# Open Arithmetic Fortran (OAF). Una herramienta para el soporte de múltiples aritméticas de punto flotante.

Jorge Aguirre[1]
Cynthia Dematteis[2]

## Resumen

Los errores de redondeo que se propagan al usar una aritmética finita de números reales, constituyen uno de los problemas centrales del cálculo numérico. Ello ha motivado que se escriban diversos paquetes para aritméticas alternativas a las nativas - flotante en simple y doble precisión- No obstante para que un programa haga uso de uno de ellos debe realizarse un importante trabajo de adaptación del programa.
En este trabajo se describe el diseño, implementación y uso de OAF, un sistema que permite variar la aritmética de números reales - distintas precisiones y rangos, intervalar, etc. - sobre la que ejecutará un programa FORTRAN sin necesidad de modificar el programa fuente.
El sistema esta integrado por un precompilador, e interfaces para cada paquete de implementación de una aritmética de números reales a ser usado. OAF transforma el programa para que ejecute sobre un procesador virtual de aritmética real las interfaces implementan este procesador sobre la aritmética elegida.

**Palabras clave**: FORTRAN, error de redondeo, punto flotante, tipo abstracto de datos, análisis numérico, experimentación numérica, compiladores, esquema de traducción.

## 1- Introducción

Como es bien conocido, la propagación de errores de redondeo constituye un importantísimo problema del cálculo numérico[For/67][Wil/60][Col91]Joh82]. Frente a él y desde los mismos inicios de la computación digital, se han desarrollado numerosos paquetes aritméticos, que implementan diversas representaciones de los números reales y sus consecuentes aritméticas. Esta problemática no ha perdido vigencia pese a tratarse de un tema pionero, a tal punto que dio origen a las primeras bibliotecas de procedimientos. Los paquetes que implementan nuevas aritméticas de números reales han continuado apareciendo durante los últimos años, por ejemplo de punto flotante de precisión variable [Bai/94] o de aritmética intervalar [Hol/96]. La disponibilidad de estos paquetes permite al usuario seleccionar la aritmética adecuada a su problema o experimentar con varias de ellas. No obstante, el cambio de una aritmética a otra suele suponer un alto costo. Los paquetes contienen procedimientos capaces de realizar cada una de las operaciones o funciones elementales; para utilizarlos, cada expresión del programa debe ser descompuesta en operaciones simples, que se correspondan con los procedimientos que integran el paquete. Esto supone que el usuario deba traducir cada expresión como una

[1] Universidad Nacional de Río Cuarto. email: jaguirre@dc.uba.ar
[2] Universidad de Buenos Aires. email: cynthia.dematteis@informix.com

secuencia de invocaciones explícitas a estos procedimientos y albergar los resultados intermedios en variables temporales creadas ad hoc.

Como los programas de cálculo suelen ser muy extensos y las expresiones aritméticas constituyen un alto porcentaje de su texto, la descomposición mencionada supone un importante esfuerzo de programación y constituye una importante fuente de introducción de errores de programación, a la vez que obscurece la comprensión del programa. Esto limita seriamente el uso de dichos paquetes como herramienta de experimentación numérica.

La herramienta que se describe en este trabajo, OAF -Open Arithmetic FORTRAN- permite superar las dificultades mencionadas, si se ha usado como lenguaje de programación a FORTAN, cosa habitual en las grandes aplicaciones de cálculo numérico.

OAF precompila un programa FORTRAN, produciendo como salida otro programa FORTRAN que corre sobre una máquina virtual que implementa el *Tipo Abstracto de Datos (TAD) número real* [Wir82], usando los métodos provistos por alguno de los citados paquetes aritméticos. Para implementar esta máquina virtual sobre un paquete dado, sólo hace falta escribir una simple interface. Para correr un programa sobre una aritmética diferente - ya implementada - solo hay que precompilarlo con OAF, sin necesidad de introducir ninguna modificación en su código - como se ve en el ejemplo mostrado en la sección 2.1 - esto basta para que se generen todos los cambios necesarios al programa, adaptándolo para usar la aritmética deseada. Así el usuario estándar podrá usar una biblioteca de aritméticas predefinidas y sólo necesitará realizar la precompilación de su programa mediante OAF para lograr que el mismo quede implementado sobre la aritmética elegida. Por otro lado para incorporar un nuevo paquete que implementa una aritmética real sólo debe escribirse una interface simple, que puede estar escrita en FORTRAN. Esto permite que la tarea pueda ser emprendida por el usuario final sin mayor dificultad.

OAF es absolutamente portable, ya que esta implementado sobre lenguaje c++ estándar. Mientras que su diseño, basado en un esquema de traducción [Aho/86], – figura 2- lo hace que lo hace de fácil modificación.

# 2- Uso de OAF

## *2.1- COMPILACIÓN DE UN PROGRAMA USANDO OAF*

Para utilizar OAF con un programa Fortran se necesitan los siguientes archivos y bibliotecas:

- Paquete aritmético. – biblioteca de módulos objeto que implementan las operaciones básicas en la nueva aritmética <arit-pack>
- Rutinas que implementan la interface con el paquete aritmético a ser usado <call-interface>
- Archivo de especificación de parámetros de las rutinas <rut_pars>
- Archivo opcional <arit_common> con la definición de una zona common para comunicación de las rutinas del paquete

Normalmente el usuario dispondrá de todos estos archivos, no obstante si quisiera incorporar un

nuevo paquete deberá definirlos de la forma que se esquematiza más adelante y se describe en la documentación de OLAF.

El proceso necesario para correr un programa sobre una nueva aritmética consta de los siguientes pasos:

**i- Precompilación**

OAF se invoca con el siguiente comando:

```
oaf <programa_fuente> <rut_pars> [<programa_salida>
[<arit_common> [debug]]]
```

donde:

<*programa_fuente*> es el nombre del programa fuente a ser procesado;

<*rut_pars*> y <*arit_common*> son los archivos ya descriptos;

<*programa_salida*> es el nombre del programa de salida a ser generado por OAF;

<*debug*> igual a 1, habilita la opción de seguimiento de la ejecución (tracing).

**ii- Compilación**

<*programa_salida*> debe compilarse con un compilador Fortran obteniéndose <*programa_objeto*>

**iii- Vinculación –link-edition-**

<*programa_objeto*> debe vincluarse conjuntamente con <call-interfase> y <arit-pack>

**iv- Ejecución**

Finalmente se ejecuta el programa obtenido en el paso anterior.
La entrada y la salida será compatible con el formato especificado en el programa original.

**Ejemplo**

Sea *sample.for* un archivo que contiene el programa mostrado en la figura 1.
Este programa, debido a los errores de redondeo producidos y al hecho de que los resultados temporales se almacenan con mayor precisión que las variables, arroja un resultado inesperado, que desaparece la ser corrido sobre una aritmética de mayor precisión después de haber sido adaptado por OAF.

```
program sample

    real a,b,c,t,r,res
    a=11921.
```

```
b=11800.
c=1000000000.
t=c+a
r=c+b
res = (  t  -  r  )–120
res2= ((c+a)-(c+b))-120
print *, "res = ", res  ," y res2 = ",
end
```

(contenido del archivo *sample.for*)

La compilación y ejecución de este programa imprime:

res = -16.0000 y res2 =1.00000

Sin embargo según la lógica del programa ambos resultados deberían ser iguales.

Para experimentar con una precisión mayor se decide adaptarlo mediante OAF a una aritmética de esas características. Para ello se ejecuta la siguiente secuencia de comandos:

**precompilación:**

oaf sample.for newreal.int new_sample.for

**compilación:**

F77l3 new_sample.for (produce: *new_sample.obj)*

**vinculación:**

```
LINK new_sample double_int double_pack
```

(produce: *new_sample.exe)*

**Ejecución:**

new_sample

res = 1.0000 y res2 =1.00000

El aumento de precisión ha hecho que ahora se obtenga el resultado esperado. Se observa que la adaptación del programa a la nueva aritmética no requirió realizar ningún cambio sobre su texto.

## *2.2- INCORPORACIÓN DE UN NUEVO PAQUETE ARITMÉTICO*

Para incorporar un nuevo paquete a la biblioteca de OAF es necesario escribir los siguientes elementos:

- **Interfaces de invocación**: Un procedimiento, o subrutina, que compatibilice los criterios de llamado entre la maquina virtual que implementa OAF y cada uno de los procedimientos del paquete que ejecutan las operaciones básicas de la aritmética real que el paquete implementa.

Las interfaces de invocación sólo deben implementar el llamado al correspondiente procedimiento del paquete para cada operación elemental, ya que OAF se ocupa del control de la máquina virtual.

Las operaciones provistas por el paquete deben contener al menos a:
las operaciones aritméticas elementales, diádicas y monádicas,
las conversiones de tipo al nuevo formato de representación del tipo real y viceversa y
las operaciones relacionales sobre el nuevo tipo real.

A continuación se muestra, a modo de ejemplo el prototipo de las interfaces de invocación para operaciones diádicas y relacionales.

Prototipo de las interfaces de invocación de los procedimentos correspondientes a operaciones diádicas:

```
SUBROUTINE nombre_interno(operando1, operando2)
CHARACTER operando1(longitud), operando2(longitud)
```

Si esta subrutina corresponde a la implementación de la operación ⊕
ella deberá implementar la siguiente semántica:
operando1 = operando1 ⊕ aperando2

Prototipo de las interfaces de invocación de los procedimientos correspondientes a operaciones relacionales:

```
LOGICAL FUNCTION nombre_interno(operando1,
                                operando2 )
CHARACTER operando1(longitud), operando2(longitud)
```

Si esta interface corresponde a la operación relacional (>)
ella debe implementar la siguiente semántica:
retornar: operando1 (>) aperando2

- **archivo de parámetros.** Este archivo comunica a OAF la longitud en bytes de la representación un *número real* en el paquete y vincula los nombres estándar de las operaciones – usados en FORTRAN, o definidos por OLAF - con sus nombres internos. Es un archivo de texto, cuya primera línea contiene la longitud mencionada y la restantes:

Nombre_estándar, Nombre_interno ; para cada una de las operaciones elementales.

A continuación se muestran algunas líneas del archivo de parámetros usado en el ejemplo de uso de la sección 2.1.

```
30;
+,suma;
   ....
```

```
=,asigna_usuario;
   ....
LE,esmenorig;
   ....
POS,positivo;
   ....
IntToTDA,convInt_a_TDA;
TDAtoReal,convertir_a_Real;
```

- **common del paquete**. Este archivo de texto sólo es necesario si el paquete hace uso de algún área common para compartir información con el programa, caso en el que debe contener su especificación.

## 3- Facilidades de entrada salida.

OAF mantiene la misma entrada y salida del programa original. Para esto cada vez que un número real es leído, su lectura se realiza con el formato especificado en el programa original, sobre una variable del correspondiente tipo nativo y luego se convierte y almacena su valor en el formato en que la aritmética usada implementa al TAD número real. Recíprocamente, para ejecutar una operación de salida de un valor real, OAF convierte su representación en la aritmética usada a la representación estándar y luego usa la especificación de formato dada por el usuario para ejecutar la operación de salida.

Esto garantiza que la versión obtenida con OAF sea absolutamente compatible con los juegos de datos con que corría el programa original. Si se desea obtener un formato de entrada salida específico para la nueva representación, se necesario hacerlo a través de rutinas escritas ad hoc.

## 4- Diseño e implementación de OAF.

La figura 1 muestra la forma en que se ha generado el precompilador.
OAF ha sido generado mediante un Esquema de traducción, especificado en el lenguaje *yacc* [Aho/86]. Se ha usado $c^{++}$ como lenguaje de programación huésped.

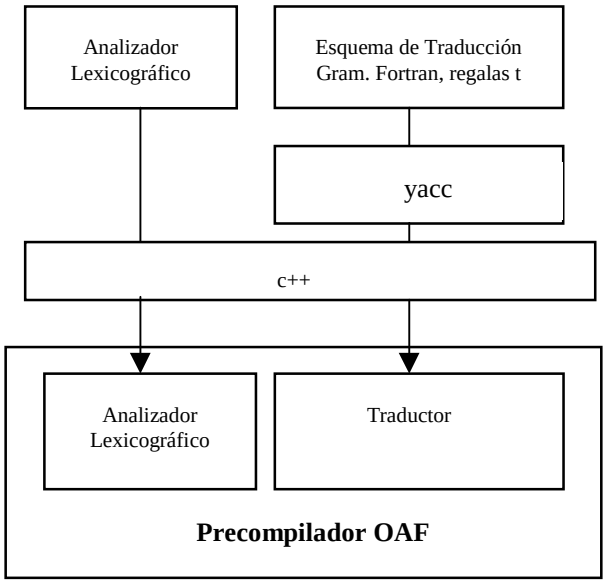


Figura 1

Como es sabido un esquema de traducción especifica un traductor o compilador de un lenguaje, mediante su gramática, enriquecida con reglas de traducción. Estas reglas son trozos de programa escritos en un lenguaje de programación. La gramática que se ha utilizado está basada en la que provee GNU.

El analizador lexicográfico ha sido programado directamente en $c^{++}$, sin usar *lex*, contraparte clásica de *yacc*, para la genración de analizadores lexicográficos; esta decisión ha sido tomada en parte por consideraciones de eficiencia, pero sobre todo, debido a que el diseño del lenguaje FORTRAN, por ser anterior al desarrollo de la teoría de lenguajes formales, no permite que el reconocimiento de un símbolo pueda realizarse sin tener en cuenta el contexto en que aparece. Situación que obliga a complicar la especificación, forzando el estilo de especificación de *lex*.

El diseño basado en un Esquema de Traducción, conjuntamente con la modularidad con que ha sido desarrollado el código, hace que sea posible extender o modificar la funcionalidad de OAF con bajo esfuerzo, ya que su estructura misma vincula el código con la componente sintáctica a la que corresponde.

El sistema OAF está compuesto por dos subsistemas –Figura 2 - , uno fijo que no requiere modificación para ser usado con distintos paquetes aritméticos y otro que constituye la interface con el paquete a ser utilizado; obviamente está última debe ser escrita ad hoc. Si bien se pretende presentar a OAF con una biblioteca de paquetes aritméticos y sus correspondientes interfaces, se ha buscado que la incorporación de una nueva aritmética requiera el menor esfuerzo posible y que pueda ser abordada por un usuario final, sin necesidad de usar otro lenguaje. Para ello se ha usado una interface FORTRAN entre el programa generado por OAF y el paquete aritmético usado y se ha diseñado al precompilador de manera que reciba los datos que necesita sobre el paquete en un archivo de texto.

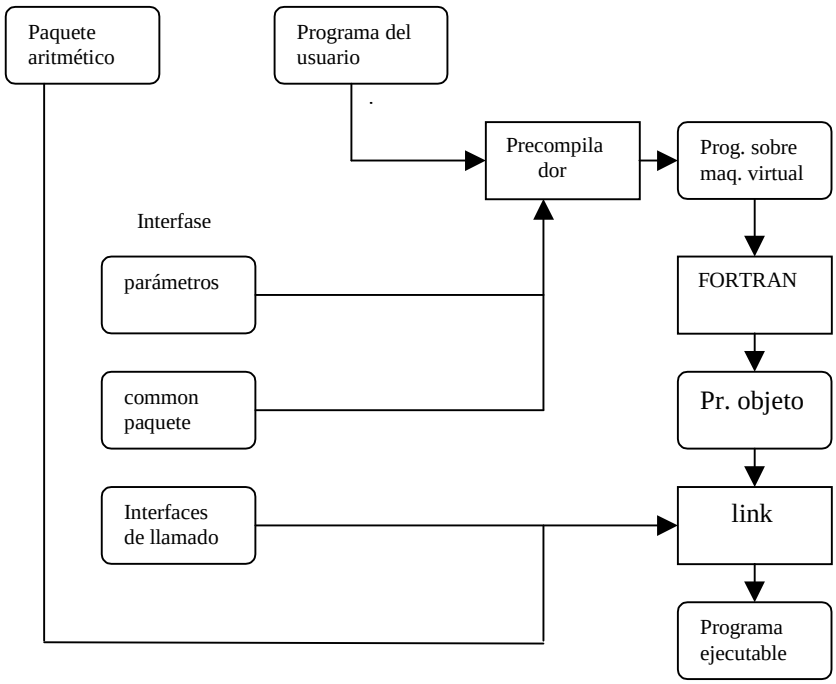


Figura 2

El precompilador genera código para la máquina virtual que provee las operaciones básicas del tipo abstracto de datos –TDA- *número real*. Todas las expresiones del programa de entrada son reemplazas por código para dicha máquina.

La maquina virtual tiene la arquitectura de una máquina pila [Mat/70][Aho/86]. Esta arquitectura cuenta con una pila de números reales y dispone de dos operaciones con un operando *load dirección* y *store dirección*, operaciones que están destinadas a apilar el valor alojado en una dirección y a copiar a una dirección el valor del tope, respectivamente. Las demás operaciones carecen de operando explícito y actúan sobre la pila de la siguiente forma:

las operaciones diádicas ejecutan
t = subtope ⊕ tope ; desapilar; apilar t

mientras que las monádicas
tope = ⊕ tope

Así la sentencia A = B+C*D se traduce:

load *dirección_de_B*
load *dirección_de_C*
load *dirección_de_D*
mult
add
store *dirección_de_A*

La elección de esta arquitectura para la máquina virtual permite encapsular dentro de ella el manejo de variables temporales, cuyo tiempo de vida coincide con el de la evaluación de la expresión. No obstante OLAF adminisrta temporales para otros usos.

Cada implementación del TDA *número real* se logra mediante el uso de un paquete de aritmética real y su correspondiente interface como lo muestra la figura 2.

El precompilador administra un espacio de memoria virtual destinado a alojar los valores de variables del programa en la representación del TAD número real que implementa la aritmética elegida. La longitud que ocupará un número real es comunicada al precompilador por la interface –archivo de parámetros -. La longitud de un valor real es la única información sobre la implementación del ADT que requiere el precompilador, dado que la máquina virtual mediante la interface resuelve todos los otros aspectos.
El precompilador realiza, sobre este espacio de memoria virtual, tanto el tratamiento de los bloques *common* como de las sentencias *equivalence.*
Tanto la memoria virtual como la pila están implementadas sobre un arreglo bidimensional de bytes –MEM- cuya cantidad de filas es la longitud de un número real y cuya cantidad de columnas el tamaño de la memoria virtual, de forma tal que la palabra de dirección i ocupa la i-esima columna (MEM (1, i) .. MEM(*longitud*,i)).

La interface responde al *pattern adapter* [Gam/96] y realiza la adaptación entre las convenciones de invocación usadas por el paquete a ser incorporado y la máquina virtual de OAF. El lenguaje en que está escrito el paquete es irrelevante. Sólo es necesario que el sistema operativo provea un linker que permita vincular módulos escritos en distintos lenguajes.

## 5- Alcances y Limitaciones de la primera versión

OAF traduce correctamente todas los constructores del lenguaje FORTRAN 77, siempre y cuando el programa manipule los números reales como valores del TAD número real, esto es haciendo uso exclusivamente de las operaciones del TAD y de las que sobre ellas se construyan, sin utilizar las carcterísticas particulares de la representación, mediante artificios de programación. Esto supone que las conversiones de tipo deben ser realizadas usando los métodos que el lenguaje provee para ello, coerciones y funciones implícitas de conversión.
Esta restricciones impiden superponer mediante declaraciones *common* o *equivalence* valores reales con valores de otro tipo.
Estos requerimientos de programación coinciden, por otra parte, con los que debe cumplir un programa FORTRAN para ser independiente de la implementación.

Como se ha dicho, la entrada salida del programa migrado por OAF a una nueva aritmética es absolutamente compatible con la del programa original. Esto brinda una absoluta compatibilidad con los lotes de datos de que se disponía. Si se desea aprovechar las características de la nueva representación en la entrada salida, por ejemplo, ingresar o imprimir intervalos como números reales, se deberán escribir los correspondientes procedimientos o se podrá dotar al paquete de los mismos. En este caso para lograr la nueva entrada salida se hace necesario introducir modificaciones en el código del programa.
La primera versión de OAF impone ciertas restricciones sobre el uso de la declaración *equivalence* entre elementos internos de arreglos.

## 6- Conclusiones y trabajos futuros.

Se ha obtenido una versión operativa de OAF, que cumplen con los objetivos esperados y puede transformarse en una herramienta útil. La creación de una biblioteca de paquetes que implementen distintas aritméticas permitirá su uso por la comunidad de cálculo numérico y sólo su adopción como herramienta habitual por dicha comunidad testificará su utilidad.
Se planea, constituir una biblioteca de paquetes. También se tiene interés de implementar una segunda versión para FORTRAN 90.

## BIBLIOGRAFÍA.


[Aho/86] Aho A., Sethi R. and Ullman J., Compiladores: Principiios, Técnicas y herramientas, Addison-Wesley Iberoamericana, 1990

[Bai/94] David H. Bailey, A Fortran-90 Based Multiprecision System, NAS Scientific Computation Branch, NASA Ames Research Center, California, Octubre 1994.

[For/67] Forsythe G. and Moler C.,Computer Solution of Linear Algebraic Systems, Englewood Cliffs, NJ: Prentice-Hall, 1967

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

[Gol/91] Goldberg D., What every computer Scientist should know about Floating-Point Arithmetic, ACM Computing Surveys, 1991

[Hol/96] Holbig C., Dilverio T.and Claudio D., Bibliotecas Aplicativas Intervalares, Universidad Federal da RioGrande do Sul, 1996

[Joh/82] Johnson, Lee W., and Riess, R Dean, Numerical Analysis, 2nd de. Reading, Mass: Addison-Wesley, 1982

[Knu/69] Knuth D.E., The art of Computer Programing, Tomo II, Floating Point Arithmetic,1980
[Mat/70] Matula D., A Formalization of Floating Point Numeric Base Conversion, IEEE Trans. Comp. , 1970
[Mat/76] Matula D., Radix Arithmetic: Digital Algorithms for Computer Architecture, Applied Computation Theory. Englewood Cliffs, NJ: Prentice Hall, 1976
[Pra/96] Pratt, Programming Language, Design and Implentation, 1996.
[Ste/74] Sterbenz, Floating Point Computation, Englewood Cliffs, NJ: Prentice Hall
[Sto/80] Stoer, J. and Bulrisch, Introduction to Numerical Analysis, New York: Springer-Verlag, Chapter 1, 1980
[Usa/87] José M. A. Usategui y Anagasti Pedro, Arquitectura de computadores: fundamentos e intriducción al paralelismo. Paraninfo.1987
[Wil/60] Wilkinson J, Error Analysis of Floating Point Computacions, Num. Math, 1960
[Wil/63] Wilkinson J, Rounding Errors in Algebraic Processes, London: Her Majesty's Stationery Office and Englewood Cliffs, NJ: Prentice-Hall, 1963
[Wir/82] Wirth Niklaus, Introducción a la programación sistemática, El Ateneo, 1982.