Doble Grado Ingeniera Informtica - Ciencias Matemticas, Vol. xx (xxxx), Núm. x, Págs. 1–181

Teoría de la Programación: ¿Matemática Fundamental o Aplicada?

por

David de Frutos Escrig

Me he animado a escribir este breve ensayo con el ánimo de situar lo mejor posible a mis alumnos del *Doble Grado en Ingeniería Informática y Ciencias Matemáticas* a la hora de estudiar, y superar con el mayor éxito posible, la asignatura de *Teoría de la Programación*, que se encuentran de modo obligatorio en su Cuarto curso.

Al efecto, comenzaré explicando el principal objetivo que se persigue de manera específica, al haber incluído como obligatoria esta materia en su curriculum. La misma forma parte de hecho de la intersección entre los curricula de los dos grados que se combinan, y ello no sólo ha de verse a efectos del consiguiente "doble valor" de la misma, sino fundamentalmente al carácter complementario que se perseguía al instaurar el Doble grado. En concreto, se buscaba la simbiosis entre Matemáticas e Informática para mejor aprender, y estar capacitado para utilizar en el futuro, aquellas materias que se benefician de modo natural de lo que aportan por separado los dos grados combinados.

Muy probablemente, la Teoría de la Programación ocupa un lugar muy especial en ese capítulo de asignaturas que de manera natural ocupan un lugar en la intersección entre estas dos disciplinas cientifícas: Informática y Matemáticas. Partiendo en consecuencia de esa idea de colaboración entre dos disciplinas, dejaré de lado la discusión filosófica sobre si las bases de la Informática en sus diversos aspectos no tecnológicos se deberían considerar directamente Matemáticas, manteniendo una separación terminológica entre vocablos clásicamente informáticos y matemáticos, sin que ello suponga en absoluto que mi opinión personal vaya en la dirección de separar (conceptualmente) las Matemáticas y la Informática (Fundamental).

Teoría de la Programación en sentido amplio y como Semántica de Lenguajes de Programación

Pero es hora de concentrarnos en los objetivos técnicos de la asignatura. Para empezar hay que dejar claro que la enorme amplitud que el nombre de la asignatura en principio sugiere queda muy lejos de la habitual especificidad que tiene, no sólo la presente, sino buena parte de las que en el mundo se ofrecen más o menos con esta denominación. Si hubiésemos optado por una aproximación dual a la hora de fijar su denominación, no importándonos pecar por demasiado precisos en vez de por lo contrario, un nombre mucho más preciso sería el de *Semántica (Formal)* de los *Lenguajes de Programación*. Multitud de los conceptos, e incluso disciplinas completas,

que deberían aparecer en un compendio de la Teoría de la Programación, quedan en consecuencia fuera, aunque eso no quiere decir en absoluto que no tengan nada que ver, sino que o bien juegan un papel complementario y de hecho son cubiertas, al menos en parte, en otras materias del curriculum, o bien precisan del estudio preciso previo de la Semántica para contar con los fundamentos que hagan posible su desarrollo posterior con el necesario rigor. Enumero a continuación aquellos de esos conceptos o disciplinas que me parece inevitable citar aquí:

1. **Metodología de la Programación y Algorítmica**. Asumiendo la identificación conceptual entre *Algoritmo* y *Programa*, estas disciplinas nos muestran cómo ha de combinarse el uso de las *Estructuras de Datos* y la *Programación Estructurada*, en un sentido amplio, para ser capaces de desarrollar soluciones computacionales (i.e. programas) que resuelvan correctamente los problemas (resolubles) que se nos puedan presentar. El soporte lógico-matemático de tales metodologías nos permite poder garantizar a priori la corrección de tales programas, así como su utilidad en la práctica en términos de coste, concretada en el conocimiento de los recursos (*Memoria* y *Tiempo*) que supondrá su ejecución. En principio, la Metodología de la Programación se plantea en términos abstractos, de manera que las soluciones que nos facilite sean trasladables a cualquier lenguaje de programación. Pero al mismo tiempo estas metodologías aparecen (o deberían al menos hacerlo) en toda asignatura relacionada con la Programación, entendida en el sentido de resolver problemas concretos con programas concretos. Naturalmente, en la medida en que nuestras soluciones lleguen a ser programas escritos en un determinado Lenguaje de Programación, necesitaríamos el conocimiento preciso de su *Semántica* para poder fundamentar aseveraciones de corrección, pues si no sabemos "lo que va a hacer exactamente" nuestro programa, difícilmente podremos asegurar que va a hacer "lo que queremos". Sin embargo, es absolutamente obvio que cuando uno "aprende" un lenguaje de programación, se supone que capta a un nivel informal, pero suficientemente preciso, lo que "debería hacer" cada instrucción del mismo, y por ende cada programa, de manera que asumiendo que el lenguaje funciona en efecto así, uno puede en efecto *probar* que los programas hacen lo que deseamos (asumiendo naturalmente que los hemos hecho bien, incluyendo en esto tanto el fondo (son en efecto correctos), como la forma (han seguido las citadas metodologías)).

   Pero si tras el estudio de una Teoría de la Programación contamos en efecto con la definición precisa de la Semántica del lenguaje de programación que estamos manejando, podremos prescindir de la asunción incorporada antes a nuestro razonamiento, con lo que la corrección quedaría ahora probada en términos absolutos, sin tener que asumir nada.

2. **Teoría de la Computabilidad**. Al aparecer los primeros ordenadores, y con ellos los programas que resuelven problemas más y más complejos en un tiempo más que razonable, la borrachera de éxito llevó a pensar que tras el necesario desarrollo tecnológico que hiciera los ordenadores más y más potentes en términos de velocidad, los mismos servirían para resolver cualquier

problema, al menos cualquiera con una mínima aplicabilidad práctica. La cura de humildad llegó, curiosamente, de la mano de quienes buscaban probar la *Universalidad*, no de la Informática, sino de las Matemáticas. Los *Problemas Indecidibles*, los *Teoremas de Incompletitud de Gödel*, y las limitaciones de las *Funciones Computables*, van a señalarnos que determinados problemas jamás podrán ser resueltos (al menos en su generalidad) por ningún programa que trate todos sus casos a la vez. Establecida esta limitación, surge la Teoría de la Computabilidad, para estudiar desde distintas ópticas esa *Clase de Problemas* que sí pueden ser resueltos, los cuáles se corresponden con el concepto de *Función Computable (o Recursiva)*. Nos limitaremos a citar al respecto el resultado (parcialmente informal) que consituye la llamada *Tesis de Church*, que a partir de múltiples *Caracterizaciones* de las funciones computables, que fueron probadas efectivamente equivalentes, nos lleva a concluir que podemos dar por bueno (¡mientras no se demuestre lo contrario!) que las funciones computables son en efecto aquéllas cuyo cálculo puede realizarse con un *método automático cualquiera*. Una de esas caracterizaciones la consituyen las *Máquinas de Turing*, que no pueden ser más simples, indicándonos por tanto que podemos radicar la fuerza de la computación en los pequeños cambios en la memoria (asignación) y las disquisiciones condicionales posiblemente repetidas (condicional y bucles) sobre el orden en que hemos de ejecutar las instrucciones que conforman un programa (imperativo iterativo). Al tiempo, la presentación equivalente en términos de las *Funciones Recursivas*, nos indica que alternativamente podríamos ulilizar la *Recursión* en vez de la *Iteración*, como de hecho solemos hacer en la práctica cuando programamos con *Procedimientos Recursivos* o directamente *Lenguajes Funcionales*, siendo incluso posible combinar iteración y recursión, sin que por ello podamos aspirar a aumentar la expresividad de nuestros lenguajes y programas,

Nuestra brevísima introducción anterior a la Teoría de la Computabilidad ha puesto de manifiesto los elementos con los que la misma interviene a la hora de desarrollar la Teoría de la Programación: de una forma u otra, iteración y/o recursión estarán en la base de todo Lenguaje de Programación, y por tanto su funcionamiento preciso en ellos será elemento esencial a la hora de definir rigurosamente sus Semánticas.

3. **Teoría de Autómatas y Compilación de Lenguajes de Programación**. Todo lenguaje, *natural* o *fomal*, viene definido por medio de unos determinados entes básicos (átomos, palabras, sílabas, ...) que se combinan de una determinada forma (estructura, reglas gramáticales), para dar lugar a los elementos válidos del lenguaje (palabras del lenguaje formal, frases del lenguaje natural). Pero esto sólo constituye la sintáxis del lenguaje, la cual termina definiendo los discursos que podemos hacer utilizando el lenguaje, separándolos de aquellos otros inadmisibles en el mismo, bien porque utilizan átomos fuera de los propios, o porque los combinan de manera no permitida.

   En ocasiones un lenguaje formal definido meramente en tales términos sintácticos resulta interesante per se. Algunos ejemplos interesantes de ello podrían

ser: la representación de enteros en base diez, o las *expresiones aritméticas*, que combinan valores (enteros, por ejemplo), por medio de las operaciones usuales. En estos casos la utilidad del lenguaje es la de representar (biyectivamente) los elementos de un determinado dominio abstracto (p.e. los números enteros) por medio de determinadas secuencias no vacías de átomos (las cifras y el signo $-$).

Pero cuando los lenguajes adquieren verdadero interés es cuando, además, cuentan con una Semántica que permitirá asignar un significado no trivial a las frases válidas. El mismo obviamente será función de su sintáxis, pero habitualmente no de un modo inyectivo, pudiéndose tener muchas frases válidas con un mismo significado. Por ejemplo, cuando escribimos $(3+2)*(5-1) = 20$ estamos *calculando* el *significado* de la expresión (válida) a la izquierda, que es el valor indicado, resultado de efectuar ordenadamente las operaciones que la conforman. De este modo, cuando escribimos $(3+2)*(5-1) = 4*5$ estamos diciendo que las dos expresiones valen lo mismo (20), y no que $(3+2) = 4$ y que $(5-1) = 5$, lo que obviamente no sería cierto.

La Semántica va a definir dichos significados, y en nuestro caso lo hará incidiendo en la estructura sintáctica de las frases válidas apoyándose en el *Principio de Composicionalidad*, que nos dice que el significado no cambia si sustituimos en una frase una subfrase por otra con el mismo significado. En el ejemplo de las expresiones, $(3+2)*(5-1) = (7-2)*(2*2)$, ya que $3+2 = 7-2$ y $5-1 = 2*2$, lo cual podría verse como corolario del hecho de que el cálculo de ambas expresiones pasaría por la expresión intermedia $5*4$.

A lo largo del curso esta *Composicionalidad* aparecerá una y otra vez, y en ocasiones será todo lo que necesitemos para definir la Semántica, junto con las reglas que capturen el significado de cada operación que aparezca en la definición de la estructura (sintáctica) del lenguaje.

La Compilación, si la centramos en la *Traducción* del *Lenguaje Objeto* al correspondiente *Lenguaje Máquina* de la Máquina concreta sobre la que pretendemos correr (ejecutar) los programas compilados, es de hecho una Semántica Formal (Concreta) que consigue definir el significado efectivo de los programas compilados, cuando se combina a su vez (compone, en términos de composición de funciones matemáticas) con el Funcionamiento Físico de la Máquina, que estaría definiendo la Semántica del Lenguaje Máquina, por medio de los Procesos de Cómputo a los que da lugar.

Si volvemos al punto en que señalamos que en la práctica podemos razonar sobre el funcionamiento de nuestros programas, asumiendo que conocemos el funcionamiento de sus instrucciones a un nivel informal, cuando ejecutamos esos programas tras compilarlos sobre una determinada máquina, todo lo que hemos concluido se dará en la práctica en tanto y cuando la *Implementación* del lenguaje definida por el *Compilador* utilizado sea consistente con el citado conocimiento abstracto informal que utilizamos en nuestros razonamientos.

En consecuencia, es absolutamente cierto que la Semántica inducida por el Compilador es una Semántica Formal, que está definiendo el significado de los

programas cuando se compilen de esa forma, pero en los detalles de la aseveración anterior encontramos también las (buenas) razones para no considerarla en absoluto La Definición Semántica del lenguaje, dado que:

a) No tendríamos Una semántica, sino una por cada Compilador (y Máquina sobre la que se ejecute el código máquina generado), sin la posibilidad además de poder compararlas, para poder afirmar (justificadamente) que cada programa del lenguaje hace lo mismo en todos los casos (lo que representa el concepto de Portabilidad de Código). Históricamente éste ha sido un problema radical para quienes pretendían que sus programas escritos, por ejemplo en Fortran, tuvieran *un* (buen) comportamiento (idéntico), se compilaran y ejecutaran en una u otra máquina.

b) Incluso, aunque "por definición" tomaramos un determinado Compilador como La Definición Formal del lenguaje, se trataría de una definición de tan "bajo nivel", que resultaría absolutamente imposible utilizarla en razonamientos no triviales.

En consecuencia, la Teoría de la Programación busca definir marcos adecuados para definir la Semántica de un modo **preciso**, pero lo más **sencillo** y **manejable** posible. De esa manera podremos hacer razonamientos sobre programas arbitrariamente complejos, limitandonos a repetir razonamientos sencillos sobre las componentes básicas de los mismos, en base a reglas asociadas a las operaciones sintácticas del lenguaje. De este modo la Composicionalidad nos facilitará "de gratis" el significado global y comportamiento de los programas grandes, simplemente Componiendo de la forma oportuna los significados de sus componentes.

4. Lenguajes de Programación. Cuando aparece este descriptor en un curriculum de Informática suele corresponder a un Estudio Descriptivo de distintos lenguajes más o menos utilizados en la práctica, incidiendo también en los distintos *Paradigmas* en que se encuadran. A este respecto destacan el paradigma Imperativo, el Funcional, el Lógico, la Orientación a Objetos, ... Y el recorrido de las características de los distintos lenguajes puede hacerse enumerandolos uno tras otro, o de una manera más abstracta, enumerando las distintas característcas (estructuras de datos, condicionales, bucles, procedimientos, ...) y las múltiples variantes con que podemos cubrir cada una de ellas, destacando las (habitualmente) sutiles (¡pero por ello mismo, importantes de detectar!) diferencias entre ellas, e ilustrando las eventuales ventajas e inconvenientes que se siguen de ellas.

   De nuevo esto no es lo que haremos en esta asignatura, por diversas razones:

   a) En general, todos los lenguajes reales son demasiado grandes e incluyen características muy sutiles que buscan ofrecer al usuario una multiplicidad de alternativas (técnicamente redundantes) de manera que pueda elegir entre ellas, bien buscando una mayor simplicidad y/o eficiencia en cada caso concreto; o bien, simplemente en base a los gustos o hábitos personales, a los que esté acostumbrado.

b) La definición de la Semántica Abstracta, tratando de combinar simplicidad y naturalidad, resulta ser una tarea bastante más compleja de lo que se podría esperar. De resultas de lo cual, no sería en absoluto adecuado condenarnos a no entender nada, por culpa de tener que presentar las definiciones de muchas construcciones (con lo cual la definición ya resultaría muy compleja, simplemente en base a su tamaño) que además localmente también serían complicadas, al tenerse que capturar esas sutiles diferencias, que inexorablemente serán difíciles de precisar.

c) A pesar de que, como dijimos antes, la Iteración y/o la Recursión estarán en la base de cualquier lenguaje que permita expresar todas las Funciones Computables, la forma en que uno y otro mecanismo aparecen en los lenguajes correspondientes a cada uno de los paradigmas termina siendo absolutamente dispar, por lo que si intentaramos ilustrarlos todos terminaríamos enfrentados a una serie de definiciones aparentemente muy diferentes. De esta forma, sería muy difícil aislar y manejar adecuadamente los (pocos) principios básicos en los que se fundamenta la definición de las Semánticas Formales de los Lenguajes de Programación.

En consecuencia, a lo largo de la asignatura nos centraremos sobre todo en la Semántica de Lenguajes Imperativos, aunque al final brevemente se tratará el tema de los Lenguajes Funcionales, que de hecho son los que en principio manejan conceptos más cercanos a los que aparecen en los Dominios que aparecen en las definiciones de las semánticas, que terminan siendo siempre Funciones, más o menos Abstractas.

Teoría de la Programación como ejemplo de Matemática Aplicada

Aunque en principio el título de Matemáticas correspondiente a este Doble Grado se corresponde con uno de Matemáticas Generales (Fundamentales o Puras, como se las llamaba antes), entendemos que todo título de Matemáticas persigue la capacitación a los alumnos que lo cursan para su Aplicación posterior a la resolución de todo tipo de problemas, para lo cual deberían ser capaces de *Modelizar* adecuadamente esos problemas en términos puramente Matemáticos, vía la correspondiente *Abstracción*.

Veremos que la Teoría de la Programación cae absolutamente en este terreno, dado que:

1. El universo del mundo real que se ha de modelizar no cuenta a priori con una definición formal de partida. Si bien esto podría considerarse falso cuando nos ponemos como referencia Lenguajes previamente conocidos, para los que se supone una semántica informal conocida, soportada además por Implementaciones con las que se puede consolidar dicho conocimiento vía la *Experimentación*. La misma vendría dada simplemente por la escritura de programas que pretendemos resuelvan determinados problemas concretos, cuya "corrección"

puede constatarse sobre casos concretos, simplemente ejecutando el programa con los correspondientes datos (de entrada).

2. De una u otra manera vamos a poder intuir cómo deben ser los objetos matemáticos por medio de los cuáles definiremos la Semántica (valores del *Dominio Semántico*) y la forma en que debemos Operar dichos valores (Operaciones sobre el Dominio) junto con las propiedades de éstas que nos permitirán precisar la definición de la Semántica.

Cuando realizamos la Modelización de un Problema Físico, el dominio (natural) en principio sería el de los *Reales*, y las Funciones con "buenas propiedades" definidas sobre las mismas (continuas, derivables, integrables, ...), lo cual nos suele exigir de inmediato *generalizar* los reales en los *Complejos*. Y naturalmente, el estudio de ciertas propiedades exigirá un razonamiento de *Orden Superior*, manejándose *Espacios Funcionales*, y adentrándonos por tanto en el (más abstracto) *Análisis Funcional*. En consecuencia, pueden llegar a ser necesarios objetos muy sofisticados, con una Teoría muy compleja sobre ellos. Ahora bien, la Matemática (Pura) se ha venido adelantando a esas necesidades desarrollando teorías con aplicabilidad a priori desconocida, con la "justificación" de que en muchos casos terminan por necesitarse más tarde teorías más o menos similares a algunas de las previamente desarrolladas, con la concuencia de que sus (buenas) propiedades (o al menos parte de ellas) ya se conocerán.

Pero por Matemática Aplicada deberíamos entender la aplicación en el sentido arriba expuesto de cualquier Teoría Matemática, no necesariamente imbricada con el Análisis y las Ecuaciones Diferenciales (como una malhadada deformación del uso del lenguaje ha venido a hacernos creer, al menos en la Universidad Española), que nos permita realizar la modelización de un problema del mundo real. Por ejemplo, las Estructuras Algebraicas serán de utilidad allá donde se adivine la presencia de un *Grupo*, un *Anillo*, etc. ; o los Modelos Probabilistas, allá donde se detecte un comportamiento aparentemente "aleatorio".

Como hemos sugerido antes, las Funciones van a seguir siendo nuestra herramienta principal para la definición de las Semánticas, pero sin embargo en nuestro caso el Dominio inicial de partida no puede ser para nada Continuo, pues la unidad natural de información que maneja todo ordenador es la *Palabra* (de determinado número (t) de bytes, cada uno con 8 bits), de manera que manejamos los valores de un conjunto con a lo sumo 2^{8t} elementos. Por tanto, Computar en un ordenador (convencional) es algo (Localmente) *Discreto*. En consecuencia, en un principio, toda la Matemática Continua (al menos la Clásica) parece de poca o nula aplicabilidad en este caso. Esto efectivamente va a ser así, en el sentido de que los objetos básicos que vamos a manejar son en efecto discretos. De hecho, los números *Enteros* constituirán habitualmente nuestro dominio básico, abstrayendo el concepto de palabra, "olvidándonos" de su longitud fija (y por tanto limitada), pero haciendo énfasis en que en todo caso será *Finita*. Ssin embargo, a la postre la *Continuidad* (en un determinado sentido abstracto) va a ser absolutamente capital para conseguir las necesarias buenas propiedades de nuestros Dominios Semánticos. No es éste el momento de detallar el significado preciso de la continuidad aquí, pero sí para

avanzaros, como estudiantes de Matemáticas, que si bien se trata de continuidades "no habituales", no son en cambio conceptualmente novedosas en absoluto, pues se corresponden con la *Continuidad Topológica* en un determinado *Espacio Topológico*, por "raro" que éste pueda llegar a ser.

Y si bien no precisaré aquí dicha noción de continuidad, sí que justificaré la conexión imponderable entre continuidad y computabilidad, cuando utilizamos un ordenador para "computar" Funciones sobre los Reales, lo que evidentemente hacemos constantemente, en particular cuando los utilizamos en la modelización de Problemas Físicos. Si $r \in R$, y queremos calcular $f(r)$, tal cosa es sencillamente **imposible** si f no es **continua en** r. ¿Por qué? Para empezar, eso de que "conocemos" (exactamente) r no tiene sentido: en el mejor de los casos podremos aspirar a "conocer" aproximaciones arbitrariamente buenas, por ejemplo, las k primeras cifras decimales, con k tan grande como nos vaya haciendo falta. En tal caso, para calcular $f(r)$ tendremos que ir calculando las "aproximaciones" $f(r_k)$, a cuyos valores reales les sucede exactamente lo mismo, de modo que lo que iremos teniendo son aproximaciones $f(r_k)_l$. Eventualmente, nos podremos "fiar" de alguna de ellas para dar por buenas las primeras m cifras decimales de $f(r)$. Pero evidentemente nada de esto tendría sentido si f no fuera Continua, al menos localmente en el punto r.

En consecuencia, hemos de estar abiertos para combinar conceptos Algebraicos Discretos, como los que definen la Sintáxis Estructurada de los Lenguajes, con razonamientos Continuos, que se generarán cuando se visualice la Semántica de los Programas a nivel *Global*: en vez de definir y razonar sobre las ejecuciones de un Programa concreto sobre unos Datos concretos a nivel *Local*, reflejando la forma en que utilizamos los programas en la práctica, lo haremos a nivel Global, buscando y explotando las relaciones que existen entre dichas ejecuciones cuando variamos (de la forma adecuada), sea el programa o los datos. Con ello aparece el tratamiento a nivel **Funcional**, que permitirá resolver nuestro problema Local como caso particular de su definición Global (sabremos lo que hace cada programa concreto, porque habremos definido (simultáneamente) lo que hacen todos los programas, relacionando lo que hacen unos con lo que hacen otros (posiblemente) más simples en algún sentido).

Y si lo discreto es en principio nuestro campo de trabajo, la herramienta por autonomasia va a ser la **Inducción**. A lo que va a tener que estar abierto el alumno es al hecho de que aunque hablemos en singular de esta herramienta, en realidad hay **muchos tipos de inducción diferentes**, más allá de la inducción simple y la inducción completa, ambas sobre los Naturales, a la que estamos acostumbrados. Habrá que ir viendo con cuidado cuál o cuáles tendremos que utilizar en cada momento, pues no resultarán en absoluto intercambiables entre ellas. Como se irá viendo, la clave para elegir bien estará en descubrir cuál es el elemento discreto que juega un papel protagonista en cada caso, para poner el foco sobre él, generando así el tipo de inducción adecuado.

Pero como ya se ha dicho, la Continuidad terminará jugando también un papel importante, precisamente cuando la idea de definir la Semántica de cada programa, relacionando unas con otras, se lleve al límite. Esas relaciones conformarán un Sistema de Ecuaciones (Infinito), y la forma en que "resolveremos" estos sistemas a "Alto

Nivel" es manejando *Teoremas de Punto Fijo*, ligados inevitablemente a nociones de Continuidad. Al tiempo, cuando sea posible un tratamiento puramente Composicional a nivel Local, podremos considerar el sistema como una Definición Inductiva, limitandonos por tanto a razonar a un nivel puramente Discreto. De modo que por lo general necesitaremos una sabia combinación de técnicas discretas y continuas para definir nuestras semánticas.

E inevitablemente tendremos que razonar con técnicas *Lógicas*, lo que más allá del uso constante que se hace de la Lógica en Matemáticas cuando se realizan *Demostraciones*, aparecerá con un papel más protagonista, primero cuando se utilicen definiciones (en la *Semántica Operacional*) con Reglas Lógicas; y al final de la asignatura, cuando la *Semántica Axiomática* eleve aún más el nivel de Abstracción de la Semántica, al pasar de una descripción "precisa" de la misma a un marco en el que se define de forma indirecta, por medio de Propiedades Lógicas que nos indican determinadas propiedades (Postcondición) que cumplirá la ejecución de un programa sobre cualesquiera datos que cumplan unas determinidas propiedades (Precondición). Las propiedades se expresarán por medio de Fórmulas Lógicas, y por tanto el manejo de éstas jugará un papel vital en la definición y manejo de esta última Semántica.

Y en condiciones puntuales manejaremos otros conceptos matemáticos, guiados por la Regla de Oro de la Matemática Aplicada, que nos anima a utilizar en cada momento todo aquello que conozcamos (a ser posible bien), que veamos que nos puede ser útil para razonar en el marco del Modelo escogido para la Modelización del problema en estudio.

Concluiremos esta sección desmontando la posible asunción (proveniente de nuevo del Medio Contínuo) de que Modelizar realizando una Abstracción supone dar por bueno que los resultados que obtengamos pueden contener (pequeños) errores de Aproximación, y que por tanto resulta admisible un uso "un poco liberal" de las técnicas matemáticas, "confiando" en que cuando ello se hace de manera "experta", los errores que se cometan van a ser en efecto "pequeños", y su efecto por tanto asumible. Esta flexibilización del Principio de Precisión, que se supone que rige toda Matemática Pura podría resultar admisible, y en cierto sentido es incluso ineludible, en un marco Continuo, en el que estamos condenados a trabajar con Aproximaciones. Pero ya hemos dicho que la definición de la Semántica de Lenguajes es algo esencialmente Discreto, y aquí no caben medias tintas: las cosas o están (completamente) bien, o completamente mal. El corolario que obtenemos de esto es que debemos ser absolutamente cuidadosos al realizar nuestros razonamientos, comprobando, sin dar lugar a dudas, que se cumplen las premisas para poder aplicar cada técnica o resultado previo que apliquemos, y haciéndolo con exquisito cuidado, para así evitar cualquier "pequeño fallo", que en el marco discreto en principio devendrá en catastrófico. Esta obligada exactitud nos lleva entonces a lo habitualmente asumido en el terreno de la *Matemática Pura*: los resultados que obtengamos serán (totalmente) correctos o simplemente falsos.

A continuación veremos que la relación entre la Teoría de la Programación y la *Matemática Pura* ha ido de hecho mucho más allá: el desarrollo de la Teoría de la Programación, nos ha obligado a desarrollar nuevas disciplinas matemáticas, que si

bien podrían no serlo tanto, en el sentido de que en un pasado ya podrían haberse estudiado mínimamente, ahora se ha visto que resultan de vital importancia para el desarrollo de la Teoría, y por tanto han pasado a ser motivo de un profundo estudio a un nivel Fundamental, para que de esta forma los resultados obtenidos puedan ser en efecto Aplicados en el desarrollo de la Teoría de la Programación.

Teoría de la Programación como motor de desarrollo de nueva Matemática Fundamental

Si bien ya hemos visto que los conceptos matemáticos básicos necesarios para realizar la Teoría de la Programación, son algunos de los que más se manejan dentro de la Matemática Discreta, sobre todo a partir de la constatación de que es necesario incorporar razonamientos de Continuidad, surge la necesidad de manejar nuevas Estructuras, cuya Teoría habrá que estudiar, para así poder concluir que los objetos definidos por la misma tienen las propiedades necesarias para poder afrontar la definición y estudio de unas determinadas Semánticas. No daremos aquí en absoluto las correspondientes definiciones, pero sí diremos que en particular nos estamos refiriendo a los denominados *Dominios Semánticos*, basados en los *Ordenes Parciales Completos*, que combinan en la línea indicada una noción discreta (la de orden aquí) con una Completitud de corte inequivocamente Topológico.

Cuando se pretende hacer Matemática Aplicada y se constata que la Matemática Fundamental que tenemos a mano no parece suficiente para abordar la Modelización del problema que queremos resolver, tendremos que de alguna manera extender esa Matemática Fundamental. Ello ha de hacerse de manera que no se viole nuestro Principio de Rigor (absoluto) a la hora de utilizar útiles matemáticos en nuestra modelización. Lo deseable es separar por completo el desarrollo de esa Nueva Matemática Fundamental de su ulterior aplicación, con lo que pasará a ser simplemente una parte más de la Matemática Fundamental.

Naturalmente, es importante tener una estimación razonable de nuestras necesidades, si en principio nuestro interés por esa nueva Matemática Fundamental se circunscribe a la aplicación concreta que le queremos dar. En concreto, en algunos casos se tratará simplemente de encontrar los medios para garantizar una determinada Propiedad en el marco de una Teoría conocida, y en tal caso se tratará de llegar a demostrar un nuevo Teorema adecuado (con las Premisas que sean oportunas) para lograr que se cumpla dicha Propiedad. Muy habitualmente nos bastará para ello con pequeños remiendos o extensiones de resultados conocidos, que por la razón que fuere no fueron estudiados en su día con la generalidad que ahora necesitamos. Aún cuando éste sea el caso, y por tanto en principio nos baste con una solución ad-hoc que cubra los casos que ahora se nos presentan, probablemente resulte interesante que antes de lanzarnos a probar esa generalización específica, nos planteemos en qué sentido la posibilidad de realizarla nos está indicando que el resultado en cuestión es de hecho cierto en un marco más general, de manera que en vez de ponernos a demostrarlo en el marco reducido en que ahora lo necesitamos, lo definamos y tratemos de probar en ese sentido general.

Aún en el caso de que la demostración general pudiera resultar algo más difícil que en el caso particular que ahora necesitamos, siempre podríamos argüir que los casos probados sobrantes quedan ahí de remesa, de manera que si en el futuro se nos presenta cualquiera de ellos, lo tendríamos probado "por adelantado". Pero es que, para colmo, en muchas ocasiones la prueba específica para el caso particular que necesitamos podría mostrarse a posteriori mucho más oscura y compleja que la nueva prueba (más) general que eventualmente obtengamos. Esto es habitual que suceda cuando la "geometría" del marco total sea mucho más regular que la del caso restringido de partida. Un ejemplo paradigmático absolutamente procedente en este caso, es el de la Inducción: cuando pretendemos probar inductivamente una determinada propiedad P, nos podemos encontrar con que la Hipótesis de Inducción no es suficiente para probar la Tesis. Sin embargo, en ocasiones vemos (felizmente) que si reforzamos P obteniendo una cierta propiedad Q (en principio más potente, y por tanto "intuitivamente" más difícil de probar), la Prueba Inductiva de ésta sí es posible, obteniendose por tanto como Corolario la prueba de P.

En otras ocasiones nos encontramos sin embargo con que las Estructuras Matemáticas (bien) conocidas no resultan adecuadas para abordar la necesaria Modelización. Ese será el caso cuando estudiemos la Semántica Denotacional. Se ha tenido que "inventar" toda una nueva Teoría, la *Teoría de Dominios*, comprobandose que sus (buenas) propiedades son suficientes para la aplicación que se perseguía. Como en el caso anterior, el grado de Generalidad con el que definamos una tal Teoría tendrá que buscar un equilibrio entre la generalidad que consigamos y la dificultad de lograrla, eventualmente teniendo en cuenta la posible aplicabilidad futura de los casos "sobrantes" que por el momento no se necesiten.

Sobre todo en libros de texto, es habitual que con el ánimo de no asustar en un principio con definiciones demasiado Abstractas, se comience presentando casos particulares, en ocasiones demasiado específicos. Es posible que no resultando aparentemente difícil el desarrollo de estos casos se evite que el estudiante salga huyendo desde el principio. Pero si más tarde va a ser necesario abordar una presentación completa de una Teoría notablemente más compleja, es posible que la constatación de que lo utilizado en el caso particular no valga en absoluto en el caso general pueda ser motivo de confusión, o incluso una cierta frustración. Evidentemente, ello nunca será cierto al revés, de modo que a la postre se estudiarán dos aproximaciones diferentes, cuando con la segunda ya habría valido. Además en situaciones como ésta uno podría empeñarse en buscar la manera de generalizar la demostración del caso particular para obtener una general, cuando ello será imposible si en la de partida se usan de una forma esencial las especificidades del correspondiente caso particular.

En definitiva, cuando estudiamos Teoría de la Programación hemos de estar abiertos a que, de vez en cuando, se abran ventanas en nuestro estudio en las que se desarrollará Nueva Teoría Matemática, más o menos general, teniendo que tolerarse la posible sensación de excesiva abstracción que podría surgir, con la confianza de que de inmediato podremos continuar con el desarrollo de la Teoría de la Programación según lo deseado, aplicando la Nueva Teoría. Ciertamente no estará de más que antes de empezar nos hayan adelantado, al menos en parte, las características necesarias de esa nueva teoría que vamos a necesitar para poder en efecto proseguir, a fin de

contar con la debida motivación, refrendada de inmediato a continuación al aplicarla con éxito cubriendo nuestras necesidades.

Beneficios de la Teoría de la Programación en el campo de la Informática

De todo lo expuesto en las secciones anteriores probablemente habremos sacado la conclusión de que, como disciplina científica, la Teoría de la Programación es una materia esencialmente Matemática, tanto Aplicada como Fundamental. Pero naturalmente, su motivación es totalmente Informática: la de expresar rigurosamente el efecto de la ejecución de nuestros programas informáticos, derivando el mismo de la definición de la semántica de las distintas construcciones (sintácticas) con que cuenta el lenguaje sobre el que estén escritos.

Efectivamente, en términos históricos, este planteamiento de Formalización a posteriori de una realidad preexistente (los primeros Lenguajes de Programación que se tenían felizmente implementados, y con los que se trabajaba en la práctica a partir del conocimiento informal de su semántica), que se corresponde al pie de la letra con el de la Modelización Abstracta, es el que rigio en su día la génesis de la Teoría de la Programación. Pero ya hemos indicado antes que al realizar la misma ya se era consciente de que lo que se estaba haciendo era en efecto una labor de abstracción, en el sentido estricto del término, pues lo que se pretendía modelizar (Un Lenguaje) era intrínsicamente difuso, pues las "vistas" que se tenían del mismo (sus distintas implementaciones) coincidían en efecto en lo que hacía referencia a lo más básico, pero terminaban diferiendo inexorablemente cuando se consideraban programas complejos. Ello era así, bien porque estos manejaban construcciones del lenguaje más o menos exóticas, cuyo comportamiento "deseado" nadie terminaba de tener claro; o bien porque combinaban construcciones básicas de un modo enrevesado, generandose sutiles interacciones que tampoco se resolvían del mismo modo en las distintas implementaciones.

En consecuencia, con estas primeras formalizaciones, en ocasiones pretendidamente incompletas, dejando precisamentede lado aquellas construcciones menos claras, no terminaba de quedar claro hasta qué punto se estaba en efecto definiendo La Semántica del Lenguaje. Pero en los casos en que finalmente se tuvo éxito, terminandose de definir Una Semántica precisa, al menos de una buena parte del lenguaje, se concluyó que el esfuerzo había merecido la pena, por mucho que éste pudiera parecer en un principio excesivo (como efectivamente lo fue), en base a la aplicabilidad inmediata de lo obtenido. Como justicación, se argumentaba que era esperable que tras lo aprendido sobre el tema, en el futuro se pudieran definir Semánticas de un modo mucho más efectivo, de manera que su utilidad fuera mucho mayor, requeriendo en cambio un esfuerzo mucho menor de desarrollo. Evidentemente, este tipo de "pronósticos positivos", que se han presentado en muchos casos como autojustificación de un tremendo esfuerzo invertido sin aparentes notables resultados, no se han mostrado siempre acertados, pero por fortuna en el caso que nos ocupa no pudieron serlo más. A continuación, enunciaré diversas direcciones en las que el

estudio de la Teoría de la Programación ha tenido y seguirá teniendo consecuencias, no sólo muy positivas, sino incluso imprescindibles, para el desarrollo satisfactorio de la Informática.

1. **Definición de (nuevos) Lenguajes de Programación**. Como hemos dicho repetidamente, la Teoría de la Programación surge en principio como Modelización de unos Lenguajes preexistentes. Estos eran las gallinas desde las que se querían sintetizar unos huevos de los que hipotéticamente podrían nacer unas nuevas gallinas prototipo uniformes, como representantes más o menos ideales de la raza (poliforme) analizada. El principal objetivo de esta síntesis unificadora no era tanto aislar (e ignorar) las diferencias entre individuos de la raza (las distintas Implementaciones), como sobre todo expresar de una manera razonablemente sencilla cómo se comportaban, en lo fundamental, todos ellos. Cuando modelizamos cualquier realidad de la naturaleza, nos encontramos ineludiblemente con unas criaturas que no sabemos cómo han surgdo, por lo que tenemos que especular e intuir cómo podrían ser los huevos de dónde nacieron. Pero cuando se vio que estábamos haciendo exactamente lo mismo con los Lenguajes de Programación, obligandonos a Abstraer y a inventar Aproximaciones de Realidades preexistentes conocidas sólo a nivel de *Observación* de su comportamiento, pero sin ninguna definición formal (al menos mínimamente "legible"), se constató que se estaba actuando de un modo absolutamente masoquista, ya que las criaturas analizadas en este caso eran absolutamente artificiales y habían sido creadas con un comportamiento 100 % preciso y conocido: el que establecía su compilador inicial. El problema radicaba en que esa definición era tan indirecta y compleja, que ni el propio creador era capaz de describir cómo iba a comportarse su criatura ante situaciones complejas (programas y/o datos para los mismos). En definitiva, habíamos optado por crear como primeros humanoides complejísimos Frankestein variopintos, cuyo comportamiento posteriormente pretendíamos describir de una forma absolumante precisa, a la vez que razonablemente sencilla. Y todo ello saltando del cero al infinito, sin tiempo para ir avanzando poco a poco.

   En el caso de la naturaleza, obviamente no podemos cambiarla para que su funcionamiento pase a ser sencillo de prever ... pero si estamos hablando de criaturas que hemos creado totalmente nosotros, y además utilizando materiales absolutamente conocidos (abstraemos aquí la realización física de los ordenadores, dando por bueno que ésta se ajusta al modelo (relativamente sencillo) que describe su funcionamiento sobre el papel), la solución "científicamente" más sensata es ignorar a las extrañas criaturas creadas en un principio, y pasar a crear otras de una manera mucho más preclara, ¡empezando precisamente por ese huevo sencillo, que es su definición formal!

   Naturalmente, no se trata de echar por la borda todo lo acumulado durante décadas de desarrollo de la Informática, con la gestación y evolución de centenares, si no miles, de lenguajes que han ido buscando su diseminación máxima, bien como lenguajes "de propósito general", o para ser utilizados en marcos más o menos específicos, teniendo en cuenta las características y necesidades de

cada uno de ellos. A la hora de diseñar nuevos lenguajes, seguiremos teniendo a los lenguajes más populares como referencia, pero lo haremos de la forma más inteligente posible, intentando quedarnos con sus virtudes, pero eliminando al tiempo los problemas que su uso haya comportado. Evidentemente, no siempre virtudes y defectos son independientes, y en ese caso deberemos sopesar con acierto hasta donde hemos de preservar las virtudes, sin que ello suponga inexorablemente acarrear el peso de importantes defectos.

Estas son las líneas en las que se ha basado este proceso de Formalización de los Lenguajes de Programación:

a) **Definición Formal como Única definición primigénea de los Lenguajes**. Por mucho que queramos tener en mente características difusas más o menos importantes de los lenguajes existentes previamente, la formalización, para ser verdaderamente útil, exige el compromiso de que la Definición Formal que obtengamos sea ahora la única definición del lenguaje. La ventaja inmediata es que ésta pasa a ser "correcta por definición", sin que podamos (ni tengamos que) validarla respecto a ninguna idea (informal) preconcebida, por mucho que las hayamos tenido en mente a la hora de gestarla. A partir de este punto tendremos que asegurarnos de que las gallinas que nazcan del mismo, que incluirán tanto las distintas implementaciones, como las definiciones informales transmitidas a los usuarios del lenguaje, sean cien por cien compatibles con La Definición Formal previa, de manera que podremos entender fácilmente y de un modo uniforme el comportamiento de todos estos "clones" de la definición en cuestión.

b) **Desarrollo y Validadación de Implementaciones soportada Formalmente**. Evidentemente, la definición formal de un lenguaje no permite su uso en la práctica (más allá de la escritura de programas pequeños "a mano sobre el papel"). Necesitamos desarrollar Compiladores e Intérpretes que permitan la Ejecución de los programas escritos en dichos lenguajes sobre las correspondientes Máquinas. Pero a diferencia de la interpretación personal por parte del implementador, de lo plasmado informalmente por los creadores del lenguaje, ahora aquél cuenta con la Definición Formal del mismo, que eso sí, deberá respetar a rajatabla. Pero además esa corrección no será suficiente con intuirla, sino que tendremos (¡y podremos!) *Demostrarla Formalmente*, siempre y cuando la implementación se realice con el mismo rigor que la definición previa. Ciertamente, ello no será fácil, pues se tendrá que trasladar una Definición Abstracta, buena sobre el papel, a una Implementación Concreta, razonablemente Eficiente, y que ha de tener en cuenta las características específicas de la Máquina sobre la que se realice. De nuevo habrá que buscar un equilibrio, pues cuanto más partido se intente sacar a estas especificidades, más concreta será la implementación, y por tanto mayor la distancia entre la Definición Formal y su realización por medio de la Implementación, con la consecuente mayor dificultad a la hora de demostrar la Corrección de

ésta última respecto a la primera.

Desarrolladas y validadas distintas implementaciones, tendremos garantizada de inmediato una importante propiedad que históricamente antes echamos en falta: la *Portabilidad* de nuestros programas. Todo programa funcionará ahora de igual manera (en lo que se refiere a su comportamiento observable) sobre toda Implementación validada, pues su comportamiento será el que señale la (Única) Semántica Formal del lenguaje. Naturalmente, en principio quedarían aquí al margen los factores de *Eficiencia*, pues la rapidez de la ejecución de los programas y las necesidades de *Memoria* estarán inexorablemente ligadas a las características de cada Máquina, y a la forma en que cada compilador o intérprete las explote, al implementar cada una de las características del lenguaje. Si bien actualmente incluso estas cuestiones de eficiencia se empiezan a tener también en cuenta a la hora de validar una implementación, pues se persigue que la portabilidad incluya la garantía de una determinada *Calidad* en términos de *Rendimientos* (*Tiempo* de ejecución y necesidades de *Memoria*).

c) **Programación Productiva y Segura**. Los programadores que utilicen un Lenguaje, tendrán ahora a su disposición como Manual del mismo, su Definición Formal, quizás edulcorada de algún modo para facilitar su comprensión, pero a sabiendas de que cualquier duda sobre su funcionamiento preciso sólo podría ser resuelta por la consulta (y comprensión precisa) de la citada definición. De lo anterior se deriva una importantísima propiedad irrenunciable de toda Semántica Formal, que es su Simplicidad Conceptual, naturalmente dentro de lo posible. Ciertamente, esta simplicidad conceptual es requisito de toda teoría matemática que pretendamos que se use de un modo amplio. Como dijimos, las primeras labores de formalización de lenguajes preexistentes se encontraron con la tremenda dificultad de capturar "correctamente" multitud de elementos que se habían ido añadiendo durante la "evolución" de los lenguajes, con el (buen) propósito de hacerlos más manejables y eficientes, pero sin ser conscientes de las inevitables (extrañas) interacciones que se generaban en ocasiones entre unas y otras. Por fortuna, ahora que "somos dueños" absolutos del lenguaje al definirlo, podemos optar por construcciones "sencilllas" que permitan su definición formal simple, lo cual no sólo significa que ésta sea "pequeña", sino sobre todo fácil de entender por el programador "a alto nivel", de modo que capturado el significado de las instrucciones del lenguaje, se obtenga sin un esfuerzo adicional la comprensión de lo que hace cualquier programa que las utilice, por grande que éste sea.

Así que, facilitada la comprensión del lenguaje, al programador le será más fácil desarrollar programas correctos, pero además esta noción de Corrección pasa a poder tener ahora un significado absoluto, no estando subordinada a que el programador ha entendido correctamente lo que va a hacer "en la práctica" cada instrucción del lenguaje. Ahora el signifi-

cado de todo programa pasa a ser el que dice Su Semántica Formal, y por tanto es susceptible de ser Demostrado Correcto con las técnicas de *Verificación*, y en caso de que así lo sea, esa validación nos dará la total garantía de Corrección del programa, primero sobre el papel, pero de inmediato también en la práctica, al combinarla con la corrección de las implementaciones arriba comentada.

2. **Renovación de los procesos de creación de nuevos lenguajes y de la Metodología de la Programación**. Cuanto más claro tengamos el modo de precisar la semántica de los lenguajes, combinando Precisión y Simplicidad, más fácil será plantearnos distintas variantes de las construcciones que forman parte de la sintáxis de los lenguajes de cada tipo (imperativo, funcional, ...) , y por tanto compararlas y elegir entre ellas a la hora de definir un nuevo lenguaje. Naturalmente, todo cambio en el lenguaje podría o debería acarrear cambios en la Metodología más adecuada para sacar partido a las novedades (o viceversa, eventuales cambios deseables en la Metodología que hayamos intuído, pueden sugerir nuevos lenguajes que se espera que faciliten su aplicación práctica). Nos encontramos por tanto con que la Formalización, lejos de constituir una obligación compleja más, se convierte en una herramienta imprescindible a la hora de facilitar los medios para programar de una forma más sencilla, incrementandose en particular la garantía de que los programas que escribimos hagan en realidad lo que deben.

   Inexorablemente, nos encontraremos con críticos que argumentarán que nuestra imposición del conocimiento de algo tan complejo como la Semántica Formal de un lenguaje resulta simplemente inasumible por quien sólo pretende realizar rápidamente programas pequeños y sencillos que resuelven los problemas (pequeños y fáciles) con los que él se va a encontrar. Hasta podríamos llegarles a dar la razón en parte en este punto. En efecto, en tanto y cuando optemos por diseñar lenguajes cuyas construcciones básicas (entendiendo aquí por tales las que más se usan) sean justamente particularmente sencillas, y por tanto fáciles de usar, quienes se limiten a utilizar éstas podrían hacerlo incluso con un modelo más simple de la semántica, siempre que el mismo sea compatible con la definición formal completa del lenguaje, sin que tuvieran necesidad de acercarse siquiera a ésta, al menos por el momento. De hecho, este proceso incremental de ir presentando modelos más y más complejos, para ir definiendo la semántica de lenguajes, según estos se complican al ir incorporando nuevas características y/o sutilezas, es el que suele emplearse en casi todo los textos de Teoría de la Programación, y así lo haremos nosotros también en este curso.

3. **Incorporación de la Formalización a la Ingenería del Software**. Sería de necios empezar a hablar de este tema sin referirme a la innegable controversia que levanta, incluso entre expertos del campo del mundo académico, por no decir del profesional. Evidentemente, la incorporación de todo formalismo supone un coste añadido, sobre todo cuanto lo exigimos a rajatabla. Siendo puramente pragmáticos, podríamos defender que uno puede renunciar a comprobar que lo que hace es totalmente correcto, en tanto que la simplicidad de

lo que hace, y la confianza en que (con la práctica) se ha aprendido a hacer las cosas bien, nos permita "garantizar" que lo que hacemos es correcto (si no nos hemos "confundido" ...). Ahora bien, hay situaciones en las que pudiendose pasar por el aro de comprobar la total calidad antes de poner un producto en venta, hemos de hacerlo ineludiblemente, por mucho que ello suponga un coste adicional de producción. Como ejemplo paradigmático tenemos los *Sistemas Críticos* (aviónica, equipos de tratamiento médico, ...), donde hemos de agotar todos los medios para intentar garantizar la máxima *Seguridad*, y los *Sistemas* (excesivamente) *Complejos*, en los que los sistemas a desarrollar son tan grandes, que aún suponiendo probabilidades mínimas de cometer un error de programación en cada punto, la probabilidad global de que se produzca algún error sería seguro 1, de no optarse por un *Método Formal* que nos permita garantizar la corrección.

Curiosamente en este punto nos encontramos con un efecto importantísimo del desarrollo de la Teoría de la Programación, que ha ido más allá de lo que se perseguía inicialmente con ella. En efecto, cuando en puntos anteriores nos hemos referido a la Corrección "garantizada" de nuestros programas, estábamos en realidad diciendo, aunque quizás no hayamos hecho hincapié en ello, a que toda corrección es siempre relativa a las correspondientes expectativas, o sea a lo que "pretendíamos hacer". En terminos de Verificación, nos estamos refiriendo aquí a la *Especificación de Problemas*, que define exactamente (usualmente usando *Fórmulas Lógicas*) el comportamiento esperado de nuestros programas. Evidentemente, si pretendemos Demostrar Formalmente la Corrección, lo primero que necesitamos es esa Especificación, respecto a la cual esperamos que el programa sea correcto. Naturalmente, cuando nos contentemos con una "prueba" informal, usualmente haremos la misma con respecto a una descripción más o menos informal (pero supuestamente, suficientemente precisa) del problema que queremos resolver. Pero evidentemente dicho nivel de informalidad no parece muy pertinente por ejemplo cuando hemos de firmar un *Contrato* para suministrar a una compañia un sistema que le facilite determinados servicios que necesita. La *Especificación de Requisitos* aparece entonces como vehículo (supuestamente preciso) de (inter)comunicación entre cliente y empresa de servicios: el cliente deberá precisar con detalle sus necesidades, y la empresa entenderlos, fijando el coste del producto y la viabilidad del mismo. Evidentemente, la no existencia de unos Lenguajes Formales de Especificación de Requisitos multiplica los litigios a la hora de valorar el cumplimiento de estos contratos, incluso asumiendo la buena voluntad por ambas partes, simplemente por causa de una posible "incomprensión" por algunade las partes, a la hora de interpretar las cláusulas del contrato.

En consecuencia, la Teoría de la Programación ha pasado a ir más allá de la definición de la Semántica de los Lenguajes de Programación, adentándrose en el terreno de la Especificación de Sistemas. Ello ha venido obligado, en particular, por la complejidad de los *Sistemas Concurrentes*, en los que la "impredecible" (en el tiempo) *Sincronización* entre las distintas componentes del sistema obliga a realizar descripciones abstractas que garanticen la corrección del sistema,

al margen de las sincronizaciones concretas que tengan lugar entre ellas, incluyendo incluso el orden entre las mismas. Ya podemos imaginarnos que la Formalización de la concurrencia exige añadidos y cambios fundamentales con respecto a la Modelización de la Computación Secuencial, pero sin entrar en detalles podemos proclamar que el desarrollo de ésta ha permitido también la rápida evolución de los Métodos Formales para el Desarrollo de Sistemas, que se utiliza felizmente hoy por la industria especializada para desarrollar sistemas complejos cuya corrección puede garantizarse (módulo, por supuesto, una determinada especificación precisa de los mismos). Y para llegar a todo esto confío en que os resulte de interés y utilidad la presente asignatura.

David de Frutos Escrig, Dpto. de Sistemas Informáticos y Computación,
Fac. CC. Matemáticas (Despacho 310C), Universidad Complutense de Madrid
Correo electrónico: defrutos@sip.ucm.es