8B ISW

Investigación sobre optimización de código intermedio y objeto

Compiladores e interpretes

Yahir Alejandro Medrano Gómez
UNIVERSIDAD POLITECNICA DE DURANGO

Proceso de optimización del código intermedio y objeto.

La fase de optimización trata de mejorar el código intermedio, de modo que en la siguiente fase resulte un código de máquina más rápido de ejecutar.

Después de la etapa de análisis semántico, se suele generar una representación intermedia explícita del programa fuente. Dicha representación intermedia se puede considerar como un programa para una máquina abstracta. La forma más simple de generar dicho código es el código de tres direcciones o código de tercetos.

Código de Tres Direcciones.

Es una manera linear de representar un grafo de subexpresiones en donde los nombres explícitos corresponden a los nodos interiores del grafo (figura 3), muy útil principalmente a la hora de optimizar. Las instrucciones se representan utilizando cuando mucho un solo operador en la parte derecha de la expresión, es decir, una expresión del tipo "x + y * z" se representaría de la manera "t1 = y * z; t2 = x + t1".

El código de tres direcciones se basa en 2 conceptos principalmente en 2 conceptos: direcciones e instrucciones.

En el diseño del código intermedio, es necesario escoger un buen conjunto de operadores. Usando el código de 3 direcciones, el compilador puede representar código intermedio y ayudarse en la implementación más optima del código final.

Una proposición de código de 3-direcciones se puede implantar como una estructura tipo registro con campos para el operador, los operandos y el resultado. La representación final será entonces una lista enlazada o un vector de proposiciones. Hay dos formas principales de implementar el código de tres direcciones:

Cuádruplas.

Una cuádrupla es una estructura tipo registro con cuatro campos que se llaman (op, result, arg1, arg2).

El campo op contiene un código interno para el operador.

Por ejemplo, la proposición de tres direcciones x = y + z se podría representar mediante la cuádrupla (ADD, x, y, z). Las proposiciones con operadores unarios no usan el arg2. Los campos que no se usan se dejan vacíos o un valor NULL. Como se necesitan cuatro campos se le llama representación mediante cuádruplas.

Tripletas.

Para evitar tener que introducir nombres temporales en la tabla de símbolos, se hace referencia a un valor temporal según la posición de la proposición que lo

calcula. Las propias instrucciones representan el valor del nombre temporal. La implementación se hace mediante registros de solo tres campos (op, arg1, arg2).

En la notación de tripletes se necesita menor espacio y el compilador no necesita generar los nombres temporales. Sin embargo, en esta notación, trasladar una proposición que defina un valor temporal exige que se modifiquen todas las referencias a esa proposición. Lo cual supone un inconveniente a la hora de optimizar el código, pues a menudo es necesario cambiar proposiciones de lugar.

Una forma de solucionar esto consiste en listar las posiciones a las tripletas en lugar de listar las tripletas mismas. De esta manera, un optimizador podría mover una instrucción reordenando la lista, sin tener que mover las tripletas en sí.

Principales fuentes para optimizar

Una transformación de un programa se denomina local si se puede realizar observando solo las proposiciones de un bloque básico; en caso contrario se denomina global.

Transformaciones que preserven la función: hay varias formas en que un compilador puede mejorar un programa sin modificar la información que calcula. La eliminación de subexpresiones comunes, la propagación de copias, la eliminación de código inactivo y el cálculo previo de constantes; son ejemplos comunes de dichas trasformaciones que preservan la función.

A continuación, se describen las primeras tres:

- Subexpresiones comunes: una concurrencia de una expresión E se denomina subexpresión común si E ha sido previamente calculada y los valores de las variables dentro de E no han cambiado desde el cálculo anterior. Se puede evitar recalcular la expresión si se puede utilizar el valor calculado previamente.
- Propagación de copias: esta transformación es usada en conjunto con las subexpresiones comunes haciendo o propagando copias que ayuden dichas transformaciones.
- Eliminación de código inactivo: Una variable esta activa en un punto de un programa si su valor puede ser utilizado posteriormente; en caso contrario está inactiva en ese punto. Una idea afín es el código inactivo o inútil, proposiciones que calculan valores que nunca llegan a utilizarse.

Optimización de ciclos: un punto muy importante para las optimizaciones son los ciclos o bucles, especialmente los ciclos internos donde los programas tienden a emplear la mayor parte de su tiempo. El tiempo de ejecución de un programa se

puede mejorar si se disminuye la cantidad de instrucciones en un lazo interno.

Compilador optimizador

Un compilador optimizador es un compilador que trata de minimizar ciertos atributos de un programa informático con el fin de aumentar la eficiencia y rendimiento. Las optimizaciones del compilador se aplican generalmente mediante una secuencia de transformaciones de optimización, algoritmos que transforman un programa para producir otro con una salida semántica equivalente pero optimizada.

Generalmente hay varios aspectos que se desean optimizar:

1. Optimización temporal: reducir el tiempo de ejecución del programa.
2. Optimización espacial: reducir la cantidad de espacio en memoria que ocupa el programa en ejecución.
3. Reducir el tamaño del programa.
4. Minimizar la potencia consumida por un programa (debido a las computadoras portátiles).

La optimización se realiza después de la generación de código de todo el programa o de un elemento ejecutable del programa por ende es dependiente del contexto. La condición que ha de cumplir es que el código optimizado se comporte igual que el código de partida, excepto por ser más rápido u ocupar menos espacio.