Modernización de la Enseñanza de la Traducción

Manuel E. Bermúdez
University of Florida
Computer and information Science and Engineering
Gainesville, Florida, 32611
manuel@cise.ufl.edu

Resumen

En el currículum de Ciencias de la Computación, el curso de compiladores agoniza. Ha sido relegado a un papel "especializado, opcional" en el currículum de la ACM 2001. Sin embargo, algunos tópicos fundamentales se cubren únicamente en ese curso, tales como análisis sintáctico, traducción automática con aplicaciones fuera del área de compilación, y mecanismos de especificación de lenguajes tales como las expresiones regulares. En este artículo presentamos el esquema de un curso completamente reorganizado y modernizado sobre el tema de traducción por computador. El enfoque se asemeja a una espiral: los temas se discuten en forma repetida y cada vez con mayor profundidad, acompañados por proyectos de implementación que los ilustran. Abandonamos el objetivo tradicional de implementar un compilador de producción, al igual que otros tópicos que son demasiado específicos a la compilación. Discutimos áreas de aplicación, tales como la minería de datos y la ingeniería de software.

Palabras clave: compilador, traducción, currículum de Ciencias de la Computación.

Abstract

In the Computer Science curriculum, the compilers course is dying. It has been relegated to a "specialized, optional" role in the ACM 2001 curriculum. However, some fundamental topics are covered only in that course, such as syntax analysis, computer translation with applications outside compilation, and language specification mechanisms such as regular expressions. In this paper, we present the outline of a completely reorganized and modernized course on computer translation. The approach resembles a spiral: topics are discussed repeatedly and in increasing depth, accompanied by implementation projects that illustrate them. We abandon the traditional goal of implementing a production compiler, as well as other topics that are too compilation-specific. Applications are discussed in areas such as data mining and software engineering.

Keywords: compilers, translation, Computer Science curriculum**.**

1. Introducción.

Hace poco más de una década, en el currículum de Ciencias de la Computación de las mejores universidades del mundo, se consideraba que un curso en construcción de compiladores era fundamental para la formación del estudiante de pre-grado. Era raro encontrar un programa académico en Ciencias de la Computación o Informática que no tuviera un curso obligatorio de construcción de compiladores. Se consideraba motivo de vergüenza que un graduado de una universidad de renombre fuera ignorante del área y las técnicas enseñadas en el curso de compiladores. Esta situación ahora ha cambiado. Durante la última década, el curso de compiladores en la mayoría de los programas de Computación e Informática han estado en declinación.

En este artículo discutimos las razones de esta declinación, y las razones por las que creemos que una reforma es necesaria. Luego presentamos el bosquejo de un curso completamente reorganizado y modernizado para la enseñanza de la traducción por computadora. Nuestro enfoque tiene la intención de rescatar tópicos que todavía consideramos que son fundamentales para la formación del profesional en computación e informática, pero que corren el riesgo de quedar a un lado como efecto secundario de la declinación del curso de compiladores. A la vez, dejamos por fuera tópicos que son demasiado específicos al área de compiladores, y cuya presencia (y énfasis) han contribuido al problema.

El resto de este artículo está organizado de la siguiente manera. En la Sección 2 discutimos las circunstancias de la declinación del curso de compiladores. En la Sección 3 discutimos los problemas que sufren los cursos de compiladores en todas partes hoy día, y justificamos nuestra tesis de que una reforma es necesaria. En la Sección 4 presentamos nuestra versión del

curso moderno de Traducción por Computadora, discutimos su organización en forma de espiral, y las ventajas de nuestro enfoque. En la Sección 5 discutimos aplicaciones de nuestro enfoque, fuera del área de compilación. Finalmente, en la Sección 7, presentamos conclusiones.

2. La Declinación del Curso de Compiladores.

Nadie cuestiona hoy día el hecho de que el curso de compiladores es menos fundamental en el currículum de Ciencias de la Computación, de lo que era hace algunos años. Para esto hay muchas razones.

Primero, tenemos la madurez de la disciplina de Computación e Informática. Las técnicas de construcción de compiladores han evolucionado a un paso mucho menor en años recientes, y en general en Computación se tiende a dar mayor importancia a las tecnologías más recientes. El curso tradicional de compiladores era una excelente oportunidad para que el estudiante se familiarizara con una gran variedad de estructuras de datos: árboles, tablas de símbolos, grafos, etc. Hoy día, esos tópicos se consideran suficientemente cubiertos en cursos básicos como el de Estructuras de Datos. Otra razón para la declinación es la proliferación de lenguajes y paradigmas en años recientes, en reacción a lo cual los expertos en el currículum, programas académicos, y escritores de libros de texto, todos han concentrado sus esfuerzos en el diseño e implementación de lenguajes de programación, en lugar de las técnicas que se cubrían tradicionalmente en el curso de compiladores. Como resultado, hoy día un programa académico de Ciencias dela Computación es mucho más probable que tenga, como requisito obligatorio para el estudiante, un curso de Lenguajes de Programación, que un curso de Compiladores. Otra razón más es que los programas académicos son conscientes de que hoy día son pocos lo graduados de un programa que se ganarán la vida escribiendo o manteniendo compiladores, y las habilidades requeridas para ese tipo de trabajo son altamente especializadas. En forma similar, cada vez menos profesores trabajan en compiladores como su área principal de investigación, y conforme áreas nuevas y dinámicas en Computación han ido emergiendo, tales como redes y asuntos relacionados con la WWW, el número de profesores capacitados y dispuestos a enseñar el curso de compiladores ha disminuido en forma considerable..

Finalmente, está el asunto de las tendencias en la disciplina y en el currículum. Durante discusiones que llevaron a la adopción del Currículum ACM 2001, la profesora Mary Shaw de la Universidad Carnegie-Mellon dijo: --"*Organicemos nuestros cursos alrededor de las ideas en lugar alrededor de los artefactos ... Las escuelas de ingeniería no enseñan diseño de calderas – enseñan la termodinámica. Aún así, dos de nuestros principales cursos de software – construcción de compiladores y sistemas operativos – son dinosaurios del tipo "artfacto de sistema*". [1]. El currículum recomendado por la ACM en 2001 resolvió este problema relegando el curso de compiladores al estatus de material "avanzado/suplemental", es decir, material electivo.

3. Reforma del Curso de Compiladores.

La discusión anterior con respecto a la declinación del curso de compiladores se puede resumir en una frase: los tiempos cambian. De hecho, cuando los tiempos cambian, hay ganadores y perdedores, y a primera instancia parecería haber poco campo en el futuro del programa académico de Computación para el curso de compiladores. Sin embargo, algunos de los problemas que han hecho que el curso de compiladores se haya vuelto menos popular en años recientes, son debidos a la organización misma del material. Procedemos a enumerar algunos de estos problemas.

- Con alta frecuencia, el curso de compiladores es orientado hacia la implementación de un compilador. El objetivo tradicional de hacer del curso un ejercicio en la escritura de compiladores "de verdad" es en parte la razón por la que el curso se considera demasiado especializado, y menos que fundamental en el currículum.
- El proyecto del curso frecuentemente tiene un conflicto con la secuencia de temas en el curso. El proyecto normalmente consiste de implementar, desde cero (o casi cero), un compilador, quizás con la ayuda de herramientas tales como `lex [2]y yacc [3]`. Los estudiantes implementan el compilador conforme progresa el período académico, desarrollando los diversos componentes del compilador, conforme se discuten los correspondientes temas en el curso. Muchas veces los temas en el libro de texto no empatan con los componentes del compilador. Para poder diseñar un componente del compilador, es muchas veces necesario comprender cómo el diseño afectará otros componentes del compilador, los cuales en ese momento no han sido discutidos todavía en el curso. Debido a que el tamaño del compilador es excesivo para un solo estudiante, muchas veces el instructor asigna equipos de estudiantes para que desarrollen uno o más componentes. El resultado de esto es que estudiantes individuales aprenden solamente un componente del compilador, o solamente una parte del lenguaje que se implementa. El estudiante no adquiere la visión global del proceso de traducción, sino quizás hasta el final del curso. Para ese estudiante, el curso es como una novela de misterio: el desenlace ¡y la trama ! no son revelados sino hasta el último minuto. Además, si el compilador se implementa desde cero, el estudiante no experimenta resultados reales sino hasta el final del curso. Si la implementación es solo parcialmente exitosa, o sufre problemas a última hora, el estudiante muchas veces termina el curso sin haber nunca obtenido un compilador funcional. Debido a todo esto, el curso de compiladores es

con frecuencia percibido por los estudiantes como "esotérico, difícil, y frustrante". Todo esto reduce su popularidad.

- El libro de texto clásico sobre Compiladores, por Aho, Sethi y Ullman [4], conocido como el libro del Dragón Rojo, no es pedagógicamente idóneo para un curso de pre-grado. El libro es grande, lo abarca todo, es más apropiado para cursos de pos-grado, y sus propios autores reconocen que es demasiado grande e incómodo para ser utilizado en el curso típico de pre-grado. El material se discute en un enorme grado de profundidad, y muchos instructores caracterizan la utilización del libro como "—leer 10 páginas, y brincarse 30". La segunda edición del libro fue publicada en 1986, y dada la declinación del curso de compiladores, los autores mismos dudan que se vaya a publicar una tercera edición en el futuro.
- La frustración con el libro del Dragón Rojo ha llevado a muchos autores a escribir su propia versión. Sin embargo, se puede decir con certeza que todos los demás libros de compiladores consisten del subconjunto favorito de temas de ese autor, ***de los mismos temas, en la misma secuencia y el mismo orden***, que los temas en el libro del Dragón Rojo. [5,6,7,8,9,10,11,12,13,14,15,16,17, 18,19]. En la siguiente sección detallaremos nuestra propia secuencia de tópicos.
- Aparte de cualquier consideración del libro de texto, el instructor típico del curso de compiladores se ve en la imposibilidad de cubrir todo el material en un solo curso. Las secuencias de dos cursos de compiladores ya se fueron a la extinción, de modo que el problema del exceso de material persiste. Para resolver este problema, los instructores tienden a utilizar diversas técnicas. Enfatizan temas, o bien los des-enfatizan, o bien los dejan por fuera por completo, en una forma desorganizada, muchas veces basado nada más en su propia familiaridad (o carencia de ella) con el tema. Agregando a esto el hecho de que cada vez menos instructores de compiladores son expertos en compilación, el resultado es que las decisiones tomadas muchas veces no son las mejores desde el punto de vista de la pedagogía.

El problema principal, entonces, con los cursos de compiladores, ha sido precisamente eso – el enfoque en la compilación. Los principios subyacentes a la traducción, el reconocimiento sintáctico y procesamiento semántico, ***transcienden la compilación***. Además, la popularidad decreciente del curso de compiladores se debe, en nuestra opinión, a la disparidad entre los tópicos del curso, y los tópicos del proyecto de implementación. Ahora procedemos a describir nuestra solución a estos problemas.

4. El Nuevo Diseño.

Aquí presentamos nuestro nuevo diseño del curso. Está basado en las siguientes premisas:

- El curso se titula Traducción por Computadora con Aplicaciones. Aunque la compilación sigue siendo una de las aplicaciones más comunes (y la más dominante), hemos eliminado un buen número de temas específicos a la compilación, tales como generación de código para un procesador real, y aritmética de punto flotante. También discutiremos la aplicación de la traducción en otras áreas de la Computación, como los procesadores de líneas de comando, y las interfaces de usuario.
- El diseño del curso gira alrededor del proyecto de implementación. El proyecto consiste de mantener y extender un compilador "inicial", en lugar de implementar uno desde cero. El compilador inicial es la implementación de un lenguaje imperativo, parecido al lenguaje C, que *es* ***mínimo***. Los estudiantes extienden y mantienen el compilador, agregando construcciones nuevas conforme progresa el período lectivo.
- En el nuevo diseño, cubrimos únicamente los temas que son fundamentales al proceso de traducción, como el análisis sintáctico, el manejo de símbolos, la revisión de tipos, y la generación de código para una máquina abstracta.
- Específicamente eliminamos el objetivo tradicional de construir un compilador "real".

La Tabla 1 muestra una comparación directa de las dos secuencias de temas: viejo y nuevo. En la secuencia vieja, los tópicos se discuten en el orden en el cual el compilador en sí realiza su trabajo. Este orden resulta ser el orden en el cual un escritor de compiladores con experiencia desarrollaría su sistema, razón por la cual (a propósito) este orden se ha cuestionado pocas veces en el pasado.

En el nuevo enfoque, al estudiante se le entrega un compilador completamente funcional, para un lenguaje mínimo, imperativo, similar al C. Este lenguaje tiene variables de tipo entero y booleano, y la habilidad de declararlas. El lenguaje contiene una instrucción de asignación, una instrucción `while`, una instrucción `if`, y una instrucción `print`. El compilador inicial implementa solamente el operador del menos unario, y los operadores binarios de suma y ≤. Una función intrínseca `read` se utiliza para leer variables.

	Diseño Viejo	Diseño Nuevo
1.	Introducción	Introducción
2.	Descripción del Lenguaje	Descripción del Lenguaje Inicial
3.	Análisis Léxico	Descripción del Compilador Inicial
4.	Análisis Sintáctico	Traducción de Operadores
5.	Análisis Semántico	Traducción de Instrucciones
6.	Generación de Código	Traducción de Tipos de Datos
7.	Optimización de Código	Traducción de Subprogramas
8.	Estructuras de Tiempo de Ejecución	Traducción de Vectores
9.	Emisión de Código Final	Traducción de Estructuras
10.		Aplicaciones

Tabla 1. El Diseño del Curso de Compiladores/Traducción

La implementación del compilador utilizará `yacc` (o bien software similar para los lenguajes Java y C#), con un pre-procesador que traduce desde gramáticas con partes derechas regulares hasta gramáticas libres de contexto apropiadas para esos paquetes de software. La Figura 1 muestra la sintaxis del lenguaje inicial, al que hemos llamado `Tiny`.

```
Tiny       -> 'program' Name ':' Dclns Block '.'
Dclns      -> 'var' (Dcln ';')+
           ->
Dcln       -> Name list ',' ':' Type
Type       -> 'integer'
           -> 'boolean'
Block      -> '{' Statement list ';' '}'
Statement  -> Name '=' Expression
           -> 'output' '(' Expression ')'
           -> 'if' '(' Expression ')' Statement 'else' Statement
           -> 'while' '(' Expression ')' Statement
           -> Block
Expression -> Term
           -> Term '<=' Term
Term       -> Term '+' Primary
           -> Term
Primary    -> '-' Primary
           -> 'read'
           -> Name
           -> '<integer>'
           -> '(' Expression ')'
Name       -> '<identifier>'
```

Figura 1. La Sintaxis de Tiny.

El compilador inicial tiene un diseño altamente flexible, extensible, y modificable. Desde el primer día, el estudiante trata con un compilador ya instrumentado para el crecimiento previsto. Se están construyendo implementaciones en C++, Java y C#, para maximizar la flexibilidad del instructor. El nuevo enfoque se asemeja a una espiral: el estudiante repetidamente visita cada componente del compilador (analizador léxico, analizador sintáctico, analizador de semántica estática o restrictor contextual, y generador de código), para agregar nuevas construcciones al lenguaje. Con cada visita, el estudiante obtiene una comprensión más fondo del compilador, sus componentes, su arquitectura, y su estructura. Así, se progresa de lo más sencillo a lo más complejo, en lugar de progresar desde la "parte delantera" hacia la "parte trasera" del compilador.

La diferencia entre los dos enfoques se describe en las Tablas 2 y 3. La desventaja del enfoque tradicional es evidente: cada tema, por ejemplo, análisis sintáctico, debe ser absorbido en su totalidad para que el estudiante implemente las diversas construcciones del lenguaje. Además, la implementación del analizador sintáctico se lleva a cabo de un solo golpe, para el lenguaje en su totalidad, antes de seguir adelante con el siguiente componente del compilador.

	Análisis Léxico	Análisis Sintáctico	Semántica Estática	Generación de Código
Operadores	●	●	●	●
Instrucciones				
Tipos de Datos				
Funciones				
Vectores				
Estructuras	▼	▼	▼	▼

Tabla 2. El Diseño Tradicional del Curso de Compiladores.

	Análisis Léxico	Análisis Sintáctico	Semántica Estática	Generación de Código
Operadores	●→			→
Instrucciones	●→			→
Tipos de Datos	●→			→
Funciones	●→			→
Vectores	●→			→
Estructuras	●→			→

Tabla 3. El Diseño del Curso Nuevo de Traducción.

En contraste, en el nuevo enfoque, el proceso entero de traducción de los operadores del lenguaje (por ejemplo), se discute e implementa, antes de seguir adelante con la siguiente construcción en el lenguaje, más compleja que la anterior.

Esto involucra la modificación del compilador completo, desde su parte delantera, hasta la parte trasera. Temprano en el curso, por ejemplo en el caso de la implementación de operadores, el estudiante imita lo que ve en el compilador, sin tener que entenderlo a fondo todavía. Más adelante en el curso, conforme la comprensión del compilador por parte del estudiante va aumentando, el estudiante se encontrará listo para enfrentarse a la tarea de implementar construcciones más complejas. Quizás la mayor ventaja es que el estudiante experimenta un compilador funcional desde el primer día, y el proyecto exitoso mantiene el compilador funcionando en forma continua durante el período lectivo. La arquitectura del sistema de escritura de traductores se muestra en la Figura 2.

La versión actual del sistema está escrita en C, y utiliza `lex` y `yacc`. La gramática de Tiny, que se muestra en la Figura 2, permite expresiones regulares en las partes derechas de las reglas de producción. Esta gramática se transforma en una gramática libre de contexto pura, en la notación (anticuada) requerida por yacc, a través de un programa pre-procesador llamado `pgen`. Este programa también genera, en forma automática, el código C necesario para la construcción del árbol de análisis sintáctico.

5. Aplicaciones.

Aunque la compilación es el ejemplo principal de traducción por computador, existen muchas situaciones fuera del área de compiladores, en las cuales los principios de la traducción se utilizan. Ejemplos de esto incluyen la minería de datos, los procesadores de líneas de comandos, la traducción y la extensión de lenguajes de marcado como HTML y XML, e interfaces de usuario tanto gráficas como no-gráficas. Parte de la investigación que se reporta aquí involucra la búsqueda de buenos ejemplos de tales aplicaciones, e incorporar su discusión al diseño del curso.

En cuanto a la pedagogía, existe una ventaja adicional potencial. Es bien sabido que en Computación los estudiantes muchas veces se gradúan sin haber nunca participado en la implementación de un programa verdaderamente grande, con más de unas 20,000 líneas de código. La razón es sencilla: ningún curso, o secuencia de curso, puede imponer semejante carga sobre el

estudiante. Aún así, un tema recurrente entre educadores en Computación es la procedencia del *prácticum*, el cómo familiarizar al estudiante con un programa grande, bien estructurado, mantenible, y de buena calidad.

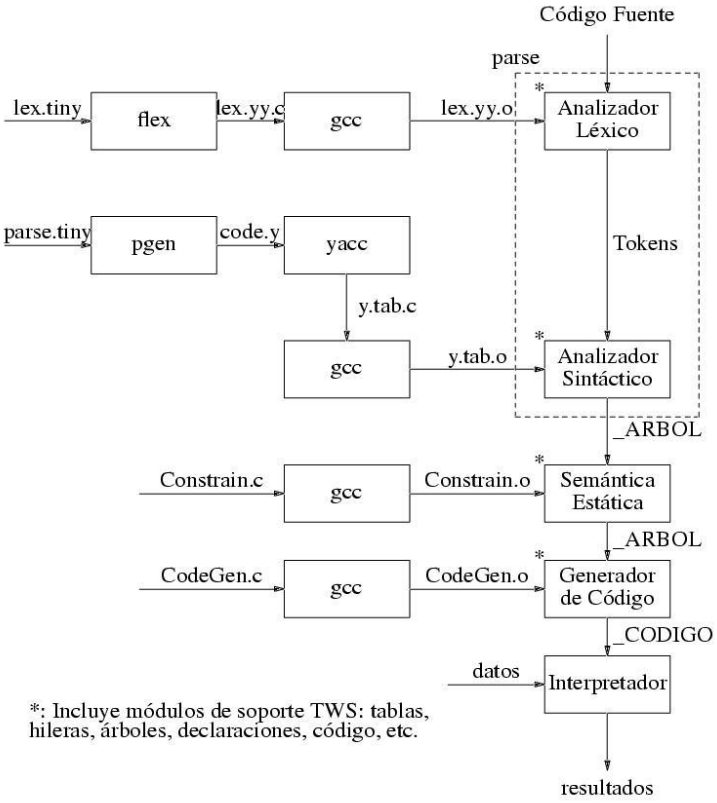

Figura 2. Arquitectura del Sistema de Escritura de Traductores

Creemos que un curso de traductores, estructurado en forma apropiada, puede ser el vehículo a través del cual el estudiante adquiere su primera experiencia con un programa grande, llevando a cabo mantenimiento extenso sobre él, y dirigiéndose a los temas de re-diseño y re-uso de un programa grande.

6. Conclusiones.

Hemos presentado nuestro nuevo diseño de un curso de traducción para estudiantes de pre-grado de Computación o Informática. El nuevo diseño es radicalmente distinto al diseño tradicional. En lugar de discutir los temas en el orden en el cual el compilador realiza su trabajo, en nuestro diseño los temas se discuten en orden ascendente de complejidad. Consideramos que este enfoque es pedagógicamente mucho mejor. Además, discutimos solamente los temas que son fundamentales en la Computación, y descartamos la mayoría de los temas altamente especializados y específicos a la compilación.

Referencias.

[1] ACM Computing Curricula, Final Draft, December 15, 2001, www.computer.org/education/cc2001/final
[2] M.E. Lesk and E Schmidt, Lex - Lexical Analyzer generator, http://dinosaur.compilertools.net
[3] Stephen C. Johnson, Yacc – Yet Another Compiler-Compiler, http://dinosaur.compilertools.net
[4] Aho, A. Sethi, R., and J. Ullman, Compilers – Principles, techniques and Tools, Segunda edición, Addison-Wesley, Reading, Massachussetss, 1986.
[5] Steven John Metsker, Building Parsers With Java, Addison Wesley, 2001.
[6] Watt, D., and Deryck Brown, Programming Language Processors in Java, Prentice Hall, 2000.
[7] James Holmes, Building Your Own Compiler with C++, Prentice Hall, 1995.

[8] Fraser, C., and David Hanson, A Retargetable C compiler: Design and Implementation, Benjamin Cummings, Redwood City, California, 1995.
[9] Jim Holmes, Object-Oriented Compiler Construction, Prentice Hall, Englewood Cliffs, New Jersey, 1995.
[10] Pittman, T., and James Peters, The Art of Compiler Design: Theory and Practice, Prentice Hall, 1992.
[11] Fischer, C., and Richard Leblanc, Crafting a Compiler with C, Benjamin Cummings, Redwood City, California, 1991.
[12] Allen Holub, Compiler Design in C, Prentice Hall, Englewood Cliffs, New Jersey, 1990.
[13] Peter Rochenberg, A Compiler Generator for a Microcomputer, Prentice Hall, Englewood Cliffs, New Jersey, 1989.
[14] Peter Lee, Realistic Compiler Generation, MIT Press, Cambridge, Massachusetts, 1989.
[15] Fischer, C., and Richard Leblanc, Crafting a Compiler, Benjamin Cummings, Menlo Park, California, 1988.
[16] Peter C. Capon, Compiler Engineering Using Pascal, Macmillan, 1988.
[17] Tremblay, J., and Paul Sorenson, The theory and Practice of Compiler Writing, McGraw-Hill, New York, New York, 1985.
[18] Arthur Pyster, Compiler Design and Construction, PWS Publishers, Boston, Massachusetts, 1980.
[19] Barret, W., Bates, R., Gustafson, D., and John Couch, Compiler Construction, Theory and Practice, Segunda edición, Science Research Associates, Chicago, Illinois, 1979.