Revista Antioqueña de las
Ciencias Computacionales y la Ingeniería de Software

ISSN: 2248-7441

www.fundacioniai.org/raccis
raccis@fundacioniai.org

Logic in the curricula of Computer Science

Lógica en los planes de estudios de las Ciencias Computacionales

Margareth Quindeless
LENOVO UK. *quindelessM(A)UKLenovo.com*

INFORMACIÓN DEL ARTÍCULO

Tipo
Reflexión

Historia
Recibido: 30-08-2014
Correcciones: 19-11-2014
Aceptado: 04-12-2014

Keywords
Software development, formal methods, logical reasoning, Discrete Mathematics.

Palabras clave
Desarrollo de software, métodos formales, razonamiento lógico, Matemáticas Discretas.

ABSTRACT
The aim of the programs in Computer Science is to educate and train students to understand the problems and build systems that solve them. This process involves applying a special reasoning to model interactions, capabilities, and limitations of the components involved. A good curriculum must involve the use of tools to assist in these tasks, and one that could be considered as a fundamental is the logic, because with it students develop the necessary reasoning. Besides, software developers analyze the behavior of the program during the designed, the depuration, and testing; hardware designers perform minimization and equivalence verification of circuits; designers of operating systems validate routing protocols, programing, and synchronization; and formal logic underlying all these activities. Therefore, a strong background in applied logic would help students to develop or potentiate their ability to reason about complex systems. Unfortunately, few curricula formed and properly trained in logic. Most includes only one or two courses of Discrete Mathematics, which in a few weeks covered truth tables and the propositional calculus, and nothing more. This is not enough, and higher level courses in which they are applied and many other logical concepts are needed. In addition, students will not see the importance of logic in their careers and need to modify the curriculum committees or adapt the curriculum to reverse this situation.

RESUMEN
El objetivo de los programas en Ciencias Computacionales es formar y capacitar a los estudiantes para que comprendan los problemas y construyan sistemas que los solucionen. Este proceso requiere aplicar un razonamiento especial para modelar las interacciones, las capacidades, y las limitaciones de los componentes involucrados. Un buen plan de estudios debe involucrar la utilización de herramientas que ayuden en estas tareas, y una que se podría considerar como fundamental es la lógica, porque con ella los estudiantes desarrollan el razonamiento necesario. Además, los desarrolladores de software analizan el comportamiento del programa durante el diseño, la depuración, y las pruebas; lo diseñadores de hardware realizan minimización y verificación de equivalencia en los circuitos; los diseñadores de sistemas operativos validan los protocolos de enrutamiento, la programación, y la sincronización; y la lógica formal subyace en todas estas actividades. Por lo tanto, una sólida formación en lógica aplicada les ayudaría a los estudiantes a desarrollar o potencializar su capacidad para razonar sobre sistemas complejos. Desafortunadamente, pocos planes de estudios forman y capacitan adecuadamente en lógica. La mayoría sólo incluye uno o dos cursos de Matemáticas Discretas, en el que en unas cuantas semanas se abarca las tablas de verdad y el cálculo proposicional, y nada más. Esto no es suficiente, y se necesitan cursos de nivel superior en los que se apliquen éstos y muchos otros conceptos lógicos. Además, los estudiantes no le ven la importancia a la lógica en sus carreras y se necesita que los comités curriculares modifiquen o adapten los planes de estudios para revertir esta situación.

1. Introducción

La informática moderna está experimentando una revolución sutil. La Web ha convertido los computadores en cajeros de banco, tiendas, y diversos dispositivos de información. Hoy estos aparatos admiten funciones más críticas, como el control del tráfico aéreo, la supervisión de pacientes, y la recaudación de impuestos, que a su vez exigen un mayor grado de seguridad, fiabilidad, y solidez en los sistemas en todos los niveles. Incluso periódicos como The Economist, The New York Times, y la revista Byte, se han unido desde hace tiempo a la demanda por una mayor fiabilidad de los sistemas de cómputo [1-4].

Desde las Ciencias Computacionales se han propuesto varias técnicas para diseñar, desarrollar, e implementar sistemas fiables, seguros, y robustos; y se ha logrado un progreso significativo en el desarrollo de lenguajes de programación de base matemática, y de técnicas y herramientas para la especificación, el diseño, y la verificación de los mismos. Los lenguajes como Java

incluyen características para apoyar el desarrollo de código seguro [5]; y los diseñadores aplican especificaciones técnicas y de análisis a una amplia gama de proyectos, como en la aviación, el control de tráfico aéreo, los controladores de reactores nucleares, los sistemas de conmutación telefónica, y las políticas de seguridad de organismos como la OTAN [6]. En empresas como Intel e IBM, los diseñadores de hardware desarrollan microprocesadores utilizando sistemas de verificación formal basados en la investigación en lógica aplicada y algoritmos [7, 8]; técnicas de análisis de programas como la depuración estática [9] y la comprobación estática extendida [10] se han aplicado a los productos de consumo; y mediante el análisis de requisitos se han detectado errores en los procesadores de texto comerciales [11]. Las mejoras resultantes en la calidad de los productos demuestran la utilidad de estas técnicas en la construcción de sistemas en el mundo real [12-14].

Por todo esto es que la formación en Ciencias Computacionales necesita actualizarse. Todos los involucrados en tecnologías de la información, desde diseñadores a desarrolladores, desde gerentes a investigadores, deben entender los fundamentos matemáticos y lógicos de la construcción de sistemas complejos. Desafortunadamente, los planes de estudios típicos no involucran ningún intento sistemático por hacer que los estudiantes conozcan y practiquen estos fundamentos en la vida real; es más, la mayoría ni siquiera los incluye en los planes de estudios.

2. Desarrollo de sistemas fiables, seguros, y robustos

Diferentes aplicaciones requieren diferentes grados y formas de protección, seguridad, y robustez. Por ejemplo, un administrador web debe garantizar que los scripts CGI no causen volcados de núcleo; el software de comercio electrónico que la información sensible siempre esté cifrada al ingresar en una red pública; y los sistemas de control de tráfico aéreo disparar alarmas cuando las aeronaves están muy cerca. Por otro lado, la identificación de estos requisitos es una habilidad específica del dominio, es decir, no se pueden definir razonablemente los requisitos de un sistema bancario sin conocer los tipos de transacciones que los usuarios necesitan llevar a cabo, ni el nivel de sensibilidad de la información utilizada en cada transacción. Sin embargo, las habilidades para indicar requisitos y analizar sistemas hacen parte de un dominio independiente e implican expresar ideas en notaciones formales y establecer relaciones entre los estados. La lógica proporciona las bases matemáticas para estas actividades [15]. Su papel en el razonamiento sobre los sistemas es evidente cuando:

- Los desarrolladores deben analizar el comportamiento de los programas. Las afirmaciones sobre la conducta son declaraciones lógicas acerca de la semántica del lenguaje, y a través de reglas lógicas el análisis de programas deriva la información relacionada con estas afirmaciones [16, 17].

- Los diseñadores de hardware utilizan puertas con una correlación directa a la lógica booleana. La minimización de circuitos y la comprobación de equivalencias se basan en la equivalencia lógica de los diseños [18].

- Los diseñadores de redes proponen protocolos de enrutamiento, programación, y sincronización. Los diseñadores de bases de datos desarrollan protocolos para el control de concurrencia y el bloqueo de datos, y establecer protocolos de corrección requiere un lenguaje de especificación y un marco de verificación correspondiente [19].

En cada caso, la lógica sirve como una herramienta para modelar sistemas y como un lenguaje para expresar sus propiedades deseadas. También proporciona los marcos de análisis requeridos, ya sea a través de procedimientos de decisión automatizados o de pruebas semi-automáticas basadas en técnicas estándar, como la inducción y las derivaciones ecuacionales.

El papel de la lógica en el diseño sistemático no es accidental, porque además de ser la base de las Ciencias Computacionales en los últimos años ha desempeñado un rol cada vez mayor en las áreas de aplicación. El modelo de base de datos relacional, que se basa en la lógica de primer orden, revolucionó el diseño de base de datos y sigue siendo el modelo preeminente [20], y lo mismo sucede con los lenguajes semánticos de programación cuyo centro lo conforman el cálculo y la lógica constructiva [21]. Las ideas de tipos de seguridad y de recolección de desperdicios surgen a partir de estos estudios, y finalmente han entrado en la corriente principal de la informática. Abundan otras aplicaciones significativas, como lo evidencian los resultados de la investigación, y una cuarta parte de los premios Turing han sido entregados a contribuciones relacionadas con la lógica en las Ciencias Computacionales [22]. En reconocimiento de este impacto, a menudo la lógica es llamada el *cálculo de las Ciencias Computacionales*. De hecho, John McCarthy, ganador de este premio en 1971, dijo en 1960 que era razonable esperar que la relación entre la computación y la lógica matemática fuera tan fructífera en el siglo XXI como la que existió en el siglo XIX entre el análisis y la física.

3. Habilidades para el desarrollo de sistemas

El desarrollo de sistemas robustos y complejos requiere tres destrezas principales: 1) habilidad para identificar los requisitos de robustez para una aplicación, 2) capacidad para indicarlos con precisión, y 3) habilidad para analizar un programa en relación con sus requisitos de robustez; pero la mayoría de estudiantes universitarios carece de ellas. Los investigadores han observado que los estudiantes tienen dificultades para formalizar declaraciones en notaciones lógicas y para desarrollar argumentos que las soporten. Muchos, si no la mayoría, describen las propiedades de sus propios programas sólo a nivel de aplicación, no a nivel de especificación. De hecho, a menudo expresan resentimiento cuando deben defender la correctitud u otras propiedades de su código. Incluso, afirman que es demasiado difícil y que requiere mucho tiempo poder apreciar las conexiones entre el razonamiento de los programas y un buen desarrollo.

La primera observación refleja un problema conocido y de larga data. Los estudios de Almstrum [23] muestran que los estudiantes de Ciencias Computacionales generalmente tienen más dificultades para comprender los conceptos lógicos que los no-lógicos. Sin embargo, las otras observaciones son mucho más inquietantes. No es sólo que los estudiantes no puedan razonar acerca de sus programas, sino que no entienden por qué es importante hacerlo; además, han desarrollado actitudes negativas hacia todo el proceso. Sin embargo, esto es importante para el trabajo basado en computadores. Dennis Frailey, un investigador de Ingeniería de Software, subraya que el razonamiento lógico es una de las habilidades esenciales de la que carece la mayoría de graduados en Ciencias Computacionales [24], aunque se reconoce como una de las dos habilidades que más se necesita en el desempeño profesional. El problema no se puede descartar como un caso de desacuerdo entre las universidades y la industria sobre las habilidades necesarias de los profesionales. Comprender por qué un programa resuelve un problema particular es una habilidad fundamental que los empleadores valoran en sus empleados, pero que los estudiantes no la aprendan o no la desarrollen, peor aún que no la aprecien, sugiere un problema en los planes de estudios existentes.

4. Planes de estudios a la altura de las necesidades

Dos problemas principales plagan los planes de estudios:

1. El rol periférico de la lógica fomenta la impresión de que es irrelevante para el programa. En los cursos que los estudiantes consideran directamente relacionados con sus perfiles no se discuten las posibles aplicaciones de la lógica ni los conceptos lógicos involucrados en el razonamiento.

2. En lugar de ello, normalmente los conceptos lógicos básicos son relegados a unas pocas semanas de un curso de Matemáticas Discretas, pero cubriendo sólo las tablas de verdad y la lógica proposicional, y, si hay tiempo y le interesa al profesor, las técnicas básicas de prueba. Además, las aplicaciones a problemas reales están más allá del conocimiento y el dominio de los estudiantes, por lo que estos cursos a menudo lo ignoran. Por el contrario, los cursos de lógica superior, que se ofrecen en algunas universidades y son tomados por un número de estudiantes menor, ignoran las solicitudes y llegan demasiado tarde en los programas. Alternativamente, algunos ofrecen cursos de lógica que sirven facultades de filosofía o de matemáticas, y, aunque pueden dedicar más tiempo a algunos aspectos de la lógica, no hacen hincapié en las aplicaciones en Ciencias Computacionales.

   En pocas palabras, el problema es, por un lado, la separación entre la teoría y la práctica, y por el otro, que pocos estudiantes aprecian la utilidad de material teórico aislado. Además, la presentación tradicional del problema de la lógica tiene dos defectos:

3. Los cursos presentan sólo una visión limitada de la lógica. Los estudiantes la ven sobre todo en el contexto del álgebra booleana o en componentes de hardware simples, con acercamientos a otros roles de la lógica en lenguajes de especificación y procedimientos de decisión. Esto refuerza la creencia de que la lógica es irrelevante para la mayoría de contenidos.

4. Los cursos presentan la lógica en un estilo deductivo y pasivo que entra directamente en conflicto con la mayoría de los estilos de aprendizaje de los estudiantes. En consecuencia, pocos la internalizan o aprecian.

Los estilos de aprendizaje tienen gran impacto en cómo los estudiantes responden al material y a la didáctica del curso [25]. La investigación sobre estos estilos intenta correlacionar los rasgos de personalidad comunes con el bajo éxito de las formas particulares de instrucción, la didáctica, y la evaluación. Y desconocen que estudiantes con diferentes estilos de aprendizaje pueden responder de manera muy diferente al mismo curso. Felder [26] describe los estilos de aprendizaje a lo largo de cinco ejes: visual vs verbal, inductivo vs deductivo, activo vs reflexivo, secuencial vs global, y sensorial vs intuitivo. Los planes de estudios convencionales en lógica se orientan a estudiantes intuitivos, verbales, deductivos, reflexivos, y secuenciales. Las encuestas a estudiantes de ingeniería sugieren que este conjunto de preferencias representa aproximadamente 1/3 de los estudiantes hombres y 17% de los estudiantes mujeres [27]. Si bien estas cifras sólo sirven como guía, sugieren fuertemente que se necesita una variedad más amplia de métodos de enseñanza para animar a los estudiantes a aprender o desarrollar habilidades fuertes en lógica aplicada. Resultados similares se han visto en otras disciplinas, como se describe en la revisión de Tobías [28].

En conjunto, los cuatro problemas mencionados dan como resultado pocos graduados en Ciencias Computacionales, y los egresados no desarrollan los cimientos ni las habilidades necesarios para desarrollar sistemas fiables, seguros, y robustos. Atender sólo un sub-conjunto de los problemas es insuficiente; para mejorar las reacciones de los estudiantes a las técnicas lógicas e incrementar su capacidad para utilizarlas en situaciones prácticas se requiere mejoras curriculares que aborden las cuatro áreas problemáticas simultáneamente.

5. Una propuesta de solución

Una solución debe tener como objetivo integrar el razonamiento lógico aplicado y las herramientas software lógicas en los cursos existentes y ampliamente utilizados en Ciencias Computacionales. Por varias razones un enfoque de este tipo se adapta bien para abordar los problemas:

1. Una perfecta integración debería incrementar en gran medida la motivación de los estudiantes. Se proporcionan ejemplos concretos de cómo la lógica sirve como una herramienta valiosa en el trabajo práctico. El éxito de la cobertura de las gramáticas libres de contexto en cursos de compiladores demuestra la utilidad de este tipo de fusiones. Los

estudiantes que se quejan de la falta de pertinencia de otro material en la teoría de los cursos de computación aceptan conferencias sobre estas gramáticas en el contexto del diseño del compilador.

2. La integración y el uso de herramientas software lógicas les ayudan a los estudiantes a que captan mejor los conceptos concretos, en lugar de los abstractos, de las situaciones de aprendizaje. Mientras los estudiantes trabajan con herramientas software para analizar sus programas, entienden qué teoremas median en un dominio específico y cómo verificar las relaciones entre los procedimientos.

3. La integración del material en varios cursos refuerza conceptos lógicos durante todo el plan de estudios, lo que incrementa la capacidad de los estudiantes para comprenderlo y aplicarlo.

La idea es desarrollar módulos didácticos estructurados con directrices sobre cómo integrarlos en los cursos regulares, como bases de datos, Inteligencia Artificial, Ingeniería de Software, arquitectura, compiladores, y programación concurrente. Cada uno contendrá aproximadamente tres semanas de material, y les permitirían a las facultades integrar contenidos sobre modelado lógico y desarrollo de sistemas robustos sin necesidad de nuevos cursos. En la mayoría de casos, en estos módulos se podría aplicar la perspectiva del modelado lógico y las especificaciones robustas a temas existentes en los planes de estudios. Al incluirlos en una clase sólo habría necesidad de desplazar un poco las temáticas del curso. Esta estrategia aumentaría las oportunidades para que las facultades puedan cubrir el material de la lógica a través de múltiples cursos de manera uniforme; esto va más allá de los esfuerzos reportados en otras instituciones que integran el material formal sobre cursos aislados [29-31]. Cada módulo podría consistir de un libro de texto corto, notas de clase en línea, presentaciones, problemas de programación, conjuntos de problemas, y las directrices para el uso de herramientas.

Algunas personas, entre investigadores y profesores, han clamado por una mayor integración de la lógica en los planes de estudio de las Ciencias Computacionales [32-34]; una encuesta desarrollada en una facultad en esta área indica la existencia de una creencia general de que se necesitan técnicas más rigurosas en los cursos de desarrollo de software [35]; además, la experiencia de algunos investigadores apoya la integración de herramientas software lógicas y el enfoque general de módulos propuesto. Por ejemplo, un curso de Matemáticas Discretas de segundo año podría basarse totalmente en herramientas para reforzar los conceptos matemáticos [36]. Los profesores podrían llevar a cabo sus propias pruebas en este mismo sentido, y desarrollar módulos de Ingeniería de Software en los que se involucren los conceptos de lógica avanzada. Los estudiantes aprenderían a declarar propiedades formales acerca de programas simples, y explorarían las pruebas inductivas ecuacionales para el establecimiento de esas propiedades. De esta forma se darían cuenta en qué grado se integra la lógica a sus cursos, tanto en términos del material como desde la perspectiva de los estudiantes. Pero esto necesita voluntad e iniciativa, sobre todo, que los profesores estén realmente preparados para formar en lógica avanzada.

6. Conclusiones

Los planes de estudios de las Ciencias Computacionales deben proporcionarles a los estudiantes una base sólida en las habilidades de razonamiento necesarias para diseñar y desarrollar artefactos computacionales. La lógica, el llamado *cálculo de las Ciencias Computacionales*, subyace en la mayoría de los enfoques para la construcción de sistemas fiables, seguros, y robustos. Por lo tanto, los planes de estudios deben desarrollar estas habilidades en los estudiantes para usar la lógica como herramienta en su desempeño profesional. Una alternativa para lograrlo es mediante el ofrecimiento de módulos concretos de materiales de enseñanza, incluyendo notas de clase, sugerencias de asignación, y herramientas guía. Pero lo más importante es que los profesores se preparen adecuadamente para materializar estas iniciativas y para responder a las necesidades educativas de este siglo, porque la lógica es fundamental para comprender y solucionar los problemas complejos de la sociedad actual.

Referencias

[1] Cairncross, F. (1998). Midnight's children. The Economist, 348(8086), pp.15-17, September 19.

[2] Politicians Woo Voters (1998). Flaw in email programs points to an industrywide problem. The New York Times, July 30.

[3] Halfhill, T. (1998). Crash-proof computing. Byte 23(4), pp. 60-74.

[4] Minasi, M. (1999). The software conspiracy: Why software vendors produce faulty products, how they can harm you, and what you can do about it. McGraw-Hill.

[5] Gosling, J.; Joy, B. & Steele G. (2014). The Java language specification. Addison-Wesley Professional.

[6] Clarke, E. & Wing, J. (1996). Formal methods: State of the art and future directions. ACM Computing Surveys 28(4), pp. 626-643.

[7] Kurshan, R. (1997). Formal verification in a commercial setting. Proceedings 34th ACM/IEEE Design Automation Conference, pp. 258-262. June 9-13, Anaheim, USA.

[8] Schlipf, T. et al (1997). Formal verification made easy. IBM Journal of Research and Development 41(4-5), pp. 567-576.

[9] Liblit, B. & Nainar, P. (2012). Applications of static analysis and program structure in statistical debugging. Doctoral Dissertation, University of Wisconsin.

[10] James, P. & Chalin, P. (2010). Faster and more complete extended static checking for the Java Modeling Language. Journal of Automated Reasoning 44(1-2), pp. 145-174.

[11] Jackson, D. & Damon, C. (1996). Elements of style: Analyzing a software design feature with a counterexample detector. IEEE Transactions on Software Engineering 22(7), pp. 484-495.

[12] Craigen, D.; Gerhart, S. & Ralston, T. (1993). An international survey of industrial applications of formal methods. Technical Report NIST GCR 93/626, U.S. National Institute of Standards and Technology.

[13] Eiriksson, A. (2000). The formal design of 1M-gate ASICs. Formal Methods in System Design - Special issue on formal methods for computer-added design 16(1), pp. 7-22.

[14] Jacky, J. et al (1997). Experience with Z: Developing a control program for a radiation therapy machine.

Proceedings 10th International Conference of Z users on the Z formal specification notation, pp. 317-328. April 3-4, Reading, UK.
[15] van Leeuwen, J. (1990). Formal models and semantics - Handbook of Theoretical Computer Science. MIT Press.
[16] Bourdoncle, F. (1993). Abstract debugging of higher-order imperative languages. Proceedings ACM SIGPLAN Conference on programming language design and implementation, pp. 46-55. June 21-25, Albuquerque, USA.
[17] Flanagan, C. et al (1996). Catching bugs in the web of program invariants. Proceedings ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 1-10. May 21-24, Philadelphia, USA.
[18] Hachtel, G. & Somenzi, F. (2006). Logic synthesis and verification algorithms. Springer.
[19] Holzmann, G. (1990). Design and validation of computer protocols. Prentice-Hall.
[20] Codd, E. (1970). A relational model for large shared data banks. Communications of the ACM 13(6), pp. 377-387.
[21] Barendregt, H. (1985). The lambda calculus: Its syntax and semantics. North-Holland.
[22] ACM (1991). ACM Turing Award Lectures: The first twenty years 1966-1985. ACM Press.
[23] Almstrum, V. (1996). Investigating Student difficulties with mathematical logic. In Dean, N. & Gerard, M. (Eds.), Teaching and Learning Formal Methods, pp. 131-160. Academic Press.
[24] Frailey, D. (2011). Careers in computing - How to prepare and what to expect. ACM Colloquium. November 16.
[25] Lawrence, G. (2009). People types and tiger stripes: Using psychological type to help students discover their unique potential. Center for Applications of Psychological Type.
[26] Felder, R. (1993). Reaching the second tier: Learning and teaching styles in college science education. Journal of College Science Teaching 23(5), pp. 286-290.
[27] McCaulley, M. (1990). The MBTI and individual pathways in engineering design. Engineering Education 80, pp. 537-542.
[28] Tobias, S. (1992). Revitalizing undergraduate science: Why some things work and most don't. Research Corporation.
[29] Gries, D. & Schneider, F. (1993). A logical approach to discrete math. Springer-Verlag.
[30] Sobel, A. (1996). Experience integrating a formal method into a software engineering course. ACM SIGCSE Bulletin 28(1), pp. 271-274.
[31] Tremblay, G. (1998). An undergraduate course in formal methods: "Description is our business". ACM SIGCSE Bulletin 30(1), pp. 166-170.
[32] Myers, J. (1990). The central role of mathematical logic in computer science. ACM SIGCSE Bulletin 22(1), pp. 22-26.
[33] Morgenstern, L. & Thomason, R. (2000). Teaching knowledge representation: Challenges and proposals. In Cohn, A.; Giunchiglia, F. & Selman, B. (Eds.), KR Morgan Kaufmann, pp. 725-733.
[34] Johnson, S. et al (1998). Report on the 21st Century Engineering Consortium Workshop a forum on formal methods education. March 18-19, Melbourne, USA.
[35] Palmer, T. & Pleasant, J. (1995). Attitudes toward the teaching of formal methods of software development in the undergraduate computer science curriculum: a survey. ACM SIGCSE Bulletin 27(3), pp. 53-59.
[36] Barwise, J. & Etchemendy, J. (1994). Hyperproof: Logical reasoning with diagrams. CSLI Lecture Notes. University of Chicago Press.