tema de la tesis de María Alejandra Zuluaga, cuyo principio era detectar automáticamente los cortes patológicos perpendiculares a la arteria.

Un año después apareció CAAVAT, un software para cuantificación de tejido adiposo visceral en imágenes de TAC, el cual constituye uno de los factores relacionados con el síndrome metabólico, predictor de un episodio cardiovascular, explicó la profesora. En él trabajaron los estudiantes doctorales Ricardo Mendoza y Duván Gómez y contó con la participación del médico radiólogo Luis Felipe Uriza, del Hospital San Ignacio de Bogotá.

El año pasado, arrancó el proyecto de cuantificación de la aireación pulmonar en pacientes que sufren el Síndrome de Deficiencia Respiratoria Aguda (SDRA), en el que participan Alfredo Morales (estudiante doctoral) y Maciej Orkisz (investigador del laboratorio francés CREATIS). El propósito es desarrollar una herramienta que en el futuro permita ajustar los parámetros de ventilación mecánica para cada paciente, pues, según explicó la ingeniera, "el problema médico consiste en cuantificar el volumen de aire pulmonar: el 5 % (promedio mundial) de ingresos a urgencia y cuidados intensivos se debe a este síndrome y el 40 % de esos pacientes mueren porque los parámetros de ventilación no están correctamente ajustados; si se inyecta demasiado volumen de aire o hay demasiada presión, el pulmón puede lesionarse, y si se inyecta muy poco aire, el paciente no puede respirar (hipoxemia)".

Hasta el momento se está trabajando sobre un modelo animal, que utiliza imágenes tridimensionales de pulmones de cerdos. El procedimiento aún no puede aplicarse a los humanos por los altos volúmenes de radiación a los que deberían someterse, pero la investigadora considera que ya se tiene una base para continuar los estudios.

El proyecto más reciente pretende la identificación temprana de enfermedades neurodegenerativas como la demencia frontotemporal y el trastorno afectivo bipolar. Es la tesis doctoral de Ricardo Mendoza, en codirección con John Puentes de Télecom Bretagne, y cuenta con la asesoría de Fabio González de la Universidad Nacional y la participación del radiólogo Luis Felipe Uriza, del Hospital San Ignacio, y de la neuropsicóloga Diana Matallana. Para ello se está desarrollando un sistema de búsqueda y recuperación de imágenes por contenido, mediante el cual el médico pueda ingresar la imagen de un paciente a un repositorio de imágenes y el sistema le entregue aquellas similares que ya han sido diagnosticadas con la enfermedad, de modo que pueda compararlas y le ayude a saber si es posible que se trate de la misma patología. ■

**Marcela Hernández** cuenta con una Maestría en Ingeniería de Sistemas y Computación de la Universidad de los Andes y es magíster en Ingeniería Biomédica del INSA de Lyon (Francia). Realizó su doctorado en el laboratorio CREATIS (Center for Research and Applications in Image and Signal Processing) en procesamiento de imágenes vasculares, patrocinada con una beca industrial. Resultado de su tesis doctoral fue Maracas (Magnetic Resonance Angiography Computer Assisted Analysis), un software que ganó el primer premio de Siemens en Journées Françaises de Radiologie, 2000 (París) y que fue vendido a Hitachi Japón en el 2005.

# Actualidad en **verificación de programación**

Éric Tanter, ingeniero de software y profesor titular de la Universidad de Chile, respondió por escrito un cuestionario sobre los temas principales de su conferencia *"Gradual Language-Based Verification Will Change the World"*.

**¿Qué lo llevó a investigar el tema de la verificación en la programación?**

Cuando empecé mi carrera de investigador, me interesaba en cómo darle mecanismos al programador para que tuviera más poder para hacer programas más adaptables, dinámicos y demás. Luego empecé a cuestionarme, ¿cómo alguien puede usar esas herramientas y saber que lo que hace tiene sentido? Esto me ha llevado a preguntarme cómo podemos establecer propiedades acerca del comportamiento de ciertos programas. Primero investigué mecanismos de verificación dinámica, como el monitoreo, y después comencé a interesarme, cada vez más, en la posibilidad de verificar estas propiedades antes de ejecutar los programas, es decir, análisis y verificación estática de programas.

**¿Qué se puede entender por verificación basada en el lenguaje?**

Esta expresión es una traducción del inglés *language-based verification*. Este término

> **Los lenguajes de programación más modernos están integrando una combinación de verificación estática y dinámica. Solo que la mayoría son sistemas de tipos simples, que únicamente aseguran propiedades débiles".**


**Éric Tanter es doctorado en la Universidad de Nantes y la Universidad de Chile en 2004.**

pretende definirse por oposición a técnicas de verificación que podríamos clasificar de "extrínsecas", es decir externas al programa: uno parte de un programa ya escrito, y luego, en otra herramienta, escribe una especificación de la propiedad que quiere verificar, y, finalmente, trata de demostrar que el programa cumple con ella.

La verificación basada en el lenguaje, en cambio, se podría clasificar de "intrínseca" o interna. Aquí la idea es que la programación, la especificación y la verificación se hacen de manera entrelazada, al mismo tiempo, mientras uno está desarrollando un sistema. Los programadores ya están acostumbrados a una forma simple de verificación basada en el lenguaje: programar en un lenguaje con verificación estática de tipos, como Java o C#. En dichos lenguajes, uno alterna entre especificar (= describir una interfaz o firma de método, con los tipos esperados), programar, y verificar (= el sistema de tipo verifica automáticamente que el que implementa una interfaz realmente cumple con lo prometido, y el que usa la interfaz lo hace de manera consistente).

La idea es que uno puede definir sistemas de tipos mucho más expresivos que los habituales, que permiten especificar y verificar propiedades mucho más interesantes sobre los programas.

### ¿Por qué puede ser conveniente una verificación gradual?

La verificación gradual se refiere a la posibilidad de combinar verificación estática (cuando uno está escribiendo un programa) y verificación dinámica (cuando uno está ejecutando un programa). En un mundo ideal, uno quisiera que todo sea verificado estáticamente, porque así evita tener problemas durante la ejecución de sus programas. Sin embargo, esto tiene su costo en complejidad para el programador. En particular, cuando está explorando alternativas en una etapa inicial de un desarrollo, puede ser molesto o incluso imposible tener que establecer propiedades fuertes sobre su programa.

No hay que olvidar que la programación es un arte que abarca un espectro amplio desde la artesanía hasta la producción de calidad industrial en sistemas críticos. Está claro que no todos los sistemas requieren del mismo nivel de rigurosidad. Y más aún: un mismo sistema no necesita la misma rigurosidad en distintas etapas de su evolución. Un lenguaje que soporta verificación gradual permite que el programador pueda apoyarse en el nivel de verificación estática que más le corresponda, y que pueda evolucionar como quiera. Hoy en día, uno tiene que elegir un lenguaje de programación dado, el cual fija la norma de rigurosidad de una vez por todas.

Los lenguajes de programación más modernos, justamente, están integrando una combinación de verificación estática y dinámica. Solo que la mayoría son sistemas de tipos simples, que únicamente aseguran propiedades débiles.

### ¿Qué tan factible es que la industria de software adopte prácticas de verificación gradual basada en el lenguaje?

Como decía, la industria de software ya está adoptando el tipado gradual, de una forma u otra. Por ejemplo, Facebook y Microsoft desarrollaron sistemas opcionales de verificación estática de tipos para PHP y JavaScript, llamados Hack y TypeScript. Google desarrolló Dart, que también tiene una integración de tipado estático y dinámico. Por otro lado, ciertos lenguajes estáticamente tipados, como C# o Scala, soportan mecanismos para postergar la verificación al tiempo de ejecución. La pregunta es saber si los sistemas de tipos más expresivos e interesantes, como los tipos dependientes, serán integrados a lenguajes *mainstream*, y de qué manera. Creo que si vienen con un mecanismo de verificación gradual, facilitará ampliamente su adopción.

### ¿Cómo pueden los desarrolladores de software conocer y adoptar este tipo de conocimiento?

Respecto de la integración de tipado estático y dinámico, como mencioné, hay muchos lenguajes modernos de interés, con respaldo industrial. Respecto de sistemas de tipos más expresivos, pero no graduales (¡aún!), lenguajes interesantes de mirar son Idris y F*. Todos estos vienen con amplia documentación en línea, lo que los pone al alcance de todos. ■

Sitio web: http://pleiad.cl/etanter