Open Arithmetics Fortran 90 *

Germán E. Regis, Marcelo Arroyo y Jorge Aguirre

Departamento de Computación
Facultad de Cs. Exactas Fco-Qcas y Naturales
Universidad Nacional de Río Cuarto
{gregis, marroyo, jaguirre}@dc.exa.unrc.edu.ar

Palabras clave: FORTRAN, error de redondeo, análisis numérico, compiladores, aritmética de punto flotante, sistemas de tipos, coerción, sobrecarga de operadores.

Resumen

Los errores de redondeo introducidos por las diferentes representaciones finitas de los números reales en los lenguajes de programación y su propagación a través de la operaciones aritméticas constituye el problema central del cálculo numérico. Cada aritmética particular de punto flotante induce una forma de propagación distinta. Por ese motivo se han desarrollado numerosas bibliotecas que implementan aritméticas alternativas a las provistas por estos lenguajes. Sin embargo su utilización en los programas que realizan largas secuencias de operaciones como los destinados al cálculo numérico, principal aplicación de estas bibliotecas, implica un costoso trabajo de adaptación de estos programas a la nueva aritmética. En este artículo se describe el desarrollo y uso de OAF90, una herramienta que realiza dicha adaptación de manera automática para programas Fortran 90, uno de los lenguajes mas utilizados en el campo de aplicación del calculo numérico.

OAF90 junto con su predecesor para Fortran 77 permiten incorporar aritméticas externas en cualquier programa Fortran.

1. Introducción

En una computadora sólo pueden representarse estructuras finitas, obviamente por lo tanto, ningún conjunto infinito podrá ser representado en su totalidad. Así para representar a los números enteros se elige generalmente un rango de representación de los mismos. Aquellos que estén fuera de él escaparán a la posibilidad de representación. Pero como el conjunto los números reales es denso, ni aún limitando el alcance de la representación a un rango es posible representar a todos los que estén dentro de él; por tal motivo también se debe limitar la precisión.

Este problema conduce a las dos representaciones computacionales usadas en los lenguajes de programación: la de punto fijo [Knu73] y la de punto flotante [Knu73]. La primera conserva el valor absoluto y la segunda el error relativo. La primera es usada en lenguajes de uso contable y la segunda en lenguajes destinados al cálculo numérico, utilizado en la matemática aplicada, la física y la ingeniería, en cuyos campos es utilizada desde mucho antes del advenimiento de las computadoras.

En la representación de punto flotante se introduce forzosamente un diferencia de valores entre el número real y su representación, a esta diferencia entre un número real exacto y su representación aritmética se la denomina *error de redondeo*[Wil63].

Es bien conocido que este tipo de errores, y su propagación a través de operaciones aritméticas pueden causar grandes fallas en sistemas de computación [Gol91][Joh82], produciendo a quienes los utilizan graves

*Este trabajo ha sido desarrollado en el marco de un proyecto subsidiado por la SCyT de la Universidad Nacional de Río Cuarto

daños; como casos destacados se pueden señalar los de los misiles: "Ariane 5" [1],"Misil Patriot" [2] que constituyen ejemplos de catástrofes citadas.

Un ejemplo concreto de los problemas citados es el siguiente programa FORTRAN:

```
PROGRAM EJEMPLO_ERROR_REDONDEO
  REAL A,C,SUB1,SUB2,RES1,RES2
  A = 10**8
  C = 1
  SUB1 = C+A
  SUB2 = C+A-1
  RES1 = (SUB1-SUB2)*100
  RES2 = ((C+A)-(C+A-1))*100
  PRINT *, RES1," = ", RES2
END
```

Pese a que el resultado del mismo debería ser "100.0000 = 100.0000", luego de compilarlo y ejecutarlo, nos muestra "0.0000000 = 100.0000".

Los lenguajes de programación más utilizados sólo proveen una o dos representaciones de números reales, que algunas varían en su precisión pero mantienen las mismas propiedades. Por tal motivo, desde los comienzos de la computación digital, se han desarrollado numerosas bibliotecas que implementan distintas aritméticas reales, como la de punto flotante de precisión variable [Bai94] o la intervalar [Hol96]. Estas bibliotecas permiten experimentar, analizar y obtener soluciones prácticas que no pueden alcanzarse utilizando las aritméticas provistas por el lenguaje.

Este cambio de aritméticas en un programa puede ser muy costoso, debido a que se debe traducir en él, cada una de las operaciones y funciones por invocaciones a las correspondientes operaciones y funciones que implementa la biblioteca. Para realizar el cálculo se deben crear estructuras que puedan alojar la nueva representación de los números, descomponer las expresiones en operaciones simples y administrar la variables temporarias necesarias para conservar los resultados intermedios.

En los programas de cálculo, donde las expresiones constituyen un alto porcentaje de su código, dicha traducción no sólo requiere de un significativo trabajo de programación, sino que también representa una fuente de introducción de errores; esto genera un rechazo a la utilización de dichas bibliotecas como alternativa para tratar los errores de redondeo.

Existen paquetes que implementan distintas aritméticas y hay un precompilador "OAF" [Agu01], que hace transparente el proceso de adaptación de un programa a uno de estos paquetes, pero éste precompilador sólo funciona para Fortran 77. Una enorme cantidad de usuarios Fortran siguen por lo tanto limitados en el uso de aritméticas no estándar.

En este trabajo se describe el desarrollo de una herramienta, OAF90 (Open Arithmetics Fortran), que *automatiza* el proceso de transformación de un programa existente, con el fin de cambiar su aritmética real por otra, definida en una biblioteca externa. El lenguaje para el cual se implementó es Fortran 90, uno de los más usados en la ingeniería.

OAF90 es un precompilador que toma como argumento un programa Fortran y lo adapta automáticamente para que utilice una aritmética alternativa en reemplazo de su tipo real. Sólo es necesario definir un módulo que sirve como interfaz entre cualquier programa FORTRAN y la biblioteca de funciones que implementa dicha aritmética, la cual puede estar implementada en cualquier lenguaje de programación.

Así el usuario Fortran podrá experimentar en sus programas la utilización de distintas aritméticas sin necesidad de modificar manualmente el código de sus programas, sólo necesita precompilarlos con OAF90.

Mientras que la vieja versión de OAF [Agu01], implementada previamente en el grupo, se basa en un precompilador tradicional que incluye todas las etapas clásicas de un compilador, excepto las de optimización de código, OAF90 se basa en el polimorfismo que incorpora Fortran 90 y como las extensiones

[1]El 4 de junio de 1996, el cohete Ariane 5 lanzado por la Agencia Espacial Europea, explotó 40 segundos después del despegue causando una perdida de 500 millones de dolares; la causa fue un error numérico en el sistema de referencia inercial.

[2]El 25 de febrero de 1991, un misil de defensa Patriot no pudo interceptar a un misil de ataque Scud Iraquí, causando la muerte a 28 soldados; la causa fue la perdida de precisión en el calculo de tiempo de la trayectoria del misil Scud.

de Fortran deben respetar la compatibilidad hacia sus versiones anteriores, se supone que podrá ser también utilizado en versiones posteriores.

Las dos versiones de OAF: la anterior para Fortran no polimórfico (77, IV, II y original) y OAF90 para versiones polimórficas de Fortran, permitirán su utilización a cualquier usuario de Fortran de aquí en más, independientemente del compilador que se use.

También se desarrolló un asistente que guía la definición de los módulos interfaces, si bien éstos módulos fueron concebidos para interactuar entre una biblioteca externa que implementa una aritmética real y cualquier programa FORTRAN, pueden por si sólos implementar una aritmética real definiendo en él todas las operaciones y funciones correspondientes a dicha aritmética. Haciendo uso de este asistente, el usuario final podrá, en caso de ser necesario, definir fácilmente sus propias aritméticas para ser utilizadas por sus programas.

En este artículo se presenta una descripción general del proceso de traducción 2 y su forma de uso 2.1. Luego se especifican las características del módulo interfaz 3. Este módulo siguiendo el *pattern adapter* [Gam/95], será el intermediario entre las bibliotecas aritméticas y el programa modificado.

Una vez detallada la estructura de los módulos interfaces en Fortran, se describe el proceso de traducción que realiza el precompilador desarrollado 3.2.

2. Adaptación de un programa a una aritmética alternativa usando OAF90

Si se dispone de una biblioteca que implementa las operaciones y funciones de una aritmética real alternativa y su correspondiente interfaz con OAF90, cualquier usuario Fortran puede adaptar sus programas para que utilicen esta nueva aritmética en reemplazo de la real provista por dicho lenguaje.

Si bien la utilización de bibliotecas de rutinas es algo común en la programación, incorporarlas en programas existentes, es una tarea que requiere mucho esfuerzo de programación; más aun, cuando se trata de incorporar bibliotecas numéricas en programas de cálculo en los cuales gran porcentaje de su código son expresiones que deberían adaptarse a las mismas.

Es aquí donde OAF90 muestra su funcionalidad al realizar dicha modificación en forma automática. Así el usuario Fortran, no necesita conocer las operaciones y subrutinas que implementan las bibliotecas, ni preocuparse por introducir en su programa las modificaciones para usarlas en reemplazo de las estándar. Simplemente *precompila* su programa con OAF90 y luego lo compila junto con las bibliotecas elegidas. De esta manera, podrá experimentar sus programas con distintas aritméticas sin tener que invertir tiempo en modificarlos, ya que OAF90 lo realiza por Él.

Figura 1: Proceso de adaptación programas Fortran a una aritmética alternativa usando OAF90

Para reemplazar en un programa Fortran su aritmética real por otra definida en una biblioteca externa, OAF90 utiliza un módulo que hace de intermediario entre ellos. Éste módulo o interfaz, definido en Fortran, especifica la nueva aritmética describiendo su estructura, operaciones y conversiones con el tipo real de Fortran.

En él se declara el Tipo Abstracto de Datos **newreal** que implementa las funcionalidades que presenta el tipo numérico real de Fortran. Para ello en cada especificación de sus operaciones y funciones se invocan a las correspondientes implementadas por la biblioteca externa. Esta biblioteca puede estar programada en otro lenguaje, como por ejemplo “C”.

Para realizar la modificación, OAF90 lee el código de un programa original Fortan y lo traduce en un nuevo programa, reemplazando el tipo **real** por el **newreal** ya definido en el módulo interfaz. De esta manera la funcionalidad del nuevo programa no se altera, ya que solo se modifica la estructura de los números reales y sus correspondientes operaciones y funciones.

Una vez obtenido el nuevo código del programa, se genera mediante el compilador Fortran, el correspondiente programa objeto. De igual manera se procede con el código del módulo interfaz. Luego se los combina mediante el linkeditor en un programa ejecutable que respeta el funcionamiento del original, pero generando resultados con las propiedades de la nueva aritmética.

2.1. Uso del precompilador

Para adaptar un programa Fortran de manera tal que utilice una aritmética real externa en reemplazo de la estándar con OAF90, se debe disponer del archivo fuente del programa a modificar, el modulo o interfaz y los archivos de la biblioteca externa a incorporar.

La forma de invocarlo es:

```
OAF90 <Programa fuente Fortran> <modulo interfaz>
```

Este proceso arroja como resultado el programa adaptado, un reporte de cambios realizados **Reporte** y una lista de archivos que son incluidos dentro del programa **OtrosArchivos**, los cuales deberían procesarse.

Una vez modificado el código del programa, el usuario debería compilarlo nuevamente junto con el módulo interfaz de la nueva aritmética. Por ejemplo con el compilador Fortran libre de GNU:

```
g95 <modulo interfaz> <Programa Precompilado> -o <Archivo ejecutable>
```

Con el fin de hacer más ameno este proceso en ambientes gráficos se implementó un Front-end. El mismo posee una ventana en la cual el usuario deberá indicar el nombre del archivo correspondiente al programa a modificar y el módulo interfaz de la aritmética deseada. Luego de apretar el botón **Precompilar**, se mostrarán los cambios realizados como así también los archivos incluidos en el programa que deberían procesarse para mantener la coherencia de los reales utilizados. Para compilar el programa, el usuario simplemente debe apretar el botón **Compilar**, así se invocará al compilador con el módulo interfaz y el programa adaptado, generando como salida un programa ejecutable con las mismas funcionalidades que el original pero que utiliza la nueva aritmética.

2.2. Caso de uso con una aritmética intervalar

La idea de análisis intervalar existe desde varias décadas atrás, este presenta una alternativa para tratar la densidad de los números reales dentro de sistemas de computación, donde ésta se ve limitada. Si bien existen bibliotecas que la implementan, su aplicación dentro de programas existentes no tubo mucha adhesión ya que la misma presenta una sintaxis distinta a las demás aritméticas como la de punto fijo y punto flotante.

En esta aritmética, un intervalo representa un rango de números, podemos pensar en él como la aproximación de número a través de sus cotas inferior y superior. Por ejemplo si se suma el intervalo [1,3] con el intervalo [2,4], se obtiene el intervalo [3,7], ya que éste contiene la suma de cualquier par de elementos de sendos intervalos.

Como caso de estudio de OAF90, se implementó la aritmética intervalar directamente sobre el módulo interfaz usando doble precisión. Si con esta implementación se compila y ejecuta el programa "EJEMPLO ERROR REDONDEO" mostrado en la sección 1, se obtiene el siguiente resultado:

```
 [ 100.000000000000  ,  100.000000000000 ]
  =
 [ 100.000000000000  ,  100.000000000000 ]
```

La versión precompilada, transparente al usuario, producida por OAF90 es el programa siguiente:

```
PROGRAM EJEMPLO_ERROR_REDONDEO
  USE MODULE_NEWREAL
  IMPLICIT TYPE(NEWREAL) (A-H)(O-Z)
  TYPE(NEWREAL)  A,C,SUB1,SUB2,RES1,RES2
  A = 10**8
  C = 1
  SUB1 = C+A
  SUB2 = C+A-1
  RES1 = (SUB1-SUB2)*100
  RES2 = ((C+A)-(C+A-1))*100
  CALL PRINT_NR ( RES1)
```

```
  PRINT *," = "
  CALL PRINT_NR (RES2)
END
```

3. Diseño de OAF90

En el trabajo previo para Fortran 77 [Agu01], la idea general de la traducción, fue generar espacio extra en memoria dentro del programa, mediante arreglos de caracteres, para mantener los valores de los nuevos reales, correspondientes a cada variable real del mismo. También se genera código en las expresiones para cada operación sobre los nuevos reales, necesitando así estructuras: como pilas, para el procesamiento de expresiones; tablas de símbolos, para mantener la información de las variables y su correspondiente dirección de almacenamiento.

Fortran 90 posee grandes cambios respecto de su predecesor, entre otros: objetos, punteros, polimorfismo, arreglos dinámicos, tipos derivados; es por ello que se descartó el proceso de traducción mencionado. A cambio OAF90 utiliza estas nuevas características para dicho fin.

La idea en la cual se basa OAF90 para que un programa Fortran pueda usar una aritmética real alternativa es:

- Utilizar un módulo interfaz conteniendo el *tipo abstracto de datos* **newreal** para que funcione como intermediario entre la biblioteca numérica y cualquier programa Fortran. Para ello se utilizan las extensiones de Fortran 90 como **módulos**, definiciones de **tipos derivados** (definidos por el usuario) y **sobrecarga de operadores** [Wat90], estos últimos a través de procedimientos, funciones y operadores **genéricos**[Ada92].

- Modificar mínimamente el archivo del programa fuente original, para asegurar así, la nueva funcionalidad del mismo. Esto se logra debido a que el módulo interfaz vuelve transparente para cualquier programa Fortran la nueva aritmética a través del tipo derivado **newreal** definido en él. De esta manera sólo se necesitan algunos cambios para reemplazar el tipo real de Fortran por el nuevo. Algunas modificaciones al código fuente necesarias son las concernientes a: declaraciones de variables, subrutinas y funciones, inicializaciones y funciones de entrada/salida.

Figura 2: Diseño de OAF90

Este diseño evita tener estructuras dentro del programa original, como así también, lidiar con punteros, arreglos dinámicos, generar código para las expresiones, etc. .

Esta minimización de modificaciones en el código del programa original mantiene su comprensión sintáctica original.

De esta manera OAF90 presenta gran flexibilidad para la incorporación de nuevas aritméticas, y portabilidad para futuras versiones de Fortran.

3.1. Implementación del módulo interfaz para una aritmética alternativa

El módulo interfaz debe presentar para cualquier programa Fortran las mismas funcionalidades que la aritmética real de Fortran, es decir, que estén definidos en él la estructura y operaciones del tipo derivado **newreal**, como así también la interacción con otros tipos mediante coerciones. Este modulo interfaz esta implementado como un módulo (MODULE) de Fortran, manteniendo encapsulada toda su funcionalidad con independencia de cualquier programa Fortran que lo utilice.

Si bien este módulo ha sido concebido para servir sólo como interfaz entre los programas y la biblioteca que implementa una aritmética alternativa, en él se puede implementar autónomamente una aritmética alternativa completa sin hacer uso de una biblioteca *ad-hoc*.

En el módulo se definen:

- **La estructura (representación) de los nuevos números reales**.

- **Las funciones de conversión**.

- **Las operaciones y relaciones**.

- **Las funciones intrínsecas**.
- **La sobrecarga de operadores**.
- **Las coerciones con los tipos de Fortran**.
- **Procedimientos de Entrada/Salida**.

3.1.1. La estructura (representación) de los nuevos números

El módulo contiene la especificación de la estructura de los nuevos reales mediante la declaración de un *tipo derivado* de Fortran. Este tipo derivado está compuesto por una serie de campos de tipos provistos por dicho lenguaje. Por ejemplo en caso de una aritmética intervalar, su estructura podría estar definida como:

```
TYPE NEWREAL
  REAL INF
  REAL SUP
END TYPE
```

En caso de utilizar una biblioteca externa, la estructura del tipo derivado debe tener un tamaño suficiente como para poder alojar a la estructura implementada por dicha biblioteca. Por ejemplo si la estructura que utiliza la aritmética externa es de 32 bits, el tipo derivado podría estar definido como un arreglo de 4 caracteres.

3.1.2. Las funciones de conversión

Otra de las funcionalidades que presenta el módulo es la posibilidad de obtener un número de la aritmética alternativa a partir de un real de Fortran, como así también, en forma inversa, poder obtener un real de Fortran a partir de un **newreal**. Estas conversiones son necesarias para mantener la compatibilidad con el resto del programa original.

Estas conversiones están definidas en el módulo como subrutinas. Por ejemplo, un modelo para la primera sería:

```
SUBROUTINE REAL_TO_NEWREAL ( NR , R )
    TYPE (NEWREAL) NR
    REAL R
    INTENT (OUT) NR
    INTENT (IN) R
    <Llamado a la función de biblioteca o código propio>
END SUBROUTINE
```

En caso de utilizar una biblioteca externa que implemente la nueva aritmética, aquí simplemente se encuentra la invocación a la correspondiente función de conversión de la misma.

3.1.3. Las operaciones y relaciones

Una operación es una función que toma como argumentos elementos de un conjunto y devuelve como resultado otro elemento del mismo. Las operaciones que presenta Fortran sobre los números reales, son las que deben ser definidas en el módulo para el tipo derivado **newreal** (Adición +, Sustracción **-**, Multiplicación *, División / y Exponenciación **).

Para cada una de ellas el módulo las debe definir declarando funciones de la siguiente manera:

```
FUNCTION NR_<OP>_NR ( NR1 , NR2 )
    TYPE (NEWREAL) NR1, NR2, NR_<OP>_NR
    INTENT (IN) NR1, NR2
    INTENT (OUT) NR_<OP>_NR
    <Llamado a la función de biblioteca o código propio>
END FUNCTION
```

Se puede ver a una relación como una función que toma como argumentos elementos de un conjunto y devuelve como resultado un valor lógico, VERDADERO o FALSO, según éstos satisfagan dicha relación o nó. Las relaciones que presenta Fortran sobre los números reales y que se deben especificar dentro del módulo para la nueva aritmética newreal son: Igual **.EQ.**, Distinto **.NE.**, Menor **.LT.**, Menor o igual **.LE.**, Mayor **.GT.** y Mayor o igual **.GE.**.

Estas relaciones se declaran como funciones en forma análoga a las operaciones, exceptuando que en éstas últimas el valor arrojado como resultado es de tipo lógico.

Si estas operaciones y relaciones fueran implementadas por una biblioteca aritmética definida tanto en Fortran como en otro lenguaje, como "C", el cuerpo de las funciones anteriores se reduce a la invocación de la correspondiente función definida en dicha biblioteca.

3.1.4. Las funciones intrínsecas

Se denominan funciones intrínsecas, a aquellas que provee el lenguaje, es decir, que ya están implementadas en él y que pueden ser utilizadas en cualquier programa. Son ejemplos de funciones intrínsecas de Fortran: SIN (seno), COS (coseno), SQRT (raíz cuadrada).

En el módulo deben estar definidas todas aquellas funciones intrínsecas que puedan ser invocadas con argumentos reales. En caso de estar implementadas por una biblioteca externa, su código se limita a la correspondiente invocación de la misma.

3.1.5. Sobrecarga de operadores

OAF90 utiliza la sobrecarga de operadores [Wat90] para resolver los problemas de crear una ligadura entre una variable real y un valor de tipo newreal. Ésta funcionalidad incorporada por Fortran 90, es explotada por el módulo interfaz para invocar implícitamente la subrutina de conversión ya definida con la misma sintaxis de la asignación "=".

El módulo interfaz implementa la sobrecarga declarando **interfaces** y en ellas se invocan a las funciones o subrutinas definidas en los puntos anteriores según corresponda.

Por ej. para la sobrecarga de la asignación:

```
INTERFACE ASSIGNMENT (=)
    MODULE PROCEDURE REAL_TO_NEWREAL
END INTERFACE
```

Gracias a estas sobrecargas de operadores el módulo aritmético permite que dentro del programa original, tengamos asignaciones de valores reales a variables del nuevo tipo newreal, evitando así, modificar el código de las mismas.

Como lo define la sintaxis de Fortran, para realizar alguna de las operaciones que él provee, la escribimos con notación infija, por ejemplo **A + B** en caso de la adición. De igual manera expresamos las relaciones, por ejemplo **A .EQ. B** para la igualdad. El módulo implementa la misma notación las para las operaciones y relaciones de la nueva aritmética, nuevamente, por medio de la sobrecarga de operadores. Ésto no solo provee una sintaxis mas clara y elegante de las operaciones, sino que también evita la traducción de las mismas dentro del código del programa. Tampoco es necesario la especificación de precedencia entre ellas.

3.1.6. Coerciones

Dentro de un programa Fortran podemos tener expresiones que exijan la coerción de constantes al tipo newreal, por ejemplo "A+10.3-B+2" (A y B originalmente de tipo real). Para resolver este problema el módulo interfaz sobrecarga a los operadores de la nueva aritmética de manera que puedan recibir cualquier combinación de tipos válida en sus argumentos. Para cada una de estas combinaciones el módulo interfaz implementa las coerciones necesarias. Esto se realiza invocando a la correspondiente función de conversión y a la operación ya definida entre operandos del tipo **newreal**.

Al igual que con las operaciones aritméticas, podemos tener expresiones o condiciones en las cuales se presenten relaciones heterogéneas, es decir, con argumentos newreal y de otros tipos numéricos. Por ello, nuevamente, el módulo interfaz define la compatibilidad de la nueva aritmética para este tipo de expresiones o condiciones, en forma análoga a lo dicho para las operaciones

Fortran 90, además de respetar la sintaxis de su predecesor para las relaciones, agrega las siguientes formas de expresarlas: Igual ==, Menor <, Menor o igual <=, Mayor >, Mayor o igual >= y Distinto / =. Gracias a que el compilador asocia la nueva sintaxis con la anterior, el módulo interfaz no necesita sobrecargar dichos símbolos.

3.1.7. Entrada/Salida

Fortran 90 provee dos formas equivalentes de manipular la entrada/salida con tipos derivados, actual soporte para estructura de los números de la nueva aritmética dentro del módulo interfaz:

- Especificando cada campo del mismo.
- Manteniendo a la variable como objeto atómico e invocando la respectiva acción sobre ella.

Aunque es claro que, dada la segunda opción, el módulo interfaz no necesita especificar estas funciones de entrada/salida para la nueva aritmética, existen casos en los cuales no funcionan. Dado que los objetos dinámicos deben reservar memoria explícitamente, en caso invocar a una de las funciones mencionadas con ellos, el compilador dará un error. Por lo tanto si en la estructura del tipo derivado existen objetos dinámicos (ver 3.1.1), el módulo debe especificar estas funciones de entrada/salida para la nueva aritmética como subrutinas.

Por otra parte, si la nueva aritmética está implementada en una biblioteca externa, o sea, si la estructura del tipo derivado fue construida *ad hoc* para corresponderse con el tamaño esperado por dicha biblioteca, se invoca al correspondiente procedimiento de la misma.

3.2. Diseño del precompilador

Aquí se describe el proceso de traducción de un programa Fortran a otro, que mantenga su funcionalidad, pero que utilice la nueva aritmética.

Gracias a que el módulo interfaz transparenta la interacción entre Fortran y la biblioteca alternativa, es decir, la sintaxis de las asignaciones, operaciones, relaciones y funciones es igual que la que presenta Fortran para sus reales, las modificaciones que debe realizar el precompilador de OAF90 en el código del programa original se limitan a las siguientes:

- **Sentencias INCLUDE**: Estas declaraciones de Fortran, indican la inclusión sintáctica de uno o varios archivos Fortran en el programa. Estas son frecuentemente utilizadas para modularizar el programa en distintos archivos. Por lo tanto, el precompilador incorpora los nuevos módulos al proceso de traducción.
- **Sentencias USE**: A diferencia de las declaraciones INCLUDE, éstas no son una inclusión sintáctica, sino que definen una verdadera relación de uso de un módulo, respetando la semántica de las declaraciones *privadas* "PRIVATE" y *Públicas* "PUBLIC" existentes en ellos. La traducción de éstos, depende de la semántica particular del problema y se deja a criterio del usuario. El precompilador informa los módulos usados por el programa.
- **Declaraciones de variables**: Estas son las declaraciones que el precompilador modifica, para indicar que el tipo real es ahora, **newreal**. Es por ello que cuando encuentra la declaración de una variable real "REAL", la reemplaza el tipo por "TYPE (NEWRAL)".

  En las declaraciones de variables en Fortran, éstas pueden estar inicializadas. Esta forma de crear ligaduras distinta a la asignación (funcionalidad implementada por módulo newreal vista en 3.1.5), el precompilador simula dicha inicialización generando una asignación al comienzo de las sentencias del correspondiente bloque.
- **Declaraciones de constantes**: Las definiciones de constantes en Fortran se realizan con la declaración "PARAMETER". En este tipo de declaraciones, el precompilador identifica primeramente si las constantes son de tipo real, en base a su declaración. Luego, en caso de ser así, quita la variable de la declaración "PARAMETER", y en su reemplazo genera una asignación de su valor al comienzo de las sentencias del correspondiente bloque.

Aquí se puede observar que, luego de la traducción se "pierde" la definición de *constante*. No obstante, en la hipótesis de la validez del programa original, es decir, que fue compilado con éxito. Fue condición necesaria para ello, que a la variable en cuestión no estuviese del lado izquierdo de ninguna asignación, ni como argumento modificable de ninguna subrutina o función. Es por ello que podemos garantizar la equivalencia entre el programa original y éste modificado, puesto que la única asignación que tiene dicha variable es al principio del bloque, manteniéndose **constante** en el resto de él.

- **Sentencias IMPLICIT**: Como lo expresa su traducción, esta declaración se utiliza para declarar implícitamente el tipo de ciertas variables, las cuales no aparece explícitamente declaradas al principio del bloque. El precompilador redeclara los IMPLICIT reales predeterminados de Fortran como de tipo newreal y redefine de igual manera los IMPLICIT reales declarados en el programa.
- **Inicializaciones DATA**: Por intermedio de esta declaración, Fortran permite inicializar variables. Si bien la sintaxis es distinta de la declaración de constantes, su tratamiento es análogo; el precompilador identifica las variables reales que contiene la declaración, las quita de la misma y genera sus asignaciones.
- **Funciones y subrutinas**: Los parámetros de las funciones y subrutinas y los valores de retorno de esta última son tratados de igual modo que las declaraciones de variables excepto en el caso en que el tipo de retorno esté en el perfil de la misma. En éste último caso el precompilador reemplaza el tipo por newreal.
- **Sentencias READ, WRITE y PRINT**: El precompilador resuelve automáticamente las operaciones de entrada salida sobre el nuevo tipo utilizando las operaciones de entrada salida estándar para los tipos derivados. No obstante OAF90 facilita la posibilidad de implementar la entrada salida mediante procedimientos definidos *ad-hoc* como se analizó en 3.1.7.

  En el último caso, el precompilador reestructura la entrada salida de manera tal que ésta utilice para las variables **newreal** los procedimientos específicos del nuevo tipo que deberán estar definidos en el módulo interfaz.

3.3. Implementación del precompilador

Como se analizó en la sección anterior, los cambios que debe realizar el precompilador dentro del programa para reemplazar el tipo real por otro definido en un módulo externo son mínimos. Es por ello, que para implementar la traducción mediante el precompilador, no se necesita analizar toda la gramática del programa mediante un parser, solo basta un reconocedor de expresiones regulares que pueda ejecutar acciones asociadas a ellas. Estas acciones deberían estar definidas en algún lenguaje que permita manipular fácilmente cadenas de texto, y además, que provea algún tipo de estructura como herramienta auxiliar para procesarlas.

Una de las herramientas existentes que cumplen con estos requisitos es AWK. Se eligió, no sólo por su simplicidad y potencia, sino también porque es libre y está implementada para varias plataformas, entre otras, Linux y Windows.

3.3.1. Proceso de precompilación

El precompilador, abre el archivo con el programa Fortran, antes de comenzar el análisis del mismo, inicializa las variables y estructuras que utiliza en la traducción; estas son:

- **NroLinea**: en ella mantiene el número de la línea en proceso.
- **Tabla de Entidades**: en ella mantiene el nombre del programa y de las distintas funciones y subrutinas definidas en el. Esta tabla delimita los bloques en donde se declaran las variables.
- **Tabla de Variables**: en ella mantiene la información de cada variable real que encuentre.
- **Tabla de asignaciones generadas**: aquí el precompilador almacena temporalmente las asignaciones generadas correspondiente a inicializaciones.

- **Nuevo Archivo**: por cuestiones de eficiencia y avalado por la tecnología actual en cuanto a los tamaños de memoria existentes, durante el proceso de traducción el precompilador utiliza esta estructura (arreglo de strings) como medio temporal en el proceso de traducción.

Una vez inicializadas las estructuras y variables a utilizar, el precompilador comienza con el proceso de analizar, línea por linea, el archivo original. Procesa aquellas que presenten patrones coincidentes con las citadas en 3.2 y además, debe procesar las siguientes expresiones:

- **PROGRAM**: Al inicio del programa. El precompilador no solamente guarda la entidad en la tabla entidades, sino que además interpola una línea con la sentencia "USE MODULE_NEWREAL", la cual indica al programa que use el nuevo módulo aritmético. También reemplaza las declaraciones implícitas predeterminadas de Fortran para que sean del nuevo tipo real, añadiendo la línea "IMPLICIT TYPE(NEWREAL) (A-H, O-Z)".
- **Primera línea de sentencias en una entidad**: En caso de que en el programa original existieran inicializaciones de variables o declaraciones de constantes, el precompilador generó en la tabla **asignaciones** , las asignaciones que las simulan. Éste es el lugar apropiado para incorporarlas al nuevo archivo. La variable **FlagAsig**, modificada en cada incorporación de la tabla, le indica al precompilador si tiene o no inicializaciones simuladas mediante asignaciones para la entidad actual.

Al final del proceso, mediante la regla **END**, el precompilador guarda el programa traducido en un archivo con el mismo nombre que el original que contenía el programa, pero añadiendo el prefijo "new"; así si el programa estaba en un archivo llamado "Program.f90" su traducción estará en uno llamado "newProgram.f90".
Además genera un archivo con los cambios realizados, recorriendo cada entidad y sus respectivas variables. También genera el archivo **OtrosArchivos** que informa el nombre de los archivos referenciados por este programa, los cuales deberían traducirse.

3.4. Asistente para generar módulos interfaces para implementar nuevas aritméticas

Aquellos usuarios avanzados de Fortran que quieran incorporar nuevas aritméticas reales en sus programas mediante OAF90, ya sean aritméticas implementadas en bibliotecas o programarlas directamente en Fortran, deben implementar el módulo interfaz correspondiente a dicha aritmética. Para facilitar esta implementación,OAF90 provee un asistente gráfico que los genera automáticamente a partir de una configuración que contiene las siguientes especificaciones:

- **Estructura**: declaración del tipo derivado NEWREAL.
- **Conversión desde Real**: declaración de la subrutina para convertir un real de Fortran en un newreal.
- **Conversión a Real**: declaración de la subrutina para convertir a partir de un newreal, un real de Fortran.
- **Operaciones**: declaraciones de las funciones que implementan las operaciones aritméticas con operandos de nuevo tipo newreal. Las operaciones son: + (adición), - (sustracción), * (multiplicación), / (División) y ** (exponenciación).
- **Relaciones**: declaraciones de las funciones que especifican las relaciones aritméticas entre números de nuevo tipo newreal. Las relaciones a especificar son: = (igual), y < (menor).
- **Entrada/Salida**: en caso de ser necesario, aquí se implementan las subrutinas de entrada/salida para la nueva aritmética. Ellas son: READNR (Lectura) y PRINTNR (Escritura).
- **Funciones Intrínsecas**: aquí se definen las funciones predefinidas de Fortran que se deseen utilizar.

.

Una vez completados estos datos, el asistente genera el módulo interfaz con todas las funcionalidades descriptas en 3.1 que transparenta la nueva aritmética para cualquier programa Fortran.

4. Conclusiones y trabajos futuros

El polimorfismo por sobrecarga que ha sido incorporado a FORTRAN, a partir FORTRAN 90, ha permitido la implementación de un adaptador de programas a aritméticas de números reales alternativas (OAF 90), sin necesidad de reemplazar las operaciones y funciones implícitas del sistema por llamadas explicitas los módulos que los implementan.

Se ha logrado que OAF 90 realice automáticamente la adaptación de un programa Fortran a una aritmética alternativa sin que el usuario tenga que introducir ningún cambio en su programa, ni tenga que escribir ningún código adicional, siempre y cuando se cuente con una interfaz OAF para la aritmética usada. Una vez escrita una interfaz para una aritmética real, la interfaz puede ser usada para cualquier programa Fortran.

Cualquier Programa FORTRAN que cumpla con la norma de buena programación de respetar el sistema de tipos de Fortran puede ser procesado por OAF. Como el sistema de tipos de FORTRAN no es un sistema fuerte (strong type system [Pra00]) el compilador Fortran 90 acepta programa que violan su sistema de tipos, por ejemplo que subdividan la representación de un valor real en bytes y operen sobre estos para componer un nuevo valor real. Este tipo de violaciones en rigor implementan un nuevo sistema de tipos de semántica indecidible que no puede procesarse automáticamente.

Los casos de uso con los que se ha experimentado el sistema han mostrado la ductilidad y facilidad de uso de OAF 90, tanto para adaptar programas a un aritmética real alternativa cuya interfaz y biblioteca ya están disponibles, como para crear una nueva interfaz e incluso una nueva aritmética usando el asistente del sistema.

El sistema es compatible con bibliotecas escritas en C, C++ o cualquier otro lenguaje de programación. La interfaz, que respeta el pattern adapter, deberá en este caso compatibilizar los diferentes protocolos de invocación.

Se piensa implementar interfaces para bibliotecas existentes de aritméticas reales alternativas que permitan la difusión del uso del sistema OAF por la comunidad de usuarios de FORTRAN. También sería interesante estudiar la posibilidad de enriquecer las facilidades de entrada salida manejadas automáticamente por OAF.

Referencias

[Aho90] Aho A., Sethi R. and Ullman J., *Compiladores: Principiios, Técnicas y herramientas*, Addison-Wesley Iberoamericana, 1990.

[Aho83] Alfred V. Aho, Jeffrey D. Ullman, John E. Hopcroft, *Data Structures and Algorithms*, Addison Wesley, 1983.

[Bai94] David H. Bailey, *A Fortran-90 Based Multiprecision System*, NAS Scientific Computation Branch, NASA Ames Research Center, California, 1994.

[Hol96] Holbig C., Dilverio T.and Claudio D., *Bibliotecas Aplicativas Intervalares*, Universidad Federal da RioGrande do Sul, 1996.

[Agu01] Aguirre J., Dematteis Cynthia *Open Arithmetic Fortran (OAF). Una herramienta para el soporte de múltiples artiméticas de punto flotante*, Anales del CACIQ 2001, 2001.

[Knu73] Knuth, D. E., *Fundamental Algorithms, vol. 3 of The Art of Computer Programming*, Addison-Wesley, Reading, Massachusetts, 1973.

[Gam/95] Gamma , E. et al. *Design Patterns. Elements of Reusable Object Oriented Software*, Addison Wesley 1995.

[Gol91] Goldberg D., *What every computer Scientist should know about Floating-Point Arithmetic*, ACM Computing Surveys, 1991.

[Joh82] Johnson, Lee W., and Riess, R Dean, *Numerical Analysis, 2nd de. Reading*, Mass: Addison-Wesley, 1982.

[Sto80] Stoer, J. and Bulrisch, *Introduction to Numerical Analysis*, New York: Springer-Verlag, Chapter 1, 1980.

[IEEE] IEEE Computer Society, *IEEE Floating-point representations of real numbers*, http://www.math.grin.edu/ stone/courses/fundamentals/IEEE-reals.html, 1996

[Wat90] David A. Watt *Programing Languaje Concepts and Paradigms*, Prentice Hall Internacional (UK) Ltd. 1990

[Ada92] Jeanne C. Adams, Walter Brainerd, J. T. Martin, Brian Smith, J. L. Wagener, *Fortran 90 Handbook*, McGraw-Hill Book Company, 1992

[FFS03] Free Software Foundation, *Effective AWK Programming: A User's Guide for GNU Awk* http://www.gnu.org/software/gawk/manual/gawk.html, 2003

[Wil63] Wilkinson J, E.A., *Rounding Errors in Algebraic Processes*, Printece-Hall, 1963.

[Pra00] Terrence W. Pratt, Marvin V. Zelkowitz *Programming Languages: Design and Implementation, 4th edition*, Prentice Hall PTR, 2000.

[App98] Adrew W. Appel *Modern compiler implementation in java*, Cambridge University Press, 1998.