Detección automática de defectos normativos, una invitación a la colaboración*

Manuel Giménez[1], Sergio Mera[1], and Fernando Schapachnik[1]

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

Resumen En la edición anterior de estas mismas jornadas presentábamos nuestro enfoque sobre la verificación automática de documentos normativos. Un año después, con mucho trabajo en el haber, presentamos a la comunidad FL, nuestro lenguaje deóntico, junto con una serie de herramientas destinadas a encontrar incoherencias en documentos normativos. Lo hacemos mediante un caso de estudio que pone de relieve la capacidad de encontrar fallas para nada evidentes. Este artículo configura un llamado a la comunidad de investigadores y profesionales para colaborar en el análisis de casos de estudio reales, más complejos aún, como una forma de emprender el ciclo virtuoso de investigación y desarrollo con vistas a mejorar la calidad de las normas que nos regulan.

1 Introducción

En la edición anterior de estas mismas jornadas presentábamos nuestro enfoque sobre la verificación automática de documentos normativos [1]. Se trataba de explorar las similitudes entre las *especificaciones de software* –aquellos documentos que describen cómo debe comportarse un sistema informático, y que son elaborados antes de la programación del mismo– y los documentos normativos de carácter operativo, de uso diario, que a grandes rasgos describen cómo deben comportarse las personas, físicas o jurídicas.

En este año que pasó realizamos una intensa tarea, tanto desde el punto de vista teórico como el práctico y creemos que es momento de ampliar el grupo de colaboradores e involucrar a la comunidad. En el plano teórico, nos preguntamos hasta qué punto ambos documentos prescriptivos, especificaciones y reglamentos, eran similares. El resultado de dicho análisis, positivo en general, se repasa brevemente en la siguiente sección.

Elaboramos también un lenguaje formal, llamado FL, junto con una serie de herramientas que, dada una descripción de un sistema normativo escrita en FL, analizan automáticamente dicho sistema en búsqueda de incoherencias. En la sección 3 presentamos el lenguaje y las herramientas asociadas mediante sencillos ejemplos de uso. Allí también discutimos qué tipo de análisis podemos realizar.

* Financiado parcialmente por PICT 2007 502, UBACyT 20020090200116 y UBACyT 2002009020084.

Para mostrar el poder de la herramienta la sección 4 presenta un caso de estudio. Si bien ficticio, su inspiración es real, al igual que algunos de sus fragmentos, y pone de relieve cómo una maquinaria lógica puede servir para exponer defectos que estando presentes no son para nada evidentes.

El objetivo de este trabajo es hacer un paréntesis en las publicaciones exclusivamente académicas ([2,3]) y mostrarle al profesional, al abogado a cargo de la redacción de normas, el poder de la herramienta construida, ya que creemos que sólo con su ayuda podremos encarar el resto del trayecto: ajustar lenguaje y herramientas de manera de que codificar un nuevo documento normativo sea costo-eficaz en relación al valor de la información que las herramientas provean sobre el mismo, y los potenciales problemas evitados. Por este motivo la sección final del presente artículo es una convocatoria a la comunidad para encontrar casos de estudio reales. Nos interesan normativas reales y defectuosas, donde sepamos que hay problemas, contratos complicados[1] o normativas que se estén construyendo, para analizarlas en conjunto y contribuir a su mejora.

En definitiva, el presente artículo busca despertar el interés de la comunidad –y por qué no la crítica– como paso previo a lograr su colaboración con el objetivo común de realizar un aporte hacia la mejora de la calidad de las normas que nos regulan.

2 Similitudes y diferencias

En [3] presentamos una comparación detallada de la que aquí haremos solamente un resumen.

Al analizar el tipo de construcciones deónticas que se utilizan en las especificaciones de software y en los sistemas normativos, encontramos que

- obligaciones contrarias al deber (conocidas como *CTD*, por *Contrary-To-Duty Obligations*, o también como *obligaciones con reparación*),
- enmiendas a reglas anteriores,
- permisos,
- jerarquías, y
- ontologías

son elementos que las especificaciones de software también usan, aunque de una manera un poco distinta. Esto no conforma un problema, dado que la adaptación es posible.

Existen sin embargo otros elementos normativos, como ser:

- operadores deónticos anidados, como en "*El juez está obligado a obligar a la policía a ...*",

[1] Entendemos que desde el punto de vista del Derecho una reglamentación y un contrato son entes distintos. Sin embargo, al nivel de abstracción en el que se producen nuestros análisis ambos son documentos normativos que prescriben el comportamiento de uno o varios agentes, y por ende podemos cubrirlos de manera unificada.

– modificaciones auto-referenciales, como cuando una norma modifica a otra dentro del mismo sistema normativo, o
– validez deóntica condicional, como cuando una obligación entra en vigencia sujeta a la existencia de otra (por ejemplo, "*si al momento de la puesta en vigencia el sujeto estuviese obligado a la conducta X, tendrá entonces también que cumplir con la conducta Y*").

Creemos que estos últimos elementos no se encuentran presentes en las especificaciones de software, y que su característica común es que ponen a los operadores deónticos como elementos de primera clase (en el sentido lógico de que los mismos operadores pueden ser parámetros de otros). De todas maneras, consideramos que la mayor parte de los documentos normativos que nos interesan, esto es, aquéllos de carácter operativo, que regulan el quehacer diario de los individuos, pueden expresarse perfectamente sin recurrir a elementos de este último tipo.

Esto es una buena noticia, porque significa que una gran cantidad de documentos normativos pueden analizarse con técnicas y herramientas que se pensaron originalmente para tratar con elementos de software.

La ventaja de poder usar herramientas existentes es múltiple y no tiene que ver solamente con la disponibilidad inmediata. Estas herramientas, conocidas en el mundo del software como *model checking tools* comenzaron a desarrollarse cerca del año 1981 ([4]). Si tomamos como referencia al año 2007, donde el premio máximo de la Computación, el *ACM Turing Award* se le otorgó a Clarke, Emerson y Sifakis por su trabajo pionero en esta área ([5]), estamos hablando de más de dos décadas para alcanzar la madurez. Herramientas que comiencen desde cero deberán recorrer nuevamente un largo y difícil camino para alcanzar niveles aceptables de expresividad y rendimiento. Por otra parte, las herramientas del mundo del software continúan siendo mejoradas regularmente, debido a factores de gran peso que no tienen que ver exclusivamente con la intersección entre la informática y lo legal, área aún incipiente.

En la sección siguiente describiremos nuestra propuesta de lenguaje y herramientas asociadas, que se montan sobre trabajos ya existentes en el mundo del software.

3 Introducción a FL

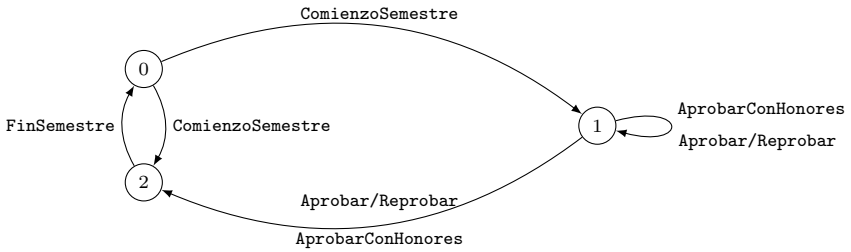
Nuestro corpus de herramientas, llamado FormaLex, se centra en el *análisis de coherencia* de documentos deónticos escritos en FL, un lenguaje basado en la *lógica temporal lineal*, LTL.

¿Qué es *coherencia*? Se trata de la formalización de la idea intuitiva de ausencia de contradicciones dentro de un sistema normativo[2]. Si bien precisaremos

[2] Somos perfectamente consciente de que *cierto* tipo de normas requieren de ambigüedades y por ende posibles contradicciones para ser aprobadas. Sin embargo, hay otras, de carácter operativo, que regulan el quehacer diario de los individuos, que podrían prescindir de estos problemas.

el concepto un poco más adelante, aquí adelantaremos que dicha ausencia de contradicciones no puede reducirse a la mera consistencia lógica [6].

Dado que el funcionamiento de nuestras herramientas está basado en el uso de *model checkers*, conviene repasar cómo funcionan éstos. Sus algoritmos reales son bastante complicados aunque conceptualmente lo que hacen es simple: toman como entrada una descripción del sistema a analizar, y una fórmula. La descripción del sistema se expresa mediante un *autómata*, y representa los *estados posibles* del sistema y sus *transiciones*, definidas como los cambios posibles entre un estado y otro. Entonces, la fórmula de entrada es analizada en términos de los posibles recorridos definidos por el autómata, verificando si siempre es posible satisfacerla sin importar el camino elegido. Supongamos que queremos modelar una examen con tres resultados posibles (reprobar, aprobar o aprobar con honores), y que el examen ocurre en el marco de un semestre, que está delimitado por la ocurrencia de los eventos `ComienzoSemestre` y `FinSemestre`. Esto podría representarse con el siguiente autómata:



La exploración de este autómata genera una serie de *trazas* lineales, cada una correspondiendo a un camino distinto dentro del autómata. Cada traza es en realidad un *modelo* y todas juntas conforman la *clase de modelos* sobre la que se evalúan las fórmulas. Veamos algunas trazas posibles:

`ComienzoSemestre` → `FinSemestre` → ...
`ComienzoSemestre` → `Aprobar` → `FinSemestre` → ...
`ComienzoSemestre` → `Reprobar` → `FinSemestre` → ...
`ComienzoSemestre` → `AprobarConHonores` → `FinSemestre` → ...
`ComienzoSemestre` → `Reprobar` → `Aprobar` → ...
...

La palabra *lineal* en el nombre de la lógica LTL no debe generar confusiones. Las bifurcaciones debidas a la multiplicidad de posibilidades en el autómata[3] *son* tenidas en cuenta. La diferencia entre las lógicas lineales y las llamadas *branching*, como por ejemplo CTL, tienen que ver con conceptos técnicos como expresividad de fórmulas, y se sabe que ni LTL ni CTL se incluyen por completo la una a la otra. LTL es considerada como una mejor opción a la hora de caracterizar comportamiento [7].

[3] Como por ejemplo elegir entre las distintas transiciones que salen del estado 1, en el autómata mostrado.

FL, nuestro lenguaje deóntico presentado en [3,2], se basa en las siguientes premisas:

- Su principal objetivo es encontrar *problemas de coherencia* en documentos normativos. Utilizamos para eso una noción de coherencia muy pragmática: no puede haber comportamientos que estén a la vez permitidos y prohibidos para los mismos individuos, como tampoco estar prohibidos y ser obligatorios. Tampoco puede haber obligaciones lisas y llanas y a la vez obligaciones con reparaciones (CTDs), ni puede haber CTDs prohibidas por otras reglas, etc. La lista completa de casos que configuran una incoherencia se encuentra en [3].
- El lenguaje está divido en dos partes: una *background theory*, o *teoría marco*, y un conjunto de reglas. Mientras que las reglas son fórmulas LTL con operadores deónticos para expresar permisos, prohibiciones y obligaciones, la teoría marco provee mecanismos sencillos para describir la clase de modelos sobre los que predican las reglas. Allí se expresan cosas como precedencia de eventos (e.g., el día ocurre antes de la noche), unicidad (e.g., las personas nacen una sola vez), etc.
- Los modelos son lineales –y por ende se llaman *trazas*–, y cada uno describe un posible comportamiento *legalmente válido* de los agentes involucrados. Dicho de otra forma, los comportamientos que no cumplen con las reglas son descartados.
- Si algo es obligatorio, entonces debe valer en todo modelo legalmente válido y por ende $O(\varphi)$ se interpreta como la fórmula LTL $\Box\varphi$, es decir, la fórmula que dice que en todo estado del sistema φ es válida.
- Si bien el lenguaje se basa en una traducción a LTL la sintaxis original de las reglas se preserva, y eso permite realizar algunos análisis a un nivel meta-lógico. Los detalles de dicho análisis se presentan en [3].

En FL la prohibición de algo es la obligación de su contrario ($F(\varphi) \equiv O(\neg\varphi)$). Las obligaciones CTD se escriben como $O_\rho(\varphi)$, y $F_\rho(\varphi)$ se interpreta como $O_\rho(\neg\varphi)$, es decir, "*obligatorio* $\neg\varphi$*, pero si eso no se cumple, entonces obligatorio* ρ". Para el lector avanzado en el tema, vale la pena remarcar que nuestra forma de codificar el lenguaje evita la mayoría de las llamadas "paradojas deónticas" (ver [3] para más detalles).

El permiso se piensa como ausencia de prohibición, pero se trata no como un operador que modifica el conjunto de comportamientos legalmente válidos, sino como un predicado que el resto de los modelos legalmente válidos deben cumplir. Si eso no sucede se considera que el sistema normativo bajo análisis (SNBA para abreviar) tiene un *problema de coherencia*: indica que algo está permitido cuando en realidad no lo está. Si el usuario de la herramienta marca el permiso en cuestión como una excepción a una prohibición, como en el caso de F(matar) + P(matar en defensa propia), entonces la representación interna de las reglas afectadas se cambia para reflejar eso. En el ejemplo, se cambiaría a F(matar excepto en defensa propia)[4].

[4] Esto puede hacerse automáticamente para casos sencillos y requiere de intervención manual en otros.

El principal componente de la teoría marco es la *acción*. Una acción puede estar sucediendo o no en cada instante de tiempo. En FL las acciones se interpretan como *señales digitales*, que pueden estar prendidas o apagadas por una cantidad arbitraria de estados consecutivos. Las acciones pueden representar acciones del agente implícito propiamente dichas (por ejemplo, **action** `ConducirAuto`) o eventos externos, no controlables (e.g., **action** `Choque`). No hay noción explícita de roles, de manera tal que si se necesitan el sujeto de la acción debe codificarse dentro de la misma (e.g., `ConducirAutoJuan` y `ConducirAutoPedro`).

Algunos requerimientos parecen a veces fáciles de formalizar, como por ejemplo el de tener registro para poder conducir un auto. Parecería que alcanza con prohibir la acción `ConducirAuto` si no hay una acción previa `ObtenerRegistro`. Esta facilidad es sólo aparente, ya que los individuos no sólo consiguen autorizaciones para conducir vehículos, también pueden perderlas, de manera que si también consideramos como posible a la acción `RevocarRegistro`, escribir una fórmula que establezca si una persona puede o no manejar tiene la misma complejidad que el problema de contar paréntesis para ver si están o no balanceados. Esto puede ser muy difícil o directamente imposible dependiendo de la lógica utilizada. En FL existe la noción de *intervalo*, similar a los *fluentes* de [8]. Un intervalo está delimitado por sus acciones de comienzo y fin, de manera tal que no hay anidamiento ni se puede cerrar un intervalo ya cerrado. Durante la vigencia de un intervalo se hace verdadera una variable proposicional, de manera tal que el caso del ejemplo se puede escribir como el intervalo

interval `autorizado_a_conducir` **delimited by actions**
`ObtenerRegistro-RevocarRegistro`

seguido de la fórmula $F(\neg is\_autorizado\_a\_conducir \wedge$ `ConducirAuto`$)$, donde el prefijo *is_* seguido del nombre del intervalo es una variable proposicional que se hace verdadera dentro del mismo.

Los intervalos también se pueden utilizar para acotar la ocurrencia de otras acciones:

interval `periodo_escolar` **delimited by actions** `ComienzoDeCursada-FinDeCursada`
action `RendirExamen` **occurs only in scope** `periodo_escolar`

Existe otra visión de la obligación, una donde lo obligatorio no es algo que debe suceder todo el tiempo, sino algo que indefectiblemente debe ocurrir, pero sólo una vez, en general dentro de un periodo de tiempo. Esta obligación se suele denominar *obligación no persistente*, y hay dos formas de representarla en FL. El operador $O^E(\varphi)$ establece la obligación de que en algún momento valga φ, dejando la obligación de existir una vez que se cumple. Si se desea considerar cotas temporales (e.g., "*Los libros de la biblioteca deben devolverse dentro del periodo escolar*"), los intervalos pueden usarse dentro de la obligación estándar: $O(\Diamond_{\texttt{periodo\_escolar}}\texttt{DevolverLibro})$ [5].

[5] En LTL el diamante ($\Diamond$) se interpreta como "*en algún momento en el futuro*". Los diamantes de FL, que incluyen el intervalo, como por ejemplo $\Diamond_{\texttt{periodo\_escolar}}$, se interpretan como "*en algún momento dentro del periodo escolar*".

FL también provee *contadores*, permitiendo utilizar expresiones como $O(\texttt{clp} > 0 \rightarrow \Diamond(\texttt{clp} == 0))$, donde la variable entera `clp`, *cantidad de libros prestados*, es incrementada ante cada `RetirarLibro` y decrementada con cada `DevolverLibro`. La fórmula se lee "*es obligatorio que cada vez que* `clp` *se haga mayor que cero, en algún momento posterior vuelva a ser cero*", y es interesante porque es la manera correcta de expresar que cada libro prestado debe ser devuelto, ya que la fórmula intuitiva –e incorrecta– $O(\texttt{RetirarLibro} \rightarrow \Diamond\texttt{DevolverLibro})$ es satisfecha al retirar varios libros y devolver sólo uno, que claramente no es lo deseado.

4 Caso de estudio

Para mostrar el poder de FL vamos a analizar un fragmento de un hipotético reglamento universitario. Utilizaremos aquí los conceptos de acción, intervalo, contadores, obligaciones, prohibiciones, permisos y varias formas de análisis de coherencia.

Este caso de estudio se centra en conflictos que tienen su raíz en los ayudantes-alumnos, es decir, alumnos que también son docentes. Muchas veces las reglamentaciones universitarias parecen olvidar a esta categoría intermedia, y marcan una diferencia tajante entre estudiantes y docentes, llevando a conflictos normativos como los que mostraremos. Si bien el caso es ficticio, la inspiración es real. Se realizaron algunas simplificaciones por cuestiones de espacio, pero no alteran el espíritu general.

El fragmento con el que trabajaremos es el siguiente:

1. Capítulo 1, Estudiantes.
   (a) Todo individuo que se haya inscripto a una carrera y que aún no se haya graduado en la misma es considerado un estudiante de esta Universidad.
   (b) Los estudiantes deben mostrar respeto mutuo. Las faltas disciplinarias graves se castigarán impidiendo el acceso a las instalaciones universitarias durante el año posterior a la falta.
   (c) Los estudiantes tienen los siguientes derechos: ..., participar en actividades de investigación, ...
2. Capítulo 2, Docentes.
   (a) Existen tres categorías docentes: c1) ayudantes-alumnos, c2) auxiliares, c3) profesores.
   (b) Los docentes se eligen por concurso, y los aspirantes deben anotarse en el mismo a partir de la fecha de apertura. La selección se realizará de acuerdo a los siguientes criterios: [omitidos por no ser relevantes para el caso de estudio].
   (c) Los aspirantes a ayudantes-alumnos deberán ser estudiantes al momento del concurso[6].

[6] La regla se concibió de esta manera dado que se desea que esta categoría esté destinada exclusivamente a alumnos, pero permitiendo que si se reciben en el medio del ejercicio del cargo no deban renunciar. En el caso real el puesto dura un año.

(d) Los docentes deben cumplir sus funciones a partir de los 30 días siguientes a la finalización del concurso en el que hayan resultado seleccionados.
(e) Si bien se permite el trabajo desde el hogar, se requiere que los docentes pasen al menos un día a la semana en las instalaciones de la Casa de Estudios.

3. Capítulo 3, Investigación.
   (a) Las actividades científicas sólo pueden ser realizadas por miembros de grupos de investigación.
   (b) Los grupos de investigación están conformados por profesores y auxiliares docentes.
4. Capítulo 4, Biblioteca de la Universidad.
   (a) Todo libro recibido en préstamo debe ser devuelto durante el mes en curso[7].
   (b) Estudiantes y docentes deberán pagar una multa si no devuelven los libros a término.
   (c) Debido a que el presupuesto del que disponen los estudiantes es módico, la multa aplicable a ellos deberá ser baja.
   (d) Se espera que los docentes sean un ejemplo de conducta. Por ese motivo, la multa aplicable a ellos será superior a la de los alumnos.

Modelemos primero el capítulo de los alumnos. Para abreviar, vamos a omitir, suponiendo ya escritas, las declaraciones de las acciones que limitan a los intervalos, como éste, que utilizaremos para definir qué es un estudiante:

interval `estudiante` **delimited by actions** `Inscribirse-Graduarse`

Con respecto a la disciplina, los alumnos no deben cometer faltas graves o se les prohibe el acceso durante un año. Para modelarlo definiremos dos intervalos. El primero abarca desde la falta hasta un año después. El segundo intervalo servirá para describir qué significa estar adentro del edificio universitario.

interval `prohibición` **delimited by actions** `CometerFalta-UnAñoDespués`
interval `dentro_del_edificio` **delimited by actions** `Entrar-Salir`

La prohibición se escribe así:

$$F_{\Diamond_{\texttt{prohibición}}(\neg \mathit{is\_dentro\_del\_edificio})}(\texttt{CometerFalta}) \tag{1}$$

y se lee como sigue: la falta no debe suceder, pero si ocurre, durante el periodo de prohibición el estudiante no puede estar adentro del edificio (*is_dentro_del_edificio* es una variable booleana que se hace verdadera entre las acciones que delimitan el intervalo `dentro_del_edificio`).

El artículo 1c permite a los estudiantes hacer investigación, en lo que podría pensarse como un *permiso antitético* [9]: un permiso cuya función es invalidar futuras prohibiciones:

[7] Si bien sería más realista estipular un plazo, la codificación formal se torna más larga y compleja, así que preferimos esta versión más simple por razones de espacio.

$$P(is\_estudiante \wedge \texttt{HacerInvestigación}) \tag{2}$$

Pasemos al concurso docente. Debemos modelar el plazo del concurso y sus posibles resultados: ser elegido en alguna de las categorías, o no resultar seleccionado.

action `ElegirGanadores` **output values** { doc_c1, doc_c2, doc_c3, no_seleccionado }
interval `concurso` **delimited by actions** `AbrirConcurso-ElegirGanadores`
action `Postularse` **only occurs in scope** `concurso`

Por simplicidad sólo modelaremos el requisito para la categoría c1 (ayudantes-alumnos) que consiste en todavía ser estudiante:

$$O(\Diamond_{\texttt{concurso}}(\texttt{Postularse} \rightarrow is\_estudiante)) \tag{3}$$

Los docentes tienen responsabilidades que comienzan 30 días después de la selección:

macro *docente* = `Postularse` $\wedge$
(`ElegirGanadores`.*doc_c1* $\vee$ `ElegirGanadores`.*doc_c2* $\vee$
`ElegirGanadores`.*doc_c3*)
interval `periodo_de_gracia` **delimited by actions** `ElegirGanadores-30DíasDespués`
interval `en_el_puesto` **delimited by actions** `30DíasDespués-+inf`
interval `semana` **delimited by actions** `ComienzaSemana-TerminaSemana`
occurs only in scope `en_el_puesto` **repeatedly**

Es obligatorio realizar al menos una visita semanal:

$$O(docente \rightarrow \Diamond_{\texttt{semana}}\texttt{Entrar}) \tag{4}$$

Con respecto al capítulo 3, la restricción de las actividades científicas a miembros de los grupos de investigación se escribe como:

$$F(\texttt{HacerInvestigación} \wedge \neg\texttt{UnirseAGrupoDeInvestigación}) \tag{5}$$

mientras que el requisito de ser profesor o auxiliar docente para estar en un grupo de investigación es:

$$F(\texttt{UnirseAGrupoDeInvestigación} \wedge \neg(\texttt{ElegirGanadores}.doc\_c2 \vee \texttt{ElegirGanadores}.doc\_c3)) \tag{6}$$

Debemos modelar también el préstamo de libros, similar tanto para alumnos como para docentes. Para eso utilizaremos un contador llamado `clp` (cantidad de libros prestados) y demarcaremos los meses. Hacemos notar que la duración exacta de los meses no es importante y por ende podemos abstraernos de ella.

counter `clp` **increases with action** `RetirarLibro`
decreases with action `DevolverLibro`
interval `mes` **delimited by actions** `ComienzaMes-TerminaMes` **repeatedly**

Aunque los artículos 4c y 4d tiene una función principalmente justificativa hay una consecuencia prescriptiva, incluso al nivel de abstracción en el que nos estamos manejando, y es que las multas deben diferenciarse. Para eso haremos uso del operador # que permite declarar incompatibilidades: `MultaParaAlumnos` # `MultaParaDocentes`.

El artículo 4b es una CTD para el 4a, así que los codificamos como:

$$O_{\texttt{MultaParaAlumnos}}(\mathit{is\_alumno} \rightarrow \Diamond_{\texttt{mes}}(\texttt{clp} > 0 \rightarrow \Diamond_{\texttt{mes}}(\texttt{clp} == 0))) \quad (7)$$

$$O_{\texttt{MultaParaDocentes}}(\mathit{docente} \rightarrow \Diamond_{\texttt{mes}}(\texttt{clp} > 0 \rightarrow \Diamond_{\texttt{mes}}(\texttt{clp} == 0))) \quad (8)$$

El análisis de FormaLex revela tres problemas de coherencia. Primeramente, la reparación en caso de no devolver los libros retirados de la biblioteca es controvertida para el caso de docentes que son también alumnos, y por eso la herramienta señala un caso de *reparaciones conflictivas*, ya que hay trazas donde el agente implícito es en efecto un docente y un alumno a la vez. Si miramos el SNBA, nada impide que los estudiantes se hagan docentes, más bien al contrario, dado que para eso existe la categoría de ayudante-alumno.

¿Qué tipo de multa se le debería aplicar a los ayudantes-alumnos que no devuelvan los libros a tiempo? Cualquier conclusión a la que se llegue mirando los artículos 4c y 4d, que son de carácter motivacional y están allí para dejar constancia del espíritu de la norma, es discutible. Es tan cierto que los ayudantes-alumnos deben ser un ejemplo de conducta en tanto miembros del cuerpo docente, como que tienen un presupuesto módico, ya que su estipendio es simbólico. Dado que la multa probablemente vaya a ser decidida por un empleado administrativo de la biblioteca, hay un alto riesgo de que diferentes soluciones sean aplicadas a casos idénticos. Sería mucho mejor que esta controversia fuera saldada en la misma norma, por quienes la promulgan. Éste es el sentido de la advertencia que emite la herramienta.

El segundo problema es más complicado y está relacionado con la norma que pide una visita semanal (art. 2e). La herramienta detecta que la reparación de la regla que prohibe las faltas disciplinarias (1) contradice la obligación a la visita semanal (4). En efecto, existe la posibilidad de que un estudiante cometa una falta, por ende se le prohiba entrar al establecimiento, se postule para ser docente, sea elegido como ayudante-alumno y no pueda cumplir con su visita semanal a la Universidad.

Una posible solución sería prohibir la postulación de estudiantes castigados:

action `Postularse` **only occurs in scope** `concurso` **requires that** $\neg$`CometerFalta`

De todas formas el problema persistiría, ya que el estudiante podría ahora postularse no habiendo aún cometido la falta, cometerla, y recién ahí ser elegido, produciéndose la misma situación. La solución real consiste en restringir la acción `ElegirGanadores` de manera tal que sólo estudiantes sin faltas puedan ser nombrados como ayudantes-alumnos:

macro *ser_elegido_ay-alumno* = `ElegirGanadores`.*doc_c1*

$$F(\texttt{CometerFalta} \wedge \mathit{ser\_elegido\_ay\text{-}alumno})$$

Esta fórmula declara que no está permitido cometer la falta y luego resultar elegido (o visto desde la perspectiva del jurado, elegir a alguien que cometió previamente una falta). Incluso con esta fórmula, podría suceder que alguien fuera nombrado en el cargo y luego cometiera la falta. En este caso tampoco podría cumplir con la reparación de no visitar la universidad durante un año, ya que eso iría en contra de su obligación como docente. Esta situación también es advertida por la herramienta, dado que la reparación ante la falta cometida se vuelve imposible de cumplir. Si el usuario considera que debería ser posible reparar la falta aún en este caso, es necesario que introduzca modificaciones en la reglamentación para contemplarla. Si, por otro lado, considera que es correcto que las reglas para los docentes sean más estrictas y que no esté contemplado que puedan reparar sus faltas, entonces el aviso de la herramienta puede ser ignorado.

El tercer problema es la colisión normativa entre permitir la investigación científica sólo a auxiliares y profesores (reglas 5 y 6) con el permiso para los alumnos de la regla 2. La solución es, o bien remover el permiso para los alumnos, o incluir al menos a los ayudantes-alumnos dentro de los grupos de investigación.

Este sistema normativo tiene otro aspecto interesante. Supongamos que alguien propusiese una manera distinta de expresar el artículo 2c. Una que supuestamente sea más fiel al espíritu de no permitir que los graduados ocupen las categorías de ayudantes-alumnos. La propuesta consiste en definir qué es un graduado:

interval `graduado` **delimited by actions** `Graduarse-+inf`

y prohibir directamente la postulación de graduados. Cuando se consulta a la herramienta por la validez de esta nueva escritura:

? $F(\Diamond_{\texttt{concurso}}(\texttt{Postularse} \wedge \textit{is\_graduado}))$

ésta responde que la prohibición no es válida, ya que hay trazas donde un estudiante se gradúa y luego se inscribe de nuevo (digamos, en otra carrera) antes de postularse. Esto perfectamente satisface el requerimiento de que durante el concurso los postulantes deban ser estudiantes (3), ya que el agente implícito es *también* un estudiante. La conclusión es que la forma en la que está escrita la regla no es consistente con el efecto normativo buscado: restringir la categoría de ayudante-alumno exclusivamente a los no graduados; de manera tal que la propuesta alternativa de regla es de hecho la correcta. Este "bug" fue extraído de un reglamento real de una Universidad Nacional. En el caso real, el resto de la normativa está inclinada a favorecer que los graduados se anoten en nuevas carreras, con lo que los encargados de sustanciar el concurso tampoco pueden basarse en usos y costumbres para impedir la inscripción de graduados, situación que en la práctica sucede.

5 Un llamado a la acción

¿Alcanza un caso de estudio ficticio para demostrar la validez de un enfoque? Sabemos ciertamente que no, pero nos gustaría que sirva otro propósito: el de

agitar la curiosidad y despertar el entusiasmo de nuestros colegas investigadores que entiendan que legislar es y seguirá siendo un proceso humano, sin por eso dejar de creer que las herramientas informáticas pueden ayudar a mejorar la calidad de las normas que producimos y que nos rigen.

No somos ingenuos: las grandes leyes de la Patria probablemente sean muy difíciles de formalizar, tanto por su contenido como por la natural falta de precisión que es necesario introducir para lograr suficiente consenso para aprobarlas. Pero no todas las normas son de este tipo. Hay muchísimas de un carácter mucho más operacional, cuyo texto se usa poco para litigar y mucho para guiar el accionar cotidiano de las personas. Hay ciertos tipos de problemas que ni siquiera imaginamos, y tal vez no haya mucho que podamos hacer contra ellos. Otros, sin embargo, sí los conocemos: incoherencias, casos no contemplados, ambigüedades, y creemos que flaco favor le hacen los legisladores a los legislados al no intentar evitar ese tipo de fallas, especialmente si se pueden detectar automáticamente.

Sabemos que el mecanismo no es perfecto ni completo. Tampoco lo es el *testing* que se emplea en la producción de software, que es otro proceso humano: a pesar de que no encuentra todas las fallas, hoy en día se considera negligente no someter a testing los sistemas informáticos antes de ponerlos a disposición del público, ya que si hay falencias que se pueden encontrar y corregir a tiempo, ¿por qué no hacerlo?

Una pregunta de investigación sumamente interesante es la costo-eficacia del proceso. La información que devuelve la herramienta, ¿guarda relación con el esfuerzo insumido en expresar la normativa como fórmulas? ¿no será mejor que un experto dedique la misma cantidad de tiempo a una revisión minuciosa de las mismas?

Son éstas las preguntas que queremos encarar, y para las cuáles convocamos a la comunidad: queremos analizar casos reales de normativas, y tomando eso como base entrar en el círculo virtuoso de investigación-desarrollo, donde las normativas analizadas sirvan para detectar los límites de la herramienta, de manera tal que al extenderse sea ésta la que ilumine los ángulos sombríos de las normas, para recomenzar el ciclo buscando extender la expresividad y encontrar más tipos de defectos con menos esfuerzo de codificación. Esperamos reencontrarnos en SID 2012 para relatar los devenires de este interesante trayecto.

Referencias

1. Gorín, D., Mera, S., Schapachnik, F.: Verificación automática de documentos normativos: ¿ficción o realidad? In: SID 2010, Simposio Argentino de Informática y Derecho, 39JAIIO. (2010) 2215–2229
2. Gorín, D., Mera, S., Schapachnik, F.: Model Checking Legal Documents. In: Proceeding of the 2010 conference on Legal Knowledge and Information Systems: JURIX 2010. (2010) 111–115
3. Gorín, D., Mera, S., Schapachnik, F.: A Software Tool for Legal Drafting. In: FLACOS 2011: Fifth Workshop on Formal Languages and Analysis of Contract-Oriented Software, Elsevier (2011) 1–15 Enviado. Una versión extendida está disponible en http://publicaciones.dc.uba.ar/Publications/2011/GMS11.

4. Pnueli, A.: A temporal logic of concurrent programs. Theoretical Computer Science **13** (1981) 45–60
5. Clarke, E., Emerson, E., Sifakis, J.: Model Checking: Algorithmic Verification and Debugging. Communications of the ACM **52** (2009) 74–84
6. Hansen, J., Pigozzi, G., van der Torre, L.W.N.: Ten philosophical problems in deontic logic. 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)
7. Vardi, M.Y.: Branching vs. linear time: Final showdown. In: TACAS 2001: Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, London, UK, Springer-Verlag (2001) 1–22
8. Giannakopoulou, D., Magee, J.: Fluent model checking for event-based systems. ACM SIGSOFT Software Engineering Notes **28** (2003) 266
9. Stolpe, A.: A theory of permission based on the notion of derogation. J. Applied Logic **8** (2010) 97–113