UN ENTORNO INTEGRADO DE PROGRAMACIÓN PARA EL LENGUAJE FUNCIONAL HOPE+

Velázquez Iturbide, J. Ángel
Belmonte Artero, Melchor
Castillo Sánchez, Juan Antonio

Dpto. de Lenguajes y Sistemas Informáticos e Ingeniería de Software
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo s/n, Boadilla del Monte, 28660 Madrid, España

RESUMEN

Se presenta un entorno integrado de programación para el lenguaje funcional Hope+. El conjunto de facilidades y herramientas proporcionadas permiten interactuar con el exterior, editar programas, evaluar expresiones y gestionar el entorno, todo ello bajo una interfaz de usuario homogénea. Las principales novedades residen en las facilidades de evaluación de expresiones: elección de la estrategia de evaluación y combinación de varios modos de evaluación.

PALABRAS CLAVE: entornos de programación, entornos integrados, interfaces de usuario, programación funcional, estrategias de evaluación, modos de evaluación.

1. INTRODUCCIÓN

La programación funcional ha conocido un gran auge en la década de los años ochenta, evolucionando desde el lenguaje Lisp, con sus diversos dialectos, hasta lenguajes puramente funcionales con evidentes mejoras sintácticas y semánticas [8]. Esto unido a la gran sencillez del paradigma funcional ha conllevado que sea un tema cada vez más frecuente de la enseñanza de la programación [13].

Sorprendentemente, los entornos de programación para lenguajes funcionales modernos suelen ser muy pobres (p.ej. ICHope, CAML Light); además es frecuente que sólo haya implementaciones para máquinas grandes. Quizá pueda encontrarse la explicación en que estos lenguajes están en fase de investigación y, por tanto, no se presta tanta atención al entorno como a la implementación eficiente del mismo lenguaje. La falta de entornos es una importante traba para la difusión de estos lenguajes, en concreto para su uso docente. Sin embargo, la sencillez sintáctica y semántica de los lenguajes funcionales permite prever que no es muy costoso construir entornos integrados de programación con grandes ayudas al programador.

En la ponencia se describe un entorno de programación para el lenguaje funcional

Hope+ [7, 11]. El entorno se ha concebido para un uso docente, aunque también puede usarse para producción a pequeña escala. A continuación se describen brevemente las características principales del entorno desde el punto de vista del usuario; para más detalles sobre la implementación, el lector debe remitirse a [4, 5, 15].

La estructura de la comunicación es la siguiente. En el apartado 2 se describe brevemente la interfaz de usuario proporcionada. Posteriormente, el apartado 3 expone las facilidades de evaluación del entorno y el capítulo 4, el resto de las facilidades. El apartado quinto compara el entorno con otros existentes. Por último, en el sexto apartado se hace un balance del entorno y se citan las mejoras previstas.

2. INTERFAZ DE USUARIO

El entorno presenta una interfaz de usuario moderna basada en menús de texto. De esta forma, se evita que el usuario deba aprender un nuevo lenguaje -el de mandatos de manejo del entorno- distinto de Hope+ y se evita la comisión de muchos errores de entrada -el usuario no debe escribir lo que desee sino simplemente seleccionarlo-. El formato adoptado de menús es el que se ha hecho prácticamente estándar en productos comerciales: estructura arborescente de menús, menú principal de barra, menús secundarios desplegables, selección de opciones con ayuda del cursor o de iniciales, etc.

En cuanto a la presentación de la pantalla, está dividida de forma que la línea superior contiene el menú principal, la inferior contiene mensajes informativos y el resto de la pantalla es área de trabajo. En la figura 1 puede verse el aspecto de la pantalla cuando el usuario quiere hacer alguna operación de alto nivel sobre una función **map**.

El menú principal agrupa las facilidades del entorno en cuatro categorías; en el apartado 3 se describen las facilidades de evaluación y en el apartado 4, el resto.

3. EVALUACIÓN DE EXPRESIONES FUNCIONALES

Un programa funcional se ejecuta evaluando cualquier expresión funcional que solamente contenga identificadores definidos de función y de constante; como resultado se obtiene otra expresión más sencilla, que se llama el valor de la expresión inicial. De manera abstracta, la evaluación de expresiones funcionales se define mediante una *semántica operacional*, que consta de un conjunto de reglas de reescritura de expresiones y una estrategia de aplicación de aquéllas.

Nuestro entorno basa su interfaz para evaluación en el uso de menús y un editor para que la interfaz de usuario sea homogénea en todo el entorno. El usuario edita primero la expresión y posteriormente selecciona la opción de evaluar. El entorno mantiene una lista de las expresiones editadas por lo que muchas veces no es necesario escribir una expresión completa, sino simplemente seleccionar o modificar una ya

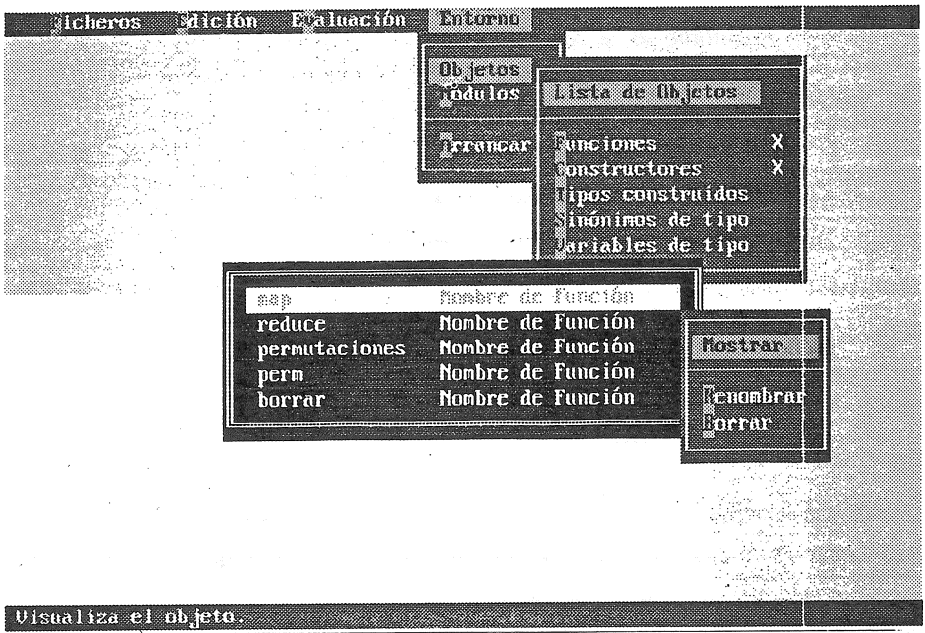

Fig. 1. Aspecto de la interfaz de usuario

existente. Obsérvese que la interacción de evaluación es distinta de la usual en intérpretes, basada en lo que se llama el "bucle de lectura, evaluación e impresión".

El entorno desarrollado presenta dos novedades. Primero, el usuario puede *elegir la estrategia de evaluación* de expresiones funcionales. Segundo, puede *combinar varios modos de evaluación* durante la misma. Veamos ambas facilidades más detalladamente.

(a) Elección de la estrategia de evaluación

Normalmente un entorno sólo permite evaluar expresiones funcionales siguiendo una estrategia prefijada por el constructor del entorno. Aunque potencialmente hay muchas estrategias de evaluación, nuestro entorno incorpora las dos más extendidas -evaluación impaciente y evaluación perezosa-. La disponibilidad de ambas estrategias permite usar la más conveniente para cada caso. Por supuesto, presenta una gran ventaja docente: los alumnos pueden experimentar y comprender mejor ambas estrategias.

(b) Combinación de varios modos de evaluación

Una vez elegida una estrategia de evaluación, el usuario puede evaluar expresiones con la ayuda del intérprete. Sin embargo, es útil disponer de herramientas que den más información que el mero valor final. En nuestro caso el usuario puede conocer expresiones intermedias que se producen durante la evaluación; estas expresiones dependen de los sucesivos modos de evaluación elegidos.

Un *modo de evaluación* es una opción que indica cuántos pasos de reescritura se van a realizar en la evaluación de una expresión, es decir, controla el progreso de la

evaluación. El entorno incluye tres modos: evaluación normal, paso a paso o avanzando hasta un punto de ruptura. El primero produce la evaluación usual en un solo paso, el segundo provoca una sola reescritura y el tercero produce la evaluación de la expresión hasta llegar a un punto de ruptura. Por último, también puede abandonarse una evaluación que, mediante las opciones segunda o tercera, está en un estado intermedio.

Una *función de punto de ruptura* es una función que provoca la interrupción de la evaluación de una expresión cuando el siguiente paso de reescritura es su aplicación a alguna subexpresión. Las funciones de punto de ruptura se seleccionan y liberan, pudiendo haber varias simultáneamente. El concepto de punto de ruptura se ha tomado de depuradores de programas imperativos y de Lisp, aunque con un fin distinto: aquí se centra la atención en la expresión que hay al ir a aplicar ciertas funciones, mientras que en los depuradores se quiere examinar el valor de variables en dicho momento.

La facilidad de evaluación paso a paso es muy útil cuando se está aprendiendo la programación funcional, sobre todo para entender la recursión y la diferencia entre las dos estrategias de evaluación. La evaluación con puntos de ruptura también es útil para comprender la recursión, así como medio de depuración.

Como facilidad adicional, una evaluación paso a paso ya concluida puede almacenarse en un fichero para su edición y examen posterior.

Veamos con un ejemplo sencillo el uso de las distintas estrategias y modos de evaluación. Supongamos la función **factorial** definida de la siguiente forma:

```
dec factorial : num -> num ;
--- factorial (n) <= if n=0 then 1 else n*factorial(n-1) ;
```

El marcado de **factorial** como función de punto de ruptura permite conocer mejor el proceso de evaluación. Eligiendo la estrategia impaciente, la evaluación de punto en punto de ruptura de **factorial(3)** produce las expresiones:

```
factorial(3)
↓
3 * factorial(2)
↓
3 * (2 * factorial(1))
↓
3 * (2 * (1 * factorial(0)))
↓
6 : num
```

La elección de una estrategia perezosa produce prácticamente las mismas expresiones salvo que la aplicación de la función se realiza sin haber evaluado previamente sus argumentos:

factorial(3)
↓
3 * factorial(3-1)
↓
3 * (2 * factorial(2-1))
↓
3 * (2 * (1 * factorial(1-1)))
↓
6 : num

Para ver otras combinaciones de modos de evaluación, veamos un último ejemplo en el que primero se avanza hasta un punto de ruptura, después se avanza paso a paso hasta la siguiente aplicación de **factorial** y ahí se termina en un solo paso. De nuevo se elige evaluación perezosa para ver mejor su comportamiento.

factorial(3)
↓
3 * factorial(3-1)
↓
3 * (if 3-1=0 then 1 else (3-1) * factorial((3-1)-1))
↓
3 * (if 2=0 then 1 else 2 * factorial(2-1))
↓
3 * (if false then 1 else 2 * factorial(2-1))
↓
3 * (2 * factorial(2-1))
↓
6 : num

4. OTRAS FACILIDADES

El entorno incluye otras facilidades imprescindibles para un entorno integrado y amistoso. Las facilidades se agrupan en tres categorías distintas:

(a) Interacción con el exterior

El entorno tiene las opciones usuales de interacción con el sistema operativo: cambiar de directorio activo, consultar el directorio activo, cargar el programa contenido en un fichero, almacenar un programa en un fichero y terminar su ejecución. Para dar mayor flexibilidad, el usuario dispone de un editor de ficheros independiente del editor de programas funcionales que puede usarse para editar texto: p.ej. una evaluación almacenada.

(b) Edición de programas Hope+

El usuario puede editar partes de un programa funcional. Para permitir mayor flexibilidad, hay dos modalidades de edición. Por una parte puede editarse el programa completo, pero también puede editarse independientemente una declaración (variable de tipo, tipo de datos nuevo o renombrado, función o uso de un módulo). El editor es de pantalla, al igual que el editor de ficheros y el de expresiones funcionales. Al terminar una sesión de edición, el programa escrito se analiza y sólo se permite la salida del editor si aquél es correcto sintácticamente o si quiere abandonarse lo editado.

(c) Gestión del entorno

Un último conjunto de facilidades necesarias son aquéllas que permiten al usuario manejar fácilmente las declaraciones incluidas. Así, puede consultarse la relación de identificadores del programa por categorías sintácticas (variables de tipo, tipos nuevos, tipos renombrados, constructores y funciones) y, tras seleccionar uno, consultarse su declaración, cambiar de identificador o borrar la declaración. También puede realizarse la misma gestión (excepto el borrado) sobre módulos cargados en el entorno. Por último, es posible borrar todas las declaraciones realizadas hasta el momento.

5. TRABAJOS RELACIONADOS

Existe una amplia oferta de entornos muy buenos dedicados al lenguaje funcional más extendido, Lisp. Nuestro objetivo actual no es superarles, sino emular sus prestaciones, aprovechándose de las ventajas que dan los lenguajes funcionales modernos, p.ej. una semántica operacional clara. El lector interesado puede consultar la relación de artículos sobre entornos de Lisp contenidos en [1, 3].

Actualmente conocemos dos entornos dedicados a Hope. El primero, llamado ICHope, es un pequeño entorno para ordenadores personales, distribuido como suplemento de un número de la revista *Byte* [2] dedicado a la programación declarativa. El entorno está basado en el bucle de lectura, evaluación e impresión y, por tanto, incluye un lenguaje de mandatos. Aunque es fácil de utilizar, y por tanto adecuado para el propósito divulgador con el que se distribuyó, tiene graves inconvenientes para desarrollos grandes.

La única facilidad de edición es la que proporcione la línea de mandatos del sistema operativo; por tanto puede ser casi inmanejable. También presenta algunos problemas el analizador sintáctico y el sistema de recogida de basura.

En [10] se comenta otro entorno, que contiene facilidades de manejo de ficheros, edición y evaluación. Pueden editarse programas en modo de texto o dirigido por sintaxis, admite declarar operadores disfijos e incluye un sistema de transformación de programas con Hope como metalenguaje y lenguaje objetivo. El artículo menciona ciertos instrumentos de análisis de programas y de rastreo de aplicación de funciones, pero no deja claro si están incluidos en el entorno. El entorno está implementado en el mismo lenguaje Hope, lo que limita su eficiencia y excluye su uso en ordenadores personales; sin embargo, demuestra que la programación funcional es un instrumento válido para desarrollar programas no triviales. Sólo contempla la evaluación perezosa para realizar entrada/salida.

Sólo conocemos otro entorno de lenguajes funcionales modernos para ordenador personal. Se trata de una implementación de un subconjunto, llamado CAML Light [9], del lenguaje CAML. Este entorno es aun peor que ICHope para el usuario. No se distribuye con un manual de usuario claro y tiene integradas muy pocas facilidades de evaluación. Como contrapartida, ofrece una amplia biblioteca de módulos y la posibilidad de trabajar con el intérprete o con un compilador, en cuyo caso crea ficheros ejecutables.

6. VALORACIÓN Y TRABAJOS FUTUROS

Dado que los entornos de programación no cubren muchas de las etapas del ciclo de vida, no son tan complejos como otros entornos de desarrollo de software [6]. Dentro de esta relativa sencillez, podemos analizar distintos aspectos del mismo.

Un aspecto importante de un entorno es la *integración* de las distintas herramientas dentro del entorno. Un entorno de desarrollo de software está integrado si las distintas herramientas que incorpora están integradas en presentación, datos, control y proceso [14]. En líneas generales podemos decir que estas condiciones se cumplen. Así, la integración de presentación se asegura con un uso homogéneo de una interfaz basada en menús y editores. La integración de datos se garantiza con una única representación de los programas en memoria; para garantizar una evaluación eficiente, los programas se guardan en un código intermedio en forma de grafos [4, 7, 12]. La integración de control se garantiza con una programación modular de las facilidades y herramientas del entorno. La integración de control se ha intentado conseguir mediante un buen diseño de los menús, que haga fácil el uso de las distintas herramientas del entorno por el programador.

Otro aspecto importante de un entorno es su *eficiencia*, tanto de espacio de almacenamiento como de tiempos de respuesta. En nuestro caso, éstos son relativamente largos durante los procesos de traducción e interpretación.

Hay algunas mejoras que pueden hacerse al entorno. En primer lugar, pueden eliminarse algunas fuentes identificadas de ineficiencia. En segundo lugar, la interfaz puede mejorarse mediante la definición de teclas de función y el impedimento de elegir opciones de menú que no tienen sentido en un momento dado (p.ej. almacenar si no se ha editado nada). Tercero, la edición puede mejorarse haciéndose parcialmente dirigida por sintaxis; esto permitiría hacer más fácilmente comprobaciones en tiempo de edición. Por último, hay que experimentar para ver si el estilo de interacción y el diseño de menús (la integración de proceso antes mencionada) es adecuado. Esto es especialmente importante para hacer pruebas rápidas durante la depuración.

REFERENCIAS

[1] Sección especial sobre Lisp, *Communications of the ACM*, vol. 34, nº. 9 (septiembre 1991).

[2] R. Bailey, "A Hope tutorial", *Byte*, agosto 1985, págs. 235-258.

[3] D.R. Barstow, H.E. Shrobe y E. Sandewall, *Interactive Programming Environments*, McGraw-Hill, 1984.

[4] M. Belmonte Artero, "Intérprete de Hope", proyecto fin de carrera, Facultad de Informática, Universidad Politécnica de Madrid, junio 1991.

[5] J.A. Castillo Sánchez, "Un entorno de programación para el lenguaje funcional Hope", proyecto fin de carrera, Facultad de Informática, Universidad Politécnica de Madrid, junio 1992.

[6] S.A. Dart, R.J. Ellison, P.H. Feiler, y A.N. Habermann, "Software development environments", *Computer*, vol. 20, nº. 11 (noviembre 1987), págs. 18-28.

[7] A.J. Field y P.E. Harrison, *Functional Programming*, Addison-Wesley, 1988.

[8] P. Hudak, "Conception, evolution and application of functional programming languages", *ACM Computing Surveys*, vol. 21, n.º 3 (septiembre 1989), págs. 354-411.

[9] M. Mauny, "Functional programming using CAML Light", informe técnico, INRIA, mayo 1991.

[10] I. Moor, "Realistic functional programming", en *Functional Programming: Languages, Tools and Arquitectures*, coordinado por S. Eisenbach, Ellis Horwood, 1987, págs. 94-108.

[11] N. Perry, "Hope$^+$", informe técnico IC/FPR/LANG/2.5.1/7, Dept. of Computing, Imperial College, University of London, octubre 1989.

[12] S.L. Peyton Jones, *The Implementation of Functional Programming Languages*, Prentice-Hall, 1987.

[13] Á. Sánchez Calle y J.Á. Velázquez Iturbide, "Fun, rigour and pragmatism in functional programming", *SIGCSE Bulletin*, vol. 23, n.º 3 (septiembre 1991), págs. 11-16.

[14] I. Thomas y B.A. Nejmeh, "Definitions of tool integration for environments", *IEEE*

Software, vol. 9, nº. 2 (marzo 1992), págs. 29-35.

[15] J.Á. Velázquez Iturbide, M. Belmonte Artero y J.A. Castillo Sánchez, "Arquitectura de un entorno integrado de programación funcional", *Primer Congreso Nacional de Programación Declarativa (PRODE'92)*, Madrid, 28-30 septiembre 1992, póster aceptado.