ERRORES CON OPERACIONES DEL TIPO ‘+=’ Y OPERACIONES DE INCREMENTO/DECREMENTO

Los errores en la generación de código se produce cuando se utilizan operadores del tipo ‘+=’ (asignación junto con una operación) y con los operadores de incremento/decremento de las operaciones unarias.

Operaciones del tipo ‘+=’:

Las expresiones que incluyen en el mismo operador tanto la asignación como la operación a realizar y los operando son variables locales o parámetros (es decir, cuando se hace uso de los índices obtenidos por el SSA), me dan problemas en la generación de código, dado que hay información implícita que tengo que obtener.

Si tengo el código siguiente: ‘a += b’, sé que tengo que generar código equivalente a ‘a = a + b’.

En un proceso secuencial es sencillo:

a = 20; --> a1 = 20;

b = 3; --> b1 = 3;

a += b; --> a2 += b1; --> a2 = a1 + b

Porque el valor del índice SSA para esa variable (PreviousIdILName) es siempre uno menos.

Los problemas surgen cuando estas expresiones se encuentran dentro de un bucle, nos centraremos en la instrucción marcada en amarillo:

**AST INICIAL:**

while ((a = b + a) != 0) {

b = b + a;

b = b + 1;

a = a + 1;

}

**AST tras VSSA:**

while ((a1 = b0 + a0) != 0){

b1 = b0 + a1;

b2 = b1 + 1;

a2 = a1 + 1;

}

**AST tras VSSA2:**

a3 ← a0

b3 ← b0

while (

a3 = θ(a0,a1,a2)

b3 = θ(b0,b2)

(a1 = b3 + a3) != 0

a3 ← a1

){

b1 = b3 + a1;

b2 = b1 + 1;

b3 ← b2

a2 = a1 + 1;

a3 ← a2

}

Si la instrucción subrayada en amarillo, en vez de ser ‘b = b + a’, fuese ‘b += a’; el resultado debería ser el mismo, es decir, al final tiene que generar código para ‘b1 = b3 + a1’. Es decir, el valor de PreviousIdILName para b no es 0, sino 3.

Operaciones incremento/decremento:

Estas operaciones llevan implícitas una asignación. Sin embargo, no tienen un nodo asignación dentro de su estructura y en el VisitorSSA no se ha contemplado esta posibilidad.

El error se produce cuando el operando es una variable local o parámetro (se está haciendo uso del algoritmo SSA) y es la última instrucción de un bloque para una determinada variable. En ese caso, al ser la última instrucción para una variable concreta, se creará una sentencia MOVE que permita asignar el valor de la variable a la variable con mayor índice SSA hasta ese momento (es decir, estamos actualizando el valor de la variable). Esta instrucción MOVE está asociadas a esa última asignación que se realiza sobre la variable, pero como un incremento/decremento, no tiene un nodo asignación, no va a crear esa sentencia.

Por otra parte, la numeración asociada a las expresiones de incremento/decremento, tampoco es correcto. Si tengo el siguiente código:

a = x;

++a; // a = a + 1;

La numeración que se obtiene con el SSA es el siguiente:

a1 = x0;

++a1; // a2 = a1 + 1; ¿?

Sin embargo, en la operación de incremento no se refleja la asignación implícita que tiene y no incrementa el valor del índice SSA. Se debería obtener: ++a2.

Nos encontramos con el mismo problema que en el caso anterior. Debemos averiguar cuál es el valor previo (el índice SSA) para esa variable, para poder hacer el incremento sobre el valor anterior y asignárselo a la nueva variable.

Para solucionarlo se debería modificar lo siguiente:

* En UnaryExpression se añade un nuevo atributo que sea un MoveStatement, de modo que se le podrá asociar a las operaciones de incremento/decremento.
* En VisitorSSA tengo que incrementar el valor del identificador en una expresión unaria de tipo incremento/decremento. Es decir:

a = x; --> a1 = x0;

++a; --> ++a2; // En lugar de a1++ como tenía hasta ahora.

Para ello, al llegar al nodo SingleIdentifierExpression, tiene que saber que hay que incrementar el valor del índice SSA. Por tanto, la variable tiene que encontrarse como parte izquierda de una asignación. En una operación de incremento/decremento no tiene nodo asignación y su atributo LeftExpression estará a true si su nodo padre está a true (obtiene su valor del valor que tenga el nodo padre).

En UnaryExpression, de forma provisional, cuando se trata de una operación de incremento/decremento, se pone LeftExpression a true para que actualice el valor en el nodo SingleIdentifierExpression y después, se restaura dicho valor por el valor que tuviese su nodo padre.

* En VisitorSSA2 hacemos un nuevo recorrido para UnaryExpression para que genere, si es necesario, una sentencia MoveStatement asociada a la expresión. (Similar al recorrido de AssignmentExpression y MoveStatement).
* En VisitorCodeGeneration, para una operación de incremento/decremento se genera código del tipo a = a ?? 1. Para ello, la primera parte se genera como esta (la que corresponde a la variable de la parte izquierda) y para la siguiente parte, busca el PreviousIdILName (donde tengo el mismo problema del caso anterior).
* A todos los recorridos de UnaryExpression, se añade la visita a MoveStatement (si existe).