Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de ATI (Asociación de Técnicos de Informática). _Novática_ edita también _Upgrade_, revista digital de CEPIS (_Council of European Professional Informatics Societies_), en lengua inglesa.

<http://www.ati.es/novatica/>
<http://www.upgrade-cepis.org/>

ATI es miembro de CEPIS (*Council of European Professional Informatics Societies*) y tiene un acuerdo de colaboración con ACM (*Association for Computing Machinery*). Tiene asimismo acuerdos de vinculación o colaboración con AdaSpain, AI2 y ASTIC

Coordinación Editorial y Redacción Central (ATI Madrid)
Padilla 66, 3º, dcha., 28006 Madrid
Tlf.914029391; fax.913093685 <*novatica@ati.es*>
Composición, Edición y Redacción ATI Valencia
Reino de Valencia 23, 46005 Valencia
Tlf./fax 963330392 <*secreval@ati.es*>
Administración y Redacción ATI Cataluña
Via Laietana 41, 1º, 1ª, 08003 Barcelona
Tlf.934125235; fax 934127713 <*secregen@ati.es*>
Redacción ATI Andalucía
Isaac Newton, s/n, Ed. Sadiel, Isla Cartuja 41092 Sevilla
Tlf./fax 954460779 <*secreand@ati.es*>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza
Tlf./fax 976235181 <*secreara@ati.es*>
Redacción ATI Asturias-Cantabria <*gp-astucant@ati.es*>
Redacción ATI Castilla-La Mancha <*gp-clmancha@ati.es*>
Redacción ATI Galicia
Recinto Ferial s/n, 36540 Silleda (Pontevedra)
Tlf.986581413; fax 986580162 <*secregal@ati.es*>
Suscripción y Ventas: <*http://www.ati.es/novatica/interes.html*>, o en ATI Cataluña y ATI Madrid
Publicidad: Padilla 66, 3º, dcha., 28006 Madrid
Tlf.914029391; fax.913093685 <*novatica.publicidad@ati.es*>

Imprenta: 9·Impressió S.A., Juan de Austria 66, 08005 Barcelona.
Depósito Legal: B 15.154-1975
ISSN: 0211-2124; CODEN NOVAEC

Portada: Antonio Crespo Foix / © ATI 2003

novática
UPGRADE
Revista de la Asociación de Técnicos de Informática

MAYO · JUNIO 2003
163

SUMARIO

Monografía del próximo número:
«Ingeniería del Software: estado de un arte»

Lenguajes informáticos

Evaluación parcial de programas y sus aplicaciones

Pascual Julián Iranzo
Departamento de Informática, Universidad de Castilla - La Mancha; Grupo ELP (Extensiones de la Programación Lógica) del DSIC, Universidad Politécnica de Valencia

<Pascual.Julian@uclm.es>

Resumen: *la Evaluación Parcial (EP) es una técnica de transformación automática de programas que persigue, entre otros objetivos, la optimización de programas con respecto a ciertos datos de entrada; de ahí que también haya recibido el nombre de especialización de programas. Este artículo trata de presentar la EP a un público no experto que desarrolla su actividad en otros campos de la informática. Con este fin se traza una panorámica general del área y de sus técnicas más importantes. Si bien se define con precisión qué se entiende por evaluación parcial y se ilustran sus métodos mediante diversos ejemplos sencillos, este trabajo se ocupa, principalmente, de los objetivos de la evaluación parcial y de sus aplicaciones en diversas áreas científicas y tecnológicas, incidiendo en los beneficios que pueden extraerse de ella.*

Palabras clave*: compiladores e intérpretes, especialización de programas, EP, evaluación parcial, generación automática de programas, mantenimiento del software, optimización de programas, transformación de programas, verificación de programas.*

1. Introducción

La transformación de programas es un método para derivar programas correctos y eficientes partiendo de una especificación ingenua y más ineficiente del problema. Esto es, dado un programa ***P***, se trata de generar un programa ***P'*** que resuelve el mismo problema y equivale semánticamente a ***P***, pero que goza de mejor comportamiento respecto a cierto criterio de evaluación. En la literatura se ha propuesto una gran variedad de transformaciones para mejorar el código. Una de las mejor estudiadas es la *Evaluación Parcial* (EP) de programas, que ofrece un marco unificado para la investigación acerca de los procesadores de lenguajes, en particular, compiladores e intérpretes [16]. La EP es una técnica de transformación de programas que consiste en la especialización de programas respecto a ciertos datos de entrada, conocidos en tiempo de compilación, por lo que también se la denomina *especialización* de programas.

En general, las técnicas de EP incluyen algún criterio de parada para garantizar la terminación del proceso de la transformación. La EP es, por tanto, una técnica de transformación automática, lo cual la distingue de otras técnicas de transformación de programas tradicionales [8] [19]. La EP ha sido aplicada a los lenguajes imperativos tanto como a los declarativos y a una gran variedad de problemas concretos. Una panorámica general sobre este campo y su área de aplicación se presenta en [16]. Un breve pero excelente tutorial sobre EP (si bien centrado en la especialización de programas imperativos y funcionales) es [10]. Otros trabajos que examinan aspectos concretos del área son [2] [14].

La intención al escribir este artículo ha sido mostrar los fundamentos teóricos de la EP, y algunas de sus aplicaciones, en un formato asequible a un lector no especializado. Este artículo se ha estructurado como sigue.

En la sección 2, se describe con precisión el concepto de EP y se introducen algunas de sus principales técnicas. En la sección 3 se presentan algunos de los objetivos más importantes perseguidos por la EP, como son: aumentar la eficiencia de los programas; aumentar la productividad en el desarrollo de las aplicaciones; mejorar la reusabilidad y modularidad del software y conseguir evaluadores parciales autoaplicables (esto es, evaluadores parciales capaces de especializarse a ellos mismos). En la sección 4 se exponen varias de las aplicaciones de la EP en áreas industriales y en otros campos de investigación más básica. Esta sección es una primera aproximación que sólo pretende llamar la atención del lector sobre cada uno de esos campos de aplicación, ya que un tratamiento más técnico está fuera del alcance del artículo.

Posteriormente, el lector puede completar su formación atendiendo a sus intereses; para ello se incluye un anexo en el que se facilitan una serie de direcciones de sitios web donde encontrar información general sobre el campo de la EP, grupos de investigación que desarrollan una actividad intensa en este área y evaluadores parciales desarrollados para diferentes lenguajes (tanto declarativos como imperativos). Finalmente se presentan las conclusiones, seguidas del anexo antes citado.

2. La Evaluación Parcial

La idea de especializar funciones con respecto a uno o varios de sus argumentos es vieja en el campo de la teoría de funciones recursivas, donde recibe el nombre de *proyección*. Esta posibilidad está implícita en el teorema *s-m-n* de Kleene que establece que, dada una función computable $f \in F^{n+m}$, cuyo tipo es $f: S^{n+m} \rightarrow S$, se cumple que la función $f_{e1,\ldots,en} : S^m \rightarrow S$, tal que

$$f_{e1,\ldots,en}(d_1, \ldots, d_m) = f(e_1, \ldots, e_n, d_1, \ldots, d_m),$$

es computable.

Ejemplo 1: sea N el conjunto de los números naturales. La función *suma:* $N^2 \rightarrow N$, definida como $suma(x, y) = x + y$, puede especializarse para un valor conocido de x, por ejemplo *2*, transformándose en una función $suma_2 : N \rightarrow N$, tal que $suma_2(y) = 2 + y$.

Se debe notar que en el ejemplo 1 los valores conocidos simplemente se substituyen por los correspondientes parámetros formales de la función; no se realiza ningún cómputo. Además, en este proceso, que recuerda a la

currificación de funciones en los lenguajes funcionales, el objetivo de lograr una mayor eficiencia está ausente. Los problemas de eficiencia eran irrelevantes para las investigaciones efectuadas por Kleene, centradas en establecer los límites entre lo que es o no es computable. El hecho es que la técnica propuesta por Klenee produce funciones especializadas cuyo comportamiento operacional, en algunos casos, es 'peor' que el de las originales. Por otro lado, la evaluación parcial trabaja con programas (textos) más bien que con funciones matemáticas. Por consiguiente, la EP es una técnica que va más allá de la simple proyección de funciones matemáticas.

La EP establece como ejecutar un programa cuando sólo conocemos parte de sus datos de entrada. De forma más precisa, dado un programa ***P*** y parte de sus datos de entrada ***in1***, el objetivo de la EP es construir un nuevo programa $\boldsymbol{P_{in1}}$ que cuando recibe el resto de los datos de entrada ***in2***, computa el mismo resultado que produce ***P*** al procesar toda su entrada ***in1*** + ***in2***. Esto último asegura la corrección de la transformación efectuada.

El programa que realiza el proceso de EP recibe el nombre de *evaluador parcial* y el resultado de la EP, el programa $\boldsymbol{P_{in1}}$, se denomina *programa especializado*, *programa evaluado parcialmente* o también *programa residual*. La idea que se esconde detrás del proceso de EP consiste en: i) realizar tantos cómputos como sea posible en tiempo de EP, haciendo uso de los datos de entrada conocidos ***in1***, también denominados datos *estáticos* (por contra posición con los datos de entrada desconocidos ***in2***, que son denominados *datos dinámicos*, y que sólo se conocen en tiempo de ejecución del programa residual); ii) generar código relacionado con aquellos cálculos que no puedan realizarse por depender de los datos de entrada desconocidos. Así pues, un evaluador parcial realiza una mezcla de acciones de cómputo y de generación de código; esta es la razón por la que Ershov denominó a la EP *computación mixta* (*mixed computation*).

Cuando se realizan cómputos en tiempo de EP, haciendo uso de los datos de entrada conocidos, decimos que hay *propagación de la información*. El objetivo de la EP es obtener la mejor de las especializaciones posibles maximizando la propagación de la información. Se espera que el programa resultante pueda ejecutarse de forma más eficiente ya que, usando el conjunto de datos (parcialmente) conocidos, es posible evitar algunas computaciones (en tiempo de ejecución) que se realizarán (una única vez) durante el proceso de EP. Para cumplir estos fines, la EP utiliza, además de la computación simbólica, algunas técnicas bien conocidas provenientes de la transformación de programas (funcionales) [8], procurando su automatización:

1. **Definición**: permite la introducción de funciones nuevas o la extensión de las existentes.
2. **Instanciación**: permite especializar una función asignando datos de entrada conocidos a sus argumentos.
3. **Desplegado** (*unfolding*): permite el reemplazamiento de una llamada a función por su respectiva definición.
4. **Plegado** (*folding*): como su nombre indica, es la transformación inversa del desplegado, es decir, el reemplazamiento de cierto fragmento del código por la correspondiente llamada a función.

La aplicación reiterada de cada una de estas reglas de transformación supone la introducción de nuevas instrucciones de programa y da lugar a una secuencia de transformación de la que, finalmente, se extraerá el programa transformado. Una estrategia que reduce el alto grado de indeterminismo existente en la aplicación de las reglas de la transformación, es la presentada en el siguiente esquema básico de transformación:

1. **Repetir** hasta que convenga:
- buscar una **definición**, e
- **instanciar** la definición para permitir,
- pasos de **desplegado** en diversos puntos, seguidos por,
- un **plegado** de las instrucciones resultantes haciendo uso de alguna de las instrucciones obtenidas.

2. Extraer el programa transformado.

De entre todas estas técnicas, el desplegado es la herramienta de transformación fundamental de la EP. Para programas funcionales, los pasos de plegado y desplegado sólo involucran ajuste de patrones (*pattern matching*). En el caso de los programas lógicos, el ajuste de patrones se substituye por la unificación, obteniéndose así una mayor potencia de propagación de la información. Una técnica específica empleada en la EP es la denominada *especialización de puntos de control* del programa, que combina las reglas de definición, desplegado y plegado. Esta técnica puede entenderse como un proceso consistente en una definición al que le sigue una sucesión de pasos de desplegado (tantos como sea posible) que se detienen en cuanto se reconoce una configuración *"*ya vista*"* anteriormente, momento en el que se genera código para llamar a esa configuración *"*ya vista*"* (es decir, se realiza un paso de plegado). En un lenguaje imperativo, un *punto de control* es una etiqueta del programa; en un lenguaje declarativo puede considerarse que es la definición de una función o predicado.

La idea es que una etiqueta o una función del programa ***P*** pueda aparecer en el programa especializado $\boldsymbol{P_{in1}}$ en varias versiones especializadas, cada una correspondiente a datos determinados conocidos en tiempo de EP. Es conveniente notar que esta técnica es un reflejo del esquema básico de transformación anteriormente esbozado. Otra de las técnicas empleadas por la EP es la *abstracción*[1], consistente en generalizar una expresión cuando no es posible su especialización; en cierto sentido, puede verse como la transformación inversa de la instanciación y puede caracterizarse en términos de un proceso de definición al que le sigue uno de plegado.

A continuación aclaramos algunos de los conceptos y técnicas de transformación introducidos, mediante una serie de ejemplos. Con el fin de mantener la discusión en términos tan simples como nos ha sido posible, se han seleccionado los ejemplos atendiendo a su sencillez y adecuación para ilustrar aspectos concretos del proceso de EP. En esta línea de propiciar la sencillez, los ejemplos 2 y 3 hacen uso de una sintaxis funcional (algebraica) en la que los programas son conjuntos de ecuaciones[2] que definen funciones (recursivas).

Esta es una visión muy conveniente a nuestros fines, pues nos permite escapar de los detalles y la complejidad que introduciría el uso de lenguajes de programación de corte imperativo.

Ejemplo 2: sea el fragmento de un programa ***P*** que computa la función x^n:

```
(1)   exp(0,X) =  1
(2)   exp(N,X) =  X * exp(N-1,X)
```

Si queremos especializar el programa para el dato de entrada conocido N=3, los pasos que podría realizar un hipotético

evaluador parcial serían:

```
% Definición
(3)   exp3(X) = exp(3,X)
% Instanciación, desplegado y computación
 simbólica
(4)   exp(3,X) = X * exp(3-1,X)
(5)   exp(3,X) = X * exp(2,X)
(6)   exp(3,X) = X * (X * exp(2-1,X))
(7)   exp(3,X) = X * (X * exp(1,X))
(8)   exp(3,X) = X * (X * (X * exp(1-1,X)))
(9)   exp(3,X) = X * (X * (X * exp(0,X)))
(10)  exp(3,X) = X * (X * (X * 1))
(11)  exp(3,X) = X * (X * X)
% Desplegado final
(12)  exp3(X) = X * (X * X)
```

La ecuación (4) se ha obtenido a partir de la ecuación (2) por instanciación. La (5) de (4) por computación simbólica. La ecuación (6) se ha obtenido desplegando la llamada `exp(2,X)` en la ecuación (5), esto es, sustituyendo `exp(2,X)` por su definición (suministrada por la ecuación (2)). El resto de las acciones se generan por aplicación reiterada de pasos de computación simbólica y desplegado. Finalmente, de este conjunto de ecuaciones podemos extraer el programa especializado $\boldsymbol{P}_3$:

```
exp3(X) = X * (X * X)
```

que computa la función x^3.

De manera simple, podemos entender que la nueva definición de la función `exp3(X)` constituye una especialización del punto de control `exp(N,X)` en la que no ha sido necesario realizar pasos de plegado, debido a que el programa especializado no contiene llamadas a función. Tampoco se han requerido pasos de abstracción para lograr la especialización. Por otra parte, nótese que las ecuaciones (1) y (2), que definen la función original `exp(N,X)`, no forman parte del programa residual debido a que `exp3(X)`, en particular, no contiene llamadas a `exp(N,X)`.

Ejemplo 3: consideremos de nuevo el programa del ejemplo 2. Si queremos especializar el programa con respecto a la expresión `exp(M,B)*exp(N,B)`, los pasos que podría realizar un hipotético evaluador parcial serían:

```
% Definición
pexp(M,N,B) = exp(M,B) * exp(N,B)
% Instanciación, desplegado y com exp(putación
simbólica
pexp(0,N,B) = exp(0,B) * exp(N,B)
pexp(0,N,B) = 1 * exp(N,B)
pexp(0,N,B) = exp(N,B)
% Instanciación, desplegado y computación
simbólica
pexp(M,0,B) = exp(M,B) * exp(0,B)
pexp(M,0,B) = exp(M,B) * 1
pexp(M,0,B) = exp(M,B)
% Desplegado y computación simbólica
pexp(M,N,B) = B * exp(M-1,B) * exp(N,B)
pexp(M,N,B) = B * exp(M-1,B) * B * exp(N-1,B)
pexp(M,N,B) = B * B * exp(M-1,B) * exp(N-1,B)
% Plegado
pexp(M,N,B) = B * B * pexp(M-1,N-1,B)
```

Pudiendo extraerse el programa especializado:

```
pexp(0,N,B) = exp(N,B)
pexp(M,0,B) = exp(M,B)
pexp(M,N,B) = B * B * pexp(M-1,N-1,B)
exp(0,X) =  1
exp(N,X) =  X * exp(N-1,X)
```

que permite el cómputo de la expresión `exp(M,B)*exp(N,B)` de manera más eficiente que el programa original.

La definición de la función `pexp(M,N,B)` constituye una especialización del punto de control `exp(N,X)` que facilia el cómputo de la expresión `exp(M,B)*exp(N,B)`. En este ejemplo se ha podido realizar un paso de plegado, debido a que en el proceso de especialización hemos reconocido la aparición de una regularidad: la expresión «ya vista» `exp(M-1,B)*exp(N-1,B)`, que puede substituirse por su definición `pexp(M-1,N-1,B)`, dando lugar a un paso de plegado. El programa especializado, si bien no obtiene una mejora en cuanto al número de multiplicaciones a realizar, consigue fundir las secuencias de llamadas recursivas iniciadas por `exp(M,B)` y `exp(N,B)`, que deberían realizarse por separado, en una única secuencia en la que se simplifica el control. Si se hubiese partido de un programa iterativo, escrito en un lenguaje de programación convencional, el efecto hubiese sido la combinación de dos bucles en uno solo. Aunque al comentar los ejemplos 2 y 3 no se ha hecho énfasis en el control del proceso de EP, estos ejemplos ilustran una forma de EP, denominada *online*, en la que todas las decisiones de control (e.g. ¿qué evaluar?, ¿cuándo parar?) se toman en tiempo de EP. El próximo ejemplo sirve para ilustrar una aproximación diferente a la EP *online*.

Ejemplo 4: consideremos de nuevo la función del ejemplo 2 pero expresada en un lenguaje imperativo como el lenguaje C.

```
int exp(int n, int x)
     {
          int exponencial = 1;
          while (n>0) {
           exponencial = exponencial * x;
              n = n - 1;
          };
          return exponencial;
      }
```

Un evaluador parcial típico de un lenguaje imperativo necesita que se le proporcione como entrada el texto del programa así como ciertas especificaciones que le permitan distinguir entre los datos estáticos y los dinámicos. El usuario podría indicar que el primer argumento es estático y el segundo dinámico. Con estas indicaciones el evaluador parcial realiza un análisis del programa en el que introduce una serie de anotaciones, indicando qué partes del código del programa original serán evaluadas en tiempo de EP. En el código que se muestra a continuación estas anotaciones aparecen en negrita.

```
int exp(int n, int x)
    {
          int exponencial = 1;
          while (n>0) {
          exponencial = exponencial * x;
          n = n - 1;
          };
          return exponencial;
    }
```

Una vez realizado este análisis, el usuario debe suministrar valores a los argumentos estáticos, e.g. asignar el valor 3 al argumento n de la función. Entonces, el evaluador parcial

obtiene el siguiente programa residual:

```
int exp3(int x)
    {
        int exponencial = 1;
        exponencial = exponencial * x;
        exponencial = exponencial * x;
        exponencial = exponencial * x;
        return exponencial;
    }
```

Los evaluadores parciales también permiten un postproceso de *compresión de código* que conduciría al programa transformado óptimo:

```
int exp3(int x)
    {
        int exponencial;
        exponencial = x * x * x;
        return exponencial;
    }
```

Este ejemplo revela que uno de los efectos de la EP consiste en eliminar la parte «interpretativa» de los programas (imperativos): el bucle en el que se comprueba reiteradamente si «`n>0`» ha desaparecido del programa residual.

La clasificación de las variables del programa en estáticas o dinámicas recibe el nombre de *división*. Esta tarea es más compleja de lo que podría suponerse a simple vista, siendo difícil de implementar de forma automática, por lo que suele requerir la intervención humana. El proceso de computar una división adecuada partiendo de una división inicial provisional de las variables que se consideran la entrada del programa, recibe el nombre de *binding-time analysis* pues en él se determina el momento en el cual puede computarse el valor de una variable, es decir, cuándo un valor se enlaza a la variable. En el ejemplo 4 se ha descrito la EP como un proceso consistente en varias fases: *binding-time analysis*, generación de anotaciones y EP propiamente dicha. Este tipo de aproximación se denomina EP *offline*, por contraposición con la EP *online*. Debido a que el proceso de *binding-time analysis* es siempre aproximado y porque en un evaluador parcial *online* la decisión de qué expresión debe evaluarse se toma en tiempo de EP, lo que supone una ventaja, los evaluadores parciales *online* permiten alcanzar una mayor precisión en la especialización de los programas que los evaluadores parciales *offline*. En los evaluadores parciales *offline* es crítico encontrar la división adecuada ya que ésta determina el grado de especialización que se alcanzará [10].

Un tema importante relativo a la EP es su capacidad para reestructurar los puntos de control. La reestructuración de puntos de control tiene que ver con las relaciones que se establecen entre los puntos de control del programa original y los del programa residual. Podemos distinguir las siguientes capacidades de reestructuración:

- *Monovariante*: cualquier punto de control del programa original da lugar, como mucho, a un punto de control en el programa residual; «como mucho» quiere decir que un punto de control del programa original puede desaparecer (esto es., puede dar lugar a cero puntos de control) en el programa residual. El ejemplo 2 muestra una reestructuración monovariante, en la que el punto de control `exp(N,X)` del programa original da lugar a un único punto de control especializado `exp3(X)` en el programa residual.
- *Polivariante*: cualquier punto de control del programa original puede dar lugar a uno o más puntos de control en el programa residual. El ejemplo 3 muestra una reestructuración polivariante, en la que el punto de control `exp(N,X)` del programa original da lugar a dos puntos de control, `exp(N,X)` y `pexp(M,N,X)`, en el programa residual.
- *Monogenético*: cualquier punto de control del programa residual se produce a partir de un único punto de control del programa original. El ejemplo 2 muestra una reestructuración monogenética, en la que punto de control especializado `exp3(X)`, en el programa residual, se produce a partir del punto de control `exp(N,X)` del programa original.
- *Poligenético*: cualquier punto de control del programa residual se produce a partir de uno o más puntos de control del programa original. El ejemplo 3 muestra una reestructuración poligenética, en la que el punto de control especializado `pexp(M,N,X)`, en el programa residual, combina varias definiciónes de función del programa original: la función `exp(N,X)` y el operador «*».

Una buena técnica de EP debe ofrecer una capacidad de reestructuración tanto polivariante como poligenética. En [1] [17] se detalla un algoritmo de EP *online*, capaz de producir especialización polivariante y poligenética, que engloba la EP de programas lógicos y la EP de programas funcionales.

3. Objetivos de la Evaluación Parcial

En esta sección se discuten varios de los objetivos más importantes de la EP.

3.1. Aumento en la eficiencia de los programas

Una de las principales motivaciones de la EP es el aumento en la *eficiencia* (*speedup*) de los programas. Debido a que parte de los cómputos se han realizado previamente, en tiempo de EP, esperamos que el programa especializado sea más rápido que el programa original. Es común realizar una medida del aumento de la eficiencia, obteniendo la razón entre el tiempo de ejecución del programa original y del especializado [16]. A la hora de medir la eficiencia de la EP, debe considerarse también el coste del tiempo de especialización del programa original. La EP es claramente ventajosa cuando un (procedimiento de un) programa debe ejecutarse reiteradamente para una porción de su entrada, ya que entonces el coste que pueda suponer la especialización del programa será ampliamente amortizado por las sucesivas ejecuciones del programa especializado, que será más 'rápido' que el original. Neil D. Jones argumenta en [16] que la EP puede ser ventajosa incluso para una única ejecución, ya que muchas veces sucede que el tiempo invertido en la especialización del programa original sumado al tiempo de ejecución del programa especializado es inferior al tiempo que resultaría si se ejecutase el programa original con toda su entrada.

Como ejemplo, puede afirmarse que el evaluador parcial de programas declarativos multiparadigma INDY (ver Anexo) consigue mejoras en la eficiencia de los programas especializados que en muchos casos superan el 100%.

3.2. Productividad, reusabilidad y modularidad

Otro objetivo de la EP es propiciar la *productividad* en el desarrollo de programas mediante el aumento de la *reusabilidad* y la *modularidad* del código. De todos es sabido que es más fácil establecer el significado declarativo (correcto) para una especificación sencilla de un problema;

por contra, la ejecución de esta especificación como programa puede resultar ineficiente.
Un evaluador parcial puede facilitar y hacer más ágil el desarrollo de los programas al permitir explotar una biblioteca de plantillas genéricas y simples (cuyo significado declarativo sea, sin duda, el esperado) que posteriormente se especializan de forma automática para producir un código más eficiente. La corrección del proceso de EP asegura la corrección semántica del programa especializado. Disponer de una biblioteca de plantillas genéricas, para las tareas más comunes, también mejoraría la reusabilidad del código.

Por otra parte, cuando programamos, muchas veces nos encontramos con un conjunto de tareas similares para resolver y que corresponden a diferentes aspectos de un mismo problema. Una forma de afrontar esta situación es escribir un procedimiento específico y eficiente para cada una de estas tareas. Podemos enumerar dos desventajas en esta forma de proceder:
1. Debe de realizarse un exceso de programación, lo que aumenta el coste de creación del programa y de su verificación.
2. El mantenimiento del programa se hace más dificultoso, ya que un cambio en las especificaciones puede requerir el cambio de cada uno de los procedimientos.

Con ser grave la primera de las deficiencias apuntadas, la segunda es la que puede producir mayores costes a largo plazo. Muchos estudios indican que el mayor coste en el *ciclo de vida* de un programa no es el coste inicial de diseño, codificación y verificación, sino el coste posterior asociado al mantenimiento del programa mientras está en producción y uso. Una solución alternativa, que elimina las deficiencias comentadas anteriormente, consiste en escribir un procedimiento altamente parametrizado capaz de solucionar cada uno de los aspectos del problema. Nuevamente, podemos apuntar dos deficiencias:
1. La dificultad propia de programar un procedimiento genérico que cubra todas las alternativas de forma eficiente.
2. La ineficiencia inherente a este tipo de procedimientos, ya que un procedimiento altamente parametrizado gastará mucho de su tiempo de ejecución en la comprobación e interpretación de los parámetros y (relativamente) poco tiempo en los cómputos que debe realizar.

La EP puede ayudarnos a vencer esta disyuntiva. Podemos escribir un procedimiento genérico altamente parametrizado (posiblemente ineficiente) y utilizar un evaluador parcial para especializarlo, suministrando los valores de los parámetros adecuados para cada una de las tareas especificas a resolver. Esto permite obtener automáticamente un conjunto de procedimientos especificos y eficientes, tal y como deseábamos, sin aumentar las tareas de programación y mantenimiento.

3.3. Autoaplicación y generación automática de programas

Uno de los objetivos más perseguidos en el campo de la evaluación parcial es lograr evaluadores parciales autoaplicables. Esto es, evaluadores parciales que pueden especializarse con respecto a ellos mismos. La autoaplicación permite llevar a la práctica los resultados teóricos formulados por Futamura y Ershov, que propician la generación automática de programas y que hoy se conocen como proyecciones de Futamura. La primera proyección afirma que se puede compilar un programa fuente especializando su intérprete con respecto a dicho programa fuente. La segunda dice que se puede obtener un compilador mediante autoaplicación del evaluador parcial, esto es., especializando el propio evaluador parcial con respecto a un intérprete del lenguaje. La tercera establece que es posible obtener un generador de compiladores (capaz de transformar un intérprete en un compilador) mediante una doble autoaplicación del evaluador parcial. Así pues, la EP permite la compilación (primera proyección), la generación de compiladores (segunda proyección) y la generación de generadores de compiladores (tercera proyección).

Dentro de la generación automática de programas una de las aplicaciones más notables es la *generación de compiladores dirigida por la semántica* [20]; por ello entendemos lo siguiente: dada una especificación de un lenguaje de programación, basada en una semántica formal[3], transformarla automáticamente en un compilador. La motivación para la generación automática de compiladores es clara: el ahorro en esfuerzos de programación que supone la construcción de un compilador, que en ocasiones no es correcto con respecto a la semántica propuesta para el lenguaje que compila. La corrección de la EP permite que la transformación automática de una especificación semántica de un lenguaje en un compilador haga desaparecer estos errores. Las tareas de diseñar la especificación de un lenguaje, escribir el compilador y mostrar la corrección del compilador, se reducen a una sola tarea: escribir la especificación del lenguaje en una forma adecuada para ser la entrada de un generador de compiladores. Como puede apreciarse, la EP y la autoaplicación tienen unas posibilidades muy prometedoras. Aunque todavía se necesita algún esfuerzo de investigación para entender perfectamente su teoría y sus técnicas prácticas, la EP ya ha tenido sus primeras aplicaciones industriales en diversos campos tecnológicos.

4. Aplicaciones de la Evaluación Parcial

La EP ha sido aplicada intensivamente tanto a los lenguajes imperativos (e.g., el lenguaje C) como a los declarativos (e.g., el lenguaje PROLOG, el lenguaje Scheme y otros) y extensivamente a una gran variedad de problemas concretos, una muestra de los cuales se presenta a continuación:

4.1. Mantenimiento del software y comprensión de programas

En [6] [7] se desarrollan técnicas para el mantenimiento del software y la comprensión de programas (*program understanding*) basadas en la EP. Los autores desarrollan un evaluador parcial para programas escritos en el lenguaje FORTRAN que aplican a la comprensión de viejos programas científicos (para la gestión de centrales atómicas y satélites de comunicaciones). Con el paso del tiempo, estos programas se han hecho inmanejables, debido a numerosas extensiones, requiriéndose un gran esfuerzo para su mantenimiento. El evaluador parcial utiliza, principalmente, propagación de constantes y simplificación de alternativas por una de sus ramas con el fin de obtener un programa (especializado para ciertos valores de sus variables) más simple y fácil de entender.

4.2. Panificación de tripulaciones en compañías aéreas

En una compañía aérea, después de los gastos en combustible se sitúan en cuantía los gastos que originan el movimiento de las tripulaciones. En las grandes compañías aéreas estos gastos pueden superar los mil millones de dólares. Por consiguiente, se ha invertido un gran esfuerzo en mejorar la planificación relativa a estos aspectos. En [3] se presenta una solución a este problema basada en el uso de la EP para la

mejora de los complicados programas de planificación que son necesarios. El sistema de planificación de tripulaciones desarrollado por Carmen Systems AB incorpora un evaluador parcial, escrito en el lenguaje funcional Haskell, que consigue aumentos en la eficiencia con respecto a los programas originales de entre un 30% y 65%. Si se tiene en cuenta que la ejecución de este tipo de programas requiere varias horas, ese aumento en la eficiencia supone una ganancia considerable. En la actualidad el sistema es utilizado por Lufthansa, la mayor compañía aérea de la Unión Europea.

4.3. Protocolos de procedimiento de llamada remota

La EP también se ha empleado para la optimización de *software* de sistemas. Un protocolo de procedimiento de llamada remota permite que un procedimiento remoto se ejecute aparentemente como si se tratase de uno local, siendo transparente para el usuario del sistema el hecho de que, en realidad, los cómputos tengan lugar en una máquina remota dentro de un sistema operativo distribuido. En [18] se informa de la optimización de partes del protocolo RPC (*Remote Procedure Call*) de Sun, mediante el empleo de Tempo [12], un evaluador parcial para el lenguaje C. Se han obtenido mejoras en la eficiencia del orden del 1.35 en los procedimientos de codificación/ decodificación de argumentos y de hasta un 3.75 en el protocolo RPC mismo.

4.4. Simulación de circuitos

Los simuladores de circuitos toman como entrada una descripción de un circuito eléctrico, construyen las ecuaciones diferenciales que describen su comportamiento y las resuelven mediante el empleo de métodos numéricos. Berlin y Weise [4] citan mejoras importantes en la eficiencia como resultado de especializar un simulador de circuitos, escrito en el lenguaje funcional Scheme, para un circuito particular.

4.5. Software adaptable

Recientemente, en [9] se ha estudiado cómo emplear la EP para incluir un comportamiento adaptable en programas Java existentes. Para conseguir este tipo de comportamiento se emplean las denominadas *clases especializadas*. La idea que se esconde detrás de las clases especializadas consiste en asociar implementaciones alternativas a una clase Java ordinaria. Las implementaciones alternativas se diferencian por su estado interno (una serie de predicados sobre los atributos). El comportamiento adaptable consiste en utilizar la implementación alternativa adecuada al estado interno del objeto. De esta forma, el objeto se adapta por sí mismo a la situación descrita por la clase especializada. Aquí, el papel de la evaluación parcial es suministrar, automáticamente, una implementación optimizada para cada caso específico.

Las técnicas de EP y adaptabilidad del software también se han aplicado a los sistemas operativos [5]. Por último, en [11], se sugiere como utilizar las clases especializadas para integrar la EP en el proceso de desarrollo del *software*. También, en esta referencia, se comentan otras aplicaciones concretas de la EP en el campo de la ingeniería del *software.*

4.6. Verificación de programas

Una de las mayores dificultades a la hora de emplear técnicas de verificación formal, como la comprobación de modelos (*model checking*), es la generación de modelos a partir del programa fuente que puedan constituir la entrada de diversas herramientas para la comprobación de modelos. Para cubrir esta tarea es necesario el empleo de técnicas muy sofisticadas de transformación, abstracción y análisis de programas [13]. En [15] se utiliza una combinación de las técnicas de interpretatación abstracta y EP para la construcción de modelos abstractos, a partir del código fuente de programas escritos en el lenguaje ADA, para la verificación de dichos programas. Posteriormente, esas mismas técnicas se han aplicado al lenguaje Java dentro del proyecto Bandera (ver Anexo).

5. Conclusiones

En este artículo se ha introducido la técnica de transformación automática de programas denominada evaluación parcial y se han descrito sus características y principales objetivos. También se han presentado algunas de sus aplicaciones prácticas más recientes, de interés científico y técnologico, como son: mantenimiento del *software* y comprensión de programas; optimización de *software* de aplicación y de sistemas; simulación de circuitos; obtención de software adaptable; y verificación de programas. Por la relevancia de los objetivos perseguidos, su incidencia positiva en el desarrollo del *software* y el interés de sus aplicaciones, se puede afirmar que las ventajas prácticas de la evaluación parcial están fuera de toda duda y que el uso de sus técnicas en la ingeniería del *software* puede resultar altamente beneficioso.

Referencias

[1] E. Albert, M. Alpuente, M. Falaschi, P. Julián, and G. Vidal. «Improving Control in Functional Logic Program Specialization». In *Proc. of SAS*, Springer LNCS 1503:262-277, 1998.

[2] M. Alpuente, M. Falaschi, and G. Vidal. «Partial Evaluation of Functional Logic Programs». *ACM TOPLAS*, 20(4):768-844, 1998.

[3] L. Augustson. «Partial Evaluation in Aircraft Crew Planning». In *Proc. of PEPM*, *ACM Sigplan Notices*, 32(12):127-136, 1997.

[4] A. Berlin and D. Weise. «Compiling Scientific Code Using Partial Evaluation». *IEEE Computer*, 23(12):25-37, 1990.

[5] A. Black, C. Consel, C. Cowan, C. Krasik, C. Pu, E.N. Volanschi, and J. Walpole. «Specialization classes: An object framework for specialization». In *fifth IEEE IWOOOS*, pp 286-300. 1996. Also available at <http:// compose.labri.fr/papers/#spec_decl>.

[6] S. Blazy and P. Facon. «Partial evaluation for the understanding of fortran programs». In *Soft. Engineering and Knowledge Engineering*, pp 517-525, 1993.

[7] S. Blazy and P. Facon. «An automatic interprocedural analysis for the understanding of scientific application programs». In *Dagstuhl Int'l Seminar on Partial Evaluation*, Springer LNCS 1110:1-16, 1996.

[8] R.M. Burstall and J. Darlington. «A Transformation System for Developing Recursive Programs». *Journal of the ACM*, 24(1):44-67, 1977.

[9] C. Consel, C. Cowan, G. Muller, and E.N. Volanschi. «Declarative specialization of object-oriented programs». In *ACM SIGPLAN conf. on OOPSLA*, pp 286-300, 1997. Also available as TR RR-3118, INRIA, 1997.

[10] C. Consel and O. Danvy. «Tutorial notes on Partial Evaluation». In *Proc. of POPL*, pp 493-501, ACM Press, 1993.

[11] C. Consel, L. Hornof, J. Lawall, R. Marlet, G. Muller, J. Noyé, S. Thibault, and E.N. Volanschi. «Partial Evaluation for Software Engineering». In *ACM Computing Surveys*, 30(3), 1998.

[12] C. Consel, L. Hornof, F. Nöel, J. Noyé, and E.N. Volanschi. «A Uniform Approach for Compile-Time and Run-Time Specialisation». In *Proc. of Dagstuhl Seminar on Partial Evaluation*, Springer LNCS 1110:54-72, 1996.

[13] J. Corbett, M. Dwyer, J. Hatcliff, S. Laubach, C.S. Pasareanu, Robby, and H. Zheng. «Bandera : Extracting finite-state models from java source code». In *Proc. of ICSE 2000*, 2000. Available at <http:// www.cis.ksu.edu/santos/bandera/>.

[14] J. Gallagher. «Tutorial on Specialisation of Logic Programs». In *Proc. of PEPM*, pp 88-98. ACM Press, 1993.

[15] J. Hatcliff, M.B. Dwyer, and S. Laubach. «Staging static analyses using abstraction-based program specialization». In *Proc. of PLILP*.

Springer LNCS 1490, 1998.
[16] N.D. Jones, C.K. Gomard, and P. Sestoft. «*Partial Evaluation and Automatic Program Generation*». Prentice-Hall, 1993.
[17] P. Julián-Iranzo. «*Especialización de Programas Lógico Funcionales Peresosos*». PhD thesis, DSIC-UPV, May. 2000.
[18] R. Marlet, G. Muller, and E.N. Volanschi. «Scaling up Partial Evaluation for optimizing the sun commercial rpc protocol». In *Proc. of PEPM*, *ACM Sigplan Notices*, 32(12):116-126, 1997.
[19] A. Pettorossi and M. Proietti. «Transformation of Logic Programs: Foundations and Techniques». *Journal of Logic Programming*, 19,20:261-320, 1994.
[20] M. Tofte. «*Compiler Generators: What They Can Do, What They Might Do, and What They Will Probably Never Do*». *EATCS Monographs* 19. Springer-Verlag, 1990.

Anexo: la Evaluación Parcial en Internet

Sin animo de ser exhaustivo, en este anexo se enumeran y comentan algunas direcciones de interés sobre EP. Debido a la inestabilidad de la estructura de los sitios web de Internet, la información aquí suministrada es presumible que no sea un fiel reflejo de la realidad futura y tenga fecha de caducidad.

Informaciones de caracter general

<http://www.dina.kvl.dk/~sestoft/pebook/**>:** texto completo y material en línea relacionado con el libro *«Partial {E}valuation and Automatic Program Generation*» de N.D. Jones, C.K. Gomard y P. Sestoft [16].

Proyectos y grupos de investigacion

– Proyecto Bandera <http://www.cis.ksu.edu/santos/bandera/>: proyecto del laboratorio SAnToS (*Laboratory for Specification, Analysis and Transformation of Software*) de la Universidad Estatal de Kansas. El objetivo del proyecto es la definición de técnicas de transformación para la construcción de modelos que permitan la verificación de programas.
– Grupo CLIP Lab <http://www.clip.dia.fi.upm.es/**>: e**l grupo CLIP Lab (*Computational Logic, Implementation, and Parallelism Laboratory*) de la Universidad Politécnica de Madrid. Este grupo tiene una amplia experiencia en el campo de la transformación y optimización de programas declarativos mediante técnicas de especialización abstracta.
- Proyecto Compose <http:// compose.labri.fr/>: proyecto conjunto del INRIA (*Institut National de Recherche en Informatique et en Automatique*), el IRISA (*Institut de Recherche en Informatique et Systèmes Alèatoires*) y la Universidad de Rennes para el diseño y desarrollo de programas y sistemas adaptables mediante el empleo de técnicas de EP.
– Grupo DTAI <http://www.cs.kuleuven.ac.be/~dtai/**>:** el grupo DTAI (*Declaratieve Talen en Artificiele Intelligentie*) de la Universidad Católica de Leuven, Bélgica. Realizan una extensa investigación en el campo de la EP de programas lógicos.
– Grupo ELP <http://www.dsic.upv.es/users/elp/**>:** Grupo ELP (Extensiones de la Programación Lógica) de la Universidad Politécnica de Valencia. Sus áreas de interés se orientan, fundamentalmente, a la integración de lenguajes lógicos y funcionales y a la EP de programas lógico-funcionales, así como otras técnicas de transformación y manipulación de programas.
- Grupo TOPPS <http://www.diku.dk/forskning/topps/**>: e**l grupo TOPPS (*Teori Og Praksis for ProgrammeringsSprog*) del Departamento de Ciencias de la computación de la Universidad de Copenhague (DIKU - *Datalogisk Institut Kobenhavns Universitet*) puede considerarse como el grupo pionero en el estudio de la EP. El grupo ha impulsado una buena parte de los desarrollos teóricos y prácticos en este área de conocimiento.

3. Evaluadores parciales y otros productos

– C-Mix <http://www.diku.dk/forskning/topps/activities/cmix/**>:** evaluador parcial *offline* para el lenguaje C desarrollado por el grupo TOPPS del DIKU, Dinamarca.
– DPPD <http://www.staff.soton.ac.uk/~mal/systems/dppd.html**>:** DPPD es el acrónimo de *«Dozens of Problems for Partial Deduction library of benchmarks»*, una librería de programas lógicos, con los que realizar *benchmarks*, mantenida por M. Leuschel.
– ECCE <http://www.staff.soton.ac.uk/~mal/systems/ecce.html**>:** evaluador parcial *online* para programas lógicos implementado por M. Leuschel en la Universidad Católica de Leuven, Bélgica.
– INDY <http://www.dsic.upv.es/users/elp/indy/**>: e**l sistema INDY (*Integrated Narrowing - Driven specialization system*) es un evaluador parcial *online* para programas lógico-funcionales desarrollado por E. Albert y G. Vidal en la Universidad Politécnica de Valencia, España.
- JSCC <http://compose.labri.fr/prototypes/jscc/**>:** especializador de clases Java desarrollado, dentro del proyecto Compose, en la Universidad de Rennes, Francia.
– Mixtus <http://www.sics.se/ps/mixtus.html**>:** evaluador parcial *online* para el lenguaje Prolog desarrollado por D. Sahlin en el SICS (*Swedish Institute of Computer Science*), Suecia.
– PEVAL <http://www.dsic.upv.es/users/elp/peval/>: un evaluador parcial para el lenguaje de programación multiparadigma Curry, desarrollado por E. Albert y G. Vidal en la Universidad Politécnica de Valencia, España, y Michael Hanus de la Universidad Christian-Albrechts de Kiel, Alemania.
- SAGE <http://www.cogsci.ed.ac.uk/~corin/goedel.html**>:** SAGE (*Self-Applicable Goedel partial Evaluator*) es un evaluador parcial autoaplicable para programas Goedel desarrollado por Corin Gurr en la Universidad de Bristol, Inglaterra.
– Schism <http://compose.labri.fr/prototypes/schism/>: evaluador parcial *offline* autoaplicable desarrollado por Charles Consel, para un subconjunto del lenguaje Scheme.
– Similix <http://www.diku.dk/forskning/topps/activities/similix.html>: Similix fue implementado por Bondorf y Danvy en el DIKU, Dinamarca. En su origen, fue un evaluador parcial *offline* autoaplicable para un subconjunto (de primer orden) del lenguaje Scheme. En la actualidad, se ha logrado extender a un amplio subconjunto de dicho lenguaje con características de orden superior.
– SML-mix <http://www.diku.dk/forskning/topps/activities/sml-mix.html**>:** prototipo de evaluador parcial para un subconjunto apreciable del lenguaje funcional Standard ML. Fue desarrollado primariamente por L. Birkedal y M. Welinder en el DIKU, Dinamarca.
– Tempo <http://compose.labri.fr/prototypes /tempo/**>:** evaluador parcial *offline* para el lenguaje C que está siendo desarrollado conjuntamente por el IRISA, el INRIA y la Universidad de Rennes dentro del proyecto Compose. Tempo puede considerarse como el evaluador parcial para C más preciso de entre todos los existentes.

Notas

[1] La abstracción es una técnica compleja cuya caracterización precisa está fuera del alcance pretendido en esta introducción.
[2] En este contexto, las ecuaciones juegan el mismo papel que las instrucciones en los lenguajes de programación más convencionales.
[3] Nótese que la semántica operacional de un lenguaje puede considerarse que es un intérprete (abstracto) del lenguaje.

Agradecimientos

Agradezco a **María Alpuente** la lectura atenta de un primer borrador de este trabajo y sus valiosos comentarios que han contribuido grandemente a su mejora.