Projet `clang-LLVM`

Salwa Kobeissi
kobeissi@unistra.fr

Date Limite: Mercredi 28 Mars 2018 midi

Résumé

Après avoir passé deux sessions TP, pendant lesquelles nous avons exploré l'infrastructure clang-llvm et ses outils, nous présentons maintenant un projet où nous devons appliquer ce que nous avons appris et exploiter au mieux clang-llvm. Ce document vous présente le projet et ses objectifs. Veuillez suivre les étapes décrites, soyez attentifs aux notices mentionnés dans ce document et utilisez la documentation que vous avez vu en TP. Ceux-ci devront vous être utiles pour bien mener ce projet et atteindre les objectifs souhaités.

Ce projet sera fait par binôme et devrait être soumis sur moodle par binôme avant le Mercredi 28 Mars 2018 (jusqu'à midi).
Le projet vise à implémenter un profilage mémoire en affichant les accès mémoires.
Le projet est accompli par une passe, de type module, qui modifie l'IR de chaque fonction définie dans le module de telle sorte que tous les accès mémoires (load et store) soient affichés sur la sortie standard lors de l'exécution du programme. Pour chaque accès, le message indiquera s'il s'agit d'un load ou d'un store ainsi que l'adresse mémoire visée. Dans le cas des stores, le message indiquera également la valeur écrite en mémoire.

Afin de réaliser l'objectif de ce projet, veuillez suivre les étapes suivantes :

1. Votre passe doit ajouter une déclaration pour la fonction « printf » dans l'IR d'un module.
   Pour information, la fonction « printf » est déclarée dans le standard du langage C de la manière suivante :
   int printf(const char *format, ...) ;
   Afin de déclarer une nouvelle fonction dans le LLVM IR, nous devons :
   a. Construire le type de la fonction qui est une variable de type llvm : :FunctionType dans le passe

      Notice1 : on peut utiliser pour le type de "printf" : TypeBuilder<int(char *, ...), false>

b. Créer la fonction (variable de type llvm : :Function qui se nomme printf) dans le passe. Ceci permettra aussi de l'insérer dans le module afin de pouvoir l'appeler plus tard.

2. Nous notons que dans un module IR, les strings sont uniquement enregistrés comme des variables globales. Donc, afin de définir les messages à afficher sur la sortie standard lors de l'éxecution du programme, il faut définir des variables globales qui ont comme valeur les messages et donc, la passe doit les ajouter au module IR.
   Dans les tests sur lesquels vous appliquez votre passe, veuillez prendre en compte uniquement les valeurs écrites en mémoire et qui sont de type entier.
   En plus de l'adresse mémoire et de la valeur écrite, il faut imprimer un message après chaque store qui est : "I am storing %ld at address %p\n ", et un message pour chaque load qui est : "I am loading address %p\n" .

   a. créer les llvm : :Constant et leur affecter les valeurs des messages désirées dans la passe
   b. créer les llvm : :GlobalVariable dans la passe en se basant sur ces constants

      Notice2 : pour faire ceci, nous avons besoin de choisir un «linkage type », il existe différents choix de type de linkage à voir dans la documentation llvm.

      En créant ces variables, vous les injectez directement dans le module LLVM IR.

3. Pour créer un appel dans le module, nous allons utiliser la fonction "printf" que nous avons déjà créée et ses paramètres.
   Par exemple, l'appel attendu pour le projet sera : printf("I am loading address %p\n", pointer) ;

   Notice3 : Utiliser llvm : :IRbuilder pour créer et insérer un appel à la fonction printf avec sa liste de paramètres (tableau) : le llvm : :Constant qui contient le pointeur vers la variable globale (le message), l'adresse mémoire visée, et la valeur écrite (si l'instruction est un store).

   Pour les paramètres de "printf", il faut utiliser les variables globales que nous avons créées précédemment.
   Cependant pour pouvoir les utiliser correctement, il faut utiliser le type llvm : :Constant (descendant de llvm : :Value) et plus précisément la forme "getelementptr".

   Notice4 : Vous pouvez avoir un exemple de ceci dans l'IR en cherchant un code qui imprime des textes.

4. Pour ajouter les appels de printf dans le bon endroit :
   a. Pour les stores, Il faut chercher l'endroit où on utilise l'instruction du type llvm : :StoreInst, chercher les valeurs de l'adresse mémoire et de la valeur écrite et ajouter le printf.
   b. Pour les loads, il faut chercher l'endroit où on utilise llvm : :LoadInst, chercher la valeur de l'adresse mémoire et ajouter le printf.

   Vous pouvez trouver Les méthodes qui vous permettent de trouver les valeurs de l'adresse mémoire et la valeur écrites dans la classe de référence de llvm LoadInst et StoreInst.

Une fois que vous avez écrit votre passe vous pouvez tester votre code en utilisant l'exemple fourni test.c. La sortie doit ressembler au suivant :

```
I am storing 0 at address 0x7fff89f54c78
I am storing 0 at address 0x7fff89f54c7c
I am loading address 0x7fff89f54c7c
I am loading address 0x7fff89f54c7c
I am loading address 0x7fff89f54c7c
I am storing 0 at address 0x7fff89f54c60
I am loading address 0x7fff89f54c7c
I am storing 1 at address 0x7fff89f54c7c
I am loading address 0x7fff89f54c7c
I am loading address 0x7fff89f54c7c
I am loading address 0x7fff89f54c7c
I am storing 1 at address 0x7fff89f54c64
I am loading address 0x7fff89f54c7c
I am storing 2 at address 0x7fff89f54c7c
I am loading address 0x7fff89f54c7c
I am loading address 0x7fff89f54c7c
I am loading address 0x7fff89f54c7c
I am storing 2 at address 0x7fff89f54c68
I am loading address 0x7fff89f54c7c
I am storing 3 at address 0x7fff89f54c7c
I am loading address 0x7fff89f54c7c
I am loading address 0x7fff89f54c7c
I am loading address 0x7fff89f54c7c
I am storing 3 at address 0x7fff89f54c6c
I am loading address 0x7fff89f54c7c
I am storing 4 at address 0x7fff89f54c7c
I am loading address 0x7fff89f54c7c
I am loading address 0x7fff89f54c7c
I am loading address 0x7fff89f54c7c
I am storing 4 at address 0x7fff89f54c70
```

```
I am loading address 0x7fff89f54c7c
I am storing 5 at address 0x7fff89f54c7c
I am loading address 0x7fff89f54c7c
```