

La Lógica en las Ciencias Computacionales

Edgar SERNA M.

Instituto Tecnológico Metropolitano. Facultad de Ingenierías
edgareserna@itm.edu.co
Medellín, Colombia.

Alexei SERNA A.

Instituto Antioqueño de Investigación. Grupo de investigación CCIS
alexei.serna@fundacioniai.org
Medellín, Colombia.

RESUMEN

En este artículo se hace un análisis a la necesidad de incluir a la lógica en los procesos formativos en Ciencias Computacionales (CC). Se parte de un recorrido a la historia de la lógica en estas ciencias, posteriormente se describe la relación y la necesidad de incluirla en los procesos formativos relacionados con ellas, y al final se analiza qué, cuándo y qué tan profundo se debería trabajar la lógica en la formación en CC. Se trata de una revisión al estado de esta cuestión y a la importancia de incluir la lógica tanto en los pregrados como en los posgrados relacionados con estas Ciencias y las TI. Los resultados de estas investigaciones tienen amplio impacto en el desarrollo de las CC, porque deben estar reflejadas en los procesos de formación relacionados al igual que con el papel cada vez mayor de la lógica.

Palabras Clave: Lógica, formación, Ciencias Computacionales, resolución de problemas, abstracción.

1. INTRODUCCIÓN

En los últimos años se han desarrollado poderosas herramientas para verificar especificaciones de sistemas hardware y software. La industria de TI se ha dado cuenta del impacto y la importancia que tienen en sus procesos de diseño e implementación, al punto que empresas como Intel, Siemens, BT, AT&T e IBM las investigan e incorporan en sus departamentos de planeación y producción. Para trabajar en estos procesos se requiere una formación formal básica que les permita a los estudiantes y profesionales que trabajan en esos entornos obtener suficientes competencias para *utilizar*, *razonar* y *potencializar* los sistemas. El cambio de las TI para favorecer al acceso a los datos con base en Internet y el procesamiento, generó un incremento en la demanda por profesionales calificados que puedan razonar acerca del software sofisticado basado en agentes autónomos y capaces de interactuar con otros agentes para recopilar la información necesaria en las grandes redes. La base de estos desarrollos son las Ciencias Computacionales, en las que se requiere

aplicar y manejar adecuadamente la lógica como componente formal.

En este trabajo se analiza este campo temático para responder a las necesidades formativas actuales, y se proporciona las bases acerca de la formación que requieren en lógica los estudiantes y profesionales para desempeñarse en este campo. Además, se hace una introducción a los marcos lógicos utilizados para modelar y razonar acerca de los sistemas informáticos. El objetivo es proporcionar un contenido que se pueda utilizar para diseñar cursos de formación en lógica computacional, como una contribución para que los estudiantes se puedan adaptar rápidamente a entornos profesionales actuales en medio del acelerado y cambiante entorno de las TI.

2. LÓGICA Y CIENCIAS COMPUTACIONALES

La Magna Charta Universitatum [1], emitida con ocasión de los 900 años de la fundación de la Universidad de Bolonia, resumió la misión de las universidades en la sociedad moderna. Ese mismo año y por iniciativa del ministro francés de educación, la Sorbonne Declaration [2] estableció el reconocimiento mutuo de las respectivas titulaciones como un objetivo común de los cuatro países signatarios: Francia, Alemania, Italia y Reino Unido. Muchos otros países aceptaron esas ideas y expresaron su disposición de unirse al proyecto.

Para enfatizar su compromiso, los ministros de educación de 29 de estos países se reunieron en Bolonia en 1999, y se comprometieron en una declaración conjunta [3] a establecer un espacio europeo en educación superior para el año 2010. El objetivo del Bologna Process era lograr que los sistemas europeos de formación superior confluyeran

hacia un sistema más transparente, mediante el cual los diferentes sistemas nacionales utilizaran un marco común basado en tres ciclos formativos: Licenciatura, Maestría y Doctorado. Desde entonces, los países signatarios han organizado varias reuniones —Praga 2002, Berlín 2005, Londres 2007, Leuven 2009, Budapest 2010— para revisar los objetivos del proceso a corto plazo y clarificar cuestiones para su implementación.

Posteriormente, aunque de manera tangencial, otros países entraron al proyecto con el objetivo de adoptar los principios que en él se discuten e implementarlos en sus propios sistemas de formación. En América Latina se está trabajando seriamente al respecto y algunos países han comenzado a modificar sus estructuras en pregrado y posgrado. La idea es no rezagarse con respecto a los progresos que se hacen en Europa y EE.UU., porque las TI requieren actualización constante y profesionales capacitados para su explotación. En este contexto, la revisión y re-estructuración de los planes de estudio en Ciencias Computacionales se realizan bajo la supervisión de investigadores interesados en el tema, y se ha llegado a conclusiones como que es necesario contar con una profesionalización en desarrollo de software, es decir, una Ingeniería de Software reconocida como *Ingeniería* [4, 5].

Aunque esta ingeniería se considera actualmente como sucesora directa de la Ingeniería de Sistemas y los planes de estudio desarrollados están fuertemente basados en las experiencias locales relacionadas con el mantenimiento de más de 30 años de la misma, los cambios en el sistema formativo brindan la oportunidad de hacer una revisión importante de los objetivos y contenidos en ambas áreas de formación. La revisión que se presenta en este trabajo se llevó a cabo teniendo en cuenta el papel y el contenido de la lógica en la formación en Ciencias Computacionales, especialmente en el área de la Ingeniería de Software.

A comienzos de siglo pasado la axiomatización de la lógica de Hilbert [6] consideraba a la lógica como una teoría igualmente axiomatizada. Según este enfoque es posible demostrar teoremas por medio de los métodos matemáticos tradicionales, sin embargo, para entonces no existían algoritmos que soportaran la construcción de deducciones. El primer avance significativo se debe a Gentzen [7] con el desarrollo de la técnica natural de deducción y el cálculo

sucesivo, con los que creó un kit de herramientas sintácticas especiales para probar automáticamente un teorema. Posteriormente, el modelo teórico de Herbrand [8] aportó la probabilidad de insatisfacción sobre los universos de Herbrand, e hizo posible la re-escritura del problema de decisión de primer orden como una fórmula proposicional que permite expandirlo sobre dichos universos, con lo que el problema de demostrar un teorema se redujo a la revisión de las fórmulas a través de un kit de herramientas de lógica proposicional.

Cuando los primeros computadores fueron accesibles, Davis y Putnam [9] utilizaron los resultados de Herbrand y elaboraron el primer algoritmo de computador para demostrar el teorema. Newell y Simon [10] desarrollaron el *General Problem Solver* y Newell, Shaw y Simon [11] diseñaron el sistema *Logic Theorist*, trabajos que impactaron el desarrollo de la Inteligencia Artificial contemporánea. El primero utiliza el algoritmo del *British Museum*, un método de búsqueda horizontal a ciegas de bajo rendimiento, basado en los axiomas y las reglas de inferencia de Russell y Whitehead [12].

La investigación del problema de insatisfacción de las fórmulas CNF —Conjunctive Normal Form— canónicas y la generalización de la regla de resolución proposicional para las fórmulas de primer orden, llevaron a la idea de la resolución lógica proposicional o resolución básica. Robinson [13] definió la noción de la resolución de primer orden con base en que, si la resolución existe, es posible utilizar dos cláusulas que contengan las instancias básicas cuando aparece una pareja literal básica complementaria. Además, reconoció que la condición para que exista una resolución básica es que se pueda unificar el par literal original, y con este principio creó el cálculo de resolución de primer orden. Posteriormente, otros trataron de desarrollar estrategias de resolución para simplificar la implementación, pero hasta el momento las principales estrategias resultantes son la resolución semántica, la resolución lineal —cálculo completo—, la resolución de entrada lineal y la resolución de unidad, posibles de implementar pero todavía incompletas.

En los 60 se demandó la aplicación de la lógica en conexión con el análisis y la síntesis, lo que significó

que las propiedades de un programa se describieran mediante fórmulas lógicas —axiomas—, y que fuera posible tratar de responder las preguntas acerca del funcionamiento correcto del mismo. La base formal de este enfoque era principalmente la lógica de Hoare [14], cuyos aportes contienen los resultados más importantes [15-17]. Los llamados sistemas de pregunta/respuesta fueron herramientas útiles, en los que la respuesta era una supuesta consecuencia/inferencia de las fórmulas que describen las propiedades del programa. Estos sistemas fueron dotados de una técnica especial que adicionalmente podía generar cierto tipo de respuestas acerca de la consistencia del teorema, siempre que se hubiera demostrado a sí mismo. Aquí se utilizaron en la formalización a la lógica de segundo orden e incluso a la temporal [15, 18]. Los trabajos de Cliff [19], Hoare [20], Loeckx y Sieber [21] y Goos, Hartmanis y Leuwen [22], ilustran que ésta es un área importante en el desarrollo y aplicación de la lógica.

A principios de los 80 y utilizando la estrategia de entrada lineal incompleta, Kowalski [23] y Colmerauer [24] desarrollaron el teorema de pruebas del Prolog para las cláusulas de Horn de primer orden, que se utilizaban para definir las declaraciones lógicas de un programa. La investigación acerca de las capacidades de este sistema concluyó que, sobre la base del principio del modelo dado por los átomos de la primera capa, era posible desarrollar la noción del modelo mínimo de Herbrand y de cierto tratamiento de negación. Dado que el resultado de un trazo de asignación ordenado y definido en un reticulado completo siempre tiene un punto fijo, entonces una de las interpretaciones de Herbrand es un subconjunto del conjunto de átomos de la primera capa sobre su universo —es decir, un sub-conjunto de la base de Herbrand—, donde el conjunto de todas sus interpretaciones es un super-conjunto de la base.

El resultado del conjunto de todas las interpretaciones de Herbrand es un reticulado completo con dos operadores y una relación del sub-conjunto. Una consecuencia directa de la función asignada a la lógica de un programa sobre esta red es la preservación de la ordenación, por lo que al menos tiene un punto fijo. Se ha demostrado que este es el modelo mínimo de Herbrand de la lógica de un programa, con lo que se definió la extensión del

punto fijo de la lógica de primer orden [25], que juega un papel significativo en el enfoque DATALOG de las bases de datos relacionales y las bases del conocimiento.

La investigación de los sistemas de deducción lógica tuvo gran impacto en el desarrollo de la teoría de modelos, principalmente en la evaluación de las cuestiones de la axiomatización [26]. La aparición de los lenguajes funcionales y el desarrollo de las tecnologías de programación paralela y concurrente condujeron a la introducción de nuevas herramientas en la teoría de la programación. Una de ellas es λ -cálculo, presentada por Church [27] y utilizada para el tratamiento unificado de lógicas de orden diferente —nulo, primero, superior— [28]. La evolución de los lenguajes de programación involucró la evolución de herramientas como π -cálculo y μ -cálculo, entre otras.

Desde los años 60 los kits de herramientas de Inteligencia Artificial, la teoría de la programación y la teoría basada en el conocimiento incluyen, como herramientas lógicas no clásicas, a las diversas versiones de la lógica temporal, la lógica modal, las lógicas de valores diversos, las lógicas relevantes, la lógica no-monótona y la lógica difusa.

3. LÓGICA Y FORMACIÓN EN CIENCIAS COMPUTACIONALES

Inicialmente se formaba en lógica sólo como una técnica descriptiva, pero ya en los años 60 las aplicaciones suponían el conocimiento de amplias áreas de la misma, lo que implicaba que los temas básicos de la lógica matemática se debían incluir en los planes de estudio de los cursos avanzados. Desde entonces la lógica se convirtió en una herramienta para otros campos de las Ciencias Computacionales, como la Inteligencia Artificial, la teoría de bases de datos relacionales y el análisis y síntesis de programas. Desde finales de esa década las bases de la lógica comenzaron a hacer parte de las lecturas y de la formación en los primeros años y, dado que los campos mencionados comenzaron a jugar un papel importante en las aplicaciones, se desplazó luego a los años superiores. Debido a estos reconocimientos y a las necesidades en el desarrollo de las TI, desde finales de los 70 se empezaron a publicar trabajos acerca de la lógica para estudiantes de Ciencias Computacionales y para especialistas avanzados [29-

36], al mismo tiempo que se publicaron manuales de resúmenes [37-43].

3.1 Qué, cuándo, cuánta lógica

Luego de revisar la relación histórica de la lógica y las Ciencias Computacionales y de analizar la necesidad de incluirla desde los primeros años de pregrado y posgrado, a continuación se presenta una propuesta de trabajo para lograr mejores resultados en los procesos formativos.

En la fase actual del desarrollo de las Ciencias Computacionales se conoce cuál es el conocimiento y qué parte de la lógica se debe formar a los estudiantes, y se concluye que tanto en el currículo como en la formación básica se le debe capacitar en lógica proposicional y de primer orden, mediante un enfoque orientado al lenguaje, el alfabeto, la sintaxis, la semántica, las reglas de re-escritura, la noción de consecuencia semántica, el teorema de demostración de problemas, el problema de la decisión, la solución semántica de la prueba de teoremas, el tratamiento sintáctico de la lógica y el principio de los sistemas de deducción. Además, que se debe incluir el cálculo de resoluciones y un método arbitrario, conectados por el sistema de Hilbert [44], como el más importante de los dos sistemas de deducción conocidos.

También se debe formar en las nociones de correctitud y completitud, de tal manera que se sienten las bases necesarias para la introducción de las lógicas temporal, de Hoare, de unidad, de punto fijo y la descriptiva [45], utilizadas en la formación en diversas áreas de las Ciencias Computacionales. Aunque no se recomienda utilizarla, en cierta medida, en la formación en bases de datos, teoría de la programación y la Inteligencia Artificial, pero es un punto de partida para introducir las lógicas no clásicas [46].

Luego que los estudiantes reciben esta fundamentación se podrán introducir a los lenguajes de programación funcional [47]. Los cursos de pregrado enfocados en la teoría de base de datos o la Inteligencia Artificial no son los únicos que requieren el conocimiento de la lógica fundamental, también los estudios de posgrado se deben fundamentar aquí.

Luego de analizar los currículos relevantes de

algunas universidades en las que se imparte la lógica en la formación en posgrados, se puede concluir que los campos más importantes de formación en esta área son:

- Los fundamentos teóricos y los métodos de prueba de teoremas.
- Los problemas de formalización y axiomatización: problemas de las teorías axiomatizadas.
- Cálculo en el marco de la programación funcional.
- La programación lógica y sus fundamentos teóricos: lógica de punto fijo y el modelo Herbrand de un programa.
- Lógicas no clásicas: lógica temporal, lógica polivalente, lógica difusa, lógica descriptiva, entre otras.

Todos estos temas son importantes para incluirlos en el marco de un programa de posgrado, al tiempo que se justifica la existencia de temas como lógica no clásica a nivel de pregrado, al menos como curso electivo.

4. CONCLUSIONES

En este trabajo se presenta un breve recorrido histórico acerca de la lógica y las posibilidades y ventajas de la formación en Ciencias Computacionales con base en ella. Estas ideas parten de la conceptualización que introdujo el Tratado de Bolonia para la formación en informática, pero que redundan en todo el planeta.

El acercamiento propuesto hace hincapié en la formación acerca de los principales métodos de inferencia, incluyendo la resolución no causal, y ofrece una introducción a la teoría y aplicación de las lógicas de muchos valores y otras que se difunden y aplican en la industria de TI.

La formación en lógica se debe realizar no sólo en el marco de las matemáticas sino también en el de la formación en Ciencias Computacionales, pero con un aspecto inter y multi-disciplinar para lograr eficiencia, lo que significa que ambas áreas del conocimiento tienen que colaborar [48]. Lo básico debe ser inherente a la formación lógico-matemática, como la abstracción, los lenguajes, las semánticas, los conectores, las interpretaciones, las tautologías, el cálculo proposicional y lógica computacional. Pero en los posgrados se debe incluir aspectos

avanzados, como sintaxis vs semánticas, representación de conocimiento, tareas problemáticas de la *vida real*, deducción y métodos formales, resoluciones y lógicas de muchos valores [49].

Además, se deben utilizar las herramientas informáticas actuales para que la formación sea más interesante para los estudiantes: lógica de programación, máquinas de inferencia visual con ejemplos preparados y herramientas CASE, matemática discreta aplicada, entre otras. También es necesario mantener una actualización constante mediante seminarios, asignaturas de aplicación práctica en la industria y laboratorios experimentales [50].

5. REFERENCIAS

- [1] Bolonia (1988). [Magna Charta Universitatum](#). En línea. [Mar. 2012].
- [2] Sorbonne (1988). [Sorbonne Declaration](#). En línea. [May 2012].
- [3] CRE (2000). [The Bologna Declaration on the European space for higher education: An explanation](#). En línea. [May 2012].
- [4] Serna, M. E. (2011). [Ingeniería de sistemas para el siglo XXI: Una propuesta desde la academia](#). *Proceedings Ninth LACCEI Latin American and Caribbean Conference (LACCEI'2011)*, August 3-5, Medellín, Colombia.
- [5] Serna, M. E. (2011). [La “Ingeniería de Software” es Ingeniería](#). *Revista RACCIS*, 1(1), pp. 34-43.
- [6] Hilbert, D. (1920). Die Grundlagen Der Elementaren Zahlentheorie. *Mathematische Annalen*, 104, pp. 485-494. Ewald, W. (Trad.), *The Grounding of Elementary Number Theory*. En: Mancosu, P. (Ed.), [From Brouwer to Hilbert: The debate on the foundations of mathematics in the 1920's](#) de Mancosu. Oxford University Press, 1997, pp. 1157-1165.
- [7] Gentzen, G. (1918). [Untersuchung Über das logische Schliessen](#). *Mathematische Zeitschrift*, 39, pp. 176-210.
- [8] Herbrand, J. (1971). [Logical writings](#). Springer.
- [9] Davis, M. & Putnam, H. (1960). [A Computing Procedure for Quantification Theory](#). *Journal of the ACM*, 7(3), pp. 201-215.
- [10] Newell, A. & Simon, H. A. (1956). [The logic theory machine: A complex information processing system](#). *IRE Transactions on Information Theory*, 2(3), pp. 61-79.
- [11] Newell, A., Shaw, J. C. & Simon, H. A. (1959). [Report on a general problem-solving program](#). *Proceedings of the International Conference on Information Processing*. Paris, pp. 256-264.
- [12] Russell, B. & Whitehead, A. N. (1997). *Principia Mathematica* to *56. Cambridge University Press.
- [13] Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1), pp. 23-41.
- [14] Hoare, C. A. R. (1969). [An Axiomatic Basis for Computer Programming](#). *Communications of the ACM*, 12(10), pp. 576-580.
- [15] Manna, Z. & Pnueli, A. (1974). [Axiomatic approach to total correctness of programs](#). *Acta Informatica*, 3(3), pp. 243-263.
- [16] Manna, Z. & Waldinger, R. (1971). [Toward Automatic Program Synthesis](#). *Communications of the ACM*, 14(3), pp. 151-165.
- [17] Pnueli, A. (1977). [The temporal logic of programs](#). *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*. Rhode Island, USA, pp. 46-57.
- [18] Kröger, F. (1987). [Temporal Logic of Programs](#). Springer.
- [19] Cliff, B. J. (1980). [Software development. A rigorous Approach](#). New York: Prentice Hall.
- [20] Hoare, C. A. R. (1985). [Mathematical Logic and Programming Languages](#). Prentice-Hall.
- [21] Loeckx, J. & Sieber, K. (1987). [The Foundations of Program Verification](#). Wiley.
- [22] Dershowitz, N. (2004). *Verification: Theory and Practice*. Springer.
- [23] Kowalski, R. A. (1979). [Algorithm = Logic + Control](#). *Communications of the ACM*, 22(7), pp. 424-436.
- [24] Colmerauer, A. (1970). [Les systèmes-q ou un formalisme pour analyser et synthétiser des phrases sur ordinateur](#). Internal publication 43, Département d'informatique de l'Université de Montréal.
- [25] Abiteboul, S., Hull, R. & Vianu, V. (1995). [Foundations of Databases](#). USA: Addison-Wesley.
- [26] Ebbinghaus, H. D., Thomas, W. & Flumm, J. (1994). [Mathematical Logic](#). Springer.
- [27] Church, A. (1932). [A set of Postulates for the Foundation of Logic](#). *Annals of Mathematics, second series*, 33, pp. 346-366.
- [28] Andrews, P. B. (2002). [An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof](#). Springer.
- [29] Bergmann, E. & Noll, H. (1977). [Mathematische Logik mit Informatik-Anwendungen](#). Springer-Verlag.
- [30] Richter, M. (1978). [Logikkalküle](#). Teubner.
- [31] Gallier, J. H. (1986). [Logic for Computer Science](#). Longman Higher Education.
- [32] Schöning, U. (1987). [Logik für Informatiker](#). Verlag.
- [33] Börger, E. (1989). [Computability, Complexity, Logic](#). North-Holland.

- [34] Fitting, M. (1995). [First-Order Logic and Automated Theorem Proving](#). Springer.
- [35] Nerode, A. & Shore, R. A. (1997). [Logic for Applications](#). Springer-Verlag.
- [36] Huth, M. & Ryan, M. (2004). [Logic in Computer Science - Modelling and Reasoning about Systems](#). Cambridge University Press.
- [37] Barr, A. & Feigenbaum, E. (1981). [Handbook of Artificial Intelligence](#). Heuristic Press.
- [38] Bledsoe, W. & Loveland, D. (1984). [Automatic Theorem Proving after 25 years](#). American Mathematical Society.
- [39] Boyer, R. S. & Moore, J. S. (1997). [A Computational Logic Handbook](#). Morgan Kaufmann.
- [40] Gabbay, D. M., Hogger, C. J. & Robinson, J. A. (1994). [Handbook of Logic in Artificial Intelligence and Logic Programming](#). Oxford University Press.
- [41] Abramsky, S., Gabbay, D. M. & Maibaum, T. E. S. (1995). [Handbook of Logic in Computer Science](#). Oxford University Press.
- [42] Agostino, M. D., Gabbay, D. M., Hähnle, R. & Possega, J. (1999). [Handbook of Tableau Methods](#). Springer.
- [43] Krantz, S. G. (2002). [Handbook of Logic and Proof Techniques for Computer Science](#). Birkhäuser.
- [44] Goguen, H. & Goubault-Larrecq, J. (2000). [Sequent combinators: a Hilbert system for the lambda calculus](#). *Mathematical Structures in Computer Science*, 10(1), pp. 1-79.
- [45] Küsters, R. (2001). [Non-Standard Inferences in Description Logics](#). Springer-Verlag.
- [46] Qualls, J. A. & Sherrell, L. B. (2010). [Why computational thinking should be integrated into the curriculum](#). *Journal of Computing Sciences in Colleges*, 25(5), pp. 66-71.
- [47] Hoare, C. A. H. (2004). [Communicating Sequential Processes](#). McGraw Hill.
- [48] Wing, J. M. (2008). [Five deep questions in computing](#). *Communications of the ACM*, 51(1), pp. 58-60.
- [49] Denning, P. J. (2009). [The Profession of It: Beyond Computational thinking](#). *Communications of the ACM*, 52(6), pp. 28-30.
- [50] Serna, M. E. (2010). Formal Methods and Software Engineering. *Revista Virtual Universidad Católica del Norte*, 30, pp. 158-184.