Universidad Nacional del Sur

Tesis de Doctor en Ciencias de la Computación

Generación automática de hipótesis vía razonamiento automatizado con aplicación a ciberseguridad

José Nicolás Paredes

Bahía Blanca Argentina

2021

Prefacio

Esta Tesis es presentada como parte de los requisitos para optar al grado académico de Doctor en Ciencias de la Computación, de la Universidad Nacional del Sur, y no ha sido presentada previamente para la obtención de otro título en esta Universidad u otras. La misma contiene los resultados obtenidos en investigaciones llevadas a cabo en el Departamento de Ciencias e Ingeniería de la Computación, durante el período comprendido entre abril de 2016 y noviembre de 2020, bajo la dirección del Dr. Marcelo A. Falappa, Profesor Asociado del Departamento de Ciencias e Ingeniería de la Computación y del Dr. Gerardo I. Simari, Profesor Adjunto del Departamento de Ciencias e Ingeniería de la Computación.

José Nicolás Paredes
jose.paredes@cs.uns.edu.ar
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Bahía Blanca, Marzo de 2021.

Agradecimientos

Me gustaría agradecer a todos los que fueron partícipes de alguna manera en este largo proceso de 5 años para lograr finalizar este doctorado. Por supuesto, en primer lugar, quiero agradecer muchísimo a mis directores Marcelo y Gerardo por confiar en mí, sin ustedes no hubiera podido conseguir este objetivo. Creo que fue un camino duro, siempre me tuvieron paciencia y me recibieron con muchísima predisposición desde el primer día. Valoro mucho más allá de sus conocimientos técnicos, el trato como persona que siempre tuvieron conmigo. Quisiera hacer un reconocimiento especial para vos Gerardo, ya que considero que muchas veces hiciste más de lo que te correspondía en tus obligaciones como director y siempre estuviste muy cerca en todo este camino, muchas gracias.

Por otro lado, también quiero agradecer a Vanina porque más allá de que no haya cumplido un rol formal en la dirección de la tesis, fuiste muy importante en los trabajos que llevamos a cabo y en la práctica también en cierta medida cumpliste un rol de dirección.

Obviamente no puedo olvidarme de agradecer a todos los chicos de la salita/oficinas anexas con los cuales compartí todos estos años: Rama, Mario, Mati, Charly, Fede, Martin, Ro, Noni, Gotti, Lu, Dana, Maiso, Vir, Fabio, Jime, Juli, Anita, Ceci, Lean, Anto y Santi. Muchísimas gracias por las charlas de siempre en la salita/pabellón y los almuerzos en el comedor (aunque algunos piensen que es una sala de reuniones). Son todos grandes profesionales y excelentes personas, realmente siempre me sentí uno más y valoro mucho haber compartido con ustedes porque considero que para crecer siempre hay que rodearse de personas mejores que uno, muchas gracias en serio. También quiero incluir en los saludos a Fulla y Harry con los que no llegué a compartir como compañero pero si pude conocerlos por transitividad con compañeros de beca de ustedes que luego fueron compañeros míos, gracias por la buena onda siempre; realmente es muy valioso cuando estás en una ciudad nueva y no conoces a nadie.

En especial agradezco por las salidas, el aguante y las juntadas de siempre a Charly, Ariel, Mario, Mati, Rama, Fede, Brown, Maiso, Vir y Noni. Hicieron que mi vida sea más fácil todos estos años y me llevo un montón de recuerdos, anécdotas y grandes momentos. Además, quiero hacer un reconocimiento especial a Charly porque fue quien me recibió en mi llegada a bahía, me recibiste en tu casa casi sin conocerme, creo que sos una gran persona y un gran amigo. Tampoco quiero olvidarme de vos Ariel, casi no compartimos tiempo en el DCIC porque ya te habías recibido, pero me caíste muy bien desde el primer día, sos una gran persona y te considero un amigo; todavía no entiendo cómo podés hacer tantas cosas bien (tocar la guitarra, cocinar bien, investigar, etc), lo cual me genera envidia y admiración.

Tampoco quiero olvidarme de los amigos que hice en bahía por fuera de la universidad: Clari, Diego, Anto, Juli, Juan, Lu y Juancho. Muchísimas gracias por tantos momentos, salidas y juntadas geniales, son personas extraordinarias, sin dudas son parte importante de esta gran experiencia.

Quiero agradecer también a todos los que forman parte del Departamento de Ciencias e Ingeniería de Computación, ya que fui muy bien recibido desde el primer día, me brindaron el lugar y las herramientas para poder desarrollar mi trabajo, y confiaron en mí a pesar de haber hecho mi carrera de grado en otra universidad. Aprovecho también para agradecer a la Universidad Nacional de Entre Ríos, la cual fue una parte importantísima para mi formación, así como a todos los profesores que formaron parte de ese camino inicial. En especial quiero agradecer a dos profesores que fueron parte de esa formación de grado, Cristian Pacífico y Mónica Tugnarelli. Cristian muchas gracias por confiar en mí, por ser incondicional y por alentarme a tomar este camino el cual tampoco tenía mucha idea cómo era. Mónica muchas gracias por confiar en mí desde siempre, valoro mucho que realmente siempre quisiste lo mejor para mí. Mónica y Cristian, ambos son personas increíbles y excelentes docentes, aunque parezca exagerado los considero amigos. También agradezco a los amigos con los que compartí la carrera de grado: Kevin, Guille, Anto y Rodo. Ustedes también son parte de esto, agradezco todos los momentos que compartimos, muchas gracias por todo!!!

Por supuesto, necesito agradecer al Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) por haber confiado en mi capacidad y financiar mi formación de posgrado por todos estos años, lo cual hubiera sido muy difícil de otra forma.

Por último, quisiera hacer los agradecimientos más importantes de todos, mi familia, que siempre estuvo a pesar de todo. Agradezco a cada uno de mis hermanos: Agu, Gera, Alfredo, Salo, Carlitos y también Andrea. Agu, Gera, Alfredo, Salo y Carlitos con ustedes compartí muchísimas cosas que sólo nosotros conocemos y ustedes saben que haría cualquier cosa que necesiten. Le dedico parte de esta tesis a ustedes y a mis sobrinos: Fran, Mica, Benja y Abel. Agradezco muchísimo a mis padres por toda la formación como persona, estoy seguro que ustedes quieren lo mejor para mí. En especial quiero agradecerte a vos mamá porque sé que siempre hiciste todo y más para que mis hermanos y yo estemos bien y seamos mejores, tenés una fuerza de voluntad inigualable y sos muy inteligente, esta tesis está especialmente dedicada para vos.

Muchísimas gracias a todos!!!

Resumen

Esta tesis se enfoca en el estudio de sistemas para la generación automatizada de hipótesis contextualizadas en la detección de comportamiento malicioso en plataformas sociales. Como primer aporte de esta tesis, se presenta una aproximación inicial para un sistema como el mencionado anteriormente y se consideran dos enfoques ligeramente diferentes acotando la detección de comportamiento malicioso a un tipo de problema específico bautizado como deduplicación adversarial. Para el primer enfoque, se pone mayor énfasis en la generación de hipótesis a partir de la utilización de reglas lógicas bien definidas, aunque la esencia de su funcionamiento está apoyada en los resultados que puedan ser obtenidos de aplicar técnicas de aprendizaje automatizado con anterioridad. Luego para el segundo enfoque, se realiza mayor hincapié en la utilización de técnicas de aprendizaje automatizado, específicamente clasificadores, como estrategia para atacar el problema y la generación de hipótesis es llevada a cabo por reglas más simples que son activadas cuando el resultado de los clasificadores supera cierto umbral.

Sin embargo, el objetivo general de esta tesis es avanzar hacia el desarrollo de sistemas más robustos que no se encuentran acotados a un solo problema de comportamiento malicioso en plataformas sociales, sino que considere la multiplicidad de los mismos y aproveche la relación que pueda haber entre ellos. Por esta razón, el principal aporte de esta tesis es la presentación de la arquitectura NetDER para razonar sobre comportamiento malicioso en plataformas sociales, la cual en principio, busca servir de guía para la implementación de software en dicho dominio. Asimismo, en esta misma dirección, otro aporte realizado es el estudio de los fundamentos teóricos involucrados en la implementación de una versión particular de NetDER. Más específicamente, la generación de hipótesis está apoyada en un proceso conocido como de respuesta a consultas, por lo cual fue necesario investigar su incidencia en este modelo, y a partir de dicho estudio se llega a un interesante conjunto de resultados que varían de la tratabilidad del tiempo

polinomial a la indecidibilidad, dependiendo de las características que estén disponibles. Adicionalmente, se desarrolla un caso de uso para ilustrar cómo el enfoque puede ser aplicado en un dominio de ciberseguridad para razonar sobre productos en riesgo basados en publicaciones de foros de la *Darknet*.

Finalmente, como último aporte se realiza una evaluación experimental de la arquitectura NetDER, considerando las cuestiones de diseño y fundamentos teóricos estudiados a lo largo de esta tesis. Asimismo, debido a la dificultad de obtener *datasets* adecuados con *ground truth*, lo cual es necesario para llevar adelante evaluaciones de desempeño, fue necesario desarrollar un *testbed* general (dejando disponible públicamente su código) diseñado con el propósito de generar trazas completas de actividades de publicación involucrando potencialmente todo tipo de contenido malicioso como lo pueden ser *noticias falsas*, actores *maliciosos*, *botnets*, enlaces a *malware*, discursos de odio, etc. Los resultados obtenidos fueron satisfactorios, debido a que en general son estadísticamente significativos y constituyen un paso importante para avanzar al logro del objetivo general que es disponer de sistemas robustos de generación automatizada de hipótesis que puedan utilizarse para resolver problemas de comportamiento malicioso en plataformas sociales.

Abstract

In this thesis we focus on the study of systems for the automated generation of hypotheses in order to detect malicious behavior on social media. The first contribution of this thesis is the development of an initial approach for a system such as the one we mentioned above, where two slightly different approaches are considered, limiting the detection of malicious behavior to a specific kind of problem called adversarial deduplication. For the first approach, greater emphasis is placed on the generation of hypotheses from the use of well-defined logical rules, although they are essentially based on the results that can be obtained from the prior application of machine learning techniques. Then, for the second approach, greater emphasis is placed on the use of machine learning techniques, specifically classifiers, as a strategy to attack the problem and the generation of hypotheses is carried out by simpler rules that are activated when the result of the classifiers exceeds a certain threshold.

The general objective of this thesis is however to advance towards the development of more robust systems that are not limited to a single problem of malicious behavior on social media, but rather consider their multiplicity and take advantage of the relationship that may exist between them. For this reason, the main contribution of this thesis is the presentation of the NetDER architecture to reason about malicious behavior on social media, which in principle seeks to serve as a guide for the implementation of software in this domain. Also, in this same direction, another contribution is the study of the theoretical foundations involved in the implementation of a particular version of NetDER. More specifically, the generation of hypotheses is supported by a process known as query answering; therefore, we need to research its incidence in this model, and from this study an interesting set of results is reached that vary from polynomial-time tractability to undecidability, depending on the features that are available. Additionally, a use case

is developed to illustrate how the approach can be applied in a cybersecurity domain to reason about at-risk products based on *Darknet* forum posts.

Finally, as a last contribution, an experimental evaluation of the NetDER architecture is carried out, considering the design issues and theoretical foundations studied throughout this thesis. Also, due to the difficulty of obtaining adequate datasets with ground truth, which is necessary to carry out performance evaluations, it was necessary to develop a general testbed (making its code publicly available) designed with the purpose of generating complete traces of posting activities potentially involving all types of malicious content, such as fake news, malicious actors, botnets, links to malware, hate speech, etc. The results obtained were satisfactory, because in general they are statistically significant and constitute an important step to advance towards the achievement of the general objective, which is to have robust systems for the automated generation of hypotheses that can be used to solve problems related to malicious behavior on social media.

Índice general

Capítulo 1

Introducción

1.1. Motivación

Existen una gran cantidad de problemas prácticos donde puede ser muy adecuado abordarlos utilizando enfoques de razonamiento basados en principios bien definidos, siendo objeto de estudio de áreas de investigación como la intersección de Inteligencia Artificial y Base de Datos. En este sentido, en muchas ocasiones, realizar este tipo de razonamiento involucra responder consultas de acuerdo a los datos disponibles, lo cual en términos más específicos puede utilizarse como base para un *Sistema de Generación Automatizada de Hipótesis*. Ésto puede ser útil en escenarios donde un experto debe considerar múltiples fuentes de información, llevar a cabo un razonamiento, y luego elaborar una conclusión para poder tomar decisiones. Como ejemplos de tales dominios se puede mencionar a un médico cuando evalúa la información disponible antes de aseverar uno o más diagnósticos, un auditor gubernamental cuando recopila información en busca de posibles fraudes en el uso de fondos públicos, o a un analista de ciberseguridad que realiza esencialmente el mismo proceso pero con diferentes datos cuando busca al culpable de un ciberataque.

Todos estos escenarios con estas características son muy interesantes como objeto de estudio; sin embargo, cuando adicionalmente se debe tratar con información que se propaga a través de una red, el entorno de interés se vuelve aun más atractivo y desafiante. Teniendo en cuenta estas propiedades, uno de los dominios de aplicación con mayor notoriedad en la actualidad son las *Redes Sociales* debido principalmente al rol cada vez más preponderante que ocupan las mismas en nuestras vidas, otorgándonos una nueva

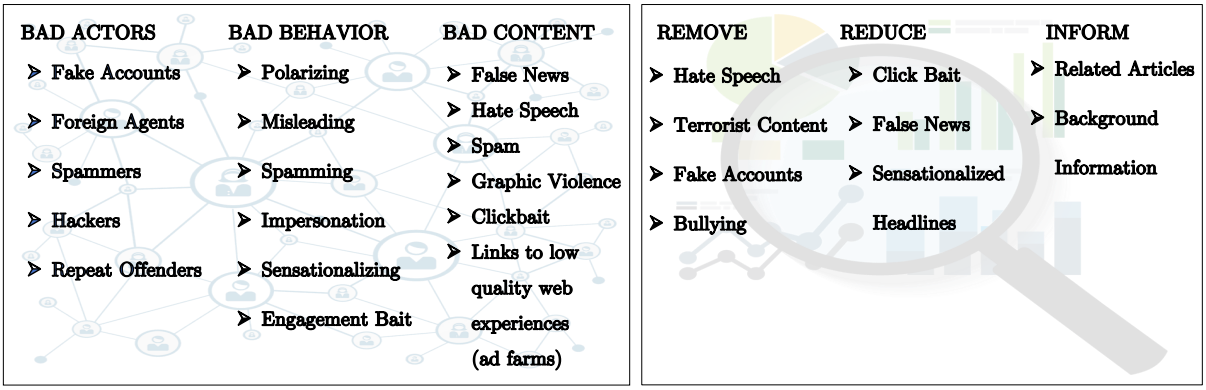

Figura 1.1: Una posible clasificación de problemas relacionados con Comportamiento Malicioso en Plataformas Sociales (izquierda), y de posibles estrategias de acción en base a los problemas detectados (derecha). Figura construida en base a [Fac18a, Fac18b]

capacidad para compartir información (incluyendo cuestiones personales) y conocimiento de una manera fácil y rápida, lo cual la convierte en una herramienta muy poderosa y con alcances que sobrepasan nuestra imaginación. Hoy en día, las personas pasan gran parte de su tiempo en estas vidas virtuales y, asimismo, obtienen todo tipo de información a través de este mundo virtual, muchas veces utilizándola para tomar decisiones o al menos formar opinión (que en el futuro puede desembocar en una decisión) en relación a diversos aspectos como política, economía, seguridad, gastronomía, entretenimiento, y muchos otros. Como consecuencia de su rol cada vez más central y al simple hecho de que como toda herramienta puede ser utilizada sin intenciones de provocar algún perjuicio o, por el contrario, con la intención de provocar algún tipo de daño, surgen nuevos problemas vinculados a comportamientos no deseados en este nuevo mundo, los cuales nosotros llamamos en forma general *Comportamiento Malicioso en Plataformas Sociales*.

La Figura 1.1 está basada en información publicada por Facebook [Fac18a, Fac18b] respecto a su visión de los tipos de problemas que pueden surgir en plataformas como la que ellos ofrecen, y las diferentes estrategias que ellos proponen para mitigar efectos dañinos una vez que sean detectados. Se puede observar la gran variedad de dimensiones de este problema general, complejo y desafiante, por lo cual particularmente Facebook (como otras empresas) se ve en la obligación de tomar medidas al respecto como la creación de diferentes equipos de trabajo dedicados a su identificación. Con este objetivo, la estrategia que siguen consta de 3 partes: a) remoción de cuentas y contenido que violan

sus términos y condiciones, b) reducción de la distribución de noticias falsas y contenido engañoso como *clickbait* [1], c) proporcionar información adicional a las personas con el fin de contextualizar las publicaciones que visualizan. En relación a la parte a) de la estrategia planteada, detallan que, si bien las cuentas que difunden noticias falsas no violan sus términos y condiciones, con frecuencia si son violados en otras categorías, tales como *spam*, discursos que promueven odio, o cuentas falsas, en cuyo caso se consideran plausibles de ser removidas. Por ejemplo, una página que dice ser administrada por estadounidenses y en realidad es administrada desde otro país se considera como una identidad falsa por lo que se elimina junto con todo el contenido, el cual probablemente haya tenido noticias falsas. Por otro lado, la parte b) apunta a penalizar comportamientos que se encuentran en una zona gris respecto a la violación de las reglas de la plataforma, por ejemplo, *clickbait*, enlaces compartidos más frecuentemente por *spammers*, enlaces a páginas web de baja calidad, páginas o sitios web que comparten repetidamente noticias falsas, entre otros. Por último, la parte c) implica admitir que no es factible eliminar completamente comportamientos no deseados, por lo cual se busca desarrollar herramientas que ayuden a las personas a decidir qué pueden leer con cierto nivel de confianza, y en caso de considerarlo incluso si merece ser compartido. Por ejemplo, se menciona la capacidad de utilizar verificadores de hechos (*fact checkers*) a cargo de terceros como fuente de artículos relacionados a algún tema, de manera que sean mostrados en publicaciones cuando sea oportuno. La Figura 1.2 nos permite obtener un panorama del número de acciones de remoción llevadas a cabo por Facebook por mes. Estas acciones son una de las más radicales que pueden tomarse, por lo que se requiere evidencia bastante contundente para pagar el menor costo posible. Facebook aclara que si bien llevan a cabo tareas de investigación y las acciones que correspondan contra todo tipo de comportamiento malicioso, como alteración artificial de estadísticas de perfiles (*fake engagement*), *spam*, y amplificación artificial, en estos reportes solo se enfocan en lo que llaman "Comportamiento No-Auténtico Coordinado"(*Coordinated Inauthentic Behavior*) que se caracteriza por ser más complejo de detectar por constituirse de una red de cuentas coordinadas que ejecutan alguna operación de influencia con objetivo de manipular el debate público. *Cabe destacar que la forma en que Facebook lleva adelante estas tareas está basada en una combinación de herramientas de aprendizaje automatizado (*machine learning*) y reglas codificadas ad-hoc de acuerdo a la tarea específica de interés.* [MIT20]

[1] Contenido con título o imagen diseñados para tentar al lector a hacer click e ingresar, típicamente con el objetivo de que se carguen publicidades asociadas o realizar acciones de phishing.

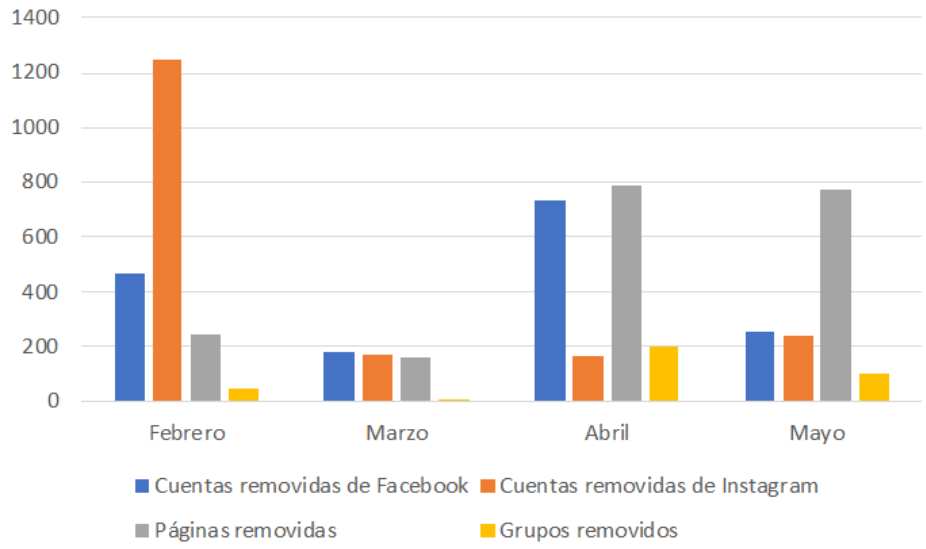

Figura 1.2: Cantidad de páginas, grupos, y cuentas de Facebook e Instagram removidas entre Febrero y Mayo de 2020. Figura construida en base a [Fac20d, Fac20a, Fac20c, Fac20b]

A continuación, se ahondará con más detalles en algunos de los más conocidos de estos problemas.

1.2. Información falsa

El término *información falsa* es bastante general, por lo que para tener un panorama más claro acerca de lo que nos estamos refiriendo podemos tomar como referencia la categorización propuesta en [ZSBK19] basada en 8 tipos diferentes:

- Fabricadas: Historias completamente ficticias desconectadas en su totalidad de hechos reales. No es un tipo nuevo y existe desde el comienzo del periodismo; algunas de las más conocidas combinan en sus historias políticos y extraterrestres, como la historia de que Hillary Clinton adoptó un bebé extraterrestre.
- Propaganda: Este es un caso especial de historias fabricadas cuyo objetivo es perjudicar el interés de un partido político particular y usualmente tienen un contexto político. Este tampoco es un tipo nuevo sino que fue ampliamente utilizado durante

la Segunda Guerra Mundial y la Guerra Fría. Las consecuencias de este tipo de información falsa son tales que pueden cambiar el curso de la historia, por ejemplo, cambiando el resultado de una elección presidencial.

- Teorías conspirativas: Historias que intentan explicar una situación o un evento apoyándose en alguna conspiración, la cual por definición no ha sido probada. Con frecuencia, estas historias son acerca de actos ilegales que son llevados a cabo por gobiernos o individuos poderosos. Entre los ejemplos más conocidos se puede mencionar la teoría de *Pizzagate*, donde se acusa al jefe de campaña de Hillary Clinton de estar involucrado en una red de abuso de menores [Wik17b] o conspiraciones acerca del asesinato de Seth Rich, quien estuvo relacionado a las filtraciones de e-mails de la DNC en EEUU [Wik17a].

- Engaños: Noticias que contienen hechos que son falsos o imprecisos y son presentados como legítimos, y también suelen llamarse *verdad a medias* [MW] o *factoid* [Tim]. Los ejemplos más conocidos en esta categoría se relacionan muchas veces con muertes falsas de celebridades, como por ejemplo la muerte de Adam Sandler [Sno17a].

- Sesgadas: Historias que se encuentran extremadamente sesgadas, es decir, solo es contada desde un punto de vista específico, y en particular para contextos políticos se conoce como *noticias ultra-partidarias*, donde se direcciona el foco hacia una persona/partido/situación/evento. Como ejemplo se puede mencionar la amplia difusión de comunidades de la *derecha alternativa* usando pequeñas comunidades web como 4chan [HODC+16].

- Rumores: Historias cuya veracidad es ambigua o nunca fue confirmada; asimismo se puede decir que este tipo es uno de los más comunes y en consecuencia ha sido ampliamente estudiado. Como ejemplo, se puede mencionar historias alrededor de las bombas en la Maratón de Boston de 2013 asegurando que los sospechosos se hicieron ciudadanos el 9/11 [Sno17b].

- Clickbait: Esta categoría hace referencia al uso deliberado de titulares y miniaturas de contenidos en la web en forma engañosa. Nuevamente, tampoco es algo novedoso sino que ya ha sido utilizado años atrás, durante "la era de los periódicos", produciendo un fenómeno conocido como periodismo amarillista [Cam01]. Sin embargo, la diferencia actual es su utilización en redes sociales en la forma de descriptores del

contenido publicado que son agregados con la intención de engañar para incrementar su tráfico con fines de lucro o popularidad [Pol17]. En cierta medida, se puede considerar uno de los tipos menos severos de información falsa, porque si un usuario lee o ve el contenido entero entonces puede distinguir si el titular y/o la miniatura era engañosa.

- Noticias satíricas: Historias que contienen mucha ironía y humor, y es uno de los tipos que ha capturado considerable atención en la web en los últimos años; sin embargo, como sus artículos son generalmente difundidos a través de redes sociales este hecho, en ocasiones, es obscurecido, pasado por alto o ignorado por los usuarios y se corre riesgo de que se lo valore literalmente sin ninguna verificación adicional. Como ejemplo se puede mencionar sitios como TheOnion [Oni], y SatireWire [Sat] donde se publica este tipo de contenido.

Un tipo particular de información falsa son las llamadas *Noticias Falsas*, que tienen sus orígenes en el género literario satírico y se caracterizan generalmente por ser publicados en sitios web y luego ser difundidos en redes sociales. Se las puede definir como una publicación online en relación a aseveraciones de hechos de las que se tiene conocimiento de su falsedad [KW17]. Este término comenzó a ser ampliamente conocido cuando se pudo determinar que durante las elecciones presidenciales de EEUU en 2016 se produjo una difusión descontrolada de este tipo de noticias principalmente a través de Facebook. En [Buz16] se muestra que en el final de los últimos 3 meses de campaña las noticias falsas con mayor popularidad referidas a la elección generaron más interacciones (*engagement*) que las noticias más populares de los principales medios de comunicación tales como el New York Times, Washington Post, Huffington Post, NBC News y otros (ver Figura 1.3). Específicamente, durante los últimos 3 meses teniendo en cuenta solo noticias referidas a las elecciones, las 20 historias falsas más populares de sitios engañosos y blogs ultra-partidarios generaron 8.711.000 reacciones, comentarios, o fueron compartidas en Facebook; en contraste, las 20 historias de los 19 principales sitios web de noticias generaron un total de 7.367.000 reacciones, comentarios, o fueron compartidas en dicha plataforma. Es así como hasta los últimos 3 meses de la campaña el contenido en cuestión de los principales medios superó fácilmente al de las noticias falsas, pero a medida que se acercaron las elecciones, las interacciones vinculadas a noticias falsas se disparó y la relación mencionada anteriormente se invirtió.

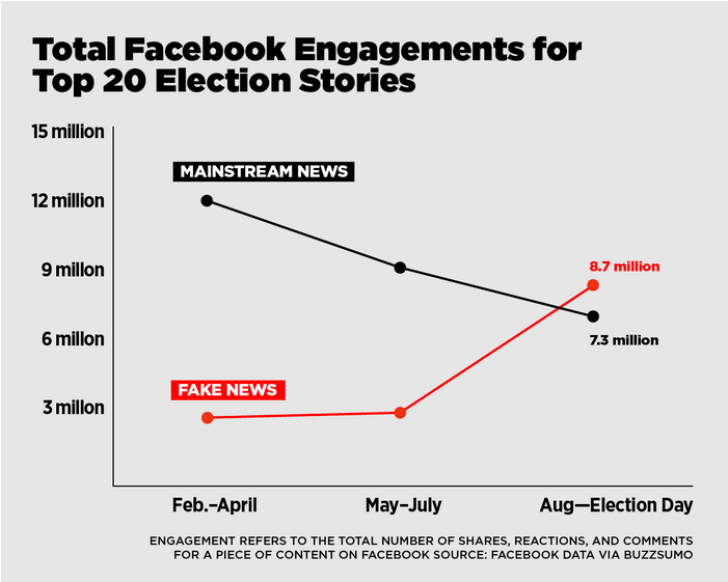

Figura 1.3: Representación de la cantidad de interacciones generadas por noticias falsas respecto a portales de noticias reconocidos durante los meses previos a la elección presidencial de EEUU en el año 2016. Figura extraída de [Buz16].

Por otro lado, como es de esperarse, Twitter no estuvo ajeno al asunto y también fue utilizado como medio de propagación de este tipo de contenido [Qua17]. Esto puede desprenderse del análisis hecho en [HKBN18] usando datos de dicha plataforma, dando como resultado que las noticias falsas estuvieron casi tan presentes como las noticias verosímiles durante los días anteriores y posteriores al evento electoral. Los autores de [HKBN18] llegaron a dicha conclusión luego de considerar 7 millones de tweets publicados en EEUU durante los primeros 11 días de noviembre, teniendo en cuenta que la elección fue el 8 de ese mes. Aproximadamente un millón de esos tweets incluían enlaces a contenido político, 256.725 de los cuales apuntaban a medios de comunicación creíbles y 203.591 a fuentes de noticias falsas. Como consecuencia de esto, y basado en la combinación de enlaces a fuentes falsas, WikiLeaks, y medios de comunicación rusos, los usuarios estadounidenses de Twitter tuvieron acceso a más contenido conspiracional, polarizante y con información errónea que a noticias producidas en forma profesional (ver Figura 1.4).

La propagación de información falsa puede traer las consecuencias menos esperadas, como paranoia y acciones desmedidas basadas en información no oficial y sin ningún tipo de confirmación. Como referencia de tales situaciones se puede mencionar casos de personas que fueron erróneamente señaladas como portadoras de COVID-19, derivando

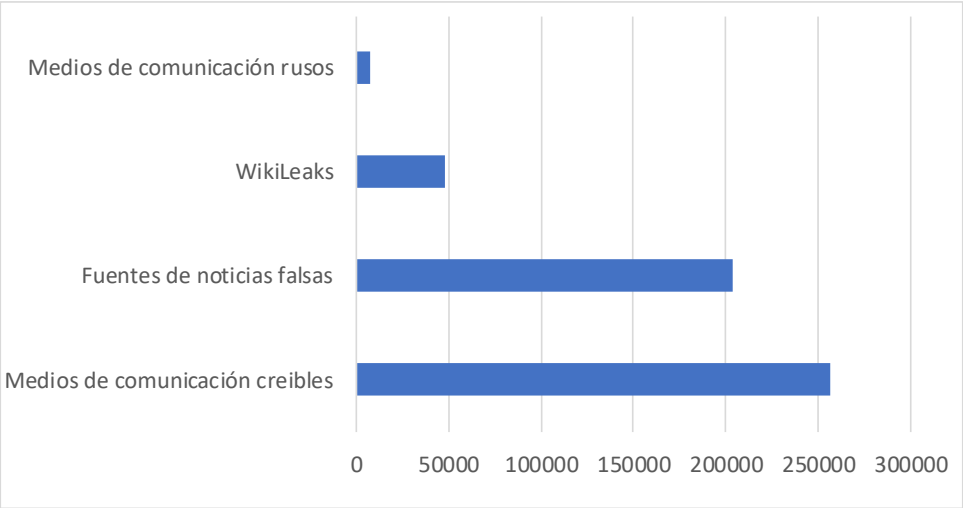

Figura 1.4: Total de enlaces encontrados en una muestra de tweets publicados del 1 al 11 de Noviembre de 2016. Figura construida en base a [Qua17].

en todo tipo de amenazas [Inf20]; personas que generan confusión divulgando información que se enuncia como oficial pero en realidad no lo es como es el caso de un anuncio de cuarentena cuando todavía no había sido dispuesta [Pag20]; personas que ocasionan daños económicos directamente a empresas propagando información falsa sobre cierre debido a casos positivos de COVID-19 [LaN20], entre muchos otros ejemplos.

Se puede notar que es un gran problema, complejo y con muchas aristas; sin embargo, se están realizando muchos esfuerzos para atacar estas cuestiones. Sitios como "https://www.politifact.com/", "https://www.snopes.com/", o "https://confiar.telam.com.ar/" son muestra de algunos de esos esfuerzos.

1.3. Bots y botnets

Los *bots* en sí mismos no son un problema de por sí; por el contrario, son parte de la automatización que permite a las empresas escalar eficientemente para brindar un mejor servicio a sus clientes. En este sentido, el mundo del desarrollo y el despliegue de aplicaciones puede considerarse un emblema, ya que aprovechan enfoques como *infraestructure as code* y *frameworks* nativos de la nube para desplegar rápidamente y escalar automáticamente su base de aplicaciones (*app footprint*) para satisfacer los incrementos en la demanda de los clientes pero reduciendo los costos. Desde el punto de vista del

usuario final o cliente, es muy posible que hayamos utilizado alguna vez o utilicemos a diario *bots* como lo son Siri, Alexa o Google Assistant, para encontrar productos, servicios e información para facilitarnos la vida. Otro ejemplo es cuando interactuamos con *chatbots* de sitios web para encontrar un pasaje aéreo, informarse sobre las condiciones de un préstamo, y muchos otros bienes y servicios valiosos. Al fin y al cabo, los *bots* son esencialmente piezas de código programados para proporcionar rápidamente información o algún consejo bajo ciertas condiciones o datos de entrada.

Sin embargo, las organizaciones que proporcionan productos y servicios a través de la web o la publicación de datos a través de *APIs* (*application programming interface*) muchas veces se topan con el problema de que se produzcan demasiadas solicitudes web defectuosas generadas por *bots*. Esto se encuentra ligado al crecimiento de la popularidad de ataques automatizados a los mismos, como por ejemplo la apropiación de cuentas a través de técnicas de *ciberataques* como "relleno de credenciales" (*credential stuffing*). En la actualidad, más y más actores maliciosos están utilizando *bots* automatizados para cometer fraudes y avanzar hacia sus objetivos malintencionados, y se previó para el año 2020 que sólo en EEUU dichos actores maliciosos causarían 12 mil millones de dólares en pérdidas. Esto sólo tomando en cuenta el aspecto financiero que es el más directo a lo cual debería sumarse las experiencias malas de los clientes que producen daños a la marca y reputación de una empresa. Como se dijo anteriormente, no todo el tráfico web automatizado por *bots* es malicioso, por lo que es necesario distinguir los "buenos" de los "malos" (ver Figura 1.5). A continuación, se detallan algunos ejemplos de *bots*, en base a [Sig19], que llevan a cabo una variedad de funciones que pueden ayudar a las empresas y a sus clientes:

Chatbots. Interactúan con humanos a través de texto o audio permitiendo la comunicación para ayudarlos a acelerar tareas y objetivos orientados a información. Google Assistant y Alexa son quizás los ejemplos más visibles desde el punto de vista del usuario, siendo dichas aplicaciones las encargadas de llevar a cabo tareas como reaccionar a preguntas de audio, encontrar respuestas y datos, y luego presentarlos para que los usuarios puedan realizar tareas comunes, como buscar una película, comprar entradas y luego encontrar estacionamiento cerca del cine. Detrás de escena, las APIs trabajan para procesar las preguntas o los comandos que se realizan en lenguaje natural y luego obtienen los datos necesarios en respuesta a las consultas.

Shopbots. Recorren Internet en busca de precios bajos en los artículos de interés

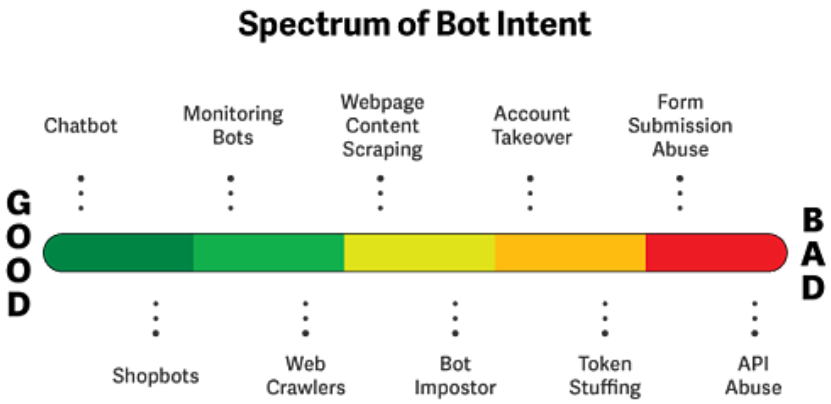

Figura 1.5: Representación del espectro de *bots* de acuerdo a qué tan buenas o malas son sus intenciones. Figura extraída de [Sig19].

para un determinado usuario, los cuales luego se potencian a través de otros medios como *motores de búsqueda* o *sitios web de agregación* (por ejemplo, TripAdvisor) y comercio electrónico. Generalmente, se paga una tarifa o comisión de suscripción a los operadores de *shopbots* que generan tráfico y, por lo tanto, transacciones al sitio de compra.

Monitoring bots. Verifican el estado, la disponibilidad, y la capacidad de respuesta de los sitios web; tal es el caso de `isitdown.com`, un sitio independiente que proporciona información del estado de tiempo real, incluyendo interrupciones, de sitios web y otros tipos de servicios.

Web crawlers. Son desplegados por motores de búsqueda tales como Google que usan estos *bots* para indexar contenido de sitios web con la intención de que produzcan resultados de búsqueda efectivos para los usuarios. También son conocidos como *arañas web* o *bots de Internet*, cuya función es navegar por la web de forma automatizada para buscar contenido y analizarlo tomando en cuenta datos como la frecuencia de palabras clave, enlaces en una página, enlaces rotos, mapas del sitio y validación de código HTML.

En el otro extremo tenemos *bots* que fueron programados con intenciones maliciosas, en cuyo caso son diseñados para ejecutar código repetidamente y generar tráfico web por sí mismos desencadenando grandes cantidades de solicitudes web contra páginas de inicio de sesión, de comerciantes minoristas, bancos o cualquier organización que almacene datos personales o financieros valiosos y los haga accesibles a través de un aplicación web o API.

A continuación, se detallan algunos escenarios con una característica común: permitir a actores maliciosos lograr algún objetivo fraudulento:

Account takeover. Cuando se producen violaciones de seguridad, el resultado suele ser la adquisición de grandes cantidades de credenciales de usuario. Estas credenciales son publicadas en la *DarkWeb* (para más detalles ver Sección 1.4), donde los actores maliciosos pueden comprarlas y luego usar *bots* automatizados para probar rápidamente los nombres de usuarios y contraseñas en los flujos de autenticación de los principales sitios minoristas y financieros de consumo, lo cual se conoce como relleno de credenciales. Una vez que se encuentran las credenciales válidas, se utilizan para apropiarse de las cuentas del sitio web y bloquear usuarios legítimos. Asimismo, los atacantes adquieren la *información de identificación personal* (*PII*, siglas del inglés) y métodos de pago almacenados en dichas cuentas para llevar a cabo todo tipo de fraude, desde la creación de nuevas cuentas con la *PII* robada, hasta la compra de bienes y servicios con la información de pago almacenada.

Token stuffing. Ocurre cuando *bots* se apoderan de pedidos hechos por clientes basándose en técnicas como adivinar por fuerza bruta IDs de token de la orden y; luego, cuando se han apropiado del carrito de compras, pueden redirigir el pedido para que se envíe a otro lugar.

Form submission abuse. Los bots abusan de los formularios en sitios web accesibles públicamente por una variedad de razones maliciosas, como por ejemplo pueden realizar *SQL injection*, escaneos, o publicación de contenido no deseado a través de estos formularios.

Webpage content scraping. Está claro que crear contenido web cuesta dinero; sin embargo, no todos quieren pagar por dicho contenido valioso, por lo que, en lugar de eso, algunos estafadores despliegan *bots*, con frecuencia aparentando ser *web crawlers*, para recolectar el contenido de la página web para su uso en otros lugares.

Bot imposter. Similarmente, otras solicitudes web maliciosas automatizadas se hacen pasar como si fueran hechas por un *bot* de búsqueda de Google o Bing dentro de un contexto de comercio electrónico con el objetivo de que estos *bots* impostores puedan recopilar datos de precios e inventario para obtener una ventaja competitiva en el mercado.

API abuse. Se puede decir que las APIs constituyen una pieza clave de la web moderna al permitir que las organizaciones intercambien datos, gran parte de ellos muy sensibles. Aprovechando esta situación se convierten en objetivos de distintos ataques

como es el caso de los *bots* automatizados que se implementan para sondear y extraer datos de las APIs. Por ejemplo, los adversarios pueden falsificar la información del *encabezado XFF* para ejecutar ataques de adquisición de cuentas contra portales orientados a clientes (*customer-facing portals*).

En términos más específicos, situándonos en un contexto de información falsa, los bots son programas que a su vez son parte de una *bot network* (*botnet*) y que son responsables de controlar la actividad online de diversas cuentas falsas con el objetivo de diseminar información falsa. En este dominio también pueden distinguirse diferentes tipos con capacidades diferentes; por ejemplo, algunos sólo comparten contenido, otros promocionan contenido (por ejemplo, a través de manipulación de votos en Reddit), y otros publican contenido propio.

Uno de los casos más emblemáticos se dio durante la campaña de elecciones presidenciales del año 2016 en EEUU (evento que parece haber marcado un antes y un después en lo referido a estos temas) a partir del cual se han dado a conocer reportes que muestran que Rusia creó cuentas falsas y *bots* para difundir historias falsas. Altos funcionarios de inteligencia y de otras áreas del gobierno de EEUU afirman que los rusos han utilizado algoritmos para segmentar enormes poblaciones en miles de subgrupos de acuerdo a características que los definan como religión, creencias políticas, o preferencias de programas de TV y música. Una vez hecho esto, el próximo paso fue elaborar manualmente mensajes del estilo propaganda y direccionarlos a aquellos que se consideren más susceptibles de influencia utilizando para esto *bots* y personas que estén a cargo de las cuentas publicando dicho contenido. En este sentido, investigadores de la *University of Southern California* en 2017 descubrieron que casi el 20 % de los tweets políticos entre el 16 de septiembre y el 21 de octubre de 2016 fueron generados por *bots* de origen desconocido [Tim17], y más tarde en 2018 Twitter admitió que más de 50 mil cuentas vinculadas a Rusia usaron su servicio para publicar material automatizado sobre las mencionadas elecciones, lo cual habría tenido un alcance de al menos 677.775 estadounidenses [The18].

1.4. Actividades de hacking

Muchas de las personas detrás de las operaciones cibernéticas, que se originan fuera de los laboratorios gubernamentales o centros militares, dependen de una comunidad significativa de hackers que interactúan a través de una variedad de foros en línea, como

un medio para permanecer en el anonimato y llegar a colaboradores geográficamente dispersos. En este punto es en el cual entran en escena los sitios de la *Darknet* y *Deepnet* los cuales son ampliamente utilizados para comunicaciones que requieren la mayor privacidad posible, y por esta razón es que dichos sitios están basados en tecnologías como *The Onion Router* (*Tor*) que puede describirse como un software gratuito dedicado a proteger la privacidad de sus usuarios al ocultar el análisis del tráfico como una forma de vigilancia de la red. Con este objetivo, el tráfico de red en Tor es guiado a través de varios servidores operados por voluntarios (también llamados "nodos"). Aquí, cada nodo de la red encripta la información que pasa ciegamente, sin registrar de dónde proviene el tráfico ni hacia dónde se dirige, lo que no permite ningún seguimiento. Efectivamente, esto permite no sólo la navegación anónima (la dirección IP revelada será la del último nodo), sino también la elusión de la censura. Aquí, usaremos *Darknet* para denotar la comunicación anónima proporcionada por las redes criptográficas como *Tor*, que contrasta con la *Deepnet* que comúnmente se refiere a sitios web alojados en la parte abierta de Internet (*Clearnet*), pero no indexado por los motores de búsqueda.

Dentro de este contexto, emergen mercados donde los usuarios venden sus productos que usualmente son bienes y servicios relacionados a *hacking* malicioso, drogas, pornografía, armas y servicios de software. Al mismo tiempo, los vendedores con frecuencia promocionan sus productos en foros para atraer la atención hacia sus bienes y servicios, aunque sólo una pequeña fracción de esos productos están relacionados a *hacking* malicioso. Los foros propiamente dichos son plataformas orientadas al usuario que tienen como propósito permitir la comunicación y propician las condiciones adecuadas para el surgimiento de una comunidad de personas con ideas afines, independientemente de su ubicación geográfica. Particularmente, en la *Darknet* se crean foros tratando de aprovechar la capacidad de disponer mayor confidencialidad en la comunicación para sus miembros, y si bien la estructura y la organización de los foros alojados allí pueden ser muy similares a los foros web más familiares, los temas y las inquietudes de los usuarios varían claramente. Los foros dirigidos a *hackers* maliciosos presentan principalmente discusiones sobre programación, *hacking* y ciberseguridad [NDG+16].

En la Figura 1.6 se puede observar que se requiere cierto tiempo desde que una vulnerabilidad[2] es conocida hasta que se logre obtener (si se logra) un *exploit* para la misma.

[2]No todas las vulnerabilidades llegan a ser explotadas y que aunque exista el *exploit* no necesariamente éste llega a derivar en un *malware*.

Fecha	**Evento**
Febrero de 2015	*Microsoft identifica la vulnerabilidad de Windows MS15-010/CVE 2015-0057 para la ejecución remota de código. No existía un exploit conocido públicamente hasta ese momento.*
Abril de 2015	*Un exploit para MS15-010/CVE 2015-0057 fue encontrado en venta en un mercado de la Darknet por 48 bitcoins, equivalentes a un valor entre 10000 y 15000 dólares.*
Julio de 2015	*FireEye identificó que el "Dyre Banking Trojan", diseñado para robar números de tarjetas de crédito, explotó esta vulnerabilidad para llevar a cabo su objetivo.*

Figura 1.6: Ejemplo de sucesos que pueden ser identificables hasta que un *malware* es propagado para cumplir sus propósitos maliciosos. Tabla construida en base a [NDG+16].

Como en este caso luego puede que sea publicado en un mercado de la *Darknet*, para que finalmente alguien lo compre y con el tiempo se utilice para crear *malware*. *Esto nos sugiere que esta información, en caso de estar disponible, podría utilizarse para prevenir futuros ciberataques*. En esta dirección se pueden mencionar trabajos como [AMN+18, NSS18] e iniciativas de empresas privadas como CYR3CON[3] que buscan aprovechar información proveniente de las discusiones de los foros de la *Darknet* y *Deepnet*. Uno de los reportes basado en una colaboración de dicha empresa con *WhiteSource* [CW19] muestra que 85 de las 100 vulnerabilidades de seguridad *open source* más comunes tuvieron algún nivel de discusión entre *hackers* en comunidades *online* durante 2019. Asimismo, dichas vulnerabilidades son discutidas incluso 6 meses después de ya haber sido explotadas y, por otro lado, aunque en ocasiones ya no puedan considerarse novedosas es posible que surjan nuevamente en las discusiones de la comunidad de *hackers* a medida que aparecen en nuevos *exploits* o *malware*.

La Figura 1.7 muestra la correlación entre los tipos de vulnerabilidades (*CWEs*[4]) más comunes y su nivel de popularidad en las comunidades de *hackers* (*CyRating*[5]) durante 2019. De hecho, los 4 *CWEs* que consiguieron el *CyRating* más alto estaban entre los 5 *CWEs* más comunes de ese año. Por lo tanto, se puede observar que los *hackers* atacan algunos *CWE* específicos más que otros, lo cual podría interpretarse asumiendo que ellos prefieren aprender una habilidad y mantenerla, es decir, son competentes en tipos muy específicos de *exploits*. Hay varias razones para esto, pero se puede mencionar el hecho de que muchas herramientas automatizadas hacen que estas vulnerabilidades sean más fáciles de explotar, incluso para *hackers* principiantes (*script kiddies*). Por otro lado, cuanto más común es un *CWE*, más *hackers* lo estudiarán, aprenderán a explotarlo y lo discutirán. La

[3] https://www.cyr3con.ai/

[4] *CWEs* son identificadores de tipos de debilidades de *hardware* y *software* definida por MITRE.

[5] *CyRating* es un valor para medir el nivel de riesgo de vulnerabilidades definida por CYR3CON.

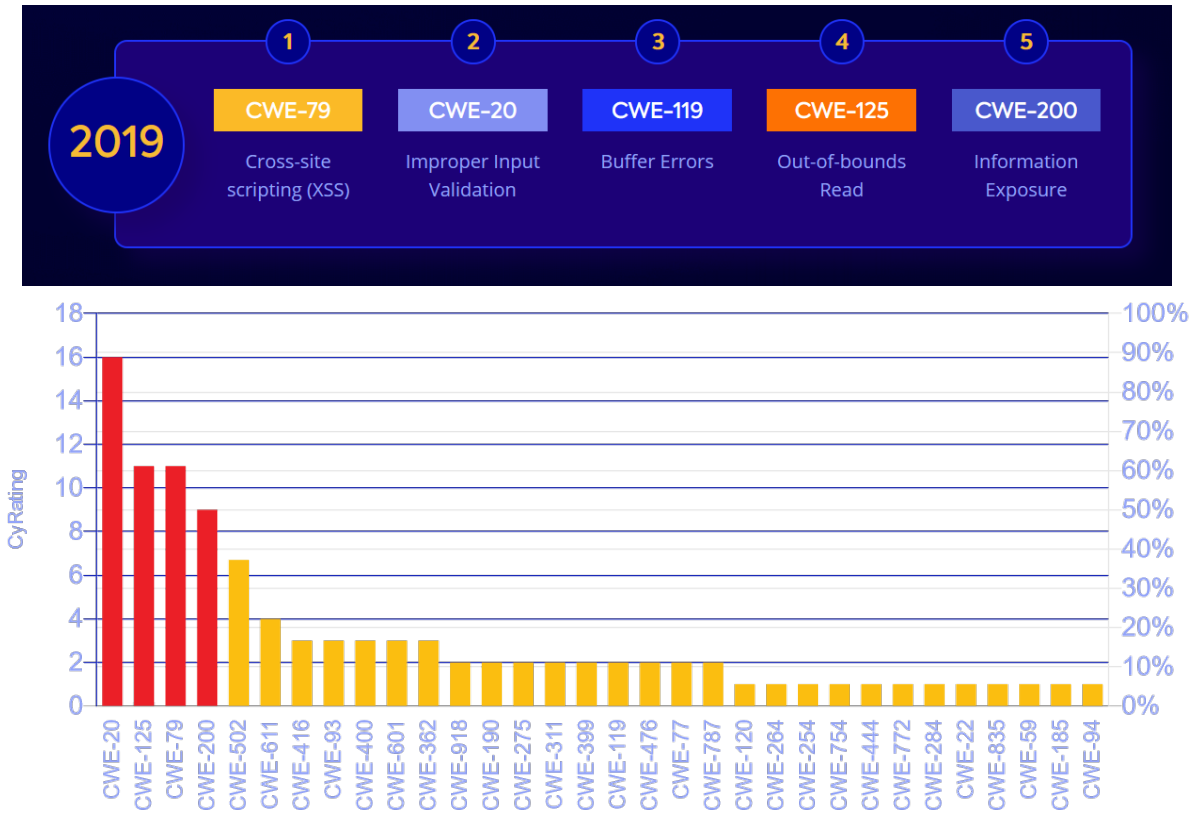

Figura 1.7: Representación de los tipos de vulnerabilidades (CWEs) más comunes en 2019 (posición superior) y volumen de discusión en comunidades de hackers (CyRating) de diferentes CWEs durante el mismo año (posición inferior). Figura extraída de [CW19].

investigación de seguridad y las comunidades de código abierto, a su vez, pueden centrarse en descubrir y reparar estos *CWE* para evitar futuras vulnerabilidades.

Los resultados de estos trabajos y muchos más, nos indican que es posible realizar un análisis inteligente de discusiones sobre actividades de *hacking* para llevar a cabo tareas de *ciberdefensa*, haciendo uso de la información relevante que se encuentra disponible.

1.5. Publicaciones surgidas de esta tesis

A continuación se enumeran los trabajos científicos derivados en el proceso de elaboración de la presente tesis junto con la mención al capítulo que tiene vinculación.

- "Leveraging Probabilistic Existential Rules for Adversarial Deduplication" [PMSF18]: José N. Paredes, Maria Vanina Martinez, Gerardo I. Simari y Marcelo A. Falappa. PRUV@IJCAR, 2018. El resultado de este trabajo se describe con más detalle en la Sección 3.1.

- "First Steps towards Data-driven Adversarial Deduplication" [PSMF18]: José N. Paredes, Gerardo I. Simari, Maria Vanina Martinez y Marcelo A. Falappa. Information, 2018. El resultado de este trabajo se describe con más detalle en la Sección 3.2.

- "NetDER: An Architecture for Reasoning About Malicious Behavior" [PSMF20b]: José N. Paredes, Gerardo I. Simari, Maria Vanina Martinez y Marcelo A. Falappa. Information Systems Frontiers, 2020. El resultado de este trabajo se describe con más detalle en el Capítulo 4.

- "Combining Existential Rules with Network Diffusion Processes for Automated Generation of Hypotheses" [PSM$^+$20]: José N. Paredes, Gerardo I. Simari, Maria Vanina Martinez, Marcelo A. Falappa y Paulo Shakarian. 2020 (En evaluación para una revista internacional). El resultado de este trabajo se describe con más detalle en el Capítulo 5.

- "Detecting Malicious Behavior in Social Platforms via Hybrid Knowledge- and Data-driven Systems" [PSMF20a]: José N. Paredes, Gerardo I. Simari, Maria Vanina Martinez y Marcelo A. Falappa. 2020 (En evaluación para una revista internacional). El resultado de este trabajo se describe con más detalle en el Capítulo 6.

1.6. Organización de la tesis

El resto de la tesis se encuentra organizada de la siguiente forma:

- En el Capítulo 2 se realiza una descripción de la familia de lenguajes ontológicos Datalog+/– que será utilizada como base para las tareas de razonamiento. Luego, se detallan extensiones que incorporan mayores capacidades de representación de conocimiento incierto, y finalmente se realiza una descripción del formalismo que será utilizado como base para modelar procesos de difusión en redes complejas.

- En la Sección 3.1 del Capítulo 3 se propone utilizar *reglas existenciales probabilísticas* para modelar soluciones de ingeniería de conocimiento orientadas a un problema al que se nombra como *deduplicación adversarial*. Este problema es una generalización de un problema clásico de Base de Datos conocido como *resolución de entidades* o *deduplicación*, cuyo propósito es determinar un mapeo entre múltiples entidades virtuales a su correspondiente conjunto de entidades de la realidad. En esta nueva variación del problema, las suposiciones acerca de por qué se producen entidades duplicadas cambian debido a que ahora su existencia pudo haber sido planificada por alguien (adversario). Esta suposición es particularmente adecuada en dominios como el de ciberseguridad donde, por ejemplo, actores maliciosos que operan en mercados y foros de hackers desean mantener sus verdaderas entidades ocultas (empleando diferentes perfiles en la misma o diferentes plataformas), pero al mismo tiempo necesitan ser identificados por sus clientes para mantener cierta reputación. En estas condiciones se considera conveniente trabajar bajo una suposición de mundo abierto, por lo que se propone aprovechar la utilización de reglas que permitan crear hipótesis de deduplicación probabilísticas sobre objetos potencialmente desconocidos que luego puedan hacerse conocidos aplicando nuevas reglas al recibir nueva información. Este tipo de problema puede considerarse como una de las formas de comportamiento malicioso, y esta propuesta es una primera aproximación basada en reglas lógicas hacia la detección de comportamientos de este estilo.

- En la Sección 3.2 del Capítulo 3 se propone utilizar *herramientas basadas en aprendizaje automatizado* para atacar el problema de deduplicación adversarial realizando experimentos tomando como escenario mercados y foros de *hackers* maliciosos. Para dicha tarea se utilizaron datos obtenidos de colaboraciones con una empresa de ciberseguridad. Sin embargo, debido a la dificultad de obtener datos etiquetados con la respuesta correcta, las identidades duplicadas fueron creadas antes de realizar los experimentos dividiendo cada identidad real en dos nuevas, proporcionando la mitad de las publicaciones a cada una. Específicamente, se entrenaron diferentes clasificadores considerando solo información textual y se emplearon para predecir si dos entidades en realidad representan a una sola. Aprovechando que cada clasificador para cada publicación puede dar una respuesta diferente, se utiliza el número de veces que la respuesta es afirmativa para crear hipótesis de deduplicación. Esta propuesta es una primera aproximación basada en herramientas de aprendizaje

automatizado para atacar un tipo particular de comportamiento malicioso, pero teniendo en mente el desarrollo de herramientas más generales que permitan atacar el problema en su forma más general.

- En el Capítulo 4 se presenta una *arquitectura para razonar respecto a comportamiento malicioso* con el objetivo de guiar el desarrollo de software para atacar este tipo de problemas. La propuesta requiere la instanciación de tres módulos:

  1. el módulo de ingesta de datos encargado de tareas de preparación de la información, el cual incluye limpieza de datos, unificación de esquemas, etc., así como otras cuestiones de más alto nivel como confianza, manejo de incertidumbre o la propia creación de las bases de conocimiento para los otros dos módulos,
  2. el módulo de razonamiento ontológico que almacena tanto la base conocimiento ontológico (referido al dominio) como el conocimiento básico de la estructura y estado de la red subyacente, y además proporciona servicios de inferencia a través de respuestas a consultas, y
  3. el módulo de difusión de la red que se encarga de ejecutar procesos de difusión simulando la propagación de información para que luego el módulo descripto en el ítem 2 pueda verificar si se cumplen determinadas condiciones.

- En el Capítulo 5 se desarrollan las bases teóricas de los módulos de razonamiento ontológico y de difusión de la red descriptos en el ítem anterior, y con el objetivo de llevar a cabo tareas de razonamiento orientadas a responder consultas se definen tres políticas para determinar la interacción entre estos dos módulos mencionados. Asimismo, considerando la tarea de responder consultas se realiza un estudio de la relación existente entre expresividad de los lenguajes subyacentes a los dos módulos y costos de cómputo, incluyendo casos en los cuales se vuelve indecidible. Como contribución final se presenta un caso de uso mostrando cómo es posible instanciar los diferentes módulos en un escenario del mundo real, particularmente en un dominio de ciberseguridad.

- En el Capítulo 6 se llevan a cabo experimentos instanciando diferentes variantes de NetDER en diferentes escenarios de propagación de noticias falsas combinando información real y sintética, con el objetivo de evaluar su desempeño haciendo foco principalmente en tres tareas (i) determinar quién es responsable de propagar una noticia falsa, (ii) detectar usuarios maliciosos, y (iii) detectar quiénes pertenecen

a una botnet diseñada para diseminar noticias falsas. Como contribución adicional se diseña un *testbed* que puede ser configurado para instanciar diferentes escenarios de propagación de contenido malicioso, y se deja disponible públicamente su código para futuros esfuerzos en la misma dirección.

- Finalmente, en el Capítulo 7 se realiza un análisis de los trabajos relacionados y en el Capítulo 8 se presentan las conclusiones y el trabajo futuro.

1.7. Sumario

En este capítulo introductorio se realizó un análisis de la variedad de problemas interesantes para ser abordados, se pudo observar que existen relaciones entre ellos, que muchos de ellos son muy complejos, que se están realizando distintos esfuerzos para atacarlos, y que se requiere un abordaje general que idealmente pueda aprovechar las ventajas que pueden otorgar las herramientas basadas en aprendizaje automatizado combinadas con conocimiento de más alto nivel como pueden ser los lenguajes basados en lógica.

Capítulo 2

Conceptos preliminares

A continuación, se realizará una descripción general de conceptos relativos a representación de conocimiento y razonamiento, por un lado, en ontologías en la Sección 2.1 y, por otro lado, en redes complejas en la Sección 2.2. Este material es considerado básico para que el desarrollo del contenido presentado en esta tesis sea autocontenido.

2.1. *Datalog+/–*

En esta sección se llevará adelante un repaso de los conceptos fundamentales involucrados en la conocida familia de lenguajes ontológicos *Datalog+/–*, lo cual resulta sumamente importante para el desarrollo de todo el trabajo presentado en esta tesis. Asimismo, se toma como base el trabajo y los ejemplos presentados en [SMM+17] para la elaboración de esta sección.

Es conocido que los enfoques que utilizan principios y técnicas de *representación de conocimiento y razonamiento* (*Knowledge Representation and Reasoning*) basados en lógica tienen numerosas aplicaciones en escenarios con diferentes características; una de ellas es poder obtener representaciones conceptuales de alto nivel sobre algún dominio de interés. Estas representaciones son comúnmente conocidas como *ontologías*, y pueden ser vistas como una manera de limitar lo que es "interesante", pero considerando como objetivo principal que el conocimiento esté disponible en un formato que las computadoras puedan intercambiar y procesar. Más específicamente, las ontologías se definen por conjuntos de *términos de representación* expresados de manera explícita (*parte axiomática*) o implícita

(*parte terminológica*), los cuales formalmente definen una teoría lógica. Estos términos permiten la definición de varios objetos y construcciones como *clases*, *relaciones* y *funciones*, entre otros. Básicamente, hay tres niveles en una especificación formal de una ontología, y su utilización adecuada permite responder consultas a partir de fuentes de datos incompletas y conocimiento específico del dominio. Es decir, la consulta se responde teniendo en cuenta la información descripta en la parte axiomática y terminológica de la ontología.

- *Metanivel*: abarca el conjunto de *categorías* a representar; por ejemplo, usuarios, malware, publicaciones, etc. considerando una perspectiva de *modelado*.
- *Nivel intensional*: incluye el conjunto de *elementos conceptuales* que son instancias de las categorías y el conjunto de *reglas* que determinan una estructura sobre los elementos conceptuales. Por ejemplo, los conceptos usuarios, malware y publicaciones, cada uno con sus atributos específicos, se utilizan para representar las categorías mencionadas en el metanivel. Como ejemplos de reglas, se puede mencionar conocimiento relativo a que todo malware es una pieza de software, o que para cada publicación existe un usuario que la creó, etc.
- *Nivel extensional*: contiene el conjunto de *instancias* de los elementos conceptuales que se especifican en el nivel anterior; por ejemplo, johnny28 es un usuario, johnny28 es el autor de la publicación p349, etc.

Como ya se mencionó, la definición de ontologías implica la utilización de un lenguaje lógico subyacente y, justamente en esta cuestión *Datalog+/–* es muy conocido dentro del área, dando lugar a una familia de lenguajes ontológicos. A continuación, se detallan las cuestiones fundamentales para emplear ontologías definidas a través de *Datalog+/–*.

Las unidades de representación básicas de los lenguajes en *Datalog+/–* son un conjunto infinito de *constantes* Δ, un conjunto infinito de *variables* $\mathcal{V}$ y un conjunto infinito de elementos especiales llamados *valores nulos* Δ_N; los dos primeros conjuntos se comparten con muchos lenguajes lógicos, mientras que el último es menos común. Los valores nulos pueden verse como un tipo especial de variables: describen valores que *aún no se conocen*; mientras que dos nulos diferentes pueden representar el mismo valor, se supone que dos constantes diferentes representan valores diferentes. Los objetos más básicos se denominan *átomos* (o *fórmulas atómicas*), que se obtienen a partir de *símbolos predicativos* aplicados a

constantes, variables y nulos; por ejemplo, *perro*(*fido*) es una fórmula atómica construida aplicando el símbolo predicativo *perro* a la constante *fido*, lo que indica que Fido es un perro. Una instancia de base de datos se modela como un conjunto de átomos que solo contienen constantes. Asimismo, se realizan las siguientes asunciones:

- existe un orden lexicográfico sobre $\Delta \cup \Delta_N$ donde cada símbolo Δ_N se sigue a cada símbolo en Δ.
- $\mathbf{X}$ se utiliza para denotar una secuencia de variables $X_1, ..., X_k$ con $k \geq 0$.
- se dispone de un *esquema relacional* $\mathcal{R}$, el cual es un conjunto finito de símbolos predicativos.

Como es usual, un *término* t puede ser una constante, un nulo, o una variable; una *fórmula atómica* (o *átomo*) tiene la forma $p(t_1, ..., t_n)$, donde p es un predicado n-ario y $t_1, ..., t_n$ son términos. Un término o átomo es básico (*ground*) si no contiene valores nulos ni variables; y una *instancia* I para un esquema relacional $\mathcal{R}$ es un conjunto de átomos con predicados tomados de $\mathcal{R}$ y argumentos tomados de $\Delta \cup \Delta_N$.

Homomorfismos. Uno de los conceptos fundamentales para la semántica de *Datalog+/–* es la noción de *homomorfismo* entre estructuras relacionales. Considere dos estructuras relacionales definidas como $A = \langle X, \sigma^A \rangle$ y $B = \langle Y, \sigma^B \rangle$, donde $dom(A) = X$ y $dom(B) = Y$ son los dominios de A y de B, y σ^A y σ^B son sus signaturas (las cuales se componen de relaciones y funciones), respectivamente. Entonces, un *homomorfismo* de A a B es una función $h : dom(A) \to dom(B)$ que "preserva estructura" en el siguiente sentido:

- Para cada función n-aria $f^A \in \sigma^A$ y elementos $x_1, ..., x_n \in dom(A)$ se cumple que: $h(f^A(x_1, ..., x_n)) = f^B(h(x_1), ..., h(x_n))$[1] y
- Para cada relación n-aria $R^A \in \sigma^A$ y elementos $x_1, ..., x_n \in dom(A)$ se cumple que: si $(x_1, ..., x_n) \in R^A$, entonces $(h(x_1), ..., h(x_n)) \in R^B$[2].

[1]Teniendo en cuenta que cualesquiera de los lenguajes en consideración no hace uso de símbolos funcionales, esta condición no es necesaria en este caso porque se satisface trivialmente.

[2]Los superíndices son usados en símbolos de funciones y relaciones como una aclaración de la estructura en la cual están siendo aplicados.

La importancia de los homomorfismos se apoya en el hecho de que se puede establecer una conexión con el proceso de respuesta a consultas conjuntivas sobre base de datos relacionales, la cual puede enunciarse de la siguiente forma: *considere una consulta conjuntiva booleana Q y una instancia de base de datos J, entonces $J \models Q$ si y sólo si existe un homomorfismo de la instancia canónica de base de datos I^Q, básicamente construida a partir de variables y predicados de Q, a J*. Ahora bien, considerando las características de los lenguajes de interés, se requiere extender esta noción de homomorfismos para contemplar valores nulos. Por lo tanto, estos homomorfismos extendidos desde un conjuntos de átomos A_1 a un conjunto de átomos A_2 se definen como mapeos $h : \Delta \cup \Delta_N \cup \mathcal{V} \to \Delta \cup \Delta_N \cup \mathcal{V}$ tales que:

- si $c \in \Delta$ entonces $h(c) = c$,
- si $c \in \Delta_N$ entonces $h(c) \in \Delta \cup \Delta_N$,
- si $r(t_1, ..., t_n) \in A_1$ entonces $h(r(t_1, ..., t_n)) = r(h(t_1), ..., h(t_n)) \in A_2$

De una manera análoga es posible extender este concepto a conjunciones de átomos.

2.1.1. Sintaxis y semántica

Asumiendo que se dispone de un esquema relacional $\mathcal{R}$, un *programa Datalog+/–* consiste de un conjunto finito de *dependencias de generación de tupla* (*TGDs*), *restricciones negativas* (*NCs*), y *dependencias de generación de igualdad* (*EGDs*).

Dependencia de generación de tupla. Una *dependencia de generación de tupla* (*TGD*) es una regla de primer orden (*FO*) σ que permite conjunciones de átomos cuantificados existencialmente en la parte conocida como cabeza:

$$\sigma : \forall \mathbf{X} \forall \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y}) \to \exists \mathbf{Z} \Psi(\mathbf{X}, \mathbf{Z})$$

donde $\mathbf{X}, \mathbf{Y}, \mathbf{Z} \subseteq \mathcal{V}$, $\Phi(\mathbf{X}, \mathbf{Y})$ y $\Psi(\mathbf{X}, \mathbf{Z})$ son conjunciones de átomos. Se debe tener en cuenta que las TGDs con múltiples átomos individuales en la cabeza pueden ser convertidas en un conjunto de TGDs con sólo un átomo en dicha parte de la regla. Por esta razón, se asumirá que todos los conjuntos de TGDs fueron convertidos como se mencionó antes.

Satisfacción de TGDs. Una instancia I sobre un esquema $\mathcal{R}$ *satisface* σ, denotado $I \models \sigma$, si cuando existe un homomorfismo h que mapea los átomos de $\Phi(\mathbf{X}, \mathbf{Y})$ a átomos de I, entonces existe una extensión h' de h que mapea $\Psi(\mathbf{X}, \mathbf{Z})$ a átomos de I.

Restricción negativa. Una *restricción negativa* (*NC*) es una regla de primer orden μ que permite expresar negación:

$$\mu : \forall \mathbf{X}\ \Phi(\mathbf{X}) \rightarrow \bot$$

donde $\mathbf{X} \subseteq \mathcal{V}$ y $\Phi(\mathbf{X})$ es una conjunción de átomos.

Satisfacción de NCs. Una instancia I sobre un esquema $\mathcal{R}$ *satisface* μ, denotado $I \models \mu$, si para cada homomorfismo h se cumple que $h(\Phi(\mathbf{X})) \not\subseteq I$.

Dependencia de generación de igualdad. Una *dependencia de generación de igualdad* (*EGD*) es una regla de primer orden ϵ de la forma:

$$\epsilon : \forall \mathbf{X}\ \Psi(\mathbf{X}) \rightarrow X_i = X_j$$

donde $X_i, X_j \in \mathbf{X} \subseteq \mathcal{V}$ y $\Phi(\mathbf{X})$ es una conjunción de átomos.

Satisfacción de EGDs. Una instancia I sobre un esquema $\mathcal{R}$ *satisface* ϵ, denotado $I \models \epsilon$, si cuando existe un homomorfismo h tal que $h(\Phi(\mathbf{X})) \subseteq I$ se cumple que $h(X_i) = h(X_j)$.

Por cuestiones de simplicidad, cuando se definan TGDs, NCs, o EGDs los cuantificadores universales serán omitidos pero se asumirá que todas las variables en el cuerpo se encuentran cuantificadas existencialmente.

Programa *Datalog+/–*. Un *programa Datalog+/–* , denotado Σ, es un conjunto finito $\Sigma_T \cup \Sigma_{NC} \cup \Sigma_E$ de TGDs, NCs, y EGDs, donde el esquema del mismo, denotado $\mathcal{R}(\Sigma)$, se encuentra definido por el conjunto de predicados que aparecen en Σ.

Ontología *Datalog+/–*. Una *ontología Datalog+/–* , denotada $O = (D, \Sigma)$, consiste de una base de datos finita D y un programa *Datalog+/–* Σ.

Ejemplo 2.1 *Considere la ontología Datalog+/– $KB = (D, \Sigma)$ donde D y $\Sigma = \Sigma_T \cup \Sigma_E$ son definidos de la siguiente forma:*

$$\begin{aligned}
\Sigma_T \ = \{ \ & r_1 : \mathit{restaurant}(R) \rightarrow \mathit{business}(R), \\
& r_2 : \mathit{restaurant}(R) \rightarrow \exists F\ \mathit{food}(F) \wedge \mathit{serves}(R, F), \\
& r_3 : \mathit{restaurant}(R) \rightarrow \exists C\ \mathit{cuisine}(C) \wedge \mathit{restaurantCuisine}(R, C), \\
& r_4 : \mathit{business}(B) \rightarrow \exists C\ \mathit{city}(C) \wedge \mathit{locatedIn}(B, C), \\
& r_5 : \mathit{city}(C) \rightarrow \exists D\ \mathit{country}(D) \wedge \mathit{locatedIn}(C, D)\}, \\
\Sigma_E \ = \{ \ & \mathit{locatedIn}(X, Y), \mathit{locatedIn}(X, Z) \rightarrow Y = Z\},
\end{aligned}$$

$$
\begin{aligned}
D \;= \{\;& \mathit{food}(\mathit{bifeDeChorizo}), \quad \mathit{food}(\mathit{soupAl'Oignon}),\\
& \mathit{foodType}(\mathit{meat}), \quad \mathit{foodType}(\mathit{soup}),\\
& \mathit{cuisine}(\mathit{argentine}), \quad \mathit{cuisine}(\mathit{french}),\\
& \mathit{restaurant}(\mathit{laCabrera}), \quad \mathit{restaurant}(\mathit{laTartine}),\\
& \mathit{city}(\mathit{buenosAires}), \quad \mathit{city}(\mathit{paris}),\\
& \mathit{country}(\mathit{argentina}), \quad \mathit{country}(\mathit{france}),\\
& \mathit{locatedIn}(\mathit{laCabrera}, \mathit{buenosAires}),\\
& \mathit{serves}(\mathit{laCabrera}, \mathit{bifeDeChorizo}), \quad \mathit{serves}(\mathit{laTartine}, \mathit{soupeAlOignon})\}.
\end{aligned}
$$

Esta ontología modela una base de conocimiento para restaurantes—podría ser usada, por ejemplo, como el modelo subyacente de un sistema de recomendación y revisión (tales como TripAdvisor o Yelp).

■

La conjunción de las sentencias de primer orden asociadas con las reglas de un programa *Datalog+/–* se denota Σ_p. Un *modelo* de Σ es una instancia para $\mathcal{R}(\Sigma)$ que *satisface* Σ_p. Para una base de datos D y un conjunto de TGDs Σ sobre $\mathcal{R}$, el conjunto de *modelos* de D y Σ, denotado $\mathit{mods}(D, \Sigma)$, es el conjunto (posiblemente infinito) de todas las instancias I tales que:

- $D \subseteq I$ y
- cada $\sigma \in \Sigma$ es satisfecho en I (es decir, $I \models \Sigma$).

Por otro lado, una ontología *Datalog+/–* $O = (D, \Sigma)$ es *consistente* si el conjunto de modelos $\mathit{mods}(D, \Sigma)$ no es vacío.

La semántica de Σ sobre una base de datos de entrada D, denotada $\Sigma(D)$, es un modelo I de D y Σ tal que para todo modelo I' de D y Σ existe un homormorfismo h tal que $h(I) \subseteq I'$; una instancia como la descripta es denominada modelo *universal* de Σ con respecto a D. Intuitivamente, un modelo universal no contiene información de más ni de menos de la que el programa Σ requiere.

En general, existe más de un modelo universal de Σ con respecto a D, pero todos los modelos universales son (por definición) homomórficamente equivalentes; es decir, para cada par de modelos universales M_1 y M_2, existe un homomorfismo h_1 y h_2 tal que

$h_1(M_1) \subseteq M_2$ y $h_2(M_2) \subseteq M_1$. Por lo tanto, $\Sigma(D)$ es único, si se considera la equivalencia homomórfica.

Como es de esperarse, además de representar conocimiento, resulta interesante poder razonar a partir del conocimiento representado. Por lo tanto, ahora el foco es puesto en la capacidad de razonar a partir de ontologias *Datalog+/–*.

2.1.2. Respuesta a consultas conjuntivas *Datalog+/–*

Una de las formas más conocidas de razonamiento es a través del proceso de respuesta a consultas utilizando el conocimiento que se encuentra disponible. Con la intención de lograr tal capacidad de razonamiento, primero resulta necesario definir qué es una consulta en este contexto. Una consulta conjuntiva (CQ) sobre $\mathcal{R}$ tiene la forma:

$$\underbrace{q(\mathbf{X})}_{cabeza} = \underbrace{\exists\ \mathbf{Y}\ \phi(\mathbf{X}, \mathbf{Y})}_{cuerpo},$$

donde $\phi(\mathbf{X}, \mathbf{Y})$ es una conjunción de átomos (posiblemente con igualdades, pero no desigualdades) con las secuencias de variables $\mathbf{X}$ e $\mathbf{Y}$ y posiblemente constantes, pero sin valores nulos y con q siendo un predicado que no está en $\mathcal{R}$. Una consulta conjuntiva booleana (BCQ) *Datalog+/–* sobre $\mathcal{R}$ es una consulta conjuntiva *Datalog+/–* de la forma $q()$, la cual con frecuencia se escribe como el conjunto de todos sus átomos sin cuantificadores. Asimismo, el conjunto de *respuestas* a una consulta conjuntiva *Datalog+/–* $q(\mathbf{X}) = \exists\ \mathbf{Y}\ \phi(\mathbf{X}, \mathbf{Y})$ sobre una instancia I, denotada $q(I)$, es el conjunto de todas las tuplas t sobre Δ, para las cuales existe un homomorfismo $h : \mathbf{X} \cup \mathbf{Y} \to \Delta \cup \Delta_N$ tal que $h(\phi(\mathbf{X}, \mathbf{Y})) \subseteq I$ y $h(\mathbf{X}) = t$. La respuesta a la consulta conjuntiva booleana *Datalog+/–* $q()$ sobre una instancia de base de datos I es "Sí", denotado $D \models q$, si $q(I) \neq \emptyset$.

En términos formales, *responder consultas* bajo TGDs, es decir, la evaluación de consultas conjuntivas y consultas conjuntivas booleanas sobre bases de datos bajo un conjunto de TGDs es definida de la siguiente forma: el conjunto de *respuestas* a una consulta conjuntiva q sobre una base de datos D y un conjunto de TGDs Σ, denotado $ans(q, D, \Sigma)$, es el conjunto de todas las tuplas t tal que $t \in q(I)$ para todo $I \in mods(D, \Sigma)$. Análogamente, la respuesta a una consulta conjuntiva booleana q sobre D y Σ es "Sí", denotado $D \cup \Sigma \models q$, si $ans(q, D, \Sigma) \neq \emptyset$. Observe que para el proceso de respuesta a consultas *Datalog+/–*, las instancias homomórficamente equivalentes son indistinguibles; es decir,

dadas dos instancias I e I' que son homomórficamente equivalentes, $q(I)$ y $q(I')$ coinciden. Por lo tanto, las consultas pueden ser evaluadas sobre cualquier modelo universal.

Similarmente, el problema de decisión relativo al proceso de respuesta a consultas conjuntivas booleanas *Datalog+/–* es definido de la siguiente manera: dada una base de datos D, un conjunto Σ de TGDs, una consulta conjuntiva booleana *Datalog+/–* q y una tupla t sólo con constantes, el mencionado problema de decisión implica comprobar si $t \in ans(q, D, \Sigma)$.

Para el caso del proceso de respuesta a consultas conjuntivas booleanas en *Datalog+/–* con TGDs, agregar restricciones negativas es computacionalmente fácil debido a que para cada restricción $\forall \mathbf{X}\ \phi(\mathbf{X}) \rightarrow \bot$ sólo se tiene que verificar si la consulta conjuntiva booleana $\exists \mathbf{X}\ \phi(\mathbf{X})$ es evaluada como falsa en D bajo Σ; si una de estas verificaciones falla, entonces la respuesta a la consulta conjuntiva booleana original q es "No" y, en caso contrario, dichas restricciones pueden simplemente ignorarse cuando se lleve a cabo el proceso de respuesta a la consulta conjuntiva booleana q.

Por otro lado, agregar EGDs sobre bases de datos con TGDs y restricciones negativas no incrementa la complejidad del proceso de respuesta a consultas conjuntivas booleanas siempre y cuando éstas sean *no conflictivas* [CGL12]. Intuitivamente, esto asegura que, si el *chase* (que se describe más adelante) falla (debido a violaciones fuertes de EGDs), entonces cuando "nuevos" átomos son creados en el *chase* por la aplicación de la regla de *chase* para EGDs, aquellos átomos que son lógicamente equivalentes a los "nuevos" también tienen garantías de ser generados en ausencia de las EGDs. Esto último permite otorgar garantías de que las EGDs no influyen sobre el *chase* con respecto al proceso de respuesta a consultas.

Cabe destacar que hay dos formas muy conocidas de procesar reglas para responder consultas: por *encadenamiento hacia adelante* (el *chase*) y por *encadenamiento hacia atrás*, la cual usa las reglas para reescribir las consultas de diferentes formas con el objetivo de producir una consulta que directamente pueda ser mapeada a los hechos. Para este caso, la operación fundamental es la unificación entre parte de un objetivo actual (una consulta conjuntiva o un hecho) y alguna cabeza de regla. Para los propósitos de esta tesis, sólo se considerará el procedimiento *chase*, el cual será descripto a continuación.

2.1.3. El procedimiento chase

El proceso de respuesta a consultas bajo TGDs generales es indecidible [BV81] y en *Datalog+/–*, generalmente, este proceso es llevado a cabo utilizando el *chase*. Dado un programa Σ, $\Sigma(D)$ puede ser definido como el mínimo punto fijo de un operador monotónico (considerando la equivalencia homomórfica). Esto se puede lograr a través de una aplicación exhaustiva del *chase*, originalmente introducido para verificar la implicación de dependencias y comprobar si una consulta se encuentra contenida [GOPS12]. En términos informales, el *chase* ejecuta las reglas de Σ empezando a partir de D utilizando una estrategia de encadenamiento hacia adelante para inferir átomos nuevos e inventar nuevos valores nulos cuando un cuantificador existencial necesita ser satisfecho. En general, cuando se utiliza la palabra "*chase*", se hace referencia tanto el procedimiento como a su salida.

Regla del chase para TGDs. Considere una base de datos D y una TGD σ de la forma $\phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} \psi(\mathbf{X}, \mathbf{Z})$. Entonces, σ es *aplicable* a D si existe un homomorfismo h que mapea los átomos de $\phi(\mathbf{X}, \mathbf{Y})$ a átomos de D. Sea σ aplicable a D y h_1 un homomorfismo que extiende h de la siguiente forma: para cada $Z_j \in \mathbf{Z}$, $h_1(Z_j) = z_j$ donde z_j es un valor nulo "nuevo", es decir, $z_j \in \Delta_N$, z_j no aparece en D y z_j sigue el orden lexicográfico de todos los valores nulos ya introducidos. La *aplicación* de σ a D agrega a D el átomo $h_1(\psi(\mathbf{X}, \mathbf{Z}))$ si aún no estaba en D. La regla del *chase* descripta anteriormente también es conocida como *oblivious*.

El algoritmo *chase* para una base de datos D y un conjunto de TGDs Σ consiste de una aplicación exhaustiva de la regla del *chase* para TGDs utilizando una estrategia de primero en anchura (saturación de niveles), lo cual produce como salida un *chase* (posiblemente infinito) para D y Σ.

Formalmente, el *chase de nivel hasta 0* de D relativo a Σ, denotado $chase^0(D, \Sigma)$, es definido como D, asignando a cada átomo en D el *nivel* (*de derivación*) 0. Para cada $k \geq 1$, el *chase de nivel hasta k* de D relativo a Σ, denotado $chase^k(D, \Sigma)$, es construido de la siguiente forma: sean $I_1, ..., I_n$ todas las imágenes posibles de los cuerpos de TGDs en Σ relativas a algún homomorfismo tal que:

- $I_1, ..., I_n \subseteq chase^{k-1}(D, \Sigma)$ y
- el nivel más alto de un átomo en cada I_i es $k - 1$.

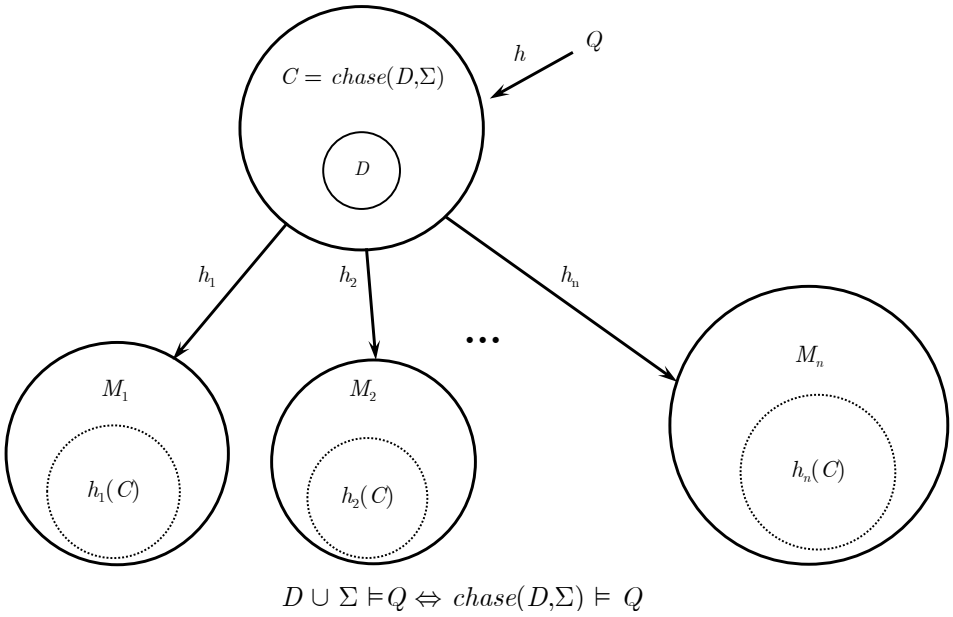

Figura 2.1: Representación gráfica de la estructura de datos producida por el *chase*, la cual permite responder consultas sobre una ontología *Datalog+/–* debido a su naturaleza de modelo universal. Figura extraída de [SMM+17].

Entonces, se lleva a cabo cada correspondiente aplicación de TGD sobre el $chase^{k-1}(D, \Sigma)$, eligiendo las TGDs aplicadas y homomorfismos en un orden lexicográfico y lineal (fijo), respectivamente, y se asigna a cada nuevo átomo el *nivel* (*de derivación*) k. El *chase* de D relativo a Σ, denotado $chase(D, \Sigma)$, es definido como el límite del $chase^k(D, \Sigma)$ para $k \to \infty$.

El *chase* (posiblemente infinito) de una base de datos D relativo a un conjunto de TGDs Σ, denotado $chase(D, \Sigma)$, es un *modelo universal*, es decir, existe un homomorfismo que mapea elementos de $chase(D, \Sigma)$ en elementos de cada $B \in mods(D, \Sigma)$ [CGL12] (cf. Figura 2.1). Por lo tanto, una consulta conjuntiva booleana *Datalog+/–* q puede ser evaluada sobre el *chase* para D y Σ, es decir, $D \cup \Sigma \models q$ es equivalente a $chase(D, \Sigma) \models q$. Se asumirá que los valores nulos introducidos en el *chase* son nombrados a través de skolemización, lo cual proporciona la ventaja de hacer que el *chase* sea único; Δ_n es por lo tanto el conjunto de todos los posibles valores nulos que pueden ser introducidos en el *chase*.

Regla del chase para EGDs. Considere una base de datos D y una EGD ϵ de la forma $\phi(\mathbf{X}) \to X_m = X_n$. Entonces, ϵ es *aplicable* a D si existe un homomorfismo h que

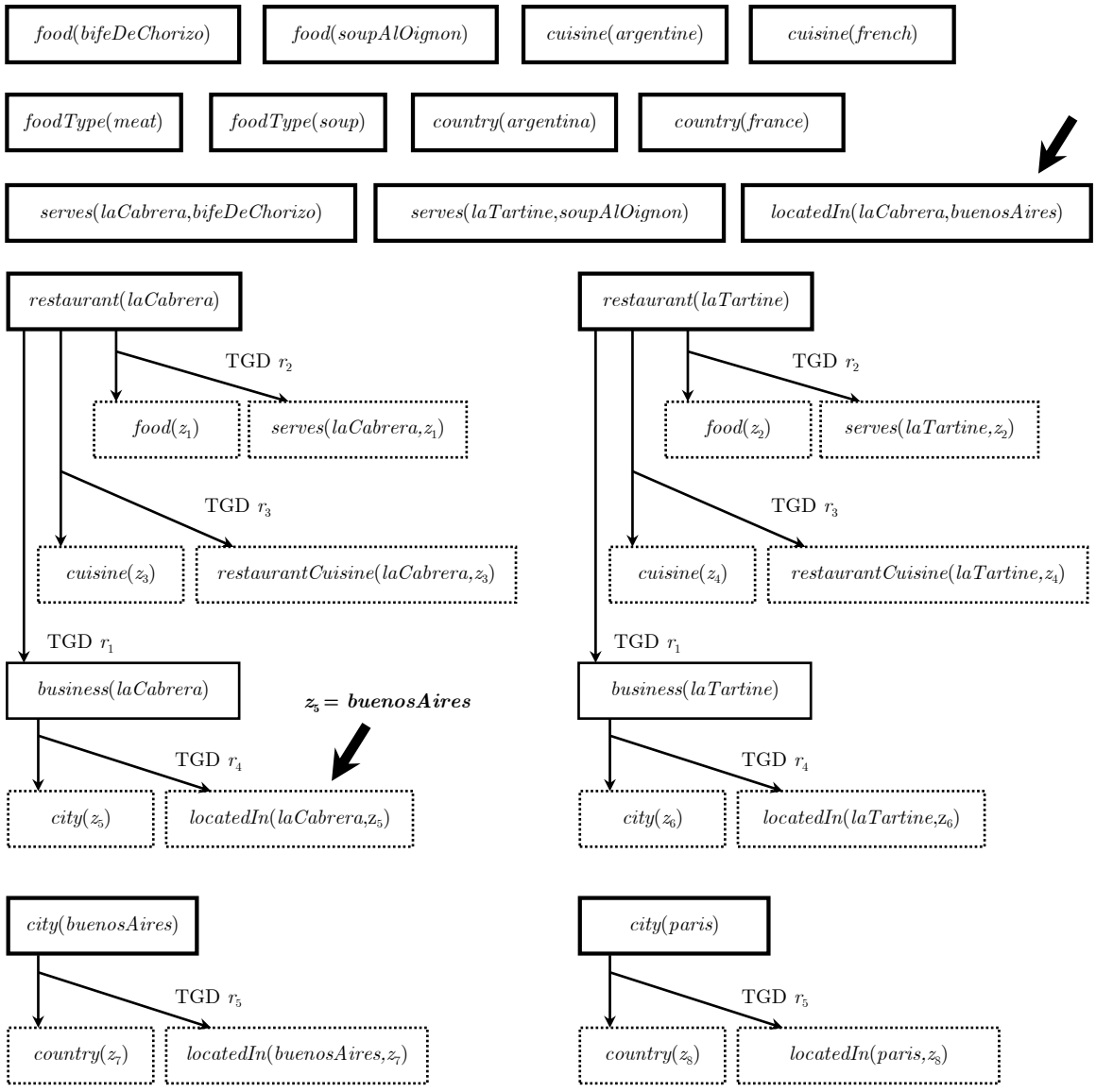

Figura 2.2: Representación gráfica del procedimiento *chase* cuando es llevado a cabo para la ontología *Datalog+/–* considerada en el Ejemplo 2.1. Figura extraída de [SMM+17].

mapea los átomos de $\phi(\mathbf{X})$ a átomos de D. Sea ϵ aplicable a D, entonces la *aplicación* de ϵ a D resulta en:

- El *chase* falla si $h(X_m)$ y $h(X_n)$ son constantes diferentes; esto se conoce como una *violación fuerte*. Caso contrario,
- reemplazar $h(X_m)$ por $h(X_n)$ si $h(X_m)$ es un valor nulo y $h(X_n)$ es una constante o $h(X_m)$ precede a $h(X_n)$ en el orden lexicográfico.

Como se dijo anteriormente, responder consultas generales sobre una ontología *Datalog+/–* es indecidible. Sin embargo, existen diferentes fragmentos de dicha familia de lenguajes que garantizan decidibilidad y varían en diferentes clases de complejidad cuando se lleva a cabo el mencionado proceso (ver [SMM+17] para un revisión resumida de tales fragmentos y las relaciones entre ellos).

A continuación, se describe un ejemplo sobre el resultado de la aplicación del *chase* sobre una base de conocimiento *Datalog+/–*.

Ejemplo 2.2 *La Figura 2.2 muestra la aplicación del procedimiento chase sobre la ontología Datalog+/– considerada en el Ejemplo 2.1. Los átomos en los rectángulos con bordes más grueso son parte de la base de datos, mientras que aquellos cuyos bordes tienen líneas punteadas corresponden a átomos con valores nulos (denotados con z_j). Asimismo, las flechas apuntan al mapeo de z_5 a la constante "buenosAires" cuando el procedimiento chase es llevado a cabo.*

Para facilitar el entendimiento, se puede mencionar que la TGD r_1 es aplicable en D debido a que existe un mapeo a partir de los átomos restaurant($laCabrera$) y restaurant($laTartine$) al cuerpo de la regla. La aplicación de r_1 genera los átomos business($laCabrera$) y business($laTartine$). Considere la siguiente consulta conjuntiva booleana:

$$q() = \exists X\, restaurant((laTartine)) \wedge locatedIn(laTartine, X),$$

esta consulta pregunta si existe una ubicación (locatedIn/2) para el restaurante (restaurant/1) laTartine. Para este caso, la respuesta es "Sí"; en el chase, es posible observar que después de aplicar las TGDs r_1 y r_4, se obtiene el átomo locatedIn($laTartine, z_6$) donde z_6 es un valor nulo—si se pregunta por el restaurante laCabrera se debería obtener

la misma respuesta porque el átomo $locatedIn(laTartine, z_5)$ *también es producido. Ahora, considere la consulta conjuntiva:*

$$q'(X, Y) = restaurant(X) \wedge locatedIn(X, Y);$$

para esta consulta, sólo se obtiene una respuesta, identificada como $(laCabrera, buenosAires)$*, debido a que el valor nulo* z_5 *eventualmente será mapeado a buenosAires. En el caso de laTartine, no se obtiene una respuesta correspondiente a la ciudad donde está ubicada: a pesar de esto, se sabe que existe una ciudad correspondiente a dicha ubicación, pero no es posible establecer cuál es. En cada modelo de la base de conocimiento,* z_6 *toma un valor diferente del dominio. Esto puede ser visto en el chase (un modelo universal de la base de conocimiento), debido a que* z_6 *no unifica con una constante cuando se lleva adelante el chase con* D *y* Σ*.*

■

2.2. El formalismo MANCaLog

Teniendo en cuenta las características de los problemas de interés (ver Capítulo 1 para mayores detalles) que se buscan abordar mediante el desarrollo del contenido de esta tesis, resulta de suma importancia disponer de la capacidad de modelar procesos de difusión en redes complejas. Por esta razón, resulta necesario realizar una descripción de alto nivel de los conceptos involucrados en el formalismo MANCaLog, presentado en [SSS13, SSC13]. Dicho formalismo será utilizado como base y sólo sufrirá ligeras modificaciones y generalizaciones en los desarrollos realizados en los Capítulos 4 y 5.

En este contexto, se asume que se dispone de un grafo dirigido $G = (V, E)$, el cual permite representar una red, donde V es un conjunto de nodos y E es un conjunto de arcos que conectan los mencionados nodos. Adicionalmente, se provee un conjunto de *etiquetas* $\mathcal{L} = \mathcal{L}_f \cup \mathcal{L}_{nf}$ donde $\mathcal{L}_f$ representa el subconjunto de *etiquetas fluentes* (que pueden ser modificadas durante el proceso de razonamiento) y $\mathcal{L}_{nf}$ representa el subconjunto de *etiquetas no fluentes* (que no pueden ser modificadas durante el proceso de razonamiento). La función de estas etiquetas es ser asignadas a componentes de la red (nodos y arcos) con la intención de definir características y atributos de los mismos. En este sentido, las etiquetas permiten representar conocimiento incierto a través de la asociación con intervalos de confianza para establecer niveles de creencia, por lo cual, para definir el

estado de la red resulta fundamental el concepto de *átomo de red* y *mundo*. El primero de ellos, el *átomo de red* es definido por el par $\langle L, bnd \rangle$ donde $L \in \mathcal{L}$ es una etiqueta y $bnd \subseteq [0, 1]$ es un intervalo de confianza. Luego, un *mundo* w hace referencia a un conjunto de átomos de red tales que para cada $L \in \mathcal{L}$ existe exactamente un átomo de la forma $\langle L, bnd \rangle$. Asimismo, los átomos de red pueden combinarse para obtener *fórmulas de red* a través de la utilización de conectivos lógicos estándar: $\wedge$, $\vee$, $\neg$.

Debido a que los mundos son conceptos claves para determinar el estado de la red, resulta necesario determinar cuando los átomos de red son satisfechos por dichos mundos, y esto sucede para un átomo de red $\langle L, bnd \rangle$ si y sólo si existe otro átomo de red $\langle L, bnd' \rangle$ perteneciente al mundo considerado tal que $bnd' \subseteq bnd$.

La representación del conocimiento propiamente dicha es realizada a través de dos unidades provistas para tal fin: los *hechos* y las *reglas*. Los hechos permiten definir las propiedades que valen para los componentes de red vinculándolos con átomos de red considerados en un conjunto de puntos de tiempo determinados. Esto significa que la noción de *tiempo* se hace presente y se debe destacar que para este caso se considera el *tiempo* como un conjunto de puntos discretos en el rango $[0, t_{max}]$. Específicamente, los hechos tiene la siguiente forma general:

$$(A, C) : [t_1, t_2]$$

donde A es átomo de red, C es un componente de la red y $[t_1, t_2] \subseteq [0, t_{max}]$.

Por otro lado, las *reglas* permiten establecer la dinámica del conocimiento en la red compleja haciendo que el conocimiento disponible evolucione a través del tiempo mediante la realización de un proceso conocido como *difusión*. Estas reglas del lenguaje tienen la siguiente forma general:

$$\underbrace{L}_{objetivo} \overset{\Delta t}{\leftarrow} \underbrace{f}_{criterio\ de\ objetivo}, \underbrace{(nc_{edge}, nc_{node}, h)}_{criterio\ de\ vecinos} \underbrace{if}_{función\ de\ influencia}$$

donde:

- L es una etiqueta.
- $\Delta t \geq 0$.
- f es una fórmula sobre átomos de red basados en etiquetas no fluentes.

- nc_{edge}, nc_{node} son fórmulas sobre átomos de red basados en etiquetas no fluentes y relativas a átomos de arco y nodos, respectivamente.
- h es una conjunción de átomos de red (posiblemente con etiquetas fluentes).
- if es una función de influencia.

Intuitivamente, la regla puede ser leída de la siguiente forma: "La etiqueta L de los nodos que satisfacen el criterio descripto por f son influenciados (luego de Δt pasos de tiempo) por un conjunto de vecinos que satisfacen el criterio descripto por nc_{edge}, nc_{node} y h a un nuevo grado de creencia determinado por la función if."

Puede deducirse que las funciones de influencia cumplen un rol clave y dentro de este lenguaje sólo se consideran aquellas que tienen la siguiente forma:

$$if : \mathbb{N} \times \mathbb{N} \rightarrow [0, 1] \times [0, 1]$$

Intuitivamente, dichas funciones sólo consideran la relación entre la cantidad de vecinos que efectivamente tiene un nodo y la cantidad de vecinos que satisfacen un determinado criterio (vecinos calificados) generalmente determinado por la regla en cuestión.

En términos semánticos, el formalismo contempla la existencia de *estructuras de datos* que mapean puntos de tiempo a otras estructuras de datos, las cuales a su vez relacionan componentes de la red con mundos. De esta manera es posible determinar cuál es el estado de la red que satisface todo el conocimiento representado en los hechos y reglas disponibles. Concretamente, esto es realizado a través de la implementación de un operador de punto fijo que comienza a partir de un estado inicial de la red con incertidumbre total hasta alcanzar un estado final estable. Teniendo en cuenta las características particulares de este lenguaje, esto es posible llevarlo a cabo en tiempo polinomial en complejidad de datos.

2.3. Sumario

A lo largo de este capítulo se ha realizado un repaso general de los conceptos relativos a la familia de lenguajes ontológicos *Datalog+/–* y el formalismo MANCaLog para el modelado de procesos de difusión en redes complejas. Los conceptos abordados aquí serán esenciales para el desarrollo de todo el contenido presentado en esta tesis, pero particularmente *Datalog+/–* resulta fundamental para el desarrollo del Capítulo 3, y junto a MANCaLog son muy necesarios para los Capítulos 4 y 5.

Capítulo 3

Hacia un sistema de generación automatizada de hipótesis

En este capítulo, se presentará una primera aproximación para un sistema de generación automatizada de hipótesis considerando dos puntos de vista diferentes. Para este caso, el sistema será diseñado para detectar un tipo particular de comportamiento malicioso denominado *deduplicación adversarial*. En el primero de estos enfoques, el énfasis está puesto en una estrategia basada en lenguajes lógicos, mientras que en el segundo se puntualiza en una estrategia apoyada en técnicas basadas en aprendizaje automatizado. Más allá de estas diferencias, ambos enfoques son parte de los primeros pasos hacia el desarrollo de herramientas generales para abordar diferentes problemas de este estilo. Asimismo, ambos enfoques fueron desarrollados considerando que un abordaje exitoso en este contexto requiere:

- *Utilización de conocimiento experto del dominio de aplicación.*
- *Utilización de conocimiento obtenido a partir de los datos disponibles.*
- *Generación de hipótesis sobre las instancias que resuelven el problema.*
- *Validación de hipótesis por parte de analistas humanos.*

El problema de *deduplicación* de objetos (también conocido como *resolución de entidades*) es un problema clásico en el área de limpieza de datos [GM12]; la idea básica es que las bases de datos contienen objetos que están *potencialmente duplicados* (registros

aparentemente diferentes que corresponden a la misma entidad u objeto en el mundo real) y que necesitan ser identificados y mezclados [BN08, EIV07]. Tradicionalmente, las técnicas para resolver este problema se basan en similitudes sobre atributos pertenecientes a los pares que están bajo análisis [GM12]. Sin embargo, la definición del problema considerado aquí es una generalización de las definiciones mayormente tomadas en la literatura donde situaciones no deseadas en las que el hecho de que existan múltiples registros que se corresponden con el mismo objeto del mundo real es causado por una combinación de errores administrativos en la entrada de datos (simples errores tipográficos), ambigüedad en nombres u otros atributos (por ejemplo, el resultado de transcripciones tales como "Kurt Gödel" mapeado a "Kurt Goedel"), definición de formatos y abreviaciones inconsistentes, valores ausentes (por ejemplo, números de teléfonos no proporcionados) o valores cambiantes (por ejemplo, un registro que tiene un número de teléfono mientras que otro tiene uno actualizado). Aunque estas suposiciones reflejan adecuadamente muchos escenarios del mundo real, éstas no pueden ser asumidas para otras variantes de dicho problema. Un ejemplo de tales escenarios es en el dominio de ciberseguridad, particularmente dentro de foros y mercados de *hackers* maliciosos en la *Darkweb* y *Deepnet*. En este contexto se puede observar una característica interesante: aunque actores maliciosos desean mantenerse anónimos respecto de organismos de seguridad que puedan estar monitoreando, también necesitan conservar una reputación para con sus clientes y seguidores, por lo que requieren en cierta medida poder ser identificados. Es así que, típicamente, un mismo actor opera bajo diferentes nombres, manteniendo ciertos aspectos constantes de manera que otros puedan reconocerlos. Además, y tal vez lo más importante, también dejan pistas involuntarias que pueden ser útiles en esfuerzos de deduplicación. Esto es lo que se ha bautizado como el problema de *deduplicación adversarial*. A continuación, se presenta un ejemplo simplificado del tipo de información que se encuentra disponible en los mercados y foros de *hackers* en la *Darknet* y *Deepnet*.

Ejemplo 3.1 *Considere el siguiente esquema relacional respecto a la información sobre usuarios y sus publicaciones en foros de* hackers *en la* Darknet *y* Deepnet*:*

$$user(UID, Nick)$$
$$post(PID, UID, TID)$$
$$hasPosted(UID, TID)$$
$$writStSim(UID, UID)$$

el cual permite representar usuarios de esos foros, publicaciones hechas por esos usuarios, temas de las publicaciones, qué usuarios han publicado sobre qué temas y la similitud de estilo de escritura entre usuarios. Además, UID y PID son las claves primarias para las relaciones user y post, respectivamente. Entonces, una instancia de base de datos D_1 sobre este esquema, podría contener la siguiente información:

$$\begin{aligned} D_1 \ = \{ \ & d_1 : post(p_1, a_1, t_1), \\ & d_2 : post(p_2, a_2, t_2), \\ & d_3 : post(p_3, a_3, t_3), \\ & d_4 : user(a_1, blackstar), \\ & d_5 : user(a_2, archer), \\ & d_6 : user(a_3, angel\_eyes), \\ & d_7 : writStSim(a_1, a_2)\} \end{aligned}$$

donde t_1 es "private sqli tool", t_2 es "Sql Injection" y t_3 es "trojan servers as a backdoor".

■

3.1. Un enfoque basado en reglas lógicas

A continuación, se presenta una primera aproximación para un sistema de generación automatizada de hipótesis, en el cual se pone mayor énfasis en la utilización de reglas que permitan generar dichas hipótesis con la intención de abordar el problema de deduplicación adversarial. Si bien el enfoque está basado en un formalismo de razonamiento y representación de conocimiento, también se busca que se complemente con los resultados provenientes de aplicar técnicas de aprendizaje automatizado. En este caso, el formalismo lógico utilizado es una extensión de *Datalog+/–*, el cual considera eventos probabilísticos y negación estratificada. La utilización de eventos probabilísticos es necesaria debido a las características de incertidumbre del problema abordado y la negación estratificada es requerida debido al tipo de reglas consideradas, las cuales más adelante serán descriptas con mayor detalle.

3.1.1. *Datalog+/–* probabilístico con negación estratificada

En esta sección, se presenta una extensión de *Datalog+/–* que considera eventos probabilísticos (definido en [GLMS13]) y además en este caso se incorpora la posibilidad de

modelar negación estratificada. Se asume la existencia de los siguientes conjuntos disjuntos (infinitos contables): un conjunto Δ de constantes, un conjunto $\mathbf{V}$ de variables y un conjunto Δ_n de nulos. Constantes diferentes representan valores diferentes (suposición de nombre único), mientras que nulos diferentes pueden representar el mismo valor. También se asume un orden lexicográfico sobre $\Delta \cup \Delta_n$, con cada símbolo en Δ_n siguiendo a todos los símbolos en Δ. $\mathbf{X}$ denota la secuencia de variables $X_1, \ldots, X_k$ con $k > 0$.

Considere el esquema relacional $\mathcal{R}$ con un dominio de datos posiblemente infinito Δ, un conjunto finito de predicados de base de datos (relaciones) $r \in \mathcal{R}$ y un conjunto de predicados pre-calculados para evaluar comparaciones que son " $=$ ", " $\neg$ ", " $\leq$ ". Cada relación $r \in \mathcal{R}$ tiene una aridad asociada n, entonces r tiene la forma $r(A_1, \ldots, A_n)$, donde cada atributo A_i tiene un dominio asociado $Dom(A_i) \subseteq \Delta$. Por cuestiones de simplicidad, se puede suponer que los A_i son diferentes y que los diferentes predicados no comparten atributos. Sin embargo, diferentes atributos pueden compartir el mismo dominio. Luego, una fórmula atómica (o átomo) tiene la forma $p(t_1, \ldots, t_n)$, donde $t_i \in Dom(A_i) \cup \mathbf{V}$ y una conjunción de átomos con frecuencia se la identifica con el conjunto de todos sus átomos. Una instancia de base de datos D para $\mathcal{R}$ es un conjunto (posiblemente infinito) de átomos básicos (*ground*) con predicados de $\mathcal{R}$ y argumentos de Δ.

También es importante recordar que, un *homomorfismo* es un mapeo $h : \Delta \cup \mathbf{V} \cup \Delta_n \to \Delta \cup \mathbf{V} \cup \Delta_n$ tal que (i) si $c \in \Delta$ entonces $h(c) = c$, (ii) si $c \in \mathbf{V} \cup \Delta_n$ entonces $h(c) \in \Delta \cup \Delta_n$, y (iii) h puede extenderse naturalmente a átomos, conjuntos de átomos y conjunciones de átomos, es decir, $h(p(t_1, \ldots, t_n)) = p(h(t_1), \ldots, h(t_n))$.

Por otro lado, debido a la necesidad de modelar *negación estratificada*, se vuelve importante hacer la distinción entre la parte *positiva* (sin negación) y la parte *negativa* (con negación) de una conjunción de átomos $\Phi(\mathbf{X})$, lo cual es denotado como $pos(\Phi(\mathbf{X}))$ y $neg(\Phi(\mathbf{X}))$, respectivamente.

Estos detalles adicionales también tienen un impacto en la definición de las reglas. Considere una regla arbitraria r, ya sea TGD, NC, o EGD, y denótese $body(r)$ al cuerpo de dicha regla; entonces, esa parte de la regla puede distinguirse entre la parte positiva y la parte negativa, que en estos casos particulares se denota como $body^+(r)$ y $body^-(r)$, respectivamente. Esto también resultará útil más adelante, cuando se defina la semántica operacional del lenguaje a través de una modificación del procedimiento conocido como *chase*. Adicionalmente, se asume que el programa Σ es *estratificado* [GRS14], lo cual

significa que Σ puede ser particionado en una serie de conjuntos mutuamente disjuntos nombrados como $\Sigma_1, ..., \Sigma_k$ tal que:

- Si Σ_i contiene un predicado p que aparece positivamente (no negado) en alguna regla, entonces no hay ninguna regla en $\Sigma_{i+1} \cup ... \cup \Sigma_k$ con p en su cabeza.
- Si Σ_i contiene un predicado p que aparece negado en alguna regla, entonces no hay ninguna regla en $\Sigma_i \cup ... \cup \Sigma_k$ con p en la cabeza.

Se dice entonces que $\langle \Sigma_1, ..., \Sigma_k \rangle$ es una *estratificación* de Σ. Nuevamente, como se mencionó en el Capítulo 2 , en este caso también puede asumirse que las TGDs consideradas tienen un solo átomo en su cabeza.

A continuación, se define satisfacción para TGDs, EGDs y NCs considerando negación estratificada.

Satisfacción de TGDs. Una instancia de base de datos I sobre un esquema relacional $\mathcal{R}$ *satisface* una TGD σ, si cada vez que existe un homomorfismo h que mapea los átomos de $body^+(\sigma)$ a átomos de I, pero h no mapea los átomos de $body^-(\sigma)$ a átomos de I, entonces existe una extensión h' de h que mapea los átomos de $head(\sigma)$ a átomos de I.

Satisfacción de EGDs. Una instancia de base de datos I sobre un esquema relacional $\mathcal{R}$ *satisface* una EGD $\epsilon : body^+ \wedge body^- \rightarrow X_i = X_j$, si cada vez que existe un homomorfismo h que mapea los átomos de $body^+(\epsilon)$ a átomos de I, pero h no mapea los átomos de $body^-(\epsilon)$ a átomos de I, entonces se verifica que $h(X_i) = h(X_j)$. Caso contrario, se dice que ϵ se encuentra violado en D.

Satisfacción de NCs. Una instancia de base de datos I sobre un esquema relacional $\mathcal{R}$ *satisface* una NC $\mu : body^+ \wedge body^- \rightarrow \bot$, si no existe un homomorfismo h que mapea los átomos de $body^+(\epsilon)$ a átomos de I, pero además h no mapea los átomos de $body^-(\epsilon)$ a átomos de I. Caso contrario, se dice que μ se encuentra violado en D.

Anotaciones probabilísticas

De acuerdo a lo propuesto en [GLMS13], a fin de incorporar conocimiento probabilístico a *Datalog+/–* mediante anotaciones, se asume que se dispone de un conjunto finito de constantes Δ_M, un conjunto de nombres de predicados $\mathcal{R}_M$ tal que $\mathcal{R} \cap \mathcal{R}_M = \emptyset$ y un conjunto infinito de variables $\mathbf{V}_M$, tales que $\mathbf{V} \cap \mathbf{V}_M = \emptyset$. Estos conjuntos dan origen

a las correspondientes bases de Herbrand y universos de Herbrand compuestos de todos los posibles átomos (básicos y no básicos, respectivamente) que pueden ser formados, los cuales se denotan con $\mathbf{H}_M$ y $\mathbf{U}_M$, respectivamente.

Ahora, considere un modelo probabilístico que represente una distribución de probabilidad conjunta sobre un conjunto (finito) $X = \{X_1, ..., X_n\}$ de variables booleanas aleatorias, tales que existe un mapeo 1 a 1 desde X al conjunto de todos los átomos básicos en $\mathbf{H}_M$. Como ejemplos de este tipo de modelos probabilísticos se puede mencionar la *Lógica de Markov* [RD06] y las *Redes Bayesianas* [Pea89]. Una anotación (probabilística) λ, relativa a M, es un conjunto (finito) de expresiones $\langle A = x_i \rangle$, donde A es un átomo sobre $\mathcal{R}_M$, Δ_m y V_M, $x_i \in \{0, 1\}$. Luego, una anotación probabilística es *válida* si y sólo si para dos expresiones cualesquiera diferentes $A = x, B = y \in \lambda$, no existe una substitución θ tal que $\theta(A) = \theta(B)$. Asimismo, un *escenario probabilístico* es una anotación probabilística válida λ, para la cual $|\lambda| = |X|$ y todas las expresiones $\langle A = x_i \rangle$ son tales que A está instanciado. El conjunto de escenarios en M se denotará como $scn(M)$. Una anotación probabilística es equivalente a una fórmula que consiste de la disyunción de todos los escenarios que pertenecen al conjunto denotado por la anotación. Por lo tanto, haciendo un ligero abuso de la notación, a veces se podrán combinar anotaciones con operadores lógicos.

De manera intuitiva, una anotación probabilística es usada para describir un evento en el cual las variables aleatorias en un modelo probabilístico M son compatibles con la configuración de las variables descritas por λ, es decir, cada X_i tiene el valor x_i. Esto resulta útil para añadir anotaciones probabilísticas λ a fórmulas del lenguaje para producir fórmulas anotadas $F : \lambda$, donde F es un átomo, una TGD, o una EGD. Intuitivamente, la anotación significa que F solo puede sostenerse cuando sucede el evento asociado con λ. Observe que cuando los valores de una variable aleatoria no son especificados, en una anotación probabilística, la variable se encuentra *sin restricciones*, en particular, una fórmula anotada con una anotación probabilística vacía significa que la fórmula se sostiene en todos los mundos. Adicionalmente, una fórmula anotada $F : \lambda$ es llamada *átomo probabilístico* cuando F es un átomo, una *TGD probabilística* (pTGD) cuando F es una TGD, y una *EGD probabilística* (pEGD) cuando F es una EGD.

Base de conocimiento *Datalog+/–* probabilística. Una base de conocimiento *Datalog+/–* probabilística es una terna $KB = (O, M, af)$, donde O es un conjunto de

átomos, TGD y EGDs, M es un modelo probabilístico y af es una función de anotación que mapea fórmulas a anotaciones probabilísticas relativas a M.

Por cuestiones de facilidad en la presentación, en ocasiones se usará la notación $\sigma : e$ para denotar $af(\sigma) = e$. Asimismo, para una $KB = (O, M, af)$ y un escenario $\lambda \in scn(M)$, la base de conocimiento *Datalog+/–* (no probabilística) *inducida* por λ, es denotada como O_λ. En términos más formales, O_λ consiste de todas las fórmulas F_i tales que $\lambda_i \subseteq \lambda$ para algún $F_i : \lambda_i \in O$.

Ejemplo 3.2

$$(w_1) \quad user(UID_1, N_1) \wedge hasPosted(UID_1, T_1) \wedge user(UID_2, N_2)$$
$$hasPosted(UID_2, T_2) \wedge friend(UID_1, UID_2) \rightarrow hasPosted(UID_1, T_2) : e_1(T_1, T_2)$$

La intuición de esta regla es: "Todo usuario que ha publicado sobre un tema T_1 y que es amigo de otro que ha publicado sobre un tema T_2, también ha publicado sobre T_2". Observe que $e_1(T_1, T_2)$ hace referencia a un evento asociado con la probabilidad de que un usuario publique algo relacionado a T_2 dado que ya ha publicado algo sobre T_1.

$$(\epsilon_1) user(UID_1, N_1) \wedge user(UID_2, N_1) \rightarrow UID_1 = UID_2$$

En este caso la intuición de la regla es: "Dos usuarios no pueden compartir el mismo nombre de usuario". Observe que en este caso particular la regla está anotada con el evento vacío, es decir $af(\epsilon_1) = \emptyset$, por lo que dicho evento siempre se sostiene.

■

Cabe destacar que los modelos probabilísticos, como el utilizado en el ejemplo, pueden ser aprendidos a partir de información del dominio, lo cual permitiría inferir sobre posibles usuarios duplicados y correlaciones entre ciertos eventos, tales como publicaciones de usuarios dados temas específicos.

3.1.2. Un chase en dos fases

Con el objetivo de poder modelar los tipos de reglas de interés acordes al problema que se intenta abordar, se necesita extender la semántica operacional clásica de *Datalog+/–* utilizando, como suele hacerse para estos casos, una versión modificada del procedimiento

chase. Como se mencionó en el Capítulo 2, este procedimiento es la base de la semántica para la aplicación de reglas extendiendo el algoritmo *chase* (*oblivious*) para verificación de implicaciones de dependencias de datos [MMS79, BV84] o para completar bases de datos deductivas [AHV95]. Tener en cuenta que a diferencia de [GLMS13] donde se presenta un procedimiento *chase* para ontologías *Datalog+/–* probabilísticas, en este caso se presenta un *chase de dos fases* que además incluye negación estratificada y EGDs. Nuevamente, como primer paso se requiere definir lo que constituye uno de los bloques de construcción para este procedimiento, en relación a uno de los tipos de reglas que consideramos: TGDs. Adicionalmente, en este escenario, se necesita considerar cómo se realiza la propagación de anotaciones probabilísticas.

Regla de chase para pTGD. Considere una base de conocimiento $KB = (O = (D, \Sigma), M, af)$ y una pTGD $\sigma \in O$ de la forma $\phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} \Psi(\mathbf{X}, \mathbf{Z}) : e_\sigma$; entonces, σ es *aplicable* a D si y solo si existe un homomorfismo h tal que sucede uno de los siguientes casos:

a) 1) h mapea átomos de $pos(\Phi(\mathbf{X}, \mathbf{Y}))$ y $neg(\Phi(\mathbf{X}, \mathbf{Y}))$ a los átomos probabilísticos $\{\alpha_1 : e_1, ..., \alpha_k : e_k\} \subseteq D$ y $\{\alpha_{k+1} : e_{k+1}, ..., \alpha_j : e_j\} \not\subseteq D$, y

   2) $h(e_\sigma \wedge \bigwedge_{i=1,...,k} e_i)$ es una anotación probabilística válida.

b) 1) h mapea los átomos de $pos(\Phi(\mathbf{X}, \mathbf{Y}))$ y $neg(\Phi(\mathbf{X}, \mathbf{Y}))$ a los átomos probabilísticos $\{\alpha_1 : e_1, ..., \alpha_k : e_k\} \subseteq D$ y $\{\alpha_{k+1} : e_{k+1}, ..., \alpha_j : e_j\} \subseteq D$, y

   2) $h(e_\sigma \wedge \bigwedge_{i=1,...,k} e_i \wedge \bigwedge_{i=k+1,...,j} \neg e_i)$ es una anotación probabilística válida.

Supongamos que σ es aplicable a D, y h' es un homomorfismo que extiende h de la siguiente forma: para cada término $X_i \in \mathbf{X}$, $h'(X_i) = h(X_i)$ y $h'(Z) = z_j$, donde Z está existencialmente cuantificado en $head(\sigma)$ y z_j es un nuevo valor *nulo*, es decir, $z_j \in \Delta_n$, z_j no está presente en D, y z_j sigue el orden lexicográfico de todos los nulos que ya fueron incorporados. La aplicación de σ en D agrega a D uno de los siguientes átomos probabilísticos, de acuerdo al caso donde σ es aplicable:

a) $h'(\Psi(\mathbf{X}, \mathbf{Z})) : h'(e_\sigma \wedge \bigwedge_{i=1,...,k} e_i)$.

b) $h'(\Psi(\mathbf{X}, \mathbf{Z})) : h'(e_\sigma \wedge \bigwedge_{i=1,...,k} e_i \wedge \bigwedge_{i=k+1,...,j} \neg e_i)$.

Como segundo paso, se aplican las dependencias de generación de igualdad. La razón para realizar esta parte del procedimiento como un paso separado sobre el resultado del

chase para pTGD, es buscar evitar interacciones no deseadas entre TGDs y EGDs que pueden llevar a un estado de indecidibilidad [CGL12], teniendo en cuenta que la forma estándar para evitar esto es a través de separabilidad, lo cual es una restricción demasiado fuerte para los casos considerados aquí.

Regla de chase para pEGD. Considere una base de conocimiento *Datalog+/–* probabilística, definida como $KB = (O = (D, \Sigma), M, af)$, y una pEGD $\epsilon \in \Sigma$ de la forma $\Phi(\mathbf{X}) \rightarrow X_m = X_n : e_\epsilon$. Entonces, ϵ es aplicable a D si y sólo si existe un homomorfismo h tal que:

a) h mapea los átomos de $\Phi(\mathbf{X})$ a átomos probabilísticos $\{\alpha_1 : e_1, ..., \alpha_k : e_k\} \subseteq D$, y

b) $h(e_\epsilon \wedge \bigwedge_{i=1,...,k} e_i)$ es una anotación probabilística válida.

Supongamos que ϵ es aplicable a D y que $\Phi_{X_m} = \{\alpha_1 : e_1, ..., \alpha_l : e_l\} \subseteq \{\alpha_1 : e_1, ..., \alpha_k : e_k\}$ es el conjunto de átomos probabilísticos tal que X_m está presente en cada $\alpha_{i=1,...,l}$; entonces la aplicación de ϵ en D resulta en:

a) el *chase* falla si $h(X_m)$ y $h(X_n)$ son constantes; esto se conoce como una *violación fuerte*. En otro caso,

b) para todo $\alpha_i : e_i \in \Phi_{X_m}$ reemplazar $h(X_m)$ por $h(X_n)$ y e_i por $h(e_\epsilon \wedge \bigwedge_{j=1,...,k} e_j)$, si $h(X_m)$ es un valor nulo y $h(X_n)$ es una constante, o $h(X_m)$ precede a $h(X_n)$ en el orden lexicográfico.

A continuación, se aborda la forma en que se puede utilizar el procedimiento *chase* para responder consultas de deduplicación probabilísticas.

3.1.3. Programas de deduplicación

Ahora, se define una subclase de programas *Datalog+/–* probabilísticos diseñados específicamente para capturar, de manera declarativa, los requerimientos necesarios a fin de realizar una tarea de deduplicación, situados en un entorno adversarial y bajo la suposición de mundo abierto. Tomando en cuenta esto, se asume que existe un conjunto $\mathcal{E} \subseteq \mathcal{R}$ identificadas como *relaciones de entidad*, es decir, relaciones que representan entidades de interés en el dominio de aplicación. Las tuplas en estas relaciones tienen identificadores

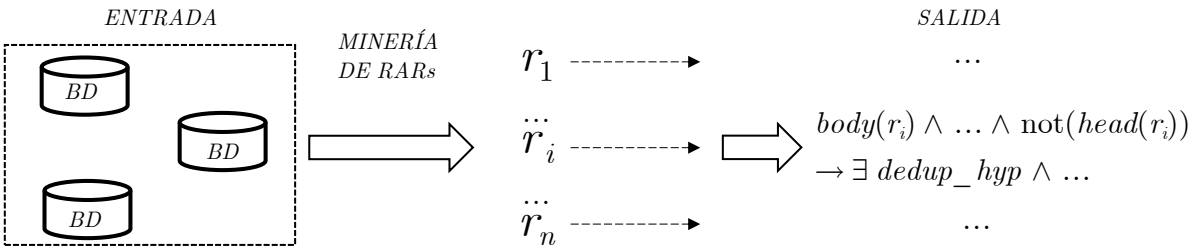

Figura 3.1: Descripción general de un posible proceso de aprendizaje de dhTGDs.

(claves) que permiten comparar extensiones del mismo predicado en diferentes instancias y poder obtener una traza de sus cambios. Los identificadores de tuplas pueden incorporarse simplemente agregando a cada predicado $E \in \mathcal{E}$ un atributo extra (usualmente denotado con T), el cual actúa como una clave primaria en E. Finalmente, también hay un predicado predefinido llamado *dedup_hypth*. La idea de esta propuesta está inspirada en las *relational matching dependencies* (RMDs) [BKL13], que proveen una forma declarativa de definir condiciones bajo las cuales registros (o valores de atributos) deberían hacerse iguales (también se establece por medio de funciones de *matching* cómo debería establecerse esta igualdad). Sin embargo, las RMDs se desarrollan bajo una suposición de mundo cerrado, es decir, todo lo que no se encuentra contenido en la instancia de la base de datos se asume que es falso (o no existe). Considerando el tipo de aplicaciones del dominio en el que se está interesado, dicha suposición no es realista, y por lo tanto es conveniente considerar que las instancias de base de datos con las que se está trabajando son incompletas. Bajo tales escenarios, las RMDs no son suficientemente expresivas para capturar todos los *matching* potenciales, debido a que en ocasiones no se pueden establecer condiciones específicas para encontrarlas, pero si es posible crear *hipótesis* basadas en evidencia (más débil) de otra naturaleza. Por lo tanto, en estos casos la posibilidad de *invención de valores* proporcionada por las reglas existenciales se vuelve crucial.

Las pTGDs se modifican ligeramente, de manera que puedan ser utilizadas para inferir *matchings* potenciales (o *hipótesis de matching*). Para esto, se asume que para todo atributo A_j que aparece en una relación n-aria $R \in \mathcal{R}$, donde el esquema de R es $R(A_1, ..., A_n)$, existe una relación de similitud binaria $\approx_{A_j}$ que es reflexiva y asimétrica. Por lo cual, se dispone de la capacidad de expresar condiciones de similitud sobre valores de atributos como las RMDs, dado que el predicado "$\approx$" simplemente debe ser usado en el cuerpo.

dhTGDs. Una *Dependencia de Generación de Tupla con Hipótesis de Deduplicación* (o dhTGD) σ es simplemente una pTGD de la siguiente forma:

$$\forall \mathbf{XY} \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\ \mathit{dedup\_hypth}(X, Z) \wedge \Gamma(\mathbf{X}, \mathbf{Z}) : e$$

donde $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ son secuencias de variables, X y Z son variables y $X \in \mathbf{X}$, $Z \in \mathbf{Z}$, $\Phi(\mathbf{X}, \mathbf{Y})$ es una conjunción de átomos y átomos negados sobre $\mathcal{R}$, $\Gamma(\mathbf{X}, \mathbf{Z})$ es una conjunción de átomos sobre $\mathcal{R}$ y e es una anotación probabilística válida.

Las dhTGDs son una clase especial de pTGDs que generan *hipótesis de deduplicación*. Esencialmente, proporcionan la capacidad de crear hipótesis sobre *matchings* potenciales, que se mantienen y que luego pueden llegar a confirmarse cuando se dispone de nueva información en la fuente de datos. En términos informales, las dhTGDs capturan situaciones en las cuales se tienen razones para creer que una entidad puede estar duplicada, pero no se conoce *cuáles* son las otras entidades que le corresponderían. La Figura 3.1 describe una posible forma en la que las dhTGDs podrían ser automáticamente derivadas a partir de los datos. Como puede observarse, los algoritmos de minería de reglas de asociación relacional son aplicados sobre un conjunto de fuentes de datos disponibles; cada una de tales reglas da origen a una dhTGD como salida, caracterizando situaciones que violan la correspondiente regla de asociación y, por lo tanto, son la base de una hipótesis de entidad duplicada. Más específicamente, un algoritmo de minería de reglas de asociación relacional (tales como [DT99]) puede ser usado para descubrir patrones en fuentes de datos disponibles, las cuales son reglas de la forma $r : \mathit{body}(r) \rightarrow \mathit{head}(r)$; por lo tanto, se considera que las entidades que satisfacen las condiciones del cuerpo (*body(r)*), pero no aquellas de la cabeza ($\mathit{head}(r)$), son potenciales candidatos para entidades con identidades duplicadas (ver Ejemplo 3.3 para un escenario concreto).

pEGDs como complemento de dhTGDs. Para esta propuesta, las dependencias de generación de igualdad son aplicadas como un segundo paso después de que todas las dhTGDs son aplicadas; el objetivo de este parte del procedimiento es completar la base de datos con información que puede estar disponible relativa a los valores nulos en los átomos *dedup_hypth*. Un tipo de pEGDs, con potencial utilidad en estas situaciones, podría derivarse automáticamente a partir de una dhTGD σ tomando como base la negación de la parte negada de σ ($\mathit{body}^{-}(\sigma)$) para formar el cuerpo de la pEGD. La Figura 3.2 ilustra este procedimiento y, por otro lado, a continuación se presenta un nuevo ejemplo basado en el Ejemplo 3.1.

Ejemplo 3.3 *Considere el Ejemplo 3.1 y suponga que se ha obtenido una regla de asociación relacional que declara que "Si un usuario ha hecho alguna publicación en relación a algún tema t_1, entonces también ha publicado referido al tema t_2". Esto puede ser codificado en la siguiente dhTGD:*

$$(\omega_2) \quad user(UID, N) \wedge hasPosted(UID, t_1) \wedge not(hasPosted(UID, t_2)) \rightarrow \\ \exists Z\, dedup\_hypth(UID, Z) \wedge hasPosted(Z, t_2)$$

La intuición de esta regla es "Si un usuario tiene alguna publicación referida al tema t_1, pero no tiene ninguna publicación referida al tema t_2, entonces se genera la hipótesis de que existe otro usuario que le corresponde (cuya identidad es actualmente desconocida) y que ha hecho alguna publicación referida al tema t_2".

Como se muestra en la Figura 3.2, una dependencia de generación de igualdad puede ser derivada a partir de ω_2 utilizando los átomos de $body^-(\omega_2)$ y algunas condiciones adicionales que, cuando son verdad, pueden ayudar a obtener la identidad de la entidad, hasta ahora desconocida, que participa en la hipótesis de deduplicación:

$$(\epsilon_2) \quad dedup\_hypth(UID_1, UID_2) \wedge user(UID_3, N) \wedge hasPosted(UID_3, t_2) \wedge \\ writStSim(UID_1, UID_3) \rightarrow UID_2 = UID_3$$

La intuición de esta regla es "Si se tiene una hipótesis de deduplicación entre un usuario UID_1 y un usuario UID_2 (desconocido en ese momento, por la forma en que se aplican las reglas), además, UID_3 ha hecho alguna publicación referida al tema t_2 y el estilo de escritura de UID_1 es similar al de UID_3, entonces UID_2 y UID_3 se igualan y la hipótesis se completa, por lo que UID_1 puede corresponderse a UID_3". ∎

Tomando en cuenta todo lo que se ha desarrollado hasta aquí, el siguiente ejemplo ilustra cómo el procedimiento *chase* se desarrolla sobre el escenario del Ejemplo 3.1

Ejemplo 3.4 *Considere la base de datos D_1 presentada en el Ejemplo 3.1, una base de conocimiento Datalog+/– probabilístico extendida representando un programa de deduplicación $KB_2 = (O_2, M, af)$ donde $O_2 = (D_1, \Sigma_2)$ y M_1 es el mismo modelo probabilístico del Ejemplo 3.2. El conjunto Σ_2 contiene la TGD (clásica):*

$$(\sigma_1) \quad user(UID, N) \wedge post(PID, UID, T) \rightarrow hasPosted(UID, T),$$

y también contiene la dhTGD ω_2 y la pEGD ϵ_2 tomadas del Ejemplo 3.3. La función de anotación af es definida de la siguiente manera:

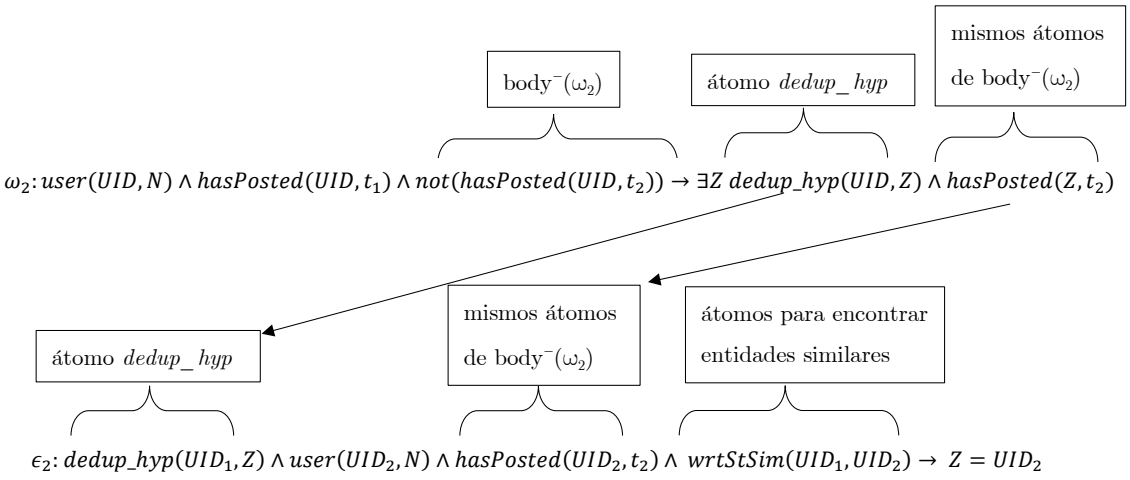

Figura 3.2: Ilustración de un posible procedimiento con el cual una pEGD puede ser derivada a partir de una dhTGD existente (σ).

- *af*$(\omega_2) = e_1(t_1, t_2)$, *donde el evento* $e_1(t_1, t_2)$ *está asociado con la probabilidad de que un usuario haya publicado algo relacionado a* t_2, *dado que ha publicado algo relacionado a* t_1.

- *af*$(\epsilon_2) = e_2$, *donde el evento* e_2 *está asociado con la probabilidad de que dos usuarios con similar estilo de escritura se correspondan y constituyan, en realidad, un único usuario.*

La Figura 3.3 ilustra el chase para este ejemplo. Los rectángulos con bordes en negrita representan átomos en D_1, *los rectángulos con bordes de lineas punteadas representan átomos que contienen valores nulos, y los rectángulos con los bordes comunes representan átomos sin valores nulos obtenidos de alguna dhTGD o pEGD. Los eventos asociados con cada átomo se representan por debajo del rectángulo correspondiente; todos los átomos en* D_1 *se asocian con eventos vacíos y los eventos propagados se representan junto a la flecha de la correspondiente dhTGD o pEGD. Observe que la dhTGD* ω_2 *es aplicable porque* $body^+(\omega_2)$ *es mapeado a* D_1 *y* $body^-(\omega_2)$ *no es mapeado a* D_1*; también advierta cómo el evento* $e_1(t_1, t_2)$ *es propagado a través de los átomos generados. Luego, como la pEGD* ϵ_2 *es aplicable, el valor nulo* z_1 *es instanciado y los eventos* $e_1(t_1, t_2)$ *y* e_2 *son propagados. Finalmente, por simplicidad,* M *se especifica directamente como una distribución de probabilidad conjunta, considerando todos los mundos posibles* λ_i *relativos a los eventos mencionados antes (cf. Figura 3.4).* ■

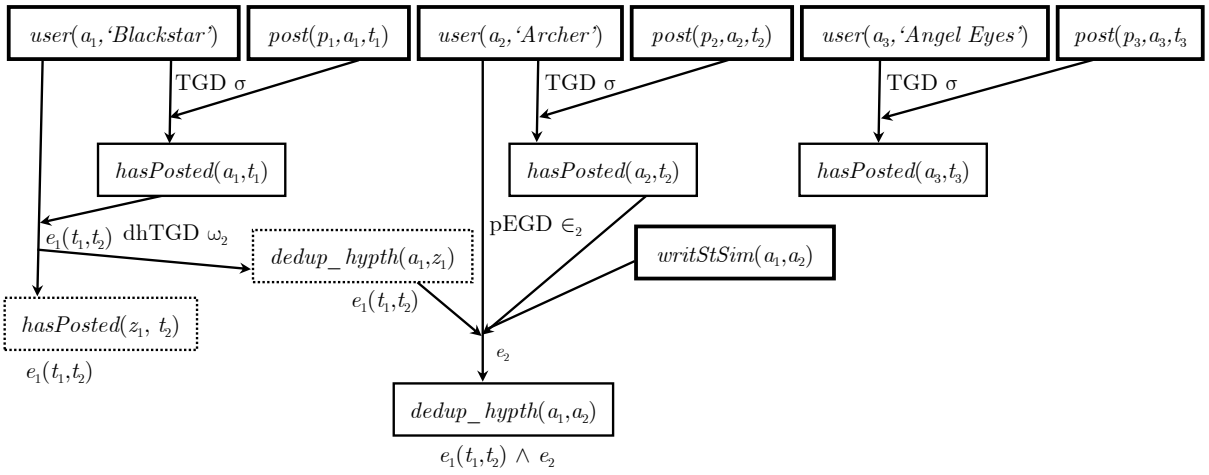

Figura 3.3: Representación gráfica del *chase* para el Ejemplo 3.4.

λ_i	$e_1(t_1, t_2)$	e_2	*Probabilidad*
λ_1	*true*	*true*	0,76
λ_2	*true*	*false*	0,19
λ_3	*false*	*true*	0,04
λ_4	*false*	*false*	0,01

Figura 3.4: Distribución de probabilidad conjunta sobre los eventos $e_1(t_1, t_2)$ y e_2 usados en el Ejemplo 3.4. En general, tales distribuciones pueden ser especificadas, en forma compacta, a través de modelos probabilisticos tales como una red bayesiana.

En el ejemplo anterior, se puede apreciar que la aplicación de la dhTGD ω_2 produce una hipótesis de deduplicación para el usuario a_2, pero la falta de información provoca que el candidato para dicho duplicado permanezca como valor nulo. Sin embargo, la subsecuente aplicación de la pEGD ϵ_2 permite concluir que el usuario a_2 (quien tiene un estilo de escritura similar a a_1) es una opción probable para completar la hipótesis (el evento $e_1(t_1, t_2) \wedge e_2$ tiene probabilidad 0,76 de acuerdo a la Figura 3.4). Tener en cuenta que una dhTGD podría ser aplicada múltiples veces, por lo que si hay otro posible candidato, átomos *dedup_hypth* nuevos podrían ser generados si son necesarios.

3.1.4. Encontrando duplicados: umbral en consultas de deduplicación

El problema de identificar entidades, aparentemente diferentes, con una base de conocimiento (o un conjunto de bases de conocimiento) para determinar que realmente se corresponden a la misma entidad del *mundo real*, puede ser abordado como un problema de respuesta a una consulta: dada una base de conocimiento, se desea tener una forma, que se apoye en principios bien definidos, para responder preguntas relativas a la ocurrencia de entidades duplicadas. En este sentido, el procedimiento *chase* presentado anteriormente puede ser utilizado para responder consultas de la forma:

$$Q(X) = \mathit{dedup}(\mathit{eid}, X, \theta)$$

donde *eid* es una entidad de interés específica y $\theta \in [0, 1]$ representa un umbral para el valor de probabilidad. Las respuestas a consultas de esta forma simplemente consisten en valores posibles para la variable X, para los cuales el procedimiento *chase* generó átomos $\mathit{dedup\_hypth}(\mathit{eid}, \mathit{val})$ con evento asociado e con probabilidad $Pr_M(e) \geq \theta$. La Figura 3.5 ilustra, de manera general, este proceso de respuesta a consultas. Se puede observar que, en este proceso se considera un conjunto de base de datos junto con un conjunto de dhTGDs. En la primera fase del procedimiento *chase*, un conjunto de átomos de *dedup_hypth* es generado y, en la segunda fase, un conjunto de pEGDs es usado para asignar valores concretos a valores nulos en tales átomos. A su vez, ambas fases involucran mantener un registro de los eventos probabilísticos, los cuales definen las condiciones bajo las que las inferencias se sostienen. La salida es un conjunto de átomos *dedup_hypth* con los valores de probabilidad, los que pueden ser usados para responder la consulta de entrada.

A continuación, el siguiente resultado declara que el procedimiento *chase* en dos fases descripto en la Subsección 3.1.2 puede ser detenido después de un número finito de pasos a fin de responder consultas de deduplicación con umbral, siempre que las TGDs clásicas subyacentes pertenezcan a un fragmento de *Datalog+/–* donde las consultas conjuntivas son decidibles; dicha condición puede ser garantizada por restricciones sintácticas tales como *guardedness* [CGL12], *stickiness* [CGP12], *aciclycity*, así como sus contrapartes [CGK13, CGP12, FKMP05], o por restricciones semánticas tales como *shyness* [LMTV12], *finite expansion sets* (fes), *bounded treewidth sets* (bts), y *finite unification sets* (fus) [BMRT11] (cf. [SMM$^+$17, Sección 1.3.3] para una breve descripción de muchas condiciones como las mencionadas).

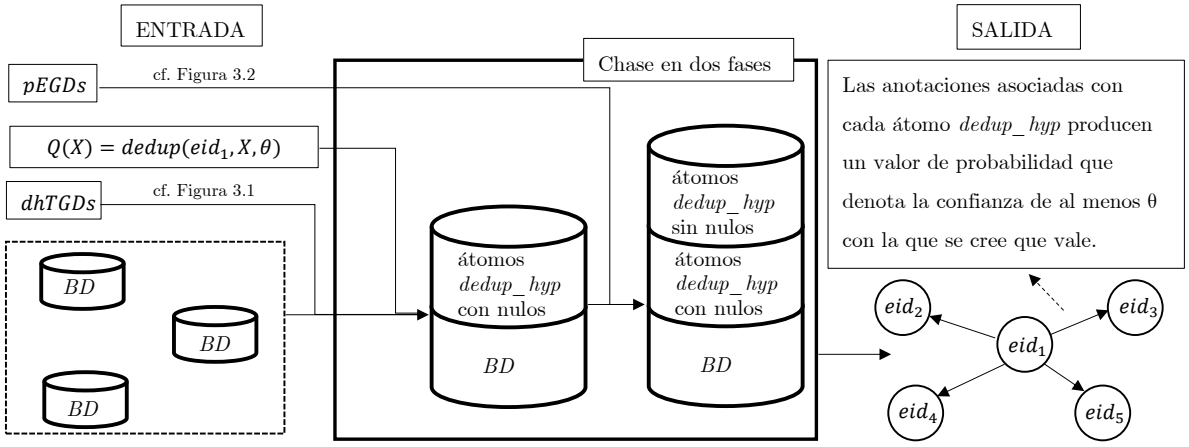

Figura 3.5: Descripción general de la aplicación *Datalog+/–* probabilístico para responder consultas de deduplicación.

Proposición 3.1 *Considere una ontología Datalog+/– probabilística con negación estratificada definida como* $KB = (O = (D, \Sigma), M, af)$ *y una consulta de deduplicación con umbral* $Q(X) = dedup(eid, X, \theta)$. *Si O pertenece a un fragmento de Datalog+/– para el cual una porción finita de chase*(D, Σ) *puede ser computada para responder consultas conjuntivas, entonces existe una porción finita del procedimiento chase probabilístico en dos fases que puede ser usada para responder la consulta de deduplicación con umbral Q sobre KB.*

Prueba 3.1 (Bosquejo) Esencialmente, este resultado se sostiene debido a que en la primer fase del *chase* se aplican pTGDs y dhTGDs (que son un caso especial de pTGDs) en el orden dado por la estratificación, la segunda fase es simplemente una aplicación de pEGDs sobre una estructura fija. Ambas fases involucran la propagación de eventos probabilísticos, los cuales son usados junto con el modelo probabilístico para verificar qué átomos superan el umbral proporcionado en la consulta.

3.2. Un enfoque basado en aprendizaje automatizado

En esta sección se lleva adelante la misma primera aproximación de sistema de generación automatizada de hipótesis para abordar el problema de deduplicación adversarial,

pero ahora haciendo énfasis en la utilización de técnicas de aprendizaje automatizado, específicamente clasificadores, basadas en análisis de contenido escrito y empleando reglas muy simples que determinen cuando una hipótesis es generada. Estas reglas solo consideran que el resultado obtenido de aplicar clasificadores superen un cierto umbral. Observe que este enfoque, en términos generales, no es algo nuevo y ha sido aplicado en diversos problemas, tales como identificación y atribución de *malware*, entre otros. Sin embargo, para este tipo de problemas sí puede considerarse novedoso. Esencialmente, se busca desarrollar un método *simple*, por el cual publicaciones escritas por usuarios puedan ser automáticamente analizadas, para luego generar *hipótesis de deduplicación*. Luego, analistas humanos podrían intervenir para llevar a cabo un análisis más profundo. Los pasos para el desarrollo de este enfoque pueden ser resumidos de la siguiente forma:

- Obtener información de discusiones *online* realizadas en foros y mercados en la web; este es un trabajo continuo que generalmente es llevado a cabo, de manera semiautomática, por especialistas [SGS16].
- Preparar los datos realizando diferentes procesos de limpieza.
- Entrenar uno o más clasificadores de aprendizaje automatizado para reconocer publicaciones escritas por cada usuario. Para este paso, se asume que las publicaciones hechas por diferentes nombres de usuarios corresponden a diferentes usuarios. Esta suposición también se vuelve importante cuando se analizan los resultados obtenidos en la fase de prueba. Tener en cuenta que se podría llevar a cabo, de manera alternativa, entrenando un único clasificador multi-clase que discrimine entre todos los usuarios. Se decidió entrenar un clasificador por usuario debido a dos razones principales:
  - Grandes cantidades de usuarios (clases en este caso) son más difíciles de administrar y el clasificador resultante sería menos flexible, ya que con un clasificador por usuario es posible incorporar *features* que sólo aplican a ciertos usuarios.
  - Tal vez lo más importante es que sería adecuado ser flexibles con respecto a la incorporación de nuevos usuarios, lo que provocaría que, con el enfoque de clasificador único, éste tuviera que re-entrenarse con cada incorporación.
- Aplicar los clasificadores a *pares* de publicaciones nuevas hechas por los usuarios X e Y; si el clasificador de X declara que una publicación escrita por Y fue escrita por X, o viceversa, entonces se genera una *hipótesis de deduplicación*.

Número de usuarios n	*Número de pares posibles* $\binom{n}{2}$
50	1.225
100	4.950
150	11.175
200	19.900
500	124.750
1.000	499.500
2.000	1.999.000
5.000	12.497.500
10.000	49.995.000

Figura 3.6: Número de pares no ordenados de usuarios que necesitan ser inspeccionados en un análisis basado en fuerza bruta.

- El conjunto de hipótesis de deduplicación se envían a analistas humanos para un procesamiento adicional.

Puede observarse que si no se dispone de información adicional sobre los autores de las publicaciones, dados n usuarios se tendrían que analizar $\binom{n}{2}$ pares para ver si realmente se corresponden al mismo usuario. Luego, para 50 usuarios, esta cantidad es 1225 y para 100 se tienen 4950, lo cual se vuelve inviable si se quiere analizar manualmente. Por lo tanto, un análisis basado en fuerza bruta con grandes cantidades de usuarios es imposible (cf. Figura 3.6). El objetivo general del método propuesto es reducir, en gran medida, el conjunto de pares que los analistas humanos deben observar.

Análisis de texto a través de n-gramas: Cuando se busca extraer elementos básicos de contenido escrito, una herramienta común es el uso de *n-gramas*, los cuales pueden definirse de diferentes formas. En este caso, se adopta la definición comúnmente usada de un n-grama, que es considerarlos como una secuencia de n caracteres. La ventaja de usar n-gramas, en lugar de analizar directamente el texto, es que errores tipográficos, variaciones ortográficas y otro tipos de diferencias pueden sobrellevarse debido a que estos textos, a pesar de ser sintácticamente diferentes, producen conjuntos de n-gramas que están estrechamente relacionados.

Ejemplo 3.5 *Considere la palabra "software" y algunos errores ortográficos/variaciones intencionales comunes usadas en comunidades* online*: sofware, softwarez y sophwarez. Si*

se considera los 2-gramas y 3-gramas para cada uno de estos términos, se llega a los siguientes conjuntos:

	2-gramas	*3-gramas*
software	*so, of, ft, tw, wa, ar, re*	*sof, oft, ftw, twa, war, are*
sofware	*so, of, fw, wa, ar, re*	*sof, ofw, fwa, war, are*
sofwarez	*so, of, ft, tw, wa, ar, re, ez*	*sof, oft, ftw, twa, war, are, rez*
sophwarez	*so, op, ph, hw, wa, ar, re, ez*	*sop, oph, phw, hwa, war, are, rez*

Se puede apreciar que, incluso las dos variaciones más diferentes (software y sophwarez) aún comparten varios n-gramas. El valor de n es un parámetro que puede ser configurado; se estima que n en un rango $[3, 7]$ *funciona bien en este tipo de análisis [NDG+16].* ■

Por lo tanto, la hipótesis de trabajo es que clasificadores de aprendizaje automatizado, apropiadamente entrenados, pueden detectar indicios no intencionales presentes en la forma de escribir de personas quienes están intentando ocultarse atrás de múltiples perfiles en foros y mercados *online*. A continuación, se presenta el diseño y los resultados de un conjunto de experimentos como parte de los pasos iniciales para abordar problemas como los mencionados.

3.2.1. Evaluación empírica

La evaluación del enfoque propuesto con datos reales (para tener una idea general del tipo de información considerada, ver Figura 3.7), la cual fue realizada en dos experimentos principales, adopta la siguiente configuración básica:

- *Dataset*: compuesto de la tabla de publicaciones, la cual contiene 89.766 tuplas y la tabla de usuarios, la cual contiene 128 usuarios.
- *Preparación y limpieza de datos*: se removieron las etiquetas *HTML* en las publicaciones usando la herramienta *BeautifulSoup*[1], se removieron *URLs*, espacios extras y cadenas de caracteres conteniendo una combinación de letras y números. También, se descartaron publicaciones que contenían menos de 140 caracteres (es decir,

[1]https://www.crummy.com/software/BeautifulSoup

postId	postContent	postedDate	scrapedDate	forumsId	recordedDate	userId	topicId
11e9d297a29899f6a69c3da05391acaa	[u", the only good way to work in computer security straight is to be a "black hat" then at some point (arrested a few to many times, you get a wife/family) you decide you can't do it any more. the other way is to go into sysadmin and at some point, once you have experience going into nothing but security. , u'', i see way to many people about trying to sell themselves as pen-testers and security consultants and know fuck all, don't be one of those people., u']	11/10/2008	1/22/2017	56	11/10/2008	352820	481581
da1a4e396af0e1b87ba0d3a81b1e6277	your only recourse is using a second 3.60 or updated unit, or ps3 to download the games, then transfer them to your vita or qcma. and that only works for games you purchased, it isn't a gateway to piracy.	6/3/2017	6/5/2017	134	6/3/2017	75065	821237
92ac4e1e3cdb886080c6af2a528673eb	and i don^t particularly agree with touting this method either. why use wm_vsh_menu to call wmm functions to prepare the cfw when psnpatch does a fine job...?it^s not explained here but i assume it^s to run jb folder games to go online. however there is a good reason for psnpatch to remove cobra hooks stopping many jb folder games from working. bypassing this feature is not a smart idea for most users. i don^t recommend to do it. cfw users should always stick to iso & the psnpatch method. as to the title given the fact that the only users who require jb folders are cheaters & modders i wouldn^t actually hold my breath when telling them they will "never" get banned if they follow these steps... assuming that these steps will prevent a ban is not the same as knowing they will....	3/19/2017	6/23/2017	93	3/19/2017	1807	946623

forumsId	boardsName
56	null
134	vitahacks
93	thread locked

topicId	topicName
481581	[u'white hat hacker carrer']
821237	bought psn games on 3.63question (self.vitahacks)
946623	thread locked

Figura 3.7: Fragmento de las tablas de publicaciones de *hacking*, foros y temas obtenidos del *dataset* en consideración para los experimentos que fueron realizados.

cualquier contenido más corto que el máximo largo de un mensaje *SMS* o tweet antes de la expansión) o que contengan cualquiera de las siguientes cadenas de caracteres “quote from:”, “quote:”, “wrote:”, “originally posted by”, “re:”, o “begin pgp message”. Esto resultó en 40.453 publicaciones “limpias” correspondientes a 54 usuarios.

- *Generación de features*: Se utilizó la conocida técnica *TF-IDF* (*term frequency–inverse document frequency*) para obtener vectores de *features* basados en n-gramas, los cuales consisten en la asignación de pesos a *features* en forma tal que dicho valor incrementa proporcionalmente en relación a su número de apariciones en el documento, pero además considerando una relación inversamente proporcional con sus apariciones en el corpus completo.

- *Clasificadores*: Diferentes enfoques estándar de aprendizaje automatizado han sido implementados a través de una conocida librería de Python[2]:

  - *Decision Trees*
  - *Logistic Regression*
  - *Multinomial Bayesian Networks*

[2]http://scikit-learn.org/stable

- *Random Forests*
- *Support Vector Machine*, tanto con *linear kernels* como con *radial basis function (rbf) kernels*

- *Hiperparámetros*: Se exploraron diferentes valores de dos principales hiperparámetros: *max_df* y *n-gram range*. El primero es una cota en la frecuencia con la que una *feature* ocurre en una publicación (esencialmente, a medida que la frecuencia aumenta, el contenido de información de una *feature* se reduce) y, en cambio, el último determina el largo de las sub-cadenas en las cuales el texto es dividido. Esto resultó en un conjunto de 52 clasificadores instanciados (cf. Figura 3.8).

  Para algunos clasificadores instanciados, también se aplicó una cota en el número de *features* tomadas en cuenta por el mismo (*max_features*). Este espacio fue explorado manualmente probando el efecto de diferentes configuraciones, en relación a tiempo de ejecución y desempeño (más *features* producen mejor desempeño, hasta un cierto punto). La selección final quedó conformada de la siguiente manera:

  - DT2 es DT1 con $max\_features = 2500$.
  - DT3 es DT1 con $max\_features = 2000$.
  - DT9 es DT8 con $max\_features = 3000$.
  - DT5 tiene $max\_features = 3000$.
  - DT6 tiene $max\_features = 2000$.
  - DT18 es DT17 con $max\_features = 5000$.
  - MNB6 es MNB5 con $max\_features = 3000$.

  Este conjunto de clasificadores instanciados fue generado por medio de una exploración manual de los hiperparámetros, buscando las combinaciones que tuvieran el mayor potencial para conseguir los valores más altos de precisión y *recall*.

Evaluación empírica fase 1: Evaluación amplia de diferentes clasificadores y configuración de hiperparámetros

El objetivo de este primer conjunto de experimentos fue encontrar los clasificadores instanciados de mejor desempeño entre los 52 mostrados en la Figura 3.8. Con este objetivo en mente, se realizaron tres ensayos de los siguientes pasos:

ID	*Tipo de clasificador*	*max_df*	*n-gramas*
SVC1	*SVM–linear kernel*	0,02	[3, 3]
SVC2	*SVM–linear kernel*	0,03	[3, 3]
SVC3	*SVM–linear kernel*	0,02	[4, 4]
SVC4	*SVM–linear kernel*	0,01	[4, 4]
SVC5	*SVM–linear kernel*	0,03	[5, 5]
SVC6	*SVM–linear kernel*	0,03	[3, 4]
SVC7	*SVM–linear kernel*	0,06	[3, 4]
SVC8	*SVM–rbf kernel*	0,01	[3, 3]
SVC9	*SVM–rbf kernel*	0,05	[3, 3]
SVC10	*SVM–rbf kernel*	0,03	[3, 3]
SVC11	*SVM–rbf kernel*	0,05	[4, 4]
SVC12	*SVM–rbf kernel*	0,08	[4, 4]
SVC13	*SVM–rbf kernel*	0,05	[5, 5]
SVC14	*SVM–rbf kernel*	0,03	[5, 5]
DT1	*Decision Tree*	0,05	[3, 3]
DT2	*Decision Tree*	0,05	[3, 3]
DT3	*Decision Tree*	0,05	[3, 3]
DT4	*Decision Tree*	0,02	[3, 3]
DT5	*Decision Tree*	0,03	[3, 3]
DT6	*Decision Tree*	0,03	[3, 3]
DT7	*Decision Tree*	0,009	[3, 3]
DT8	*Decision Tree*	0,03	[4, 4]
DT9	*Decision Tree*	0,03	[4, 4]
DT11	*Decision Tree*	0,02	[4, 4]
DT12	*Decision Tree*	0,05	[4, 4]
DT13	*Decision Tree*	0,01	[4, 4]
DT14	*Decision Tree*	0,05	[5, 5]
DT15	*Decision Tree*	0,03	[5, 5]
DT16	*Decision Tree*	0,02	[5, 5]
DT17	*Decision Tree*	0,01	[5, 5]
DT18	*Decision Tree*	0,01	[5, 5]
MNB1	*Multinomial Bayesian Network*	0,02	[3, 3]
MNB2	*Multinomial Bayesian Network*	0,01	[3, 3]
MNB3	*Multinomial Bayesian Network*	0,008	[3, 3]
MNB4	*Multinomial Bayesian Network*	0,007	[3, 3]
MNB5	*Multinomial Bayesian Network*	0,005	[3, 3]
MNB6	*Multinomial Bayesian Network*	0,005	[3, 3]
MNB7	*Multinomial Bayesian Network*	0,005	[4, 4]
MNB8	*Multinomial Bayesian Network*	0,005	[4, 4]
MNB9	*Multinomial Bayesian Network*	0,003	[5, 5]
LR1	*Logistic Regression*	0,03	[3, 3]
LR2	*Logistic Regression*	0,02	[3, 3]
LR3	*Logistic Regression*	0,03	[4, 4]
LR4	*Logistic Regression*	0,01	[4, 4]
LR5	*Logistic Regression*	0,03	[5, 5]
LR6	*Logistic Regression*	0,01	[5, 5]
RF1	*Random Forest*	0,03	[3, 3]
RF2	*Random Forest*	0,02	[3, 3]
RF3	*Random Forest*	0,03	[4, 4]
RF4	*Random Forest*	0,02	[4, 4]
RF5	*Random Forest*	0,03	[5, 5]
RF6	*Random Forest*	0,02	[5, 5]

Figura 3.8: Descripción completa de la configuración de los hiperparámetros para todos los clasificadores instanciados.

- Elegir 10 usuarios aleatorios.
- *Fase de entrenamiento*: Para cada usuario u_i, entrenar un clasificador C_i usando entre 200 y 500 publicaciones escritas por ellos (ejemplos positivos, donde el número real dependía de la disponibilidad de las publicaciones) y el mismo número de publicaciones escritas bajo otros nombres (ejemplos *presumiblemente* negativos).
- *Fase de prueba*: Para cada usuario, se toman 20 publicaciones de prueba, que no fueron utilizados para entrenar ninguno de los clasificadores, y se consulta a los clasificadores de cada uno de los 10 usuarios. Esto produce una *matriz de confusión* M donde cada columna corresponde a un usuario y cada fila a un clasificador. $M(i, j)$ contiene el número de publicaciones predichas por el clasificador C_i como correspondientes al usuario u_j. Observe que una matriz de confusión perfecta, en este caso, debería tener un valor de 20 a lo largo de toda la diagonal y valores cero en todas las otras celdas. La Figura 3.9 muestra una instancia real de dicha matriz donde las columnas corresponden a usuarios y las filas a clasificadores. Cada celda $M(i, j)$ contiene el número de publicaciones (a lo sumo 20) para las cuales el clasificador C_i respondió *si* para una publicación cuya autoría corresponde al usuario u_j.
- Con el objetivo de evaluar el desempeño de cada clasificador, se hace uso de las siguientes nociones:
  - *Verdaderos positivos (vp)*: Sucede cuando una publicación de prueba escrita por un usuario u_i es clasificada correctamente por el clasificador C_i; el número de verdaderos positivos para C_i es encontrado en $M(i, i)$.
  - *Falsos positivos (fp)*: Sucede cuando una publicación de prueba escrita por un usuario u_i es clasificada incorrectamente por un clasificador $C_j \neq C_i$; el número de falsos positivos para C_i puede ser encontrado por calcular $\sum_{i \neq j \wedge 0 \leq j \leq 9} M(i, j)$.
  - *Verdaderos negativos (vn)*: Sucede cuando una publicación de prueba escrita por un usuario u_i es clasificada correctamente por un clasificador $C_j \neq C_i$; el número de verdaderos negativos para cada clasificador es simplemente $(10 - 1) * 20 - \sum_{i \neq j \wedge 0 \leq j \leq 9} M(j, i) = 180 - \sum_{i \neq j \wedge 0 \leq j \leq 9} M(j, i)$.

	352792	**20307**	**117723**	**43315**	**161133**	**282143**	**353596**	**352809**	**13585**	**146319**
C_{352792}	**19**	0	0	1	1	0	0	0	0	1
C_{20307}	7	**17**	6	5	9	9	2	3	2	9
C_{117723}	5	5	**19**	6	4	3	4	6	1	9
C_{43315}	9	2	1	**17**	2	0	2	6	1	8
C_{161133}	5	4	4	7	**17**	2	5	0	0	13
C_{282143}	4	9	9	8	2	**14**	4	4	0	9
C_{353596}	12	7	11	16	8	3	**17**	8	6	14
C_{352809}	3	6	4	8	4	3	8	**10**	0	9
C_{13585}	1	2	9	4	4	7	9	2	**17**	9
C_{146319}	12	5	7	10	10	4	9	5	1	**17**

Figura 3.9: Ejemplo de una matriz de confusión para el experimento 1.

- *Falsos negativos (fn)*: Sucede cuando una publicación de prueba escrita por un usuario u_i es clasificada incorrectamente por un clasificador C_i; el número de falsos negativos para C_i es calculado como $20 - M(i, i)$.

Tomando en cuenta la matriz de confusión M y los cálculos hechos arriba, es posible derivar:

$$\textit{precisión}(C_i) = \frac{vp(C_i)}{vp(C_i) + fp(C_i)}$$

y

$$\textit{recall}(C_i) = \frac{vp(C_i)}{vp(C_i) + fn(C_i)}.$$

Los valores obtenidos son métricas estándar usadas para evaluar desempeño de clasificadores. Asimismo, la media armónica de estos dos valores, conocida como la medida F1, con frecuencia es usada como una buena forma de comparar el desempeño de un conjunto de clasificadores. Finalmente, se calculó la media de las tres ejecuciones para obtener los resultados finales, los cuales se reportan a continuación.

Resultados de la fase 1

Los resultados de este conjunto de experimentos son mostrados en la Figura 3.10 (desempeño de clasificación) y la Figura 3.11 (tiempo de ejecución); cada gráfico de la primera muestra los valores obtenidos para precisión, *recall* y la medida F1. La clásica relación costo/beneficio entre precisión y *recall* puede ser observada en cada gráfico, aunque todas las instancias tuvieron valores altos para *recall*, los clasificadores que lo hicieron mejor

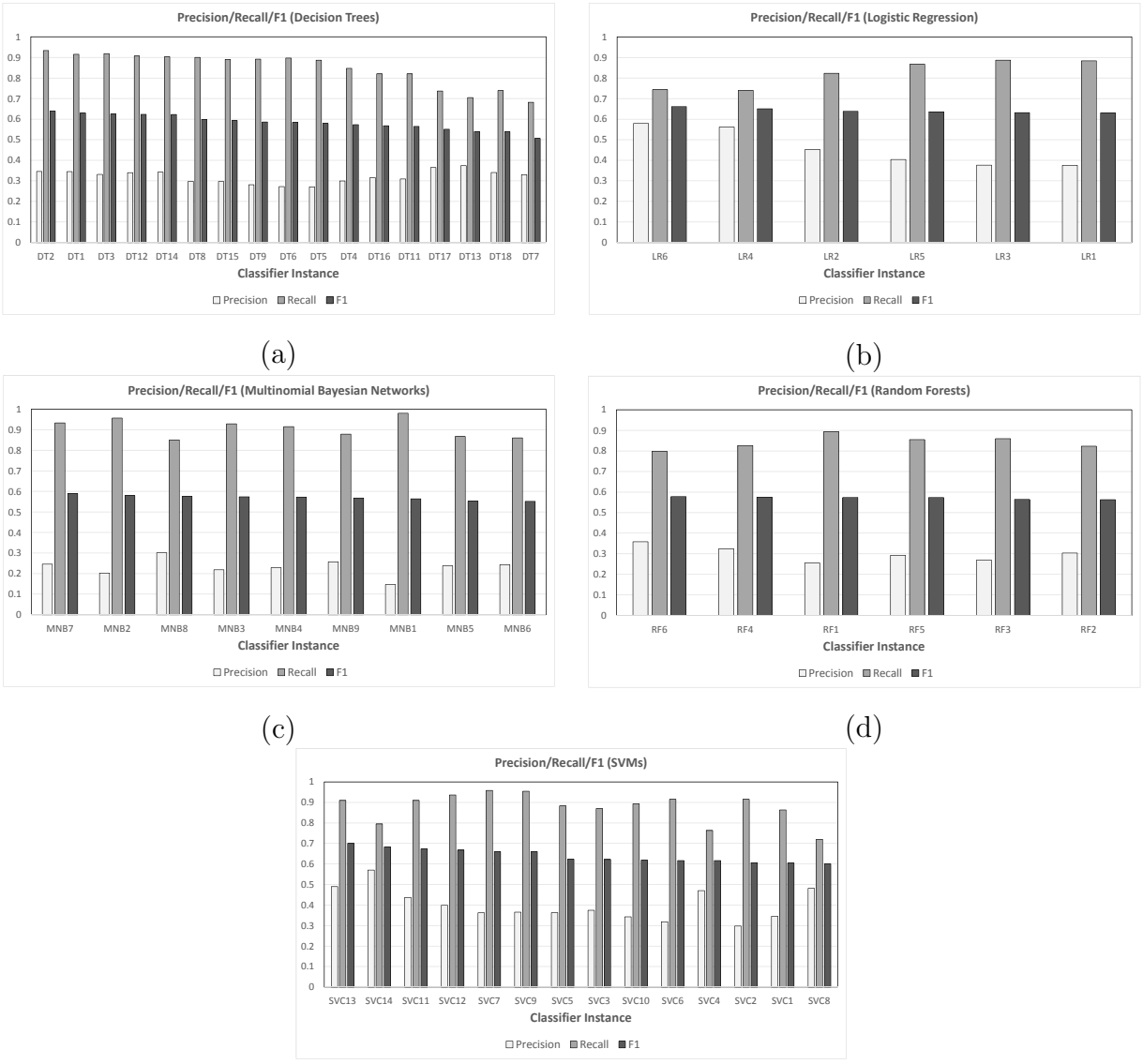

Figura 3.10: Valores de precisión, *recall* y F1 para todos los clasificadores evaluados en la fase 1: (**a**) *Decision Trees*, (**b**) *Logistic Regression*, (**c**) *Multinomial Bayesian Networks*, (**d**) *Random Forests* y (**e**) *Support Vector Machines*. En cada gráfico, los clasificadores instanciados son ordenados de forma descendiente en relación a la medida F1.

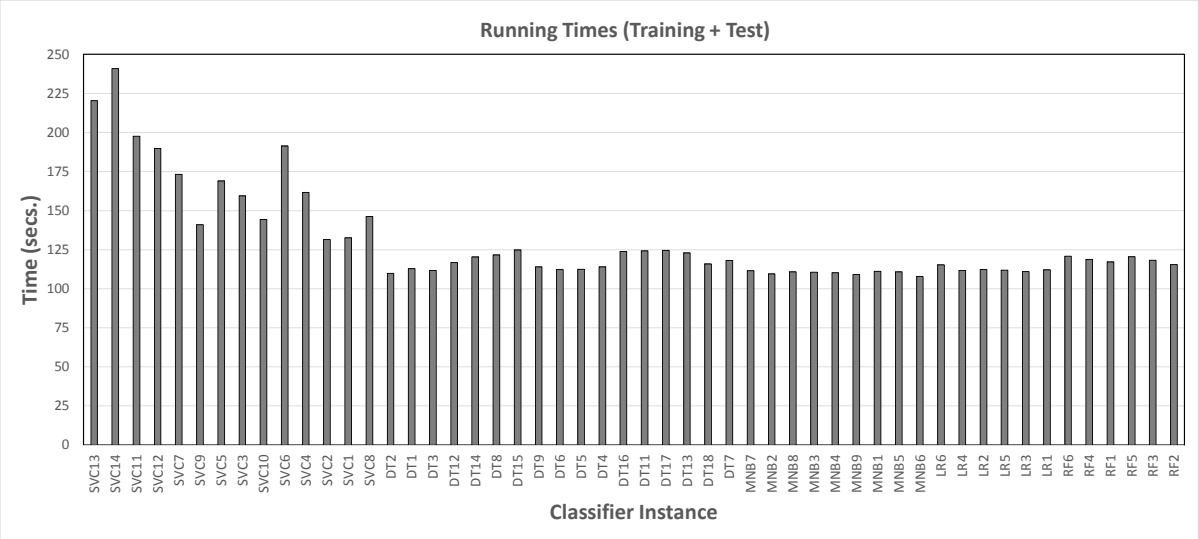

Figura 3.11: Tiempo de ejecución requerido para el entrenamiento y evaluación de cada clasificador instanciado en la fase 1, ordenados por la medida F1 como fue hecho en la Figura 3.10.

con respecto a esta medida pagaron un alto costo en precisión (cf. MNB2, el cual presume un *recall* de 0,96 pero solo 0,2 en precisión). La Figura 3.12 agrupa los dos de mejor desempeño para cada tipo de clasificador.

Es importante destacar que, dado que se utilizaron 10 usuarios en esta fase, si se intentara resolver el problema utilizando una respuesta aleatoria, entonces la probabilidad de tener éxito seria 0,1. Por lo tanto, los clasificadores con la mejor precisión en la Figura 3.12 (SVC13, SVC14, LR6, y LR4) tuvieron de 4,9 a 5,8 veces mejor desempeño que este *baseline*.

Evaluación empírica Fase 2: Diseminando duplicados conocidos

En esta segunda parte, se seleccionaron diversos clasificadores, dependiendo de su desempeño en la fase 1, y se evaluó su capacidad para encontrar duplicados. Para esto, se consideraron los de mejor desempeño para conformar clasificadores "puros", y también se construyeron diferentes *instancias agregadas* aplicando el método de *bootstrap aggregation*, en el cual un conjunto de clasificadores es usado en forma conjunta y una votación por mayoría produce el resultado final. La elección fue hecha entre aquellos mostrados en

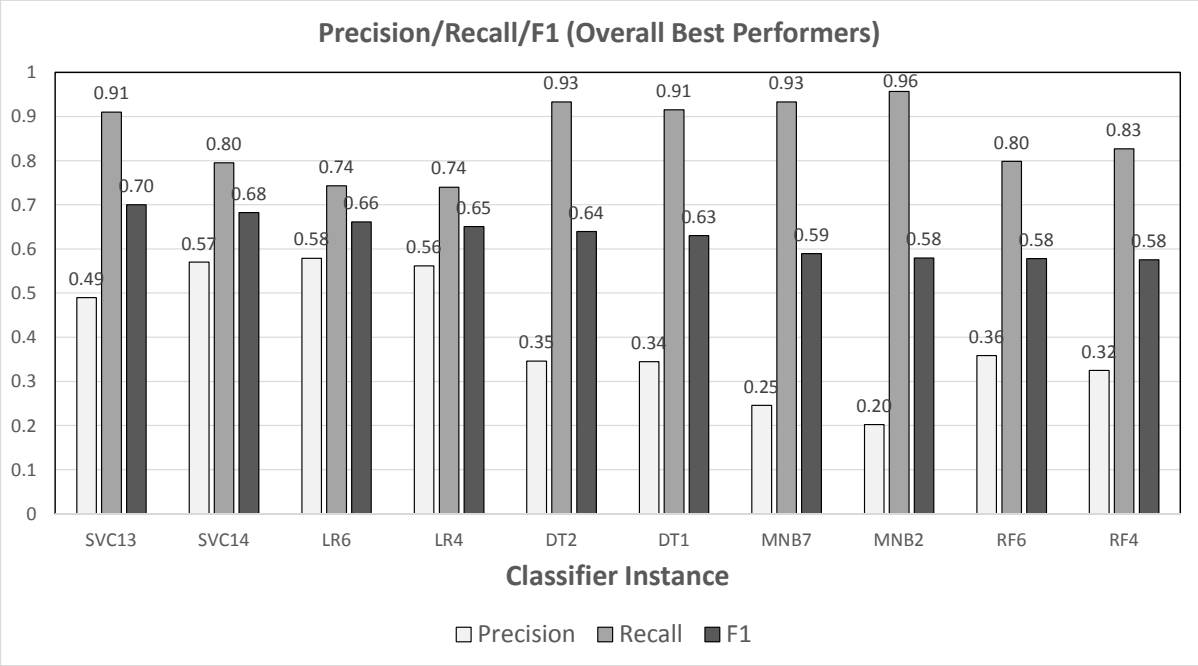

Figura 3.12: Mejores dos desempeños de clasificadores por cada gráfico de la Figura 3.10, ordenados de acuerdo a la medida F1.

la Figura 3.12, principalmente por su desempeño con respecto a la medida F1, lo cual resultaría en SVC13 y SVC14. Sin embargo, aún cuando SVC14 tuvo un desempeño ligeramente mejor que LR6, se eligió este último para conformar el segundo clasificador "puro" debido a que se buscó priorizar la diversidad. Como consecuencia, la lista de clasificadores fue la siguiente:

- SVC13
- LR6
- E1 = {SVC13, SVC14, LR6, LR4}
- E2 = {SVC13, SVC14, LR6}
- E3 = {SVC13, SVC14, LR4}
- E4 = {SVC13, LR4, LR6}
- E5 = {SVC14, LR4, LR6}

Uno de las desafíos importantes para llevar a cabo los experimentos en este dominio es lo difícil (o imposible) que resulta obtener datos con *ground truth*, por lo que se tuvo que crear artificialmente pares de duplicados por medio de la división de las publicaciones de k usuarios en dos conjuntos, y luego entrenar clasificadores independientes con esos datos (haciendo que los k usuarios originales se conviertan en $2k$ usuarios). Ahora bien, la definición de verdadero/falso positivo y verdadero/falso negativo depende del tipo de par que está siendo considerado:

- Para un par de usuarios que es conocido (o, más bien, asumido) que son diferentes:
  - *Verdadero negativo*: el clasificador dice *no*.
  - *Falso positivo*: el clasificador dice *si*.

  Verdaderos positivos y falsos negativos son imposibles en este caso.
- Para un par de usuarios que es conocido que son duplicados:
  - *Verdadero positivo*: el clasificador dice *si*.
  - *Falso negativo*: el clasificador dice *no*.

  Verdaderos negativos y falsos positivos son imposibles en este caso.

Considere la tarea de elegir p pares de usuarios (u_i, u_j) en forma aleatoria y presentar 20 publicaciones de prueba de cada usuario al correspondiente clasificador del usuario opuesto (esto es, publicaciones del usuario u_i al clasificador C_j, y viceversa). Para $k = 2$, dado que se tiene $54 + 2 = 56$ usuarios (los originales más los duplicados), se tiene

$$\binom{54 + 2 = 56}{2} = 1540$$

pares posibles. Entonces, con la intención de simular un porcentaje razonable de duplicados en el número de pares de prueba, se realizaron ejecuciones variando el valor de $p \in \{20, 40, 60, 80\}$ y se calcularon los valores resultantes de precisión, *recall* y F1, tomando la media aritmética sobre dos ejecuciones para cada uno. Estos resultados son presentados en la Figura 3.13.

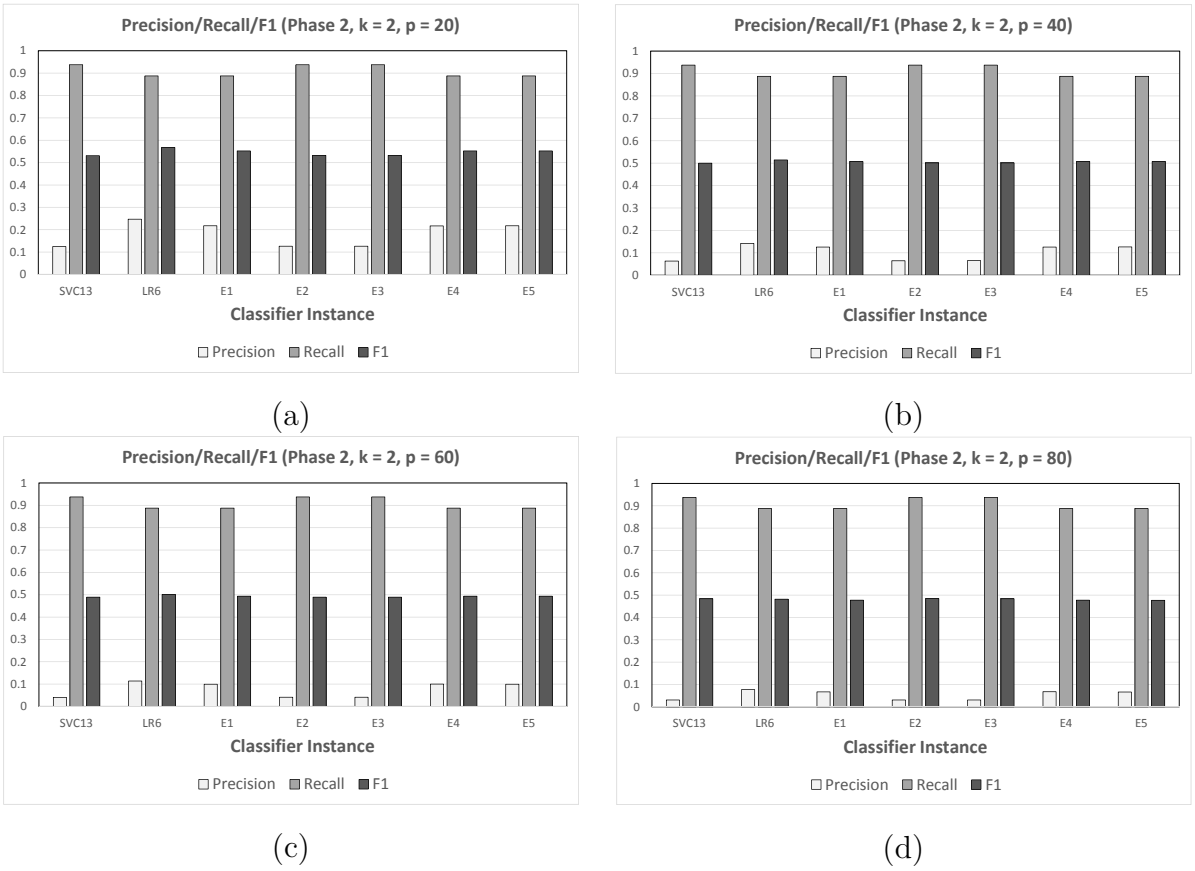

Figura 3.13: Evaluación de desempeño de los clasificadores elegidos (dos puros, 5 agregados) sobre 54 usuarios más dos duplicados creados en forma artificial.

Resultados de la fase 2

La primera cuestión que puede observarse cuando se analizan los resultados es que el desempeño de todos los clasificadores con respecto a la precisión fue mucho más bajo que los obtenidos en la fase 1. Esto puede ser explicado por el fuerte incremento en el número de usuarios con respecto a los experimentos previos (de 10 a 56), lo cual otorga muchas más oportunidades para que los clasificadores produzcan falsos positivos. Como se mencionó anteriormente, si se utilizara un clasificador que funcione respondiendo *si* de manera aleatoria 1 de n veces (para n usuarios) sólo tendría una precisión de $1/56 \approx 0{,}0178$. Por lo tanto, los valores de 0,2468 (LR6) y 0,2176 (E1), los dos mejores con $p = 20$, fueron alrededor de 13,84 y 12,22 veces mayor, respectivamente. Adicionalmente, la precisión se vuelve más baja a medida que p incrementa, alcanzando 0,0769 (LR6) y 0,0673 (E1) para $p = 80$ (aun alrededor de 4,32 y 3,78 veces mejor que la elección

aleatoria, respectivamente). Por otro lado, el *recall* se mantuvo relativamente alto para todos los clasificadores, independientemente del valor de p, y fue considerablemente mejor para SVC13 (y todos los clasificadores agregados por SVC) comparado con LR6, el cual a su vez tuvo una precisión más alta. Finalmente, considere los últimos dos pasos del enfoque propuesto al principio de esta sección, los cuales involucran la generación de *hipótesis de deduplicación*. Recuerde que lo que se define como falsos positivos en estos experimentos, en realidad pueden ser manifestaciones de duplicados (pares de nombres de usuarios que realmente corresponden a la misma persona) desconocidos. Una manera de tratar con la incidencia relativamente alta de falsos positivos que se observó cuando se computó la precisión, es configurar un valor t como umbral representando el número de veces que las publicaciones presentadas a un clasificador diferente derivaron en una respuesta positiva antes de emitir una hipótesis de deduplicación. Tomando la misma configuración de este experimento, se computó el número de hipótesis generadas por cada clasificador (considerando que el número total de pares no ordenados era 1540, el porcentaje de este total es indicado en cada caso). De esta manera, por ejemplo, en el caso de SVC13, 94,28 % de los pares fue identificado como potenciales duplicados, con un umbral $t = 10$:

- Para $t = 10$:
  - SVC13: 1452 (94,28 %)
  - LR6: 451 (29,28 %)
  - E1: 605 (39,28 %)
  - E2: 1444 (93,76 %)
  - E3: 1445 (93,83 %)
  - E4: 609 (39,54 %)
  - E5: 599 (38,89 %)
- Para $t = 15$:
  - SVC13: 1425 (92,53 %)
  - LR6: 140 (9,09 %)
  - E1: 204 (13,24 %)
  - E2: 1396 (90,64 %)

- E3: 1396 (90,64 %)
- E4: 203 (13,18 %)
- E5: 207 (13,44 %)

Dos conclusiones pueden ser esbozadas a partir de estos resultados. Primero, al observar las medidas de calidad finales, la precisión más baja producida por SVC13 en la fase 1 (así como en la fase 2 para los clasificadores agregados dominados por clasificadores basados en SVC) tenía, en realidad, causas de mayor alcance de lo que se podía sospechar, debido a que la tasa de falsos positivos fue claramente bastante mayor (como lo demuestra el pequeño cambio en el número de hipótesis al aumentar el valor de t). Segundo, aún cuando el desempeño de LR6 también fue bajo en la fase 2, se pudo reducir el número de pares a inspeccionar en alrededor del 70 % para $t = 10$ (es decir, al menos la mitad de las publicaciones de prueba) y en alrededor del 90 % para $t = 15$ (es decir, al menos tres cuartos de las publicaciones de prueba). Este fenómeno también fue observable para los clasificadores agregados en los cuales clasificadores basados en *logistic regression* tenían un rol importante (E1, E4 y E5).

Ejemplo 3.6 *Las siguientes publicaciones son algunos ejemplos de un par de usuarios que, según LR6, podrían haber sido escritos por la misma persona:*

- *Publicaciones del usuario* 353596*: "lol i have that pasted to the front door of my officebut if we really want to get technical here...these are all "cheap" translations of binary. i will type up a full explanation from home tonight. i also dug up and old piece of software that i have that is wonderful for quick lookups. i will post that as well."*

  "well "%path%" is the variable name. "path" is a term. it is always more important to learns terms and concepts rather than syntax that is specific to an environment.but yeah"

  "didnˆt mean to burst your bubble about the script. i also saw some of the samples but none of them conviced me that the user thought they were really talking to a human being. most were prolly playing along like i do there are some msn ones also... anyone has visited they didnˆt sign up for the forums. but thatˆs ok they can lurk for a while :whatsup:"

- *Publicaciones del usuario* 352820*: "there are a few newish tools for but the issues with bt in the first place was poor implementation on behalf of the manufacturers, most of which were fixed."*

  "well if talking about power friendly then a hd in lan enclosure would be best. you could build something that would consume less power but it would be more expensive, unless you are looking for and upwards."

  "the only good way to work in computer security straight is to be a "black hat" then at some point (arrested a few to many times, you get a wife/family) you decide you can't do it any more. the other way is to go into sysadmin and at some point, once you have experience going into nothing but security. , u", i see way to many people about trying to sell themselves as pen-testers and security consultants and know fuck all, don't be one of those people."

Habiendo identificado a un par de usuarios como una hipótesis de deduplicación, los analistas expertos humanos pueden observar más de cerca sus publicaciones, actividades y otros datos para confirmarlos o rechazarlos.

3.3. Sumario

En este capítulo, se presentó una primera aproximación para un sistema automatizado de generación de hipótesis que permita atacar problemas de comportamiento malicioso en plataformas sociales. Considerando la complejidad de los problemas, el dominio fue acotado a un problema específico conocido como deduplicación adversarial y se llevaron adelante dos enfoques diferentes para el desarrollo de dicho sistema. En el primero de ellos, se puso mayor énfasis en la utilización de reglas lógicas y en el segundo, en la utilización de técnicas basadas en aprendizaje automatizado. En el siguiente capítulo, se presenta una arquitectura de un sistema que permita un abordaje más general de los problemas de comportamiento malicioso en plataformas sociales.

Capítulo 4

La arquitectura NetDER: Un modelo para razonar sobre comportamiento malicioso

En este capítulo, se presenta una arquitectura con la capacidad de razonar sobre problemas de comportamiento malicioso en plataformas sociales. Dicha arquitectura es desarrollada desde el punto de vista de diseño de sistemas con la intención de que sirva de guía para la implementación de software en el dominio correspondiente.

4.1. Diseño de la arquitectura NetDER

A continuación, se presenta una arquitectura general orientada a la generación de hipótesis en base a los datos disponibles, conocimiento ontológico y modelos de difusión en redes multicapa. La idea principal detrás de combinar ontologías y procesos de difusión es potenciar la sinergia que surge de razonamientos complejos y explicables (como lo que proporciona el primero) y la representación y razonamiento con procesos de difusión complejos en redes sociales (como lo que proporciona el último).

Configuración inicial. Se asume la disponibilidad de un conjunto de *entrada de datos* con las siguientes propiedades:

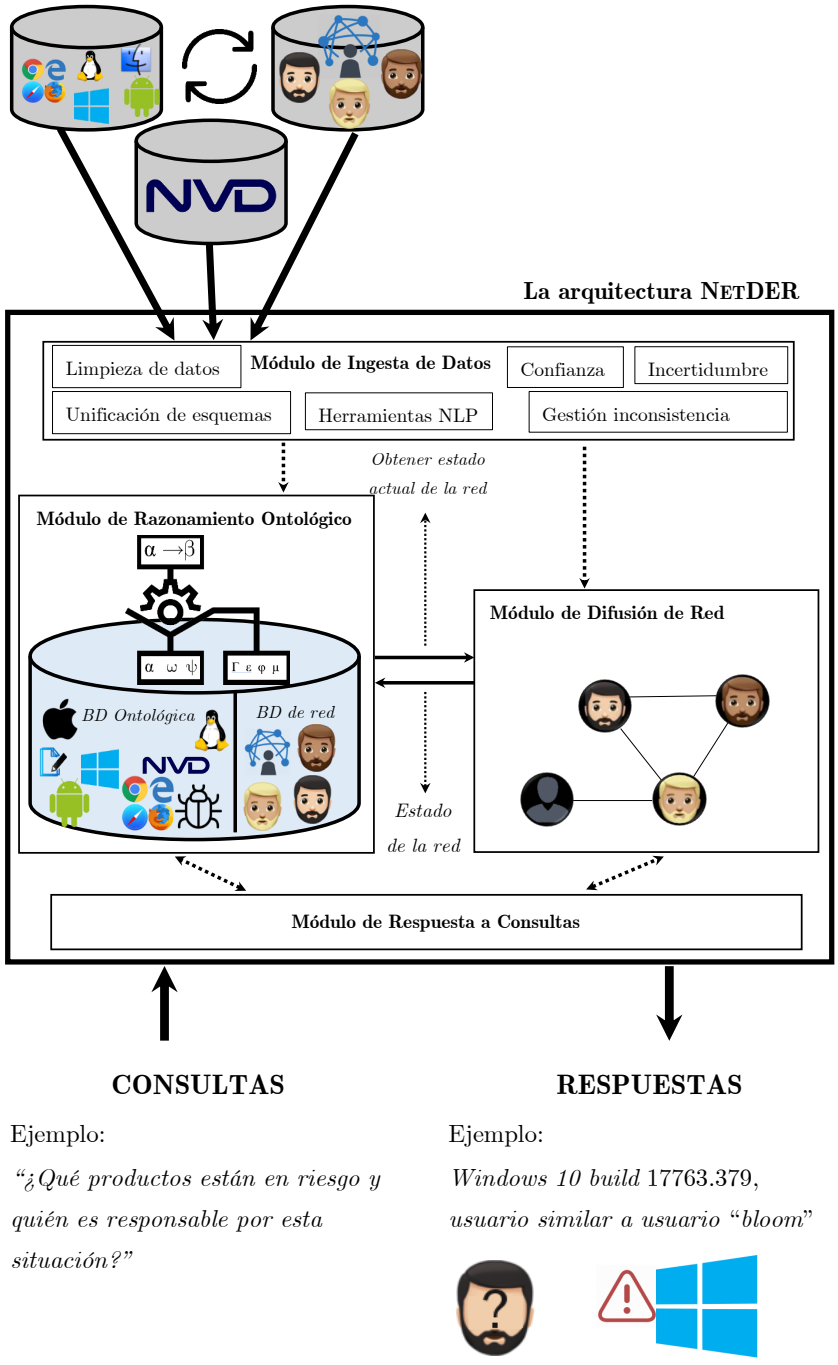

Figura 4.1: Descripción general de la arquitectura NetDER.

- *Dinamismo*: Las entradas pueden ser vistas como *flujos de información* que son constantemente modificados y actualizados; nueva información puede, por lo tanto, cambiar el resultado de razonamientos llevados a cabo en el pasado.

- *Heterogeneidad*: La información proviene de diferentes fuentes y puede ser estructurada en diferentes formatos.

- *Incertidumbre*: Todos los entornos, excepto los más simples, producen información intrínsecamente incierta. Ejemplos claros de estos entornos son el clima, el mercado de valores, el comportamiento humano, etc.

- *Incompletitud*: La información disponible puede ser parcialmente completa; sin embargo, debido al dinamismo nueva información puede estar disponible en el futuro, y en ese caso se obtiene un panorama más completo. Observe que la incompletitud también puede ser vista como un caso especial de incertidumbre.

Esta arquitectura se encuentra dividida en cuatro módulos principales relacionados: el *Módulo de Ingesta de Datos* (DIM), el *Módulo de Razonamiento Ontológico* (ORM), el *Módulo de Difusión de Red* (NDM) y el *Módulo de Respuesta a Consultas* (QAM). El DIM se encarga de tareas de preparación de los datos, el ORM está encargado de llevar a cabo tareas de razonamiento utilizando reglas basadas en lógica, el NDM mantiene modelos de redes complejas relevantes y los diferentes procesos de difusión que ocurren en ellos y por último, el QAM se encarga del proceso para responder a las diferentes consultas. En esta propuesta, se realizan las siguientes suposiciones adicionales:

- El *Módulo de Difusión de Red* y el *Módulo de Razonamiento Ontológico* son implementados en dos procesos asincrónicos y concurrentes.

- Los datos de entrada son integrados por el *Módulo Ingesta de Datos* de la arquitectura, el cual devuelve una única base de datos con la información ontológica y de red básica necesaria para el funcionamiento del Módulo de Razonamiento Ontológico y el Módulo de Difusión de Red.

- El Módulo de Razonamiento Ontológico tiene acceso al Módulo de Difusión de Red a fin de verificar la satisfacción de condiciones y también poder hacer modificaciones en la información relativa al modelo subyacente. Sin embargo, el Módulo de Difusión de Red no puede afectar de una manera directa al Módulo de Razonamiento Ontológico.

La Figura 4.1 muestra la arquitectura NetDER instanciada para el dominio de ciberseguridad, focalizando en la tarea de responder consultas, la cual es la principal funcionalidad provista. Las entradas de datos se asumen que son constantemente actualizadas y una componente de Ingesta de Datos está a cargo de producir una única base de datos (destinada al Módulo de Razonamiento Ontológico) y la red compleja (destinada al Módulo Difusión de Red); las responsabilidades de este módulo incluyen unificación de esquemas (*schema matching*), limpieza de datos (*data cleaning*), gestión de confianza (*managing trust*) sobre las fuentes de datos, entre otras. Flechas punteadas indican la capacidad de que el módulo originante pueda hacer cambios a los datos en el módulo destinatario. En este caso, las entradas de datos vienen de diferentes fuentes tales como *base de datos de productos de software* conteniendo información sobre, por ejemplo, Windows 10, sistemas operativos basados en Linux, MacOS, Android, navegadores tales como Google Chrome, Safari, Microsoft Explorer y Mozilla Firefox, *vulnerabilidades de software* tales como la información disponible en la *National Vulnerability Database* [1] y la *información de usuarios y sus interacciones en redes sociales* (Twitter, Darknet, Deepnet y otros). Entonces, se podría estar interesado en consultas referidas a productos de software que están en riesgo de ser atacados (cf. [NSS18] para un trabajo en esta dirección) y quiénes pueden ser responsables por tales ataques, como muestra la figura. Se debe tener en cuenta que las respuestas pueden ser tan detalladas como "*Windows 10 build 17763.379 está en riesgo*" o más general, tales como "*un usuario similar al usuario bloom es responsable de poner en riesgo a Windows 10*".

Modelo de Datos General. Como se discutirá más adelante, la base de conocimiento ontológica es creada (y continuamente actualizada) por el Módulo de Ingesta de Datos; entonces el modelo de datos consiste de dos partes:

- *Conocimiento de red*: Datos y conocimiento referido a un conjunto de usuarios de interés, tales como sus perfiles en diferentes plataformas, actividades, relaciones con otros perfiles y otros aspectos relevantes. El Módulo de Difusión de Red trabajará en este subconjunto con el objetivo de ejecutar el proceso de difusión y realizar la verificación de condiciones en cuanto sea requerido por el módulo de razonamiento ontológico.

- *Conocimiento de contexto*: Mantiene todos los datos y conocimiento relacionado al dominio. Considerando el ejemplo descripto en la Figura 4.1, ejemplos de conoci-

[1] https://nvd.nist.gov/

Algoritmo: Respuesta a consultas NetDER

Entrada: Una consulta Q con un vector de variables $\mathbf{X}$ (posiblemente vacío); una política para responder consultas QAP.

Salida: Un conjunto de respuestas para Q, cada una compuesta de un conjunto de pares de la forma (X, v), indicando que v es un valor asignado a una variable X en $\mathbf{X}$.

1. $KB \leftarrow$ Determinar una vista estable de los datos mantenidos dentro del ORM; **[DIM]**
2. $Net \leftarrow$ Construir la estructura de datos de la red compleja a partir del conjunto *NetDB*, la cual es un subconjunto mantenido dentro de *KB*; **[DIM]**
3. Preparar el procedimiento para responder consultas; **[ORM]**
4. Repetir
   - *a*) Ejecutar el procedimiento de difusión sobre *Net*; **[NDM]**
   - *b*) Evaluar la consulta Q sobre KB; **[ORM]**

   Hasta que la condición establecida por QAP sea satisfecha.
5. *Retornar*: Respuestas a Q derivadas a partir de la última ejecución del Paso 4*b*.

Figura 4.2: Descripción general del algoritmo para responder consultas.

miento de contexto incluyen datos sobre *software*, *malware* y vulnerabilidades. El Módulo de Razonamiento Ontológico usará este subconjunto como conocimiento inicial en el proceso de respuesta a consultas (cf. siguiente párrafo y Figura 4.2) e interactuará con el Módulo de Difusión de Red cuando sea necesario.

Respuesta de consultas. El algoritmo general que se encarga de dirigir el mecanismo para responder consultas es descripto en la Figura 4.2 y es llevado a cabo por el Módulo de Respuesta a Consultas; cada paso es etiquetado con "[DIM]" (para el Módulo Ingesta de Datos), "[ORM]" (para el Módulo de Razonamiento Ontológico) o "[NDM]" (para el Módulo de Difusión de Red), indicando qué componente es responsable de ejecutar la tarea. La *política para responder consultas*, que se brinda como parte de la entrada, determinará el número de ciclos ORM–NDM que son llevados a cabo antes de proporcionar la respuesta o respuestas obtenidas. Un aspecto interesante para señalar es que las operaciones realizadas por el Módulo de Ingesta de Datos son asincrónicas con respecto a tareas de responder consultas; ésta es la razón detrás de la necesidad de ejecutar el Paso

1: aunque los datos pueden continuar llegando y ser procesados, una vista estable debe ser establecida con el propósito de responder consultas. Claramente, dependiendo de la tasa de actualización definida por el sistema, esto significa que plantear iterativamente una misma consulta puede producir resultados diferentes. En la Sección 4.2 se proveen detalles adicionales de cómo las estructuras de la red y de la ontología se derivan a partir de los datos procesados por el Módulo de Ingesta de Datos. Finalmente, un aspecto importante a destacar es que el ciclo de la línea 4 del algoritmo no siempre está garantizado que termine; más adelante se discutirá la necesidad de alcanzar tal condición de terminación, lo cual puede lograrse de una manera tan simple como parar después de un cierto número de pasos o de una forma más compleja como a través de la definición de condiciones en la sintaxis de las reglas ontológicas y la limitación de parámetros en el sistema.

4.2. Implementando el Módulo de Razonamiento Ontológico

Como fue detallado en el Capítulo 2, las ontologías son representaciones de alto nivel sobre un dominio en particular, las cuales son descriptas usando un formalismo con una semántica de alto poder expresivo, y que se construyen bajo principios y técnicas de Representación del Conocimiento y Razonamiento (*KR&R*, siglas del Inglés) con base en lógica computacional. Tales representaciones permiten definir una estructura para limitar qué es "interesante" y su objetivo principal es tener conocimiento disponible en un formato que haga posible que las máquinas lo procesen e intercambien de una manera fácil. Para esta primera definición de la arquitectura, se puede ver a las ontologías como conjuntos de *términos representacionales* declarados de una manera explícita o implícita, lo cual formalmente define una teoría lógica. Estos términos permiten la definición de diversos objetos y construcciones tales como *clases*, *relaciones*, *funciones*, entre otros. Para esta implementación, se propone utilizar la familia de lenguajes *Datalog+/–*, empleándolo solamente (por ahora) para ilustrar las nociones fundamentales detrás del enfoque propuesto. Esta elección está principalmente motivada por su flexibilidad para representar relaciones de cualquier aridad, el hecho de que existan muchas equivalencias entre fragmentos de *Datalog+/–* y otros lenguajes estándar tales como aquellos basados en lógicas de descrip-

ción (*description logic*)[2] y su sintaxis intuitiva, la cual permite referirse explícitamente a variables (las ventajas que ofrecen estas características serán más claras en los sucesivos ejemplos).

Una extensión de la familia *Datalog+/–*

Como se mencionó anteriormente, la base de la implementación de este módulo es *Datalog+/–* y para eso realizaremos primero un repaso de alto nivel respecto a las nociones fundamentales involucradas (una discusión más detallada de este tema es realizada en el Capítulo 2). Las unidades de representación básica de los lenguajes de *Datalog+/–* son constantes, variables y un elemento especial llamado "*valores nulos*"(los primeros dos de estos elementos son compartidos con muchos lenguajes, mientras que el último es menos común). Los valores nulos pueden ser vistos como un tipo especial de variables: ellos describen valores que *no son conocidos aún*; mientras que dos valores nulos diferentes pueden representar el mismo valor, dos constantes diferentes se asume que representan valores diferentes. Los objetos más básicos se llaman *átomos* (o *fórmulas atómicas*), los cuales son obtenidos a partir de *símbolos predicativos* aplicados a constantes, variables y valores nulos; por ejemplo, *perro*(*fido*) es una fórmula atómica construida por aplicar el símbolo predicativo *perro* a una constante *fido*, expresando que Fido es un perro. Una *instancia de base de datos* es modelada como un conjunto de átomos que solo contienen constantes.

Obtención de tuplas de base de datos a partir de las fuentes de datos. En la Sección 4.1 se abordó brevemente la función del módulo de Ingesta de Datos, el cual tiene la tarea de capturar los datos desde las fuentes usadas por el sistema (cf. Figura 4.1) y preparar la base de conocimiento que va a ser usada durante el proceso, llevado a cabo por los dos módulos principales, para razonar y responder las consultas. Con la intención de brindar más claridad sobre cómo esta base de conocimiento es inicializada cuando nuevos datos llegan, se discutirán tres ejemplos, los cuales se ilustran con mayor detalle en la Figura 4.3:

- Datos provenientes de la *National Vulnerability Database (NVD)*[3]: Disponibles principalmente utilizando formatos JSON y XML, e incluye las vulnerabilidades a me-

[2]Si se desea usar una implementación provista por un estándar de la Web Semántica, esto puede ser hecho tomando las medidas adecuadas para primero traducir la sintaxis de *Datalog+/–*.

[3]https://nvd.nist.gov/vuln/data-feeds

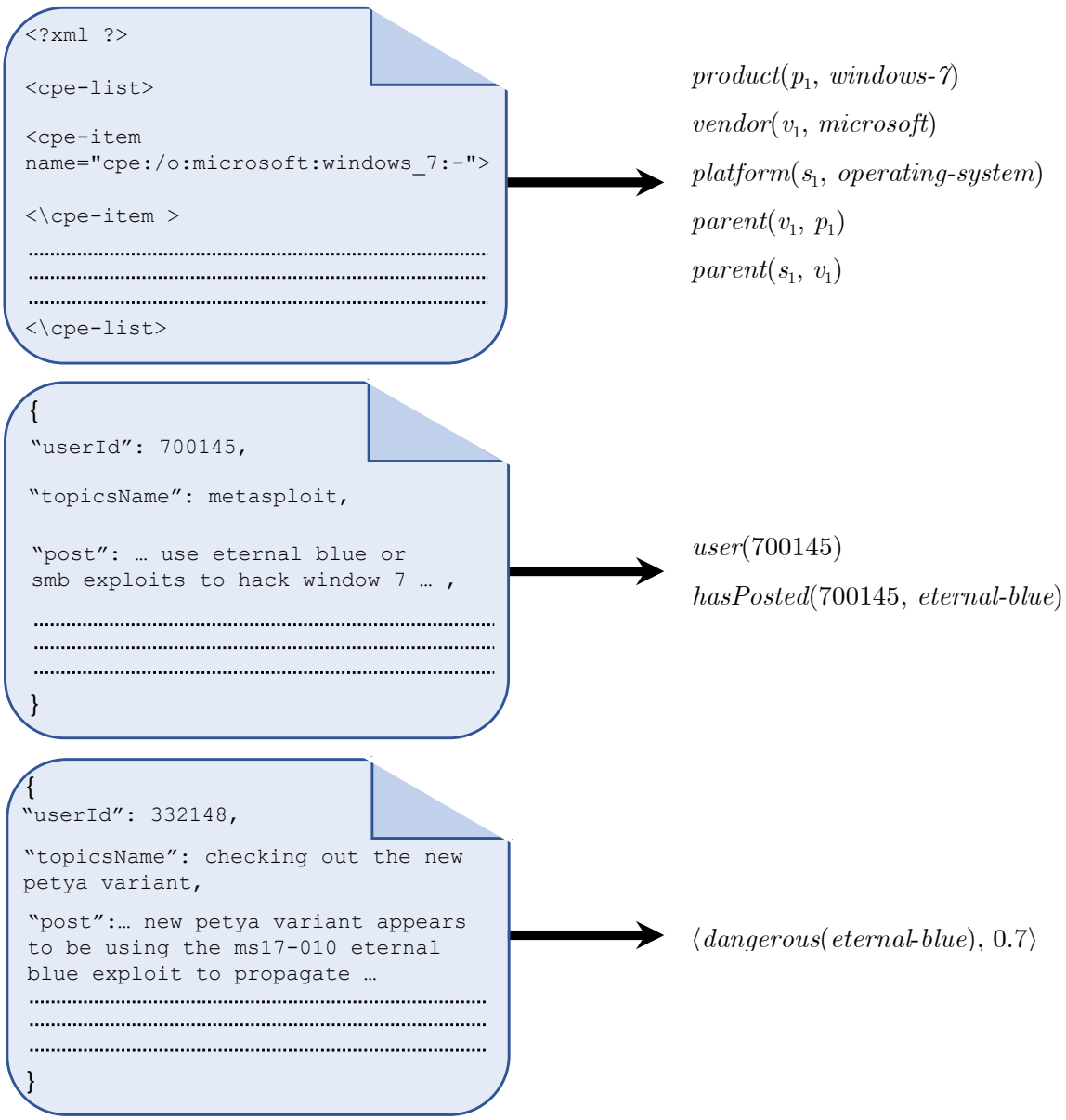

Figura 4.3: Ejemplo de cómo tuplas de base de datos pueden ser extraídas a partir de tres fuentes diferentes: datos de la *National Vulnerability Database* (*posición superior*, XML), datos obtenidos a partir de mercados y foros de la Darknet y curados por una empresa de ciberseguridad (*posición central*, JSON) y datos de Twitter (*posición inferior*, JSON).

dida que se descubren (con datos sobre las métricas de la vulnerabilidad tales como *CVSS scores*), *Common Platform Enumeration (CPE) Dictionary*, datos relacionados a configuraciones y *checklists*, entre otros.

- Datos provistos por una empresa de ciberseguridad: La Figura 4.3 (*en la parte central*) ilustra un ejemplo simplificado de los datos obtenidos a partir de las colaboraciones realizadas con CYR3CON[4], la cual se focaliza en el análisis de mercados y foros de *hackers maliciosos* en busca de información que pueda ser aprovechada para mejorar el desempeño en ciertas tareas tales como predecir la disponibilidad de *exploits* (por ejemplo, para saber qué parches deberían aplicarse primero), *CVSS scores* antes de que sean publicados por la NVD o cuándo esperar que surja cierto tipo de ataque.

- Datos de Twitter: Generalmente disponibles en formato JSON, pero también pueden ser fácilmente procesados en texto plano dada su estructura.

Como se puede ver en la Figura 4.3, el mapeo entre estas fuentes de datos y tuplas *Datalog+/–* es casi directo; los ingenieros de datos diseñan el modelo de datos basados en el contenido disponible de acuerdo al origen de los datos, y luego se requiere utilizar módulos de traducción de formatos simples con el objetivo de obtener la información preparada para incorporar a la base de datos. Es evidente que el número y complejidad de las fuentes de datos usados por el sistema tendrán un efecto directo en estas tareas, las cuales pueden volverse muy complejas si, por ejemplo, se tiene más de un origen para los mismos datos (es muy posible que haya inconsistencias), los datos están potencialmente incompletos o no son confiables, o una o más de las fuentes consideradas son actualizadas con mucha frecuencia. En este ejemplo, se muestra cómo la mayoría de estas cuestiones pueden ser evitadas a partir de la utilización de fuentes de datos que ya han atravesado un proceso de "curado" realizado por terceros; por ejemplo, CYR3CON lleva a cabo tareas de limpieza de datos, elige fuentes basadas en su confiabilidad y provee un esquema unificado para que sus clientes puedan hacer consultas.

Reglas TGDs, EGDs y NCs. Estas reglas permiten especificar estructuras de representación de diferentes tipos: las *dependencias de generación de tupla* (TGD, también conocidas como *reglas existenciales*), las dependencias de generación de igualdad (EGD) y las restricciones negativas (NC). Las TGDs son de la forma $body \rightarrow head$, donde *body*

[4]https://www.cyr3con.ai/

y *head* son fórmulas lógicas y *head* puede contener un constructor lógico especial llamado *cuantificador existencial*. Este constructor es fundamental para realizar el razonamiento lógico debido a que permite crear los *valores nulos*; estos valores representan nuevo conocimiento que es producido, aunque los valores exactos de los valores nulos no son conocidos aún (a lo cual comúnmente se hace referencia como *invención de valores*). Una instancia de base de datos *satisface* una TGD si es posible obtener un mapeo, que preserve estructura (más precisamente un objeto llamado *homomorfismo*), desde *body* a átomos de la base de datos; en tales casos, es posible usar una extensión de ese mapeo para obtener átomos basados en la fórmula *head*. A continuación, se presenta un ejemplo en el dominio de productos de *software* para ilustrar brevemente estos conceptos.

Ejemplo 4.1 *Se sabe que "todo producto de software tiene una plataforma (platform) en la cual se ejecuta (runsOn)"; este conocimiento puede ser descripto utilizando la siguiente TGD:*

$$product(P) \rightarrow \exists S platform(S) \wedge runsOn(P, S)$$

En otras palabras, si se sabe que P es un producto de software entonces es posible inferir que P tiene una plataforma en la cual se ejecuta. Observe que es posible que la plataforma sea desconocida en el momento en que la TGD sea aplicada, aunque al menos se sabe que la plataforma existe. ∎

De una manera similar, las EGDs son reglas de la forma $body \rightarrow X = Y$, donde *body* es una fórmula, y X e Y son variables que aparecen en *body*; la semántica de este tipo de reglas es que las variables mencionadas deben ser iguales. La definición de satisfacción de una EGD por una instancia de base de datos es análoga a la definición para TGDs, excepto que la consecuencia de aplicarla causa que valores se vuelvan iguales si es posible; si no, el procedimiento falla. La clase de dependencias funcionales, muy conocida en base de datos, puede ser vista como un caso particular de EGDs.

Ejemplo 4.2 *Considere que se sabe que "todo producto de software (product) tiene a lo sumo una plataforma (platform) en la que se ejecuta (runsOn)"; esto puede ser descripto usando la EGD:*

$$\begin{array}{llll} product(P) & \wedge & platform(S_1) & \wedge \\ platform(S_2) & \wedge & runsOn(P, S_1) & \wedge \\ runsOn(P, S_2) & \rightarrow & S_1 = S_2 & \end{array}$$

∎

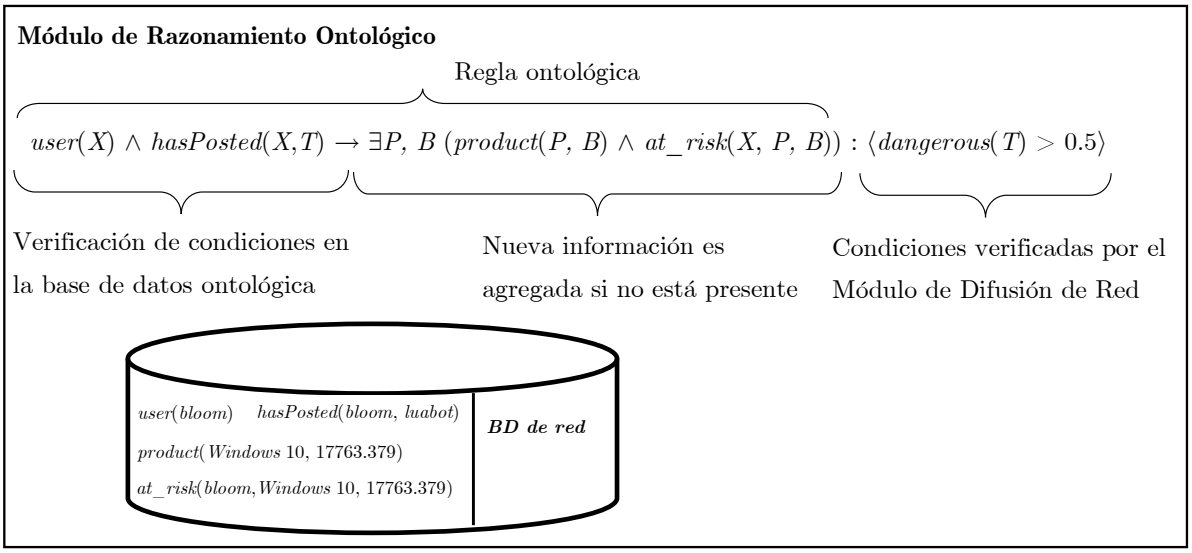

Figura 4.4: Un ejemplo del Módulo de Razonamiento Ontológico con una instancia de base datos y una TGD extendida.

El último tipo de reglas consideradas son las NCs, las cuales son de la forma $body \rightarrow \bot$, donde *body* es una fórmula y $\bot$ es un símbolo que es usado para representar una *contradicción* (toda fórmula lógica que es incondicionalmente falsa). Intuitivamente, este tipo de reglas dicen que *body* no puede ser verdad o, en otras palabras, todas las condiciones que componen *body* no pueden ser simultáneamente verdaderas. Una instancia de base de datos *satisface* una NC si no es posible encontrar un mapeo, que preserve estructura, desde *body* a átomos de la base de datos.

Ejemplo 4.3 *Considere que se sabe que "si algo es un producto de software (product) entonces no es una plataforma (platform)"; esto puede ser expresado en la siguiente NC:*

$$product(P) \quad \wedge \quad platform(P) \quad \rightarrow \quad \bot$$

Esto significa que un producto P no puede ser tanto un producto de software como una plataforma al mismo tiempo. ∎

Fórmulas ontológicas extendidas. La Figura 4.4 muestra el *Módulo de Razonamiento Ontológico* con una instancia de base de datos y una TGD específica circunscriptos al dominio de ciberseguridad. Para que la regla sea aplicada requiere que el cuerpo de la

regla (en este caso sólo tiene parte ontológica) sea satisfecho en la base de datos de la ontología y las condiciones globales sobre el estado de la red sean satisfechas en la base de datos de la red (en este caso la única parte de la red). Cuando la regla es aplicable, el nuevo conocimiento obtenido a partir de la cabeza es incorporado a la base de datos (en este caso solo de la ontología). El contenido de la base de datos muestra, en una representación formal, información sobre productos de *software*, usuarios, publicaciones, productos en riesgo de ser atacados y actores que pueden ser responsables por dichos ataques. Como se mencionó antes, las reglas constituyen la herramienta que permite crear nuevo conocimiento; en este caso, las reglas no corresponden estríctamente al formalismo clásico de *Datalog+/–*, sino que se utiliza una extensión ligeramente diferente. Estas TGDs extendidas pueden ser divididas en dos partes: la *parte ontológica* y la *parte de la red*. Si sólo se considera la parte ontológica de la regla entonces se tendría una regla *Datalog+/–* clásica, por lo que dichos elementos mantienen su sintaxis y semántica en la manera que es conocida y descripta con detalle en el Capítulo 2. En la Figura 4.4 se puede ver particularmente una regla TGD extendida donde en el cuerpo sólo se dispone de la parte ontológica y en la cabeza se especifica una parte ontológica y otra de red; al excluir la parte de red se obtiene una TGD *Datalog+/–* clásica e intuitivamente dice: *Si un usuario X ha publicado sobre un tema T entonces existe un producto de software que está en riesgo y X es responsable por dicha situación*. Es fácil ver que esta versión simplificada de la regla es satisfecha por la instancia de base de datos. La parte de red puede estar referida a componentes específicos (nodos o arcos) o puede hacer referencia a condiciones globales respecto al estado de la red; esto último es lo que se especifica en la regla de ejemplo de la figura y, en este caso particular se establece la condición de que T sea considerado *dangerous* (peligroso) con una confianza de al menos 0,5. De manera análoga, es posible extender las EGDs y las NCs. Más adelante se realizará una explicación sobre cómo se evalúan las condiciones de red y cómo se modela el proceso de difusión.

De los datos a la ingeniería de conocimiento. Más allá de los aspectos de ingeniería de datos más básicos cubiertos anteriormente (ilustrados en la Figura 4.3), el uso de expertos del dominio para diseñar y construir ontologías es esencial al menos como parte de dos esfuerzos principales. Primero, las tareas de ingeniería de datos involucran la selección de fuentes de datos, un proceso de curación y su incorporación a un modelo de datos diseñado adecuadamente; y segundo, en las tareas de ingeniería de conocimiento más generales, además de tratar con cuestiones relativas a los datos, también se debe considerar el desarrollo de reglas que capturen el conocimiento del dominio, cómo aprender

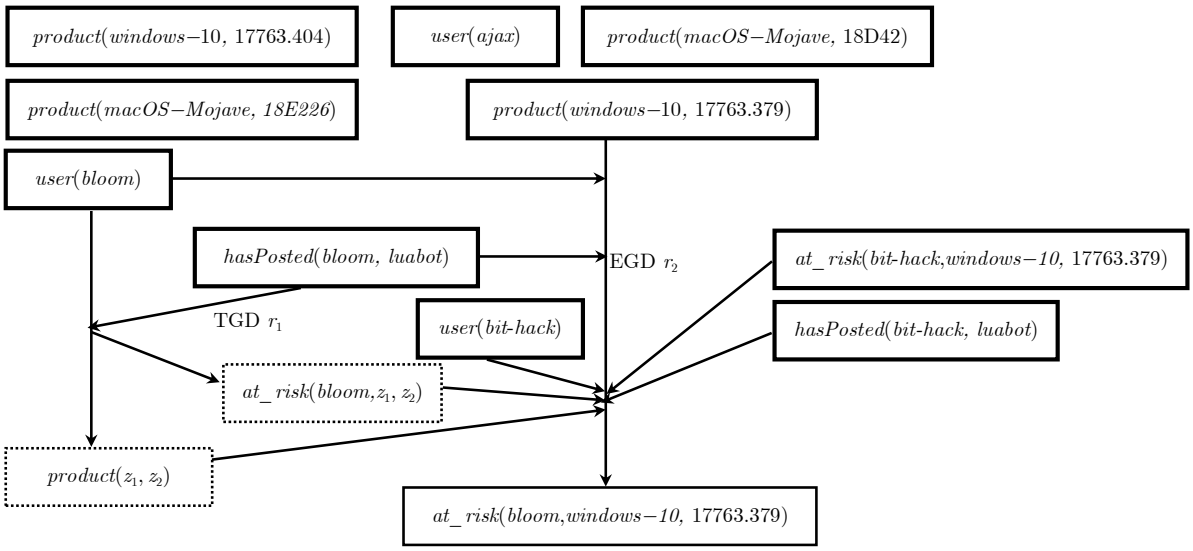

Figura 4.5: Ejemplo de la estructura de datos *chase* (producida por el procedimiento del mismo nombre).

automáticamente otras reglas y combinar ambas formas de conocimiento bien fundada y basada en principios. Siempre se debe tener en cuenta que en relación a los datos, la incorporación de mayor diversidad en la información (más expertos, más fuentes de datos, más enfoques de aprendizaje automáticos, etc.) tendrá un efecto directo en la complejidad de las tareas de ingeniería como limpieza de datos, manejo de inconsistencia, gestión de confianza, entre otras.

El procedimiento chase y la estructura de datos

La principal tarea del Módulo de Razonamiento Ontológico es responder consultas en el dominio de aplicación utilizando el conocimiento disponible. Con el propósito de lograr disponer de esta capacidad, *Datalog+/–* usa un procedimiento llamado *chase*; esencialmente, el *chase* puede ser usado para hacer explícito todo el conocimiento que está implícitamente presente en la forma de reglas y conocimiento atómico en la base de datos. Esto se logra por la *aplicación exhaustiva* de un procedimiento que inicia a partir de la base de datos, y luego cada regla es aplicada tantas veces como sea posible. Cuando una TGD es aplicada, nuevos átomos pueden ser agregados y *nuevos valores nulos* pueden ser inventados si la regla lo requiere; por otro lado, cuando una EGD es aplicada, se fuerza

que dos valores se vuelvan iguales y, en ese caso, es posible que un valor nulo introducido por una TGD pueda ser resuelto (es decir, reemplazado por una constante o equiparado a otro valor nulo). Asimismo, cuando las NCs son aplicadas el *chase* verifica si la condición no es satisfecha en la base de datos. Se debe considerar que el proceso puede fallar en ciertos casos cuando las EGDs o NCs no pueden ser satisfechas; en este caso, hay una inconsistencia en la base de conocimiento que debe ser resuelta antes que el proceso para responder consultas pueda ser llevado a cabo correctamente. La estructura de datos generada por el procedimiento *chase* (esencialmente un grafo y una colección de mapeos) es también generalmente referida como "el *chase*".

En todos los casos, excepto en los más simples, cada regla se puede aplicar varias veces y, en combinación con la invención de valores como se discutió anteriormente, es muy probable que esto lleve como resultado a un proceso infinito. Sin embargo, bajo algunas condiciones sintácticas o semánticas impuestas sobre el conjunto de reglas, es posible responder consultas sólo usando una parte limitada del *chase*, de manera que se pueda garantizar la terminación, en lugar de utilizar el *chase* completo (potencialmente infinito). La Figura 4.5 muestra el resultado de aplicar el procedimiento *chase* a una instancia de base de datos con dos reglas. Los rectángulos con bordes más gruesos representan información atómica de la base de datos, aquellos con bordes punteados representan información atómica con valores nulos (representando información parcialmente desconocida) y las flechas representan la aplicación de reglas ontológicas (en este caso TGDs y EGDs). La regla r_1 es la regla ontológica (TGD) de la Figura 4.4 y la regla r_2 es una EGD que puede ser leída como: *Si dos usuarios han publicado sobre un tema T y ambos son responsables de poner en riesgo algún producto de software, entonces dichos productos de software, en realidad, son lo mismo* (esto no significa que dicha regla deba ser satisfecha en general, sino que simplemente es una regla utilizada para completar hipótesis). Cuando el *chase* aplica la regla r_1, dos nuevos valores nulos (z_1 y z_2) son inventados y dos nuevos átomos son agregados: $product(z_1, z_2)$ y $at\_risk(bloom, z_1, z_2)$. El *chase* ahora puede continuar computando en base a los nuevos elementos inferidos, razón por la cual es posible aplicar la regla r_2; el resultado es que nuevos valores nulos son mapeados a valores conocidos: $z_1 = windows - 10$ y $z_2 = 17763.379$.

4.3. Implementando el Módulo de Difusión de Red

Los procesos de difusión de red aparecen como elementos fundamentales en la comprensión de fenómenos en un amplio rango de disciplinas, tales como ciencias de la computación [JP17], biología [MRV16], sociología [Cen15], economía [BNJU17] y física [Bia15].

Modelo de red y reglas de difusión. El modelo subyacente de este módulo representa una red cuyos nodos y arcos tienen asignados diferentes *etiquetas locales* con sus respectivos *valores de confianza*, lo cual se utiliza para modelar el conocimiento del mundo. Particularmente, estos valores de confianza permiten representar conocimiento incierto. Adicionalmente, también se dispone de *etiquetas globales* para representar el estado general de la red y sus valores dependen a su vez de los valores de ciertas etiquetas locales determinadas. Como es de esperarse, la confianza en el conocimiento representado por estas etiquetas es dinámica, y una forma de capturar dicha naturaleza dinámica es utilizar reglas lógicas. Para lograr esto se propone emplear como base el formalismo MANCa-Log [SSC13] en la implementación de este módulo, debido a que cumple con un conjunto de criterios de diseño con características fundamentales en el modelado de procesos de difusión, teniendo en cuenta el tipo de tareas de respuesta a consultas en las que se tiene interés.

Criterios de diseño para procesos de difusión multi-atributos. A continuación, se detalla brevemente los criterios de diseño propuestos en [SSC13], los cuales son esencialmente requerimientos básicos para asegurar la representación bien fundada de incertidumbre y aplicabilidad práctica:

(1) *Nodos y arcos pesados y con múltiples etiquetas*: aunque muchos *frameworks* que permiten el estudio de procesos de difusión asumen que solo hay un tipo de vértice que puede volverse "activo" y solo una posible relación entre nodos, con mucha frecuencia nodos y arcos tienen múltiples propiedades.

(2) *Representación explícita del tiempo*: un modelo con alto poder expresivo requiere la representación de relaciones temporales entre condiciones en la estructura de la red, el estado actual de las cascadas en proceso y cómo la influencia se propaga.

(3) *Relaciones temporales no-markovianas*: teniendo en cuenta que las dependencias temporales deberían ser capaces de abarcar varias unidades de tiempo, no es adecuada la

suposición de que no hay memoria (en inglés, la propiedad se conoce con el nombre de "memorylessness").

(4) *Representación de incertidumbre*: generalmente no es posible evaluar los atributos de todas las componentes de una red con total certidumbre.

(5) *Procesos en competencia*: muchas situaciones del mundo real presentan múltiples procesos de red tales que el éxito de uno depende del fracaso del otro.

(6) *Procesos no monotónicos*: tener la capacidad de representar cascadas en competencia requiere que el número de nodos que satisfacen una cierta propiedad pueda fluctuar sin la necesidad de satisfacer monotonicidad.

(7) *Tratabilidad*: los *frameworks* que abordan estos problemas deben ser computacionalmente tratables y ofrecer posibilidades de formas prácticas que mejoren la escalabilidad.

La esencia de la reglas de difusión que se necesitan representar, puede ser expresada como reglas del estilo:

$$\textit{influence},\quad \textit{neighbors},\quad \textit{target} \;\rightarrow\; L$$

donde:

- *target* es una fórmula lógica que identifica los componentes de la red que están afectados por la regla.
- *neighbors* es una fórmula lógica que especifica las condiciones que los vecinos (*neighbors*) del nodo objetivo, el cual cumple el criterio definido en *target*, deben verificar.
- *influence* es una función que determina cuánta influencia es ejercida por los vecinos (*neighbors*).
- L es una *etiqueta local* relativa a un nodo objetivo, que cumple el criterio definido en *target*, cuyo nivel de confianza es modificado por la regla.

Consultas de red. El principal servicio provisto por este módulo es proveer respuestas a subconsultas en la forma de verificación de condiciones en la red (cf. Figura 4.6), cuando

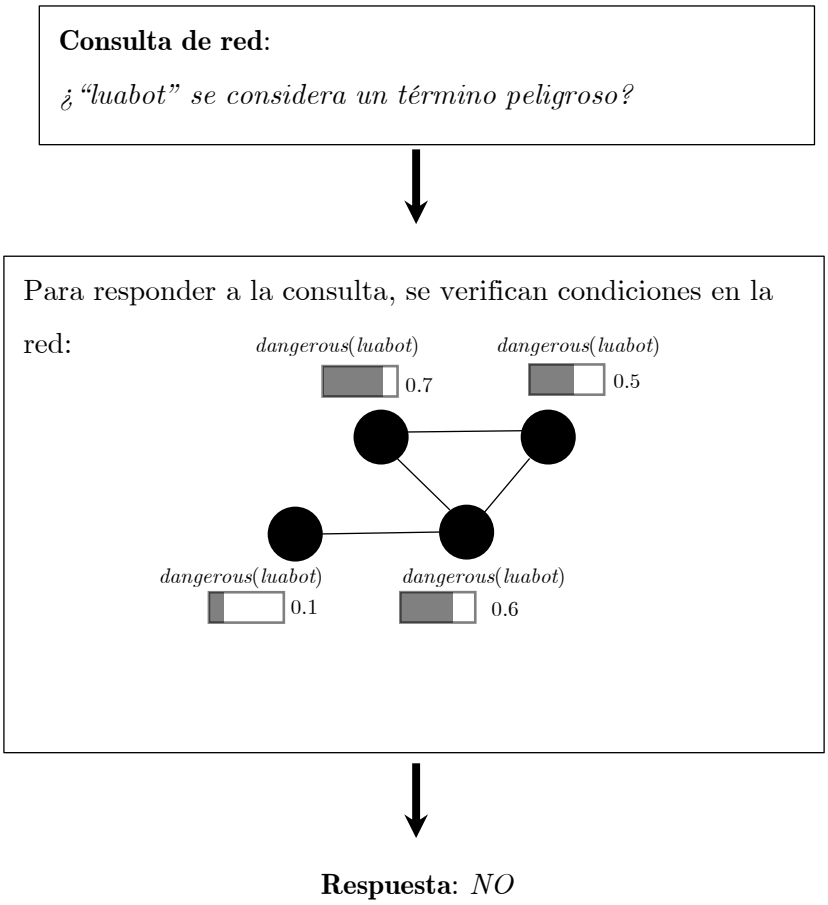

Figura 4.6: Ejemplo de una consulta y modelo de red.

el Módulo de Razonamiento Ontológico así lo requiere. Con la intención de ilustrar las ideas principales se hará foco en consultas sobre el *estado global* de la red. Este tipo de conocimiento global puede ser visto como si fuera un resumen del conocimiento local mantenido por cada nodo; específicamente, tales resúmenes son el resultado de aplicar una función sobre un conjunto de etiquetas locales. Las conocidas funciones de agregación usadas en base datos (promedio, max, min, etc.) pueden ser empleadas como instancias de tales funciones de resumen. Se espera que la elección de la función particular esté sujeta al dominio de aplicación y la etiqueta en cuestión.

Ejemplo 4.4 *Considere la red mostrada en la Figura 4.6, la cual consiste de cuatro nodos interconectados y una única etiqueta que indica la percepción de cada nodo respecto a qué tan peligrosa (dangerous) es la pieza de software "luabot"*[5]*. Cada nodo tiene una*

[5] *Linux.Luabot* es un malware descubierto a finales de 2016 que infecta sistemas basados en Linux

perspectiva diferente de este asunto; una consulta general a la red puede usar una función predefinida "average" (promedio) aplicada al conjunto de todas las etiquetas definidas como dangerous(luabot). La respuesta "Sí" o "No" depende de si el resultado del valor de la función es mayor que cierto umbral; en este caso, asumiendo que el umbral es 0,5*, la respuesta es "no".* ∎

Las respuestas a una consulta de red son computadas sobre una red que es el resultado de computar un proceso de difusión codificado por las reglas del estilo descriptas anteriormente. La red comienza con un estado inicial en el que cada componente tiene asignado diferentes niveles de confianza para cada una de las etiquetas que se encuentran disponibles; luego, se aplican las reglas tantas veces como sea posible, de manera similar al funcionamiento del procedimiento *chase*. El proceso finaliza cuando un *estado estable* de la red es alcanzado; es decir, ya no es posible aplicar reglas y cambiar los valores de confianza de alguna etiqueta.

Evolución de la red. La evolución del modelo de red ocurre en dos formas principales: la primera es la aplicación de las reglas de difusión como ya se ha mencionado y la segunda es por nueva información incorporada (nodos, arcos o etiquetas), ya sea porque nuevos datos se encuentran disponibles o como consecuencia de la aplicación de reglas ontológicas en el ORM. Como se había mencionado, se asume que los dos módulos ORM y NDM ejecutan sus procesos concurrente y asincrónicamente, por lo que pueden aparecer cambios en cualquier momento como puede observarse en el siguiente ejemplo.

Ejemplo 4.5 *La Figura 4.7 muestra un proceso de difusión para la red considerada en la Figura 4.6, el cual en este caso es regido por una única regla de difusión de red informalmente descripta en la parte superior de la Figura 4.6. Los valores de confianza de las etiquetas evolucionan a lo largo del tiempo y el resultado depende de la configuración de las etiquetas en sus respectivos vecinos (neighbors) en cada tiempo. Observe que la confianza de cada etiqueta en la red tienen un estado inicial en el tiempo* T_1 *y ellas cambian a lo largo del tiempo porque la regla es aplicada hasta que el estado de la red sea estable. Además, en el tiempo* T_3 *aparece un nuevo nodo (con borde punteado), el cual fue agregado por el Módulo de Razonamiento Ontológico trabajando en forma asincrónica. La red en el tiempo* T_4 *es usada para responder consultas.* ∎

empleando "troyanos" como medio de ataque; cf. `https://securityaffairs.co/wordpress/51155/malware/linux-luabot.html`

Regla de difusión:

"*Un nodo incrementará su creencia en 0.2 (con un máximo de 1), en el tiempo siguiente, de que un término es peligroso si al menos la mitad de sus vecinos creen que es peligroso con confianza de al menos 0.7.*"

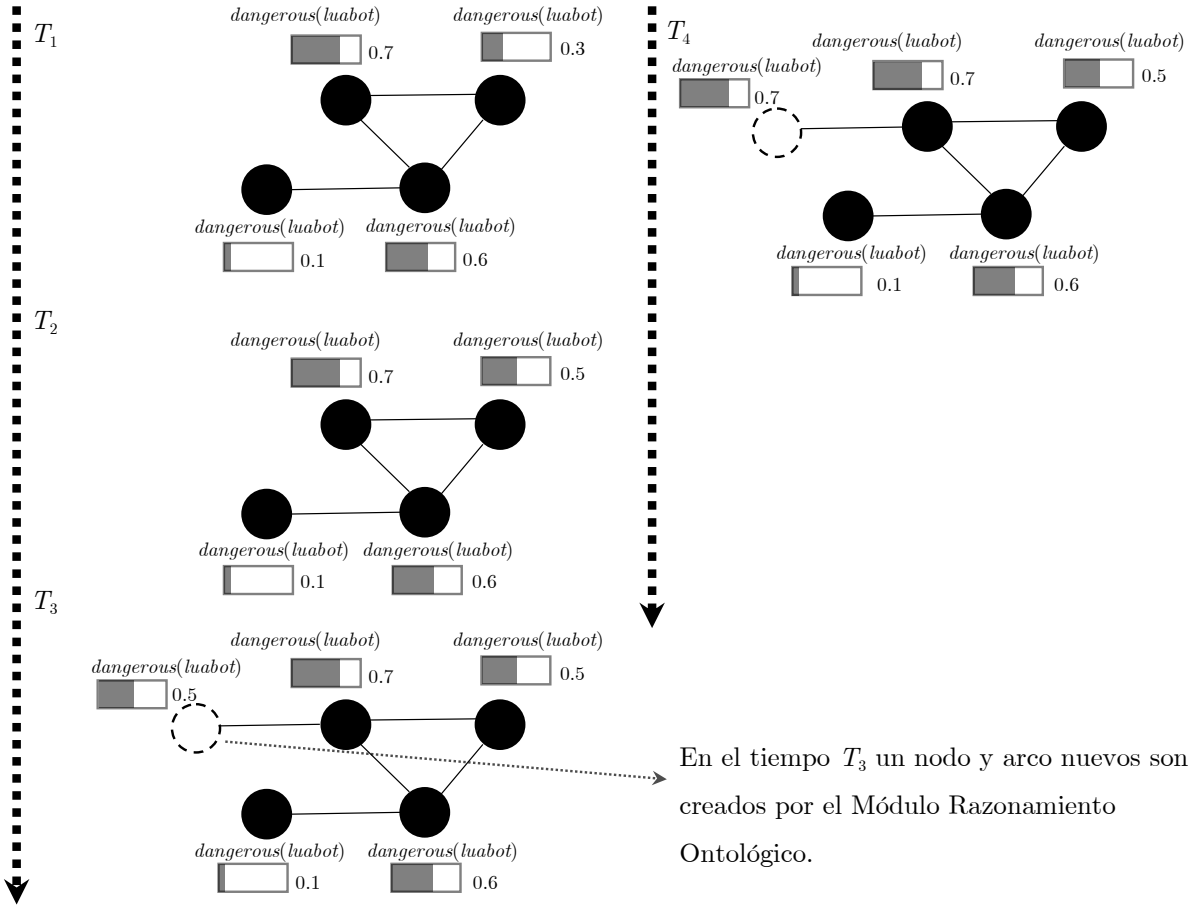

Figura 4.7: Ejemplo de un proceso de difusión.

Habiendo introducido las bases detrás del Módulo de Razonamiento Ontológico y del Módulo de Difusión de Red (y señalado la posibilidad de interacción entre ellos), es momento de pasar a discutir la principal tarea de razonamiento de la arquitectura.

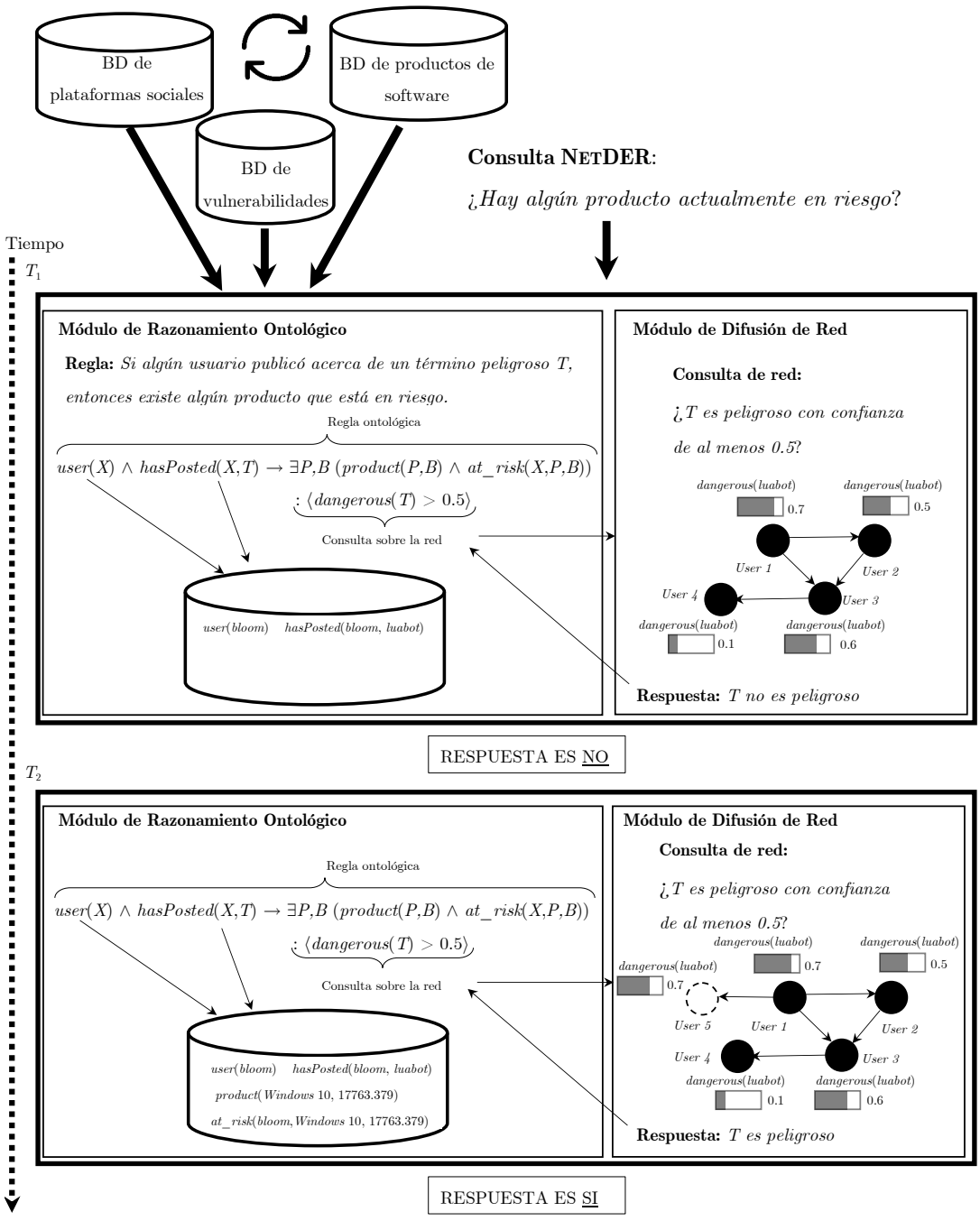

Figura 4.8: Una vista más detallada del ejemplo general mostrado en la Figura 4.1.

4.4. Consultas NetDER

Como se ha dicho anteriormente, el principal interés es disponer de la capacidad de responder consultas como parte de los esfuerzos para atacar problemas relacionados a comportamiento malicioso en plataformas sociales. Después de presentar los bloques de construcción básicos, se presenta un enfoque que busca aprovechar el poder expresivo de las ontologías y potenciarse de la sinergia que surge de combinarlas con procesos de difusión de red.

Esta arquitectura se caracteriza principalmente por (i) emplear formalismos de representación de conocimiento poderosos con la capacidad de generar hipótesis a través de la aplicación de reglas existenciales (TGDs) y EGDs, (ii) mantener y actualizar conocimiento posiblemente incompleto (haciendo uso de valores nulos) que pueda completarse en el futuro y (iii) modelar datos de red con etiquetas bajo incertidumbre. El siguiente ejemplo busca proporcionar mayor claridad respecto al proceso para responder consultas NetDER.

Ejemplo 4.6 *En el contexto general de la arquitectura ilustrada en la Figura 4.1, considere la consulta planteada en la Figura 4.8 dentro del dominio de ciberseguridad. En este caso, los datos de entrada son:*

- *Información proveniente de diferentes bases de datos acerca de productos de software.*
- *Vulnerabilidades de software.*
- *Datos provenientes de plataformas sociales: usuarios, publicaciones y sus interacciones.*

La consulta pregunta si hay o no algún producto de software en riesgo de sufrir un ataque en este momento (tipo de consulta comúnmente conocida como booleana debido a que su respuesta sólo puede ser "Sí" o "No"). La información proveniente de las diferentes fuentes debe ser integrada en una única base de conocimiento de la ontología y de la red. Observe que las respuestas a la consulta cambian desde el punto T_1 al tiempo T_2, dado que la respuesta a la consulta en el tiempo T_1 sería no, dada la información disponible; sin embargo, una vez que el Módulo de Razonamiento Ontológico aplica la regla de inferencia,

la respuesta pasa a ser Sí. Es decir, aunque el cuerpo de la regla NetDER *es satisfecha, no ocurre lo mismo con las condiciones globales sobre la red en el tiempo T_1; sin embargo, con la aplicación de otras reglas ontológicas en la base de conocimiento*[6]*, un nuevo nodo y arco son agregados en el tiempo T_2 y, después de que el proceso de difusión es ejecutado otra vez, la condición sobre la red es satisfecha y la regla en cuestión puede ser aplicada.*

■

En forma análoga, otros problemas respecto a comportamiento malicioso en plataformas sociales podrían ser abordados por diferentes instanciaciones del framework, por ejemplo:

- *Detección de noticias falsas* utilizando reglas ontológicas NetDER tales como:

  > "Si un usuario ha publicado P_1 y P_1 es similar a otra publicación P_2 que se sabe es una noticia falsa, entonces se puede hipotetizar que P_1 es una noticia falsa."

  Además, si es posible identificar el tema principal T_1 en P_1, entonces la regla ontológica NetDER podría ser anotada con condiciones sobre la red tales como "*important*(T_1)" para verificar si T_1 es considerado importante por los nodos de la red.

- *Detección de botnets* [BC16, BJC17, BJC19] utilizando reglas ontológicas NetDER similares a las discutidas para la detección de noticias falsas:

  > "*Si una cuenta es conocida que es un bot, entonces se puede hipotetizar que las cuentas asociadas (publican frecuentemente juntos, en horarios similares, etc.) también son bots y que posiblemente participen en la misma botnet.*"

- *Detección de bots combinadas con noticias falsas*; en tal caso se puede disponer de reglas ontológicas NetDER tales como:

[6] Por ejemplo, la regla podría representar algo como "Si un usuario cree que cierto software es peligroso con una confianza de al menos 0,5, entonces existe otro usuario que está relacionado al primero, es un experto y también cree lo mismo con confianza de al menos 0,5."

> "*Si es conocido que un usuario U_1 es un bot y ha publicado algo que se sabe es una noticia falsa, entonces se puede hipotetizar que existe otro usuario U_2 similar a U_1 tal que U_2 también ha publicado noticias falsas.*"

Como en los ejemplos anteriores, esta regla está destinada a funcionar como parte de los esfuerzos para generar hipótesis automáticamente basado en el conocimiento de cómo funcionan las *botnets*.

Desafíos para una implementación efectiva y eficiente de NetDER**.** La utilización de esta arquitectura para la solución de problemas de comportamiento malicioso en plataformas sociales, requiere superar distintos desafíos derivados de las características del problema y su dominio, lo cual implica profundizar trabajos de investigación y desarrollo direccionados hacia el logro de tales retos:

- *Ingeniería de conocimiento*: A fin de resolver un problema específico en el dominio general de comportamiento malicioso en plataformas sociales, la implementación de una instancia de NetDER conlleva que se involucre un experto del dominio con el propósito de definir reglas ontológicas NetDER en el Módulo de Razonamiento Ontológico y reglas de difusión en el Módulo de Difusión de Red. Está claro que los expertos podrían ser diferentes en cada caso, lo cual plantea la cuestión de integrar conocimiento proveniente de diferentes fuentes (esto en sí mismo es un desafío, discutido abajo). Asimismo, el hecho de que estas reglas puedan ser aprendidas automáticamente (o semi-automáticamente) y que se requiere obtener información atómica para su procesamiento, sugieren la necesidad de adquirir *datasets* puntuales que estén relacionados al problema específico o problemas relacionados que se estén abordando.

- *Integración de datos y conocimiento*: Los datos de entrada provienen de diferentes fuentes y, por lo tanto, es necesario que sean integradas en un esquema común. Esto es un problema clásico de gestión de datos y acceso a datos mediados por ontologías, por lo que ha sido ampliamente estudiado y existen herramientas de *software* maduras que pueden ser de gran ayuda para atacar este desafío. Esto no significa que los problemas vinculados a este aspecto no constituyan una dificultad, sino que como se mencionó anteriormente, la complejidad del dominio y el número y tipo de fuentes de datos pueden hacer que estas cuestiones se conviertan en un gran obstáculo.

- *Procesamiento inteligente de flujos de datos*: Los problemas relacionados a comportamiento malicioso en plataformas sociales involucran tareas de razonamiento que hacen uso de grandes cantidades de datos, los cuales son frecuentemente actualizados y en ocasiones no pueden ser almacenados todos a la vez. Esto conduce al planteamiento de problemas similares clásicos como el mantenimiento de vistas o procesamiento de flujos en base de datos, pero con una capa adicional de razonamiento complejo, lo que comúnmente se conoce como *stream reasoning* (procesamiento inteligente de flujos de datos) [VCvHF09].

- *Decidibilidad y tratabilidad de lenguajes de ontología*: Es ampliamente conocido que responder consultas cuando se consideran TGDs generales es indecidible [BMT11], y esto no mejora incluso cuando consideran fijos a la consulta y el conjunto de reglas lógicas [CGK13]. Sin embargo, es posible establecer ciertas restricciones en las TGDs y EGDs de manera que el proceso para responder consultas se vuelva decidible y computacionalmente tratable; tales restricciones pueden ser de dos tipos: sintácticas o semánticas. Se puede mencionar como condiciones sintácticas *guardedness* [CGL12], *stickiness* [CGP12] y *acyclicity*; y como condiciones semánticas *shyness* [LMTV12], *finite expansion sets*, *bounded treewidth sets* y *finite unification sets* [BMRT11]. Ésta es un área muy activa de investigación en teoría de base de datos y *Web Semántica* debido a que afecta directamente la expresividad y tratabilidad computacional de los lenguajes ontológicos.

- *Decidibilidad y tratabilidad de* NetDER: Debido a que en esta arquitectura el lenguaje de representación de conocimiento principal está basado en *Datalog+/–* y la principal tarea de razonamiento es llevada a cabo utilizando el procedimiento *chase* combinado con procesos de difusión, parece adecuado definir un procedimiento *chase* extendido que sea más general que el clásico. Por lo tanto, se espera que surjan al menos el mismo tipo de problemas que los mencionados arriba. Entonces, resulta necesario estudiar y analizar condiciones en las reglas lógicas, en relación a la ontología y la difusión, para garantizar la terminación y la tratabilidad computacional del proceso general para responder consultas. Adicionalmente, las *interacciones* entre el Módulo de Razonamiento Ontológico y el Módulo de Difusión de Red parecen tener un rol importante en este sentido, por lo que también se deberán tener en cuenta estas cuestiones.

4.5. Sumario

En este capítulo, se ha presentado la arquitectura NetDER, buscando que pueda ser utilizada para guiar la implementación de software destinado a resolver problemas de comportamiento malicioso en plataformas sociales. Aunque hay muchos trabajos que abordan problemas individuales con estas propiedades, las soluciones derivadas tienden a caracterizarse por su naturaleza *ad hoc* y, por lo tanto, se dificulta su utilización en conjunción con otras cuando se aborda combinaciones de dos o más problemas de comportamiento malicioso a la vez. También se debe tener en cuenta que, la utilización de reglas lógicas favorece la capacidad de disponer de *explicaciones*, debido a que es posible mostrar al usuario cómo las conclusiones fueron obtenidas (por ejemplo, mostrándoles una porción del *chase* y una animación del proceso de difusión involucrado). En el siguiente capítulo, se abordan lo detalles necesarios para posibilitar una implementación desde el punto de vista de los fundamentos teóricos, y en el siguiente a este último (Capítulo 6) se describen los resultados de una evaluación empírica.

Capítulo 5

Fundamentos teóricos de NetDER

En este capítulo se lleva adelante el desarrollo teórico para una posible implementación de la arquitectura NetDER presentada en el Capítulo 4. En la Sección 5.1 se abordan los fundamentos teóricos de la implementación del Módulo de Difusión de Red (NDM), luego en la Sección 5.2 se continúa con el Módulo de Razonamiento Ontológico (ORM), en la Sección 5.3 se profundiza sobre el Módulo de Respuesta a Consultas (QAM) y finalmente en la Sección 5.4 se presenta un caso de uso en el dominio de ciberseguridad con la intención de mostrar una posibilidad para instanciar los tres módulos en un escenario del mundo real.

5.1. El formalismo NetDiff

Ahora se presentarán los elementos principales involucrados en el formalismo NetDiff para implementar el Módulo de Difusión de Red (NDM). NetDiff generaliza el formalismo presentado en [SSS13, SSC13] para lo cual se modifica su sintaxis, se agrega un nuevo tipo de regla para resumir el estado de la red y se hacen otras generalizaciones sobre unidades básicas de representación de conocimiento en el lenguaje. Por lo tanto, NetDiff nos proveerá la capacidad para representar y razonar acerca de redes complejas y los procesos de difusión que se dan en ellas.

Se asume que las entidades (personas, agentes, etc.) se organizan en un grafo dirigido (o red) denotado $Graph = (V, E)$, donde el conjunto de nodos V corresponde a las entidades y el conjunto de arcos E modela las relaciones entre ellos. También se asume un conjunto

finito de *símbolos predicativos para etiquetas* $\mathcal{P} = \mathcal{P}_{nloc} \cup \mathcal{P}_{eloc} \cup \mathcal{P}_{glo}$, un dominio de datos posiblemente infinito Δ y un conjunto finito de *etiquetas* $\mathcal{L}$, el cual se encuentra particionado en tres conjuntos:

- *etiquetas locales de nodo* $\mathcal{L}_{nloc}$: etiquetas que pueden ser aplicadas a nodos de la red y que se crean a partir de símbolos predicativos en $\mathcal{P}_{nloc}$.
- *etiquetas locales de arco* $\mathcal{L}_{eloc}$: etiquetas que pueden ser aplicadas a arcos de la red y que se crean a partir de símbolos predicativos en $\mathcal{P}_{eloc}$.
- *etiquetas globales* $\mathcal{L}_{glo}$: etiquetas que hacen referencia al estado global de la red (éstas son una generalización de las *etiquetas locales de nodo*), no pueden ser aplicadas a ningún nodo o arco de la red sino que se aplican sobre la red en sí y, además, éstas se crean a partir de símbolos predicativos en $\mathcal{P}_{glo}$.

La idea principal de las etiquetas es que puedan ser utilizadas para representar diferentes atributos de alguna entidad (nodos en la red) y sus conexiones con otras entidades (arcos en la red). Cada *etiqueta* tiene la forma $p(t_1, ..., t_n)$, donde $t_i \in \Delta$ y $p \in \mathcal{P}$. Los símbolos predicativos $node \in \mathcal{P}$ y $edge \in \mathcal{P}$ se utilizan para identificar los componentes de la red de la siguiente manera: $node(t_1), ..., node(t_n) \in V$ y $edge(t'_1, t'_2), ..., edge(t'_n, t'_{n+1}) \in E$. En ocasiones, se utiliza $\mathcal{G} = V \cup E$ para denotar el conjunto de todos los *componentes* (nodos y arcos) en la red—-por lo tanto, $c \in \mathcal{G}$ podría ser ya sea un nodo o un arco— y la constante especial Ω será usada como una abstracción de la red en su totalidad.

Ejemplo 5.1 *Considere la red de una plataforma social* $Graph_{soc}$*, cuya estructura está definida por el grafo* $Graph_{soc} = \Big(\{node(n_1), node(n_2)\}, \{edge(n_1, n_2)\}\Big)$*. Además, se definen las etiquetas* $\mathcal{L}_{nloc} = \mathcal{L}_1 \cup \mathcal{L}_2$ *donde* $\mathcal{L}_1 = \{expert(ddos), expert(botnet), expert(trojan), expert(malware)\}$ *que permiten representar un experto en DDoS (ataque de denegación de servicio distribuido), uno en botnets, uno en troyanos y otro en malware, respectivamente. Asimismo, se definen las etiquetas* $\mathcal{L}_2 = \{bel\_dang(gooligan), bel\_dang(luabot)\}$ *para representar nodos que crean que gooligan y luabot son términos considerados peligrosos descubiertos en la Web, respectivamente. También, se define un conjunto con una única etiqueta local de arco* $\mathcal{L}_{eloc} = \{close(trojan, windows\text{-}10)\}$*, la cual representa que dos nodos conectados son cercanos (close, en Inglés) respecto a "troyanos" para "windows-10". Finalmente, se define el conjunto de etiquetas globales* $\mathcal{L}_{glo} = \{dangerous(gooligan), dangerous(luabot)\}$

para representar qué tan peligroso son considerados los términos gooligan y luabot por la red en general. ∎

Teniendo en cuenta que la mayoría de los dominios interesantes presentan incertidumbre en alguna de sus variantes, es importante que las etiquetas puedan extenderse de alguna forma para poder disponer de la capacidad para representar estas situaciones. Los *átomos de red* son las unidades formales que permiten agregar conocimiento incierto en las etiquetas, tal que todo átomo de red $\langle L, bnd\rangle$ asocia una etiqueta L con un intervalo $bnd \subseteq [0, 1]$. Adicionalmente, estos elementos pueden ser usados para definir un mundo W, el cual es un conjunto de átomos de red donde para cada $L \in \mathcal{L}$ existe exactamente un átomo $\langle L, bnd\rangle \in W$. Los mundos son importantes porque permiten describir el estado de la red, esto es, se puede pensar a dichos estados como la asociación de un mundo a cada componente de la red (nodo o arco) o la red completa para cada punto de tiempo. Por lo tanto, si se sabe el estado de la red entonces se puede saber qué conocimiento vale y cuál no; entonces un mundo W satisface un átomo de red $\langle L, bnd\rangle$ si y sólo si existe $\langle L, bnd'\rangle \in W$ tal que $bnd' \subseteq bnd$. Asimismo, los átomos de red pueden ser usados para formar *fórmulas de red* usando variables, substituciones y conectivos estándar: $\wedge$, $\vee$, $\neg$; su satisfacción puede ser definida con substituciones para obtener átomos de red conectados con operadores lógicos y su satisfacción puede ser generalizada inductivamente, como se realiza usualmente.

Otra suposición importante es que existe algún número natural t_{max} que especifica la cantidad total de tiempo que se está considerando (es decir, se trabaja con un horizonte fijo). A medida que el tiempo progresa, el peso asociado con una etiqueta puede ya sea incrementar o decrementar y, por lo tanto, volverse más o menos certero.

Para este caso, el conocimiento sobre el dominio puede ser definido usando estructuras atómicas y reglas. Las estructuras atómicas son llamadas *hechos* NetDiff y pueden tener alguna de las siguientes formas:

$$(v, \langle L_{nloc}, bnd\rangle) : [t_1, t_2]$$
$$(e, \langle L_{eloc}, bnd\rangle) : [t_1, t_2]$$
$$(\Omega, \langle L_{glo}, bnd\rangle) : [t_1, t_2]$$

donde $v \in V$, $e \in E$, $L_{nloc} \in \mathcal{L}_{nloc}$, $L_{eloc} \in \mathcal{L}_{eloc}$, $L_{glo} \in \mathcal{L}_{glo}$, $bnd \subseteq [0, 1]$ y $[t_1, t_2] \subseteq [0, t_{max}]$.

Como puede observarse, es posible distinguir tres tipos de hechos NetDiff referidos a información sobre nodos (*node*), arcos (*edge*) o una abstracción de la red completa (Ω).

Estos hechos NetDiff declaran que el átomo de red $\langle L_{nloc}, bnd\rangle$ ($\langle L_{eloc}, bnd\rangle$ y $\langle L_{glo}, bnd\rangle$, respectivamente) es verdadero para el nodo v (arco e y la red completa Ω, respectivamente) durante cada tiempo $t \in [t_1, t_2]$. Un ejemplo de un hecho NetDiff basado en el Ejemplo 5.1 es:

$$F = (node(n_1), \langle expert(malware), [1, 1]\rangle) : [0, t_{max}].$$

Aquí se especifica que el nodo $node(n_1)$ se considera, con total certeza, que es un experto en *malware* a lo largo del conjunto entero de puntos de tiempo. Ahora ya podemos enfocarnos en las *reglas* NetDiff, de las cuales hay de tres tipos: (a) *restricciones de integridad*, las cuales permiten representar dependencias en las propiedades de los componentes del grafo; (b) *reglas locales*, las cuales tienen un símbolo predicativo de $\mathcal{P}_{nloc}$ en la cabeza con variables y/o constantes que es usado para obtener etiquetas locales; y (c) *reglas globales*, las cuales tienen un símbolo predicativo de $\mathcal{P}_{glo}$ en la cabeza con variables y/o constantes que es usado para obtener etiquetas globales. La idea detrás de las etiquetas locales es que: un nodo que cumple un cierto criterio es influenciado por el conjunto de sus vecinos quienes poseen ciertas propiedades. La cantidad de influencia ejercida sobre un nodo por sus vecinos es especificada por una *función de influencia*. Como resultado, una regla local consiste de cuatro partes principales: (i) una función de influencia, (ii) un criterio de vecinos, (iii) un criterio de nodos objetivo y (iv) un criterio de etiquetas locales objetivo. Intuitivamente, (i) especifica cómo los vecinos influencian al nodo en cuestión, (ii) especifica cuáles de los vecinos pueden influenciar al nodo, (iii) especifica el criterio para determinar que nodos van a ser influenciados y (iv) es la propiedad del nodo que cambia como resultado de la influencia.

De una manera similar, las reglas globales definen cómo el estado de una o más etiquetas globales es actualizado de acuerdo a un conjunto de componentes de la red que posee ciertas propiedades. El valor actualizado de cualquier etiqueta global es determinado por una función de agregación, las cuales se definen de una manera análoga a como se definen en contextos de base de datos relacionales (tales como promedio, mínimo, máximo, etc.). Las reglas mencionadas en (a), las restricciones de integridad, tienen la siguiente estructura:

$$A(\mathbf{X}) \hookleftarrow B(\mathbf{Y})$$

donde $A(\mathbf{X})$ es una fórmula de red atómica y $B(\mathbf{Y})$ es una conjunción de fórmulas de red atómicas. Esta regla declara que $A(\mathbf{X})$ debe valer cuando la *fórmula* $B(\mathbf{Y})$ también vale.

Las reglas mencionadas en (b), las reglas locales, tienen la siguiente estructura:

$$\alpha(\mathbf{S})_{tc(\mathbf{T})} \overset{\Delta t}{\leftarrow} (nc_{edge}(\mathbf{X}), nc_{node}(\mathbf{Y}), if)$$

donde $\alpha \in \mathcal{P}_{nloc}$ ($\alpha(\mathbf{S})$ es el criterio de etiquetas locales objetivo), Δ_t es un número natural, *if* (función de influencia) es una función que determina la cantidad de influencia ejercida, $tc(\mathbf{T})$ (criterio de nodos objetivo) son conjunciones de fórmulas de red atómicas acerca de los nodos objetivos, $nc_{edge}(\mathbf{X})$, $nc_{node}(\mathbf{Y})$ (criterio de vecinos) son conjunciones de fórmulas de red atómicas acerca de arcos conectados al nodo objetivo y conjunciones de fórmulas de red atómicas acerca de lo nodos vecinos al nodo objetivo. Asimismo, $\mathbf{S}, \mathbf{T}, \mathbf{X}, \mathbf{Y}$ son secuencias de variables; en el caso particular que $\mathbf{S}, \mathbf{T}, \mathbf{X}, \mathbf{Y}$ sean todas vacías entonces se dice que la regla es básica (*ground*).

La cabeza de la regla involucra una fórmula basada en un único elemento de $\mathcal{P}_{nloc}$ que determina las etiquetas locales de nodo objetivo, lo cual esencialmente permite identificar las etiquetas que podrían ser modificadas. Más específicamente, la regla declara que cuando ciertas condiciones para un nodo y sus vecinos son satisfechas, la cota (*bnd*) cambia para los átomos de red formados con alguna etiqueta basada en $\alpha \in \mathcal{P}_{nloc}$ y que están vinculadas al nodo objetivo. Si la regla es aplicable en el tiempo t, entonces el cambio será hecho en el tiempo $t + \Delta_t$.

Las reglas mencionadas en (c), las reglas globales, tienen la siguiente estructura:

$$\beta(\mathbf{X}) \leftarrow (\alpha(\mathbf{Y})_{lt(\mathbf{Z})}, af)$$

donde $\beta \in \mathcal{P}_{glo}$, $\alpha \in \mathcal{P}_{nloc}$, $lt(\mathbf{Z})$ es una conjunción de fórmulas de red atómicas y *af* es una función de agregación. Además, $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ son secuencias de variables y cuando tales secuencias son todas vacías se dice que la regla es básica (*ground*).

Este tipo de reglas permite la obtención del estado de la red en una forma resumida, de manera que el valor de la cota (*bnd*) de las etiquetas globales formadas con $\beta \in \mathcal{P}_{glo}$ es el resultado de aplicar la función *af* a un conjunto de cotas. Cada una de estas cotas está asociada con alguna etiqueta local formada con $\alpha \in \mathcal{P}_{nloc}$ perteneciente a alguno de los nodos del grafo (el cual satisface el criterio $lt(\mathbf{Z})$).

Como es de esperarse, la unidad más general de conocimiento de red es una base de conocimiento NetDiff la cual contiene un conjunto de:

(1) Átomos básicos (*ground*) de la forma *node*/1 y *edge*/2 para representar la estructura del grafo. Por ejemplo: $node(n_1)$, $node(n_2)$, $edge(n_1, n_2)$.

(2) Etiquetas definidas en $\mathcal{L}$. Por ejemplo, *dangerous*(*luabot*).

(3) Un átomo básico (*ground*), de la forma *tmax*/1, el cual permanecerá fijo, para representar el valor t_{max}. Por ejemplo, *tmax*(2).

(4) Hechos NetDiff. Por ejemplo, F_1 en el Ejemplo 5.2 (abajo).

(5) Reglas NetDiff, ya sean restricciones de integridad, reglas locales o reglas globales. Por ejemplo, R_1 en el Ejemplo 5.2 (abajo).

Formalmente se dice que una base de conocimiento NetDiff es un par $DiffKB = (G, P)$ donde G (base de datos de red en la Figura 5.2) contiene los elementos de (1) a (4) y P solamente contiene elementos de (5).

Ejemplo 5.2 *Continuando con el Ejemplo 5.1, se definen los siguientes hechos y reglas NetDiff respectivamente :*

$$
\begin{aligned}
F_1 &= (node(n_1), \langle expert(ddos), [1{,}0, 1{,}0]\rangle) : [0, t_{max}] \\
F_2 &= (node(n_1), \langle expert(trojan), [0{,}9, 1{,}0]\rangle) : [0, t_{max}] \\
F_3 &= (node(n_2), \langle expert(ddos), [0{,}6, 1{,}0]\rangle) : [0, t_{max}] \\
F_4 &= (node(n_1), \langle bel\_dang(luabot), [0{,}7, 1{,}0]\rangle) : [0, t_{max}] \\
F_5 &= (node(n_1), \langle bel\_dang(gooligan), [0{,}8, 1{,}0]\rangle) : [0, t_{max}] \\
F_6 &= (edge(n_1, n_2), \langle close(trojan, windows\text{-}10), [0{,}8, 1{,}0]\rangle) : [0, t_{max}] \\
R_1 &= bel\_dang(luabot)_{\mathsf{T}} \overset{2}{\leftarrow} \mathsf{T}, \langle expert(ddos), [0{,}9, 1{,}0]\rangle \wedge \\
&\qquad \langle expert(trojan), [0{,}9, 1{,}0]\rangle \wedge \langle bel\_dang(luabot), [0{,}5, 1{,}0]\rangle, sftTp \\
R_2 &= bel\_dang(gooligan)_{\mathsf{T}} \overset{1}{\leftarrow} \langle close(trojan, windows\text{-}10), [0{,}7, 1{,}0]\rangle, \\
&\qquad \langle expert(trojan), [0{,}85, 1{,}0]\rangle \wedge \langle bel\_dang(gooligan), [0{,}5, 1{,}0]\rangle, sftTp \\
R_3 &= dangerous(luabot) \leftarrow \\
&\qquad bel\_dang(luabot)_{\langle expert(ddos),[0,9,1,0]\rangle \wedge \langle expert(trojan),[0,9,1,0]\rangle}, af_1 \\
R_4 &= dangerous(gooligan) \leftarrow bel\_dang(gooligan)_{\langle expert(trojan),[0,85,1,0]\rangle}, af_2
\end{aligned}
$$

$$
sftTp(V', v, NS, L) = \begin{cases} [0{,}7, 1{,}0] & si\ |V'|\,/\,|neigh(v)| > 0{,}5 \\ [0{,}0, 1{,}0] & en\ otro\ caso \end{cases}
$$

$$
\begin{aligned}
af_1(B) &= [l, u]\ donde:\ \ l = \sum_{[l_i,u_i]\in B}^{|B|} \frac{l_i}{|B|} \quad y \quad u = \sum_{[l_i,u_i]\in B}^{|B|} \frac{u_i}{|B|} \\
af_2(B) &= [l, u]\ donde:\ \ [l, u] \in B\ y\ (l+u)/2 = max\{(l_i + u_i)/2\ tal\ que\ [l_i, u_i] \in B\}
\end{aligned}
$$

Los hechos 1 *y* 2 *declaran que el nodo* n_1 *es un experto en ataques ddos con la más alta confianza y es un experto en ataques troyanos con una confianza de al menos* 0,9*, luego los hechos* 4 *y* 5 *declaran que este nodo cree que luabot y gooligan son peligrosos con una confianza de al menos* 0,7 *y* 0,8*, respectivamente. Asimismo, el hecho* 3 *describe al nodo* n_2 *como un experto en ataques ddos con una confianza de al menos* 0,6 *y el hecho* 6 *declara que hay una conexión entre los nodos* n_1 *y* n_2 *de manera que puede decirse que tales nodos son cercanos (close) respecto a troyanos en Windows 10 con una confianza de al menos* 0,8*.*

Por otro lado, la regla R_1 *declara que un nodo cree en que luabot es peligroso con un peso de al menos 0.7 (esto es especificado en la función de influencia sftTp) si al menos la mitad de sus vecinos (también especificado en sftTp) son expertos en ataques ddos y troyanos con una confianza de al menos* 0,9 *y ya creían que luabot era peligroso con un peso de al menos* 0,5 *dos pasos de tiempo atrás. La regla* R_2 *puede ser leída análogamente. Finalmente, las reglas* R_3 *y* R_4 *son globales y pueden ser leídas similarmente; por lo tanto, solo se describe intuitivamente* R_3*, la cual declara que la confianza de la red respecto al peligro de luabot es actualizada aplicando la función* af_1 *a un conjunto de creencias locales sobre el peligro de luabot, lo que a su vez se basa en los nodos que son expertos en ataques troyanos y ddos con una confianza de al menos* 0,9*.* ∎

Uno de los principales intereses de la implementación del módulo NDM es representar el conocimiento de redes y esto lo podemos hacer usando bases de conocimiento NetDiff. Sin embargo, como queremos razonar sobre redes complejas debemos tener la capacidad de responder consultas sobre la red; para lo cual se necesita saber la forma de las consultas NetDiff booleanas y ésto puede describirse de la siguiente manera:

$$\Big(\underbrace{\bigwedge_{i=0\ldots r} (v_i, \langle L_i, bnd_i \rangle)}_{(a)} \wedge \underbrace{\bigwedge_{j=0\ldots s} (e_j, \langle L_j, bnd_j \rangle)}_{(b)} \wedge \underbrace{\bigwedge_{k=0\ldots t} (\Omega, \langle L_k, bnd_k \rangle)}_{(c)} \Big) : \underbrace{[t_1, t_2]}_{(d)}$$

donde $v_i \in V$, $e_j \in E$, $L_i \in \mathcal{L}_{nloc}$, $L_j \in \mathcal{L}_{eloc}$, $L_k \in \mathcal{L}_{glo}$ y $[t_1, t_2] \subseteq [0, t_{max}]$.

La parte (a) de la consulta es una conjunción que pregunta por la satisfacción de los átomos de red (etiqueta con cota) en diferentes nodos (v_i), la parte (b) es una conjunción que pregunta por la satisfacción de los átomos de red en diferentes arcos (e_j), y la parte (c) es una conjunción que pregunta por la satisfacción de los átomos de red en toda la red (Ω). Además, la parte (d) determina los tiempos en los que se deben cumplir (a), (b) y (c).

Algorithm 1 Difusión NetDiff

Require: Base de conocimiento NetDiff $DiffKB = (G, P)$

1: ans:= $\emptyset$
2: //cada etiqueta es asociada con una cota $[0, 1]$
3: net_state_{prev}:= NetDiff *Estado Inicial*
4: net_state_{now}:= NetDiff *Estado Inicial*
5: **for** hecho in G **do**
6: //el estado NetDiff es actualizado para asegurar cada hecho NetDiff
7: net_state_{now}:= $apply(net\_state_{now}$, hecho)
8: **end for**
9: **for** r in P **do**
10: //el estado NetDiff es actualizado aplicando una vez cada regla NetDiff r
11: net_state_{now}:= $apply(net\_state_{now}, r)$
12: **end for**
13: **while** net_state_{prev} != net_state_{now} **do**
14: //while loop que implementa un operador de punto fijo
15: net_state_{prev} := net_state_{now}
16: **for** r in P **do**
17: //el estado NetDiff es actualizado aplicando una vez cada regla NetDiff r
18: net_state_{now}:= $apply(net\_state_{now}, r)$
19: **end for**
20: **end while**
21: **return** net_state_{now}

El Algoritmo 1 muestra el proceso de difusión NetDiff para una base de conocimiento NetDiff $DiffKB = (G, P)$. El proceso solo termina cuando se aplican las reglas y no hay cambios en el estado NetDiff. Cuando se habla de estados NetDiff se hace referencia a una estructura de datos, la cual asigna puntos de tiempo a nodos, arcos o Ω (la red completa), y luego éstos son mapeados a átomos de red (etiquetas con cotas). Aquí, se considera el estado NetDiff inicial como el estado donde la incertidumbre es máxima (cada etiqueta está asociada con la cota $[0, 1]$). La salida del Algoritmo 1 es el estado NetDiff final que se obtiene cuando se alcanza un punto fijo (su implementación podría realizarse de manera análoga a [SSS13, SSC13]).

Ahora, consideramos una consulta NetDiff Q y una base de conocimiento $DiffKB =$

(G, P); si *net_state* es el estado NetDiff obtenido del Algoritmo 1 sobre *DiffKB*, entonces la consulta Q se puede responder a partir de *net_state* y la siguiente función:

$$checkCondInState(cond, net\_state) \tag{5.1}$$

La función $checkCondInState(cond, net\_state)$ está a cargo de verificar las condiciones (átomos de red en *cond*) para cada nodo, arco y Ω en cada tiempo definido sobre un estado NetDiff (*net_state*) (esto se hará más claro en el Ejemplo 5.3). Para responder a la consulta, Q se evalúa sobre *net_state* invocando la función $checkCondInState(Q, net\_state)$, que debe verificar las condiciones especificadas en la consulta Q sobre el estado final NetDiff *net_state* (estado de salida del Algoritmo 1). Tenga en cuenta que la terminación del Algoritmo 1 depende de la evolución de los estados NetDiff, cuestión que más adelante analizaremos en el contexto del Teorema 5.1.

Ejemplo 5.3 *Considere las etiquetas* $\mathcal{L} = \mathcal{L}_{nloc} \cup \mathcal{L}_{eloc} \cup \mathcal{L}_{glo}$, $\mathcal{L}_{nloc} = \{expert(trojan), bel\_dang(gooligan)\}$, $\mathcal{L}_{eloc} = \{close(trojan, windows\text{-}10)\}$, $\mathcal{L}_{glo} = \{dangerous(gooligan)\}$, *una base de conocimiento NetDiff* $DiffKB = (G, P)$ *donde* $G = \{node(n_1), node(n_2), edge(n_1, n_2)\} \cup \mathcal{L} \cup \{tmax(2)\} \cup \{F_2, F_5, F_6\}$ *y* $P = \{R_2, R_4\}$ *obtenido de una simplificación del Ejemplo 5.1 y Ejemplo 5.2. Considere las siguientes consultas NetDiff:*

$$\begin{aligned}
Q_1 = \;& (node(n_1), \langle expert(trojan), [0{,}5, 1{,}0]\rangle) \wedge \\
& (node(n_1), \langle bel_dang(gooligan), [0{,}8, 1{,}0]\rangle) : [0, 2] \\
Q_2 = \;& (node(n_1), \langle expert(trojan), [0{,}6, 1{,}0]\rangle) \wedge \\
& (edge(n_1, n_2), \langle close(trojan, windows\text{-}10), [0{,}8, 1{,}0]\rangle) : [0, 2] \\
Q_3 = \;& (node(n_2), \langle bel_dang(gooligan), [0{,}6, 1{,}0]\rangle) : [0, 0] \\
Q_4 = \;& (node(n_2), \langle bel_dang(gooligan), [0{,}6, 1{,}0]\rangle) : [2, 2]
\end{aligned}$$

Por una cuestión de conveniencia sintáctica, a veces las consultas NetDiff se escribirán en forma resumida para evitar la repetición de nodos, arcos o Ω*; por lo tanto, podríamos escribir* Q_1 *como:*

$$Q_1 = (node(n_1), \langle expert(trojan), [0{,}5, 1{,}0]\rangle \wedge \langle bel\_dang(gooligan), [0{,}8, 1{,}0]\rangle) : [0, 2]$$

En esta variante de Q_1*, se hace referencia a* $node(n_1)$ *una vez en lugar de dos.*

Para responder a las consultas, el Algoritmo 1 comienza desde el estado NetDiff con máxima incertidumbre, donde cada etiqueta tiene una cota $[0, 1]$ *(parte superior en la*

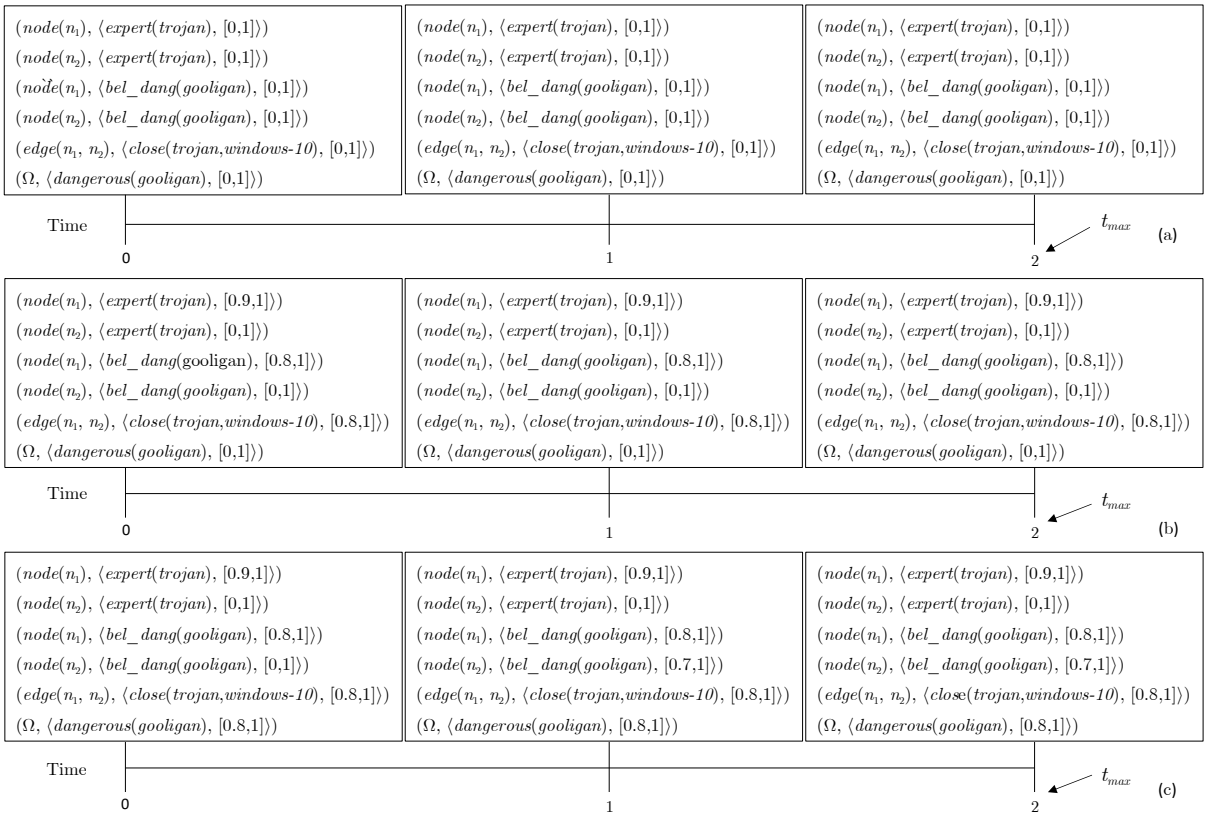

Figura 5.1: Evolución de la red para el proceso de difusión NetDiff (cf. Algoritmo 1) planteado en el Ejemplo 5.3. (a) Estado inicial de la red; (b) Estado de la red después de aplicar los hechos NetDiff F_2, F_5 y F_6; (c) Estado de la red después de aplicar las reglas R_2 y R_4.

Figura 5.1). A continuación, se actualiza el estado NetDiff para asegurar que los hechos F_2, F_5, F_6 *valgan (parte media de la Figura 5.1) y se obtiene un nuevo estado NetDiff. Luego, el estado NetDiff se actualiza aplicando cada regla una vez (parte inferior de la Figura 5.1) y se obtiene un nuevo estado NetDiff que en este caso es el punto fijo (este operador podría ser implementado de manera análoga a lo hecho en [SSS13, SSC13]). Finalmente, las consultas se pueden responder usando el estado final (parte inferior de la Figura 5.1) y la función checkCondInState*$(cond, net\_state)$ *(Función 5.1): la respuesta para la consulta* Q_1 *es "sí" porque (a) la etiqueta* $expert(trojan)$ *tiene la cota* $[0{,}9, 1{,}0]$*, que es un subconjunto de* $[0{,}5, 1{,}0]$ *(especificado en la consulta), para* $node(n_1)$ *en cada tiempo* $t \in [0, 2]$*, y (b) la etiqueta* $bel\_dang(gooligan)$ *tiene la cota* $[0{,}8, 1{,}0]$*, que es la misma cota*

especificada en la consulta, para node(n_1) en cada tiempo $t \in [0, 2]$. De manera similar, las otras consultas se pueden resolver, y luego tenemos que la respuesta para la consulta Q_2 es "Sí", la respuesta para la consulta Q_3 es "No" y la respuesta para la consulta Q_4 es "Sí". Tenga en cuenta que la única diferencia entre las consultas Q_3 y Q_4 es que se refieren a tiempos diferentes, razón por la cual la respuesta cambia. ∎

Ahora, se analizarán las condiciones bajo las cuales el Algoritmo 1 termina. Aquí, las bases de conocimiento NetDiff son generalizaciones de programas MANCaLog definidos en [SSS13, SSC13], que tienen garantías de alcanzar un punto fijo en un tiempo finito porque cada función de influencia solo depende de la relación entre los vecinos totales y una fracción de éstos que satisfacen un determinado criterio (condición (a) del Teorema 5.1). Aunque las bases de conocimiento NetDiff tienen un nuevo tipo de regla que son las reglas globales, su aplicación depende de las etiquetas locales y no de las etiquetas globales; por lo tanto, si se alcanza un punto fijo para las etiquetas locales, entonces se alcanza un punto fijo para las etiquetas globales. Por lo tanto, podemos ignorar las reglas globales para este análisis. Asimismo, las bases de conocimiento NetDiff utilizan variables pero pueden transformarse en bases de conocimiento básicas (*ground*, sin variables) porque se sabe qué etiqueta está disponible; y estas bases de conocimiento básicas (ignorando las reglas globales) son equivalentes a los programas MANCaLog. En este caso, la única diferencia es que las bases de conocimiento NetDiff pueden definir funciones de influencia más generales, pero si la condición (a) del Teorema 5.1 se cumple entonces está garantizada la terminación del Algoritmo 1. De lo contrario, se introducen condiciones adicionales para obtener garantías de terminación:

Teorema 5.1 *Dada una base de conocimiento NetDiff $DiffKB = (G, P)$, el Algoritmo 1 termina si se cumple una de las siguientes condiciones:*

a) Cada regla local en P solo contiene funciones de influencia que dependen de la relación entre los vecinos totales (elegibles) y una fracción de éstos que satisfacen determinado criterio (calificados).

b) El número de bits usados para representar los intervalos de confianza asignados a cada etiqueta está acotado por una constante b.

c) Cada regla local en P usa un valor de $\Delta t \neq 0$.

Además, el costo computacional del Algoritmo 1 depende de la condición específica que se cumpla:

- *Condición (a):* $O\Big(|DiffKB| \cdot t_{max} \cdot |V| \cdot (IF_{time} \cdot d_{\Omega}^{in} + |\mathcal{L}_{nl}| \cdot |\mathcal{L}_{el}| \cdot d_{\Omega}^{in^2})\Big)$.
- *Condición (b):* $O\Big(|DiffKB| \cdot t_{max} \cdot |V| \cdot (IF_{time} + |\mathcal{L}_{el}| \cdot |\mathcal{L}_{nl}| \cdot d_{\Omega}^{in}) \cdot 2^{2b}\Big)$.
- *Condición (c):* $O\Big(|V| \cdot (IF_{time} + |\mathcal{L}_{el}| \cdot |\mathcal{L}_{nl}| \cdot d_{\Omega}^{in}) \cdot |DiffKB|^{t_{max}}\Big)$.

En cada caso, d_{Ω}^{in} *es el máximo grado de arcos incidentes en la red e* IF_{time} *es el costo en el peor caso asociado con cualquier función de influencia usada en reglas locales de* P[1]. *Prueba en el Apéndice, Sección 9.2.*

El siguiente corolario es una consecuencia directa de las Afirmaciones 6, 7 y 8 de la prueba del Teorema 5.1 en el Apéndice, Sección 9.2.

Corolario 5.1 *Dada una base de conocimiento NetDiff* $DiffKB = (G, P)$*, si el tiempo requerido para aplicar cada función de influencia de las reglas locales en* P *está acotado por un polinomio, y se cumple una de las condiciones (a), (b) o (c) del Teorema 5.1, entonces el Algoritmo 1 para DiffKB termina en tiempo polinomial en complejidad de datos (solo cambia la red compleja).*

La siguiente observación simplemente declara que la convergencia no siempre está garantizada.

Observación 5.1 *Existe una base de conocimiento NetDiff* $DiffKB = (G, P)$ *tal que el Algoritmo 1 para DiffKB nunca termina.*

El siguiente ejemplo es suficiente para mostrar que la observación es verdadera.

Ejemplo 5.4 *Se supone que se tienen los siguientes elementos:*

- *Tres nodos:* $node(n_1)$, $node(n_2)$ *y* $node(n_3)$.

[1]Observe que para bases de conocimiento NetDiff no básicas (*non-ground*), $|DiffKB|$ depende del tamaño de la base de conocimiento básica (*ground*) obtenida a partir de *DiffKB*.

Paso	*Tiempo*	*nodos*	*L*
0	0	$node(n_1)$	[0,7, 1,0]
0	0	$node(n_2)$	[0,7, 1,0]
0	0	$node(n_3)$	[1,0, 1,0]
1	0	$node(n_1)$	[0,85, 1,0]
1	0	$node(n_2)$	[0,85, 1,0]
1	0	$node(n_3)$	[1,0, 1,0]
2	0	$node(n_1)$	[0,925, 1,0]
2	0	$node(n_2)$	[0,925, 1,0]
2	0	$node(n_3)$	[1,0, 1,0]
...	...	...	...

Figura 5.2: Evolución de la red a medida que la única regla local NetDiff es aplicada cuando el Algoritmo 1 se ejecuta sobre la base de conocimiento $DiffKB = (G, P)$ del Ejemplo 5.4. Los arcos no son incluidos debido a que no hay etiquetas locales de arco definidas.

- *Seis arcos:* $edge(n_1, n_2)$, $edge(n_2, n_1)$, $edge(n_1, n_3)$, $edge(n_3, n_1)$, $edge(n_2, n_3)$, *y* $edge(n_3, n_2)$.
- *Una única etiqueta local de nodo L, es decir,* $\mathcal{L} = \mathcal{L}_{nloc} \cup \mathcal{L}_{eloc} \cup \mathcal{L}_{glo}$ *y* $\mathcal{L}_{nloc} = \{L\}$, $\mathcal{L}_{eloc} = \{\}$, $\mathcal{L}_{glo} = \{\}$.
- *Una única función de influencia:* $if_1(V', v, net\_state, L, t) = [l_1, u_1]$ *donde:*

$$l_1 = \sum_{condition} l_i/|V'| \;\; y \;\; u_1 = \sum_{condition} u_i/|V'|$$

y

$$condition = \; \text{la respuesta de } checkCondInState\Big((v', \langle L, [l', u']\rangle) : [t, t], net\_state\Big) \text{ es “Sí” y } v' \in V'.$$

“condition” declara que la salida de if_1 *es obtenida a partir de las cotas de la etiqueta L (objetivo en la regla aplicada) para vecinos del nodo objetivo en el tiempo t (en el cual la regla local NetDiff asociada es aplicada).*

- *Tres hechos* NetDiff *y una regla local* NetDiff*:*

$$F_1 : (node(n_1), \langle L, [0{,}7, 1{,}0] \rangle) : [0, t_{max}]$$
$$F_2 : (node(n_2), \langle L, [0{,}7, 1{,}0] \rangle) : [0, t_{max}]$$
$$F_3 : (node(n_3), \langle L, [1{,}0, 1{,}0] \rangle) : [0, t_{max}]$$
$$R_1 : L_{\mathcal{T}} \overset{0}{\leftarrow} (\mathcal{T}, \langle L, [0{,}5, 1{,}0] \rangle, if_1)$$

En este contexto se define la siguiente base de conocimiento NetDiff *DiffKB* $= (G, P)$ *donde*

$$\begin{aligned} G = \; & \{node(n_1), node(n_2), node(n_3), \\ & edge(n_1, n_2), edge(n_2, n_1), edge(n_1, n_3), edge(n_3, n_1), edge(n_2, n_3), edge(n_3, n_2)\} \cup \\ & \mathcal{L} \cup \{tmax(0)\} \cup \{F_1, F_2, F_3\} \\ P = \; & \{R_1\} \end{aligned}$$

La Figura 5.2 muestra la evolución de los estados NetDiff *a medida que se ejecuta el Algoritmo 1 sobre DiffKB* $= (G, P)$*. Observe que la red nunca alcanzará un punto fijo en el tiempo* 0 *porque los intervalos para los átomos de red formados con* L *en el nodo* $node(n_1)$ *y el nodo* $node(n_2)$ *siempre pueden reducirse cuando se aplica la función de influencia. La razón es que la función de influencia toma un promedio de los límites superior e inferior de los intervalos de los átomos de red formados con* L *para cada vecino del nodo objetivo en la regla aplicada y, por lo tanto, el proceso nunca termina. Entonces la ejecución del Algoritmo 1 sobre la base de conocimiento DiffKB* $= (G, P)$ *nunca termina.* ∎

Ahora se pasa a discutir los fundamentos teóricos relativos al Módulo de Razonamiento Ontológico.

5.2. Modelado Ontológico

Ahora se presenta el lenguaje ontológico NETDER, el cual es una extensión de la conocida familia de lenguajes ontológicos nombrados como reglas existenciales (también llamado *Datalog+/–*); como se mostrará más adelante, la extensión es sólo sintáctica por conveniencia de representación. Después de presentar su sintaxis y semántica, desarrollamos un procedimiento *chase* modificado para responder consultas, el cual será muy útil y es el tema de la siguiente sección.

5.2.1. Sintaxis

A continuación, se muestran las unidades básicas de sintaxis del lenguaje y cómo permiten combinar conocimiento ontológico clásico y de red. Sea $DiffKB = (G, P)$ una base de conocimiento NetDiff con un conjunto finito de etiquetas $\mathcal{L} = \mathcal{L}_{glo} \cup \mathcal{L}_{nloc} \cup \mathcal{L}_{eloc}$ donde $\mathcal{L}_{glo}$ es un conjunto de etiquetas globales, $\mathcal{L}_{nloc}$ es un conjunto de etiquetas locales de nodo y $\mathcal{L}_{eloc}$ es un conjunto de etiquetas locales de arco; entonces se definen los siguientes elementos del lenguaje:

Definición 5.1 (Anotación local de red) *Una anotación local de red $\lambda(\mathbf{U})$, es una conjunción (finita) de pares $\langle A_1, bnd_1\rangle \wedge ... \wedge \langle A_k, bnd_k\rangle$ donde cada $A_{1\leq i\leq k}$ es un átomo sobre $\mathcal{R}$, $\mathbf{U}$, y Δ, $\mathbf{U}$ es una secuencia de variables tales que $\mathbf{U} \subseteq \mathbf{V}$, cada $bnd_{1\leq i\leq k} \subseteq [0, 1]$, y existe una substitución σ tal que $\sigma A_{1\leq i\leq k} \in \mathcal{L}_{eloc} \cup \mathcal{L}_{nloc}$.*

Definición 5.2 (Anotación global de red) *Una anotación global de red $\gamma(\mathbf{U})$ es una conjunción (finita) de pares $\langle B_1, bnd_1\rangle \wedge ... \wedge \langle B_k, bnd_k\rangle$ donde $B_{1\leq i\leq k}$ es un átomo sobre $\mathcal{R}$, $\mathbf{U}$, y Δ, $\mathbf{U}$ es una secuencia de variables tal que $\mathbf{U} \subseteq \mathbf{V}$, $bnd_{1\leq i\leq k} \subseteq [0, 1]$, y existe una substitución σ tal que $\sigma B_{1\leq i\leq k} \in \mathcal{L}_{glo}$.*

Definición 5.3 (Componente objetivo de red) *Un componente objetivo de red es un par $(c, \lambda(\mathbf{U}))$ donde c es un átomo sobre $\mathcal{R}$, $\mathbf{V}$, y Δ, $\lambda(\mathbf{U})$ es una anotación local de red, y existe una substitución σ tal que $\sigma c \in \mathcal{G}$.*

Estos elementos se ilustrarán en un ejemplo más adelante. Ahora tenemos los elementos necesarios para definir el primer tipo de regla NETDER.

Definición 5.4 (Dependencia de generación de tupla NETDER) *Una dependencia de generación de tupla (TGD) NETDER es una fórmula de la forma:*

$$\forall \mathbf{XYQSU}\ \Upsilon(\mathbf{X}) \wedge \Phi(\mathbf{Y}) \rightarrow \exists \mathbf{Z}\ \Psi(\mathbf{Q}, \mathbf{R})\ \wedge \Xi(\mathbf{S}, \mathbf{T}) : \gamma(\mathbf{U})$$

donde $\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{Q}, \mathbf{R}, \mathbf{S}, \mathbf{T}, \mathbf{U}$ son secuencias de variables, $\mathbf{Q} \cup \mathbf{S} \subseteq \mathbf{X} \cup \mathbf{Y} \cup \mathbf{U}$, $\mathbf{R} \cup \mathbf{T} = \mathbf{Z}$, $\Upsilon(\mathbf{X})$ y $\Psi(\mathbf{Q}, \mathbf{R})$ son conjunciones de átomos, $\Phi(\mathbf{Y})$ y $\Xi(\mathbf{S}, \mathbf{T})$ son conjunciones de componentes objetivos de red, y $\gamma(\mathbf{U})$ es una anotación global de red.

Las EGDs y las NCs NETDER son una extensión análoga.

Definición 5.5 (Dependencia de generación de igualdad NetDER) *Una dependencia de generación de igualdad (EGD)* NetDER *es una fórmula de la forma:*

$$\forall \mathbf{STU}\ \Upsilon(\mathbf{S}) \wedge \Phi(\mathbf{T}) \rightarrow (X_i = X_j) : \gamma(\mathbf{U})$$

donde $\mathbf{S}, \mathbf{T}, \mathbf{U}$ *son secuencias de variables,* $X_i, X_j \in \mathbf{S} \cup \mathbf{T} \cup \mathbf{U}$*,* $\Upsilon(\mathbf{S})$ *es una conjunción de átomos,* $\Phi(\mathbf{T})$ *es una conjunción de componentes objetivos de red, y* $\gamma(\mathbf{U})$ *es una anotación global de red.*

Definición 5.6 (Restricción negativa NetDER) *Una restricción negativa (NC)* NetDER *es una fórmula de la forma:*

$$\forall \mathbf{STU}\ \Upsilon(\mathbf{S}) \wedge \Phi(\mathbf{T}) \rightarrow \bot : \gamma(\mathbf{U})$$

donde $\mathbf{S}, \mathbf{T}, \mathbf{U}$ *son secuencias de variables,* $\Upsilon(\mathbf{S})$ *es una conjunción de átomos,* $\Phi(\mathbf{T})$ *es una conjunción de componentes objetivos de red, y* $\gamma(\mathbf{U})$ *es una anotación global de red.*

En estas fórmulas, a veces se omiten los cuantificadores universales. A continuación, se definen las bases de conocimiento NetDER las cuales se componen de elementos tanto del ORM como del NDM.

Definición 5.7 (Base de conocimiento NetDER) *Una base de conocimiento (KB)* NetDER *es una tupla* $KB = (D, G, \Sigma, P)$*, donde* D *es un conjunto finito de átomos con predicados de un esquema relacional* $\mathcal{R}$ *y argumentos de un conjunto de constantes* Δ*,* Σ *es un conjunto de* NetDER *TGDs, EGDs, y NCs sobre* $\mathcal{R}$*, y* $DiffKB = (G, P)$ *es una base de conocimiento* NetDiff *(cf. Página 99).*

Ejemplo 5.5 *Continuando con el Ejemplo 5.3, se muestra una instancia de una base de conocimiento* NetDER $KB = (D, G', \Sigma, P)$ *donde* G' *contiene los mismos elementos de* G *del Ejemplo 5.3 más otro conocimiento adicional y* P *contiene las mismas reglas* NetDiff *del Ejemplo 5.3. También se definen las siguientes etiquetas locales: (i) bot, la cual es una etiqueta local de nodo que permite representar un actor bot; (ii) similar, la cual es una etiqueta local de arco que permite representar dos actores similares, y (iii) botnet, la cual es un etiqueta local arco que permite representar dos actores en la misma botnet.* D *contiene el siguiente conjunto de átomos:*

$$D = \{d_1 : user(n_1, ponzio),\ d_2 : earlyPoster(n_1, gooligan),\ d_3 : product(windows\text{-}10)\}$$

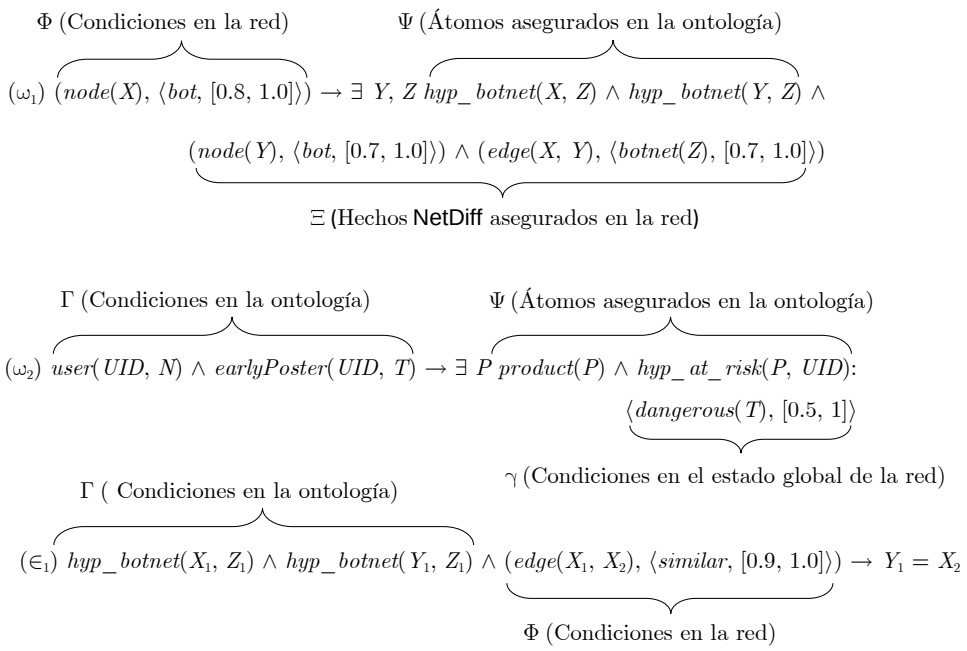

Figura 5.3: Ilustración de una instanciación de dos TGDs NETDER y una EGD NETDER (los nombres de las subfórmulas se mantienen de acuerdo a como aparecen en las Definiciones 5.4 y 5.5).

G' es una extensión de G del Ejemplo 5.3 y contiene los siguientes elementos:

$$\begin{aligned} G' = \; G \; \cup \; \{ \;\; & g_1 : node(n_3),\ g_2 : edge(n_2, n_3), \\ & g_3 : bot,\ g_4 : botnet,\ g_5 : similar \\ & g_6 : (node(n_1), \langle bot, [0{,}9, 1{,}0] \rangle) : [0, t_{max}] \\ & g_7 : (node(n_2), \langle bot, [0{,}6, 1{,}0] \rangle) : [0, t_{max}] \\ & g_8 : (edge(n_2, n_3), \langle botnet, [0{,}75, 1{,}0] \rangle) : [0, t_{max}], \\ & g_9 : (edge(n_1, n_2), \langle similar, [0{,}9, 1{,}0] \rangle) : [0, t_{max}] \\ & \} \end{aligned}$$

Σ contiene el siguiente conjunto de TGDs y EGDs NETDER *:*

$$\begin{aligned} (\omega_1)\ (node(X), \langle bot, [0{,}8, 1{,}0] \rangle) \rightarrow \; & \exists Y, Z\ (node(Y), \langle bot, [0{,}7, 1{,}0] \rangle) \wedge \\ & (edge(X, Y), \langle botnet, [0{,}7, 1{,}0] \rangle) \wedge \\ & hyp\_botnet(X, Z) \ \wedge \ hyp\_botnet(Y, Z) \end{aligned}$$

$$\begin{aligned} & (\omega_2)\ user(UID, N) \ \wedge \ earlyPoster(UID, T) \\ & \quad \rightarrow \exists U\ product(U) \ \wedge \ hyp\_at\_risk(U, UID) : \langle dangerous(T), [0{,}5, 1{,}0] \rangle \end{aligned}$$

(ω_3) $(node(X), \langle expert(S), [0{,}9, 1{,}0]\rangle \wedge \langle bel\_dang(T), [0{,}8, 1{,}0]\rangle)$
$\rightarrow \exists U, UID, N\ product(U) \ \wedge \ user(UID, N) \ \wedge \ hyp\_at\_risk(U, UID)$

(ϵ_1) $hyp\_botnet(X_1, Z_1) \ \wedge \ hyp\_botnet(Y_1, Z_1) \ \wedge \ (edge(X_1, X_2), \langle similar, [0{,}9, 1{,}0]\rangle)$
$\rightarrow Y_1 = X_2$

(ϵ_2) $hyp\_botnet(X_1, Z_1) \wedge hyp\_botnet(Y_1, Z_1) \wedge (edge(Y_1, Y_2), \langle botnet, [0{,}7, 1{,}0]\rangle)$
$\rightarrow Z_1 = Z_2$

Finalmente, P contiene el mismo conjunto de reglas NetDiff del Ejemplo 5.3. ∎

5.2.2. Semántica

La semántica del lenguaje ontológico NetDER se definirá de forma operativa. Por lo tanto, ahora el foco estará en extender el conocido procedimiento *chase* para trabajar con reglas NetDER, lo cual será esencial para calcular las respuestas a las consultas.

Un chase modificado

El procedimiento *chase* es modificado para considerar los componentes objetivos de red en las TGDs y EGDs, y las anotaciones globales de red en las fórmulas cuando se obtienen las consecuencias derivadas de la ontología.

Regla del chase para TGDs NetDER. Considere una base de conocimiento NetDER $KB = (D, G, \Sigma, P)$, un período de tiempo $[t_1, t_2] \subseteq [0, t_{max}]$ y una TGD NetDER $\sigma \in \Sigma$ de la forma $\Upsilon(\mathbf{X}) \wedge \Phi(\mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{Q}, \mathbf{R}) \wedge \Xi(\mathbf{S}, \mathbf{T}) : \gamma_\sigma(\mathbf{U})$, donde $\mathbf{Q} \cup \mathbf{S} \subseteq \mathbf{X} \cup \mathbf{Y} \cup \mathbf{U}$, $\mathbf{R} \cup \mathbf{T} = \mathbf{Z}$, $\Upsilon(\mathbf{X})$ y $\Psi(\mathbf{Q}, \mathbf{R})$ son conjunciones de átomos, $\Phi(\mathbf{Y})$ y $\Xi(\mathbf{S}, \mathbf{T})$ son conjunciones de componentes objetivos de red, y $\gamma_\sigma(\mathbf{U})$ es una anotación global de red. Entonces, σ es aplicable a (D, G) en el período de tiempo $[t_1, t_2]$ si y sólo si:

- existe un homomorfismo h que mapea átomos de $\Upsilon(\mathbf{X})$ tal que $h(\Upsilon(\mathbf{X})) \subseteq D$,
- existe un homomorfismo h_1 que extiende h tal que la respuesta de la función *checkCondInState*(Q_1, *net_state*) (Función 5.1, Página 103) es "Sí", donde $Q_1 =$

$(h_1(\Phi(\mathbf{Y}))) : [t_1, t_2]$ y *net_state* es la salida del Algoritmo 1 sobre la base de conocimiento NetDiff $DiffKB = (G, \emptyset)$[2] (obtener estado actual de la red en la Figura 4.1), y

- la respuesta de la función *checkCondInState*(Q_2, *net_state*) es "Sí", donde $Q_2 = (\Omega, h_1(\gamma_\sigma(\mathbf{U}))) : [t_1, t_2]$ y *net_state* es el mismo que el mencionado arriba.

Sea σ aplicable a (D, G) en el período de tiempo $[t_1, t_2]$, y sea h_2 un homomorfismo que extiende h_1 de la siguiente manera: por cada $Q_i \in \mathbf{Q} \cup \mathbf{S}$, $h_2(Q_i) = h_1(Q_i)$; para cada $Z_j \in \mathbf{Z}$, $h_1(Z_j) = z_j$, donde $z_j \in \Delta_N$ es un nuevo valor nulo, z_j no ocurre en $D \cup G$, y z_j sigue lexicográficamente todos los demás valores nulos ya introducidos. Cuando se aplica σ en (D, G) entonces:

- Los átomos $\alpha_1, ..., \alpha_m$ se agregan a D donde $h_2(\Psi(\mathbf{Q}, \mathbf{R})) = \alpha_1 \wedge ... \wedge \alpha_m$, si ya no está en D,
- Si $\Xi(\mathbf{S}, \mathbf{T}) = \Big(c_1(X_1), na_1(\mathbf{Y}_1)\Big) \wedge, ..., \wedge \Big(c_{m'}(X_{m'}), na_{m'}(\mathbf{Y}_{m'})\Big)$ y la respuesta de la función *checkCondInState*($Q_{1 \leq i \leq n}$, *net_state*) es "no", donde
  - $Q_{1 \leq i \leq n} = \Big(c'_1, na'_1\Big) \wedge, ..., \wedge \Big(c'_{m'}, na'_{m'}\Big)$,
  - $c'_1 = h_2\Big(c_1(X_1)\Big), ..., c'_{m'} = h_2\Big(c_{m'}(X_{m'})\Big)$,
  - $na'_1 = h_1\Big(na_1(\mathbf{Y}_1)\Big), ..., na'_{m'} = h_1\Big(na_{m'}(\mathbf{Y}_{m'})\Big)$[3], y
  - *net_state* es la salida del Algoritmo 1 sobre la base de conocimiento NetDiff $DiffKB = (G, \emptyset)$.

  Entonces,

  - Los átomos $c'_1, ..., c'_{m'}$ se agregan a G, si aún no estaban en G,
  - Los hechos NetDiff $(c'_1, na'_1) : [t_1, t_2], ..., (c'_{m'}, na'_{m'}) : [t_1, t_2]$ se agregan a G[4].

[2] En este caso, la base de conocimiento NetDiff no tiene reglas porque solo se necesita el estado a partir de los hechos NetDiff de G.

[3] Se utiliza h_1 porque se busca evitar crear nuevas etiquetas con valores nulos.

[4] $(c, na_1 \wedge ... \wedge na_k) : [t_1, t_2]$ es una forma resumida de escribir los hechos NetDiff $(c, na_1) : [t_1, t_2], ..., (c, na_k) : [t_1, t_2]$.

El *chase* de nivel hasta 0 de (D, G) en el período de tiempo $[t_1, t_2] \subseteq [0, t_{max}]$ relativo al conjunto de TGDs NetDER Σ, denotado $chase^0_{[t_1,t_2]}(D, G)$, se define asignando a cada elemento en D y G el nivel (derivación) 0. Para cada $k \geq 1$, el *chase* de nivel hasta k de (D, G) en el período de tiempo $[t_1, t_2] \subseteq [0, t_{max}]$ relativo al conjunto de TGDs NetDER Σ, denotado $chase^k_{[t_1,t_2]}(D, G) = (D^k, G^k)$, es construido de la siguiente forma: Sean $(I_1, I'_1, I''_1), ..., (I_n, I'_n, I''_n)$ las n posibles imágenes del cuerpo (ambas partes) y anotaciones globales de red de TGDs NetDER en Σ donde I_i son imágenes de la conjunción de átomos en el cuerpo, I'_i son imágenes de la conjunción de los componentes objetivos de red en el cuerpo y I''_i son imágenes de las anotaciones globales de red, tal que:

- $I_1, ..., I_n \subseteq D^{k-1}$ para algún homomorfismo h,
- la respuesta de la función *checkCondInState*($Q_{1\leq i\leq n}$, *net_state*) es "Sí" para algún otro homomorfismo h_1 que extiende h, donde $Q_{1\leq i\leq n} = (I'_i) : [t_1, t_2]$ y *net_state* es la salida del Algoritmo 1 sobre la base de conocimiento NetDiff $DiffKB = (G^{k-1}, \emptyset)$, y
- la respuesta de la función *checkCondInState*($Q_{1\leq i\leq n}$, *net_state*) es "Sí" utilizando h_1, donde $Q_{1\leq i\leq n} = (\Omega, I''_i) : [t_1, t_2]$ y *net_state* es el mismo que el mencionado arriba.

Entonces, se puede realizar la aplicación de cada TGD NetDER correspondiente, eligiendo las TGDs NetDER aplicadas y dos homomorfismos apropiados en un orden lexicográfico y lineal (fijo), respectivamente, y luego asignar a cada nueva consecuencia obtenida el nivel (de derivación) k.

El *chase* de (D, G) para el período de tiempo $[t_1, t_2]$, denotado $chase_{[t_1,t_2]}(D, G)$, se define entonces como el límite para $k \to \infty$ de $chase^k_{[t_1,t_2]}(D, G)$. El *grafo del chase* para una base de conocimiento NetDER $KB = (D, G)$ en el período de tiempo $[t_1, t_2]$ es el grafo dirigido consistente de $chase_{[t_1,t_2]}(D, G)$ como el conjunto de nodos, los cuales tienen un arco desde a a b si y sólo si b es obtenido a partir de a utilizando un sólo paso de aplicación de una TGD NetDER.

Regla del chase para EGDs NetDER. Considere una base de conocimiento NetDER $KB = (D, G, \Sigma, P)$, un período de tiempo $[t_1, t_2] \subseteq [0, t_{max}]$ y una EGD NetDER $\epsilon \in \Sigma$ de la forma $\Upsilon(\mathbf{S}) \wedge \Phi(\mathbf{T}) \to (X_i = X_j) : \gamma_\epsilon(\mathbf{U})$, donde $X_i, X_j \in \mathbf{S} \cup \mathbf{T} \cup \mathbf{U}$, $\Upsilon(\mathbf{S})$ es una conjunción de átomos, $\Phi(\mathbf{T})$ es una conjunción de componentes objetivos de red y $\gamma_\epsilon(\mathbf{U})$

es una anotación global de red. Entonces, ϵ es aplicable a (D, G) en el período de tiempo $[t_1, t_2]$ si y sólo si:

- existe un homomorfismo h que mapea átomos de $\Upsilon(\mathbf{S})$ tal que $h(\Upsilon(\mathbf{S})) \subseteq D$;
- existe un homomorfismo h_1 que extiende h tal que la respuesta de la función *checkCondInState*(Q_1, *net_state*) es "Sí", donde $Q_1 = (h_1(\Phi(\mathbf{T}))) : [t_1, t_2]$ y *net_state* es la salida del Algoritmo 1 sobre la base de conocimiento NetDiff $DiffKB = (G, \emptyset)$; y
- la respuesta de la función *checkCondInState*(Q_2, *net_state*) es "Sí", donde $Q_2 = (\Omega, h_1(\gamma_\epsilon(\mathbf{U}))) : [t_1, t_2]$ y *net_state* es el mismo que el mencionado arriba.

Sea ϵ aplicable a (D, G) en el período de tiempo $[t_1, t_2]$; entonces, cuando ϵ es aplicado a (D, G) en el período de tiempo $[t_1, t_2]$, una de las siguientes situaciones puede ocurrir: (i) $h(X_i)$ y $h(X_j)$ son constantes y el *chase falla*, o (ii) $h(X_i)$ es reemplazado con $h(X_j)$ si X_i es un valor nulo y $h(X_j)$ es una constante o $h(X_i)$ es precedido por $h(X_j)$ en el orden lexicográfico.

Como es bien sabido, las EGDs consideradas deben cumplir la condición de separabilidad con respecto a las TGDs para evitar indecidibilidad debido a interacciones no deseadas entre estos dos tipos de reglas [CGL12].

Restricciones negativas NetDER. Considere una base de conocimiento NetDER $KB = (D, G, \Sigma, P)$, un período de tiempo $[t_1, t_2] \subseteq [0, t_{max}]$, y una NC NetDER $\mu \in \Sigma$ de la forma $\Upsilon(\mathbf{S}) \wedge \Phi(\mathbf{T}) \rightarrow \bot : \gamma_\mu(\mathbf{U})$, donde $\Upsilon(\mathbf{S})$ es una conjunción de átomos, $\Phi(\mathbf{T})$ es una conjunción de componentes objetivos de red, y $\gamma_\epsilon(\mathbf{U})$ es una anotación global de red. Como es usual, el *chase* sólo tiene que comprobar que:

- no existe un homomorfismo h tal que $h(\Upsilon(\mathbf{S})) \subseteq D$, o
- no existe un homomorfismo h_1 que extiende h tal que la respuesta de la función *checkCondInState*(Q_1, *net_state*) es "Sí", donde $Q_1 = (h_1(\Phi(\mathbf{T}))) : [t_1, t_2]$ y *net_state* es la salida del Algoritmo 1 sobre la base de conocimiento NetDiff $DiffKB = (G, \emptyset)$, y
- la respuesta de la función *checkCondInState*(Q_2, *net_state*) es "Sí", donde $Q_2 = (\Omega, h_1(\gamma_\mu(\mathbf{U}))) : [t_1, t_2]$ y *net_state* es el mismo que el mencionado arriba.

Si una de estas comprobaciones falla, entonces el *chase* falla; en otro caso, las restricciones pueden ser ignoradas para el resto del proceso.

Ejemplo 5.6 *Considere la base de conocimiento* NETDER $KB = (D, G, \Sigma, P)$ *donde* D, Σ, P *son los mismos del Ejemplo 5.5 pero* G *es el mismo del Ejemplo 5.3. Entonces, la TGD* NETDER $\omega_3 \in \Sigma$ *es aplicable a* (D, G) *en el período de tiempo* $[0, t_{max}]$ *porque: existe un homomorfismo* h_1 *que extiende* h *(en este caso,* h *es un mapeo vacío) tal que la respuesta de la función checkCondInState(*Q_1*, net_state) es "Sí", donde:*

- $Q_1 = \Big(h_1\big((node(X), \langle expert(S), [0{,}9, 1{,}0]\rangle \wedge \langle bel\_dang(T), [0{,}8, 1{,}0]\rangle)\big)\Big) : [0, t_{max}]$
- $h_1\big((node(X), \langle expert(S), [0{,}9, 1{,}0]\rangle \wedge \langle bel\_dang(T), [0{,}8, 1{,}0]\rangle)\big) =$
  $(node(n_1), \langle expert(trojan), [0{,}9, 1{,}0]\rangle \wedge \langle bel\_dang(gooligan), [0{,}8, 1{,}0]\rangle)$
- *net_state es la salida del Algoritmo 1 sobre la base de conocimiento* NetDiff *DiffKB* $= (G, \emptyset)$*, la cual puede ser vista en la parte (b) (solo los hechos* NetDiff *son aplicados) de la Figura 4.7.*

Cuando ω_3 *es aplicado a* (D, G) *en el período de tiempo* $[0, t_{max}]$*, entonces los átomos* $h_2(product(P)) = product(z_1)$*,* $h_2(user(UID, N)) = user(z_2, z_3)$*,* $h_2(hyp\_at\_risk(P, UID)) = hyp\_at\_risk(z_1, z_2)$ *se agregan a* D *(en este caso, nada es agregado a* G*).* ∎

Aunque el lenguaje ontológico NETDER es definido como una extensión de *Datalog+/–*, la base de datos de red (G) junto con la parte ontológica (D, Σ) de la base de conocimiento NETDER, con cada tipo de regla nueva, puede ser codificada en una base de conocimiento *Datalog+/–* clásica (ver Apéndice, Sección 9.1).

5.3. Respuesta a consultas NETDER

En esta sección, se implementará el Módulo de Respuesta a Consultas (QAM, cf. Figura 4.1) y se describirá cómo el proceso de respuesta a consultas se puede llevar a cabo a partir de una colaboración entre el Módulo de Razonamiento Ontológico y Módulo de Difusión de Red (ORM y NDM, respectivamente). Estos dos módulos operan como procesos *concurrentes y asíncronos*; el procedimiento de respuesta a consultas, por lo

tanto, necesita abordar cómo ocurren las *interacciones* entre estos dos módulos. Como se verá a continuación, esto conduce a un análisis no trivial de las condiciones bajo las cuales se garantiza que tal proceso terminará. Primero, se define el tipo de consultas que se consideran.

Definición 5.8 (Consulta conjuntiva NetDER) *Sean* $\mathbf{X}, \mathbf{Y}$ *dos secuencias de variables, y sea* $[t_1, t_2] \subseteq [0, t_{max}]$ *un período de tiempo; entonces una consulta conjuntiva* NetDER *es de la siguiente forma:*

$$Q(\mathbf{X}) = \exists \mathbf{Y}\, \rho(\mathbf{X}_1, \mathbf{Y}_1) \ \wedge \ \phi(\mathbf{X}_2, \mathbf{Y}_2) \ \wedge \ \gamma(\mathbf{X}_3, \mathbf{Y}_3) : [t_1, t_2]$$

donde $\mathbf{X}, \mathbf{Y}, \mathbf{X}_1, \mathbf{Y}_1, \mathbf{X}_2, \mathbf{Y}_2, \mathbf{X}_3, \mathbf{Y}_3$ *son secuencias de variables,* $\mathbf{X} = \mathbf{X}_1 \cup \mathbf{X}_2 \cup \mathbf{X}_3$, $\mathbf{Y} = \mathbf{Y}_1 \cup \mathbf{Y}_2 \cup \mathbf{Y}_3$, $\rho(\mathbf{X}_1, \mathbf{Y}_1)$ *es una conjunción de átomos sobre un esquema relacional* $\mathcal{R}$ *con argumentos de un conjunto de variables* $\mathbf{V}$ *y constantes* Δ, $\phi(\mathbf{X}_2, \mathbf{Y}_2)$ *y* $\gamma(\mathbf{X}_3, \mathbf{Y}_3)$ *son una conjunción de componentes objetivos de red y una anotación global de red, respectivamente, sobre un conjunto de símbolos predicativos para etiquetas* $\mathcal{P}$ *con argumentos obtenidos de un conjunto de variables* $\mathbf{V}$ *y constantes* Δ. *Se dice que una consulta* NetDER *es booleana cuando* $\mathbf{X} = \{\}$, *por lo cual sólo puede tener "Sí" o "No" como respuesta.*

5.3.1. Cómputo de respuestas para consultas NetDER

Una vez que se dispone de una definición de consultas NetDER, el siguiente paso es establecer una forma de obtener respuestas a esas consultas. Claramente, podría haber muchas maneras diferentes de computar las respuestas y, por lo tanto, en esta sección el foco está puesto en cómo se pueden obtener tres clases particulares de respuestas de interés, y estudiar el problema de la factibilidad de su cómputo. Tales clases se originan de cómo se maneja la interacción entre el ORM y el NDM.

Con la intención de estructurar la interacción entre los módulos, se proponen dos políticas principales y una tercera que es un caso particular de una de ellas (cf. Figura 5.4). Como se verá, estas políticas ayudarán a definir condiciones bajo las cuales el proceso de respuesta a consultas termina. En cada caso, los procesos de difusión de red y *chase* son llevados a cabo de a uno a la vez; la diferencia radica en cuándo esta secuencia de interacciones se detiene. Para ello, se definen las siguientes *Políticas de Respuesta a Consultas* (QAP):

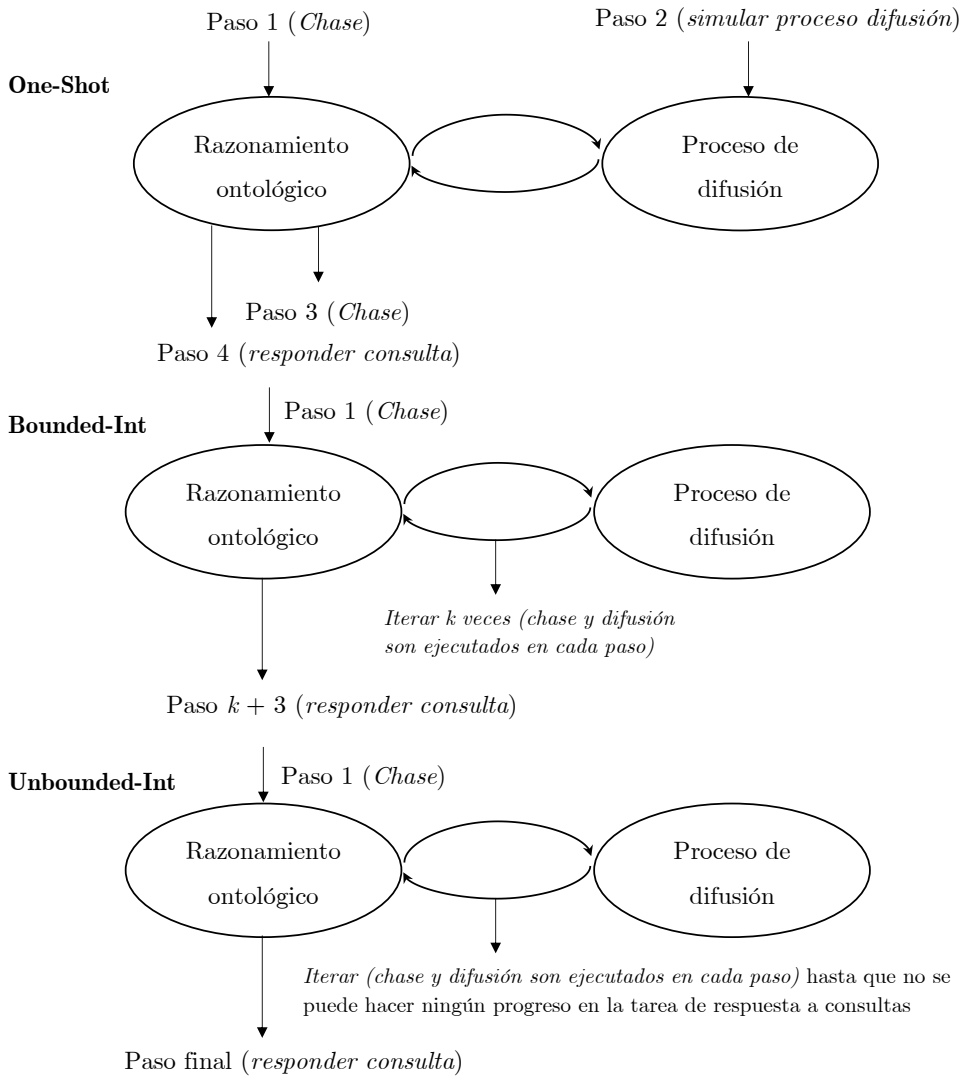

Figura 5.4: Representación gráfica de las tres posibles interacciones consideradas entre ORM y NDM durante el proceso de respuesta a consultas.

- *Política Unbounded-Int*: El proceso termina cuando la consulta puede ser respondida o una condición dada es satisfecha en la red y/o la ontología (por ejemplo, un punto fijo es alcanzado). Por lo tanto, el número de iteraciones entre el ORM y el NDM no está necesariamente acotado.

- *Política Bounded-Int*: El proceso termina cuando la consulta puede ser respondida o la cantidad de iteraciones es mayor que un número dado.

- *Política One-Shot*: El proceso termina cuando la consulta puede ser respondida o

Algorithm 2 Respuesta a consultas NetDER

Require: Consulta Q, QAP *Policy*, $KB = (D, G, \Sigma, P)$

```
1:  done:= false
2:  ans:= ∅
3:  while !done do
4:     ch:= ExecuteChase(KB)                                          [ORM]
5:     ans:= ans ∪ EvaluateQuery(Q, Policy, ch)
6:     if !done then
7:        DiffKB := (G, P)
8:        net_state := ExecuteDiffProc(DiffKB) // Algoritmo 1         [NDM]
9:        update(G, net_state)                                        [ORM]
10:       ch:= ExecuteChase(KB)                                       [ORM]
11:       ans:= ans ∪ EvaluateQuery(Q, Policy, ch)
12:    else if !done then
13:       UpdateEnvironment(D, G)
14:    end if
15: end while
16: return ans
```

después de realizar una única iteración entre el ORM y el NDM. Es decir, si el *chase* no puede responder la consulta, entonces el proceso de difusión es ejecutado y cuando se completa, el *chase* intenta responder la consulta una vez más antes de terminar el proceso. Observe que esta política es un caso específico de la política *Bounded-Int*.

El Algoritmo 2 describe el proceso de respuesta a consultas NetDER en un pseudocódigo de muy alto nivel. El primer paso es ejecutar el *chase*, luego se evalúa la consulta y si ya se puede responder o se cumplen las condiciones de terminación, *done* se establece en *true*. De lo contrario, se lleva a cabo el proceso de difusión (cf. Algoritmo 1) y se obtiene un nuevo estado NetDiff *net_state*, luego G se actualiza a partir de *net_state*. La consulta se evalúa de nuevo y, si *done* sigue siendo *false* entonces el entorno se actualiza—esto se refiere a la incorporación de datos recientemente disponibles y a la "limpieza" general que implica la adición de conocimiento a G proveniente del proceso de difusión. La actualización del entorno debe realizarse considerando el tiempo actual (t_{now}) e implementando una estrategia para manejar información en conflicto. Los detalles de tales estrategias

están fuera del alcance de esta tesis, pero hay muchas alternativas bien estudiadas que pueden ser adoptadas para este propósito.

El algoritmo itera hasta que *evaluate_query* establece *done* en *true* de acuerdo con la política *QAP* correspondiente.

Ahora, se define qué constituye una respuesta a una consulta conjuntiva NetDER.

Definición 5.9 (Respuestas a consultas conjuntivas NetDER) *Dada una consulta conjuntiva* NetDER Q*, una base de conocimiento* NetDER *KB, y una QAP Policy, entonces el conjunto de respuestas para* Q*, denotado* $ans(Q, KB, Policy)$*, es el conjunto de todas las tuplas* t *tales que* t *es una respuesta de* Q *para el Algoritmo 2 sobre KB y Policy. Si* Q *es booleana, entonces la respuesta a* $Q()$ *es Sí si y sólo si existe alguna respuesta de* Q *para el Algoritmo 2 sobre KB y Policy, y No en cualquier otro caso.*

A continuación, se muestra un ejemplo clarificador sobre el proceso de respuesta a consultas NetDER.

Ejemplo 5.7 *Considere la ejecución del Algoritmo 2 sobre la consulta* NetDER

$$Q = \exists\, U, UID\, \big(product(U) \wedge hyp\_at\_risk(U, UID)\big) : [0, t_{max}],$$

QAP Policy = One-Shot, y la base de conocimiento NetDER $KB = (D, G, \Sigma, P)$ *donde* D, G, P *son los mismos del Ejemplo 5.6 y* $\Sigma = \{\omega_2\}$ *es el mismo del Ejemplo 5.5. La primer ejecución del chase (línea 4 en el Algoritmo 2) no agrega nuevo conocimiento a* (D, G) *porque* ω_2 *no puede ser aplicada. Cuando el proceso de difusión es ejecutado (línea 8 en el Algoritmo 2), un estado final* NetDiff *es obtenido, el cual puede ser visto en la parte (c) de la Figura 4.7. Entonces,* G *es actualizado con el estado final* NetDiff *tal que ahora* G *contiene*[5] *:*

$$\begin{aligned} G = \; & \{node(n_1), node(n_2), edge(n_1, n_2)\} \cup \mathcal{L} \cup \{tmax(2)\} \cup \\ & \{\big(node(n_1), \langle expert(trojan), [0{,}9, 1]\rangle\big) : [0, t_{max}], \\ & \big(node(n_1), \langle bel\_dang(gooligan), [0{,}8, 1]\rangle\big) : [0, t_{max}], \\ & \big(node(n_2), \langle bel\_dang(gooligan), [0{,}7, 1]\rangle\big) : [1, t_{max}], \\ & \big(edge(n_1, n_2), \langle close(trojan, windows), [0{,}8, 1]\rangle\big) : [0, t_{max}], \\ & \big(\Omega, \langle dangerous(gooligan), [0{,}8, 1]\rangle\big) : [0, t_{max}]\} \end{aligned}$$

[5]Recordar que los hechos NetDiff relativos a etiquetas con cotas $[0, 1]$ pueden ser omitidos.

Ahora, el chase es ejecutado otra vez con la nueva base de conocimiento (línea 10 en el Algoritmo 2). En este caso, ω_1 es aplicable a (D, G) y los átomos $h_2(product(U)) = product(z_1)$, $h_2(product(U)) = product(z_1)$, $h_2(hyp\_at\_risk(U, UID)) = hyp\_at\_risk(z_1, n_1)$ son agregados a D, pero a G nada es agregado. Finalmente, las iteraciones entre el ORM y el NDM terminan debido a que QAP es definido como One-Shot, y la respuesta para la consulta Q es "Sí". ∎

Ahora, se presenta el principal resultado de complejidad, que establece las condiciones bajo las cuales el proceso de respuesta a consultas NETDER preserva la complejidad del lenguaje ontológico subyacente. Solo se considera *complejidad de datos* para estos resultados.

Teorema 5.2 *Sea $KB = (D, G, \Sigma, P)$ una base de conocimiento* NETDER*, $Q(\mathbf{X})$ una consulta conjuntiva* NETDER*, $trans(KB) = KB_t$ (cf. Apéndice, Sección 9.1) la base de conocimiento transformada (base de conocimiento Datalog+/– clásica) de KB, y $trans(Q(\mathbf{X})) = Q_t(\mathbf{X})$ (cf. Apéndice, Sección 9.1) la consulta transformada (consulta Datalog+/– clásica) de $Q(\mathbf{X})$. Si $\mathbf{C}$ es la clase de complejidad computacional asociada con el proceso de respuesta a la consulta $Q_t(\mathbf{X})$ sobre KB_t, entonces la complejidad del proceso de respuesta a la consulta $Q(\mathbf{X})$ sobre KB es al menos polinómica y a lo sumo $\mathbf{C}$ bajo las siguientes condiciones:*

- *política Bounded-Int con garantías de difusión y capacidad para modificar la red, o*
- *política Unbounded-Int con garantías de difusión y sin capacidad para modificar la red.*

La siguiente observación hace referencia a las entradas "*puede no terminar*" de la Figura 5.5, lo cual puede apreciarse que se sostiene por medio de un simple ejemplo.

Observación 5.2 *El proceso de respuesta a consultas* NETDER *puede no terminar cuando no hay garantías de que un punto fijo sea alcanzado al computar el resultado del proceso de difusión. Esto es simplemente porque es posible definir funciones de influencia que produzcan como salida intervalos que sean una proporción de los valores de entrada; así, por ejemplo, si la función de influencia devuelve una cota cuyo límite inferior es la suma de los límites inferiores de las cotas de entrada multiplicada por* 0,5 *y el límite superior de la cota superior es obtenida de forma análoga, entonces en dicho caso, nunca se alcanzará un punto fijo.*

QAP	Modificación de la red *permitida*		Modificación de la red *no permitida*	
	Garantías de dif.	*Sin* garantías de dif.	Garantías de dif.	*Sin* garantías de dif.
Bounded-Int	**C** ∪ PTIME [Teorema 5.2]	*Puede no terminar* [Observación 5.2]	**C** ∪ PTIME [Teorema 5.2]	*Puede no terminar* [Observación 5.2]
Unbounded-Int	*Indecidible* [Teorema 5.3]	*Indecidible* [Teorema 5.3]	**C** ∪ PTIME [Teorema 5.2]	*Puede no terminar* [Observación 5.2]

Figura 5.5: Resultados de complejidad, terminación y decidibilidad para el proceso de respuesta a consultas NetDER bajo diferentes QAPs, considerando si es posible la modificación de la estructura de la red o no y si se dispone de garantías de difusión o no. Tres tipos de resultados son mostrados: *Indecidible*, *Puede no terminar* (ciclos infinitos pueden ocurrir), "**C**", lo cual significa que el proceso de respuesta a consultas NetDER tiene la misma de clase complejidad que el proceso de respuesta a la consulta transformada sobre la base de conocimiento transformada.

Dos observaciones más—no incluidas en la tabla—son las siguientes:

Primero, la decidibilidad y complejidad del proceso de respuesta a consultas NetDER bajo la política *One-Shot* o *Bounded-Int* y con garantías de terminación del proceso de difusión requiere al menos tiempo polinomial porque se asume que el proceso de difusión requiere tiempo polinomial. En general, la clase "**C**" a la que se hace referencia en la tabla depende de las características del lenguaje ontológico subyacente. Por ejemplo, si el lenguaje ontológico solo permite *guarded* o *frontier guarded* TGDs, entonces la complejidad del tiempo es polinomial, pero si permite *weakly guarded* TGDs, entonces se vuelve exponencial (ver Teorema 5.2).

En segundo lugar, el proceso de respuesta a consultas NetDER bajo la política *Unbounded-Int* con garantías de terminación del proceso de difusión y sin capacidad para modificar la red, ya sea desde la ontología o el entorno, es un caso particular de la política *One-Shot* bajo las mismas condiciones y, por lo tanto, los resultados son los mismos. Esto se debe a que la red no se puede modificar, por lo que ejecutar el proceso de difusión k veces es lo mismo que ejecutarlo solo una vez, como se hace bajo la política *One-Shot*.

El resultado final obtenido es que el proceso de respuesta a consultas bajo la política *Unbounded-Int* con la capacidad de modificar la estructura de la red es indecidible; esto se muestra en el siguiente teorema.

Teorema 5.3 *El proceso de respuesta a consultas* NetDER *bajo la política Unbounded-int con la capacidad de modificar la estructura de la red es indecidible, incluso bajo condiciones que garantizan:*

- *terminación en el proceso de respuesta a la consulta transformada sobre la base de conocimiento transformada, y/o*
- *terminación en el proceso de difusión.*

Prueba en Apéndice, Sección 9.2.

Para concluir, en la siguiente sección se presenta un caso de uso para ilustrar la instanciacion de la arquitectura NetDER en un domino particular de aplicación, considerando los fundamentos teóricos estudiados a lo largo de este capítulo.

5.4. Caso de uso

En esta sección, se describe un caso de uso para la aplicación de NetDER con el propósito de abordar un problema real en un dominio de ciberseguridad. Este ejemplo extendido está inicialmente inspirado en trabajos que abordan diferentes problemas en este dominio, como [SBD+18, SASS18a]. En [SBD+18] los autores proponen un sistema para generar advertencias sobre amenazas cibernéticas utilizando información sobre actores maliciosos en la *Darknet* e información de usuarios de Twitter expertos en ciberseguridad. Por otro lado, en [SASS18a] los autores buscan predecir los incidentes cibernéticos utilizando la estructura de la red de respuesta obtenida a partir de las interacciones de los usuarios de los foros en la *Darknet*. Sin embargo, dos aspectos que deseamos destacar en este apartado son: (i) los *frameworks* utilizados para tales predicciones generalmente se limitan a datos externos (es decir, *Darknet*, redes sociales, etc.) y no son extensibles para considerar otras fuentes de datos de ciberseguridad como *DNS sinkholes*, datos de IDS/IPS y *honeypots*; (ii) es muy difícil obtener explicaciones sobre cómo se logra un resultado.

A continuación, se describe cómo se puede crear una instancia de NetDER para abordar el problema de (i) detectar términos peligrosos que pueden conducir a productos IT que estén en riesgo en un momento específico y (ii) determinar cuáles son los productos

que efectivamente están en riesgo. A lo largo de esta sección, se hacen las siguientes suposiciones:

- Se dispone de información acerca de publicaciones escritas por diferentes actores y se dispone también de sus interacciones en foros de comunidades de *hackers* en línea basadas en la utilización de tecnologías como la *Deepnet* y la *Darkweb*,
- Es posible determinar cuándo un actor es un *experto del dominio*; existen trabajos que abordan este problema, tales como [SASS18a], en el que los autores consideran que los actores son expertos en base a sus menciones de *Common Vulnerabilities and Exposures* (CVE)[6] relacionadas a grupos de Common Platform Enumeration (CPE)[7] importantes (definido en relación a la frecuencia en los datos) y sus respuestas a otros actores. En otros trabajos, tales como [SBD+18], se basan en la existencia de "expertos reconocidos" en plataformas sociales como Twitter,
- Un actor determinado puede ser un experto solo en algunos dominios. Para ello se utiliza la clasificación de MITRE Corporation para patrones de ataque basados en los siguientes dominios: *software*, *hardware*, *communications*, *supply chain*, *social engineering* y *physical security*. En este caso de uso, solo consideramos los tres primeros por razones de simplicidad.
- Algunas de las tareas de razonamiento se basan en el *Módulo de Ingesta de Datos* que, como se comentó en el Capítulo 4, se encarga del preprocesamiento y mantenimiento de los datos. En este caso, construye y mantiene la base de datos de usuarios, productos, sus proveedores y sus plataformas, detecta y mantiene quién es un *early poster* en relación a un término específico, procesa y mantiene la frecuencia de menciones de términos y productos, proveedores o plataformas en las publicaciones del foro, y crea y mantiene la base de datos de la red (cf. Figura 4.1).

El resto de esta sección se organiza de la siguiente manera: se comienza con un escenario simplificado, describiendo primero la parte de la red en la Sección 5.4.1 y luego la parte de

[6]Se asignan números CVE a cada vulnerabilidad incluida en la *National Vulnerability Database*, que proporciona definiciones para las vulnerabilidades y exposiciones de ciberseguridad divulgadas públicamente. Su objetivo es facilitar el intercambio de datos entre diferentes herramientas. Las entradas contienen un número de identificación, una descripción y al menos una referencia pública.

[7]CPE es un método estandarizado para describir e identificar clases de aplicaciones, sistemas operativos y dispositivos de *hardware*.

la ontología en la Sección 5.4.2. Luego, en la Sección 5.4.3 se profundiza más, desarrollando el caso de uso con mayor detalle.

5.4.1. Conocimiento de red

Se dispone de un grafo dirigido en el que cada nodo representa a un actor en algún foro de la *Darknet*, y dos nodos están conectados si uno de ellos ha respondido a una publicación del otro[8]. Los símbolos predicativos para etiquetas disponibles son los siguientes:

- *bel_dang*: permite instanciar etiquetas locales de nodo para representar el grado con el que cada nodo cree que un término determinado es peligroso.
- *expert*: permite instanciar etiquetas locales de nodo para representar el grado en el que cada nodo es experto en un dominio particular de ataques cibernéticos.
- *related*: permite instanciar etiquetas locales de nodo para representar el grado con el cual un nodo cree que algún dominio de ataque cibernético y algún término están relacionados.

El Módulo de Ingesta de Datos podría obtener las etiquetas basadas en *bel_dang* mediante el uso de herramientas de *Procesamiento de Lenguaje Natural* (NLP) para detectar términos desconocidos en las publicaciones y evaluar el grado de peligro según los CVE y/o las palabras conocidas de ciberseguridad mencionadas. Por otro lado, las etiquetas basadas en *expert* podrían obtenerse mapeando cada CVE mencionado a algún dominio de ataque cibernético de acuerdo a la información proporcionada por MITRE; entonces la experticia podría ser una proporción de CVEs en cada dominio. Finalmente, las etiquetas basadas en *related* podrían ser el resultado de aplicar herramientas de NLP a la publicación donde se menciona un término específico; estas etiquetas pueden medir la relación entre un término en una publicación y un dominio de ataque.

De manera análoga, las etiquetas globales pueden instanciarse a partir del símbolo predicativo *dangerous*, obteniendo etiquetas para representar el grado en el que la red en su conjunto cree que un término determinado es peligroso.

A continuación, se detallan las reglas NetDiff:

[8]Esta conexión podría enriquecerse con una etiqueta para representar la cercanía, lo cual podría basarse en la frecuencia de respuestas entre ellos; sin embargo, no se consideran etiquetas locales de arco por razones de simplicidad.

- $R_1 : bel\_dang(Y)_{\mathsf{T}} \overset{1}{\leftarrow} (\mathsf{T}, \langle bel\_dang(Y), [0{,}7, 1{,}0]\rangle \wedge \langle expert(X), [0{,}7, 1{,}0]\rangle \wedge \langle related(X, Y), [0{,}8, 1{,}0]\rangle, if_1)$

  Esta regla declara que todo nodo (nodo objetivo) con al menos la mitad de sus vecinos (de acuerdo a cómo se define en if_1, ver más abajo) que crean que Y es peligroso con confianza entre 0,7 y 1,0, sean expertos en el dominio X con confianza entre 0,7 y 1,0 y crean que X está relacionado con Y con confianza entre 0,8 y 1,0, será afectado (implica una actualización) con respecto a su grado de creencia acerca de que el término Y es peligroso. Dicha actualización dependerá de lo que se defina en la función de influencia if_1 y se hará efectiva en el siguiente punto de tiempo.

- $R_2 : bel\_dang(Y)_{\mathsf{T}} \overset{1}{\leftarrow} (\mathsf{T}, \langle bel\_dang(Y), [0{,}5, 1{,}0]\rangle \wedge \langle expert(X), [0{,}0, 0{,}0]\rangle, if_2)$

  Similarmente, esta regla describe que todo nodo con al menos una proporción de 0,7 de sus vecinos que crean que Y es peligroso con confianza de al menos 0,5 (y no sean expertos), será afectado con respecto a su grado de creencia acerca de que el término Y es peligroso. Dicha actualización dependerá de lo que se defina en la función de influencia if_2, la cual requiere un número mayor de vecinos que if_1 y tiene un efecto más pequeño en la etiqueta objetivo (en este caso $bel\_dang(Y)$) porque se asume que aquellos nodos que no son expertos tienen una influencia menor. Nuevamente, para esta regla el efecto se hará efectivo en el siguiente punto de tiempo.

- $R_3 : dangerous(Y) \leftarrow (bel\_dang(Y)_{\langle expert(X), [0{,}7, 1{,}0]\rangle}, af_1)$

  Finalmente, la regla R_3 declara que el grado con el cual se cree *globalmente* que Y es peligroso está basado en la aplicación de la función de agregación af_1 a la creencia de cada nodo que es experto con confianza entre 0,7 y 1,0.

Las funciones de influencia mencionadas anteriormente son definidas de la siguiente forma:

$$if_1(V', node(n)) = \begin{cases} [0{,}7, 1{,}0] & \text{si } |V' \cap A|/|A| \geq 0{,}5 \\ [0{,}0, 1{,}0] & \text{en otro caso} \end{cases}$$

donde $A = \{node(n') \in V \text{ tal que } edge(n', n) \in E\}$, y por otro lado,

$$if_2(V', node(n)) = \begin{cases} [0{,}5, 1{,}0] & \text{si } |V' \cap A|/|A| \geq 0{,}7 \\ [0{,}0, 1{,}0] & \text{en otro caso} \end{cases}$$

La función de agregación af_1 se define como $af_1(B) = [l, u]$ donde:

$$l = \sum_{[l_i,u_i]\in B}^{|B|} \frac{l_i}{|B|} \quad \text{y} \quad u = \sum_{[l_i,u_i]\in B}^{|B|} \frac{u_i}{|B|}$$

5.4.2. Conocimiento ontológico

En el Módulo de Razonamiento Ontológico se tiene disponible información sobre usuarios de diferentes foros, quiénes son los primeros en publicar respecto a términos específicos (*earlyPoster*/2), los productos (*product*/1) referidos, sus proveedores (*vendor*/1), plataformas (*platform*/1) y la frecuencia de co-ocurrencia entre un término específico y un producto, un proveedor o una plataforma (*co_occur*/3). Una versión reducida de una instancia de base de datos con las características mencionadas podría ser:

$d_1 : user(n_1, maximus)$, $d_2 : user(n_2, magik)$,
$d_3 : earlyPoster(n_1, double\text{-}kill)$, $d_4 : co\_occur(double\text{-}kill, windows\text{-}10, 0{,}75)$,
$d_5 : co\_occur(double\text{-}kill, internet\text{-}explorer, 0{,}7)$, $d_6 : platform(application)$,
$d_7 : platform(operating\text{-}system)$, $d_8 : vendor(microsoft)$,
$d_9 : product(windows\text{-}10)$, $d_{10} : parent(microsoft, windows\text{-}10)$,
$d_{11} : parent(operating\text{-}system, microsoft)\}$.

Asimismo, se define una TGD y EGD NetDER.

$$\omega_1 : user(UID, N) \wedge earlyPoster(UID, T) \rightarrow \exists P\, product(P) \wedge hyp\_at\_risk(P, UID) \wedge hyp\_vulnerability(T, P) : \langle dangerous(T), [0{,}5, 1{,}0]\rangle$$

Intuitivamente, la regla declara que si un usuario (*user*/2) es uno de los primeros en publicar sobre un término T (*earlyPoster*/2) que es considerado *globalmente* peligroso (*dangerous*/1) por la red con confianza entre 0,5 y 1,0, entonces es posible formular la hipótesis de que algún producto está en riesgo (*hyp_at_risk*/2), el usuario (*user*/2) en cuestión es responsable de eso, y también formular la hipótesis de que T aprovecha vulnerabilidades (*hyp_vulnerability*/2) del producto (*product*/1) P.

$$\epsilon_1 : product(P_1) \wedge hyp\_vulnerability(T, P_1) \wedge product(P_2) \wedge co\_occur(T, P_2, RF) \wedge RF > \theta \rightarrow P_1 = P_2$$

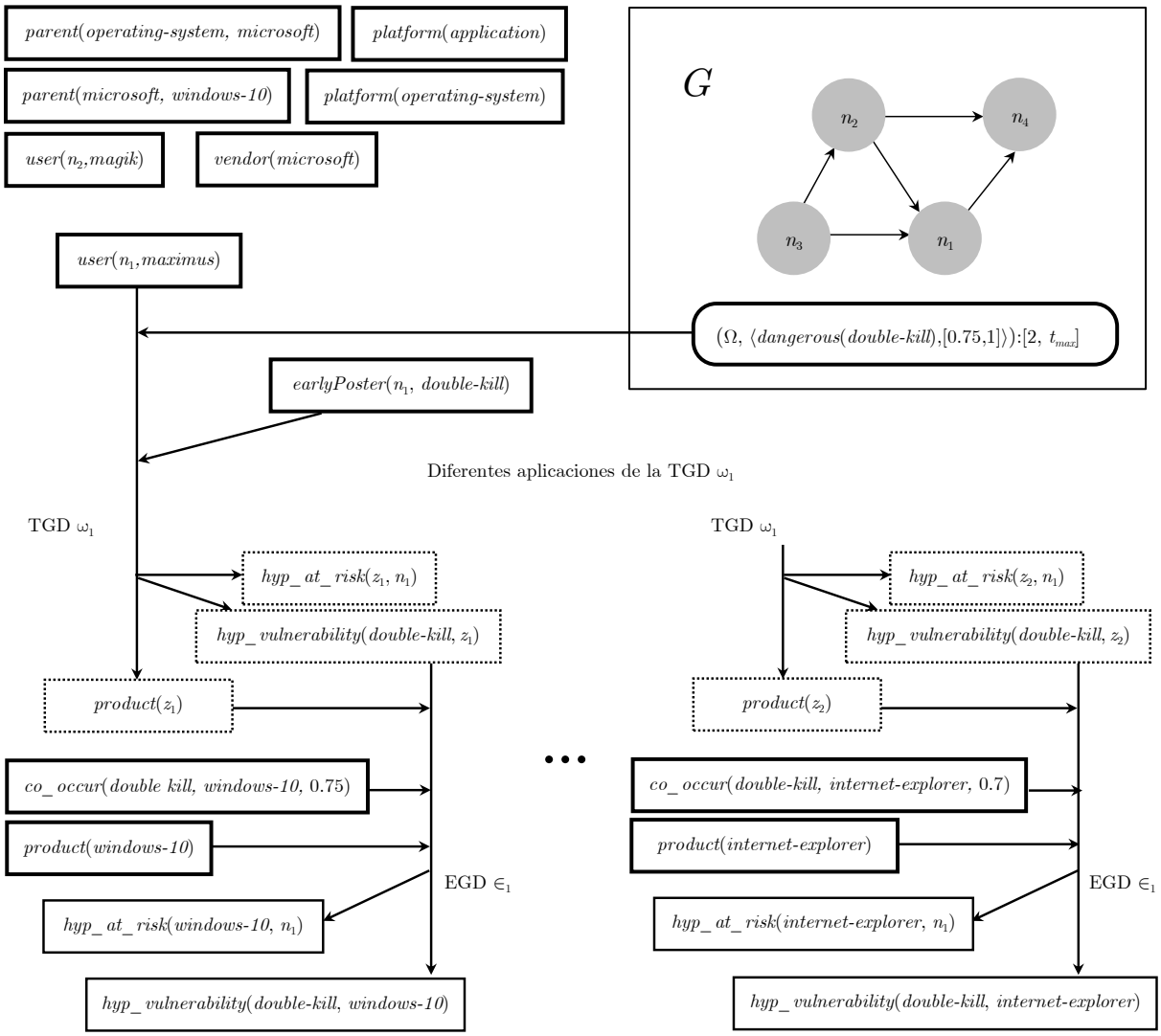

Figura 5.6: Ejemplo del *chase* basado en conocimiento ontológico y de red.

Esta regla dice que si existe una hipótesis acerca de que T aprovecha vulnerabilidades (*hyp_vulnerability*/2) de algún producto (*product*/1) P_1 y también de otro producto (*product*/1) P_2 que co-ocurre con T en publicaciones de foros con una frecuencia relativa determinada (*co_occur/3*), entonces estos productos son los mismos.

Aplicando reglas ontológicas NETDER. La Figura 5.6 muestra el resultado de aplicar el procedimiento *chase* a la base de datos D_1, la TGD ω_1, la EGD ϵ_1 y la consulta $Q = \exists X, Y \ \ hyp\_vulnerability(X, Y) : [t_{max}, t_{max}]$. Rectángulos con bordes más gruesos representan átomos de la ontología, aquellos con líneas punteadas representan átomos con

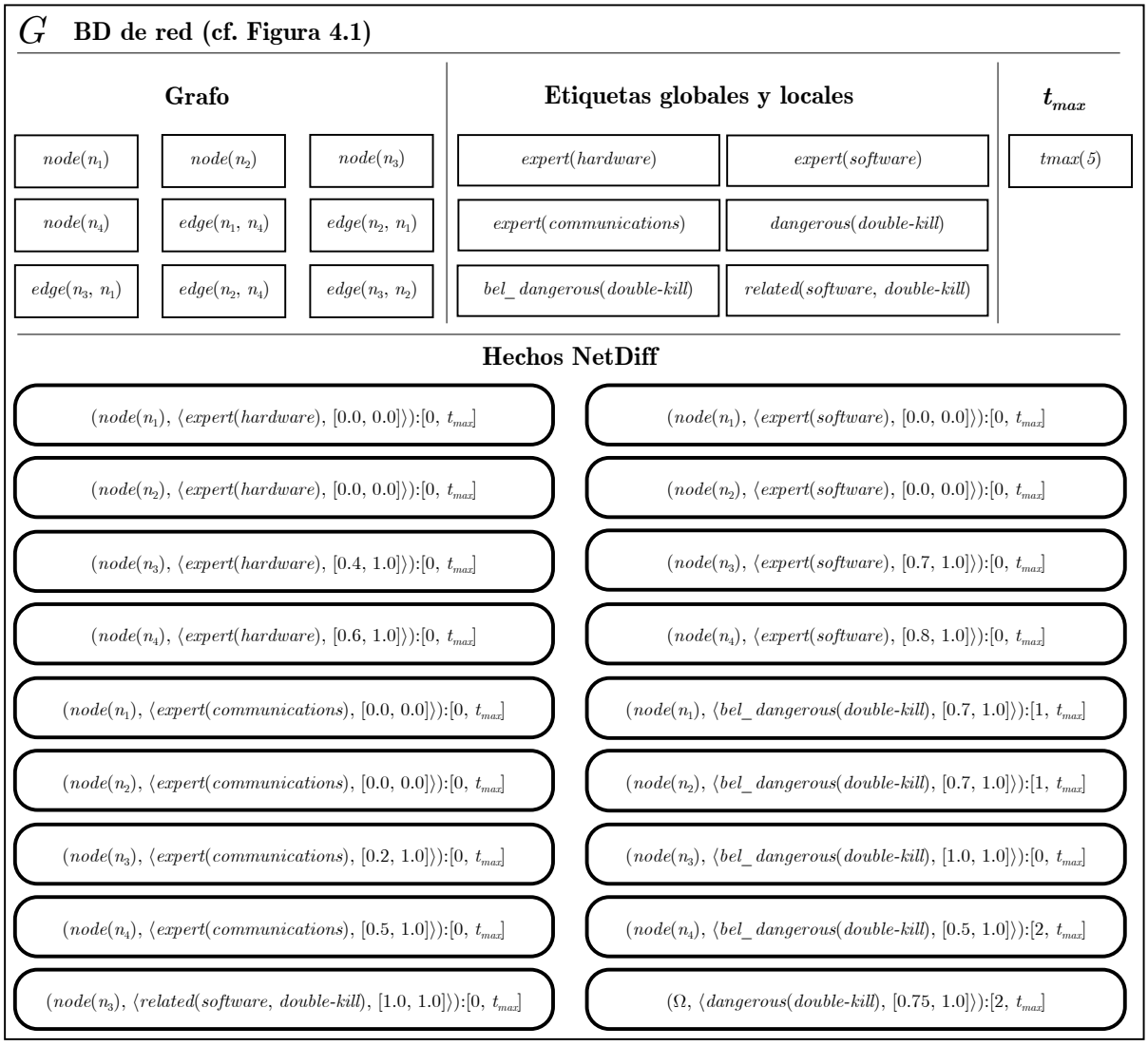

Figura 5.7: Vista detallada relativa a la BD de red vinculada a la base de conocimiento NetDER en el momento en que el *chase* es llevado a cabo en el contexto de la Figura 5.6.

valores nulos (un tipo de conocimiento parcial), y las flechas representan la aplicación de TGDs y EGDs NetDER. Además, el rectángulo con el símbolo G se utiliza para distinguir la base de datos de red (cf. Figura 4.1 para contextualizar la base de datos de red dentro de la arquitectura NetDER) y contiene una representación gráfica de las relaciones entre nodos y un hecho NetDiff (ver Figura 5.7 para mayores detalles del contenido de la base de datos de red). Cuando ω_1 es aplicado, un nuevo valor nulo z_1 es

Tiempo	*Paso*	G	L_{loc1}	L_{loc2}	L_{loc3}	L_{loc4}	L_{loc5}	L_{loc6}	L_{loc7}
0	0	$node(n_1)$	[0.0, 0.0]	[0.0, 0.0]	[0.0, 0.0]	?	?	?	?
0	0	$node(n_2)$	[0.0, 0.0]	[0.0, 0.0]	[0.0, 0.0]	?	?	?	?
0	0	$node(n_3)$	[0.7, 1.0]	[0.4, 1.0]	[0.2, 1.0]	[1.0, 1.0]	?	?	[1.0, 1.0]
0	0	$node(n_4)$	[0.8, 1.0]	[0.6, 1.0]	[0.5, 1.0]	?	?	?	?
1	1	$node(n_1)$	[0.0, 0.0]	[0.0, 0.0]	[0.0, 0.0]	?	?	?	[0.7, 1.0]
1	1	$node(n_2)$	[0.0, 0.0]	[0.0, 0.0]	[0.0, 0.0]	?	?	?	?
1	1	$node(n_3)$	[0.7, 1.0]	[0.4, 1.0]	[0.2, 1.0]	[1.0, 1.0]	?	?	[1.0, 1.0]
1	1	$node(n_4)$	[0.8, 1.0]	[0.6, 1.0]	[0.5, 1.0]	?	?	?	?
1	2	$node(n_1)$	[0.0, 0.0]	[0.0, 0.0]	[0.0, 0.0]	?	?	?	[0.7, 1.0]
1	2	$node(n_2)$	[0.0, 0.0]	[0.0, 0.0]	[0.0, 0.0]	?	?	?	[0.7, 1.0]
1	2	$node(n_3)$	[0.7, 1.0]	[0.4, 1.0]	[0.2, 1.0]	[1.0, 1.0]	?	?	[1.0, 1.0]
1	2	$node(n_4)$	[0.8, 1.0]	[0.6, 1.0]	[0.5, 1.0]	?	?	?	?
2	3	$node(n_1)$	[0.0, 0.0]	[0.0, 0.0]	[0.0, 0.0]	?	?	?	[0.7, 1.0]
2	3	$node(n_2)$	[0.0, 0.0]	[0.0, 0.0]	[0.0, 0.0]	?	?	?	[0.7, 1.0]
2	3	$node(n_3)$	[0.7, 1.0]	[0.4, 1.0]	[0.2, 1.0]	[1.0, 1.0]	?	?	[1.0, 1.0]
2	3	$node(n_4)$	[0.8, 1.0]	[0.6, 1.0]	[0.5, 1.0]	?	?	?	[0.5, 1.0]

Figura 5.8: Estado de la base de datos de red en los tiempos 0, 1, y 2. Se utilizan lo siguientes nombres: $L_{loc1} = expert(software)$, $L_{loc2} = expert(hardware)$, $L_{loc3} = expert(communications)$, $L_{loc4} = related(software, double\text{-}kill)$, $L_{loc5} = related(hardware, double\text{-}kill)$, $L_{loc6} = related(communications, double\text{-}kill)$, $L_{loc7} = bel\_dang(double\text{-}kill)$.

creado y tres nuevos átomos son producidos:

$$product(z_1),$$
$$hyp\_at\_risk(z_1, n_1),$$
$$hyp\_vulnerability(double\text{-}kill, z_1).$$

Entonces, cuando ϵ_1 es aplicado z_1 es mapeado a un valor conocido y se tiene: $z_1 = windows\text{-}10$. En este punto, ω_1 es aplicado otra vez (esto podría hacerse una cantidad de veces no acotada), un nuevo valor nulo z_2 es creado y tres nuevos átomos son producidos:

$$product(z_2),$$
$$hyp\_at\_risk(z_2, n_1),$$
$$hyp\_vulnerability(double\text{-}kill, z_2).$$

Como antes, ϵ_1 puede ser aplicado y z_2 es mapeado al valor conocido creado en el paso previo: $z_2 = internet\text{-}explorer$. Observe que, dados los datos disponibles hasta este punto, se generan dos hipótesis respecto a dos diferentes productos que son vulnerables a *double-kill*. Sin embargo, cuando se obtiene más información acerca del funcionamiento

de *double-kill*, resulta que *internet-explorer* sólo se utiliza como herramienta para descargar *malware*, explotando una debilidad particular en ese navegador. Esta sutil diferencia resalta la importancia de adoptar un enfoque de *human in the loop* (HITL) donde los expertos puedan trabajar apoyándose en herramientas automatizadas para obtener mejores resultados.

Conocimiento NetDiff. La Figura 5.6 solo incluye los detalles necesarios acerca del conocimiento NetDiff que es usado para aplicar la TGD ω_1. Por otro lado, la Figura 5.7 muestra una representación detallada del estado de G obtenido a partir del estado NetDiff final luego de la difusión. Cada rectángulo con bordes más finos definen el grafo con sus nodos y arcos, etiquetas disponibles y el valor de t_{max}; aquellos con bordes más gruesos y esquinas redondeadas representan hechos NetDiff. El conocimiento relativo a etiquetas con cotas [0,0, 1,0] no se representa explícitamente. La forma en que se alcanza ese estado NetDiff final es ilustrado a través de la Figura 5.8 describiendo el proceso de evolución de la red.

El estado inicial en el momento 0 es modificado por la regla R_2 en el momento 1 y, por lo tanto, la etiqueta *bel_dang*(*double-kill*) se actualiza en los nodos $node(n_1)$ y $node(n_2)$ en el paso 1 y 2, respectivamente. Luego, esta misma etiqueta se actualiza en el nodo $node(n_4)$ por la regla R_1 en el momento 2. El proceso finaliza cuando se aplica la regla R_3 en el momento 2, lo que provoca que la etiqueta global *dangerous*(*double-kill*) se actualice con la cota [0,75, 1,0] que se obtiene de las cotas de la etiqueta *bel_dang*(*double-kill*) en los nodos $node(n_3)$ y $node(n_4)$.

5.4.3. Un conjunto más completo de reglas

En la sección anterior, el foco estuvo puesto en cómo funciona el procedimiento *chase* con el conocimiento ontológico y de red, y el proceso de difusión para producir este último, para lo cual se utilizó un ejemplo inicial utilizando sólo una TGD y EGD NetDER. Sin embargo, en un entorno más realista, se necesita un conjunto de reglas más amplio. Ahora se presenta un conjunto más completo de TGDs y EGDs NetDER para ilustrar mejor las capacidades del enfoque.

(ω_2): $product(P) \rightarrow \exists V\ vendor(V) \wedge parent(V, P)$

(ω_3): $vendor(V) \rightarrow \exists S\ platform(S) \wedge parent(S, V)$

Intuitivamente, las reglas ω_2 y ω_3 declaran que todo producto (*product*/1) tiene asociado un vendedor (*vendor*/1) y todo vendedor (*vendor*/1) tiene asociado una plataforma (*platform*/1) respectivamente. Esto codifica un subconjunto simple de la jerarquía CPE proporcionada por MITRE.

(ϵ_2): $product(P_1) \wedge hyp\_vulnerability(T, P_1) \wedge product(P_2) \wedge co\_occur(T, P_2, RF) \wedge$
$(RF > \theta) \wedge vendor(V) \wedge parent(V, P_1) \wedge parent(V, P_2) \rightarrow P_1 = P_2.$

Esta EGD declara que si existe una hipótesis acerca de que T aprovecha vulnerabilidades (*hyp_vulnerability/2*) relativas al producto (*product*/1) P_1, existe otro producto (*product*/1) P_2 que es mencionado junto con T en publicaciones de foros (*co_occur*/3) con una frecuencia relativa determinada (este umbral debería ser más pequeño que el utilizado en ϵ_1) y estos productos tienen el mismo vendedor (*vendor*/1), entonces existen razones para creer que estos productos pueden ser los mismos.

(ϵ_3): $product(P) \wedge hyp\_vulnerability(T, P) \wedge vendor(V_1) \wedge$
$parent(V_1, P) \wedge co\_occur(T, V_2, RF) \wedge (RF > \theta) \rightarrow V_1 = V_2.$

Las siguientes reglas son similares a las mencionadas anteriormente con la diferencia que ahora el foco está puesto en vendedores y plataformas. La regla ϵ_3 declara que si hay una hipótesis acerca de que T aprovecha vulnerabilidades (*hyp_vulnerability/2*) relativas al producto (*product*/1) P, y hay un vendedor (*vendor*/1) V_2 que es mencionado junto con T en publicaciones de foros (*co_occur*/3) con una frecuencia relativa dada, entonces existen razones para creer que V_1 (el vendedor de P) y V_2 son el mismo.

(ϵ_4): $product(P) \wedge hyp\_vulnerability(T, P) \wedge vendor(V_1) \wedge parent(V_1, P) \wedge$
$platform(S_1) \wedge parent(S_1, V_1) \wedge co\_occur(T, S_2, RF) \wedge (RF > \theta) \rightarrow S_1 = S_2.$

Finalmente, la regla ϵ_4 declara que si hay una hipótesis acerca de que T aprovecha vulnerabilidades (*hyp_vulnerability/2*) relativas al producto (*product*/1) P y hay una plataforma (*platform*/1) S_2 que es mencionada junto con T en publicaciones de foros (*co_occur*/3) con una frecuencia relativa dada, entonces existen razones para creer que S_1 (la plataforma de P) y S_2 son el mismo.

Como puede verse al inspeccionar estas reglas, el proceso de ingeniería de conocimiento requerido para producirlas es complejo, posiblemente involucre la incorporación de expertos del dominio y enfoques semi-automáticos. En este punto, un aspecto que vale la pena

destacar nuevamente es que las TGDs y EGDs NetDER se utilizan como impulsores para la generación de hipótesis y su evaluación—así, por ejemplo, aunque ϵ_4 concluye que $S_1 = S_2$, la esencia de esta acción es simplemente asignar un valor conocido a un valor nulo generado previamente por ω_3; dado que el *chase* genera tantos de estos valores como sea necesario, por lo que las múltiples hipótesis de este tipo se pueden considerar a la vez o eventualmente descartar.

5.5. Sumario

En este capítulo, se han desarrollado los fundamentos teóricos para la instanciación de la arquitectura NetDER presentada en el Capítulo 4 con la intención de disponer de capacidades para la generación y evaluación automatizada de hipótesis direccionadas al apoyo en la toma de decisiones en sistemas basados en *human-in-the-loop*. Se investiga el problema de respuesta a consultas en este modelo, y se llega a un interesante conjunto de resultados que varían de la tratabilidad del tiempo polinomial a la indecidibilidad, dependiendo de las características que están disponibles. Finalmente, se desarrolla un caso de uso para ilustrar cómo el enfoque puede ser aplicado en un dominio de ciberseguridad para razonar sobre productos en riesgo basados en publicaciones de foros de la *Darknet*. En el capítulo siguiente se desarrolla una evaluación más profunda y empírica en un dominio de diseminación de noticias falsas, bots y botnets.

Capítulo 6

Evaluación empírica: Respuesta a consultas NetDER

Este capítulo tiene como principal objetivo realizar una evaluación empírica del desempeño de la arquitectura NetDER, presentada en el Capítulo 4, para *razonar sobre comportamiento malicioso en plataformas sociales*. Para lograr este objetivo se ha llevado a cabo la implementación de una versión de dicha arquitectura, para lo cual se tomaron en cuenta los fundamentos teóricos estudiados en el Capítulo 5 con el propósito de desarrollar el Módulo de Razonamiento Ontológico (ORM), el Módulo de Difusión de Red (NDM) y el Módulo de Respuesta a Consultas (QAM).

A partir de la implementación de NetDER obtenida, se lleva adelante su evaluación de desempeño focalizando la atención en tres tareas: (i) determinar quién es responsable de la publicación de un artículo de *noticia falsa*, (ii) detectar *usuarios maliciosos*, y (iii) detectar qué usuarios pertenecen a una *botnet* diseñada para difundir *noticias falsas*. Dada la dificultad de obtener datos adecuados con *ground truth*, también se desarrolla un *testbed* que combina *datasets* reales de *contenido malicioso* con redes de usuarios generadas sintéticamente y trazas completamente detalladas de su comportamiento a lo largo de una serie de puntos de tiempo. Se ha diseñado dicho *testbed* de manera que pueda instanciarse de acuerdo a diferentes tamaños y configuraciones de problemas. Adicionalmente, el código del *testbed* mencionado se encuentra disponible públicamente para ser utilizado en esfuerzos de evaluación similares.

En la siguiente sección se presentará el mencionado *testbed*, el cual es una parte fundamental para la evaluación experimental de la arquitectura NetDER que se busca lograr.

6.1. BadBot: Un testbed para difusión de publicaciones maliciosas por parte de actores individuales y botnets

Ahora se presentan los detalles de un *testbed*[1] diseñado para generar *datasets* con toda la información necesaria para evaluar el desempeño en tres diferentes tareas vinculadas a la detección de *comportamiento malicioso en plataformas sociales*:

1. Determinar quién es responsable por la divulgación inicial de una publicación maliciosa; esta tarea es referida como Responsible.

2. Determinar quién es un actor malicioso—en este escenario, *malicioso* hace referencia a actores que difunden intencionalmente publicaciones maliciosas; esta tarea es llamada Malicious.

3. Determinar quién es un miembro de una *botnet* diseñada para difundir publicaciones maliciosas—aquí, *botnets* son conjuntos coordinados de nodos que publican el mismo contenido (el cual puede ser malicioso o no) al mismo tiempo. Esta tarea es llamada Member.

Observe que esta metodología general puede ser adoptada para crear *testbeds* similares diseñados para evaluar otros conjuntos de tareas. Éste es, hasta donde se conoce, el primer esfuerzo en crear un *testbed* para la evaluación de tareas que detecten más de un tipo de *comportamiento malicioso* a la vez, lo cual es necesario para la evaluación de un sistema de soporte a decisiones como NetDER. El nombre para dicho *testbed* es "BadBot", el cual es una forma resumida de *Bad actors and Bots*. Entonces, se dispone de los siguientes componentes básicos:

- *post-dataset*: Un *dataset* de publicaciones etiquetado con *ground truth* de manera que pueda ser utilizado para distinguir contenido genuino de contenido *malicioso* (por ejemplo, noticias reales de aquellas que son falsas, o enlaces benignos de aquellos relacionados a *malware*). Cada publicación tiene una *categoría* asignada; por ejemplo, si las publicaciones son artículos de noticias, éstas pueden tener la categoría Deportes, Arte, Mundo, Política, etc.

[1] El código fuente para BadBot está disponible públicamente en `https://github.com/jnparedes/BadBot`.

Parámetro	Descripción
new-graph	Indicador para determinar si un nuevo grafo debería ser creado.
num_{nodes}	Número de nodos en el grafo $\mathcal{G}$.
num_{edges}	Número de arcos en el grafo $\mathcal{G}$.
t_{sim}	Número de puntos de tiempo en las trazas generadas.
$prop_{mal}$	Proporción de nodos *maliciosos* en $\mathcal{G}$.
num_{botnet}	Número de *botnets* a ser generadas.
$prob_{memb}$	Probabilidad de que un nodo *malicioso* sea un miembro de una *botnet*.
$prob_{nm\_post}$	Probabilidad de que un nodo *no malicioso* publique un artículo (malicioso o no) en cada punto de tiempo.
$prob_{m\_post}$	Probabilidad de que un nodo *malicioso* publique un artículo (malicioso o no) en cada punto de tiempo.
$prob_{b\_post}$	Probabilidad de que una *botnet* (es decir, todos sus miembros) publique un artículo (malicioso o no) en cada punto de tiempo.
$prob_{nm\_mal}$	Probabilidad de elegir una publicación *maliciosa* cuando un nodo no *malicioso* crea una nueva publicación.
$prob_{m\_mal}$	Probabilidad de elegir una publicación *maliciosa* cuando un nodo *malicioso* crea una nueva publicación.
$prob_{b\_mal}$	Probabilidad de elegir una publicación *maliciosa* cuando una *botnet* crea una nueva publicación.
$prob_{nm\_share}$	Probabilidad de que un nodo no malicioso comparta una publicación de sus vecinos.
$prob_{m\_share}$	Probabilidad de que un nodo malicioso comparta una publicación de sus vecinos.
$prob_{b\_share}$	Probabilidad de que una *botnet* comparta una publicación de sus vecinos.

Figura 6.1: Conjunto de parámetros del *testbed* BadBot.

- $\mathcal{G} = (V, E)$: Un grafo de nodos y arcos para representar una estructura de red social, donde cada nodo representa un usuario y cada arco una relación (tal como seguidor, amigo, admirador, etc.). Cada nodo en V tiene un indicador que establece si tiene una naturaleza *maliciosa* o no. Adicionalmente, cada nodo tiene asociado una distribución de probabilidad relativa a categorías de publicaciones, indicando sus preferencias.

- Una serie de *puntos de tiempo* iniciando en 0 y finalizando en t_{sim}. Dependiendo de la instanciación del *testbed*, cada punto de tiempo puede representar diferentes granularidades de tiempo, tales como una hora, un día, una semana, etc.—los valores de los parámetros en la Figura 6.1 deberían, por lo tanto, ser ajustados acordemente.

Metodología para la generación de trazas. El propósito principal de este *testbed* es generar *trazas* completas de acciones ocurriendo en diferentes tiempos, de acuerdo al conjunto de parámetros que caracterizan un escenario deseado, como se describe en la Figura 6.1. El proceso de generar trazas es como se detalla a continuación (cf. Sección 6.2 para un ejemplo de cómo es instanciado en el contexto de la evaluación experimental presentada en esta tesis):

1. Configuración inicial:

   a) Si *new-graph* = *true*, crear el grafo $\mathcal{G} = (V, E)$ con $|V| = num_{nodes}$ y $|E| = num_{edges}$.

   b) Para cada nodo en $|V|$, se configura el indicador de *malicioso* de acuerdo a $prop_{mal}$.

   c) Para cada nodo malicioso en $|V|$, se configura su indicador de *botnet* de acuerdo a $prob_{memb}$.

   d) Inicializa *trace* como una estructura de datos vacía que mapea puntos de tiempo y nodos a publicaciones.

2. Por cada $time_i$ desde 0 hasta t_{sim}:

   a) Cada nodo *no malicioso*, *malicioso* y *botnet* decide de acuerdo a $prob_{nm\_post}$, $prob_{m\_post}$ y $prob_{b\_post}$, respectivamente, si creará una nueva publicación o no.
   Si se decide crearla, se elige entre *malicioso* o benigno de acuerdo a $prob_{nm\_mal}$, $prob_{m\_mal}$ y $prob_{b\_mal}$, respectivamente; y luego, se selecciona una publicación de *post-dataset* de acuerdo al resultado de la elección mencionada antes y sus preferencias de categorías.

   b) Si una nueva publicación no es creada, el nodo *no malicioso*, *malicioso* o *botnet* en cuestión decide de acuerdo a $prob_{nm\_share}$, $prob_{m\_share}$ o $prob_{b\_share}$, respectivamente, si compartirá algo publicado por sus conexiones, eligiendo entre aquellas que tienen la misma categoría que la categoría *dominante* de las publicaciones hechas por sus vecinos.

   La categoría dominante es determinada calculando primero la categoría más frecuente de las publicaciones de sus vecinos, y luego, tomando la más frecuente entre dichas categorías.

c) Registrar en *trace* toda la información producida, incluidas las publicaciones y sus autores relativas al tiempo $time_i$.

3. Devolver como salida *trace*.

En la siguiente sección se describirán los detalles acerca de cómo se llevará adelante la evaluación experimental.

6.2. Diseño de una evaluación experimental

En esta sección, se comienza describiendo los detalles acerca de cómo se instancian el *testbed* BadBot y la arquitectura NetDER para llevar a cabo una evaluación empírica de esta última (Sección 6.2.1 y 6.2.2), luego se discuten las particularidades de cómo las evaluaciones de desempeño se llevan a cabo (Sección 6.2.3) y finalmente se presentan los resultados obtenidos (Sección 6.3).

6.2.1. Configuración Básica: Instanciación del testbed BadBot

El dominio de aplicación es instanciado basado en el *testbed* BadBot, eligiendo valores para los diferentes parámetros y datos con el objetivo de crear un entorno realista. El conjunto de publicaciones está comprendido de artículos de noticias basados en una selección de 10,000 elementos de [Wan17]; debido a que el *dataset* no incluye categorías, se asigna aleatoriamente una de cinco categorías posibles (A, B, C, D, or E) creadas sintéticamente para cada artículo. Con el propósito de proveer estructura a la asignación aleatoria de categorías para artículos de *noticias falsas*, se asigna la categoría A con probabilidad 0,7, y las categorías restantes se asignan con probabilidad $0{,}3/4 = 0{,}075$, mientras que para noticias verosímiles la asignación es hecha a través de una distribución uniforme.

El grafo de la red social $\mathcal{G} = (V, E)$ es generado aleatoriamente basado en el algoritmo R-MAT [CZF04, KGB15][2], el cual está diseñado para crear instancias generadas aleatoriamente que imitan la estructura con frecuencia observada en datos del mundo real. Para esta evaluación, se configura $num_{nodes} = 150$ y $num_{edges} = 495$ para construir cada instancia, buscando lograr un buen equilibrio entre el tamaño, la densidad y el costo

[2]Se utilizó la implementación disponible en:`https://github.com/farkhor/PaRMAT`.

computacional de realizar muchas ejecuciones. Para definir el alcance temporal de cada traza, se configura $t_{sim} = 15$, lo cual se basa en una granularidad de 12 horas correspondientes a aproximadamente una semana de actividad de publicaciones. Finalmente, otros parámetros, que para esta evaluación se consideran fijos, son configurados de la siguiente forma[3]:

$$
\begin{aligned}
num_{botnet} &= 1 \\
prob_{nm\_post} &= 0{,}05 \\
prob_{m\_post} &= 0{,}5 \\
prob_{b\_post} &= 0{,}5 \\
prob_{nm\_mal} &= 0{,}1 \\
prob_{m\_mal} &= 0{,}6 \\
prob_{b\_mal} &= 0{,}6 \\
prob_{nm\_share} &= 0{,}2 \\
prob_{m\_share} &= 0 \\
prob_{b\_share} &= 0
\end{aligned}
$$

Adicionalmente, un conjunto de otros parámetros fueron variados con la intención de ampliar el alcance de la evaluación, creando seis diferentes configuraciones, las cuales se nombran con los símbolos A–F; ver Figura 6.2 para más detalles (la columna "Variante de r_1" es explicada en la siguiente subsección).

En la Sección 6.2.3, se detalla cómo BadBot es efectivamente usado para generar las trazas que van a servir como entrada de datos para la arquitectura NetDER en el contexto del escenario experimental definido.

[3]Observe que usuarios *maliciosos* y *botnets* son mucho más activos que aquellos que son *no maliciosos*. Se diseñó de esta forma suponiendo que usuarios *maliciosos* y *botnets* publican sólo contenido nuevo (no comparten contenido), por lo que se configura $prob_{m\_share} = 0$ y $prob_{b\_share} = 0$, y buscan difundir tantos artículos de *noticias falsas* como sea posible. Con estas configuraciones, la probabilidad de que un usuario *malicioso* publique un artículo de *noticia falso* en cada punto de tiempo es $prob_{m\_post} * prob_{m\_mal} = 0{,}3$. Dicha probabilidad para *botnets* también es igual $prob_{b\_post} * prob_{b\_mal} = 0{,}3$, mientras que para un usuario *no malicioso* es $prob_{nm\_post} * prob_{nm\_mal} = 0{,}005$ (aunque este último aún podría compartir involuntariamente un artículo de *noticias falsas*).

Configuración	t_{max}	**Variante de** r_1	$prob_{memb}$	$prop_{mal}$
A	2	$r_{1,2}$	0,25	0,2
B	2	$r_{1,3}$	0,25	0,2
C	2	$r_{1,1}$	0,25	0,1
D	2	$r_{1,1}$	0,1	0,2
E	5	$r_{1,2}$	0,25	0,2
F	5	$r_{1,3}$	0,25	0,2

Figura 6.2: Valores de parámetros para cada configuración usada en la evaluación; los parámetros fijos son configurados como se describe en la Sección 6.2.1.

6.2.2. Instanciación de NetDER

Ahora se proporcionan los detalles de cómo la arquitectura general NetDER es implementada para posteriormente llevar a cabo su evaluación. El Módulo de Ingesta de Datos (DIM) tiene la tarea de crear los datos en cada tiempo para la base de datos ontológica y la base de datos de red (para una vista general de la arquitectura y sus módulos ver Figura 4.1), basado en el estado actual de la traza generada por el *testbed*. El esquema de la base de datos ontológica se define de la siguiente forma:

- *news*(N): N es un artículo de noticias (obtenido del *dataset* [Wan17], como se describió anteriormente);
- *category*(N, C): N se corresponde con la categoría C;
- *fn_level*(N, L): N se considera un artículo de *noticia falsa* con nivel de confianza L. Dicho valor es obtenido a partir de un clasificador externo basado en aprendizaje automatizado utilizando en esta evaluación la herramienta disponible en [Rog18];
- *early_poster*(U, N): U fue uno de los primeros usuarios en publicar el artículo N;
- *close*(U_1, U_2): los usuarios U_1 y U_2 publicaron los mismos artículos en el mismo tiempo.

La estructura de la red es obtenida directamente a partir de las trazas generadas por BadBot. Además, se tienen las etiquetas locales de nodo *pref_category*(A)–*pref_category*(E) (una por cada categoría de noticia) para representar la confianza de que

Reglas del Módulo de Razonamiento Ontológico: (*fn_level* encapsula el resultado de un clasificador basado en aprendizaje automatizado externo)

$r_{1,1}$: $news(N) \wedge category(N, C) \wedge \underline{fn\_level}(N, L) \wedge (L > 0{,}2) \rightarrow hyp\_fakenews(N) : (\langle trending(C), [0{,}3, 1]\rangle)$

$r_{1,2}$: $news(N) \wedge category(N, C) \wedge \underline{fn\_level}(N, L) \wedge (L > 0{,}2) \rightarrow hyp\_fakenews(N) : (\langle trending(C), [0{,}5, 1]\rangle)$

$r_{1,3}$: $news(N) \wedge category(N, C) \wedge \underline{fn\_level}(N, L) \wedge (L > 0{,}2) \rightarrow hyp\_fakenews(N) : (\langle trending(C), [0{,}7, 1]\rangle)$

r_2 : $news(N) \wedge \underline{fn\_level}(N, L) \wedge L > 0{,}5 \rightarrow hyp\_fakenews(N)$

r_3 : $hyp\_fakenews(N) \wedge early\_poster(UID, N) \rightarrow hyp\_is\_resp(UID, N)$

r_4 : $hyp\_is\_resp(UID, N_1) \wedge hyp\_is\_resp(UID, N_2) \wedge (N_1 \neq N_2) \rightarrow hyp\_malicious(UID)$

r_5 : $hyp\_malicious(UID_1) \wedge hyp\_malicious(UID_2) \wedge closer(UID_1, UID_2) \rightarrow$
$\exists B\ hyp\_botnet(B) \wedge member(UID_1, B) \wedge member(UID_2, B)$

r_6 : $hyp\_fakenews(N) \wedge posted(UID, N) \rightarrow hyp\_is\_resp(UID, N)$

r_7 : $hyp\_is\_resp(UID, N) \rightarrow hyp\_malicious(UID)$

r_8 : $hyp\_malicious(UID_1) \wedge hyp\_malicious(UID_2) \wedge (edge(UID_1, UID_2)) \rightarrow$
$\exists B\ hyp\_botnet(B) \wedge member(UID_1, B) \wedge member(UID_2, B)$

r_9 : $member(UID, B_1) \wedge member(UID, B_2) \rightarrow B_1 = B_2$

Reglas del Módulo de Difusión de Red:

$diff\_rule_1$: $pref\_category(Category)_{\mathsf{T}} \overset{1}{\leftarrow} \mathsf{T}, \langle pref\_category(Category), [1, 1]\rangle, if_1$

$diff\_rule_2$: $trending(Category) \leftarrow (pref\_category(Category)_{\mathsf{T}}, avg)$

$$if_1(neigh, qua\_neigh, net\_state) = \begin{cases} [1, 1] \text{ si } |qua\_neigh|/|\{v \text{ s.t. } v \in neigh \wedge (v, \langle L, [1, 1]\rangle) \in net\_state\}| > 0{,}5 \\ [0, 1] \text{ en otro caso} \end{cases}$$

La función de agregación avg es definida como $avg(B) = [l, u]$ donde:

$$l = \sum_{[l_i, u_i] \in B}^{|B|} \frac{l_i}{|B|} \quad \text{y} \quad u = \sum_{[l_i, u_i] \in B}^{|B|} \frac{u_i}{|B|}$$

Variantes de KB: (cf. Figura 6.2 para mayores detalles respecto a variantes de r_1 de acuerdo a la configuración)

KB_α $= \{r_{1.x}, r_2, r_3, r_4, r_5, r_9, diff\_rule_1, diff\_rule_2\}$

KB_β $= \{r_{1.x}, r_2, r_6, r_7, r_8, r_9, diff\_rule_1, diff\_rule_2\}$

KB_{α^*} $= \{r_2^*, r_3, r_4, r_5, r_9, diff\_rule_1, diff\_rule_2\}$

(r_2^* es una modificación de r_2 donde el predicado *fn_level* encapsula el resultado de un clasificador perfecto.)

Consultas NetDER**:**

$Q_{resp}(Y, Z)$ $= hyp\_is\_resp(Y, Z) : [t_{max}, t_{max}]$

$Q_{mal}(M)$ $= hyp\_malicious(M) : [t_{max}, t_{max}]$

$Q_{memb}(UID)$ $= \exists B\ member(UID, B)\ \wedge hyp\_botnet(B) : [t_{max}, t_{max}]$

$Q_{fnA}(X)$ $= hyp\_fakenews(X) : [t_{max}, t_{max}]$

$Q_{fnB}(N)$ $= fn\_level(N, L)\ \wedge\ (L > 0{,}2) : [t_{max}, t_{max}]$

$Q_{fnC}(N)$ $= fn\_level(N, L)\ \wedge\ (L > 0{,}5) : [t_{max}, t_{max}]$

Figura 6.3: Detalles principales de la instanciación de la arquitectura NetDER necesaria para su posterior evaluación. Ver texto para una descripción intuitiva de cada componente.

cada categoría sea la preferida por cada nodo. Estas etiquetas se actualizan en cada punto de tiempo de acuerdo a la actividad de publicación de los usuarios.

La Figura 6.3 brinda una descripción detallada acerca de cómo las partes principales de la arquitectura NetDER se instancian para llevar adelante la evaluación empírica. A continuación, se proporcionan detalles y descripciones intuitivas de cada parte; las reglas fueron escritas manualmente simulando un experto del dominio, lo cual en este caso es una tarea más simple de lo que podría ser en el mundo real debido a que se conocen las particularidades de cómo el *testbed* funciona. Sin embargo, observe que los modelos resultantes no son perfectos dada la complejidad y la naturaleza estocástica del entorno.

Reglas del Módulo Ontológico (viñeta ubicada en la parte superior de la Figura 6.3): Conjunto de reglas lógicas que conducen a la generación de hipótesis, las cuales luego resultarán en respuestas a consultas. Las reglas r_1–r_8 son *dependencias de generación de tupla* (TGDs) NetDER, mientras que r_9 es una *dependencia de generación de igualdad* (EGD) NetDER—los detalles del funcionamiento de estas reglas son proporcionados en el Capítulo 5.

El predicado subrayado *fn_level* encapsula el resultado de la llamada a un clasificador externo que, dado un artículo de noticia, retorna un nivel de confianza con el cual se cree que dicho artículo es una *noticia falsa*. Aunque en esta evaluación se utiliza la herramienta provista en [Rog18], ésta puede ser reemplazada por cualquier clasificador (o conjunto de clasificadores) si así se lo desea; más abajo se describe una variante de estas reglas que asume un clasificador perfecto con la intención de ver el efecto que esta componente tiene en el desempeño general.

• r_1 tiene tres variantes de acuerdo a diferentes umbrales. Intuitivamente, la regla declara que dado un artículo de noticia que el clasificador detecta como *falso* con nivel de confianza L, se crea una hipótesis de que dicho artículo es *noticia falsa*. La regla sólo aplica cuando la categoría del artículo es *trending* en la red (la intuición es que se asume que es más probable que las *noticias falsas* estén relacionadas con categorías *trending*). Las tres variantes se corresponden a diferentes umbrales para los niveles de confianza y *trending*. r_2 es una simplificación de r_1 que no considera categorías *trending*.

• r_3 genera hipótesis respecto a quién es responsable de difundir una *noticia falsa* basado en quién es identificado como uno de los primeros en realizar una determinada publicación

(*early_poster*/2). r_6 es una variante más simple que se aplica basada en cualquier autor de publicación (*posted*/2), no sólo en aquellos que son de los primeros.

• r_4 declara que si un usuario es responsable por al menos dos diferentes artículos de *noticia falsa*, entonces dicho usuario es *malicioso*. r_7 es una variante más simple que sólo requiere responsabilidad para un artículo.

• r_5 genera hipótesis acerca de la existencia de una *botnet* y su membresía siempre que dos usuarios publiquen el mismo contenido al mismo tiempo. r_8 es una variante que remplaza la utilización de publicaciones hechas al mismo tiempo con una conexión entre los usuarios (diseñado para ser una versión de más baja calidad de r_5, ya que ésta no es la forma en que funciona el *testbed*).

• Finalmente, r_9 simplemente declara que existe una única *botnet*.

Reglas del Módulo de Difusión de Red: La regla $diff\_rule_1$ es una regla local NetDiff, la cual realmente representa un conjunto de cinco reglas locales NetDiff básicas (*ground*)—una por cada categoría de noticias considerada para esta evaluación. Tales reglas locales NetDiff estiman qué categoría es la preferida por cada nodo actualizando las etiquetas locales de nodo obtenidas a partir de *pref_category*(*Category*)—los intervalos se utilizan como valores de puntos, donde $[0, 1]$ significa probabilidad incierta de ser preferido y $[1, 1]$ completamente cierto. Esto es hecho cuando la $diff\_rule_1$ es aplicada, es decir, los vecinos de un nodo objetivo prefieren la categoría *Category*; luego las etiquetas locales de nodo obtenidas a partir de *pref_category*(*Category*) son actualizadas de acuerdo a la función de influencia if_1. Para este caso, la función de influencia definida considera la relación entre los vecinos calificados (cumplen la condición de la regla local NetDiff) y los vecinos que han hecho alguna publicación, devolviendo la cota $[1, 1]$ si dicha proporción es mayor a 0,5 y devolviendo $[0, 1]$ en otro caso. Si la etiqueta local de nodo obtenida a partir de *pref_category*(*Category*) del nodo objetivo es actualizada con cota $[1, 1]$ entonces significa que la categoría *Category* probablemente sea la preferida por el nodo objetivo considerado, en el siguiente punto de tiempo.

La regla global $diff\_rule_2$—representando también un conjunto de 5 reglas globales básicas (*ground*)—simplemente calcula un promedio de los intervalos de las etiquetas obtenidas a partir de *pref_category*(*Category*) (específicamente el promedio de los límites inferiores de las cotas) y asigna estos valores a la etiqueta "trending" para la categoría

Category. El cálculo mencionado, es en realidad llevado a cabo por la función de agregación *avg* (ver Capítulo 5 para más detalles del funcionamiento).

Variantes de bases de conocimiento NETDER **(KBs)**: Se confeccionaron tres bases de conocimiento distintas con el objetivo de mostrar cómo diferentes esfuerzos de ingeniería de conocimiento afectan el desempeño general del sistema:

• KB_{α} está conformada por las primeras cinco reglas, las cuales fueron diseñadas para reflejar con mayor fidelidad la forma en que el mundo funciona (es decir, la forma en que los datos son generados por el *testbed* BADBOT).

• KB_{β} reemplaza r_3–r_5 con r_6–r_8, las cuales fueron diseñadas para ser versiones más débiles de aquellas a las que reemplaza.

• KB_{α^*} es una copia de KB_{α} excepto que el predicado *fn_level* encapsula el resultado de un clasificador perfecto (es decir, no sufre de falsos positivos ni falsos negativos). El objetivo de disponer de esta variante es medir el impacto del componente basado en aprendizaje automatizado del sistema, que es un módulo externo en esta evaluación.

Consultas: La Figura 6.3 (viñeta de la parte inferior) muestra seis consultas. Las primeras tres corresponden, respectivamente, a las tres tareas de detección principales que se desean llevar a cabo: RESPONSIBLE, MALICIOUS, y MEMBER. El segundo grupo de consultas están orientadas a detectar qué artículos se corresponden con *noticias falsas*; esto es una tarea secundaria, pero que tiene impacto en el resto de las tareas, que se describe brevemente en la Sección 6.3.

Observe que tales consultas se apoyan en las hipótesis generadas por las reglas del Módulo de Razonamiento Ontológico (*hyp_is_resp*, *hyp_malicious* y *hyp_botnet*). En el caso de las *botnets*, debido a que el foco de interés está puesto en sus miembros, se utiliza un cuantificador existencial para la *botnet* en sí misma y luego se apoya en el predicado *member* para obtener los resultados.

6.2.3. *Ground truth* y evaluación de desempeño

Una de las principales motivaciones detrás del diseño y uso de un *testbed* como BADBOT es la necesidad de tener acceso a *ground truth*, debido a que esto es la base para

calcular las métricas que permiten comparar qué tan bien funcionan las diferentes herramientas. En particular, se necesita ser capaz de determinar sin incertidumbre si una respuesta a una consulta es un verdadero positivo (VP, el elemento en cuestión es esperado como salida y es parte de la respuesta obtenida), un falso positivo (FP, el elemento en cuestión no es esperado como salida pero es parte de la respuesta obtenida), un verdadero negativo (VN, el elemento en cuestión no es esperado como salida y no es parte de la respuesta obtenida), o un falso negativo (FN, el elemento en cuestión es esperado como salida pero no es parte de la respuesta obtenida). A continuación, se clarifican los detalles más importantes respecto a *ground truth*:

- Un artículo de noticias se corresponde con una *noticia falsa* si el *dataset* lo clasifica como tal.
- Un usuario es *malicioso* si BadBot lo marcó como tal cuando se generó la traza.
- Un usuario u es *responsable* por un artículo de noticias n si n es *noticia falsa*, u publicó n y u es *malicioso*.
- Un usuario u es *miembro* de una *botnet* b si u fue marcado como tal por BadBot cuando se generó la traza.

Observe que el *ground truth* es definido en términos de una traza de BadBot determinada. El desempeño de una *KB* NetDER determinada en relación a una tarea de respuesta a consulta es, por lo tanto, definido en términos de los resultados básicos descriptos anteriormente. Debido a que es posible que la misma respuesta aparezca en dos (o más) puntos de tiempo, se cuenta cada respuesta *sólo una vez* (como VP, FP, VN, o FN). Sin embargo, cuando una respuesta es contabilizada como FN, puede convertirse en VP en algún punto de tiempo posterior, aunque nuevamente esto se realiza sólo una vez por cada respuesta. En resumen, se consideran respuestas que *no son dadas* como falsos negativos, pero la respuesta correcta podría aparecer cuando llega nueva información y, por lo tanto, convertirse en VP.

El siguiente algoritmo describe el flujo de trabajo general que se utilizó en estos experimentos:

1. Configurar $new\_graph = true$
2. Repetir #runs veces:

   a) Generar una traza usando BadBot con los parámetros dados;
   b) Por cada punto de tiempo desde 0 a t_{sim}:
      1) Llevar adelante la consulta usando la política NetDER *One-Shot*;
      2) Computar FPs, FNs, VPs, y VNs con respecto al *ground truth* y respuestas obtenidas en el paso previo;
   c) Calcular resultados para la traza;
   d) Configurar $new\_graph = false$
3. Calcular resultados para el experimento.

Este algoritmo general es utilizado para obtener todos los resultados reportados en la siguiente sección, usando las mismas trazas para evaluar las tres variantes de bases de conocimiento NetDER y focalizando en tres métricas de desempeño:

- *Precisión* y *Recall*: Como es usual se define como $TP/(TP + FP)$ (fracción de respuestas obtenidas que son relevantes a la consulta) y $TP/(TP + FN)$ (fracción de respuestas relevantes que de hecho se obtienen), respectivamente.

  En general, la medida F1 también se informa junto con la precisión y el *recall* (calculada como su media armónica), pero en el entorno considerado hay muchos casos en los que no está definida debido a las divisiones por cero; aunque hay formas de solucionar este problema, ninguna de ellas llevó a un resultado satisfactorio ya que, en general, dichos resultados fueron artificialmente altos. Dado que los resultados numéricos absolutos no son el foco de esta evaluación y, en cambio, se desean mostrar los efectos de realizar modificaciones en el entorno y los modelos, se decide no informar esta métrica.

- *Tiempo de detección*: Número de puntos de tiempo en la traza entre la primera ocurrencia de un evento y su incorporación en una respuesta a una consulta. Por ejemplo, la diferencia de tiempos entre la publicación de un *artículo de noticia falsa* por un nodo *malicioso* y la incorporación del correspondiente átomo *hyp_is_resp*.

En la siguiente sección, el foco estará puesto en analizar los resultados de los experimentos realizados.

6.3. Resultados: Desempeño de tres modelos en seis configuraciones

Primero se discuten los resultados de las tres tareas principales que se abordan y luego se finaliza mencionando brevemente los resultados de la tarea auxiliar de detección de *noticias falsas*.

6.3.1. Tareas principales

A continuación se presentan y discuten los resultados obtenidos en relación a las tareas principales de esta evaluación experimental. La Figura 6.4 muestra la representación gráfica de los resultados de precisión y *recall* (ver Figura 6.8 para los correspondientes valores detallados), y la Figura 6.5 aquellos relativos al tiempo de detección (ver Figura 6.9 para los correspondientes valores detallados). Todos los gráficos corresponden a medias aritméticas y desvíos estándar obtenidos sobre 100 ejecuciones, donde toda la información del *testbed* es reiniciada al principio de cada ejecución excepto la estructura del grafo (esto es realizado como un esfuerzo para que las variaciones continúen siendo manejables). El tamaño de las barras de error asociadas con muchos de los resultados obtenidos indican que hay una cantidad no trivial de variación en el entorno, la cual es de esperarse en un problema tan complejo como el que se aborda aquí. Sin embargo, el número de ejecuciones realizadas permite obtener resultados estadísticos significativos en la gran mayoría de los casos, como puede verse en las Figuras 6.6 y 6.7, donde en cada caso "$**$" denota *altamente significativo* ($p < 0{,}01$), "$*$" *significativo* ($p < 0{,}05$), un valor explícito para $0{,}05 \leq p < 0{,}1$ y "*ns*" *no significativo* ($p \geq 0{,}1$).

En el resto de esta sección, se analizarán estos resultados desde diferentes perspectivas. Se comienza con comparaciones de alto nivel entre las variantes de bases de conocimiento NetDER consideradas, contextualizadas en cada una de las configuraciones.

***KB*$_{\alpha^*}$ vs. variantes restantes.** Como es de esperarse, tener acceso a un clasificador perfecto (uno que no tiene ningún falso positivo ni falso negativo) produce mejoras de desempeño notables en comparación con las variantes "regulares". Sin embargo, algunos resultados menos esperados fueron también obtenidos:

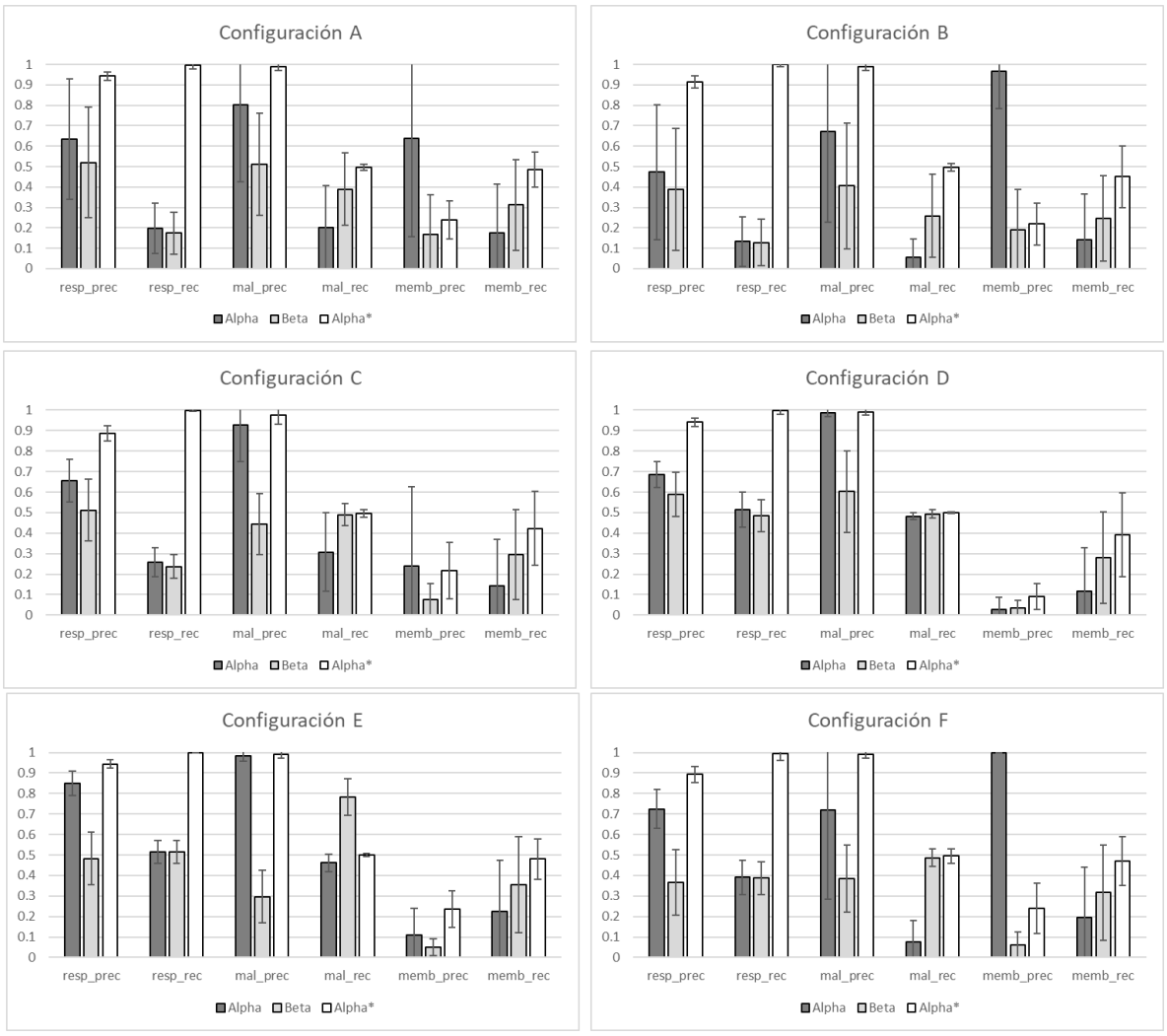

Figura 6.4: Gráficos relativos a la media aritmética de precisión y *recall* (valores más altos son mejores, barras de error representan desvío estándar) producida para KB_α, KB_β y KB_{α^*} respecto a las tres consultas en las Configuraciones A–F. (Resultados para 100 ejecuciones; cf. Figura 6.6 para resultados de significancia estadística).

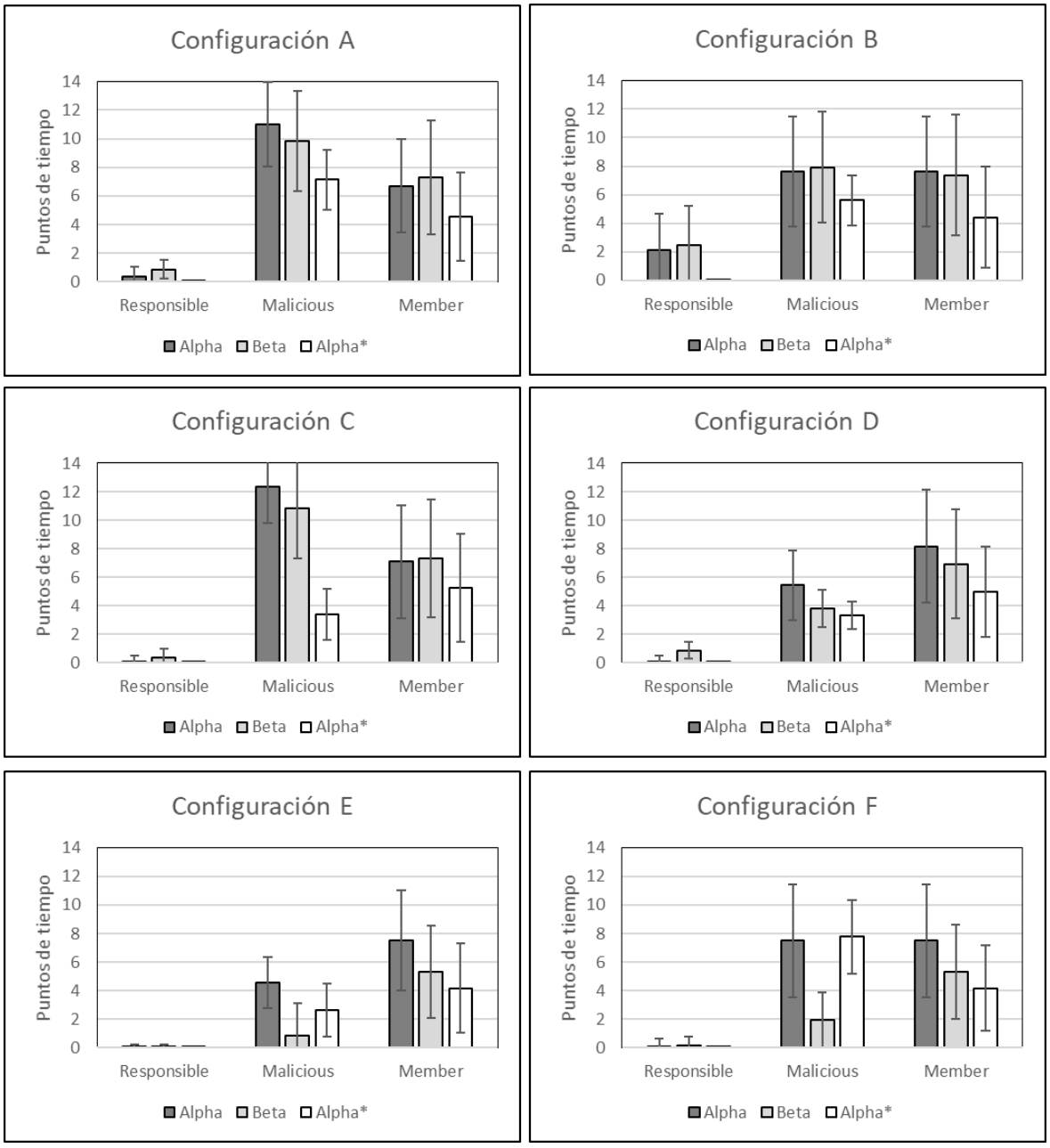

Figura 6.5: Gráficos relativos a la media aritmética de tiempos de detección (valores más bajos son mejores, barras de error representan desvío estándar) producida para KB_{α}, KB_{β} y KB_{α^*} respecto a las tres consultas en las Configuraciones A–F (Resultados para 100 ejecuciones; cf. Figura 6.7 para resultados de significancia estadística).

Configuración	**Consulta**		(KB_α, KB_β)	$(KB_\alpha, KB_{\alpha^*})$	$(KB_\beta, KB_{\alpha^*})$
A	Responsible	Precisión	**	**	**
		Recall	*ns*	**	**
	Malicious	Precisión	**	**	**
		Recall	**	**	**
	Member	Precisión	**	**	**
		Recall	**	**	**
B	Responsible	Precisión	$p = 0{,}063$	**	**
		Recall	*ns*	**	**
	Malicious	Precisión	**	**	**
		Recall	**	**	**
	Member	Precisión	**	**	*ns*
		Recall	**	**	**
C	Responsible	Precisión	**	**	**
		Recall	*	**	**
	Malicious	Precisión	**	*	**
		Recall	**	**	*ns*
	Member	Precisión	**	*ns*	**
		Recall	**	**	**
D	Responsible	Precisión	**	**	**
		Recall	**	**	**
	Malicious	Precisión	*ns*	**	**
		Recall	**	*	**
	Member	Precisión	**	**	**
		Recall	**	**	**
E	Responsible	Precisión	**	**	**
		Recall	*ns*	**	**
	Malicious	Precisión	**	*	**
		Recall	**	**	**
	Member	Precisión	**	**	**
		Recall	**	**	**
F	Responsible	Precisión	**	**	**
		Recall	*ns*	**	**
	Malicious	Precisión	**	**	**
		Recall	**	**	*ns*
	Member	Precisión	**	**	**
		Recall	**	**	**

Figura 6.6: Resultados para las pruebas t de Student con varianza desigual, dos muestras y dos colas respecto a todos los pares comparables de resultados de desempeño graficados en la Figura 6.4.

Configuración	Consulta	(KB_α, KB_β)	$(KB_\alpha, KB_{\alpha^*})$	$(KB_\beta, KB_{\alpha^*})$
A	Responsible	**	**	**
	Malicious	**	**	**
	Member	*	**	**
B	Responsible	**	**	**
	Malicious	*ns*	**	**
	Member	*ns*	**	**
C	Responsible	**	**	**
	Malicious	**	**	**
	Member	*ns*	**	**
D	Responsible	**	**	**
	Malicious	**	**	**
	Member	*	**	**
E	Responsible	$p = 0{,}08$	**	**
	Malicious	**	**	**
	Member	**	**	$p = 0{,}092$
F	Responsible	**	**	**
	Malicious	**	*ns*	**
	Member	**	**	**

Figura 6.7: Resultados para las pruebas t de Student con varianza desigual, dos muestras y dos colas respecto a todos los pares comparables de resultados de desempeño graficados en la Figura 6.5.

- Responsible: Precisión, *recall* y tiempos de detección casi perfectos a lo largo de todas las configuraciones.

- Malicious: Precisión casi perfecta a lo largo de todas las configuraciones, pero con *recall* alrededor de 0.5, indicando la presencia de falsos negativos. Tal vez la regla r_4 es demasiado restrictiva y algunos nodos maliciosos se pierden debido al requerimiento de tener dos átomos *hyp_is_resp* diferentes. Utilizar sólo uno (como lo hace r_7) puede ser una mejor opción considerando que el predictor de *noticias falsas* no comete errores.

  Respecto a los tiempos de detección, se observa peor desempeño para las Configuraciones E y F, las cuales se corresponden con procesos de difusión con mayor alcance temporal. Esto se debe a que los procesos de difusión con mayor alcance temporal influencian la detección de *noticias falsas* en KB_α y KB_β pero no tiene influencia en KB_{α^*} ya que el clasificador perfecto trabaja en forma aislada. Entonces, KB_α y KB_β generan una mayor cantidad de respuestas para las Configuraciones E y F, lo cual produce mayores posibilidades de detecciones más rápidas.

Conf.	KB_α				KB_β				$KB_{\alpha*}$			
	Precisión		*Recall*		*Precisión*		*Recall*		*Precisión*		*Recall*	
	avg	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*
A	0,64	0,29	0,20	0,12	0,52	0,27	0,17	0,10	0,94	0,02	1,00	0,02
B	0,47	0,33	0,13	0,12	0,39	0,30	0,13	0,11	0,91	0,03	1,00	0,01
C	0,66	0,10	0,26	0,07	0,51	0,15	0,24	0,06	0,89	0,04	1,00	0,00
D	0,68	0,06	0,51	0,09	0,59	0,11	0,49	0,08	0,94	0,02	1,00	0,02
E	0,85	0,06	0,52	0,06	0,48	0,13	0,51	0,06	0,94	0,02	1,00	0,00
F	0,72	0,10	0,39	0,08	0,37	0,16	0,39	0,08	0,89	0,04	1,00	0,03

Conf.	KB_α				KB_β				$KB_{\alpha*}$			
	Precisión		Recall		Precisión		Recall		Precisión		Recall	
	avg	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*
A	0,80	0,38	0,20	0,21	0,51	0,25	0,39	0,18	0,99	0,02	0,50	0,01
B	0,67	0,45	0,05	0,09	0,40	0,31	0,26	0,20	0,99	0,02	0,50	0,02
C	0,93	0,18	0,31	0,19	0,44	0,15	0,49	0,06	0,98	0,04	0,50	0,02
D	0,99	0,02	0,48	0,02	0,60	0,20	0,49	0,02	0,99	0,02	0,50	0,00
E	0,98	0,03	0,46	0,04	0,30	0,13	0,78	0,09	0,99	0,02	0,50	0,01
F	0,72	0,43	0,08	0,10	0,39	0,16	0,49	0,04	0,99	0,02	0,49	0,04

Conf.	KB_α				KB_β				$KB_{\alpha*}$			
	Precisión		Precisión		Precisión		*Recall*		*Precisión*		Recall	
	avg	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*
A	0,64	0,48	0,18	0,24	0,17	0,20	0,31	0,22	0,24	0,09	0,49	0,09
B	0,97	0,18	0,14	0,22	0,19	0,20	0,24	0,21	0,22	0,10	0,45	0,15
C	0,24	0,39	0,14	0,23	0,08	0,08	0,30	0,22	0,22	0,14	0,42	0,18
D	0,03	0,06	0,12	0,21	0,03	0,04	0,28	0,22	0,09	0,06	0,39	0,21
E	0,11	0,13	0,23	0,25	0,05	0,04	0,35	0,23	0,24	0,09	0,48	0,10
F	1,00	0,00	0,20	0,24	0,06	0,06	0,32	0,23	0,24	0,12	0,47	0,12

Figura 6.8: Resultados de precisión y *recall* en formato numérico para las tareas RESPONSIBLE (posición superior), MALICIOUS (posición media) y MEMBER (posición inferior) (cf. Figura 6.4). Media aritmética (*avg*) y desvío estándar (*stdev*) se calcularon sobre 100 ejecuciones.

- MEMBER: Mejor *recall* y tiempos de detección que las variantes restantes. Sin embargo, solo tiene mejor precisión en dos configuraciones, lo cual en un principio puede parecer contraintuitivo. A pesar de esto, considerando la interacción de factores, se puede llegar a la conclusión que—dependiendo de la configuración—el umbral de la etiqueta global en la regla r_1 es demasiado alto, o el t_{max} es demasiado bajo. Teniendo en cuenta que todos los miembros de la *botnet* publican el mismo contenido al mismo tiempo, las categorías de sus publicaciones tienen más probabilidades de empujar la cota de la etiqueta global por encima del umbral definido, incluso cuando el proceso de difusión es corto. Por otro lado, el proceso de difusión no tiene influencia en KB_{α^*}.

***KB_α* vs. *KB_β*.** La variante α fue diseñada para imitar mejor la forma en que las trazas son generadas por BADBOT, mientras que β fue diseñada pensando en utilizar versiones más débiles de varias reglas de α. Esto último conduce a peor precisión pero mejor *recall*, lo

Conf.	Tarea	KB_α		KB_β		KB_{α^*}	
		avg	*stdev*	*avg*	*stdev*	*avg*	*stdev*
A	Responsible	0,36	0,68	11	2,98	6,71	3,27
	Malicious	0,86	0,65	9,86	3,50	7,31	3,99
	Botnet Member	0,00	0,00	7,15	2,08	4,57	3,10
B	Responsible	2,09	2,60	7,64	3,86	7,64	3,86
	Malicious	2,48	2,71	7,92	3,87	7,38	4,22
	Member	0,00	0,00	5,61	1,76	4,42	3,55
C	Responsible	0,08	0,39	12,37	2,56	7,09	3,78
	Malicious	0,37	0,58	10,84	3,54	7,29	4,13
	Member	0,00	0,00	3,37	1,81	5,22	3,78
D	Responsible	0,05	0,45	5,42	2,44	8,16	3,97
	Malicious	0,86	0,58	3,8	1,33	6,93	3,85
	Member	0,00	0,00	3,31	0,98	4,97	3,18
E	Responsible	0,03	0,19	4,54	1,78	7,51	3,51
	Malicious	0,03	0,21	0,88	2,22	5,32	3,21
	Member	0,00	0,00	2,65	1,87	4,18	3,16
F	Responsible	0,09	0,56	7,51	3,95	7,51	3,95
	Malicious	0,13	0,67	1,92	1,95	5,32	3,30
	Member	0,00	0,00	7,78	2,59	4,18	2,98

Figura 6.9: Media aritmética de los resultados de *tiempo de detección* en formato numérico para las tareas Responsible, Malicious y Member (cf. Figura 6.5). La media aritmética (*avg*) y desvío estándar (*stdev*) fueron calculadas sobre 100 ejecuciones.

cual deriva en más falsos positivos pero menos falsos negativos, así como mejores tiempos de detección a lo largo de todas las configuraciones evaluadas.

A continuación, se consideran comparaciones más específicas, focalizando en la variación de los parámetros que dan origen a las diferentes configuraciones. La atención principal está puesta en el desempeño de KB_α con la intención de proveer conclusiones más específicas.

Configuración A vs. Configuración E. Estas dos configuraciones solo difieren en el alcance temporal del proceso de difusión (t_{max}):

- Responsible y Malicious: Procesos de difusión con mayor alcance temporal producen mejores resultados de precisión y *recall*, así como mejores resultados de tiem-

pos de detección.

- MEMBER: Para esta consulta, sucede lo inverso, un proceso de difusión con *menor* alcance temporal tiene un mejor desempeño. Esto probablemente está relacionado al análisis realizado en KB_{α^*} vs. el resto de variantes, en el cual un proceso de difusión con menor alcance temporal puede resultar en que las categorías de las publicaciones de miembros de la *botnet* se vuelvan las más frecuentes.

Configuración B vs. Configuración F. Estas dos configuraciones también difieren sólo en el alcance temporal de los procesos de difusión (t_{max}); comparado con el par previo, estas dos configuraciones usan una variante de la regla r_1 con un umbral más alto, lo cual significa que dicha regla es aplicada con menor frecuencia.

- RESPONSIBLE y MALICIOUS: Se produce la misma relación que en el par previo, pero mucho menos pronunciada para MALICIOUS.
- MEMBER: La precisión es similar (notablemente, ambos produjeron resultados casi perfectos) y el *recall* es ligeramente mejor para las configuraciones con procesos de difusión de mayor alcance temporal.

Los tiempos de detección no exhiben diferencias significativas entre estas dos configuraciones.

Configuración A vs. Configuración B. La única diferencia en este caso es la variante de regla r_1, como se mencionó en el par anterior (en B se define un umbral más alto):

- RESPONSIBLE y MALICIOUS: Se observa que la precisión y el *recall* son más altos para la variante con umbral más bajo. Con respecto al tiempo de detección, lo mismo es cierto para RESPONSIBLE, pero para MALICIOUS la variante con umbral más alto tiene mejor desempeño.
- MEMBER: La Configuración B claramente supera a la Configuración A en esta consulta; sin embargo, con respecto al tiempo de detección, la Configuración A tiene un desempeño ligeramente mejor.

Configuración C vs. Configuración D. Estas dos configuraciones difieren en que C tiene más miembros de *botnet*, pero menos usuarios maliciosos que D:

- Responsible y Malicious: La Configuración D generalmente supera a la Configuración C en ambas consultas, especialmente con respecto a *recall*. Con respecto a tiempos de detección, la única diferencia recae en Malicious, para la cual D funciona mucho mejor.
- Member: Lo inverso es cierto para esta consulta, en la cual C tiene mejor desempeño (mayormente en términos de precisión). Los tiempos de detección son también ligeramente mejores en la Configuración C.

Otras observaciones destacables. En general, se puede observar que la tarea más difícil es Member, la cual produjo resultados de precisión y *recall* por debajo de 0.5 en todas las configuraciones incluso cuando se tiene acceso a un predictor perfecto; es posible mencionar dos razones para dicha situación. Primero, la regla que es específica para esta tarea no es la mejor debido a que se focaliza en pares de usuarios; otras alternativas deberían ser exploradas, tal vez considerar grupos más grandes. Segundo, la información obtenida a partir del proceso de difusión es útil para esta consulta particular y KB_{α^*} no la utiliza; incorporando reglas que analicen los temas más difundidos puede llevar a obtener un mejor desempeño. Otra observación general que se puede realizar es que cuando un predictor excelente está disponible, puede ser más útil adoptar reglas menos restrictivas.

Como comentario final, se puede mencionar que dependiendo del dominio de aplicación, la importancia de los falsos positivos versus menor tiempo de detección puede variar mucho, y esto debe ser tenido en cuenta. En general, se observa que KB_{α^*} tiene menos falsos positivos que KB_{β}, pero este último tiene menores tiempos de detección. Por ejemplo, si se busca identificar personas con antecedentes penales en la frontera, una tasa baja de falsos positivos es conveniente; por otro lado, si la tarea es predecir un ciberataque, normalmente hay un tiempo limitado antes de que ocurra y los falsos positivos pueden ser aprovechados para llevar a cabo una inspección manual de casos sospechosos, aunque también se debe considerar que una cantidad excesiva de alertas puede convertirse en un problema adicional.

Conf.	*fnA*				*fnB*				*fnC*			
	Precisión		*Recall*		*Precisión*		*Recall*		*Precisión*		*Recall*	
	avg	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*
A	0,36	0,17	0,21	0,10	0,27	0,09	0,33	0,10	0,36	0,22	0,12	0,08
B	0,35	0,19	0,17	0,10	0,23	0,08	0,32	0,11	0,34	0,21	0,13	0,09
C	0,36	0,11	0,22	0,08	0,25	0,07	0,31	0,09	0,42	0,17	0,16	0,07
D	0,42	0,11	0,30	0,08	0,30	0,08	0,39	0,10	0,48	0,13	0,23	0,07
E	0,41	0,12	0,35	0,08	0,31	0,09	0,40	0,10	0,61	0,16	0,30	0,08
F	0,35	0,14	0,26	0,10	0,23	0,08	0,33	0,12	0,47	0,19	0,22	0,09

Figura 6.10: Resultados para la tarea de detección de *noticias falsas*.

6.3.2. Tarea secundaria: Detección de noticias falsas

Para esta tarea se busca identificar cuáles de los artículos publicados se corresponden con *noticias falsas*. Aunque esto se realiza principalmente a través del resultado encapsulado por el predicado *fn_level* (lo cual, como ya se mencionó, resulta simplemente de la invocación de un clasificador basado en aprendizaje automatizado), se implementaron cuatro diferentes reglas NetDER (las tres variantes de r_1, más r_2), así como tres diferentes consultas (Q_{fnA}, Q_{fnB} y Q_{fnC}) diseñadas para mostrar cómo la variación de diferentes umbrales y aspectos de la derivación lógica impactan en los resultados. Esta tarea de detección de *noticias falsas* es realizada de tres formas diferentes denominadas como *fnA*, *fnB*, y *fnC* (correspondientes a las consultas Q_{fnA}, Q_{fnB} y Q_{fnC}, respectivamente) y sus resultados en relación a precisión y *recall* son mostrados en la Figura 6.10. En resumen, la consulta Q_{fnA}—la cual está basada en un umbral bajo de una regla NetDER más otra regla que aprovecha el acceso al proceso de difusión de la red subyacente—obtiene un resultado entre la alternativa más simple y la que tiene mayor nivel de sofisticación (Q_{fnB} y Q_{fnC} simplemente encapsulan el resultado del clasificador y aplican un umbral más bajo y alto, respectivamente).

Cabe destacar nuevamente que el objetivo de este esfuerzo no es obtener las métricas de rendimiento más altas posibles para la detección de noticias falsas (o cualquier otra tarea), sino mostrar cómo los diferentes aspectos del sistema afectan el rendimiento. Anteriormente se vio cómo tres variantes diferentes del sistema NetDER—incluida una con acceso a un clasificador "oráculo" perfecto—se desempeñaron en otras tareas de detección relacionadas, lo cual agrega más complejidad al momento de abordar el problema.

6.4. Sumario

Para el desarrollo de este capítulo, se llevó adelante la implementación y prueba de una versión de la arquitectura NetDER presentada en el Capítulo 4 tomando en consideración los fundamentos teóricos estudiados en el Capítulo 5. El tipo de problemas de interés (como detectar comportamiento malicioso en plataformas sociales) para la mencionada arquitectura representan un gran desafío no sólo porque implica una familia de problemas específicos relacionados sino porque es prácticamente imposible obtener *datasets* adecuados con *ground truth*, lo cual es necesario para llevar adelante evaluaciones de desempeño. Teniendo en cuenta estas dificultades, fue necesario primero desarrollar un *testbed* general (dejando disponible públicamente su código) diseñado con el propósito de generar trazas completas de actividades de publicación involucrando potencialmente todo tipo de contenido malicioso como lo pueden ser *noticias falsas*, actores *maliciosos*, *botnets*, enlaces a *malware*, discursos de odio, etc. Esto fue el paso previo para lograr el objetivo principal que es realizar una evaluación empírica extensiva de tres variantes relativas a bases de conocimiento NetDER contextualizadas en seis configuraciones de entorno diferentes, lo cual a su vez busca develar los efectos de la calidad de las tareas de ingeniería de conocimiento, la calidad de las herramientas de aprendizaje automatizado subyacentes usadas para detectar comportamiento malicioso (particularmente *noticias falsas* en este caso) y las condiciones específicas del entorno asociado a los problemas de interés.

Capítulo 7

Trabajos relacionados

En este capítulo se discutirán diferentes investigaciones enfocadas en distintas direcciones pero vinculadas de alguna manera al trabajo desarrollado y presentado en esta tesis. Para comenzar, teniendo en cuenta que el objetivo general de esta investigación está vinculado a la generación automática de hipótesis se realiza una discusión breve en relación a los trabajos de mayor relevancia en esa dirección. En segundo lugar, también se discuten brevemente los trabajos más cercanos en relación a la utilización de lenguajes ontológicos, de manera similar a la dirección que sigue esta investigación. Una vez completadas estas discusiones preliminares, en la Sección 7.1 se discuten líneas de trabajos relacionadas a *problemas fundacionales relativos a integración* desde el punto de vista del área de Razonamiento y Representación de Conocimiento (KR&R) y en la Sección 7.2 se discuten diferentes trabajos disponibles en la literatura que abordan problemas asociados a *comportamiento malicioso en plataformas sociales*.

Abducción basada en lógica y generación automatizada de hipótesis. El razonamiento abductivo—el proceso de encontrar explicaciones a partir de observaciones—fue introducido en el siglo XIX por el filósofo Charles S. Peirce [Pei40] como una forma distinta de razonar en relación a las conocidas con los nombres de deducción e inducción. Las inferencias abductivas son mejor interpretadas como hipótesis que más adelante pueden ser confirmadas o descartadas. En inteligencia artificial, el razonamiento abductivo encuentra naturalmente su lugar como una herramienta que complementa los enfoques deductivos e inductivos en una amplia gama de aplicaciones tales como diagnosis [CT91, Poo89], razonamiento no monótono [EG95], planeamiento [Sha00, RSS$^+$15] y programación lógi-

ca [EGL97, KMM00]. También, relacionado al enfoque presentado en los desarrollos de esta tesis, se pueden mencionar trabajos que combinan razonamiento basado en lógica y razonamiento basado en probabilidades—algunos ejemplos bien conocidos de tales trabajos son el sistema Teórico de David Pool [PGA87], la Lógica de Elección Independiente [Poo97], la lógica probabilística [Nil86] y la programación lógica probabilística [NS92]. Aunque éstos y otros trabajos clásicos han sentado las bases de los desarrollos presentados en esta tesis, existen importantes diferencias: la invención de valores no se considera (que es un aspecto fundamental del razonamiento sobre objetos desconocidos), el manejo de la incertidumbre suele estar restringido por modelos o supuestos específicos, como la independencia por pares, y la difusión en redes no es una función incorporada.

Extensiones de Lenguajes Ontológicos. Parte del trabajo presentado en esta tesis está apoyado en las contribuciones hechas en extensiones a lenguajes ontológicos tales como aquellos basados en lógicas descriptivas (DLs) [BCM$^+$03, BHLS17] y reglas existenciales (también conocidas como *Datalog+/–*) [BCM$^+$03, BHLS17] los cuales son dos conocidas familias de formalismos de representación de conocimiento. En este sentido, el trabajo de [GLMS13] es uno de las más cercanos, en el cual los autores extienden a las reglas existenciales utilizando anotaciones probabilísticas con la intención de avanzar hacia el objetivo de obtener formas, con principios bien definidos, de llevar a cabo tareas de respuesta a consultas bajo incertidumbre e inconsistencia. Similarmente a la implementación del *chase* NETDER presentado aquí, los autores modifican el procedimiento *chase* para considerar la propagación de eventos probabilísticos a través de los átomos y reglas propias de la ontología. Adicionalmente, esta extensión también modela conocimiento incierto y ontológico por medio de dos mundos separados y sus interacciones. Sin embargo, aunque los eventos probabilísticos y las etiquetas locales y globales de red tienen similitudes debido a que ambos permiten manejar incertidumbre, la semántica y el objetivo general difieren principalmente en que las etiquetas utilizadas en este trabajo están especialmente diseñadas para dominios donde la información es propagada entre entidades que interactúan entre sí.

En este misma dirección también es posible mencionar el trabajo [Luk08], donde el autor presenta una extensión de $\mathcal{SHIF}(\mathbf{D})$ y $\mathcal{SHOIN}(\mathbf{D})$, dos lenguajes descriptivos expresivos que están relacionados a sublenguajes del *Lenguaje Ontológico Web* (*OWL*), conocidos como *OWL Lite* y *OWL DL*, respectivamente. P-$\mathcal{SHIF}(\mathbf{D})$ y P-$\mathcal{SHOIN}(\mathbf{D})$ son los lenguajes resultantes de estas extensiones permitiendo representar conocimiento

incierto y certero, ya sea en la forma de declaraciones terminológicas y/o asercionales respecto a instancias de conceptos y roles. Tanto en [GLMS13] como en [Luk08] se utilizan distribuciones de probabilidad sobre interpretaciones obtenidas a partir del conocimiento disponible para llevar a cabo tareas de respuesta a consultas probabilísticas.

Otro trabajo relacionado cercano es [GJLS17], en el cual se definen lógicas descriptivas por medio de extensiones de los sublenguajes $\mathcal{ALC}$ y $\mathcal{EL}$. Sin embargo, la propuesta de estos autores difiere de los trabajos mencionados anteriormente en que no sólo permite representar probabilidades con respecto a declaraciones de la *TBox* y aserciones de la *ABox*, sino que también permite hacerlo sobre conceptos y nombres de rol; una diferencia adicional es que es de naturaleza monótona.

7.1. Problemas fundacionales basados en integración de conocimiento

Teniendo en cuenta que una de las componentes de la arquitectura NetDER es el Módulo de Ingesta de Datos (DIM) y entre sus responsabilidades se encuentra la *integración de datos* (o el *intercambio de datos*), aquellos trabajos que aborden esta temática pueden considerarse cercanos a los desarrollos presentados en esta tesis. El mencionado problema puede categorizarse como clásico dentro del área de base de datos [Kol18, Mil18]. Sin embargo, en el contexto más específico de razonamiento y representación de conocimiento este problema se convierte en *mezcla de creencias* (*belief merging*) [KPP02, KPP11, FKIRS12], el cual es una generalización del problema de dinámica de creencias. En su forma general, la dinámica de creencias aborda el problema que se origina cuando un agente mantiene una base de conocimiento compuesta de un conjunto de fórmulas lógicas; cuando nueva información (también en la forma de fórmulas) está disponible, el agente debe decidir qué piezas de conocimiento (tal vez ninguna) necesitan ser cambiadas y qué modificaciones necesitan ser hechas a su base de conocimiento para evitar inconsistencias. Esto es generalmente afrontado a través del diseño y desarrollo de operadores que primero son descriptos teóricamente por medio de postulados (propiedades deseables) y luego se lleva adelante una implementación utilizando una caracterización algorítmica. Aunque para el caso de los desarrollos presentados en esta tesis algunos problemas se simplifican levemente, ya que la nueva información normalmente consistirá en las fórmulas más simples (llamadas átomos), la dinámica de creencias sigue siendo un

obstáculo a superar tanto desde el punto de vista computacional como del de gestión de datos.

Bajo el mismo contexto básico de integración de múltiples fuentes de información, claramente no se puede tener el mismo nivel de confianza en todas las fuentes. En este sentido [FGKIS13] aborda esta cuestión proponiendo operadores de revisión de creencias basados en razonamiento argumentativo, lo cual está basado en el análisis de razones a favor y en contra respecto a distintas afirmaciones con la intención de hacer el mejor uso de la información disponible. Otro ejemplo que se puede mencionar es [SMM+17], el cual se focaliza en respuesta a consultas ontológicas utilizando reportes subjetivos que varían en su confiabilidad y relevancia de acuerdo a la consulta. En dicho trabajo, los autores también aprovechan modelos simples de preferencias de usuarios como una forma de facilitar la caracterización de la confianza y relevancia de diferentes piezas de información; su propuesta está basada en la comparación entre las preferencias del usuario que realiza la consulta y aquellas del usuario que produce la información (en este caso, reseñas), asumiendo que los usuarios generalmente asignan un mayor grado de confianza a las opiniones de aquellas personas que están más alineadas con sus propios puntos de vista.

Cuando una base de conocimiento determinada es integrada o se realizan consultas sobre varias fuentes de información como si fueran una sola, y esto se lleva a cabo sin tener en consideración las consecuencias que pueden desencadenarse, surgen también otros problemas. En estos casos, pueden surgir tanto problemas de *incoherencia* como de *inconsistencia*. El primero de ellos hace referencia a situaciones en las cuales no hay forma posible de satisfacer las restricciones en la base de conocimiento—por ejemplo, considere las dos restricciones "*toda persona tiene una única dirección*" y "*toda persona debe tener al menos dos direcciones, una del trabajo y otra de su casa*" donde no existe una base de datos asociada que pueda satisfacerla a la vez. Por otro lado, la inconsistencia es un tipo de problema diferente que usualmente puede ser atacada identificando unos pocos elementos; por ejemplo, en el caso de las dos afirmaciones "*el salario de Federico es 50 mil pesos*" y "*el salario de Federico es 70 mil pesos*", asumiendo un dominio en el cual los salarios son únicos, la inconsistencia se podría resolver modificando sólo una de ellas. Generalmente se asume que la inconsistencia es causada por hechos y no por fórmulas más complejas. En tales casos, el problema puede ser atacado computando lo que se conoce como *reparaciones de datos* (*data repairs*), los cuales son subconjuntos inconsistentes

mínimos de la base de datos que pueden ser manipulados de diferentes formas para lograr un resultado consistente—cada una de tales reparaciones pueden ser interpretadas como una forma de abordar la inconsistencia. Incluso para este caso simple, el problema de computar respuestas de una manera cautelosa (esto es, aquellas que valen en todas las reparaciones de datos) ha sido demostrado coNP-completo, lo cual significa que el problema es verdaderamente difícil y resistente a soluciones computacionalmente tratables. Se refiere al lector a [LMS12] para un abordaje de estos problemas usando bases de conocimiento *Datalog+/–*, así como los trabajos [DMFS16, DMFS18] los cuales profundizan en el problema de incoherencia.

Finalmente, [GSM$^+$17] introduce el concepto de *bases de conocimiento de red* (*Network Knowledge Bases*), las cuales están diseñadas para representar múltiples plataformas sociales en un único grafo anotado donde cada nodo mantiene una base de conocimiento local y los *feeds* (las piezas de información vistas por cada usuario) son modelados por *ítems de noticias* (*news items*) que representan el origen, contenido y un indicador de si el usuario que publicó está llevando a cabo una operación de adición o remoción sobre su propia base de conocimiento. Este conocimiento puede ser utilizado para modelar cómo la información fluye a través de la red y ya se ha mostrado—en experimentos preliminares con modelos simples—ser útil en predecir reacciones de los usuarios en el contexto de contenido proveniente de la plataforma Twitter [GSMF19]. Adicionalmente, aunque el mencionado trabajo se basa en estudiar los fundamentos de cómo cada nodo decide qué creer de acuerdo al contenido visto por ellos en sus *feeds*, también podría utilizarse en el Módulo de Ingesta de Datos NetDER con el propósito de producir el modelo subyacente para el Módulo de Difusión de Red.

7.2. Comportamiento malicioso en plataformas sociales

Ahora se discuten trabajos que se han focalizado en afrontar diferentes tipos de *comportamiento malicioso en plataformas sociales*. Si bien existe una gran cantidad de material atacando esta problemática, en general, estas contribuciones están enfocadas en alguno de los problemas específicos dentro de esta clase general de comportamientos..

[WMW+13] sugirió al *crowdsourcing* como estrategia para detectar cuentas maliciosas, lo cual aprovecha la detección basada en personas distribuyendo tareas de inteligencia a usuarios de Internet que puedan identificar un patrón de anomalías exhibidas por parte de cuentas de alguna red social. El *crowdsourcing* implica el uso de un grupo grande y distribuido de trabajadores conocidos como trabajadores *crowd* para identificar comportamientos sospechosos. Los trabajadores *crowd* analizan las cuentas de las redes sociales comprobando la información de sus perfiles y deciden si las cuentas son *maliciosas* o *no maliciosas*. En el enfoque de *crowdsourcing*, se desarrolla una plataforma para los trabajadores *crowd*, de manera que puedan evaluar los perfiles de los usuarios y puedan tomar una decisión basada en el resultado del análisis realizado. Al aplicar *crowdsourcing* en dos plataformas sociales populares, Facebook y Renren, en [WMW+13] se observó que el desempeño de los trabajadores *crowd* contratados se reduce con el tiempo, aunque este método genera una ventaja ya que los votos de la mayoría se pueden utilizar para llegar a la conclusión final. Esta estrategia se considera adecuada para los proveedores de redes sociales, ya que demuestra una tasa de falsos positivos cercanos a cero. Sin embargo, una serie de inconvenientes dificultan la aplicabilidad de este método cuando se utiliza para detectar cuentas maliciosas:

- En [WMW+13] se afirma que la estrategia de *crowdsourcing* es eficaz si los proveedores de redes sociales lo adoptan en etapas iniciales. Esto muestra que el *crowdsourcing* incurrirá en un alto costo si se usa en redes sociales con una gran cantidad de usuarios preexistentes como Facebook y Twitter.

- Esta estrategia todavía requiere el conocimiento de expertos para garantizar una anotación confiable. Sin embargo, no todos los trabajadores *crowd* poseen el conocimiento experto necesario para producir una tasa de falsos positivos baja [WMW+13].

- Se debe tener mucho cuidado de no exponer los datos personales de los usuarios de las redes sociales a los trabajadores externos debido a que puede desencadenar en un problema de privacidad y esto puede incluso alentar a los trabajadores *crowd* a aprovecharse de usuarios determinados [WWZ+12].

- Existen varias plataformas de *crowdsourcing* maliciosas que utilizaron negativamente sus capacidades para controlar una gran cantidad de cuentas y obtener de manera fraudulenta una gran ganancia financiera [WWZ+12].

Otra posibilidad para detección de *comportamiento malicioso en plataformas sociales* es una estrategia basada en grafos. La interpretación de nodos y arcos en el grafo varía según el problema en cuestión y la técnica de modelado. Mientras que en algunos casos un arco puede representar una relación de amistad como podría esperarse de un grafo social obtenido a partir de datos de la plataforma Facebook, en otros casos podría representar una respuesta a una publicación [SASS18b]. La visualización de un grafo social puede ser utilizada para revelar los denominados *hubs*, que son usuarios con una gran cantidad de enlaces sociales. Los *hubs* tienen un gran potencial de interacción y comunicación dentro de la red [HKP12] y en muchas ocasiones son elegidos como blanco de ataques por parte de *usuarios maliciosos*.

Se puede pensar que dentro de los grafos sociales existen relaciones de confianza y, en general, éstas pueden dividirse en relaciones de confianza fuerte o débil. Los grafos sociales con confianza fuerte son aquellos que poseen la propiedad de mezcla rápida [MRR16, YKGF08]. En el problema de detección de cuentas *maliciosas*, esto puede verse como una red social con un "corte" de tamaño pequeño, que representa un conjunto de arcos que, cuando se elimina, dividen el grafo en dos regiones, usuarios *maliciosos* y *no maliciosos*. Los grafos sociales con relaciones de confianza fuerte tienen un número limitado de arcos de ataque entre las regiones de usuarios *maliciosos* y *no maliciosos*. Por el contrario, un grafo social con confianza débil no posee la propiedad de mezcla rápida. Otro supuesto similar a la mezcla rápida es el supuesto del "expansor" aleatorio utilizado para desarrollar el algoritmo Gatekeeper [TLSC11]. Asimismo, en [MYK10] se demostró que muchas redes sociales no se mezclan rápidamente, lo que indica que la cantidad de arcos de ataque en varias redes sociales puede ser de millones. Los arcos de ataque son los vínculos entre las regiones de usuarios *maliciosos* y *no maliciosos*. El problema de predicción de enlaces se puede usar para predecir tales arcos de ataque usando similitud de características o similitud en la estructura social [MRR16]. La primera medida de similitud considera los atributos de los nodos en el grafo social, mientras que la segunda sólo analiza el vínculo estructural que existe entre un par de nodos. Por ejemplo, la métrica de similitud propuesta por [AA03] es una de las métricas de similitud estructural más populares que se utilizan para predecir arcos de ataque [MRR16]. Dado que el objetivo del sistema de detección de cuentas maliciosas utilizando información del grafo social es identificar los nodos maliciosos, se ha demostrado que el problema de predicción de enlaces no funciona adecuadamente en una red social que exhibe una relación de confianza débil [MRR16]. Sin embargo, con el uso de una estrategia de propagación de confianza, es posible mejorar

la detección de cuentas maliciosas en plataformas sociales.

Una forma de llevar a cabo esta propagación de confianza es calcular una probabilidad de "aterrizaje normalizada en grados", la cual se asigna a cada nodo en el grafo social. Esta probabilidad corresponde a la probabilidad de que una caminata aleatoria modificada aterrice en cada nodo. La caminata aleatoria comienza desde un nodo conocido que no es malicioso, el cual distribuye su valor de confianza a los nodos vecinos. En cada paso del recorrido aleatorio, se calcula un rango de confianza, que indica la fuerza de las conexiones de confianza que existen entre los nodos. El paso de la distribución de probabilidad de la caminata aleatoria es un proceso de propagación de confianza. Es importante señalar que se puede hacer que una caminata aleatoria finalice en una etapa temprana; tal caminata aleatoria se denomina caminata corta. Una caminata aleatoria que se ejecuta durante un período prolongado producirá valores de rango de confianza uniformes para todos los nodos del grafo social. Este valor de confianza uniforme se conoce como el valor de convergencia del paseo aleatorio. La convergencia de la caminata aleatoria se basa en una serie de pasos conocidos como el tiempo de mezcla del grafo social [CSYP12, MRR16]. Uno de los algoritmos más populares para calcular el valor de confianza durante la caminata aleatoria es la iteración de potencia (*power iteration*) [CSYP12].

En el contexto de detección de usuarios maliciosos en plataformas sociales, el enfoque de caminata aleatoria se ha utilizado ampliamente. Por ejemplo, algoritmos como SybilGuard [YKGF08], Gatekeeper [TLSC11], SybilLimit [YGKX10], SybilRank [CSYP12] y SybilRadar [MRR16] utilizaron esta técnica para identificar nodos maliciosos, aunque la suposición hecha en SybilGuard, SybilLimit, SybilRank y Gatekeeper es bastante diferente de la suposición utilizada para desarrollar SybilRadar. El último algoritmo de detección de Sybil asumió que la red social exhibe una confianza débil debido a los hallazgos de diferentes experimentos, que indican que una cuenta maliciosa puede establecer gran cantidad de arcos de ataque a cuentas no maliciosas [ZL16]. Esto hace que SybilRadar detecte una gran cantidad de nodos maliciosos y supere a los algoritmos anteriores para la realización de la misma tarea, como SybilRank. Como ejemplo de cómo se usa la propagación de confianza, el proceso comienza con el administrador que identifica algunos nodos *no maliciosos* como semillas iniciales para el algoritmo. Luego, se usa una caminata aleatoria para calcular el valor de confianza, que es la probabilidad de aterrizaje para cada nodo en el grafo. Los valores de confianza se ordenan en forma descendiente para que los nodos de rango superior se coloquen en la parte superior [CSYP12]. Se ha demostrado que el

rendimiento del algoritmo de detección temprana de nodos maliciosos disminuye significativamente cuando aumenta el número de arcos de ataque [MRR16, VPGM10, YXY+16]. SybilRadar intenta mejorar el rendimiento de los primeros algoritmos de detección de Sybil mediante la introducción de una serie de etapas basadas en el análisis estructural social para refinar su rendimiento. Sin embargo, como afirman los investigadores, existe un límite para identificar de manera efectiva cuentas maliciosas con cambios temporales empleando solamente la estructura del grafo social.

Otro algoritmo que se basa en la propagación de confianza es VoteTrust [YXY+16], el cual aprovecha la aceptación de solicitudes de amistad entre cuentas de una red social. Este algoritmo aplica la iteración de potencia para calcular la probabilidad de confianza. VoteTrust se basa en la suposición de que un nodo malicioso puede detectarse mediante la aceptación de la solicitud de amistad de un nodo no malicioso. Una invitación de amigo entre pares de nodos se modela luego como un grafo dirigido con signo, donde un arco entre dos nodos toma el valor 1 o -1. Un valor de 1 en el arco indica que la solicitud de amistad es aceptada, mientras que -1 indica no aceptación. Por lo tanto, se dice que un nodo B emite voto sobre el nodo A, si B acepta o rechaza una solicitud de A. Una de las ventajas de VoteTrust es que el algoritmo exhibe un alto paralelismo en el procesamiento de grandes grafos sociales. Sin embargo, en algunas redes sociales (por ejemplo, Twitter), es posible lanzar un ataque sin necesariamente hacerse amigo de usuarios no maliciosos, lo cual de por sí limita la capacidad de VoteTrust para detectar comportamiento malicioso.

Otra manera de aprovechar la información del grafo social para detectar comportamiento malicioso en plataformas sociales es utilizar alguna estrategia de *clustering*, debido a que esta última es una característica que suelen tener los grafos sociales. Cuando se utiliza una estrategia de *clustering* sobre grafos se intenta agrupar un conjunto de nodos relacionados dentro del grafo en función de su similitud. Dos nodos se agrupan solo si están a una distancia específica entre sí. Los grupos resultantes luego de llevar a cabo el *clustering* se denominan clústeres, aunque también se los conoce como comunidades. El objetivo de un algoritmo de *clustering* basado en grafos es agrupar los nodos en clústeres considerando la estructura de los arcos del grafo de manera que incremente la cantidad de arcos dentro de cada grupo [Sch07]. Uno de los algoritmos de *clustering* sobre grafos más utilizados es *Markov cluster* (MCL), el cual recibe una matriz de transición de un grafo pesado. Al aplicar operaciones de "expansión" e "inflación", MCL agrupa los nodos de forma iterativa en el grafo y termina una vez que se obtiene una matriz estable. Los clústeres

resultantes se pueden analizar para detectar cuentas maliciosas [AA12]. En [AA12] extrajeron información correlacionada del perfil del usuario, como las URLs compartidas, la lista de amigos y los "me gusta" en páginas de Facebook con el propósito de generar una matriz ponderada para el algoritmo MCL. El resultado del algoritmo de *clustering* MCL produce tres clústeres. El primer clúster contiene cuentas clasificadas como perfiles de *spam*, el segundo perfiles normales y el tercero cuentas clasificadas como perfiles normales y de *spam*. Los autores aplicaron una técnica de voto mayoritario para resolver el tercer clúster con clases superpuestas. En [LMC+15] también se propuso un método basado en comunidades, el cual utiliza un proceso de dos pasos. El primer paso agrupa las cuentas en comunidades y el segundo paso asigna una etiqueta a cada cuenta en la comunidad según las características que exhiben las cuentas y la comunidad. Cuanto mayor sea el número de comunidades a las que pertenecen dos cuentas, mayor será su similitud.

Otra opción para detectar *comportamiento malicioso en plataformas sociales* es hacer uso de las propiedades inherentes a los grafos sociales tales como la distribución de la ley de potencia (*power law distribution*), la estructura topológica libre de escala (*scale-free topological structure*), la propiedad de mundo pequeño (*small-world*) y centralidades de los grafos [SS11]. La red libre de escala es una red con distribución de grados que sigue una ley de potencia [OSH+07]. Esto significa que la distribución de probabilidad del número de conexiones entre los nodos de la red sigue una distribución de la ley de potencia. Esta suposición también es válida para el coeficiente de *clustering*, la conectividad de vértice entre nodos y una longitud de camino promedio pequeña [SZM13]. Aunque se supone que algunas redes sociales del mundo real son libres de escala, esta suposición no ha sido generalizada a todas las redes sociales del mundo real. Las métricas de centralidad de los grafos miden la importancia relativa de cada nodo en el grafo social según la posición. Se supone que un nodo con un valor alto es más relevante; sin embargo, la definición de esta relevancia depende del dominio de aplicación del problema en consideración. La intermediación (*betweenness*) es una métrica de centralidad que determina la frecuencia con la que un nodo se ubica en el camino más corto entre otros nodos en un grafo social. La métrica representa los porcentajes de todos los caminos más cortos en una red que pasan por un nodo en particular [SS11]. Otras métricas de centralidad utilizadas en teoría de grafos incluyen cercanía, PageRank y centralidad de autovector.

En [SS11] se aplicó la métrica de intermediación para la detección de URLs de *phishing* con la intención de reducir los falsos positivos. Este enfoque se probó con datos con *ground*

truth disponible de aproximadamente 10,000 dominios seleccionados al azar de la lista negra y la lista blanca proporcionada por URIBL. Se construyó un grafo de enlace para cada dominio seleccionado y se calcularon los valores de intermediación. El estudio muestra que el valor de intermediación de los dominios de la lista blanca es notablemente más alto que el de la lista negra. La fortaleza de este enfoque es que proporciona una métrica poderosa y una herramienta eficaz que puede complementar los sistemas *antispam* basados en URLs, así como una reducción de falsos positivos. Como se observa en [DKFF15], uno de los problemas con el supuesto de las leyes de potencia tradicionales es la incapacidad para explicar los comportamientos desviados. Una extensión de la distribución de la ley de potencia llamada *PowerWall* fue utilizada para analizar las actividades de los usuarios en el muro de Facebook en un intento por capturar cuentas con *comportamiento malicioso*. Los autores analizaron las publicaciones de los usuarios en el muro e identificaron patrones de anomalías en las cuentas que publican la misma cantidad de mensajes cada dos días y otro usuario que publica cada noche sin otras actividades [DKFF15].

El aprendizaje automatizado (ML, por las siglas en inglés de *machine learning*) es uno de los enfoques con mayor auge en inteligencia artificial actualmente y, como es de esperarse, ha sido utilizado ampliamente para abordar el problema de detectar comportamiento malicioso en plataformas sociales [TRG19]. ML incorpora una variedad de métodos, como aprendizaje supervisado, no supervisado y semisupervisado. Los algoritmos de ML supervisado toman como entrada un *dataset* etiquetado y como resultado aprenden un modelo, el cual puede predecir la etiqueta de clase para nuevos datos [NFAG16]. Cuando se utiliza aprendizaje supervisado, el clasificador aprende de una gran cantidad de datos etiquetados para construir un modelo durante el entrenamiento. El aprendizaje no supervisado difiere en el sentido de que no hay datos etiquetados presentes durante la etapa de entrenamiento y el sistema aprende de los datos en sí, identificando relaciones o similitudes entre las instancias en el *dataset*. Debido a que el proceso de obtención de datos etiquetados es tedioso, un algoritmo semisupervisado toma pocos datos etiquetados además de una gran cantidad de datos no etiquetados para producir un modelo.

En el caso de aprendizaje supervisado, los datos se convierten en una serie de vectores de características (*features*) que consisten en un conjunto de valores para cada atributo [ZZC+15]. Al final del entrenamiento, se usa un modelo de clasificación para distinguir cuentas maliciosas y no maliciosas [SBS14]. Teniendo en cuenta que los métodos de ML supervisado se basan en datos etiquetados y en un conjunto de características, obte-

ner datos etiquetados no sesgados es muy difícil. Algunos estudios utilizan un enfoque manual para etiquetar sus datos [CGWJ12, MA13], al observar algunas características distintivas de los usuarios maliciosos, como seguir a gran cantidad de usuarios con menos seguidores, publicar mensajes no solicitados, publicar contenido duplicado, realizar redireccionamientos excesivos, publicar contenido que carece de inteligencia, etc. Otras alternativas para superar esta dificultad son la utilización de APIs de lista negra, *honeypots* [ARK12, CWW12, LCW10, YHG13], así como el uso de las cuentas suspendidas por los proveedores de redes sociales como en el caso del algoritmo de suspensión de Twitter [ASN+16, TGSP11]. En [SBS14] se aplicó un enfoque automatizado para el etiquetado de datos utilizando un modelo analítico con la ecuación de dos entidades: puntaje de usuario y puntaje de tweet. Este enfoque acotó los datos etiquetados a lo largo de cuatro características de interés: el número de seguidores, el número de tweets, el número de seguidos y la fecha de creación de la cuenta, y se definieron algunos umbrales para comparar las dos entidades. Según esta comparación, a la cuenta se le asigna una etiqueta de clase, que puede representar malicioso, no malicioso o celebridad. Otro trabajo que hace uso de aprendizaje supervisado para detectar *comportamiento malicioso en plataformas sociales* es [SASS18b]. En este caso se utiliza particularmente para predecir ciberataques a determinadas organizaciones, lo cual conlleva un análisis sobre la estructura de red social formada a partir de la información obtenida de foros de *hackers* en la *Darknet*.

Como se mencionó anteriormente, a diferencia del enfoque de aprendizaje automatizado supervisado, el aprendizaje no supervisado utiliza datos sin etiquetar para construir un modelo. Como tal, no se conoce a priori ningún comportamiento de ataque específico. El método no supervisado agrupa los datos en diferentes clases según sus características similares. El método intenta aprender de los datos al observar las similitudes entre instancias en *dataset*. El aprendizaje no supervisado es bastante útil en el análisis de patrones y para agrupar el spam social en campañas [LK14]. En [SGS18] se aplica una estrategia basada en causalidad utilizando aprendizaje no supervisado y se destaca que la propuesta de lo autores para la detección de *comportamiento malicioso en plataformas sociales* se realiza sin la utilización de información sobre la estructura de la red, del camino de las cascadas, el contenido publicado o la información del usuario. Debido a que de acuerdo a determinadas métricas de causalidad se determina cómo se realiza la propagación de información, se puede pensar que esta propuesta también utiliza una estrategia de propagación de confianza.

Por otro lado, los algoritmos de aprendizaje semisupervisado intentan identificar un modelo de clasificación adecuado combinando tanto datos etiquetados y no etiquetados. Debido a la dificultad de obtener datos etiquetados en la mayoría de los dominios de aplicación, como en el caso de las plataformas sociales, los algoritmos semisupervisados intentan aprender un modelo adecuado al permitir una pequeña cantidad de datos etiquetados con una gran cantidad de datos no etiquetados. Algunos algoritmos populares de aprendizaje semisupervisado incluyen la maximización de expectativas, el auto-entrenamiento, las máquinas de vectores de soporte transductivas (TSVM) y el co-entrenamiento [ZG09]. En [LXFZ13] se aplica el algoritmo TSVM para detectar ataques de *phishing* y se utilizan características (*features*) basadas tanto en imágenes como en modelo de objetos de documento (DOM) para entrenar el algoritmo TSVM. Asimismo, en [SSAS19] se proponen dos enfoques para detectar *comportamiento malicioso* relativo a cuentas de la plataforma Twitter. Uno de esos enfoques está basado en inferencia causal con una estrategia de aprendizaje semisupervisado, el cual además no se apoya en información de la estructura de la red, del camino de cascadas, del contenido publicado ni de la información de los usuarios. Por lo tanto, la detección se basa en encontrar usuarios maliciosos en las cascadas virales, dado que las cascadas virales son tan poco comunes que los usuarios que las provocan pueden considerarse sospechosos. Esto también se combina con métricas basadas en grafos (algunas de las cuales se han discutido anteriormente en esta sección). La estrategia básicamente consiste en comenzar con una pequeña cantidad de datos de entrenamiento etiquetados y luego agregar iterativamente usuarios con puntuaciones de confianza alta provenientes de los datos no etiquetados al conjunto de entrenamiento.

Para finalizar, también se pude mencionar el trabajo [NSS18] donde se emplea exitosamente una combinación de un formalismo basado en lógica con información proveniente de clasificadores (aprendizaje supervisado). Más específicamente, los autores buscan aprovechar el razonamiento basado en argumentación para reducir el conjunto de candidatos antes de realizar el procesamiento por parte del algoritmo de clasificación, lo cual está pensado con la intención de ayudar a bajar la ocurrencia de falsos positivos.

En el próximo capítulo se vinculará estas líneas de investigación y desarrollo con el trabajo presentado en esta tesis, como así también los planes para trabajos futuros.

Capítulo 8

Conclusiones y trabajo futuro

Como se ha presentado a lo largo de esta tesis, el objetivo general de esta línea de investigación es el desarrollo de *sistemas de generación automatizada de hipótesis*, motivados especialmente por encontrar soluciones a una serie problemas vinculados conocidos como *comportamiento malicioso en plataformas sociales*. Algunos de los problemas más conocidos categorizados como tales son: noticias falsas, discursos de odio, *clickbait*, *bots* y *botnets*, *trolls*, *ciber bullying*, contenido terrorista, polarización, actividades de *hacking*, deduplicación adversarial, entre otros. Varios de estos problemas particulares fueron discutidos en detalle en el Capítulo 1.

Con la intención de buscar avanzar hacia el mencionado objetivo general, en el Capítulo 3 se presentaron los primeros pasos en esta dirección, para lo cual el dominio de aplicación es acotado a un problema específico de comportamiento malicioso en plataformas sociales conocido como deduplicación adversarial. En esta primera propuesta, el problema en cuestión es abordado implementando un sistema de generación automatizada de hipótesis bajo dos enfoques diferentes: en el primero de ellos se pone mayor énfasis en la utilización de reglas lógicas combinadas con eventos probabilísticos, aunque también se busca que dichas reglas aprovechen conocimiento atómico proveniente del resultado de la aplicación de técnicas de aprendizaje automatizado de manera que estos dos mundos puedan complementarse; en el segundo enfoque el énfasis está puesto en la utilización de clasificadores (aprendizaje automatizado supervisado) y las hipótesis son generadas por reglas más simples que en el primer caso, las cuales se basan en que los resultados obtenidos por las herramientas de clasificación superen un determinado umbral.

De acuerdo a los trabajos relacionados discutidos en el Capítulo 7, se puede observar que, en general, hay muchas investigaciones que abordan la problemática de comportamiento malicioso en plataforma sociales; sin embargo, las mismas se encuentran focalizadas en alguno de los problemas particulares (por ejemplo, noticias falsas), utilizan enfoques ad-hoc lo cual hace difícil su generalización y la posibilidad de explicar los resultados. Asimismo no aprovechan el hecho de que algunos de estos problemas específicos se encuentran relacionados y podría utilizarse la resolución de uno de ellos para ayudar en la resolución de otro relacionado. Por esta razón, en el Capítulo 4, se presenta la arquitectura NetDER para razonar sobre problemas como los descriptos en el Capítulo 1, pero buscando lograr obtener herramientas más robustas que consideren cuestiones, como las mencionadas anteriormente, que son obviadas en los trabajos disponibles en la literatura, y que fueron discutidos en el Capítulo 7. La presentación de dicha arquitectura es realizada desde un punto de vista de ingeniería de sistemas, de manera que se proporcione una vista de diseño con la intención de guiar la implementación de software destinado a resolver problemas con las características que ya fueron mencionadas. NetDER cuenta con cuatro módulos principales: el Módulo de Ingesta de Datos, el Módulo de Razonamiento Ontológico, el Módulo de Difusión de Red y el Módulo de Respuesta a Consultas. La idea detrás de este enfoque es aprovechar la sinergia entre los beneficios del razonamiento ontológico que hace uso del conocimiento del dominio, los procesos de difusión involucrados en los problemas de interés y el conocimiento atómico que pueda obtenerse a partir de técnicas de aprendizaje automatizado.

En el Capítulo 5 se continuó el trabajo presentado en el Capítulo 4, pero en este caso realizando una implementación particular de NetDER y estudiando los fundamentos teóricos involucrados en el Módulo de Razonamiento Ontológico, el Módulo de Difusión de Red y el Módulo de Respuesta a Consultas. La implementación del primero de estos módulos se basó en una extensión de la familia de lenguajes ontológicos conocido como *Datalog+/–*, la cual entre otras ventajas proporciona la valiosa capacidad de razonar sobre objetos desconocidos a través de la invención de valores. Para la implementación del segundo módulo se realiza una ligera adaptación del formalismo MANCaLog, debido a que cumple una serie de propiedades deseables relativas a razonamiento en redes complejas. La implementación de este último módulo conlleva un interesante conjunto de resultados que varían de la tratabilidad del tiempo polinomial a la indecidibilidad, dependiendo de las características que estén disponibles. Asimismo, se desarrolla un caso de uso para ilustrar cómo el enfoque puede ser aplicado en un dominio de ciberseguridad para razonar sobre

productos en riesgo basados en publicaciones de foros de la *Darknet*.

En el Capítulo 6 se realizó una evaluación empírica de NetDER, para lo cual se requirió la implementación de una versión de dicha arquitectura tomando en cuenta los fundamentos teóricos estudiados en el Capítulo 5. Se debe considerar que el tipo de problemas de interés representan un gran desafío no sólo porque implican una familia de problemas específicos relacionados sino porque es prácticamente imposible obtener *datasets* adecuados con *ground truth*, lo cual es necesario para llevar adelante evaluaciones de desempeño. Teniendo en cuenta estas dificultades, fue necesario primero desarrollar un *testbed* general (dejando disponible públicamente su código) diseñado con el propósito de generar trazas completas de actividades de publicación involucrando potencialmente todo tipo de contenido malicioso como lo pueden ser noticias falsas, actores maliciosos, *botnets*, enlaces a *malware*, discursos de odio, etc. Esto fue el paso previo para lograr el objetivo principal que es realizar una evaluación empírica extensiva de tres variantes relativas a bases de conocimiento NetDER contextualizadas en seis configuraciones de entorno diferentes, lo cual a su vez busca develar los efectos de la calidad de las tareas de ingeniería de conocimiento, la calidad de las herramientas de aprendizaje automatizado subyacentes usadas para detectar comportamiento malicioso (particularmente *noticias falsas* en este caso) y las condiciones específicas del entorno asociado a los problemas de interés. Los resultados obtenidos fueron satisfactorios, debido a que en general los efectos evaluados son estadísticamente significativos y constituyen un paso importante para avanzar hacia el logro del objetivo general que es disponer de sistemas robustos de generación automatizada de hipótesis que puedan utilizarse para resolver problemas de comportamiento malicioso en plataformas sociales.

Finalmente, en el Capítulo 7 se presentaron algunos de los trabajos relacionados más recientes y relevantes.

Trabajo futuro

En busca de continuar avanzando hacia el desarrollo de sistemas con las características descriptas a lo largo de esta tesis, resulta necesario llevar adelante tareas de evaluación empírica adicionales con datos reales y/o sintéticos, considerando mayor diversidad de configuraciones, mayor diversidad de conocimiento ontológico y principalmente mayor cantidad de variantes en lo que concierne el conocimiento experto codificado en los procesos

de difusión. Esto permitirá, entre otras cosas, identificar los límites de la implementación actual en cuanto a tratabilidad computacional.

Asimismo, de acuerdo a la experiencia obtenida de los experimentos realizados hasta el momento, puede ser muy útil diseñar e implementar algoritmos que permitan reutilizar los resultados de procesos de razonamiento realizados con anterioridad, para así evitar realizar cómputos redundantes. Esto implica llevar a cabo algún tipo de proceso de razonamiento incremental que almacene lo que ya fue computado y estudiar cómo incorporar la nueva información evitando cómputos duplicados. Esto es un paso importante para avanzar hacia una implementación que pueda utilizarse en entornos reales que requieren de respuestas rápidas.

Otra alternativa interesante que puede traer grandes beneficios, es estudiar las condiciones bajo las cuales sea posible utilizar un gestor de base de datos relacional (DBMS) para almacenar las bases de conocimiento, desarrollando traducciones de la base y consultas NetDER a operaciones SQL para resolver las consultas a través de base de datos relacionales. Esto también puede contribuir a facilitar la utilización de las implementaciones desarrolladas en entornos reales.

Por otro lado, los sistemas que se buscan desarrollar deben proporcionar herramientas mediante las cuales sea posible explicar las decisiones que se toman. Por lo tanto, resulta conveniente estudiar cuáles son los mecanismos adecuados para proporcionar explicaciones pertinentes a las respuestas a consultas obtenidas. Esto es importante para que un analista experto en el dominio pueda corregir conocimiento incorrecto, considerando nuevas alternativas o tener una aceptación más fuerte respecto a las conclusiones obtenidas. A su vez, en caso de que se produjeran inconvenientes, se puede disponer de registros de mejor calidad para llevar a cabo auditorias.

Finalmente, considerando que la semántica abordada a lo largo de esta tesis vinculada a las bases de conocimiento NetDER es operacional, estudiar la forma de proporcionar una semántica de modelos puede ser un aporte valioso, debido a que puede facilitar la prueba de nuevas propiedades.

Capítulo 9

Apéndice

9.1. El lenguaje ontológico NetDER codificado en Reglas Existenciales Clásicas

En esta sección, se mostrará cómo el lenguaje ontológico NetDER puede ser mapeado a reglas existenciales clásicas; teniendo en cuenta que esta familia de lenguajes es también conocida como *Datalog+/–*, en ocasiones dicha familia de lenguajes será referida utilizando ese nombre.

Las siguientes definiciones presentan los detalles de la codificación mencionada.

Definición 9.1 (Base de conocimiento NetDER transformada) *Sea* $KB = (D, G, \Sigma, P)$ *una base de conocimiento* NetDER*, entonces la base de conocimiento transformada* $trans(KB) = (D_t, \Sigma_t)$ *es un base de conocimiento Datalog+/– (en ocasiones* $trans(KB)$ *se denotará con* KB_t*) obtenida a partir de la aplicación de las siguientes reglas:*

a) D_t *contiene el átomo* $tmax(c)$ *(c es una constante) de* G*.*

b) D_t *contiene todos los átomos correspondientes a cada nodo y arco del grafo, y a cada etiqueta definida en* G*.*

c) Los hechos NetDiff *(relativos a nodos, arcos y la red en general) y los componentes objetivos de red de las reglas son transformados de acuerdo al mapeo presentado en la*

hecho NetDiff (para etiqueta local de nodo)	$\langle node(n), (lab\_pred_1(s_1, ..., s_k), [l_1, u_1])\rangle : [t_1, t_2]$
Versión transformada	$\{node(n),\ aux\_lab\_pred_1(n, s_1, ..., s_k, l_1, u_1, t_1, t_2)\}$
hecho NetDiff (para etiqueta local de arco)	$\langle edge(n_1, n_2), (lab\_pred_1(s_1, ..., s_k), [l_1, u_1])\rangle : [t_1, t_2]$
Versión transformada	$\{edge(n_1, n_2),\ aux\_lab\_pred_1(n_1, n_2, s_1, ..., s_k, l_1, u_1, t_1, t_2)\}$
hecho NetDiff (para etiqueta global)	$\langle \Omega, (lab\_pred_1(s_1, ..., s_k), [l_1, u_1])\rangle : [t_1, t_2]$
Versión transformada	$\{aux\_lab\_pred_1(\Omega, s_1, ..., s_k, l_1, u_1, t_1, t_2)\}$
Comp. objetivo de red (nodos)	$\langle node(X), \langle lab\_pred_1(\mathbf{S}_1), [l_1, u_1]\rangle \wedge ... \wedge \langle lab\_pred_k(\mathbf{S}_k), [l_k, u_k]\rangle\rangle$
Versión transformada	$node(X) \wedge aux\_lab\_pred_1(X, \mathbf{S}_1, l'_1, u'_1, T_1, T_2) \wedge gte(l'_1, l_1) \wedge gte(u_1, u'_1) \wedge ...$ $\wedge aux\_lab\_pred_k(X, \mathbf{S}_k, l'_k, u'_k, T_1, T_2) \wedge gte(l'_k, l_k) \wedge gte(u_k, u'_k)$
Comp. objetivo de red (arcos)	$\langle edge(X, Y), \langle lab\_pred_1(\mathbf{S}_1), [l_1, u_1]\rangle \wedge ... \wedge \langle lab\_pred_k(\mathbf{S}_k), [l_k, u_k]\rangle\rangle$
Versión transformada	$edge(X, Y) \wedge aux\_lab\_pred_1(X, Y, \mathbf{S}_1, l'_1, u'_1, T_1, T_2) \wedge gte(l'_1, l_1) \wedge gte(u_1, u'_1) \wedge ...$ $\wedge aux\_lab\_pred_k(X, Y, \mathbf{S}_k, l'_k, u'_k, T_1, T_2) \wedge gte(l'_k, l_k) \wedge gte(u_k, u'_k)$
Anotación global de red	$\langle lab\_pred_1(\mathbf{S}_1), [l_1, u_1]\rangle \wedge ... \wedge \langle lab\_pred_k(\mathbf{S}_k), [l_k, u_k]\rangle$
Versión transformada	$aux\_lab\_pred_1(\Omega, \mathbf{S}_1, l'_1, u'_1, T_1, T_2) \wedge gte(l'_1, l_1) \wedge gte(u_1, u'_1) \wedge ...$ $\wedge aux\_lab\_pred_k(\Omega, \mathbf{S}_k, l'_k, u'_k, T_1, T_2) \wedge gte(l'_k, l_k) \wedge gte(u_k, u'_k)$

Figura 9.1: Mapeo a partir de elementos del lenguaje ontológico NetDER a elementos del lenguaje *Datalog+/–*.

Figura 9.1. En éstos y los siguientes casos análogos, los nuevos predicados "aux_X" introducidos en las transformaciones codifican la misma relación "X" e incorporan como posiciones adicionales las cotas de incertidumbre y las marcas de tiempo.

d) D_t contiene cada átomo de D,

e) D_t contiene cada hecho NetDiff transformado correspondiente a cada hecho NetDiff de G,

f) Por cada TGD NetDER *$\sigma \in \Sigma$ de la forma:*

$$\Upsilon(\mathbf{X}) \wedge \Phi(\mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{Q}, \mathbf{R}) \wedge \Xi(\mathbf{S}, \mathbf{T}) : \gamma(\mathbf{U})$$

Σ_t contiene una TGD σ_t de la forma:

$$\gamma_t(\mathbf{U}, T_1, T_2) \wedge \Upsilon(\mathbf{X}) \wedge \Phi_t(\mathbf{Y}, T_1, T_2) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{Q}, \mathbf{R}) \wedge \Xi_t(\mathbf{S}, \mathbf{T}, T_1, T_2)$$

donde $\gamma_t(\mathbf{U}, T_1, T_2)$ es la anotación global de red transformada de $\gamma(\mathbf{U})$, y $\Phi_t(\mathbf{Y}, T_1, T_2)$ y $\Xi_t(\mathbf{S}, \mathbf{T}, T_1, T_2)$ son las conjunciones de los componentes objetivos de red transformados de $\Phi(\mathbf{Y})$ y $\Xi(\mathbf{S}, \mathbf{T})$, respectivamente.

g) Por cada EGD NetDER *$\epsilon \in \Sigma$ de la forma:*

$$\Upsilon(\mathbf{S}) \wedge \Phi(\mathbf{T}) \rightarrow (X_i = X_j) : \gamma(\mathbf{U})$$

Σ_t contiene una EGD ϵ_t de la forma:

$$\gamma_t(\mathbf{U}, T_1, T_2) \wedge \Upsilon(\mathbf{S}) \wedge \Phi_t(\mathbf{T}, T_1, T_2) \rightarrow (X_i = X_j)$$

donde $\gamma_t(\mathbf{U}, T_1, T_2)$ es la anotación global de red transformada de $\gamma(\mathbf{U})$, y $\Phi_t(\mathbf{T}, T_1, T_2)$ es la conjunción de los componentes objetivos de red transformados de $\Phi(\mathbf{T})$.

h) Por cada NC NETDER *$\mu \in \Sigma$ de la forma:*

$$\Upsilon(\mathbf{S}) \wedge \Phi(\mathbf{T}) \rightarrow \bot : \gamma(\mathbf{U})$$

Σ_t contiene una NC μ_t de la forma:

$$\gamma_t(\mathbf{U}, T_1, T_2) \wedge \Upsilon(\mathbf{S}) \wedge \Phi_t(\mathbf{T}, T_1, T_2) \rightarrow \bot$$

donde $\gamma_t(\mathbf{U}, T_1, T_2)$ es la anotación global de red transformada de $\gamma(\mathbf{U})$, y $\Phi_t(\mathbf{T}, T_1, T_2)$ es la conjunción de los componentes objetivos de red transformados de $\Phi(\mathbf{T})$.

i) Para cada etiqueta local de nodo $L_{nloc} \in \mathcal{L}_{nloc}$, etiqueta local de arco $L_{eloc} \in \mathcal{L}_{eloc}$ y etiqueta global $L_{glo} \in \mathcal{L}_{glo}$, Σ_t contiene TGDs de la forma:

$$\begin{aligned}
&node(X) \wedge aux_L_{nl}(X, L_{nl}, L_1, U_1, T_1, T_2) \wedge \\
&gte(T_3, T_1) \wedge gte(T_2, T_3) \wedge gte(T_4, T_1) \wedge gte(T_2, T_4) \wedge gte(T_4, T_3) \\
&\qquad \rightarrow aux_L_{nl}(X, L_{nl}, L_1, U_1, T_3, T_4)
\end{aligned}$$

$$\begin{aligned}
&edge(X, Y) \wedge aux_L_{el}(X, Y, L_{el}, L_1, U_1, T_1, T_2) \wedge \\
&gte(T_3, T_1) \wedge gte(T_2, T_3) \wedge gte(T_4, T_1) \wedge gte(T_2, T_4) \wedge gte(T_4, T_3) \\
&\qquad \rightarrow aux_L_{el}(X, Y, L_{el}, L_1, U_1, T_3, T_4)
\end{aligned}$$

$$\begin{aligned}
&aux_L_g(\Omega, L_g, L_1, U_1, T_1, T_2) \wedge \\
&gte(T_3, T_1) \wedge gte(T_2, T_3) \wedge gte(T_4, T_1) \wedge gte(T_2, T_4) \wedge gte(T_4, T_3) \\
&\qquad \rightarrow aux_L_g(X, Y, L_g, L_1, U_1, T_3, T_4)
\end{aligned}$$

donde $gte(X, Y)$ codifica la relación "mayor o igual que" entre números.

Observe que T_1, T_2 son dos variables compartidas en cada componente objetivo de red y anotación global de red de cada TGD, EGD o NC, lo cual permite representar en una forma abstracta el período de tiempo definido en la consulta.

Finalmente, también resulta necesario transformar las consultas.

Definición 9.2 (Consulta conjuntiva NETDER transformada) *Sea* $Q(\mathbf{X}) = \exists\mathbf{Y}\,\rho(\mathbf{X}_1,\mathbf{Y}_1) \wedge \phi(\mathbf{X}_2,\mathbf{Y}_2) \wedge \gamma(\mathbf{X}_3,\mathbf{Y}_3) : [t_1,t_2]$ *una consulta* NETDER *tal que* $\mathbf{X} = \mathbf{X}_1 \cup \mathbf{X}_2 \cup \mathbf{X}_3$, $\mathbf{Y} = \mathbf{Y}_1 \cup \mathbf{Y}_2 \cup \mathbf{Y}_3$, *entonces la consulta conjuntiva* NETDER *transformada de* $Q(\mathbf{X})$ *es de la forma:* $trans(Q(\mathbf{X})) = \exists\mathbf{Y}\,\rho(\mathbf{X}_1,\mathbf{Y}_1) \wedge \phi_t(\mathbf{X},\mathbf{Y},t_1,t_2) \wedge \gamma_t(\mathbf{X}_3,\mathbf{Y}_3,t_1,t_2)$ *donde* $\phi_t(\mathbf{X},\mathbf{Y},t_1,t_2)$ *y* $\gamma_t(\mathbf{X}_3,\mathbf{Y}_3,t_1,t_2)$ *son la conjunción de los componentes objetivos de red transformados y la anotación global de red transformada de* $\phi(\mathbf{X}_2,\mathbf{Y}_2)$ *y* $\gamma(\mathbf{X}_3,\mathbf{Y}_3)$, *respectivamente. En ocasiones, se denota* $trans(Q(\mathbf{X}))$ *con* $Q_t(\mathbf{X})$.

El siguiente resultado declara una equivalencia entre las respuestas a consultas computadas utilizando el NETDER *chase* y aquellas obtenidas utilizando el procedimiento *chase* clásico.

Teorema 9.1 *Dada una consulta conjuntiva* NETDER $Q(\mathbf{X}) = \exists\mathbf{Y}\,\rho(\mathbf{X}_1,\mathbf{Y}_1) \wedge \phi(\mathbf{X}_2,\mathbf{Y}_2) \wedge \gamma(\mathbf{X}_3,\mathbf{Y}_3) : [t_1,t_2]$, *el conjunto de respuestas de* Q *sobre una base de conocimiento* $KB = (D,G,\Sigma,P)$ *computado utilizando el chase* NETDER, *es idéntico al conjunto de respuestas computado utilizando el chase Datalog+/– clásico sobre la correspondiente base de conocimiento transformada* $trans(KB)$ *y la consulta transformada* $trans(Q(\mathbf{X}))$.

Prueba 9.1 (Teorema 9.1) Para probar este resultado, primero se necesita mostrar que:

(1) cuando una TGD NETDER es aplicable en el *chase* NETDER, su transformación será aplicable en el *chase* clásico; y

(2) el resultado de aplicar TGDs NETDER en el *chase* NETDER es equivalente al resultado de aplicar su transformación en el *chase* clásico.

La prueba para EGDs NETDER es análoga.

Dada una TGD NETDER (ω) $\Upsilon(\mathbf{X}) \wedge \Phi(\mathbf{Y}) \rightarrow \exists\mathbf{Z}\,\Psi(\mathbf{Q},\mathbf{R}) \wedge \Xi(\mathbf{S},\mathbf{T}) : \gamma(\mathbf{U})$, esta regla es aplicable para el *chase* NETDER cuando $\Upsilon(\mathbf{X})$ (las condiciones ontológicas) es aplicable en el *chase* clásico, y por otra parte, $\Phi(\mathbf{Y})$ y $\gamma(\mathbf{U})$ (las condiciones de la red) son aplicables a (D,G) en el período de tiempo $[t_1,t_2]$ utilizando el *chase* NETDER. Es decir, la respuesta de la función *checkCondInState*(Q_1, *net_state*) es "Sí", donde

$Q_1 = \Big(h_1(\Phi(\mathbf{Y})) \wedge (\Omega, h_1(\gamma(\mathbf{U})))\Big) : [t_1, t_2]$, h_1 es un homomorfismo adecuado y *net_state* es la salida del Algoritmo 1 sobre la base de conocimiento NetDiff $DiffKB = (G, \emptyset)$. Observe que, la regla transformada de acuerdo al ítem (g) en la Definición 9.1 incorpora dos nuevas conjunciones: Φ_t y γ_t. Si suponemos que ω es aplicable, entonces sabemos que existen los correspondientes hechos NetDiff que aseguran la aplicabilidad de $\Phi(\mathbf{Y})$ y $\gamma(\mathbf{U})$, lo que significa que sus transformaciones también valen con los apropiados homomorfismos (esto es así porque las reglas auxiliares en el ítem (i) de la Definición 9.1 codifican adecuadamente los intervalos de tiempo en las reglas). Para apreciar esto, se necesita considerar cómo la aplicabilidad es decidida para las condiciones de la red, lo cual esencialmente requiere combinar los mapeos de las condiciones de la red y el período de tiempo definido en la consulta NETDER, para así obtener consultas NetDiff. Considere una consulta NetDiff $Q = \big((c, \langle L, bnd\rangle) : [t_1, t_2]\big)$ donde $c \in V \cup E \cup \{\Omega\}$, $L \in \mathcal{L}$, $bnd \in [0, 1]$ y $[t_1, t_2] \subset [0, t_{max}]$ (recuerde que, por definición de la regla del *chase* para TGDs NETDER, el *chase* NETDER necesita consultar el estado de la red usando el Algoritmo 1 y la función *checkCondInState*). Si la respuesta de *checkCondInState*(Q, net_state) es "Sí", entonces la respuesta de *checkCondInState*(Q', net_state) es "Sí" para toda consulta NetDiff $Q' = \big((c, \langle L, bnd\rangle) : [t'_1, t'_2]\big)$ donde $[t'_1, t'_2] \subseteq [t_1, t_2]$. Esto es equivalente a decir que si el hecho NetDiff $\big((c, \langle L, bnd\rangle) : [t_1, t_2]\big) \in G$ entonces cada hecho NetDiff en $\{\big((c, \langle L, bnd\rangle) : [t'_1, t'_2]\big)$ tal que $[t'_1, t'_2] \subseteq [t_1, t_2]\}$ puede ser derivado de G, lo cual en el caso más simple podría realizarse si dichos hechos pertenecen a G (lo cual se lleva adelante por el ítem (i) de la Definición 9.1). En ese caso, para responder la consulta $Q = \big((c, \langle L, bnd\rangle) : [t_1, t_2]\big)$ sólo es necesario verificar si $\big((c, \langle L, bnd\rangle) : [t_1, t_2]\big) \in G$. Por lo tanto, esto completa la prueba de que una TGD es aplicable sobre una base de conocimiento utilizando el *chase* NETDER si y sólo si la TGD transformada es aplicable sobre al base de conocimiento transformada utilizando el *chase* clásico.

Para la segunda parte (2), cuando se aplica una TGD, ambos procedimientos *chase* agregan los átomos mapeados de $\Psi(\mathbf{Q}, \mathbf{R})$ a la estructura en evolución; el *chase* NETDER agrega los átomos mapeados de $\Xi(\mathbf{S}, \mathbf{T})$ con intervalo $[t_1, t_2]$ y el *chase* clásico agrega los átomos mapeados de $\Xi_t(\mathbf{S}, \mathbf{T})$ con intervalos $[t'_1, t'_2] \subseteq [t_1, t_2]$.

Por lo tanto, las respuestas a la consulta $Q(\mathbf{X})$ sobre la base de conocimiento KB computadas usando el chase NETDER son las mismas que las obtenidas mediante el procedimiento clásico con la consulta transformada $Q_t(\mathbf{X})$ sobre la base de conocimiento transformada *trans*(KB).

□

9.2. Pruebas

Prueba 9.2 (Prueba del Teorema 5.1) La prueba de la condición (a) del Teorema es una adaptación de la prueba presentada en [SSS13, SSC13]; sin embargo, para facilitar el entendimiento se reproducen los detalles. Para cada $node(n_i) \in V$, se utilizará la notación $d_{n_i}^{in}$ para denotar el número de vecinos incidentes (a partir de cualquier tipo de arco) en el nodo $node(n_i)$ y $d_{\Omega}^{in} = max(\{d_{n_k}^{in} \text{ s.t. } node(n_k) \in V\})$. Además, se asumirá que el peor caso de cualquier función de influencia utilizada es IF_{time}. Para mayor claridad se dividirá la prueba en varias partes:

Declaración 1. Cada vez que una regla local NetDiff básica (*ground*) de la forma $L_{tc} \overset{\Delta t}{\leftarrow} (nc_{edge}, nc_{node}, if)$ provoca que la cota en un átomo de red (asociado con un cierto $node(n_i)$) se ajuste, hay a lo sumo $|\mathcal{L}_{nloc}| + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{n_i}^{in}$ verificaciones de satisfacción y es llevado a cabo un cálculo de la función de influencia. Más específicamente, hay $|\mathcal{L}_{nloc}|$ verificaciones de satisfacción en el peor caso para L_{tc}, y $|\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{n_i}^{in}$ verificaciones de satisfacción en el peor caso para nc_{edge} y nc_{node}. Observe que esta declaración vale bajo cualquiera de las tres condiciones.

Declaración 2. Si la condición (a) es satisfecha, entonces una regla local NetDiff básica (*ground*) dada puede ajustar la cota en un átomo de red de un nodo $node(n_i)$ en un tiempo definido no más que $d_{n_i}^{in}$ veces. Esto se debe a que si toda función de influencia sólo depende de la relación entre los vecinos totales (elegibles) y una fracción de éstos que satisfacen determinado criterio (calificados), entonces el número total de cotas diferentes que pueden ser asignadas es $d_{n_i}^{in}$. Observe que no es sólo el número total de posibilidades para una única aplicación de las reglas, sino de todas las aplicaciones de las reglas antes de converger.

Declaración 3. Si la condición (b) es satisfecha, entonces un átomo de red relativo a un nodo determinado considerando cada punto del tiempo produce a lo sumo $(t_{max} + 1) \cdot 2^{2b}$ modificaciones. Esto puede explicarse debido a la cota en el número de *bits*, lo cual permite representar a lo sumo 2^b números diferentes y $2^b + 2^b - 1 + ... + 1 \leq 2^b \cdot 2^b = 2^{2b}$ intervalos diferentes, por lo cual la cantidad de modificaciones para cada átomo de red relativo a un nodo considerando todos los puntos de tiempo es a lo sumo $(t_{max} + 1) \cdot 2^{2b}$.

Declaración 4. Si la condición (c) es satisfecha, entonces el número de aplicaciones para una regla local NetDiff básica (*ground*) está en $O(|DiffKB|^{t_{max}})$. La intuición detrás de esta afirmación es que el número de aplicaciones para una regla local NetDiff básica (*ground*) que puede afectar un átomo de red en un tiempo $t \in [0, t_{max}]$ depende del número de modificaciones en los tiempos $t' < t$. Por lo tanto, cualquier átomo de red relativo a cualquier nodo en el tiempo $t = 1$ sólo podría ser cambiado por algún hecho NetDiff y entonces el número de modificaciones es a lo sumo $|DiffKB|$. Similarmente, cualquier átomo de red para cualquier nodo en el tiempo $t = 1$ sólo podría ser modificado por algún hecho NetDiff (a lo sumo $|DiffKB|$ modificaciones) y a lo sumo $|DiffKB|$ aplicaciones (porque depende de las modificaciones en los tiempos $t < 1$) de $|DiffKB|$ reglas locales NetDiff—y como consecuencia, a lo sumo $|DiffKB| + |DiffKB|^2$ modificaciones.

Continuando con este razonamiento, para $t = 2$ el número de modificaciones es $|DiffKB|$ (provenientes de los hechos NetDiff) más $|DiffKB| + |DiffKB| + |DiffKB|^2$ aplicaciones de $|DiffKB|$ reglas locales NetDiff: es decir, a lo sumo $2 \cdot |DiffKB|^2 + |DiffKB|^3 + |DiffKB|$ modificaciones. Consecuentemente, se puede deducir que el número de modificaciones en el tiempo t crece en un orden $O(|DiffKB|^t)$ y cuando la suma de todas las modificaciones en cada tiempo es considerada, la complejidad es $O(|DiffKB|^{t_{max}})$. Asimismo, como la aplicación de una regla local NetDiff en el tiempo t depende de las modificaciones que fueron hechas en el tiempo $t' < t$, entonces también el número de aplicaciones de una regla local NetDiff dada crece en orden $O(|DiffKB|^{t_{max}})$.

Declaración 5. Si la condición (c) se cumple, incluso cuando $\Delta t = 1$ en todas las reglas locales NetDiff de *DiffKB*, el número de aplicaciones para una regla local NetDiff dada tiene orden $O(|DiffKB|^{t_{max}})$. Esto resulta de que, a diferencia del caso en la Declaración 4, el número de modificaciones en el tiempo $t = 2$ es a lo sumo $|DiffKB|+|DiffKB|^2+|DiffKB|^3$, en el tiempo $t = 3$ es a lo sumo $|DiffKB| + |DiffKB|^2 + |DiffKB|^3 + |DiffKB|^4$, y así sucesivamente. Claramente, el número de modificaciones en el tiempo $t \in [0, t_{max}]$ tiene orden $O(|DiffKB|^t)$ y cuando la suma de todas las modificaciones en cada tiempo es considerada, el número crece en orden $O(|DiffKB|^{t_{max}})$. Sin embargo, aunque la complejidad es la misma usando cualquier $\Delta t > 0$, utilizar un Δt más pequeño es mejor en aplicaciones prácticas porque la probabilidad de interacción entre reglas locales NetDiff es más pequeña[1].

[1]Una forma de atacar esta fuente de complejidad podría ser analizar el caso en el cual el número de interacciones entre reglas locales NetDiff está acotado usando una noción similar a *tree-width*.

Declaración 6. Si la condición (a) es satisfecha, entonces cada regla local NetDiff contribuye en costo de orden $O(t_{max} \cdot |V| \cdot (IF_{time} + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{\Omega}^{in}))$. Claramente, hay sólo t_{max} puntos de tiempo que pueden ser afectados por una regla local NetDiff y, para cada nodo y punto de tiempo, hay sólo una etiqueta local que puede ser afectada por una regla local NetDiff. Por lo tanto, considerando las Declaraciones 1 y 2, se puede decir que cada regla local NetDiff requiere a lo sumo:

$$t_{max} \cdot \sum_{i=1}^{|V|} \Big((IF_{time} + |\mathcal{L}_{nloc}| + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{n_i}^{in}) \cdot d_{n_i}^{in} \Big)$$

verificaciones de condiciones en la red, lo cual tiene complejidad $O(t_{max} \cdot |V| \cdot (IF_{time} \cdot d_{\Omega}^{in} + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{\Omega}^{in^2}))$.

Declaración 7. Si la condición (b) es satisfecha, entonces cada regla local NetDiff contribuye en costo de orden $O(t_{max} \cdot |V| \cdot (IF_{time} + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{\Omega}^{in}) \cdot 2^{2b})$. Similarmente a la Declaración 6, a partir de la Declaración 1 y Declaración 3 se deduce que cada regla local NetDiff requiere a lo sumo:

$$t_{max} \cdot \sum_{i=1}^{|V|} \Big((IF_{time} + |\mathcal{L}_{nloc}| + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{n_i}^{in}) \cdot 2^{2b} \Big)$$

verificaciones de condiciones en la red, lo cual es de orden $O(t_{max} \cdot |V| \cdot (IF_{time} + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{\Omega}^{in}) \cdot 2^{2b})$.

Declaración 8. Si la condición (c) es satisfecha, entonces cada regla local NetDiff contribuye en costo de orden $O(|V| \cdot (IF_{time} + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{\Omega}^{in}) \cdot |DiffKB|^{t_{max}})$. Similarmente a la Declaración 6 y a la Declaración 7, a partir de la Declaración 1 y la Declaración 4 se puede deducir que cada regla local NetDiff requiere a lo sumo:

$$\sum_{i=1}^{|V|} \Big((IF_{time} + |\mathcal{L}_{nloc}| + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{n_i}^{in}) \cdot |DiffKB|^{t_{max}} \Big),$$

verificaciones de condiciones en la red, lo cual es de orden $O(|V| \cdot (IF_{time} + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{\Omega}^{in}) \cdot |DiffKB|^{t_{max}})$.

Como el Algoritmo 1 es dominado por el ciclo que implementa el operador de punto fijo, resulta necesario analizar la complejidad para la convergencia de dicho ciclo *while*:

- $O(|DiffKB| \cdot t_{max} \cdot |V| \cdot (IF_{time} \cdot d_{\Omega}^{in} + |\mathcal{L}_{nloc}| \cdot |\mathcal{L}_{eloc}| \cdot d_{\Omega}^{in^2}))$ si la condición (a) es satisfecha, lo cual se deduce a partir de la Declaración 6 y el tamaño de la base de conocimiento NetDiff.

- $O(|DiffKB| \cdot t_{max} \cdot |V| \cdot (IF_{time} + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{\Omega}^{in}) \cdot 2^{2b})$ si la condición (b) es satisfecha, lo cual se deduce a partir de la Declaración 7 y el tamaño de la base de conocimiento NetDiff.

- $O(|V| \cdot (IF_{time} + |\mathcal{L}_{eloc}| \cdot |\mathcal{L}_{nloc}| \cdot d_{\Omega}^{in}) \cdot |DiffKB|^{t_{max}})$ si la condición (c) es satisfecha, lo cual se deduce a partir de la Declaración 8 incluso considerando el tamaño de la base de conocimiento NetDiff.

Prueba 9.3 (Teorema 5.2) Dada una base de conocimiento NETDER $KB = (D, G, \Sigma, P)$, la complejidad de computar las respuestas a las consultas NETDER resulta a partir de: (i) la complejidad de ejecutar el chase NETDER sobre KB, (ii) la complejidad de ejecutar el proceso de difusión sobre la base de conocimiento NetDiff $DiffKB = (G, P)$, y (iii) repetir estos dos pasos un número de veces dictaminado por la política de respuesta a consultas (QAP).

El costo de ejecutar el *chase* NETDER es a lo sumo el mismo que el requerido para ejecutar el *chase* clásico sobre la base de conocimiento transformada, lo cual puede inferirse a partir del Teorema 9.1. El costo de ejecutar el proceso de difusión con garantías de terminación se encuentra acotado por un polinomio en el tamaño de la entrada (ver Teorema 5.1). Por lo tanto, si el número de veces que se repiten estos pasos está acotado por una constante, es posible concluir que la complejidad del proceso de respuesta a consulta NETDER está acotada por un polinomio o está dominado por la complejidad de la ejecución del NETDER chase. Claramente, una constante como la mencionada existe cuando se utiliza la política *Bounded-Int*. En el caso de la política *Unbounded-Int*, el teorema sólo considera el caso en el que la red no puede ser modificada; por lo tanto, ejecutar el proceso de difusión una vez solamente es suficiente.

Prueba 9.4 (Teorema 5.3) Esta prueba estará basada en mostrar que dada la descripción de una Máquina de Turing existe un conjunto fijo de TGDs NETDER Σ y un conjunto fijo de reglas NetDiff P tal que el proceso de respuesta a una consulta conjuntiva NETDER $Q(\mathbf{X})$ fija es indecidible. Esto incluye el caso en el que se tengan garantías de terminación tanto en el proceso de respuesta a la consulta transformada sobre la base de conocimiento transformada como en el proceso de difusión. Esto es llevado a cabo mostrando que es posible simular una Máquina de Turing universal determinista (TM) M (la cual puede ejecutar cualquier Máquina de Turing determinista dada), utilizando una base

de conocimiento NETDER con las garantías de terminación mencionadas anteriormente. Por lo tanto, se asume que se dispone como entrada la descripción de una Máquina de Turing como la mencionada y el objetivo es encontrar una base de conocimiento NETDER (con las restricciones pertinentes) y una consulta NETDER booleana tal que la respuesta a la consulta es "Sí" si y sólo si la TM se detiene en una cinta de entrada vacía.

Tener en cuenta que utilizando un conjunto fijo de TGDs NETDER entonces es posible forzar que una cinta no acotada (potencialmente infinita) aparezca cuando el *chase* NETDER es ejecutado. Estas reglas requeridas para lograr esto son descriptas en detalle más abajo; observe que se utiliza el predicado $node(.)$ para referirse a celdas de la cinta y, por lo tanto, en esta prueba se utilizará indistintamente *cell* y *node*. Las siguientes suposiciones y definiciones iniciales son realizadas:

- El *chase* que se usa en el proceso de respuesta a consultas es modificado de tal manera que cada vez que se ejecuta sólo se aplica cada regla con un cuantificador existencial en la cabeza una vez, mientras que todas las demás reglas se aplican tantas veces como sea necesario. Esta suposición se puede realizar sin afectar la propiedad que se busca probar, ya que es una forma de obtener garantías de terminación en el *chase*.
- Cada vez que el Algoritmo 2 de respuesta a consultas NETDER cambia del NDM al ORM, pasa sólo un paso de tiempo.
- La base de conocimiento NetDiff es definida con $t_{max} = 1$ y las funciones de influencia son restringidas para asegurar garantías de terminación para el proceso de difusión (cf. Teorema 5.1).
- Las funciones de influencia if_{one} e if_{zero} son definidas de manera tal que siempre devuelven $[1, 1]$ y $[0, 0]$, respectivamente. Esta es una forma de asegurar garantías de terminación en el proceso de difusión debido a que cualquier átomo de red a lo sumo puede cambiar su cota a $[1, 1]$ o $[0, 0]$ sólo una vez en un tiempo determinado.
- Se dispone de las etiquetas locales de nodo $trans/5$, $state/1$ y $content/1$ para cada transición, estado y símbolo del alfabeto, respectivamente. Hay también otras etiquetas locales $cursor/0$, $right/0$, $left/0$, $self/0$, $otherRight/0$ and $otherLeft/0$ que tienen los siguientes significados:
  - $trans(s_1, a_1, s_2, a_2, dir)$ es una etiqueta local de nodo que declara "si el estado actual es s_1 y un símbolo a_1 es leído, entonces cambiar al estado s_2, escribir

a_2, y mover la cabeza en la dirección *dir*". Cada nodo ($node/1$) tiene este tipo de etiqueta con confianza $[1, 1]$.

- $state(s_1)$ es una etiqueta local de nodo que representa que "s_1 es un estado". Cada nodo ($node/1$) tiene sólo una etiqueta de este tipo con confianza $[1, 1]$ representando que el estado actual de ese nodo ($node/1$) es s_1.
- $content(a_1)$ es una etiqueta local de nodo que representa que "a_1 es el símbolo contenido". Cada nodo ($node/1$) tiene solo una etiqueta de este tipo con confianza $[1, 1]$ representando que el contenido de ese nodo ($node/1$) es a_1.
- *cursor* es una etiqueta local de nodo representando la posición del cursor. Sólo un nodo ($node/1$) tiene la etiqueta *cursor* con confianza $[1, 1]$, la cual representa que el cursor está actualmente en ese nodo ($node/1$).
- *right* es una etiqueta local de arco que permite representar qué celda se encuentra a la derecha. Esto es expresado cuando la etiqueta *right* de un arco $edge(X_1, X_2)$ que vincula dos nodos $node(X_1), node(X_2)$ tiene confianza $[1, 1]$—$node(X_2)$ es entonces la celda a la derecha de $node(X_1)$.
- *left* es una etiqueta local de arco que permite representar qué celda se encuentra a la izquierda; esto es análogo a la etiqueta anterior.
- *self* es una etiqueta local de arco que representa un vinculo al mismo nodo ($node/1$).
- *otherRight* es una etiqueta local de arco que permite representar qué celdas se encuentran a la derecha pero no son adyacentes. Esto es expresado cuando la etiqueta *otherRight* del arco $edge(X_1, X_2)$ que vincula dos nodos $node(X_1), node(X_2)$ tiene confianza $[1, 1]$—$node(X_2)$ entonces representa la celda que se encuentra a la derecha de $node(X_1)$ pero no es adyacente.
- *otherLeft* es una etiqueta local de arco que permite representar qué celdas se encuentran a la izquierda pero no son adyacentes; esto es análogo al ítem anterior.

- La base de datos de red (G) contiene un nodo en el cual el intervalo de confianza de la etiqueta *cursor* es $[1, 1]$ y un único estado finalizador s_0 de TM M el cual es codificado por el átomo $halt(s_0)$.

A continuación, se describe el conjunto Σ de TGDs NETDER:

tgd_1: $(node(X), \langle cursor, [1,1]\rangle) \rightarrow$
$\exists Y, Z(edge(X,Y), \langle right, [1,1]\rangle \wedge \langle left, [0,0]\rangle \wedge \langle self, [0,0]\rangle) \wedge$
$(edge(Y,X), \langle left, [1,1]\rangle \wedge \langle right, [0,0]\rangle \wedge \langle self, [0,0]\rangle) \wedge$
$(edge(X,X), \langle self, [1,1]\rangle \wedge \langle right, [0,0]\rangle \wedge \langle left, [0,0]\rangle) \wedge$
$(edge(Z,X), \langle right, [1,1]\rangle \wedge \langle left, [0,0]\rangle \wedge \langle self, [0,0]\rangle) \wedge$
$(edge(X,Z), \langle left, [1,1]\rangle \wedge \langle right, [0,0]\rangle \wedge \langle self, [0,0]\rangle)$

Esta TGD declara que todo nodo con el cursor tiene un nodo a la derecha, un nodo a la izquierda y un vínculo a sí mismo. Observe que esto modifica la estructura de la red subyacente necesaria para luego ejecutar el proceso de difusión.

tgd_2: $edge(X,Y) \rightarrow node(X) \wedge node(Y)$

Cada arco vincula nodos.

tgd_3: $(edge(X,Y), \langle right, [1,1]\rangle) \wedge (edge(Y,Z), \langle right, [1,1]\rangle) \rightarrow (edge(X,Z), \langle otherRight, [1,1]\rangle)$

tgd_4: $(edge(X,Y), \langle otherRight, [1,1]\rangle) \wedge (edge(Y,Z), \langle right, [1,1]\rangle) \rightarrow (edge(X,Z), \langle otherRight, [1,1]\rangle)$

Cada nodo a la derecha con respecto a otro que no es adyacente está vinculado a través de un arco que va desde este último al mencionado primero y tiene la etiqueta *otherRight* con el intervalo de confianza $[1,1]$.

tgd_5: $(edge(X,Y), \langle left, [1,1]\rangle) \wedge (edge(Y,Z), \langle left, [1,1]\rangle) \rightarrow (edge(X,Z), \langle otherLeft, [1,1]\rangle)$

tgd_6: $(edge(X,Y), \langle otherLeft, [1,1]\rangle) \wedge (edge(Y,Z), \langle left, [1,1]\rangle) \rightarrow (edge(X,Z), \langle otherLeft, [1,1]\rangle)$

Análogamente al par previo de TGDs, cada nodo a la izquierda con respecto a otro que no es adyacente está vinculado a través de un arco que va desde este último al mencionado primero y tiene la etiqueta *otherLeft* con confianza $[1,1]$.

tgd_7: $(node(X), \langle cursor, [1,1]\rangle \wedge \langle state(S), [1,1]\rangle) \wedge halt(S) \rightarrow stop$

Si la TM entra en el estado de detención, se deriva el símbolo *stop*.

A continuación, se describen las reglas locales NetDiff. Recordar que se busca codificar una Máquina de Turing determinista y, por lo tanto, sólo uno de los nodos puede influenciar a otros en cada regla.

r_1: $cursor \overset{1}{\leftarrow} \langle right, [1,1]\rangle, \langle trans(S_1, A_1, S_2, A_2, right), [1,1]\rangle \wedge \langle state(S_1), [1,1]\rangle \wedge$
$\langle content(A_1), [1,1]\rangle \wedge \langle cursor, [1,1]\rangle, if_{one}$

r_2: $cursor \overset{1}{\leftarrow} \langle left, [1,1]\rangle, \langle trans(S_1, A_1, S_2, A_2, left), [1,1]\rangle \wedge \langle state(S_1), [1,1]\rangle \wedge$
$\langle content(A_1), [1,1]\rangle \wedge \langle cursor, [1,1]\rangle, if_{one}$

r_3: $cursor \overset{1}{\leftarrow} \langle self, [1,1]\rangle, \langle trans(S_1, A_1, S_2, A_2, stay), [1,1]\rangle \wedge \langle state(S_1), [1,1]\rangle \wedge$
$\langle content(A_1), [1,1]\rangle \wedge \langle cursor, [1,1]\rangle, if_{one}$

Las reglas NetDiff r_1–r_3 codifican las transiciones: cuando se está en estado S_1, con contenido A_1 y el cursor en el nodo objetivo, entonces el nodo a la derecha (izquierda o el mismo, respectivamente) tendrá el cursor en el siguiente punto de tiempo.

r_4: $content(A_2) \overset{1}{\leftarrow} \langle self, [1,1]\rangle, \langle trans(S_1, A_1, S_2, A_2, X), [1,1]\rangle \wedge \langle state(S_1), [1,1]\rangle \wedge$
$\langle content(A_1), [1,1]\rangle \wedge \langle cursor, [1,1]\rangle, if_{one}$

r_5: $state(S_2) \overset{1}{\leftarrow} \langle self, [1,1]\rangle, \langle trans(S_1, A_1, S_2, A_2, X), [1,1]\rangle \wedge \langle state(S_1), [1,1]\rangle \wedge$
$\langle content(A_1), [1,1]\rangle \wedge \langle cursor, [1,1]\rangle, if_{one}$

Las reglas NetDiff r_4 y r_5 codifican el nuevo contenido escrito en la cinta después de una transición a partir de influenciarse a sí mismos y asignar el intervalo $[1,1]$ a su etiqueta $content(A_2)$ ($state(S_2)$, respectivamente) de acuerdo a la transición; es decir, tendrá el contenido A_2 (estado S_2, respectivamente) en el siguiente punto de tiempo.

r_6: $content(A_1) \overset{1}{\leftarrow} \langle self, [1,1]\rangle, \langle content(A_1), [1,1]\rangle \wedge \langle cursor, [0,0]\rangle, if_{one}$

r_7: $state(S_1) \overset{1}{\leftarrow} \langle self, [1,1]\rangle, \langle state(S_1), [1,1]\rangle \wedge \langle cursor, [0,0]\rangle, if_{one}$

Las reglas NetDiff r_6 y r_7 codifican la "inercia"—cada nodo que no tiene el cursor se influenciará a sí mismo de manera que se conserve el contenido actualmente en la cinta en el siguiente punto de tiempo.

r_8: $cursor \overset{1}{\leftarrow} \langle otherRight, [1,1]\rangle, \langle trans(S_1, A_1, S_2, A_2, X), [1,1]\rangle \wedge \langle state(S_1), [1,1]\rangle \wedge$
$\langle content(A_1), [1,1]\rangle \wedge \langle cursor, [1,1]\rangle, if_{zero}$

r_9: $cursor \overset{1}{\leftarrow} \langle otherLeft, [1,1]\rangle, \langle trans(S_1, A_1, S_2, A_2, X), [1,1]\rangle \wedge \langle state(S_1), [1,1]\rangle$
$\wedge \langle content(A_1), [1,1]\rangle \wedge \langle cursor, [1,1]\rangle, if_{zero}$

Las reglas NetDiff r_8 y r_9 están relacionados con los nodos a la derecha (izquierda, respectivamente) que no son adyacentes, asignando el intervalo de confianza $[0, 0]$ a su etiqueta *cursor*; es decir, no tendrán el cursor en el punto de tiempo siguiente.

r_{10}: $cursor \overset{1}{\leftarrow} \langle right, [1,1]\rangle, \langle trans(S_1, A_1, S_2, A_2, left), [1,1]\rangle \wedge \langle state(S_1), [1,1]\rangle \wedge \langle content(A_1), [1,1]\rangle \wedge \langle cursor, [1,1]\rangle, if_{zero}$

r_{11}: $cursor \overset{1}{\leftarrow} \langle left, [1,1]\rangle, \langle trans(S_1, A_1, S_2, A_2, right), [1,1]\rangle \wedge \langle state(S_1), [1,1]\rangle \wedge \langle content(A_1), [1,1]\rangle \wedge \langle cursor, [1,1]\rangle, if_{zero}$

r_{12}: $cursor \overset{1}{\leftarrow} \langle right, [1,1]\rangle, \langle trans(S_1, A_1, S_2, A_2, stay), [1,1]\rangle \wedge \langle state(S_1), [1,1]\rangle \wedge \langle content(A_1), [1,1]\rangle \wedge \langle cursor, [1,1]\rangle, if_{zero}$

r_{13}: $cursor \overset{1}{\leftarrow} \langle left, [1,1]\rangle, \langle trans(S_1, A_1, S_2, A_2, stay), [1,1]\rangle \wedge \langle state(S_1), [1,1]\rangle \wedge \langle content(A_1), [1,1]\rangle \wedge \langle cursor, [1,1]\rangle, if_{zero}$

Finalmente, las reglas r_9–r_{13} se encargan de asignar el intervalo $[0, 0]$ a los nodos correspondientes que *no* tendrán el cursor en el próximo punto de tiempo.

Dada la construcción de la base de conocimiento NetDER *KB* detallada arriba, es claro que el proceso de respuesta a consultas (cf. Algoritmo 2) imita perfectamente la Máquina de Turing de entrada M y por lo tanto M se detiene si y sólo si la respuesta del Algoritmo 2 a la consulta $Q = stop : [t_{max}, t_{max}]$ sobre la base de conocimiento *KB* y utilizando la política *Unbounded-Int* es "Sí". Tener en cuenta que esto incluye el caso en el que se tienen garantías de terminación tanto en el proceso de respuesta a la consulta transformada sobre la base de conocimiento transformada como en el proceso de difusión. Por lo tanto, se ha reducido el problema de la detención para las Máquinas de Turing universales deterministas con una cinta de entrada vacía al problema de responder consultas en NetDER bajo la política *Unbounded-Int* y con capacidad de modificar la red. Con esto concluye entonces la prueba de que este último problema es indecidible.

Bibliografía

[AA03] Adamic, L. A., and Adar, E. Friends and neighbors on the web. *Soc. Networks 25*, 3 (2003), 211–230.

[AA12] Ahmed, F., and Abulaish, M. An mcl-based approach for spam profile detection in online social networks. In *11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2012, Liverpool, United Kingdom, June 25-27, 2012* (2012), pp. 602–608.

[AHV95] Abiteboul, S., Hull, R., and Vianu, V. *Foundations of Databases*. Addison-Wesley, 1995.

[AMN+18] Almukaynizi, M., Marin, E., Nunes, E., Shakarian, P., Simari, G. I., Kapoor, D., and Siedlecki, T. DARKMENTION: A deployed system to predict enterprise-targeted external cyberattacks. In *2018 IEEE International Conference on Intelligence and Security Informatics, ISI 2018, Miami, FL, USA, November 9-11, 2018* (2018), pp. 31–36.

[ARK12] Aggarwal, A., Rajadesingan, A., and Kumaraguru, P. Phishari: Automatic realtime phishing detection on twitter. In *2012 eCrime Researchers Summit, eCrime 2012, Las Croabas, PR, USA, October 23-24, 2012* (2012), pp. 1–12.

[ASN+16] Almaatouq, A., Shmueli, E., Nouh, M., Alabdulkareem, A., Singh, V. K., Alsaleh, M., Alarifi, A., Alfaris, A., and Pentland, A. S. If it looks like a spammer and behaves like a spammer, it must be a spammer: analysis and detection of microblogging spam accounts. *Int. J. Inf. Sec. 15*, 5 (2016), 475–491.

[BC16] Benigni, M., and Carley, K. M. From tweets to intelligence: Understanding the islamic jihad supporting community on twitter. In *Social, Cultural, and Behavioral Modeling, 9th International Conference, SBP-BRiMS 2016, Washington, DC, USA, June 28 - July 1, 2016, Proceedings* (2016), pp. 346–355.

[BCM+03] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., Eds. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[BHLS17] Baader, F., Horrocks, I., Lutz, C., and Sattler, U. *An Introduction to Description Logic*. Cambridge University Press, 2017.

[Bia15] Bianconi, G. Interdisciplinary and physics challenges of network theory. *EPL (Europhysics Letters) 111*, 5 (2015), 56001.

[BJC17] Benigni, M. C., Joseph, K., and Carley, K. M. Online extremism and the communities that sustain it: Detecting the isis supporting community on twitter. *PloS one 12*, 12 (2017), e0181405.

[BJC19] Benigni, M. C., Joseph, K., and Carley, K. M. Bot-ivistm: Assessing information manipulation in social media using network analytics. In *Emerging Research Challenges and Opportunities in Computational Social Network Analysis and Mining*. Springer, 2019, pp. 19–42.

[BKL13] Bertossi, L. E., Kolahi, S., and Lakshmanan, L. V. S. Data cleaning and query answering with matching dependencies and matching functions. *Theory Comput. Syst. 52*, 3 (2013), 441–482.

[BMRT11] Baget, J., Mugnier, M., Rudolph, S., and Thomazo, M. Walking the complexity lines for generalized guarded existential rules. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011* (2011), pp. 712–717.

[BMT11] Baget, J., Mugnier, M., and Thomazo, M. Towards farsighted dependencies for existential rules. In *Web Reasoning and Rule Systems - 5th International Conference, RR 2011, Galway, Ireland, August 29-30, 2011. Proceedings* (2011), pp. 30–45.

[BN08] Bleiholder, J., and Naumann, F. Data fusion. *ACM Comput. Surv. 41*, 1 (2008), 1:1–1:41.

[BNJU17] Bekiros, S., Nguyen, D. K., Junior, L. S., and Uddin, G. S. Information diffusion, cluster formation and entropy-based network dynamics in equity and commodity markets. *European Journal of Operational Research 256*, 3 (2017), 945–961.

[Buz16] BuzzFeedNews. This analysis shows how viral fake election news stories outperformed real news on facebook. `https://www.buzzfeednews.com/article/craigsilverman/viral-fake-election-news-outperformed-real-news-on-facebook`, 2016.

[BV81] Beeri, C., and Vardi, M. Y. The implication problem for data dependencies. In *Automata, Languages and Programming, 8th Colloquium, Acre (Akko), Israel, July 13-17, 1981, Proceedings* (1981), pp. 73–85.

[BV84] Beeri, C., and Vardi, M. Y. A proof procedure for data dependencies. *J. ACM 31*, 4 (1984), 718–741.

[Cam01] Campbell, W. J. *Yellow journalism: Puncturing the myths, defining the legacies*. Greenwood Publishing Group, 2001.

[Cen15] Centola, D. The social origins of networks and diffusion. *American Journal of Sociology 120*, 5 (2015), 1295–1338.

[CGK13] Calì, A., Gottlob, G., and Kifer, M. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res. 48* (2013), 115–174.

[CGL12] Calì, A., Gottlob, G., and Lukasiewicz, T. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Semant. 14* (2012), 57–83.

[CGP12] Calì, A., Gottlob, G., and Pieris, A. Towards more expressive ontology languages: The query answering problem. *Artif. Intell. 193* (2012), 87–128.

[CGWJ12] Chu, Z., Gianvecchio, S., Wang, H., and Jajodia, S. Detecting automation of twitter accounts: Are you a human, bot, or cyborg? *IEEE Trans. Dependable Secur. Comput. 9*, 6 (2012), 811–824.

[CSYP12] Cao, Q., Sirivianos, M., Yang, X., and Pregueiro, T. Aiding the detection of fake accounts in large scale social online services. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012* (2012), pp. 197–210.

[CT91] Console, L., and Torasso, P. A spectrum of logical definitions of model-based diagnosis. *Computational Intelligence 7*, 3 (1991), 133–141.

[CW19] CYR3CON, and WhiteSource. Vulnerability prioritization through the eyes of hackers. `https://www.whitesourcesoftware.com/vulnerabilities-prioritization-by-hackers/`, 2019.

[CWW12] Chu, Z., Widjaja, I., and Wang, H. Detecting social spam campaigns on twitter. In *Applied Cryptography and Network Security - 10th International Conference, ACNS 2012, Singapore, June 26-29, 2012. Proceedings* (2012), pp. 455–472.

[CZF04] Chakrabarti, D., Zhan, Y., and Faloutsos, C. R-MAT: A recursive model for graph mining. In *Proceedings of the Fourth SIAM International Conference on Data Mining, Lake Buena Vista, Florida, USA, April 22-24, 2004* (2004), pp. 442–446.

[DKFF15] Devineni, P., Koutra, D., Faloutsos, M., and Faloutsos, C. If walls could talk: Patterns and anomalies in facebook wallposts. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2015, Paris, France, August 25 - 28, 2015* (2015), pp. 367–374.

[DMFS16] Deagustini, C. A. D., Martinez, M. V., Falappa, M. A., and Simari, G. R. Datalog+/– ontology consolidation. *J. Artif. Intell. Res. 56* (2016), 613–656.

[DMFS18] Deagustini, C. A. D., Martinez, M. V., Falappa, M. A., and Simari, G. R. How does incoherence affect inconsistency-tolerant semantics for Datalog+/–? *Ann. Math. Artif. Intell. 82*, 1-3 (2018), 43–68.

[DT99] Dehaspe, L., and Toivonen, H. Discovery of frequent DATALOG patterns. *Data Min. Knowl. Discov. 3*, 1 (1999), 7–36.

[EG95] Eiter, T., and Gottlob, G. The complexity of logic-based abduction. *Journal of the ACM 42*, 1 (1995), 3–42.

[EGL97] Eiter, T., Gottlob, G., and Leone, N. Abduction from logic programs: Semantics and complexity. *Theoretical Computer Science 189*, 1-2 (1997), 129–177.

[EIV07] Elmagarmid, A. K., Ipeirotis, P. G., and Verykios, V. S. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng. 19*, 1 (2007), 1–16.

[Fac18a] FacebookApp. Facing facts: An inside look at facebook's fight against misinformation. `https://www.youtube.com/watch?v=zgkF23nFIBw`, May 2018.

[Fac18b] FacebookApp. Hard questions: What's facebook's strategy for stopping false news? `https://about.fb.com/news/2018/05/hard-questions-false-news/`, May 2018.

[Fac20a] Facebook. April 2020 coordinated inauthentic behavior report. `https://about.fb.com/news/2020/05/april-cib-report/`, 2020.

[Fac20b] Facebook. February 2020 coordinated inauthentic behavior report. `https://about.fb.com/news/2020/03/february-cib-report/`, 2020.

[Fac20c] Facebook. March 2020 coordinated inauthentic behavior report. `https://about.fb.com/news/2020/04/march-cib-report/`, 2020.

[Fac20d] Facebook. May 2020 coordinated inauthentic behavior report. `https://about.fb.com/news/2020/06/may-cib-report/`, 2020.

[FGKIS13] Falappa, M. A., García, A. J., Kern-Isberner, G., and Simari, G. R. Stratified belief bases revision with argumentative inference. *Journal of Philosophical Logic 42*, 1 (2013), 161–193.

[FKIRS12] Falappa, M. A., Kern-Isberner, G., Reis, M. D. L., and Simari, G. R. Prioritized and non-prioritized multiple change on belief bases. *J. Philosophical Log. 41*, 1 (2012), 77–113.

[FKMP05] Fagin, R., Kolaitis, P. G., Miller, R. J., and Popa, L. Data exchange: semantics and query answering. *Theor. Comput. Sci. 336*, 1 (2005), 89–124.

[GJLS17] Gutiérrez-Basulto, V., Jung, J. C., Lutz, C., and Schröder, L. Probabilistic description logics for subjective uncertainty. *J. Artif. Intell. Res. 58* (2017), 1–66.

[GLMS13] Gottlob, G., Lukasiewicz, T., Martinez, M. V., and Simari, G. I. Query answering under probabilistic uncertainty in Datalog+ / - ontologies. *Ann. Math. Artif. Intell. 69*, 1 (2013), 37–72.

[GM12] Getoor, L., and Machanavajjhala, A. Entity resolution: Theory, practice & open challenges. *Proc. VLDB Endow. 5*, 12 (2012), 2018–2019.

[GOPS12] Gottlob, G., Orsi, G., Pieris, A., and Simkus, M. Datalog and its extensions for semantic web databases. In *Reasoning Web. Semantic Technologies for Advanced Query Answering - 8th International Summer School 2012, Vienna, Austria, September 3-8, 2012. Proceedings* (2012), pp. 54–77.

[GRS14] Gottlob, G., Rudolph, S., and Simkus, M. Expressiveness of guarded existential rule languages. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014* (2014), pp. 27–38.

[GSM+17] Gallo, F. R., Simari, G. I., Martinez, M. V., Falappa, M. A., and Santos, N. A. Reasoning about sentiment and knowledge diffusion in social networks. *IEEE Internet Computing 21*, 6 (2017), 8–17.

[GSMF19] Gallo, F. R., Simari, G. I., Martinez, M. V., and Falappa, M. A. Predicting user reactions to Twitter feed content based on personality type and social cues. *Future Generation Computer Systems (In Press)* (2019).

[HKBN18] Howard, P. N., Kollanyi, B., Bradshaw, S., and Neudert, L. Social media, news and political information during the US election: Was polarizing content concentrated in swing states? *CoRR abs/1802.03573* (2018).

[HKP12] Heidemann, J., Klier, M., and Probst, F. Online social networks: A survey of a global phenomenon. *Comput. Networks 56*, 18 (2012), 3866–3878.

[HODC+16] Hine, G., Onaolapo, J., De Cristofaro, E., Kourtellis, N., Leontiadis, I., Samaras, R., Stringhini, G., and Blackburn, J. A longitudinal measurement study of 4chan's politically incorrect forum and its effect on the web (version 3)', 2016.

[Inf20] Infobae. Le hicieron un llamado falso, le dijeron que tenía coronavirus y desataron una "caza de brujas". `https://www.infobae.com/sociedad/2020/04/02/le-hicieron-un-llamado-falso-le-dijeron-que-tenia-coronavirus-y-desataron-una-caza-de-brujas`, 2020.

[JP17] Jalili, M., and Perc, M. Information cascades in complex networks. *J. Complex Networks 5*, 5 (2017), 665–693.

[KGB15] Khorasani, F., Gupta, R., and Bhuyan, L. N. Scalable simd-efficient graph processing on GPUs. In *Proceedings of the 24th International Conference on Parallel Architectures and Compilation Techniques* (2015), PACT '15, pp. 39–50.

[KMM00] Kakas, A. C., Michael, A., and Mourlas, C. ACLP: abductive constraint logic programming. *J. Log. Program. 44*, 1-3 (2000), 129–177.

[Kol18] Kolaitis, P. G. Reflections on schema mappings, data exchange, and metadata management. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018* (2018), pp. 107–109.

[KPP02] Konieczny, S., and Pino Pérez, R. Merging information under constraints: A logical framework. *Journal of Logic and Computation 12*, 5 (2002), 773–808.

[KPP11] Konieczny, S., and Pino Pérez, R. Logic based merging. *Journal of Philosophical Logic 40*, 2 (2011), 239–270.

[KW17] Klein, D., and Wueller, J. Fake news: A legal perspective. *Journal of Internet Law (Apr. 2017)* (2017).

[LaN20] LaNueva. La cooperativa denunciará en la justicia las noticias falsas en las redes. `https://www.lanueva.com/nota/2020-6-4-6-30-25-la-cooperativa-denunciara-en-la-justicia-las-noticias-falsas-en-las-redes`, 2020.

[LCW10] Lee, K., Caverlee, J., and Webb, S. Uncovering social spammers: social honeypots + machine learning. In *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2010, Geneva, Switzerland, July 19-23, 2010* (2010), pp. 435–442.

[LK14] Lee, S., and Kim, J. Early filtering of ephemeral malicious accounts on twitter. *Comput. Commun. 54* (2014), 48–57.

[LMC+15] Liu, D., Mei, B., Chen, J., Lu, Z., and Du, X. Community based spammer detection in social networks. In *Web-Age Information Management - 16th International Conference, WAIM 2015, Qingdao, China, June 8-10, 2015. Proceedings* (2015), pp. 554–558.

[LMS12] Lukasiewicz, T., Martinez, M. V., and Simari, G. I. Inconsistency handling in datalog+/– ontologies. In *Proc. ECAI* (2012), pp. 558–563.

[LMTV12] Leone, N., Manna, M., Terracina, G., and Veltri, P. Efficiently computable datalog∃ programs. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012* (2012).

[Luk08] Lukasiewicz, T. Expressive probabilistic description logics. *Artif. Intell. 172*, 6-7 (2008), 852–883.

[LXFZ13] Li, Y., Xiao, R., Feng, J., and Zhao, L. A semi-supervised learning approach for detection of phishing webpages. *Optik 124*, 23 (2013), 6027–6033.

[MA13] Martínez-Romo, J., and Araujo, L. Detecting malicious tweets in trending topics using a statistical analysis of language. *Expert Syst. Appl. 40*, 8 (2013), 2992–3000.

[Mil18] Miller, R. J. Open data integration. *Proceedings of the VLDB Endowment 11*, 12 (2018), 2130–2139.

[MIT20] MITTechnologyReview. How facebook uses machine learning to detect fake accounts. `https://www.technologyreview.com/2020/03/04/905551/how-facebook-uses-machine-learning-to-detect-fake-accounts/`, 2020.

[MMS79] Maier, D., Mendelzon, A. O., and Sagiv, Y. Testing implications of data dependencies. *ACM Trans. Database Syst. 4*, 4 (1979), 455–469.

[MRR16] Mulamba, D., Ray, I., and Ray, I. Sybilradar: A graph-structure based framework for sybil detection in on-line social networks. In *ICT Systems Security and Privacy Protection - 31st IFIP TC 11 International Conference, SEC 2016, Ghent, Belgium, May 30 - June 1, 2016, Proceedings* (2016), pp. 179–193.

[MRV16] Malliaros, F. D., Rossi, M.-E. G., and Vazirgiannis, M. Locating influential nodes in complex networks. *Scientific reports 6* (2016), 19307.

[MW] Merriam-Webster. Definition of half-truth. `https://www.merriam-webster.com/dictionary/half-truth`.

[MYK10] Mohaisen, A., Yun, A., and Kim, Y. Measuring the mixing time of social graphs. In *Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference, IMC 2010, Melbourne, Australia - November 1-3, 2010* (2010), pp. 383–389.

[NDG+16] Nunes, E., Diab, A., Gunn, A. T., Marin, E., Mishra, V., Paliath, V., Robertson, J., Shakarian, J., Thart, A., and Shakarian, P.

Darknet and deepnet mining for proactive cybersecurity threat intelligence. In *IEEE Conference on Intelligence and Security Informatics, ISI 2016, Tucson, AZ, USA, September 28-30, 2016* (2016), pp. 7–12.

[NFAG16] Narudin, F. A., Feizollah, A., Anuar, N. B., and Gani, A. Evaluation of machine learning classifiers for mobile malware detection. *Soft Comput. 20*, 1 (2016), 343–357.

[Nil86] Nilsson, N. J. Probabilistic logic. *Artif. Intell. 28*, 1 (1986), 71–87.

[NS92] Ng, R. T., and Subrahmanian, V. S. Probabilistic logic programming. *Information and Computation 101*, 2 (1992), 150–201.

[NSS18] Nunes, E., Shakarian, P., and Simari, G. I. At-risk system identification via analysis of discussions on the darkweb. In *2018 APWG Symposium on Electronic Crime Research, eCrime 2018, San Diego, CA, USA, May 15-17, 2018* (2018), pp. 1–12.

[Oni] Onion, T. The onion. `http://www.theonion.com/`.

[OSH+07] Onnela, J.-P., Saramäki, J., Hyvönen, J., Szabó, G., Lazer, D., Kaski, K., Kertész, J., and Barabási, A.-L. Structure and tie strengths in mobile communication networks. *Proceedings of the national academy of sciences 104*, 18 (2007), 7332–7336.

[Pag20] Pagina12. Coronavirus: allanaron el domicilio desde donde supuestamente enviaron la fake news del aislamiento. `https://www.pagina12.com.ar/254776-coronavirus-allanaron-el-domicilio-desde-donde-supuestamente`, 2020.

[Pea89] Pearl, J. *Probabilistic reasoning in intelligent systems - networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, 1989.

[Pei40] Peirce, C. S. *The Philosophy of Peirce: Selected Writings*. Harcourt, New York, NY, 1940.

[PGA87] Poole, D., Goebel, R., and Aleliunas, R. Theorist: A logical reasoning system for defaults and diagnosis. In *The Knowledge Frontier*. 1987, pp. 331–352.

[PMSF18] Paredes, J., Martinez, M. V., Simari, G. I., and Falappa, M. A. Leveraging probabilistic existential rules for adversarial deduplication. In *Proceedings of the Second Workshop on Logics for Reasoning about Preferences, Uncertainty, and Vagueness co-located with the 9th International Joint Conference on Automated Reasoning, PRUV@IJCAR 2018, Oxford, UK, July 19th, 2018* (2018).

[Pol17] Politifact. The more outrageous, the better: How clickbait ads make money for fake news sites. `https://www.politifact.com/article/2017/oct/04/more-outrageous-better-how-clickbait-ads-make-mone/`, 2017.

[Poo89] Poole, D. Explanation and prediction: An architecture for default and abductive reasoning. *Computational Intelligence 5*, 2 (1989), 97–110.

[Poo97] Poole, D. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence 94*, 1-2 (1997), 7–56.

[PSM+20] Paredes, J., Simari, G. I., Martinez, M. V., Falappa, M. A., and Shakarian, P. Combining existential rules with network diffusion processes for automated generation of hypotheses. *Theory and Practice of Logic Programming* (En evaluación, 2020) (2020).

[PSMF18] Paredes, J., Simari, G. I., Martinez, M. V., and Falappa, M. A. First steps towards data-driven adversarial deduplication. *Inf. 9*, 8 (2018), 189.

[PSMF20a] Paredes, J., Simari, G. I., Martinez, M. V., and Falappa, M. A. Detecting malicious behavior in social platforms via hybrid knowledge- and data-driven systems. *Information Systems* (En evaluación, 2020) (2020).

[PSMF20b] Paredes, J., Simari, G. I., Martinez, M. V., and Falappa, M. A. NetDER: An architecture for reasoning about malicious behavior. *Information Systems Frontiers* (2020), 1–17.

[Qua17] Quartz. People shared nearly as much fake news as real news on twitter during the election. `https://qz.com/1090903/people-shared-nearly-as-much-fake-news-as-real-news-on-twitter-during-the-election/`, 2017.

[RD06] Richardson, M., and Domingos, P. M. Markov logic networks. *Mach. Learn. 62*, 1-2 (2006), 107–136.

[Rog18] Rogerio, C. Fake news detector API. `https://github.com/fake-news-detector/fake-news-detector/tree/master/api/#json-api-endpoints`, 2018.

[RSS+15] Riabov, A. V., Sohrabi, S., Sow, D. M., Turaga, D. S., Udrea, O., and Vu, L. H. Planning-based reasoning for automated large-scale data analysis. In *Twenty-Fifth International Conference on Automated Planning and Scheduling* (2015), pp. 282–290.

[SASS18a] Sarkar, S., Almukaynizi, M., Shakarian, J., and Shakarian, P. Predicting enterprise cyber incidents using social network analysis on the darkweb hacker forums. In *2018 International Conference on Cyber Conflict, CyCon U.S. 2018, Washington, DC, USA, November 14-15, 2018* (2018), pp. 1–7.

[SASS18b] Sarkar, S., Almukaynizi, M., Shakarian, J., and Shakarian, P. Predicting enterprise cyber incidents using social network analysis on the darkweb hacker forums. In *2018 International Conference on Cyber Conflict, CyCon U.S. 2018, Washington, DC, USA, November 14-15, 2018* (2018), pp. 1–7.

[Sat] SatireWire. Satirewire. `http://www.satirewire.com/`.

[SBD+18] Sapienza, A., Bessi, A., Damodaran, S., Shakarian, P., Lerman, K., and Ferrara, E. Early warnings of cyber threats in online discussions. *CoRR abs/1801.09781* (2018).

[SBS14] Singh, M., Bansal, D., and Sofat, S. Detecting malicious users in twitter using classifiers. In *Proceedings of the 7th International Conference on Security of Information and Networks, Glasgow, Scotland, UK, September 9-11, 2014* (2014), p. 247.

[Sch07] Schaeffer, S. E. Graph clustering. *Comput. Sci. Rev. 1*, 1 (2007), 27–64.

[SGS16] Shakarian, J., Gunn, A. T., and Shakarian, P. Exploring malicious hacker forums. In *Cyber Deception, Building the Scientific Foundation*. 2016, pp. 261–284.

[SGS18] Shaabani, E., Guo, R., and Shakarian, P. Detecting pathogenic social media accounts without content or network structure. In *1st International Conference on Data Intelligence and Security, ICDIS 2018, South Padre Island, TX, USA, April 8-10, 2018* (2018), pp. 57–64.

[Sha00] Shanahan, M. An abductive event calculus planner. *Journal of Logic Programming 44* (2000), 207–239.

[Sig19] SignalSciences. The bot problem: Effective detection, analysis, & blocking. `https://library.devops.com/the-bot-problem?__hstc=90482629.426fbee58b2d919d09f762589f52e19f.1596914471780.1596914471780.1596914471780.1&__hssc=90482629.1.1596914471780&__hsfp=172421138`, 2019.

[SMM+17] Simari, G. I., Molinaro, C., Martinez, M. V., Lukasiewicz, T., and Predoiu, L. *Ontology-Based Data Access Leveraging Subjective Reports*. Springer Briefs in Computer Science. Springer, 2017.

[Sno17a] Snopes. Adam sandler death hoax. `https://www.snopes.com/fact-check/adam-sandler-death-hoax-2/`, 2017.

[Sno17b] Snopes. Boston marathon bombing rumors. `https://www.snopes.com/fact-check/boston-marathon-bombing-rumors/`, 2017.

[SS11] Sadan, Z., and Schwartz, D. G. Social network analysis of web links to eliminate false positives in collaborative anti-spam systems. *J. Netw. Comput. Appl. 34*, 5 (2011), 1717–1723.

[SSAS19] Shaabani, E., Sadeghi-Mobarakeh, A., Alvari, H., and Shakarian, P. An end-to-end framework to identify pathogenic social media accounts on twitter. In *2nd International Conference on Data Intelligence and Security, ICDIS 2019, South Padre Island, TX, USA, June 28-30, 2019* (2019), pp. 128–135.

[SSC13] Shakarian, P., Simari, G. I., and Callahan, D. Reasoning about complex networks: A logic programming approach. *Theory Pract. Log. Program. 13*, 4-5-Online-Supplement (2013).

[SSS13] Shakarian, P., Simari, G. I., and Schroeder, R. MANCaLog: a logic for multi-attribute network cascades. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13, Saint Paul, MN, USA, May 6-10, 2013* (2013), pp. 1175–1176.

[SZM13] Sallaberry, A., Zaidi, F., and Melançon, G. Model for generating artificial social networks having community structures with small-world and scale-free properties. *Soc. Netw. Anal. Min. 3*, 3 (2013), 597–609.

[TGSP11] Thomas, K., Grier, C., Song, D., and Paxson, V. Suspended accounts in retrospect: an analysis of twitter spam. In *Proceedings of the 11th ACM SIGCOMM Internet Measurement Conference, IMC '11, Berlin, Germany, November 2-, 2011* (2011), pp. 243–258.

[The18] TheGuardian. Twitter admits far more russian bots posted on election than it had disclosed. `https://www.theguardian.com/technology/2018/jan/19/twitter-admits-far-more-russian-bots-posted-on-election-than-it-had-disclosed`, 2018.

[Tim] Time. The origins of writerly words. `http://time.com/82601/the-origins-of-writerly-words/`.

[Tim17] Time. Inside russia's social media war on america. `https://time.com/4783932/inside-russia-social-media-war-america`, 2017.

[TLSC11] Tran, D. N., Li, J., Subramanian, L., and Chow, S. S. M. Optimal sybil-resilient node admission control. In *INFOCOM 2011. 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 10-15 April 2011, Shanghai, China* (2011), pp. 3218–3226.

[TRG19] Tommasel, A., Rodriguez, J. M., and Godoy, D. An experimental study on feature engineering and learning approaches for aggression detection in social media. *Inteligencia Artif. 22*, 63 (2019), 81–100.

[VCvHF09] Valle, E. D., Ceri, S., van Harmelen, F., and Fensel, D. It's a streaming world! reasoning upon rapidly changing information. *IEEE Intell. Syst. 24*, 6 (2009), 83–89.

[VPGM10] Viswanath, B., Post, A., Gummadi, P. K., and Mislove, A. An analysis of social network-based sybil defenses. In *Proceedings of the ACM SIGCOMM 2010 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, New Delhi, India, August 30 -September 3, 2010* (2010), pp. 363–374.

[Wan17] Wang, W. Y. "Liar, liar pants on fire": A new benchmark dataset for fake news detection. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017* (2017), R. Barzilay and M. Kan, Eds., Association for Computational Linguistics, pp. 422–426.

[Wik17a] Wikipedia. Murder of seth rich. `https://en.wikipedia.org/wiki/Murder_of_Seth_Rich`, 2017.

[Wik17b] Wikipedia. Pizzagate conspiracy theory. `https://en.wikipedia.org/wiki/Pizzagate_conspiracy_theory`, 2017.

[WMW+13] Wang, G., Mohanlal, M., Wilson, C., Wang, X., Metzger, M. J., Zheng, H., and Zhao, B. Y. Social turing tests: Crowdsourcing sybil detection. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013* (2013).

[WWZ+12] Wang, G., Wilson, C., Zhao, X., Zhu, Y., Mohanlal, M., Zheng, H., and Zhao, B. Y. Serf and turf: crowdturfing for fun and profit. In *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012* (2012), pp. 679–688.

[YGKX10] Yu, H., Gibbons, P. B., Kaminsky, M., and Xiao, F. Sybillimit: A near-optimal social network defense against sybil attacks. *IEEE/ACM Trans. Netw. 18*, 3 (2010), 885–898.

[YHG13] Yang, C., Harkreader, R. C., and Gu, G. Empirical evaluation and new design for fighting evolving twitter spammers. *IEEE Trans. Inf. Forensics Secur. 8*, 8 (2013), 1280–1293.

[YKGF08] Yu, H., Kaminsky, M., Gibbons, P. B., and Flaxman, A. D. Sybilguard: defending against sybil attacks via social networks. *IEEE/ACM Trans. Netw. 16*, 3 (2008), 576–589.

[YXY+16] Yang, Z., Xue, J., Yang, X., Wang, X., and Dai, Y. Votetrust: Leveraging friend invitation graph to defend against social network sybils. *IEEE Trans. Dependable Secur. Comput. 13*, 4 (2016), 488–501.

[ZG09] Zhu, X., and Goldberg, A. B. *Introduction to Semi-Supervised Learning.* Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2009.

[ZL16] Zhang, Y., and Lu, J. Discover millions of fake followers in weibo. *Soc. Netw. Anal. Min. 6*, 1 (2016), 16:1–16:15.

[ZSBK19] Zannettou, S., Sirivianos, M., Blackburn, J., and Kourtellis, N. The web of false information: Rumors, fake news, hoaxes, clickbait, and various other shenanigans. *J. Data and Information Quality 11*, 3 (2019), 10:1–10:37.

[ZZC+15] Zheng, X., Zeng, Z., Chen, Z., Yu, Y., and Rong, C. Detecting spammers on social networks. *Neurocomputing 159* (2015), 27–34.