Nombre: John Alejandro González
Código: 1013336184

UNIVERSIDAD EAFIT

Lenguajes Formales y Automatas (CM0081)
Laboratorio II

Resumen

En este laboratorio se discute sobre la verificación de programas en representación intermedia de LLVM, el cual es un lenguaje de bajo nivel que se utiliza para representar programas de lenguajes de alto nivel, es un lenguaje que permite a los compiladores realizar optimizaciones de código.
Se utiliza la herramienta formal *VeLLVM* creada por *DeepSpec* para verificar programas en representación intermedia de LLVM.
Se discute la motivación del uso de las herramientas mencionadas, en los lenguajes mencionados.
Se discute la posibilidad de verificar programas que planteen problemas de teoria de numeros, ecuaciones diferenciales y grafos.
Se discute la posibilidad de revisar y estudiar formalmente sobre la implementación de verificación de programas hechos en "Brainf**ck" usando Interacion Trees.

Índice

1. Motivación

Al momento de escuchar la idea de verificar un programa pensé en verificar código en ensamblador, pues supuse que sería el lenguaje más bajo que se podría verificar, luego pensandolo bien, sobretodo pensando en que mi computador es un "Macbook Pro M2" el cual tiene una arquitectura ARM, cuya arquitectura es muy distinta a la que yo conozco y entiendo que es la arquitectura x86 de Intel, la idea de verificar ensamblador luego me parecio *"fuera de alcance"* pues existen muchas arquitecturas de computadores y no todas son iguales, y no todas tienen las mismas instrucciones. Esta idea también me llevo a pensar en verificar estructuras tales como "FPGA (Field Programmable Gate Array)" o "ASIC (Application Specific Integrated Circuit)" pues en el pasado había trabajado con estas, pero al igual que con la idea de verificar ensamblador lo encontré dificil.

Replanteando mi idea recorde un video del "MIT OpenCourseWare" llamado "5. C to Assembly" en el cual hablan sobre como el lenguaje C se traduce a ensamblador, en este mencionan que para el compilador "clang" se utiliza un lenguaje intermedio llamado "LLVM" el cual es un lenguaje de bajo nivel que se utiliza para representar programas de lenguajes de alto nivel, es un lenguaje que permite a los compiladores realizar optimizaciones de código. La idea me pareció buena cuando me di cuenta que multiples lenguajes de programación como Rust [**rust**], C++ [**cpp**], Haskell [**haskell**] y muchos otros más, utilizan LLVM como representación intermedia para sus compiladores, por lo que si se puede verificar un programa en LLVM, se puede verificar un programa en Rust, C++, Haskell, etc.

Nota: *Estuve un rato insistiendo en la idea de verificar ensamblador, pues no me gusta dejar "morir" ideas, encontré que varias empresas como Intel realizan verificación formal de sus procesadores, leyendo los documentos de estas empresas y presentaciones* **??** *logré aclarar muchas de mis dudas sobre qué y cómo se puede hacer verificación formal*

2. Detalles Técnicos y versionamiento

2.1. Sistema Operativo y versiones

Se utilizó el sistema operativo "MacOS Sonoma" versión 14.1.1, con la versión del kernel "Darwin23.1.0". Se utilizó la versión de clang "clang-15.0.0" y la versión de llvm "llvm-16.0.0". Se utilizó la versión de OPAM "2.1.5". Se utilizó la versión de Coq "8.15.2". Se utilizó la versión de Ocaml "4.13.1". Se utilizó make "3.81".

2.2. Instalación de herramientas

Las instrucciones que se encuentran en el repositorio de VeLLVM [**vellvm**] son muy claras, sin embargo hay que seguirlas en un orden distinto al que se presentan, he aquí el orden para instalar VeLLVM:

1. Instalar "OPAM" [**opam**] (Ocaml Package Manager) con el gestor de paquetes en su sistema operativo, en mi caso "Homebrew" [**homebrew**] con el comando "brew install opam".
2. Inicializar "OPAM" con el comando "opam init".
3. Agregar el repositorio de coq con el comando "opam repo add coq-released https://coq.inria.fr/opam/released".
4. Crear un Switch para la instalación de VeLLVM con el comando "opam switch create vellvm ocaml-base-compiler.4.13.1".
5. Clonar el repositorio y los submodulos con el comando "git clone –recurse-submodules git@github.com:vellvm/vel
   para el caso de este repositorio clonar con el comando "git clone –recurse-submodules git@github.com:kurtcovayne
   2023-2-lab2.git".

6. Ingresar al directorio src con el comando “cd vellvm/src”.
7. Instalar las dependencias con el comando “make opam”.
8. Para compilar tuve muchos problemas, pues tal parece que el make no está compilando las cosas en orden, para arreglar esto recomiendo un truco que es ejecutar varias veces el comando “make -j<n>” donde “n” e ir aumentando el n en caso de que falle, en mi caso “n” llegó a ser 24.

2.3. Verificación de programas en representación intermedia de LLVM

Para verificar un programa en representación intermedia de LLVM se debe seguir los siguientes pasos:

1. Crear un archivo con extensión “.ll” con el código en representación intermedia de LLVM.
2. Compilar el archivo “.ll” con el comando “clang -S -emit-llvm -o <nombre de archivo de salida><nombre de archivo de entrada>”.
3. Yo tuve problemas en “MacOS Sonoma” pues clang agrega modificadores a los parametros cuando ejecuta la instrucción call **??***(Como arreglaro)*, también llvm agrega modificadores a los loops y VeLLVM no logra reconocerlos **??***(Como arreglarlo)*, por lo que se debe arreglar el archivo “.ll” para que no tenga estos modificadores.
4. Agregar comandos de verificación al archivo “.ll” **??***(Como hacerlo)*
5. Ejecutar el comando “./vellvm –test-file <nombre de archivo de entrada>”.

2.4. Programas de interés

Se verificaron los siguientes programas:

1. Programa que realiza potenciación binaria de un número.
2. Programa que calcula el gcd de dos números usando el algoritmo extendido de Euclides.
3. Programa que calcula si existe una solución entera para la ecuación diofántica $ax + by = c$.
4. Programa que calcula la multiplicación de dos números usando la transformada rápida de Fourier en un algoritmo conocido como “Schönhage-Strassen algorithm”.

Los programas en C se pueden encontrar en el repositorio [**repo**].

2.5. Programas de futuro interes

Sería interesante planter programas a los cuales