Hérite de Type. Indique que c est un pointeur.

i)BasicBlock:

BasicBlock contient des Instructions qui forment le corps du block.

A noter que BasicBlock est value de Type Label.

----------------CODE-----------------------------------------------

BasicBlock \*pb = ...;

Instruction \*pi = ...;

Instruction \*newInst = new Instruction(...);

pb->getInstList().insert(pi, newInst); // Inserts newInst before pi in pb

-------------------------------------------------------------------

BasicBlockUtils.h ::= contient des fonctions utils sur les blocks

j) Value:

http://llvm.org/doxygen/classllvm\_1\_1Value.html

La class Value est la plus important des class dans les sources LLVM.

Elle représente une valeur type qui peut être utiliser comme un opérande d' une instruction.

Le nom d'une Value peut être une chaine vide, de ce fait le nom ne devrait servir qu’au debug.

k) User:

http://llvm.org/doxygen/classllvm\_1\_1User.html

l) Instruction:

http://llvm.org/doxygen/classllvm\_1\_1Instruction.html

Les sous classes importantes de la classe Instruction

\* BinaryOperator

This subclasses represents all two operand instructions whose operands must be the same type, except for the comparison instructions.

\* CastInst

This subclass is the parent of the 12 casting instructions. It provides common operations on cast instructions.

\* CmpInst

This subclass respresents the two comparison instructions, ICmpInst (integer opreands), and FCmpInst (floating point operands).

\* TerminatorInst

This subclass is the parent of all terminator instructions (those which can terminate a block).

m) GlobalValue:

http://llvm.org/doxygen/classllvm\_1\_1GlobalValue.html

n) GlobalVariable:

http://llvm.org/doxygen/classllvm\_1\_1GlobalVariable.html

o) EmittedFunctionDetails:

http://llvm.org/doxygen/structllvm\_1\_1JITEvent\_\_EmittedFunctionDetails.html

Vide pour le moment, mais cet objet contiendra tous les détails sur la génération du code.

p)BitstreamWriter:

http://llvm.org/doxygen/classllvm\_1\_1BitstreamWriter.html

Représente une stream en ecriture.

q) ExecutionEngine:

http://llvm.org/doxygen/classllvm\_1\_1ExecutionEngine.html

void llvm::ExecutionEngine::DisableLazyCompilation ( [bool](http://llvm.org/doxygen/X86DisassemblerDecoder_8c.html#a6156fe0b594c9754f386f6c6a30c8165)  Disabled = true  )  [inline]

DisableLazyCompilation - Quand la compilation paresseuse est débranchée (par défaut), le JIT compilera toutes les fonctions accessibles. Si la compilation paresseuse est allumée, le JIT compilera seulement une fonction et émettra des bouts pour compiler le reste quand ils seront appelés.

virtual void\* llvm::ExecutionEngine::recompileAndRelinkFunction ( Function \* F ) [pure virtual]

recompileAndRelinkFunction – Cette méthode est utilise pour forcer une fonction qui a déjà été compilé a être compile encore une fois

r) EngineBuilder():

http://llvm.org/doxygen/classllvm\_1\_1EngineBuilder.html

Class de construction pour ExecutionEngine

1. Les fonctions:
2. getGlobalContext():

http://llvm.org/doxygen/namespacellvm.html#aca44806a5debdf56d28db091cdb05e42

Retourne le contexte global : LLVMContext.

Que pour les clients qui n’utilise qu’un seul thread.

1. InitializeNativeTarget():

http://llvm.org/doxygen/namespacellvm.html#a72f848009615fed28b4d5ac17f020b65

Le programme principal doit appeler cette fonction pour initialiser la cible natale correspondant à l'hôte. C'est utile pour les applications JIT ca leurs permets d'assurer que la cible est liée correctement.

1. verifyFunction():

http://llvm.org/doxygen/namespacellvm.html#a36798313569555c6f79b4ac8cc0bf63d

Check les erreurs d'une fonction, écris les messages sur stderr. Retourne vrai si la fonction est corrompue.

CF: verifyModule().

1. WriteBitcodeToStream():

http://llvm.org/doxygen/namespacellvm.html#ac39abcf05358af4bc230f3c11cb1ca0a

Ecris un module sur un flux de sortis spécifié.