La máquina virtual MiniMe: una herramienta visual para facilitar el aprendizaje de la generación de código objeto

Juan José Tamagnini
Facultad de Matemática Aplicada
Universidad Católica de Santiago del Estero
jtamagnini@ucse.edu.ar

Salvador Valerio Cavadini
Facultad de Matemática Aplicada
Universidad Católica de Santiago del Estero
scavadini@ucse.edu.ar

Pablo Luis Berdaguer
Facultad de Matemática Aplicada
Universidad Católica de Santiago del Estero
pberdaguer@ucse.edu.ar

Resumen

MiniMe es una herramienta visual diseñada para ayudar a los estudiantes de cursos de construcción de compiladores a comprender tópicos relacionados con la fase de generación de código objeto, y a facilitarles su desarrollo. En un entorno gráfico e interactivo, la máquina virtual emula hardware sencillo y ejecuta programas codificados en un lenguaje simple y reducido que puede configurarse para emular diferentes máquinas. En cada momento, *MiniMe* visualiza el estado de distintos componentes y, junto a la instrucción que se ejecuta, resalta la sentencia del programa de alto nivel que la originó. Fue utilizada en un curso de construcción de compiladores, con gran aceptación por parte de los alumnos, quienes lograron familiarizarse rápidamente con la herramienta y su lenguaje.

Palabras clave

Máquina virtual, herramienta visual, construcción de compiladores, informática educativa.

Introducción

Actualmente, el ámbito en el cual se aplican las técnicas de traducción se ha extendido notablemente, transcendiendo las fronteras que probablemente imaginaron Grace Murray Hopper y John Backus hace ya más de medio siglo, cuando prácticamente sólo se procuraba desarrollar compiladores que convirtieran a lenguaje nativo ciertas especificaciones de bajo nivel bastante similares al código binario. Los avances que se producen en el ámbito de la informática, con interfaces más amigables y entornos más poderosos, se logran incrementando la brecha entre el hardware, la arquitectura, el lenguaje máquina, y lo que el usuario realmente percibe. Y para salvar esa distancia resulta imprescindible contar con traductores que realicen la necesaria traslación. En otros términos, se requieren complejos sistemas que conviertan especificaciones abstractas en otras especificaciones –posiblemente ejecutables– aplicando transformaciones que preserven el significado.

Como docentes de una asignatura que trata sobre la construcción de traductores, se nos presentan serios dilemas ante semejantes innovaciones, no sólo con respecto a la selección de los contenidos, sino también en lo atinente a nuestras prácticas, ya que pretendemos que los alumnos desplieguen sus potencialidades y capacidades tanto cognitivas como de aprendizaje. Consideramos que el desarrollo de un proyecto ingenieril favorece la cristalización de las teorías en la práctica real, y en consecuencia requerimos a nuestros estudiantes la construcción de un compilador de mediana complejidad. De este modo, esperamos que consigan no sólo una sólida comprensión teórica, sino que, en el marco de esa experiencia práctica, comprendan cómo operan los compiladores e intérpretes y adquieran habilidades para desarrollarlos.

Uno de los problemas de los alumnos, al enfrentarse al desafío de desarrollar un traductor, es la falta de experiencia y conocimiento del lenguaje objeto –generalmente código ensamblador– al que se pretenden traducir las especificaciones de alto nivel. La carencia o escasa disponibilidad de determinadas plataformas constituye otro obstáculo casi imposible de eludir y limita las actividades que los docentes pueden proponer a los estudiantes, quienes, además, se ven obligados a restarle precioso tiempo a la comprensión de los procesos y técnicas involucradas en la traducción, para dedicárselo al aprendizaje tanto del lenguaje como de la máquina objeto, la cual seguramente posee un complejo set de instrucciones, múltiples modos de direccionamiento y variada cantidad de registros. A esto debe añadirse que, en general, no se cuenta con herramientas que visualicen la relación entre el programa fuente y el código objeto, y la interacción entre éste y el microprocesador que lo ejecuta, lo que dificulta la depuración del código generado por los traductores que construyen los estudiantes.

Sabemos que la adecuada incorporación de los avances tecnológicos en los recursos didácticos posibilita la adquisición más completa y rápida de conocimientos. El uso de la informática, o más específicamente, de programas con algunas características propias de las herramientas multimediales, podría resultar de gran ayuda. Muchos docentes e investigadores están firmemente convencidos de que, para lograr la comprensión íntegra de conceptos y algoritmos de cierta complejidad y elevados niveles de abstracción, los estudiantes deben *verlos* en operación. Sostienen en sus estudios que la mente humana está fuertemente orientada a la visión y, en consecuencia, adquiere y asimila información a una tasa significativamente más alta si se le presentan imágenes y relaciones gráficas en lugar de hileras de texto (Lovato et al, 1995). Ese argumento brindó el sustento necesario para que los desarrolladores de sistemas de animación de algoritmos (Brown, 1988) hipoteticen que las animaciones pueden transformarse en ayudas instruccionales valiosas y para que muchos

docentes ya hayan incorporado este tipo de herramientas en sus cátedras (Blythe et al, 1994; Khuri et al, 1994; Rodger, 1996; Bilska et al, 1997; Khuri et al, 1998; White et al, 1998; Gramond et al, 1999; Hung et al, 2000).

Motivados por estos resultados, y dada la escasez de herramientas apropiadas para nuestras necesidades específicas, emprendimos un proyecto a través del cual nos proponemos diseñar, desarrollar y evaluar un conjunto de aplicaciones que nos ayuden en nuestras actividades de enseñanza y le faciliten al estudiante el aprendizaje de conceptos, procedimientos y operaciones fundamentales de los algoritmos utilizados por los procesadores de lenguajes. Uno de los primeros frutos de este proyecto es *MiniMe*, un prototipo de una máquina virtual visual que interpreta programas codificados en MiniAssembler, un lenguaje de bajo nivel con las funciones básicas para el manejo de memoria, registros de procesador, pila y entrada/salida. Si bien es importante que el diseñador de compiladores conozca en profundidad tanto la arquitectura subyacente como el lenguaje de la máquina objeto, para introducir los conceptos básicos de la generación de código consideramos que es suficiente trabajar con simples e hipotéticas computadoras, que en un entorno visual amigable muestren el estado de la memoria, la pila y los registros del procesador, permitiendo así el estudio minucioso –instrucción por instrucción– de la ejecución del código, a la vez que se facilita en gran medida la detección y depuración de errores en el código generado.

En las siguientes secciones presentamos la estructura de *MiniMe*, los detalles más relevantes sobre sus componentes, funcionalidad, interfaz y el lenguaje que ejecuta. Posteriormente describimos una experiencia llevada a cabo con ella en un curso de construcción de compiladores. Por último, incorporamos una breve mención a trabajos similares y las conclusiones.

La máquina virtual visual MiniMe

El "hardware" que emula *MiniMe* es el siguiente:

- Memoria de programa. En ella se almacenan las instrucciones del programa MiniAssembler. Su capacidad está limitada sólo por la memoria RAM libre de la máquina real (PC) que la ejecuta.
- Memoria de datos. En esta memoria se guardan los datos sobre los que opera el programa MiniAssembler.
- Pila. Separada de la memoria de datos.
- Registros:
  - *PC* (Program Counter): contiene la dirección de memoria de programa de la instrucción que se está ejecutando.
  - *SP* (Stack Pointer): apunta al tope de la pila.
  - *X, Y, Z*: se usan para almacenar los parámetros de algunas instrucciones de MiniAssembler.
  - *Comparación*: contiene el resultado de la última instrucción de comparación.
- Registros auxiliares. Son de propósito general.
- Procesador. Interpreta programas codificados en MiniAssembler.

- Pantalla. Una matriz de 20 por 20 caracteres para mostrar información de salida de los procesos.

Aprovechando las facilidades brindadas por los lenguajes y entornos visuales actuales, que agilizan el desarrollo de interfaces amigables con las cuales es posible representar gráficamente algunas abstracciones y modelos, durante el proceso de interpretación de las instrucciones de MiniAssembler se pueden apreciar múltiples aspectos de la ejecución. En cada paso, *MiniMe* visualiza:

- El programa MiniAssembler cargado en la memoria de programa y la instrucción que se está ejecutando;
- El código fuente desde el cual se generó el programa en MiniAssembler, destacando la sentencia que originó a la instrucción en ejecución;
- El valor de cada una de las posiciones de memoria utilizadas;
- El estado de todos los registros;
- La configuración de la pila;
- Estado de la pantalla.

En la Figura 1 puede observarse un detalle de las partes de la interfaz de *MiniMe*.

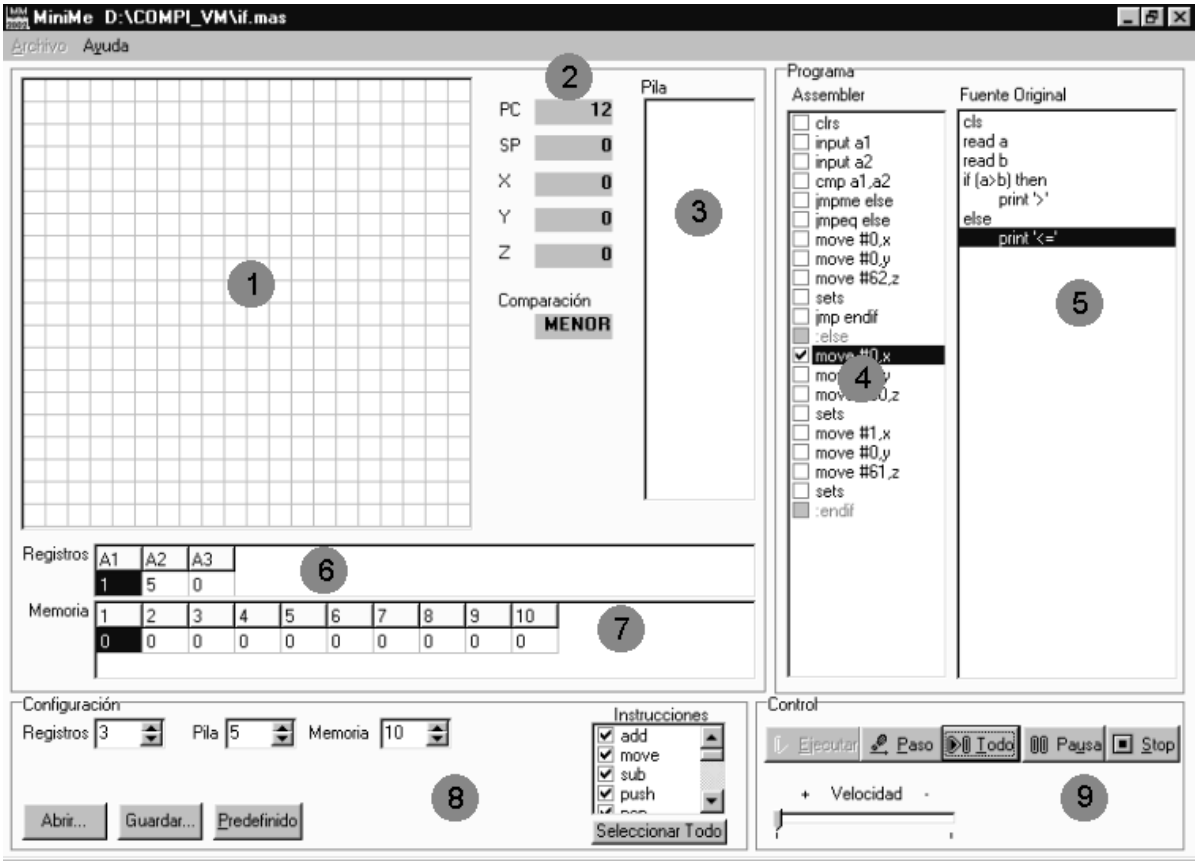

Figura 1: Detalle de las partes que componen la interfaz de MiniMe.

1. Pantalla de salida.
2. Registros principales.
3. Pila de datos.

4. Código MiniAssembler cargado en la memoria de programa de la máquina virtual.
5. Programa fuente desde donde se obtuvo el código en MiniAssembler.
6. Registros auxiliares.
7. Memoria de datos.
8. Controles de configuración de *MiniMe*.
9. Tablero de control de la ejecución del programa.

Con la opción de menú *Abrir* se selecciona el archivo que contiene el código MiniAssembler que se desea ejecutar. Una vez que *MiniMe* lo carga en la memoria de programa, la interfaz pone a disposición del usuario un conjunto de opciones y controles que permiten ejecutar las instrucciones paso a paso, resaltar la que se está ejecutando, introducir puntos de ruptura (*breakpoints*) en el código e inclusive regular la velocidad, mediante lo cual, además de facilitar la visualización del trabajo realizado por *MiniMe*, se pretende incrementar la efectividad pedagógica de la herramienta (Saraiya et al, 2004).

Dado que la aplicación fue desarrollada para estudiantes de cursos donde se enseñan las técnicas de traducción de lenguajes formales, también es posible visualizar el código de alto nivel desde el cual se obtuvo el programa en MiniAssembler. Como *MiniMe* depende –inevitablemente– de las instrucciones que ejecuta, pero no del lenguaje fuente desde el cual aquellas provienen, durante la ejecución es factible resaltar sincronizadamente tanto la instrucción en MiniAssembler como la sentencia correspondiente del programa fuente (Figura 2). La vinculación entre las sentencias de alto nivel y las instrucciones de bajo nivel se logra mediante dos directivas especiales disponibles en MiniAssembler: una (@P) señala el nombre y la ubicación del archivo desde el cual se generaron las instrucciones MiniAssembler, mientras que otra (@L) indica la línea del programa fuente que originó la instrucción. Este simple mecanismo, al posibilitar que la herramienta soporte programas fuente codificados en cualquier lenguaje de alto nivel para el cual se desarrolle un traductor a MiniAssembler, facilita la comprensión por parte del alumno de la correspondencia entre las sentencias de alto nivel y las instrucciones ensamblador producidas por el compilador encargado de la traducción, una de las principales metas del diseño de *MiniMe*.

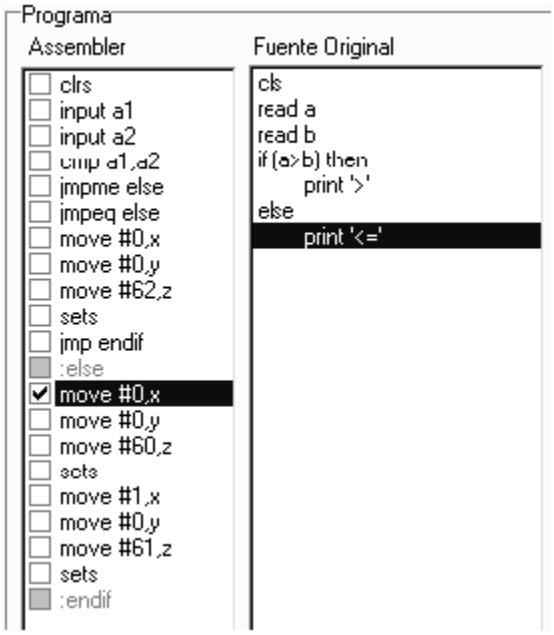

Figura 2: Correspondencia entre las sentencias de alto nivel y las instrucciones MiniAssembler.

MiniAssembler, el lenguaje de programación de MiniMe

La arquitectura de cualquier máquina abstracta está determinada por la potencia del lenguaje en el que se codifican los programas que debe ejecutar. Por lo tanto, una de las primeras y cruciales decisiones de diseño está relacionada con el lenguaje en el cual deberán codificarse los programas que serán ejecutados por *MiniMe*. Optar por algún lenguaje nativo o un ensamblador real implicaría caer en una fuerte e indeseada dependencia con el hardware. Además, exigiría que los docentes y los alumnos aprendieran a codificar programas en un lenguaje de bajo nivel predeterminado y limitaría el espectro de ejercicios que se podrían diseñar, lo cual seguramente atentaría contra el uso de la herramienta. Esas razones motivaron que MiniAssembler sea independiente del hardware y cuente sólo con 16 instrucciones. Esta capa de abstracción permite que los estudiantes inviertan la menor cantidad de tiempo posible en el aprendizaje de las particularidades del lenguaje y concentren su esfuerzo en el estudio del funcionamiento de su traductor y en la interacción del código generado con el "hardware" de la máquina virtual, otra de las metas del diseño de *MiniMe*. La Tabla 1 enumera las instrucciones de MiniAssembler junto con una breve descripción semántica. Aunque reducido, el set incluye lo necesario para experimentar con múltiples variantes de generación de código.

Instrucción	Funcionalidad	Ejemplo
ADD	Suma el [valor|registro] al registro destino.	`add #2, x`
CLRS	Limpia la pantalla de salida.	`clrs`
CMP	Compara el [valor|registro] con un registro y almacena el resultado –"MENOR", "IGUAL" o "MAYOR"– en el registro Comparación.	`cmp #2, x`
DIV	Divide el registro destino por el [valor|registro]; el resultado (entero) se almacena en el registro destino.	`div x,y`
INPUT	Solicita un valor numérico y lo carga en un registro.	`input a1`
JMP	Transfiere incondicionalmente el control a la dirección de memoria representada por la etiqueta.	`jmp loop1`

Instrucción	Funcionalidad	Ejemplo
JMPEQ	Transfiere el control a la dirección de memoria representada por la etiqueta si el contenido del registro Comparación es "IGUAL".	`jmpeq afuera`
JMPME	Transfiere el control a la dirección de memoria representada por la etiqueta si el contenido del registro Comparación es "MENOR".	`jmpme loop1`
LOAD	Carga el valor de una posición de memoria en un registro.	`load $23,a3`
MOVE	Mueve el [valor|registro] al registro destino.	`move #12,a2`
MULT	Multiplica el [valor|registro] por el registro destino; el resultado se almacena en el registro destino.	`mult a1,x`
POP	Extrae un valor desde la pila y lo asigna al registro destino.	`pop a1`
PUSH	Introduce en la pila el [valor|registro].	`push #2`
SETS	Muestra en la pantalla de salida, en la fila y columna referenciadas por los registros X e Y respectivamente, el caracter cuyo código ASCII está almacenado en el registro Z.	`sets`
STORE	Almacena el valor de un registro en una posición de memoria.	`store a3,$4`
SUB	Resta el [valor|registro] al registro destino.	`sub a1,sp`

Tabla 1: Detalle del set de instrucciones de MiniAssembler.

Cuando se pretenden estudiar los procesos de generación de código y optimización final para distintas máquinas objeto, conviene poner a disposición del usuario opciones que le permitan configurar la cantidad de registros auxiliares, el tamaño de la memoria de datos y la profundidad máxima de la pila. Sin embargo, lo anterior, aunque necesario, resulta insuficiente, y por eso en *MiniMe* también es posible seleccionar distintos "subsets" de instrucciones del lenguaje ensamblador, con lo cual, en definitiva, se logra que una única máquina virtual pueda emular distintas máquinas.

Experiencias con MiniMe

La primera experiencia con *MiniMe* en un curso universitario se realizó durante el ciclo lectivo 2003 con alumnos de la asignatura *Compiladores e Intérpretes* de la carrera Ingeniería en Computación de la Universidad Católica de Santiago del Estero.[1] Los estudiantes construyeron, organizados en grupos, el front-end y el módulo generador de código intermedio –cuádruplas– para un lenguaje de programación pequeño.[2] Dos grupos que pretendían acceder a la promoción de la asignatura debieron extender su compilador adicionándole una fase generadora de código objeto y otra de optimización. Aprovechando las posibilidades de configuración de *MiniMe*, a uno se le requirió la generación de código para una máquina de pila (*stack machine* o *zero address*) mientras que el otro lo hizo para una máquina con registros y memoria.

En ambos casos, los alumnos necesitaron sólo un par de días para familiarizarse con la herramienta y su lenguaje. Los ejemplos listos para ser usados que se incluyen con la aplicación favorecieron este proceso. A diferencia de lo acontecido en años anteriores, esta vez las consultas de los estudiantes se centraron principalmente en tópicos relacionados con las estrategias de generación de código, y desplazaron a aquellas vinculadas a cuestiones concernientes con la arquitectura y el lenguaje de la máquina objeto, tan comunes y recurrentes en los ciclos lectivos precedentes. Al concluir sus proyectos, destacaron que

[1] El sitio web de la asignatura es www.ucse.edu.ar/fma/compiladores

[2] La sintaxis del lenguaje es similar a la de Pascal. Es monoprocedural y permite sentencias de asignación, condicionales (IF-THEN-ELSE) y de ciclo (WHILE).

MiniMe les había facilitado las tareas, ya que pudieron ejercitarse en el diseño y desarrollo de generadores de código sin necesidad de trabajar con lenguajes y plataformas demasiado complejas.

Los alumnos también aportaron valiosísimas sugerencias para mejorar la herramienta. Remarcaron que poder avanzar y retroceder durante la ejecución de los programas MiniAssembler sería de gran ayuda. Esto concuerda con lo que sostienen quienes se dedican a la investigación de herramientas educativas multimediales, cuando recomiendan dotarlas con esa capacidad (Badre et al, 1993; Boroni et al, 1996; Naps et al, 2003). Manifestaron además la necesidad de contar con *breakpoints* más sofisticados –e.g. activados por condición–, situados en el código de alto nivel o activados al momento de producirse una lectura/escritura en posiciones determinadas de la memoria de datos. Asimismo, expresaron que les hubiera resultado de utilidad, al experimentar con ciertos algoritmos de optimización de código, contar con información sobre la cantidad de instrucciones ejecutadas.

Mientras evaluamos las modificaciones que requiere el diseño de *MiniMe* para que sea posible implementar las nuevas funcionalidades señaladas, y con el propósito de incrementar aún más la flexibilidad de configuración de la máquina virtual, estamos trabajando para lograr que *MiniMe* reconozca lenguajes ensambladores definidos por los usuarios, quienes de esta manera podrán especificar sus propios sets de instrucciones –i.e. su lenguaje objeto particular–. Por último, tenemos planificado portar la herramienta al entorno Linux –la actual versión se ejecuta en plataformas win32–.

En cuanto a su aplicación en el ámbito académico, analizamos la posibilidad de utilizar *MiniMe* en otras asignaturas –e.g. en las relacionadas con la arquitectura de las computadoras y la programación en lenguaje ensamblador–, lo cual permitiría efectuar distintos estudios sobre la utilidad de la máquina virtual en ambientes concretos de aprendizaje y continuar perfeccionado su diseño a partir de los comentarios y opiniones de los docentes y alumnos.

Trabajos Relacionados

En varios cursos relacionados con los lenguajes formales y la construcción de sus traductores los docentes han incorporado en sus prácticas herramientas de software que le permiten a los alumnos interactuar con modelos y objetos abstractos. Esperan que estas aplicaciones motiven a los estudiantes y les faciliten, al brindarles retroalimentación visual inmediata, el aprendizaje de conceptos y procedimientos complejos. En este sentido, merecen mencionarse toolkits como *JFLAP*[3] y el que se desarrolla en el marco del proyecto *SOFTWARE PARA LA ENSEÑANZA DE PARSING* (*SEPa!*)[4], los cuales están orientados principalmente a la enseñanza de las teorías que sustentan el diseño y desarrollo de fases correspondientes al front-end de los traductores.

A través de Internet es posible acceder a otras herramientas con prestaciones similares a las de *MiniMe*, entre las que podemos destacar a *SPIM*,[5] *The Visible Virtual Machine*[6] y la presentada en "A simulation of Knuth's mix machine as a teaching tool" (Nakanelua et al, 2004). Si bien estas máquinas virtuales fueron construidas con el objeto de apoyar los procesos de enseñanza y aprendizaje, tienen la desventaja, con respecto a *MiniMe*, de

[3] Información sobre JFLAP (**J**ava **F**ormal **L**anguages and **A**utomata **P**ackage) y su utilización académica puede encontrarse en: www.cs.duke.edu/~rodger/tools.
[4] El sitio oficial del proyecto SEPa! es: www.ucse.edu.ar/fma/sepa.
[5] En http://www.cs.wisc.edu/~larus/spim.html se encuentra información sobre este simulador para el procesador MIPS32.
[6] El sitio de esta máquina virtual visible es: http://www.cba.uri.edu/faculty/vvm/.

presentar arquitecturas rígidas –no configurables– y lenguajes complejos o demasiado restringidos como para resultar más útiles que *MiniMe* en un curso de construcción de compiladores. Estas limitaciones se deben, principalmente, a que fueron desarrolladas para asignaturas donde los alumnos toman contacto por primera vez con los detalles del funcionamiento a bajo nivel de las computadoras. En cambio, *MiniMe* permite emular diferentes máquinas y relacionar las instrucciones de los programas MiniAssembler con las sentencias de un programa en alto nivel precisamente porque está concebida para auxiliar a estudiantes que deben aprender a generar código objeto para distintas arquitecturas.

Conclusiones

En este artículo hemos presentado *MiniMe*, una máquina virtual visual especialmente diseñada para asistir a los alumnos en el proceso de compresión de los conceptos y técnicas utilizadas para la generación automática de código objeto y optimización final. *MiniMe* provee un entorno flexible con interfaces gráficas amigables para que los estudiantes puedan aprovecharla tanto en clase como en sus hogares. Las amplias posibilidades de configuración, sumado a la posibilidad de relacionar visualmente el código que ejecuta con el código fuente original, son aspectos que hacen única a esta máquina virtual. Las primeras experiencias con *MiniMe* en un curso de construcción de compiladores resultaron alentadoras, pues revelaron su potencial como herramienta facilitadora del aprendizaje de las técnicas utilizadas en las últimas fases de los traductores. Estos ensayos también permitieron detectar varios aspectos susceptibles de mejoras.

Los docentes interesados en *MiniMe* pueden descargarla gratuitamente desde www.ucse.edu.ar/fma/sepa/minime.

Referencias

[1] Badre, Albert; Stasko, John; Lewis, Clayton. (1993). "Do Algorithm Animations Assist Learning? An Empirical Study and Analysis". En: Proceedings of INTERCHI '93 Conference on Human Factors in Computing Systems. Amsterdam. pp. 61-66. ACM.

[2] Bilska, Anna; Leider, Kenneth; Procopiuc, Magdalena y otros. (1997). "A Collection of Tools for Making Automata Theory and Formal Languages Come Alive". En: Proceedings of 28º Technical Symposium on Computer Science Education SIGCSE '97. ACM. San José. EE.UU. pp. 15-19.

[3] Blythe, Stephen A.; James, Michael C.; Rodger, Susan H. (1994). "LLparse and LRparse: Visual and Interactive Tools for Parsing". En: Proceedings of 25º Technical Symposium on Computer Science Education SIGCSE '94. ACM. Phoenix. EE.UU. pp. 208-212.

[4] Boroni, Christopher; Eneboe, Torlief; Goosey, Frances y otros. (1996). "Dancing with Dynalab. Endearing the Science of Computing to Students." En: Proceedings of 27º Technical Symposium on Computer Science Education SIGCSE '96. ACM. Philadelphia. EE.UU. pp. 135-139.

[5] Brown, Marc H. (1988). "Perspectives on algorithm animation". En: Proceedings of ACM SIGCHI '88 Conference on Human Factors in Computing Systems. ACM. Washington. pp. 33-38.

[6] Gramond, Eric; Rodger, Susan H. (1999). “Using JFLAP to interact with theorems in automata theory”. En: Proceedings of the 30º SIGCSE Technical Symposium on Computer Science Education. ACM. New Orleans. EE.UU. pp. 336-340.

[7] Hung, T.; Rodger, Susan. (2000). “Increasing Visualization and Interaction in the Automata Theory Course”. En: Proceedings of the 31º SIGCSE Technical Symposium on Computer Science Education. ACM. Austin. EE.UU. pp. 6-10.

[8] Khuri, Sami; Williams, Jason. (1994). “Understanding the Bottom-up SLR Parser”. En: Proceedings of 25º Technical Symposium on Computer Science Education SIGCSE ‘94. ACM. Phoenix. EE.UU. pp. 339-343.

[9] Khuri, Sami; Sugono, Yanti. (1998). “Animating Parsing Algorithms”. En: Proceedings of 29º Technical Symposium on Computer Science Education SIGCSE ‘98. ACM. Atlanta. EE.UU. pp. 232-236.

[10] Lovato, Mona E.; Kleyn, Michael F. (1995). “Parser Visualizations for Developing Grammars with Yacc”. En: Proceedings of 26º Technical Symposium on Computer Science Education SIGCSE ‘95. ACM. Nashville. EE.UU. pp. 345-349.

[11] Nakanelua, Bobby; Curtsinger, Michael. (2004). "A simulation of Knuth's mix machine as a teaching tool". En: Journal of Computing Sciences in Colleges. Vol. 19. Nº 3. pp. 258-267.

[12] Naps, Thomas L.; Röbling, Guido; y otros. (2003). “Exploring the Role of Visualization and Engagement in Computer Science Education”. Report of the Working Group on “Improving the Educational Impact of Algorithm Visualization”. En: ACM SIGCSE Bulletin. Vol. 35. Nº 2. pp. 131-152.

[13] Rodger, Susan H. (1996). “Integrating animations into courses”. En: Proceedings of ACM SIGCSE/SIGCUE Conference on Integrating Technology in Computer Science Education. ACM. Barcelona. España. pp. 72-74.

[14] Saraiya, Purvi; Shaffer, Clifford A.; McCrickard, D. Scott; North, Chris. (2004). “Effective features of Algorithm Visualizations”. En: Proceedings of 35º Technical Symposium on Computer Science Education SIGCSE ’04. ACM. Norfolk, Virginia. EE.UU. pp. 382-386.

[15] White, Elizabeth L.; Deddens, Laura Denise; Ruby, Jeffrey. (1998). “Software Visualization of LR Parsing and Synthesized Attribute Evaluation”. Technical Report. Department of Computer Science. Universidad George Mason. Washington D. C.