Universidad Nacional de Córdoba

Facultad de Ciencias Exactas, Físicas y Naturales

Código Intermedio

Trabajo integrador final

Practica y construccion de compiladores

Autores:

Julián González

2021

Resumen

Este documento explicara los codigos intermedios usados por los compiladores GCC y CLANG/LLVM, como asi comparar sus principales caracteristicas viendo ventajas y desventajas entre ellos.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introduccion

El codigo intermedio es un codigo interno usado por el compilador para representar el codigo fuente. El codigo intermedio esta diseñado para llevar a cabo el procesamiento del codigo fuente, como es la optimizacion y la traduccion a codigo maquina.

Una de las caracteristicas mas esenciales del codigo intermedio es ser independiente del *hardware*. Por lo tanto, permite la portabilidad entre distintos sistemas.

Otra propiedad importante de todo codigo intermedio es su facil generacion a partir del codigo fuente, como asi tambien su facil traduccion al codigo maquina para la arquitectura deseada.

No existe un unico codigo intermedio, sino que hay distintos tipos y categorias, variando de compilador en compilador. Aunque un mismo compilador puede usar varios tipos de codigo intermedio en el proceso.

A continuacion, se presentan los codigos intermedios utilizados por los compiladores GCC y CLANG/LLVM, especificando las caracteristicas de cada uno y comparando sus prestaciones posteriormente.

Capítulo 2

Codigo intermedio de GCC

A continuacion se exponen las distintos codigos intermedios que GCC utiliza en la compilacion. Los distintos codigos intermedios estan relacionados en la forma que la salida de cada uno es la entrada del siguiente, avanzando desde una representacion general de alto nivel hacia una especifica de bajo nivel.

I Analizador lexico y sintactico

El analizador lexico lee la secuencia de caracteres desde la salida del preprocesador y agrupa los caracteres en secuencias llamadas lexemas. Por cada lexema, el analizador genera una token con la forma:

[*token name, attribute value*]

Donde *token name* son símbolos abstractos usados durante el análisis sintáctico, y *attribute value* es un puntero a una entrada en la tablas de símbolos. GCC no permite obtener los tokens válidos en forma de texto. Es un archivo interno. El analizador sintactico chequea si la gramática del lenguaje acepta la secuencia de tokens generados por el analizador lexico, sino reporta errores de sintaxis. Ademas, el analizador sintactico usa el primer componente de cada token para crear una representación en forma de árbol que muestre la estructura de los tokens, es decir, un árbol sintáctico. Con el *flag* -fdump-tree-original-raw se obtiene la representacion textual del arbol abstracto sintactico.

```
1    $ gcc -fdump-tree-original-raw codigo-ejemplo.c -o codigo-ejemplo
```

Listing 2.1: Comando de compilación del archivo codigo-ejemplo.c [1] para GCC.

II Analizador semantico

El analizador semantico utiliza el árbol sintáctico y la información de la tabla de símbolos para revisar la consistencia semántica del programa fuente con respecto a la definición del lenguaje.

III *Generic*

Generic es un codigo intermedio independiente del lenguaje con estructura de arbol que es generado por el *front end*. *Generic* es capaz de representar todos los lenguajes admitidos por GCC. *Generic* se produce eliminando construcciones especificas del lenguaje del arbol de parseo. GCC pasa del arbol abstracto sintactico a la representacion en *Gimple* en lenguajes como C.

IV *Gimple*

Gimple es un codigo intermedio de tres direcciones resultante de desglosar *Generic* en tuplas de no mas de tres operandos, a traves de la herramienta interna de GCC llamada *Gimplifier*. *Gimple* introduce variables temporales para poder computar expresiones complejas y permite supervisar el flujo de control a nivel inferior con sentencia secuenciales y saltos incondicionales. *Gimple* es el codigo intermedio principal de GCC (los lenguajes C y C++ se convierten a *Gimple* sin pasar por *Generic*), ademas de ser conveniente para optimizar.

Existen tres tipos de *Gimple*:

- *Gimple* de alto nivel que es lo que se obtiene despues de desglosar el *Generic*.
- *Gimple* de bajo nivel que se obtiene al linealizar todas las estructuras de flujo de control de del *Gimple* de alto nivel, incluidas las funciones anidadas, el manejo de excepciones y los bucles.
- *Gimple* SSA es el *Gimple* de bajo nivel reescrito en la forma SSA.

Con el *flag* -fdump-tree-gimple se obtiene la representacion en la forma de *Gimple*.

```
$ gcc -fdump-tree-gimple codigo-ejemplo.c -o codigo-ejemplo
```

Listing 2.2: Comando de compilación del archivo codigo-ejemplo.c [1] para GCC.

Ademas, con el *flag* -fdump-tree-all-graph GCC genera muchos archivos con la extension .cfg los cuales pueden visualizarse con una herramienta online Graphviz. Esta herramienta permite ver la evolucion del codigo en las distintas pasadas de una manera mucho mas conveniente para el usuario.

```
$ gcc -fdump-tree-all-graph codigo-ejemplo.c -o codigo-ejemplo
```

Listing 2.3: Comando de compilación del archivo codigo-ejemplo.c [1] para GCC.

V *Register Transfer Language*

Register Transfer Language es un codigo intermedio de bajo nivel semejante al lenguaje ensamblador. La mayor parte del trabajo del compilador se realiza en *Register Transfer Language*. Tiene una forma interna, formada por estructuras que apuntan a otras estructuras, y una forma textual que se utiliza en la descripción de la máquina y en los volcados de depuración impresos. El formulario textual usa paréntesis anidados para indicar los punteros en el formulario interno. Con el *flag* -fdump-rtl-final se obtiene la representacion en la forma de *Register Transfer Language* ya optimizado por el compilador.

```
$ gcc -fdump-rtl-final codigo-ejemplo.c -o codigo-ejemplo
```

Listing 2.4: Comando de compilación del archivo codigo-ejemplo.c [1] para GCC.

VI *Assembler*

Por ultimo, es posible obtener la salida en *Assembler* con el *flag* -S.

```
$ gcc -S codigo-ejemplo.c -o codigo-ejemplo.s
```

Listing 2.5: Comando de compilación del archivo codigo-ejemplo.c [1] para GCC.

Bibliografía

[1] J. Gonzalez, “Compiladores, codigo-ejemplo.”