Verificación automática de documentos normativos: ¿ficción o realidad?

Daniel Gorín[1], Sergio Mera[1], and Fernando Schapachnik[1]

Departamento de Computación, FCEyN,
Universidad de Buenos Aires, Buenos Aires, Argentina *
{dgorin,smera,fschapachnik}@dc.uba.ar

Resumen El desarrollo de toda pieza de software de cierta escala comienza por una etapa que se conoce como *especificación*, donde se describen las tareas que el software debe realizar. Estas especificaciones tienen en general una inclinación deóntica, pues indican cuáles comportamientos del sistema bajo estudio son permitidos y cuáles no lo son. Siendo un producto humano, suelen contener errores, contradicciones, casos sin cubrir, etc. Dentro de la Ingeniería del Software existen técnicas y herramientas lógico-matemáticas llamadas *métodos formales*, que analizan esas especificaciones en busca de defectos, de muy difícil hallazgo manual.
Tomando como base las similitudes entre la especificación de software y la de las normas legales, este artículo explora la idea de trasladar al terreno legislativo las técnicas y herramientas que han resultado exitosas para verificar software. Además de repasar los antecedentes académicos en la interacción Informática-Derecho, aplicamos algunas de las técnicas mencionadas a un caso de estudio real en el que encontramos "lagunas" que podrían ser abusadas, y proponemos una agenda de investigación para el área.

1. Introducción

El desarrollo de toda pieza de software de cierta escala comienza por una etapa que se conoce como *especificación*, donde se describen las tareas que el software debe realizar. Estas especificaciones tienen en general una inclinación deóntica, pues indican qué comportamientos del sistema bajo estudio son permitidos y cuáles no lo son.

Por otra parte, la propia especificación, si bien es un documento riguroso, es una construcción humana, y como tal susceptible a errores. Su materia prima son ideas y deseos de seres humanos, que pasando por el tamiz de la formalización se vuelcan en expresiones técnicas y precisas. Sin embargo, nada impide que en la especificación existan contradicciones, casos no cubiertos, y varios otras clases de defectos. También otro tipo de falencia es posible: la especificación podría ser perfectamente coherente y completa en sí misma, pero por error o impericia, no reflejar los deseos de su constructor.

* Financiado parcialmente por PICT 2007 502, UBACyT 2008 X634.

Tomemos como ejemplo un sistema de control de luces para una habitación, donde la intención original pudiese formularse como "Se desea mantener el mismo nivel de luminosidad en la habitación durante todo el día, independientemente de cuánta luz ingrese desde el exterior y manteniendo tan bajo como sea posible el consumo eléctrico". Este requerimiento inicial, podría volcarse a la siguiente especificación:

1. Llámese L_d al nivel de luminosidad que se desea tener.
2. Llámese L_n al nivel de luminosidad aportado por la luz natural.
3. Llámese E a la cantidad de energía que el sistema entrega a la lámpara que ilumina la habitación.
4. Llámese L_r al nivel de luminosidad real de la habitación.
5. El sistema deberá asegurar que durante todo el día se cumpla que $L_r \geq L_d$, sin importar el valor de L_d. Para eso:
   a) Si $L_n < L_d$, deberá incrementar E.
   b) Si $L_n > L_d$, deberá disminuir E.

Esta especificación, aunque razonable a simple vista, presenta una serie de problemas. Por un lado, se podría hacer un sistema que cumpla con ella al pie de la letra, pero que al cumplir con el punto $5a$ incremente demasiado E, violando así el requerimiento inicial de no derrochar energía. Otro problema es que si la medición de la luminosidad de la sala es influida tanto por la luz natural como por la de la propia lámpara, entonces las condiciones $5a$ y $5b$ deberían estar expresadas en términos de la medición real de la habitación (L_r), y no sólo de la que entra por la ventana (L_n).

Este tipo de errores puede ser muy difícil de detectar para los humanos. Por eso, durante décadas, la industria del software ha trabajado en un área que se conoce como *verificación formal*, que consiste en la utilización de técnicas lógico-matemáticas para encontrar errores como estos en especificaciones. Se trata de una disciplina madura, que ha dado a luz técnicas y herramientas capaces de analizar con éxito casos de gran tamaño y encontrar tempranamente casos sin cubrir y contradicciones que podrían ser costosos de reparar de manifestarse en una etapa posterior.

Ahora bien, conviene notar que no hay nada que sea inherente a lo "informático" en el ejemplo que acabamos de ver. Todo lo contrario, podría perfectamente interpretarse como un conjunto de normas que debe cumplir un ser humano o un organismo administrativo. Creemos que esta dualidad no es circunstancial y que existe una patente similitud entre la especificación de software y cierto tipo de "normas legales".

Tomando como punto de partida esta observación, este artículo explora la idea de trasladar al terreno legislativo (en un sentido amplio del término) las técnicas y herramientas que han resultado exitosas para verificar software, que repasamos brevemente en la sección 3.1.

Comenzaremos por conceptualizar, en la sección 2, la tarea legislativa y profundizar sobre el tipo de análisis que creemos interesante y factible. Luego, en la sección 3.2, recorremos el camino previo en la búsqueda por relacionar Ciencias

de la Computación y Derecho, brindando nuestra propuesta programática en la sección 5. Algunas de las ideas allí presentadas se materializan en la sección 4, donde las aplicamos a un pequeño caso de estudio tomado de una norma real y obtenemos resultados interesantes, que analizamos.

2. La tarea legislativa bajo la lupa

En el ámbito del Derecho la complejidad de las normas es una característica tan familiar que la misma disciplina ha generado una serie de métodos para lidiar con ellas: desde jerarquías de cuerpos legales, apelaciones al *Derecho Natural*, apoyo en la jurisprudencia, normas complementarias, decretos reglamentarios, facultar a ciertos funcionarios a decidir (por ejemplo, los jueces), etc. Sin embargo, muchas de estas medidas paliativas no dejan de ser tales: tienen un alto costo, introducen demoras y muchas veces derivan en que efectivamente se juzguen de manera distinta actos similares.

Por este motivo, creemos en que hay un alto valor en la generación de normas más "limpias". Existe además otro elemento: cuanto más se alejan las normas de la enunciación de principios generales y más se acercan a la determinación concreta de acciones, prohibiciones, plazos, montos, valores, etc., más cerca está de ellas el ciudadano común, que no es un experto legal, y sin embargo debe poder entender claramente cómo regirse "de acuerdo a derecho". Una enunciación más clara y precisa evitaría al ciudadano no especializado muchas dudas en estos casos.

Creemos que muchas de estas dificultades podrían desaparecer si se pudiese proveer de cierta asistencia al legislador. La línea de investigación que proponemos se centra allí, en un área que se conoce como *Automated Legislative Drafting*, y que consiste en la automatización del apoyo a la tarea legislativa.

Se debe tener en cuenta que tomamos la expresión "legislar" en un sentido extremadamente amplio, entendiendo por ello la producción de leyes, normativas y reglamentaciones, tanto en el ámbito público como privado y sin que ello presuponga la existencia de un órgano colegiado legislativo. De hecho, a fin de darle un marco bien definido a una línea de investigación que ha recibido poca atención hasta el momento, en este artículo nos concentraremos en *normas de carácter administrativo*, que hablen principalmente de permisos, prohibiciones, obligaciones y procedimientos, relativas a *organizaciones de pequeña y mediana envergadura*.

Fijaremos como marco de trabajo la siguiente abstracción de la tarea legislativa:

1. Conceptualización de la norma a formular: planteo del problema a resolver y bosquejo de las ideas principales de la norma.
2. Escritura de la norma como una serie de artículos, en lenguaje natural, pero apelando a términos técnicos cuyo significado preciso ha sido previamente estipulado.
3. Chequeo de consistencia *interna*. ¿Hay algún problema en la formulación de la norma propuesta? ¿Deja "huecos"? ¿Tiene contradicciones?

4. Chequeo de consistencia *externa*. ¿La norma contradice o anula a otra?
5. En caso de detectarse problemas, repetición de los tres pasos anteriores.
6. Presentación de la norma ante el órgano encargado de promulgarla o debatirla.

Generalmente, los pasos 3 y 4 se realizan manualmente. Un conjunto de expertos (en temas legales y/o específicos del dominio de la norma) analizan la legislación existente tratando de asegurar que la nueva norma no genere dificultades en el cuerpo legal que modifica. Esta tarea es, en esencia, análoga al análisis manual de un sistema (ya sea un diseño abstracto o código concreto) en busca de defectos y, por lo tanto, proclive al mismo tipo de errores u omisiones, especialmente a medida que aumenta la complejidad del mismo. Es allí donde se sustenta la idea de *transferir conocimiento en verificación basada en lógica y métodos formales de sistemas informáticos a la tarea legislativa*.

La siguiente es una lista parcial de aspectos donde entendemos que las técnicas conocidas de verificación formal de sistemas pueden aplicarse de manera efectiva al ámbito legislativo:

a) Inconsistencias: detección de contradicciones entre artículos de la misma norma, o entre la nueva norma y aquéllas con las que debe coexistir.
b) Redundancias: algún artículo de la norma afirma lo mismo que otra existente, o que otro artículo de ella misma.
c) "Doble derogación": si la norma B deroga a la norma A, y una nueva norma C deroga a la norma B, eso significa que la norma A vuelve a valer.
d) Cobertura parcial: Si la norma prescribe distintos comportamientos (prohibiciones o permisos) sobre distintas partes de un mismo dominio, es necesario asegurarse que estén cubiertos todos los casos. Un ejemplo de falla en tal sentido es "Los técnicos mecánicos están autorizados a ... mientras que el personal no técnico sólo podrá...". ¿Qué sucede con los técnicos especializados en disciplinas distintas a la mecánica?
e) Doble rol: si la norma describe roles que pueden cumplir determinadas funciones, y esas funciones son incompatibles entre sí, puede ser un problema que no haya una delimitación clara de las características que permiten acceder a los roles. Por ejemplo, definidos los roles de víctima y victimario de manera tal de que la víctima pueda librar de culpas al victimario si así lo desea, ¿qué pasaría en el caso de que la víctima sea una sociedad y el victimario su accionista principal?
f) Análisis de procedimientos y características temporales: si la norma prescribe plazos, es interesante detectar incoherencias en ellos. Por ejemplo, es incompatible que un trámite deba realizarse por completo en 5 días hábiles pero uno de sus actores tenga permitido demorarse una cantidad no acotada de tiempo.

Surge entonces la pregunta: ¿podría concebirse en el mediano plazo un sistema informático de asistencia legislativa como el descripto? Este sistema traba-

jaría sobre la formalización de un conjunto de normas[1] y permitiría al legislador realizar análisis tendientes a detectar situaciones problemáticas y "casos de borde" como los arriba descriptos. En la sección 5 desglosaremos esta pregunta en otras de índole más técnica. Antes vale la pena hacer algunas consideraciones importantes.

Primero, las mismas limitaciones que se aplican a la verificación de sistemas se aplicarían en este caso. En particular, partimos de la base de que no es posible realizar una verificación *total* de la norma sino sólo de ciertos aspectos que se consideren relevantes.

Segundo, así como cuestiones de la política interna de una organización suelen terminar incidiendo de muchas maneras en el desarrollo de un sistema informático, es evidente que en un órgano esencialmente político como el Congreso de la Nación, estas cuestiones inciden de manera fundamental en las leyes sancionadas. Factores como la ambigüedad o la cobertura parcial juegan un rol clave a la hora de lograr consenso entre todos los actores involucrados. Si bien creemos que, en principio, esto no es necesariamente incompatible con la verificación parcial de una norma, vale aclarar que el uso de este tipo de técnicas en un órgano legislativo de fuerte contenido político va mucho más allá de nuestros objetivos inmediatos.

3. Antecedentes

Comenzaremos por repasar muy brevemente algunas técnicas de verificación de software que consideramos podrían aplicarse con éxito sobre normas legales. A continuación exploramos antecedentes donde se intersecan el Derecho y la Informática.

3.1. El software y su verificación

Ya desde la aparición de las primeras computadoras, la humanidad ha tenido que contrastar la propensión del hombre a cometer errores con la rigurosidad de la máquina. En casos extremos, pequeños errores humanos han conducido a resultados catastróficos (e.g., [1]). Surgió entonces naturalmente la *Ingeniería del Software* como un intento de emular los logros que otras áreas del conocimiento habían conseguido mediante la sistematización del saber y la creación de *ingenierías*. Dentro de la Ingeniería del Software, el área denominada *métodos formales* trata de aplicar técnicas y procedimientos con fuerte base lógico-matemática para encontrar errores y defectos en el software.

Las técnicas utilizadas son sumamente variadas, por lo que aquí resumiremos sólo las que creemos más relevantes para este artículo: aquellas basadas en el uso de *demostradores automáticos de teoremas* y de *model checkers*. En ambos casos, el estado actual de la tecnología permitió su aplicación con éxito en casos reales de gran envergadura (e.g., [2,3]).

[1] Excede a este trabajo el problema de analizar leyes escritas en lenguaje natural. Nos referimos brevemente al tema en la sección 5.

Los *demostradores automáticos* permiten especificar una teoría T y una propiedad a demostrar P, y verificar automáticamente si P se deduce de T. Muchos de ellos ofrecen la posibilidad de sintetizar un *contraejemplo* cuando P no se deduce de T. Esto es, un modelo que verifica T pero no P. Dicho contraejemplo puede ser usado como una *explicación* de por qué determinada propiedad no se cumple.

Es conveniente recalcar que si bien en la vida cotidiana solemos manejarnos con una lógica simple de herencia aristotélica, ésta no es la única existente. Existe una amplísima variedad de lenguajes lógicos, con diferente poder expresivo y que describen distintos tipos de objetos matemáticos. Por ejemplo, la lógica proposicional no permite hablar del paso del tiempo. Otras lógicas sí lo permiten. El aumento de expresividad generalmente resulta en lógicas *más complejas*, lo que significa que el procedimiento para determinar si una sentencia es verdadera o no crece drásticamente en cuanto a tiempo de ejecución y espacio de memoria requerido. Otras, como la lógica de predicados de primer orden (aquélla que permite predicar sobre individuos) son directamente *indecidibles*, lo que significa que no hay ningún procedimiento mecánico que permita determinar la veracidad o falsedad de toda sentencia en una cantidad finita de pasos.

Por ende, no debe subestimarse la importancia de la elección del lenguaje lógico a utilizar. Por un lado, se necesita un lenguaje suficientemente expresivo como para poder modelar las particularidades del caso. Por otro lado, se debe buscar un balance entre expresividad y complejidad (en general, a mayor poder expresivo, la lógica resultante posee una complejidad computacional mayor) que permita utilizar métodos de inferencia automática eficientes que sean útiles en la práctica. En este contexto, algunas alternativas se presentan con bastante claridad:

- Lógicas para la descripción (conocidas como DL, de *Description Logics*). Se trata de una familia de lógicas modales que son ampliamente utilizadas para representación conocimiento y razonamiento sobre ontologías. La principal ventaja de esta familia de lógicas es que cuenta con herramientas eficientes de inferencia (como Racer [4] y Pellet [5], entre otras) y ofrecen un poder expresivo muy razonable para tareas de modelado, especialmente en la construcción de clasificaciones ontológicas.
- Las lógicas deónticas son una familia de lógicas que permiten expresar qué es obligatorio, prohibido o permitido en cierto contexto, a través de operadores específicamente diseñados para tal fin. Es un terreno que debe abordarse con cuidado, ya que si bien hay varios "sabores", todas ellas cuentan con formulaciones donde el resultado de la manipulación lógica es contrario a la intuición. Estos casos se conocen genéricamente como paradojas (la de Jörgenson, la de Ross, la de Chisholm, etc.) [6,7]. Sin embargo parecen ser muy adecuadas para modelar cuestiones relacionadas con el ámbito legal (por ejemplo [8,9]).
- Por otro lado, los SMT solvers (de *Satisfiability Modulo Theories*) utilizan una lógica más expresiva que DL. El lenguaje generalmente usado es el de la lógica de primer orden, en donde la interpretación de un subconjunto de los

operadores del lenguaje está fija a una teoría en particular (también llamado *dominio concreto*). Ejemplo de las teorías que estos demostradores suelen manejar son los números naturales y operadores matemáticos lineales. Los SMT solvers trabajan con lógicas más expresivas que DL, y en consecuencia con métodos de inferencia en principio menos eficientes. Sin embargo, mostraron ser muy útiles en muchos casos prácticos [2,10,11,12].

Por otro lado, algunos análisis parecen ser más apropiados para aplicar técnicas de *model checking*. A diferencia de un demostrador, un model checker permite verificar automáticamente si una propiedad se satisface en una familia de modelos en particular. Si bien esta funcionalidad es menos general que la de un demostrador, la principal ventaja del model checking es que usualmente la complejidad computacional del procedimiento es varios órdenes de magnitud menor que en el caso de la demostración automática. Model checking parece ser una herramienta más que apropiada para detectar problemas en las características temporales de una norma, dado que se trata de una familia de técnicas y tecnologías nacidas con ese fin.

3.2. El Derecho desde las Ciencias de la Computación

Los intentos de relacionar Lógica y Derecho son de larga data (e.g., [13,14]). En este sentido se destaca la posición de Kelsen, que en su "Teoría Pura del Derecho" [15], presenta una visión de la ley cercana a un cuerpo de axiomas.

Mucho más recientes son los intentos de vinculación entre el Derecho y las Ciencias de la Computación. No nos referimos a aquellos que evalúan las cuestiones relacionadas con la informática desde una perspectiva legal (licencias, patentes sobre algoritmos, privacidad, firma digital, etc.), sino al intento de solucionar algunos de los problemas relacionados con el Derecho utilizando técnicas, herramientas y conceptos que vieron la luz en las Ciencias de la Computación.

Como mencionamos anteriormente, el área general que nos compete es conocida como *Automated Legislative Drafting*. Su principal objetivo es brindar asistencia informática al legislador a la hora de elaborar leyes o reglamentaciones. Actualmente, está principalmente dirigida hacia software que posibilite instanciaciones de *templates*, y otros manejos de textos. Es decir, provee ayuda esencialmente sintáctica (ver por ejemplo [16]). Nuestra búsqueda, por el contrario, apunta a la asistencia en la detección de problemas semánticos.

Otros trabajos en el área están relacionados con la posibilidad de emitir veredicto automáticamente (eg, [17]), analizar textos en lenguaje natural para codificarlos automáticamente (eg, [18,19]), encontrar formalismos para representar el conocimiento legal (eg, [20,21]), realizar análisis de argumentaciones (eg, [22]), etc.

Más cerca del análisis de normas se encuentran [23,24], que describen sistemas donde se codifican fragmentos de la legislación sobre tránsito de Holanda, y presentan casos problemáticos, cuyo resultado deseado se conoce, para ver qué determinaría la ley sobre ellos. Dada la necesidad de codificar manualmente casos

de prueba junto con sus resultados esperados, estos trabajos están más cerca del testing que de la verificación automática.

La línea más cercana a nuestro enfoque es, tal vez, la seguida por un grupo de investigadores liderado por el argentino Gerardo Schneider, de la Universidad de Göteborg, en Suecia. En trabajos recientes ([25,26], entre otros) exploran un formalismo lógico desarrollado por ellos en la especificación de contratos y usan herramientas basadas en model-checking para la verificación y *monitoreo* de los mismos. Volveremos sobre este enfoque en la sección 5.

4. Caso de estudio

En esta sección reportamos los resultados preliminares obtenidos a partir del análisis de un caso de estudio concreto.[2] El experimento consistió en tomar el reglamento para concursos de docentes auxiliares que actualmente es utilizado en la Facultad de Ciencias Exactas y Naturales de la Universidad de Buenos Aires[3], e intentar verificar algunas propiedades sobre él. Dado que no fue nuestra intención modelarlo por completo, nos restringimos sólo a un subconjunto específico de cláusulas que nos parecieron particularmente interesantes y factibles de ser verificadas. Como resultado del análisis pudimos identificar problemas en el reglamento que hasta la fecha habían sido pasados por alto.

Evaluando las características del reglamento y el conjunto de herramientas de verificación disponibles, decidimos utilizar DiVinE [28], un model checker cuyo lenguaje de entrada son autómatas finitos enriquecidos y fórmulas de *Linear Temporal Logic* (LTL) [29]. LTL es una lógica modal cuyas modalidades son interpretadas en un contexto temporal. Dado un modelo $\mathcal{M}$ que especifica un sistema de transiciones, LTL provee modalidades que permiten describir propiedades de las *trazas* sobre $\mathcal{M}$. Una traza sobre un modelo $\mathcal{M}$ es un recorrido (posiblemente infinito) sobre las transiciones de $\mathcal{M}$. Por ejemplo, la fórmula Fp va a ser satisfecha en un modelo $\mathcal{M}$ y un estado w de $\mathcal{M}$ si en toda traza de $\mathcal{M}$ con estado inicial w existe algún estado futuro donde vale p. Al lector interesado en una introducción completa a LTL lo remitimos a [29].

Los parámetros de entrada para DiVinE son el autómata que describe el sistema de transiciones y una fórmula LTL con la propiedad a verificar en dicho sistema. DiVinE provee un lenguaje de descripción de autómatas que permite definir variables que luego pueden ser modificadas en las transiciones de un estado a otro. Además, la posibilidad de realizar una transición puede restringirse agregando guardas que verifiquen si una determinada condición es cumplida. Las posibles respuestas de DiVinE luego de verificar una propiedad son, o bien "satisfacible", indicando que la propiedad es cierta para toda traza del autómata,

[2] Por limitaciones de espacio reseñamos muy brevemente dos trabajos similares: [25,27]. Ellos se centraron en el análisis de contratos en lugar de reglamentos, y el lenguaje allí utilizado carece de la capacidad para expresar magnitudes numéricas concretas, cuestión clave en nuestro análisis.

[3] *Reglamento para la provisión de cargos de docentes auxiliares*, aprobado en 2009. Versión digital disponible en http://www.exactas.uba.ar/download.php?id=837.

o bien “insatisfacible”, en cuyo caso devuelve a modo de contraejemplo una traza donde dicha propiedad no se cumple.

La primera propiedad a verificar consistió en validar si siempre era posible cumplir los plazos establecidos por el reglamento. La forma de abstraer lo estipulado por el reglamento en ese sentido fue la siguiente. Consideramos que un concurso se inicia estando en estado *abierto* en base al siguiente artículo:

> ARTICULO 5º. Una vez aprobado el llamado a concurso, el Decano deberá declarar abierta la inscripción [...]

Mientras el concurso está abierto se realiza la inscripción de aspirantes. Luego de vencido el plazo de inscripción, a cada aspirante se lo evalúa mediante una prueba de oposición:

> ARTICULO 31º La capacidad docente del aspirante será evaluada mediante una prueba de oposición, cuya modalidad será propuesta por el Jurado [...].

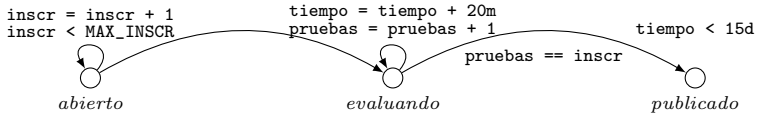
Esto lo representamos cambiando el estado del concurso a *evaluando*. Una vez terminada la evaluación, se elabora el dictamen y se publica el orden de mérito. En ese momento el concurso pasa al estado *publicado*. Ahora bien, existen ciertas restricciones temporales sobre la duración de cada una de estas etapas. Por un lado:

> ARTICULO 31º [...] La duración de las exposiciones orales no deberá ser inferior a los veinte minutos. [...]

Y por otro:

> ARTICULO 24º. El Jurado deberá producir dictamen dentro de los diez (10) días de haber recibido los antecedentes y la documentación [...]. Este término podrá ampliarse por un lapso no mayor a cinco (5) días [...].

Resumiendo, un posible autómata que modela esta sección del reglamento es el siguiente:



Suponiendo que todas las variables se inicializan en cero, mientras el autómata se encuentra en el estado *abierto* pueden realizarse inscripciones al concurso. Esto está representado mediante una transición que actualiza la variable `inscr`. No hay un límite reglamentario para la cantidad de inscriptos, pero en la práctica no puede haber infinitos inscriptos, con lo cual limitamos la cantidad a una constante arbitraria suficientemente grande. La transición hacia *evaluando* puede ocurrir en cualquier momento y sin ninguna guarda asociada (en este caso no

era de interés modelar el tiempo transcurrido en las inscripciones). Durante el estado *evaluando* se representa el hecho de tomar una prueba de oposición mediante una transición que actualiza las variables `pruebas` (la cantidad de pruebas tomadas hasta el momento) y `tiempo` (el tiempo de duración de cada prueba). El concurso puede luego pasar al estado *publicado* siempre y cuando se hayan tomado todas las pruebas de oposición. Esto se verifica mediante una guarda en dicha transición. Finalmente, y siguiendo el artículo 24, el concurso debe estar publicado en a lo sumo 15 días de haber comenzado a evaluar a los candidatos.

La propiedad que se quiere verificar es si en toda posible traza sobre este autómata el estado *publicado* es alcanzable. Esto puede ser expresado con la fórmula `F publicado`. El resultado devuelto por DiVinE al realizar la verificación fue una traza en donde dicha propiedad no se cumple: si se superan los 1080 inscriptos, no es lógicamente posible que el jurado asista a una prueba de oposición por aspirante de al menos 20 minutos y emita un dictamen en menos de 15 días.

Hay un sentido claro en el cual este ejemplo muestra un problema en el reglamento: se trata de un escenario en donde todas las acciones se ajustan a la norma pero a pesar de ello es imposible cumplir con los plazos reglamentarios. Pospongamos brevemente la discusión acerca de la *relevancia* de este escenario.

El siguiente experimento que realizamos consistió en verificar otra propiedad sobre el mismo reglamento. En este caso nos concentramos en la sección que reglamenta las inscripciones para Ayudantes de Segunda. La norma dice:

> ARTICULO 6º. [...] Se deja expresamente establecido que al momento de la inscripción los aspirantes a Ayudantes de Segunda deberán ser estudiantes de grado.

Estas condiciones fueron modeladas en DiVinE a través de un conjunto de autómatas que representan las condiciones sobre un individuo para ser estudiante de grado y su relación con un concurso de Ayudante de Segunda. Vale aclarar que el autómata que codifica las transiciones por las que pasa un estudiante hasta obtener su título de grado no se puede construir a partir del reglamento que estábamos verificando (que trata sobre concursos) sino que fue realizado en base a nuestro conocimiento sobre el funcionamiento de la Facultad.

Ante la pregunta de si existe la posibilidad de que un graduado pueda inscribirse a dicho concurso, la respuesta fue afirmativa y la traza devuelta mostró a una persona que, luego de haberse graduado en una carrera, comenzaba una nueva carrera de grado (distinta a la primera) pasando nuevamente a ser estudiante. Esto lo habilitaba a inscribirse en el concurso. Este resultado hace evidente un problema de subespecificación en el reglamento, dado que el espíritu del artículo 6º consiste en impedir que graduados ocupen cargos de Ayudante de Segunda, destinados a estudiantes.

Como fue mencionado en las secciones anteriores, estos casos de estudio son sólo resultados preliminares y deben ser vistos como una pequeña prueba de concepto. Desde este punto de vista, podemos decir que el experimento fue exitoso: aplicando técnicas estándar de verificación de software sobre una norma pudi-

mos encontrar errores en ella. Sirve, de alguna manera, para reforzar nuestra hipótesis de trabajo.

Surgen naturalmente algunas preguntas críticas. Por ejemplo, puede quedar la sensación de que una inspección manual y detallada de la norma también hubiese encontrado estos errores. ¿No invalidaría esto nuestro ejemplo? Y si sólo se pueden detectar este tipo de errores, ¿no haría esto innecesario o redundante el uso de herramientas complejas de verificación? Nuestra respuesta sería que indudablemente se podrían haber encontrado estos problemas con una inspección manual, y sin embargo, no sucedió; los errores están allí en la norma. Más en general, algo similar ocurre con el software; diríamos que son muy raros los errores que una herramienta puede detectar y que un desarrollador suficientemente educado y concentrado no fuera capaz de hallar, y sin embargo la verificación de software no es por eso innecesaria.

La siguiente pregunta sería si son *relevantes* los errores encontrados. ¿Tiene sentido, por ejemplo, preocuparse por el hipotético caso en que un concurso tenga un millar de inscriptos?

Hablar de mil aspirantes es disparatado, pero cincuenta es un número frecuente y ochenta es un número ciertamente posible. Con ochenta aspirantes, un jurado debería dedicar más de tres jornadas completas de ocho horas, sin pausas, sólo para escuchar las pruebas de oposición, y esto suponiendo que puede desatender el resto de sus obligaciones docentes y científicas. Y todavía necesita analizar los antecedentes de cada aspirante y discutir caso por caso. ¿Es en este caso el plazo estipulado de quince días para la confección del dictamen razonable? ¿Tiene sentido que el plazo sea el mismo, sean dos los aspirantes u ochenta?

Nuestra conclusión es que el contraejemplo de más de 1080 inscriptos corresponde a un escenario inconcebible, pero que surge de llevar al límite del absurdo lógico un error que sí está presente en el reglamento: no existe relación alguna entre los plazos máximos y la dimensión de la tarea que se debe llevar a cabo.

La última pregunta se relaciona con la relación costo-beneficio que estas técnicas podrían tener. En el caso del software, muchas veces es posible estimar el costo económico que una falla no-detectada podría tener para una organización; cuando éste es alto, el costo extra asociado al uso de técnicas de verificación formal se ve claramente justificado. En el caso de la normativa legal, no tenemos conocimiento de trabajos que analicen el costo que puede tener (sobre el Estado Nacional, por ejemplo), la promulgación de normas defectuosas. Entendemos que este costo puede ser difícil de estimar puesto que involucra gastos administrativos en procesos judiciales, promulgación sucesiva de decretos reglamentarios corrigiendo los anteriores, etc. En consecuencia, no podemos dar, por ahora, una respuesta concluyente.

5. Un plan de investigación para el área

Repitamos la pregunta central de este artículo: ¿podría concebirse en el mediano plazo un sistema informático de asistencia legislativa como el descripto en la sección 2?

Como primer paso en esta dirección, buscamos dar respuesta a las siguientes preguntas, de índole más técnica:

- ¿Cuál es el formalismo apropiado para codificar las normas en pos de los análisis antes mencionados? ¿Cuál es el mejor balance entre expresividad del lenguaje y complejidad del mecanismo de decisión en este dominio en particular? ¿Hay un único formalismo apropiado para todos los análisis?
- ¿Cuáles son el marco teórico y la metodología algorítmica apropiados para resolver cada uno de los análisis? ¿Podrán codificarse todos los problemas de manera tal de apoyarse en herramientas ya existentes o deberán desarrollarse nuevas?
- ¿Pueden expresarse los análisis antes mencionados y sus resultados de manera tal que sean entendibles por expertos en Derecho pero no en Lógica ni Computación?

Estas preguntas generan desafíos interesantes, que nos gustaría comentar a modo de agenda de investigación para el área:

Lenguaje natural. Como ya anticipamos, el esfuerzo de interpretar texto en lenguaje natural excede a nuestro marco de trabajo, ya que deseamos concentrarnos en el análisis de normas codificadas en algún formalismo lógico. Somos conscientes, sin embargo, de que al elegir una fórmula para codificar una frase se está haciendo un proceso de desambiguación e interpretación, que podría enmascarar problemas que existen en la norma pero no en su codificación. Este punto merece análisis futuro, teniendo en cuenta el trabajo realizado en lenguajes naturales controlados (por ejemplo, [30]).

Conocimiento común. Podemos decir que toda norma legal presupone cierto conocimiento del mundo. Este conocimiento es necesario para entender la norma en sí pero también para encontrar defectos. En nuestro caso de estudio, debimos utilizamos nuestro conocimiento sobre el proceso mediante el cual una persona pasa de estudiante a graduado y esto nos permitió encontrar una violación en la letra del espíritu del reglamento. Prevemos que el conocimiento común juegue un rol importante en general; cómo administrar esto tratando de no caer en una teoría ad-hoc para cada norma es de por sí un desafío importante.

Etiquetado. Creemos que puede ser conveniente etiquetar las normas, de manera tal de brindarle "pistas" a las herramientas de análisis. Por ejemplo, indicar que ciertos artículos corresponden a un proceso, para analizarlos con un model checker, o que cierta norma describe cómo un dominio se divide en clases. Por ejemplo, declarar que "los trabajadores son activos o pasivos" es una partición de dominio que serviría para determinar que una norma que se refiera a "...

los trabajadores que sean activos y pasivos a la vez ..." es errónea, y una que solamente hable de los derechos de los activos tal vez esté omitiendo hacerlo sobre la otra clase de la partición, los pasivos.

Un lenguaje, varias herramientas. Dada la diversidad de análisis que pretendemos realizar, y que diferentes herramientas y técnicas tienen distintas debilidades y fortalezas, tendemos a inclinarnos por un lenguaje de entrada en el que se codificarían las normas, del que luego se tomarían fragmentos para ser traducidos a otros formalismos. Así, por ejemplo, una parte que hable de procesos se traduciría a autómatas para ser manejada por un model checker, mientras que otra parte que hable de jerarquías tal vez se dejaría en manos de un razonador de DL. El etiquetado anteriormente mencionado podría ayudar a hacer esta distinción.

Poder expresivo. Este lenguaje deberá poder acomodar no sólo aspectos como los relacionados con las jerarquías (por ejemplo, para entender que un *ladrón* es un tipo particular de *delincuente*), con tiempo (para manejo de procesos), con magnitudes numéricas, y con una variedad de otras cuestiones, muchas de ellas reseñadas en [31] y otras en [6].

¿Lógica deóntica? Como mencionamos en la sección 3.1, las lógicas deónticas nacieron con la intención de razonar sobre aspectos muy cercanos al que nos aboca. Sin embargo, su desarrollo está plagado de las "paradojas" antes mencionadas. Si bien recientemente los esfuerzos que se derivan del lenguaje $\mathcal{CL}$ [9] parecen haber encontrado una vía para resolver muchas de ellas, aún queda por determinar si todas ellas pueden darse por superadas. Independientemente de esto, hay una pregunta provocativa que creemos que no debe esquivarse: ¿es necesario explicitar los operadores deónticos? Las especificaciones de software también indican deberes y prohibiciones, y no los utilizan. Entendemos que esta decisión podría traer como consecuencia la pérdida de poder expresivo (tal vez la obligación a obligar no podría modelarse, entre otras). Sin embargo, entendemos que la verificación siempre necesita ser *parcial* para ser factible, y tal vez ése sería un precio razonable a pagar.

Operadores lógicos apropiados. Si bien en todo lenguaje lógico la elección de operadores es crítica, nos enfrentamos aquí con un problema adicional, que es que en el lenguaje natural se utilizan las mismas expresiones con significados distintos. Por ejemplo, la expresión "se puede hacer a y b" puede significar el permiso para hacerlas en simultáneo o que ambas están permitidas, independientemente de la otra.

Deben sumarse a esta lista las preguntas sobre pertinencia expresadas en la sección anterior. Si la experiencia en verificación de software sirve de guía, el camino debería llevar a la par la resolución de cada una de estas consideraciones con el abordaje de casos de estudio progresivamente más complejos.

Referencias

1. Dowson, M.: The ariane 5 software failure. SIGSOFT Softw. Eng. Notes **22** (1997) 84
2. Baumann, C., Bormer, T.: Verifying the PikeOS Microkernel: First Results in the Verisoft XT Avionics Project. In: Doctoral Symposium on Systems Software Verification (DS SSV-09) Real Software, Real Problems, Real Solutions. (2009) 20
3. Daws, C., Yovine, S.: Two examples of verification of multirate timed automata with KRONOS. In: Proceedings of the 16^{th} IEEE Real-Time Systems Symposium (RTSS'95), Pisa, Italy, IEEE Computer Society Press (1995) 66–75
4. Haarslev, V., Möller, R.: Racer: An owl reasoning agent for the semantic web. In: Proceedings of the International Workshop on Applications, Products and Services of Web-based Support Systems, in conjunction with the 2003 IEEE/WIC International Conference on Web Intelligence,Ê Halifax, Canada, October 13. (2003) 91–95
5. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical owl-dl reasoner. Web Semant. **5** (2007) 51–53
6. Hansen, J., Pigozzi, G., van der Torre, L.W.N.: Ten philosophical problems in deontic logic. [32]
7. Hansen, J.: Imperatives and deontic logic: On the semantic foundations of deontic logic. Phd thesis, University of Leipzig, Germany (2008)
8. Makinson, D., van der Torre, L.W.N.: What is input/output logic? input/output logic, constraints, permissions. [32]
9. Prisacariu, C., Schneider, G.: $\mathcal{CL}$: An action-based logic for reasoning about contracts. In: WoLLIC '09: Proceedings of the 16th International Workshop on Logic, Language, Information and Computation, Berlin, Heidelberg, Springer-Verlag (2009) 335–349
10. Böhme, S., Moskal, M., Schulte, W., Wolff, B.: HOL-Boogie – An Interactive Prover-Backend for the Verifying C Compiler. Journal of Automated Reasoning (2008) 1–34
11. Jha, S., Limaye, R., Seshia, S.: Beaver: Engineering an efficient SMT solver for bit-vector arithmetic. In: Proc. Computer-Aided Verification (CAV), LNCS. Volume 5643., Springer (2009)
12. De Moura, L., Bjorner, N.: Z3: An efficient SMT solver. Lecture Notes in Computer Science **4963** (2008) 337
13. Mally, E.: Grundgesetze des sollens. elemente der logik des willens. graz: Leuschner & leubensky. (1926). In Wolf, K., Weingartner, P., eds.: Ernst Mally, Logische Schriften: Großes Logikfragment, Grundgesetze des Sollens. D. Reidel (1971) 227 – 324
14. Perelman, C.: What Is Legal Logic. Isr. L. Rev. **3** (1968) 1
15. Kelsen, H.: Teoría pura del derecho. 2 edn. Universidad Nacional Autónoma de México, Dirección General de Publicaciones, Instituto de Investigaciones Jurídicas (1982)
16. Hafner, C.D., Lauritsen, M.: Extending the power of automated legal drafting technology. In Lodder, A.R., Mommers, L., eds.: JURIX. Volume 165 of Frontiers in Artificial Intelligence and Applications., IOS Press (2007) 59–68
17. van de Ven, S., Hoekstra, R., Breuker, J., Wortel, L., El-Ali, A.: Judging amy: Automated legal assessment using owl 2. In Dolbear, C., Ruttenberg, A., Sattler, U., eds.: OWLED. Volume 432 of CEUR Workshop Proceedings., CEUR-WS.org (2008)

18. Lenci, A., Montemagni, S., Pirrelli, V., Venturi, G.: Ontology learning from italian legal texts. In: Proceeding of the 2009 conference on Law, Ontologies and the Semantic Web, Amsterdam, The Netherlands, The Netherlands, IOS Press (2009) 75–94
19. Völker, J., Fernandez Langa, S., Sure, Y.: Supporting the construction of spanish legal ontologies with text2onto. Computable Models of the Law: Languages, Dialogues, Games, Ontologies (2008) 105–112
20. Hoekstra, R., Breuker, J., Bello, M.D., Boer, A.: The lkif core ontology of basic legal concepts. In Casanovas, P., Biasiotti, M.A., Francesconi, E., Sagri, M.T., eds.: LOAIT. Volume 321 of CEUR Workshop Proceedings., CEUR-WS.org (2007) 43–63
21. Agnoloni, T., Bacci, L., Francesconi, E., Spinosa, P., Tiscornia, D., Montemagni, S., Venturi, G.: Building an ontological support for multilingual legislative drafting. In: Proceeding of the 2007 conference on Legal Knowledge and Information Systems, Amsterdam, The Netherlands, The Netherlands, IOS Press (2007) 9–18
22. Wyner, A.Z., Bench-Capon, T.J., Atkinson, K.: Three senses of "argument". Computable Models of the Law: Languages, Dialogues, Games, Ontologies (2008) 146–161
23. Haan, N.D.: Tracs: A support tool for drafting and testing law. In: Jurix 92: Information Technology and Law. (1992) 63–70
24. Breuker, J., Petkov, E., Winkels, R.: Drafting and validating regulations: The inevitable use of intelligent tools. In: AIMSA '00: Proceedings of the 9th International Conference on Artificial Intelligence, London, UK, Springer-Verlag (2000) 21–33
25. Fenech, S., Pace, G.J., Schneider, G.: Automatic conflict detection on contracts. In: ICTAC '09: Proceedings of the 6th International Colloquium on Theoretical Aspects of Computing, Berlin, Heidelberg, Springer-Verlag (2009) 200–214
26. Fenech, S., Pace, G.J., Schneider, G.: Clan: A tool for contract analysis and conflict discovery. In Liu, Z., Ravn, A.P., eds.: ATVA. Volume 5799 of Lecture Notes in Computer Science., Springer (2009) 90–96
27. Pace, G., Prisacariu, C., Schneider, G.: Model checking contracts – a case study. Automated Technology for Verification and Analysis (2007) 82–97
28. Barnat, J., Brim, L., Ročkai, P.: DiVinE 2.0: High-Performance Model Checking. In: 2009 International Workshop on High Performance Computational Systems Biology (HiBi 2009), IEEE Computer Society Press (2009) 31–32
29. Blackburn, P.e., van Benthem, J.e., Wolter, F.e.: Handbook of modal logic. Studies in Logic and Practical Reasoning 3. Amsterdam: Elsevier. (2007)
30. Pace, G., Rosner, M.: A controlled language for the specification of contracts. In: Workshop on Controlled Natural Language. (2009)
31. Pace, G.J., Schneider, G.: Challenges in the specification of full contracts. In: IFM '09: Proceedings of the 7th International Conference on Integrated Formal Methods, Berlin, Heidelberg, Springer-Verlag (2009) 292–306
32. Boella, G., van der Torre, L.W.N., Verhagen, H., eds.: Normative Multi-agent Systems, 18.03. - 23.03.2007. In Boella, G., van der Torre, L.W.N., Verhagen, H., eds.: Normative Multi-agent Systems. Volume 07122 of Dagstuhl Seminar Proceedings., Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2007)