Matemáticas para la verificación de programas

J. Borrego Díaz*, M. C. Graciani Díaz y M. J. Pérez Jiménez†

Dpto. de Ciencias de la Computación e Inteligencia Artificial
Facultad de Informática y Estadística–Universidad de Sevilla
c/ Tarfia s.n. 41012–Sevilla
E-mail: jborrego@cica.es

Resumen

En este trabajo presentamos las directrices de la formación postgrado sobre verificación formal de programas en el Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Sevilla. Los cursos en los que se articula dicha formación representan realmente una adaptación curricular, tanto para matemáticos como para informáticos, para el programa de doctorado del citado departamento en la Universidad de Sevilla.

Su contenido pretende solventar el déficit lógico–matemático de los alumnos de Ingeniería Informática así como las carencias en lógica aplicada a la programación de los alumnos que provienen de la licenciatura de Matemáticas. En el caso que nos ocupa, relacionamos los conceptos lógicos de adecuación y completitud de teorías lógicas con los correspondientes sobre sistemas de verificación de programas.

1 Introducción

La verificación formal de programas persigue el objetivo de fundamentar la práctica de programar, del mismo modo que la Lógica Matemática fundamenta los procesos deductivos en Matemáticas. Desde este punto de vista, es tarea tanto de matemáticos como de informáticos que esta fundamentación alcance sus objetivos: Justificar la adecuación de ciertos métodos de programar y, relajando la formalización, obtener una *metodología* útil –y correcta– para diseñar y verificar sistemas.

En esta comunicación describimos el contenido de los cursos sobre este tema que impartimos en el Dpto. de Ciencias de la Computación e Inteligencia Aritificial, justificando así el interés del tema como punto de encuentro de alumnos con formación bien distinta.

2 Paradigmas en CCIA

Como se afirma en [7], las Ciencias de la Computación y la Inteligencia Artificial (CCIA), al ser una ciencia relativamente nueva, se ha beneficiado del desarrollo de las restantes disciplinas, y no sólo en el conocimiento que éstas aportan. Concretamente, existen tres paradigmas básicos bajo los cuales se desarrolla toda la actividad en CCIA:

- *Teoría.* Es de origen matemático. El desarrollo clásico consiste en caraterizar los objetos de estudio, emitir hipótesis sobre éstos, demostrar las verdaderas, e interpretar los resultados.
- *Abstracción.* Es básicamente, el método científico: conjetura de hipótesis y construcción del modelo. Se hacen predicciones sobre éste, que deben ser puestas a prueba en experimientos, cuyos resultados deben ser analizados.
- *Diseño.* Es la Ingeniería. Los pasos a seguir son: estudio de los estados, su especificación, el diseño e implementación del sistema, que posteriormente debe ser verificado.

*Parcialmente financiado por el proyecto de innovación: *Un programa de adaptación curricular en el área de Ciencias de la Computación e Inteligencia Artitificial* (ICE de la Universidad de Sevilla), y por el proyecto DGES PB96–1345 del Ministerio de Educación y Cultura.

†Parcialmente financiado por el proyecto DGES PB96–1345 del Ministerio de Educación y Cultura.

En general, es difícil articular estos paradigmas de manera adecuada en un currículum, por diversas razones, siendo la principal la *histórica* división de estos métodos de investigación/desarrollo en itinerarios curriculares claramente separados. A esta dificultad son especialmente sensibles las áreas de conocimiento con vocación multidisciplinar, como CCIA. Sin embargo, sí es posible conjugar en cursos de postgrado las diversas actitudes, eligiendo temas adecuados para ello. Uno de éstos es la fundamentación de la programación: la verificación formal de programas.

3 Lógica Matemática y CCIA

La Lógica, como disciplina matemática, tiene su origen en la necesidad de fundamentar, desde la misma Matemática, los procesos de razonamiento que se usan en ésta. En general, se admite que un conjunto de conocimientos alcanza un status de ciencia cuando se formaliza su contenido, para evitar paradojas y ambigüedades, y se establece un corpus consistente con éstos. En este sentido, la Lógica Matemática surge como la herramienta necesaria para matematizar la ciencia que tiene como objeto de estudio los propios métodos matemáticos.

Esta primera orientación –estudio de fundamentos– es ampliada relativamente pronto[1]. Como ocurre cada vez que se establece una nueva línea de investigación, los métodos que se diseñan son utilizados en otros campos científicos.

En nuestro caso, la Lógica Matemática también proporcionó –y proporciona– una metodología para fundamentar y especificar los conocimientos en CCIA. En cierta medida, es responsable del carácter *científico* de ésta[2]. Podemos afirmar que los primeros resultados importantes de esta relación son los teoremas de incompletitud de K. Gödel y la demostración de la irresolubilidad mecánica (algorítmica) de ciertos problemas, debida a A. Church y A. Turing [13][3].

De hecho, el cálculo lógico es, en esencia, un proceso mecánico susceptible, al menos teóricamente, de ser ejecutado en una máquina. Este tipo de implementaciones están limitadas (y amparadas) por los teoremas de completitud, de incompletitud, de indecidibilidad y de consistencia[4].

Algunas de las propiedades fundamentales del uso de la Lógica Matemática en CCIA son [8]:

- El uso declarativo de la Lógica Matemática permite obtener de manera natural tanto la representación de conocimientos como razonar sobre éstos. Es una herramienta útil para construir y manejar bases de conocimiento.
- Es flexible, ya que una misma construcción formal admite más de una aplicación.
- Tiene un carácter introspectivo; una vez representado el conocimiento, la máquina puede preguntarse acerca de lo que conoce. Y puede responder a esas cuestiones usando métodos formales.

También deberíamos añadir como característica importante el carácter riguroso de las deducciones: existen técnicas para asegurar la *corrección* de los cálculos lógicos.

4 Un ejemplo importante: la verificación de programas

Un ejemplo muy importante, que ilustra las relaciones entre la Lógica Matemática y CCIA es su uso en los fundamentos de la programación.

Si bien lo que en la actualidad entendemos por CCIA se ha beneficiado directa e indirectamente de la lógica, también podemos considerar ésta como una herramienta de desarrollo. El ejemplo

[1] Podemos fechar el origen de la Lógica Matemática en los trabajos de A. Morgan, G. Boole y G. Frege (segunda mitad del siglo XIX). Posteriores estudios sobre conjuntos y lógica amplían el contenido de ésta para abarcar –teóricamente– toda las Matemáticas como objeto de estudio (primera mitad del siglo XX). Los estudios de K. Gödel y A. Turing amplían aún más el estudio, considerando los métodos mecánicos como nuevos objetos, dada la fuerte relación de éstos con el razonamiento formal.

[2] La Lógica Matemática es muy útil en los paradigmas *teoría* y *abstracción*.

[3] No podemos dejar de citar otro resultado importante de este tipo: el resultado de Cook (1971) sobre la NP-completitud del problema SAT, de la satisfactibilidad en lógica proposicional.

[4] Además, si deseamos que sea útil, debemos diseñar cálculos *eficientes*; es decir, tales que la cantidad de recursos que se necesitan en estos cálculos se considere como *razonable*.

más claro de este tipo de estudios es la traslación del estudio de las relaciones entre la sintaxis y la semántica en Lógica Matemática al caso de lenguajes de programación, que al fin y al cabo, pueden ser considerados como un tipo especial de lenguajes formales. La metodología es, en esencia, la misma. Sólo faltaba la traducción del cálculo lógico, para demostrar teoremas, a la generación y *demostración* de programas. El trabajo de C.A.R. Hoare [10] se puede considerar como uno de los orígenes de este tipo de estudios, junto con los trabajos de R.W. Floyd y el de S. Cook [6][5]. El espíritu de esta metodología se puede describir en los siguientes puntos, que resumen el espíritu del trabajo de C.A.R. Hoare [11]:

- *Las computadoras son máquinas matemáticas.*
- *Un lenguaje de programación es una teoría matemática.*
- *Los programas son expresiones matemáticas.*
- *La programación es una actividad matemática.*

5 Fundamentos de la verificación de programas

Desde el punto de vista de la fundamentación de la programación, como punto de encuentro de investigadores en Lógica Matemática y/o Computación, el tema de la verificación formal es un interesante punto de encuentro. A grandes rasgos, los cursos sobre este tema deben incluir:

- Lógica Matemática: Lógica de primer orden (LPO). Sintaxis y Semántica.
- Lenguajes de programación: Sintaxis y semántica

Los alumnos a los que va dirigido los cursos sobre verificación son licenciados en Matemáticas e Informática que desean incorporarse a los estudios de doctorado del Dpto. de CCIA de la Universidad de Sevilla. Debido a la clara diversidad curricular, es necesaria, en una primera etapa, la adaptación. Los módulos que integran una orientación específica hacia este tema, englobados en el curso de postgrado *Lógica de Programas*[6] y en el curso de doctorado *Fundamentos para la verificación de programas*[7] pretenden acomodar los conocimientos en teoría y ciencia de la programación de distintos curriculum, como son los de licenciados en Matemáticas y en Informática. Concretamente:

- *Teoría de conjuntos:*
  - Axiomática: conjuntos, clases, aplicaciones...
  - Ordenes. Buenos órdenes. Ordenes bien fundamentados...
- *Módulo Lógico-Matemático:*
  - Sintaxis de la LPO: términos, fórmulas, sustituciones...
  - Teorías de primer orden. Concepto de prueba.
  - Semántica de la LPO: Estructuras, modelos,...
  - La aritmética: El lenguaje de la aritmética, propiedades de la estructura $\mathbb{N}$ de los números naturales.
- *Módulo de Programación:*
  - El lenguaje de programación *While*.
  - Nociones sobre computabilidad (en la línea de [5]): La corrección parcial y el problema de la parada.
  - Semántica de programas: Interpretación.
- *Sistemas de Floyd–Hoare:*
  - Especificaciones: Corrección de programas.
  - Adecuación del sistema de Floyd–Hoare.
  - Automatización: condiciones de verificación.
  - Corrección total.

[5] Habría que añadir el estudio de la adecuación y completitud de estos sistemas para diferentes lenguajes.

[6] Pertenece al programa de formación complementaria en CCIA de la Universidad de Sevilla. http://www-cs.us.es/docencia

[7] Curso perteneciente al programa de doctorado *Lógica, Computación e Inteligencia Artificial.*

5.1 Teoría de conjuntos

La teoría de conjuntos representa un formalismo muy útil para describir, de manera formalizada y sin ambigüedades, los elementos de la verificación de programas. De hecho, permite especificar adecudadamente los objetos de estudio.

Uno de los enlaces más interesantes entre la teoría de conjuntos y la verificación de programas es la aplicabilidad de los órdenes bien fundamentados para las pruebas de corrección total (en concreto, para el test de parada).

Por otro lado, se analiza un principio muy importante: el principio de inducción sobre clases bien fundamentadas. El principio de inducción es, por otro lado, una herramienta muy utilizada en otros campos de CCIA (por ejemplo, en Algorítmica [3]), aunque no en una forma tan general.

5.2 Elementos de Lógica Matemática

En este módulo se repasan los conceptos básicos de Lógica Matemática. Se presenta a la Lógica Matemática como una forma general de representación del conocimiento matemático, susceptible de ser aplicada a la descripción formal de propiedades acerca de otros objetos. Se estudia la relación existente entre demostrabilidad y validez. Por último, nos centramos en un caso importante, el de la Aritmética, describiendo una teoría aritmética, la *aritmética de Peano*, que se puede considerar como la teoría base en CCIA, cuando se necesita utilizar un referente lógico para clasificar y estimar la potencia de resultados en CCIA. Sus axiomas son, por un lado, un conjunto finito de propiedades acerca de la estructura discreta de la estructura N junto con el esquema de axiomas de inducción, que representa en primer orden el principio de inducción.

La relación de la aritmética de Peano y sus subsistemas, con la computabilidad se pone de manifiesto cuando se estudia la representabilidad de funciones computables en este sistema.

5.3 Lenguajes de programación

Los lenguajes de programación pueden ser considerados, siguiendo la idea de de C.A.R. Hoare, como lenguajes lógicos, y dotados de una sintaxis y una semática bien formalizada. Lo que nos permite especificar los estados, las propiedades sobre éstos, y asociar a cada programa una función parcial sobre los estados, lo que nos permitirá traducir los problemas de corrección a problemas sobre funciones.

De hecho, es interesante poner de manifiesto la analogía entre la Lógica Matemática y los lenguajes de programación.

5.4 Sistemas de Floyd–Hoare

Al igual que las demostraciones en en Matemáticas, las pruebas de corrección se pueden fundamentar mediante un cálculo lógico. Por ejemplo, en las pruebas de corrección de los algoritmos es usual utilizar *invariantes*, que se basan en propiedades inductivas de ciertas estructuras.

La analogía con las Matemáticas se puede completar con la exposición de un cálculo formal simple que permita *demostrar programas* (es necesario formalizar el algoritmo mediante un lenguaje de programación). Por problemas de tiempo, nos restringimos a un caso básico: sintaxis restringida, usando sólo variables de tipo entero y sin estructuras de datos. Esta restricción no implica, como se muestra con ejemplos, que no se puedan estudiar algoritmos numéricos no triviales. Nuestro objetivo es doble. Por un lado mostrar al alumno que la práctica de programar se puede fundamentar, como se hace con las Matemáticas. Además, se introduce al alumno, con un sistema sencillo, en los problemas de adecuación y completitud de los cálculos formales.

En primer lugar presentamos el lenguaje de las especificaciones, describiendo, sin una formalización excesiva, la semántica de dichas expresiones. Como en todo cálculo lógico, se introducen a continuación los axiomas y reglas de inferencia, que permiten definir el concepto de *teorema* en la llamada Lógica de Floyd–Hoare[8].

A continuación se habla de su adecuación y completitud. Sabemos que es incompleta, es decir, existen especificaciones verdaderas que no son demostrables (pues este tipo de sistemas están afectados por los teoremas de incompleti-

[8] En nuestro caso, no especificamos los axiomas y reglas de inferencia que nos permiten manipular las pre y postcondiciones. Esto permite centrarnos en el estudio del programa. Formalmente, podemos pensar que se acepta como teoría para las especificaciones todas las propiedades básicas usuales acerca de las funciones numéricas elementales, y como reglas de inferencia las que se utilizan en la Aritmética.

tud y por las restricciones sintácticas de ciertas reglas), y se puede justificar que es adecuada (todo lo que se prueba es cierto).

La dificultad de las pruebas en este sistema (en general poco intuitivas), induce a diseñar otra forma de verificar especificaciones complejas. Con este objetivo se introduce un método *efectivo*[9], del tipo divide y vencerás, para demostrar las especificaciones. El método consiste en[10]:

1. Anotar la especificación: añadir en determinados puntos de la especificación ciertas fórmulas que expresan propiedades acerca del estado en el que se encuentra la ejecución en dicho punto.
2. Obtener, a partir de la especificación anotada, un conjunto de especificaciones simples.
3. Aplicar una serie de reglas que permiten obtener, a partir de estas especificaciones, unas fórmulas puramente aritméticas, que denominaremos *condiciones de verificación.*
4. Probar que las condiciones de verificación generadas son demostrables (en la Aritmética). Si efectivamente es así, entonces afirmamos que la especificación original lo es en el sistema de Floyd–Hoare.

Hay que hacer notar que, de todos estos pasos, el primero y el último no son en general mecanizables; necesitan la ayuda de un experto humano. En todo caso, se pueden dar una serie de pautas a seguir para una adecuada realización de éstos[11].

Hay que tener en cuenta dos características importantes de este método:

- Del hecho de que no todas las condiciones de verificación generadas por una cierta anotación sean demostrables, no se deduce que la especificación original no lo sea, en el sistema de Floyd–Hoare.
- Se demuestra que si todas las condiciones de verificación son demostrables, entonces la especificación original lo es (es decir, probamos la adecuación del método de las condiciones de verificación). Por tanto, el método es *correcto.*

[9] Formalmente no se puede considerar un algoritmo.

[10] Siguiendo la línea de [9].

[11] No obstante, el cuarto pado puede ser automatizado, en ciertos casos, con la ayuda de un demostrador automático de teoremas.

6 Complementos

Durante el desarrollo del curso suelen aparecer cuestiones diversas de interés específico para algunos alumnos, como por ejemplo:

- La relación existente entre el lenguaje de primer orden usado y la posibilidad de expresar en éste cuestiones acerca de los programas.
- El papel que juega la teoría de primer orden asociada a los estados con respecto al sistema de verificación.
- Diseño de lenguajes de programación con adecuadas restricciones.
- Obtención de otras *lógicas de programas* (lógicas temporales, dinámicas, etc.) más adecuadas para estudiar ciertas propiedades de los sistemas.

Estas cuestiones podrían servir para iniciar al alumno en la investigación, y se complementaría con la oportuna bibliografía.

7 Limitaciones de la Lógica Matemática en la verificación de sistemas

Es conveniente, una vez presentado la verificación formal, discutir las limitaciones inherentes a la formalización lógica del tema.

El uso de técnicas de Lógica Matemática en CCIA no posee, en absoluto, un carácter general. Entre las limitaciones más evidentes de su aplicabilidad en este campo se encuentran las siguientes:

- La formalización de algunas de las herramientas utilizadas en CCIA no es posible, es muy complejo o no proporciona nada nuevo. Existen otras formas de representación más eficientes que la Lógica Matemática para este tipo de problemas.
- El uso de la Lógica en sistemas basados en conocimiento permite representar conocimientos, pero debe ser complementada para estudiar cuestiones como semántica, validación, verificación, etc. Los métodos de deducción en los sistemas basados

en conocimientos combinan el cálculo lógico con otros algoritmos de razonamiento diseñados *ad hoc*.

- El manejo de fórmulas es, en general, un trabajo de enorme complejidad algorítmica. De hecho, se pueden plantear problemas no resolubles algorítmicamente (la consistencia de fórmulas de primer orden), o resolubles mecánicamente pero intratables (el problema de las satisfactibilidad en lógica proposicional). Estas limitaciones se pueden evitar si se usan fórmulas con *baja* complejidad sintáctica, y se adaptan los algoritmos para éstas.

Debemos considerar que, en algunos casos, la Lógica Matemática proporciona un rigor que, si bien es excesivo a efectos prácticos (o para problemas de gran tamaño), nos permite desarrollar –al relajar adecuadamente la formalización– una metodología muy útil para construir sistemas correctos[12].

8 Conclusiones

La experiencia docente nos dice que este tipo de cursos son necesarios, por tratarse de cursos cuyo contenido matemático sirve como contrapartida para la ingeniería, y cuyo contenido práctico complementa la *ciencia* de las matemáticas. Es indudable que esta adaptación es aconsejable para desarrollar un tema de investigación conducente a elaborar una tesis en CCIA.

La formación matemática en este campo puede ser considerada como la intención de proporcionar una sólida base formal en aquellos aspectos de las ciencias básicas (en este caso, Matemáticas) íntimamente relacionados con la Informática, para desarrollar en cursos posteriores, con flexibilidad, las técnicas necesarias para el quehacer profesional[13].

[12] Claro está que cualquier relajación no asegura la corrección formal del sistema, desde el punto de vista lógico. Véase en [4] la discusión sobre este problema. Sin embargo, un uso *semiformal* (en mayor o menor medida) de los métodos de prueba permite exponer una metodología de trabajo –incluso docente– aconsejable [3], [12].

[13] Este argumento es el defendido para la Ingeniería Informática en [1].

Referencias

[1] ALONSO AMO, F. ET AL.: *Consideraciones para el diseño de un curriulum en ingeniería informática*, en Actas de las Jornadas sobre Formación en Informática Superior para los Noventa. (Sáez Vacas, F. y González Cristóbal, J.C., eds.) Servicio de publicaciones de la E.T.S.I. Telecomunicación de la U.P.M. (1991), 105–130.

[2] APT, K.R.; OLDEROG, E.R.: Verification of Sequential and Concurrent Programs. Springer–Verlag (1991).

[3] BALBONTÍN NOVAL D.; BORREGO DÍAZ, J.; PÉREZ JIMÉNEZ M.J.: *Fundamentos de Algorítmica: Algunas cuestiones metodológicas.* en II Jornadas nacionales de innovación en las enseñanzas de las ingenierías (1996) 400–405.

[4] BARWISE, J.: *Mathematical proofs of computer system correctness.* Notices of the Amer. Math. Soc. 36 (7) 844–851 (1989).

[5] BORREGO DÍAZ, J.: *Algoritmos y Computabilidad* en Lógica Formal: Orígenes, Métodos y Aplicaciones. (Nepomuceno, A., ed.) Ed. Kronos (1995), 117–147.

[6] COOK, S.A.: *Soundness and completeness of an axiom system for program verification.* SIAM Journal of Computing 7 (1) 70–90 (1978).

[7] DENNING, P.J. ET AL.: *Computing as discipline.* Communications of ACM 32 (1) 9–23 (1989).

[8] GENESERETH, M.R.; NILSSON, N.J.: Logical Foundations of Artificial Intelligence. Morgan Kaufmann Pub. Inc. (1987).

[9] GORDON, M.J.: Programming Language Theory and its implementation. Prentice–Hall International (1988).

[10] HOARE, C.A.R.: *An axiomatic basis for computer programming.* Comm. of ACM 12, 576–580 (1969).

[11] HOARE, C.A.R.; JONES, C.B: (ED.): Essays in Computer Science. Prentice Hall (1989).

[12] DE MARNEFFE, P.A.: *Une méthode semi-formelle pour expliquer les algorithmes* Tech. Sci. Inf. 14 (10) 1323–1331 (1995).

[13] TURING, A.: *On computable numbers with an application to the Entscheidungsproblem.* Proc. London Math. Soc. 42 230–265 (1936).