Universidad Nacional del Sur

Tesis de Doctor en Ciencias de la Computación

Herramientas prácticas para argumentación estructurada probabilística con aplicación a ciberseguridad

Mario Alejandro Leiva

Bahía Blanca Argentina

2022

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 2017 y diciembre de 2021, bajo la dirección del Dr. Gerardo I. Simari, Profesor Adjunto del Departamento de Ciencias e Ingeniería de la Computación y del Dr. Marcelo A. Falappa, Profesor Asociado del Departamento de Ciencias e Ingeniería de la Computación.

Mario Alejandro Leiva
mario.leiva@cs.uns.edu.ar
Departamento de Ciencias de la Computación
Universidad Nacional del Sur
Bahía Blanca, Marzo 2022.

Resumen

El principal objetivo de estudio de esta tesis son los mecanismos eficaces y eficientes para computar respuestas en formalismos de argumentación estructurada probabilística, en particular, en el modelo DeLP3E. Dicho formalismo nos permite modelar y razonar sobre información inconsistente, incompleta, y asociada a eventos probabilísticos. En particular, el objetivo es estudiar y proponer mecanismos que nos permitan aproximar las respuestas por medio de algoritmos que se basen en toda la información disponible de cada componente del modelo. Nos enfocamos en investigar este aspecto dado que, dar una respuesta exacta a una consulta en DeLP3E no es aplicable en un tiempo razonable para instancias grandes.

La principal contribución de esta tesis es la definición y estudio de algoritmos, junto con una guía para seleccionar cúal de ellos aplicar, que nos permitan aproximar el valor de una respuesta en base a toda la información disponible en el modelo DeLP3E. A su vez, y con la finalidad de poder evaluar el desempeño de los algoritmos propuestos, otra importante contribución de esta tesis son tres generadores para crear modelos DeLP3E. Cada uno de estos generadores nos permite crear diferentes escenarios de complejidad de manera automática, y teniendo como principal característica que los valores de sus métricas son ajustables en base al valor de los parámetros de entrada que guían el proceso de generación.

A su vez, y para mostrar un caso de uso de lo desarrollado en esta tesis, se presenta P-DAQAP, una plataforma web que facilita el análisis de los proceso llevados a cabo en el modelo DeLP3E y en la Programación Lógica Rebatible (DeLP, en inglés). Se mostró, a través de casos de uso y ejemplos prácticos, que una herramienta de este tipo puede ser de gran utilidad para usuarios que requieran analizar dominios complejos y que tengan eventos probabilísticos asociados. Además de esa utilidad en entornos reales, también puede ser usada con fines educativos, ya que ofrece un conjunto de herramientas gráficas

para facilitar la lectura y entendimiento de los procesos argumentales llevados a cabo en el formalismo DeLP. P-DAQAP es una herramienta pública que consideramos de gran utilidad para la comunidad de argumentación estructurada.

Abstract

The main objective of this thesis is the study of the effective and efficient mechanisms to compute answers in formalisms of probabilistic structured argumentation, in particular, in the DeLP3E model. This formalism allows us to model and reason on inconsistent and incomplete information, as well information associated with probabilistic events. In particular, the objective is to study and propose mechanisms that allow us to approximate the answers by algorithms that are based on all the available information of each component of the model. We focused on investigating this aspect since giving an exact answer to a query in DeLP3E is not applicable in a reasonable time for large instances.

The main contribution of this thesis is the definition and study of algorithms, and a guide to select which of them to apply, which allow us to approximate the value of a response based on all the information available in the model DeLP3E. Also, and in order to be able to evaluate the performance of the proposed algorithms, another important contribution of this thesis is three generators to create DeLP3E models. Each of these generators allows us to create different complexity scenarios automatically, and having as main characteristic that the values of their metrics are adjustable based on the value of the input parameters that guide the generation process.

Also, and to show a use case of what has been developed in this thesis, P-DAQAP is presented, a web platform that facilitates the analysis of the processes carried out in the DeLP3E model and in Defeasible Logic Programming (DeLP). It was shown, through use cases and practical examples, that a tool of this type can be very useful for users who need to analyze complex domains and have associated probabilistic events. In addition to this utility in real environments, it can also be used for educational purposes, since it offers a set of graphic tools to facilitate the reading and understanding of the argumentative processes carried out in the DeLP formalism. P-DAQAP is a public tool that we consider very useful for the structured argumentation community.

Agradecimientos

Me gustaría agradecer a todos los que me acompañaron e hicieron posible este proceso. En primer lugar, quiero agradecer a dos excelentes personas y profesionales, mis directores Gerardo y Marcelo. Gracias por guiarme, aconsejarme, y principalmente por confiar en mí. Durante estos 5 años siempre estuvieron para cualquier cosa que necesité, y con la mejor predisposición. También quiero agradecer mucho a Guillermo, siempre estuvo presente para aconsejarme desde el primer día y para guiarme durante todo este proceso; aprendí mucho en nuestras reuniones, de su experiencia y sus consejos. Quiero agradecer en particular a Gerardo, una persona a quien considero un ejemplo a seguir, no solo como científico, sino también por la calidez humana. Gracias por confiar en mí, por la paciencia, los consejos, por las enseñanzas, por los retos, por formarme y guiarme.

Muchas gracias a todos mis amigos y compañeros de las salitas de becarios, este proceso fue mucho más llevadero gracias a ustedes, a las charlas, los consejos, los momentos compartidos. Todos los días era algo nuevo con ustedes, aprendí mucho y me divertí compartiendo ese espacio de trabajo. Son de lo mejor, muchas gracias. También quiero agradecer a Noni y Gotti, siempre los escuchaba con mucha atención y considero que aprendí mucho de ustedes. Tengo los mejores recuerdos y anécdotas de estos 5 años compartidos.

Quiero agradecer al Departamento de Ciencias e Ingeniería de la Computación, un lugar que me recibió de la mejor manera posible, brindándome todo lo necesario para poder desarrollar mi doctorado. Todas las personas que trabajan aquí disfrutan de lo que hacen, y eso se nota en la calidad y excelencia del lugar. También quiero agradecer a la Universidad Nacional de Santiago del Estero, por brindarme lo necesario para poder iniciarme en este camino. En este sentido, también quiero agradecer a Maximiliano, me alentó desde un principio a seguir y buscar eso que quería, me aconsejó de la mejor manera siempre. A mis amigos de Santiago, gracias por el aguante de siempre, por el acompañamiento, por los buenos recuerdos.

Por último, quiero agradecer a las personas más importantes, mi familia y mi novia. A mi familia, quien siempre dió todo lo que podía para apoyar mi formación, hoy soy lo que soy gracias a ustedes. A mis hermanos, Mayra y Germán, los quiero mucho y siempre voy a estar para todo lo que necesiten, como sé que ustedes lo están para mi. Muchas gracias hermanos. A mi madre, Silvia, sabes lo que significas para mi, sos todo lo bien que tengo, de quién más aprendí a través del ejemplo, sos lo más bueno, puro y fuerte que conozco. Este logro es en gran parte gracias a vos, a que nunca bajaste los brazos por tus hijos. Quiero dedicar parte de este logro a Valentina, mi novia, tuve la suerte de conocerte y que formes partes de este proceso y de mi vida. Me acompañaste en los momentos más difíciles, siempre positiva y alegre, inteligente, fuerte. Sos una mujer increíble. Te quiero mucho.

Muchas gracias al Instituto de Ciencias e Ingeniería de la Computación (ICIC) y al Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), por haber confiado en mi formación y financiar este proceso. Sin este soporte hubiera sido muy difícil.

Índice general

Capítulo 1

Introducción

Las bases de conocimiento de muchas aplicaciones y sistemas, que describen toda la información disponible sobre un dominio específico, normalmente contienen datos contradictorios y un grado de incertidumbre asociado a ellos. Esto es así dado que dichas bases se construyen utilizando datos de diferentes fuentes que pueden no compartir el mismo formato o ser inherentemente incompletas e inconsistentes [Hal05, Par01]. Por esto, algunos de los problemas principales que se abordan mediante formalismos para la representación del conocimiento son la capacidad para manejar información contradictoria y realizar inferencias en presencia de incertidumbre.

En la literatura existen formalismos y modelos para representar la incertidumbre y realizar inferencias sobre dicha representación, siendo uno de los más desarrollados el concepto de *probabilidad*. En la teoría de la probabilidad, al igual que en la mayoría de los modelos para representar la incertidumbre, se comienza con un conjunto de *mundos posibles*, también denominados *estados*, que representan los mundos o resultados que un agente considera posibles. Por ejemplo, lanzar una moneda (lado A y lado B), considerando dos mundos posibles (uno por cada lado de la moneda). Esto se puede representar mediante un conjunto $\mathcal{W}$ que consta de dos mundos posibles $\{w_1, w_2\}$, siendo w_1 el mundo donde la moneda queda con el lado A hacia arriba y w_2 el mundo donde el lado B queda hacia arriba (el conjunto $\mathcal{W}$ se denomina *espacio muestral*). Un *evento* es un conjunto de mundos posibles, por ejemplo, el evento "sale el lado A" correspondería con el conjunto $\{w_1\}$. En esta interpretación, en general en aplicaciones reales, existe un gran conjunto de mundos posibles (todos los resultados posibles), de los cuales el agente considera posible algún subconjunto. El conjunto de mundos que un agente considera posibles puede verse

como una medida cualitativa de su incertidumbre. Cuantos más mundos considera posibles, más incertidumbre tiene del verdadero estado de las cosas [Hal05]. Lo que el agente conoce o tiene certeza depende en cierta medida de cómo se eligen los mundos posibles y la forma en que se representan. Elegir el conjunto apropiado de mundos posibles puede ser una tarea para nada trivial. Por otro lado, y considerando un enfoque más lógico y menos numérico, la *teoría de la argumentación* nos permite representar la incertidumbre a través de un lenguaje lógico formal y una semántica para razonar en presencia de la misma. La *argumentación* es una disciplina que forma parte del área de razonamiento no monótono en la cual la representación del conocimiento se realiza a través de la especificación de *argumentos*. Cada uno de estos argumentos soporta a una *conclusión* a partir de un conjunto de *premisas*. Un formalismo argumentativo brinda la posibilidad de representar conocimiento en conflicto, dado que un argumento puede estar en contradicción con otro argumento. La ventaja de un formalismo con esas características es la tolerancia a la inconsistencia. Es decir, es posible generar conclusiones válidas a partir de conocimiento que puede ser parcialmente inconsistente, ya que el proceso argumentativo decidirá qué información prevalece frente a una situación de conflicto. Dicho proceso consiste en *razonar* a partir de la construcción y evaluación de argumentos que soportan conclusiones contradictorias. Esto último es objeto de estudio de la *argumentación rebatible*

El modelo utilizado en el desarrollo de esta tesis combina estas dos herramientas; de esta manera, es posible modelar información inconsistente, incompleta, e incierta por medio de la *argumentación rebatible* y los *modelos probabilísticos*.

1.1. Motivación

El dominio de la ciberseguridad es un claro ejemplo donde se presentan situaciones en las que se debe manipular información proveniente de diferentes fuentes. En los últimos años, los problemas en este dominio crecieron tanto en número como en variedad y complejidad [PTSM21, SSR13]. Esto generó un mayor interés en aplicar modelos de Inteligencia Artificial (IA) orientados a la detección tanto de riesgos de seguridad simples como de ciberataques sofisticados y que cuenten con la capacidad para modelar información de naturaleza compleja (variedad de formatos, incompleta, inconsistente, de naturaleza cualitativa, entre otros). En la actualidad existen muchas aplicaciones en las que la IA tiene un impacto positivo; sin embargo, son bien conocidas sus limitaciones al momento

de ofrecer explicaciones o una trazabilidad del proceso llevado a cabo para generar una salida (esto es particularmente notorio en el área del aprendizaje de máquina) [GA19]. Dada la naturaleza de estos problemas, es clara la necesidad de comprender los procesos de toma de decisión de estos sistemas; lo cual se conoce generalmente como IA explicable (XAI, por sus siglas en Inglés).

Bajo estas circunstancias, un sistema de representación de conocimiento y razonamiento automatizado para dar soporte a problemas en dominios complejos como los mencionados, debe ofrecer ciertas capacidades principales. Algunas de ellas se presentan en [SSM$^+$16]; las mismas se listan y extienden a continuación:

1. Razonar a partir de la evidencia de una manera formal.
2. Modelar la evidencia sobre eventos del dominio junto con su nivel de incertidumbre asociado.
3. Modelar reglas lógicas que permitan al sistema obtener conclusiones basadas en ciertas pruebas y aplicar dichas reglas de manera iterativa.
4. Considerar piezas de información que pueden no ser compatibles entre sí, decidir qué información es más relevante y expresar el por qué de la diferencia establecida.
5. Presentar el estado real del sistema en función de las características mencionadas anteriormente y proporcionar al usuario la capacidad de comprender cómo llega el sistema a esa conclusión.

Breve introducción al modelo utilizado

El modelo DeLP3E, originalmente presentado en [SSM$^+$16], cumple con todos los requerimientos mencionados previamente. Este formalismo está compuesto por dos modelos relacionados del dominio. El primero se denomina *modelo del entorno* (de ahora en adelante, "ME"), y es usado para describir conocimiento incierto acerca del dominio y que está sujeto a eventos probabilísticos. El segundo, denominado *modelo analítico* ("MA"), describe conocimiento del dominio que puede ser estricto o rebatible – esto será útil en el análisis de hipótesis que puedan explicar un fenómeno dado (que modelaremos mediante "consultas"). De esta manera, el formalismo nos permite modelar conocimiento incierto o incompleto a través de la *argumentación rebatible* en el MA y asociar este conocimiento

a la ocurrencia de eventos probabilísticos por medio del ME. En particular, en el MA es posible describir el dominio a través de un conjunto de reglas que modelan conocimiento estricto o rebatible, y cada una de esas reglas está asociada a eventos del ME que tienen su valor de probabilidad asociado (esta asociación se da por medio de fórmulas lógicas). De esta manera, podemos modelar información inconsistente, incompleta, e incierta por medio de la *argumentación rebatible* y los *modelos probabilísticos*. En la figura 1.1 se detallan brevemente los componentes y la semántica del formalismo DeLP3E. El MA contiene las reglas que modelan el conocimiento del dominio, el ME está formado por los eventos probabilísticos, y las asociaciones vinculan las reglas del MA con fórmulas lógicas construidas a partir de eventos del ME; de esta manera, por cada escenario posible del ME se satisface un subconjunto de reglas del MA (aquellas reglas que tienen asociadas fórmulas que se satisfacen con los valores de los eventos del escenario en particular). Finalmente, el modelo nos permite obtener un intervalo de probabilidad $[l, u]$ para cada literal *lit* del MA, lo que significa que l se deriva a partir del modelo DeLP3E con una probabilidad $p \in [l, u]$. El límite inferior de ese intervalo, l, es la suma de las probabilidades de todos los escenarios donde el literal *lit* es válido, y el límite superior, u, es igual a uno menos la suma de las probabilidades de todos los escenarios donde es válido el complemento de *lit*. En el Capítulo 2 se presenta la descripción completa y detallada de cada componente en particular.

Un modelo como el detallado presenta muchas ventajas y ofrece las herramientas necesarias para modelar y razonar sobre dominios complejos; sin embargo, como se verá en la siguiente sección, presenta algunas desventajas y problemas al momento de computar las respuestas. El principal objetivo de esta tesis, como detallamos a continuación, es el estudio y desarrollo de nuevos enfoques para computar las respuestas en el formalismo.

1.2. Objetivos

El formalismo DeLP3E nos permite modelar diferentes situaciones considerando la inconsistencia, incertidumbre, e incompletitud de la información disponible. A partir de estas capacidades, es posible modelar dominios complejos y razonar sobre ellos, por ejemplo, en el análisis de ciberamenazas. Algunas de las funcionalidades y casos de uso posible consisten en establecer los valores de los eventos observados en el sistema (evidencia observada) y en base a eso monitorear las probabilidades de un conjunto de consultas

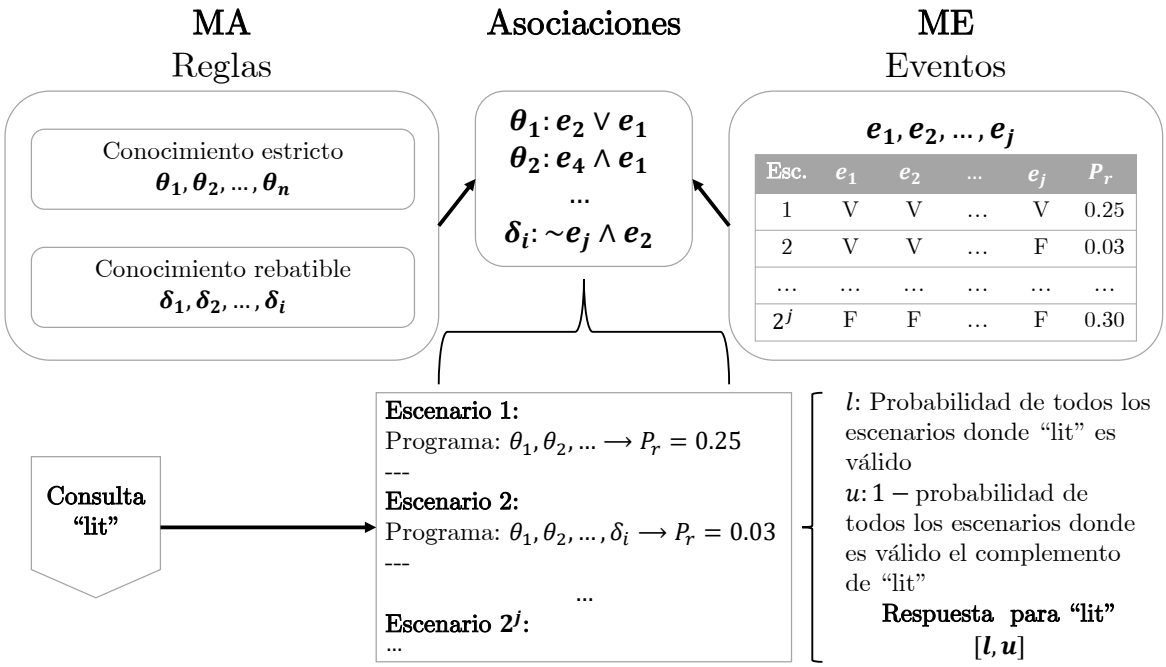


Figura 1.1: Modelo DeLP3E.

registradas. Siguiendo con el dominio de la ciberseguridad como ejemplo general, esta es una herramienta valiosa para los analistas, quienes pueden registrar consultas sobre estrategias de mitigación y técnicas de ataque de interés actual. Los resultados pueden informar, por ejemplo, los niveles de alerta de seguridad y las prioridades del esfuerzo de actualizaciones para los administradores del sistema. Otra funcionalidad importante, basada en la misma configuración que para las consultas registradas, le permitiría al usuario realizar un análisis contrafáctico. En este caso, en lugar de tomar datos y configuraciones de variables del ME de observaciones directas, el sistema permite al usuario especificar escenarios como desee y, a partir de esto, mostrar las probabilidades resultantes.

Además de estas capacidades, cada respuesta puede estar acompañada con una *explicación* sobre cómo llegó el sistema a esa respuesta. En este sentido, un sistema que implemente este modelo puede ofrecer una explicación basándose en los escenarios más probables o en las reglas del MA. En el primer caso, el sistema puede calcular un subconjunto de escenarios más probables dado el conjunto actual de observaciones, lo que le permite al analista conocer por qué una determinada consulta está relacionada con un intervalo de probabilidad en particular. Por ejemplo, un analista puede estar interesado en saber por qué el límite superior es más bajo de lo esperado, y una primera explicación

sería que se le muestre un escenario de alta probabilidad en el que se implique la negación de la consulta. Si se necesitan más detalles, también se pueden derivar explicaciones analizando las reglas y argumentos involucrados en las derivaciones; esto tiene que ver con el segundo tipo de explicaciones. En el segundo caso, además de analizar los escenarios más probables, otra posibilidad es mostrar las estructuras formadas por las reglas del MA que sustentan la consulta en el programa generado por un escenario en particular o un subconjunto de los mismos. Esto le proporciona al analista el conjunto de reglas, hechos y presunciones involucradas en la derivación, y precisamente qué papel desempeñaron, lo que puede resaltar la necesidad de revisar uno o más de estos componentes (considere, por ejemplo, hechos que provienen de una fuente de datos desactualizada que necesita ser actualizada o eliminada).

Toda esta información le proporciona a los analistas los medios para aceptar con mayor confianza la respuesta obtenida o, por otro lado, revisar la información o conocimiento involucrado que no se aplica a la situación actual. En el Capítulo 2 se presenta un caso de uso que ilustra algunas de estas capacidades.

Teniendo en cuenta estas herramientas, tanto de modelado como de la posibilidad de ofrecer explicaciones a los estados del sistema, el principal objetivo de esta tesis es el estudio de los mecanismos eficaces y eficientes para computar las respuestas en el formalismo, además de presentar formalmente las funcionalidades detalladas anteriormente para que puedan ser desplegadas en un sistema real. En particular, estudiar y proponer mecanismos que nos permitan aproximar las respuestas por medio de algoritmos que se basen en toda la información disponible de cada componente del modelo. Nos enfocamos en investigar este aspecto dado que, dar una respuesta exacta a una consulta en DeLP3E consiste en computar el intervalo de probabilidad asociado a la consulta realizada, y para esto es necesario recorrer cada escenario posible del ME y, además, por cada uno de ellos consultar por el estado de la consulta, y en base a dicho estado, actualizar los límites del intervalo; dicho procedimiento, como se detalla en el Capítulo 3, no es aplicable en un tiempo razonable para instancias grandes. Es por esto que el objetivo principal para el desarrollo de esta tesis es el estudio y definición de técnicas, junto con una guía para seleccionar cúal de ellas aplicar, que nos permitan aproximar el valor del intervalo para una respuesta en base a toda la información disponible de cada componente del modelo DeLP3E.

1.3. Contribuciones de esta tesis

En esta sección, se enumeran las contribuciones de esta tesis junto con la mención al capítulo donde se desarrollan las mismas:

1. Se realiza un estudio sobre el problema de aproximar el intervalo de probabilidad exacto en el modelo DeLP3E, obteniendo como resultado final dos enfoques de aproximación, el *enfoque basado en mundos* y el *enfoque basado en subprogramas*. También se presentaron dos algoritmos de aproximación (los mismos implementan los enfoques mencionados anteriormente), y una métrica para determinar la calidad de una aproximación generada por esos algoritmos.

   - Se define un conjunto de métricas con el objetivo de poder cuantificar la complejidad y el tamaño de cada componente de un modelo DeLP3E. Se define y clasifica cada métrica según el atributo que miden y si es posible computarla de manera tratable o aproximarse.
   - Se presenta un árbol de decisión que tiene como finalidad servir de guía para la selección del algoritmo de aproximación en base al valor de las métricas mencionadas y el costo de computarlas.

   Estos puntos se abordan en el Capítulo 3.

2. Se desarrolla e implementa un conjunto de generadores de modelos DeLP3E. Estos generadores tienen la capacidad de, a través de la configuración de sus parámetros, hacer variar las métricas de complejidad y tamaño de cada componente a generar. Estos temas se desarrollan en el Capítulo 4.

3. Se ejecuta una batería de pruebas empíricas para analizar el comportamiento de los algoritmos de aproximación bajo diferentes niveles de complejidad. Se ejecutaron cada una de estas evaluaciones sobre algunos caminos posibles del árbol de decisión mencionado anteriormente. Este tema se aborda en el Capítulo 5.

4. Se presenta la plataforma P-DAQAP, una plataforma web para facilitar el análisis del proceso argumentativo en Delp y la argumentación probabilística a través del modelo DeLP3E. Esto se detalla en el Capítulo 6.

1.4. Publicaciones surgidas de esta tesis

A continuación, se enumeran los trabajos científicos obtenidos durante el proceso de elaboración de esta tesis.

- "DAQAP: Defeasible Argumentation Query Answering Platform" [LSG+19]: **Mario A. Leiva**, Gerardo I. Simari, Sebastian Gottifredi, Alejandro J. García, Guillermo R. Simari. 13th International Conference on Flexible Query Answering Systems, 2019.
- "Cyber Threat Analysis with Structured Probabilistic Argumentation" [LSSS19]: **Mario A. Leiva**, Gerardo I. Simari, Guillermo R. Simari, Paulo Shakarian. 3rd Workshop on Advances In Argumentation In Artificial Intelligence (AI^3), 2019.
- "Towards Effective and Efficient Approximate Query Answering in Probabilistic DeLP" [LSG20]: **Mario A. Leiva**, Alejandro J. García, Gerardo I. Simari. Proceedings of the Workshop on Advances In Argumentation In Artificial Intelligence (AI^3), 2020.
- "Probabilistic Defeasible Logic Programming: Towards Explainable and Tractable Query Answering (Extended Abstract)" [LGSS21]: **Mario A. Leiva**, Alejandro J. García, Paulo Shakarian, Gerardo I. Simari. 37th International Conference on Logic Programming (ICLP), 2021. Este trabajo también fue presentado durante el consorcio doctoral (Doctoral Consortium) del mismo evento.
- "Argumentation-based Query Answering under Uncertainty with Applications to Cybersecurity": **Mario A. Leiva**, Alejandro J. García, Paulo Shakarian, Gerardo I. Simari. 2021 (En evaluación para una revista internacional).
- "A Principled Approach to Automatic Generation of DeLP Programs": **Mario A. Leiva**, Gianvincenzo Alfano, Gerardo I. Simari, Guillermo R. Simari. 2022 (En preparación para revista internacional).
- "Empirical Analysis of Approximate Query Answering over Probabilistic DeLP Knowledge Bases": **Mario A. Leiva**, Gerardo I. Simari, Guillermo R. Simari. 2022 (En preparación para revista internacional).

1.5. Organización de la tesis

Esta tesis se organiza de la siguiente manera:

- En el Capítulo 2 se presentan los conceptos centrales para el desarrollo de esta tesis. En la Sección 2.1 se describen los formalismos de argumentación y su aplicación en el dominio de la ciberseguridad. Luego, en la Sección 2.2 se presenta una revisión sobre la relación entre la argumentación y los modelos probabilísticos (argumentación probabilística). En la Sección 2.3 se describen brevemente los modelos probabilísticos, en particular, las redes bayesianas (ya que este modelo es el que utilizamos en esta tesis). Por último, en la Sección 2.4, se presenta el formalismo DeLP3E detallando cada uno de sus componentes acompañado con un ejemplo de aplicación en ciberseguridad.

- En el Capítulo 3 se estudia en detalle el proceso de generar una respuesta a una consulta en un modelo DeLP3E, es decir, como computar el intervalo de probabilidad exacto para un literal en particular. En la Sección 3.1 se comenzará presentando la intratabilidad inherente al problema de computar el intervalo de probabilidad para responder a una consulta. En la Sección 3.2, se introduce un conjunto de métricas para cuantificar la complejidad de cada componente del modelo DeLP3E (ME, MA, y Función de Anotación), dos algoritmos de aproximación (*Muestreo por mundos* y *Muestreo por subprogramas*), y una métrica de calidad que determina qué tan buena es una aproximación obtenida. Por último, en la Sección 3.3, se presenta un árbol de decisión que tiene como finalidad servir de guía para la selección del algoritmo a utilizar para aproximar el valor del intervalo de probabilidad. Este árbol de decisión tiene en cuenta la complejidad de los componentes de la base de conocimiento según el valor de sus métricas y el tiempo requerido para acceder a ella.

- En el Capítulo 4 se presentan tres generadores para crear modelos DeLP3E que nos permiten generar diferentes escenarios de complejidad de manera automática, para luego ejecutar los algoritmos de aproximación propuestos en el Capítulo 3. En la Sección 4.1 se presenta una breve descripción de los motivos para crear estos generadores; en las Secciones 4.2, 4.3 y 4.4, se presentan el *Generador de Programas PreDeLP*, el *Generadores de Redes Bayesianas*, y el *Generador de Funciones de Anotación* respectivamente. En la sección correspondiente a cada generador, se

detallan las decisiones de diseño, los parámetros, algoritmos, y ejemplos de modelos generados (también se muestra una comparativa de dichos modelos en base al valor de sus métricas). Finalmente, en la Sección 4.5, se presentan los conjuntos de modelos generados que serán usados en los experimentos del próximo capítulo.

- En el Capítulo 5 se presenta el diseño, ejecución, y análisis de un conjunto de pruebas empíricas con el objetivo de comparar el comportamiento de los algoritmos de aproximación presentados sobre diferentes modelos DeLP3E. Dichos algoritmos son los presentados en el Capítulo 3, y los modelos DeLP3E fueron generados a través de las herramientas presentadas en el Capítulo 4. El objetivo de este capítulo es comparar los algoritmos de aproximación sobre diferentes modelos DeLP3E para analizar su comportamiento con respecto a diferentes criterios. Con esto, validamos y exploramos alternativas sobre la diferentes líneas del árbol de decisión presentado al final del Capítulo 3. En la Sección 5.1 se presenta el diseño general del experimento, es decir, qué se busca analizar y la manera de realizar las pruebas. En la Subsección 5.1.1 se detalla cómo se construyó el conjunto de pruebas, y en la Subsección 5.1.2 se listas las métricas de desempeño que serán evaluadas. Finalmente, en la Sección 5.2 se presentan los resultados del experimento.

- En el Capítulo 6 se presenta P-DAQAP (Probabilistic Defeasible Argumentation Query Answering Platform), una plataforma web para *Respuesta a Consultas en Argumentación Rebatible Probabilística*, la cual ofrece una interfaz visual y un panel de control que facilita el análisis del proceso argumentativo llevado a cabo en la Progamación Lógica Rebatible (DeLP) [Gar97] y la argumentación probabilística a través del modelo DeLP3E [SSM+16]. En la Sección 6.1 se describe la motivación principal para el desarrollo de la plataforma; en la Sección 6.2 se presenta un caso de uso de la extensión DeLP3E para el análisis de ciberamenazas usando un conjunto de datos públicos; finalmente, en la Sección 6.3 se presenta la plataforma, detallando su arquitectura, el análisis sobre el programa DeLP y los grafos de Dung, y las funcionalidades que representan la extensión DeLP3E.

- En el Capítulo 7, se presentan los trabajos relacionados a los puntos centrales de esta tesis. En la Sección 7.1, se presentarán algunos trabajos que vinculan la *teoría de la argumentación* y la *teoría de la probabilidad* para modelar y razonar en dominios complejos con presencia de incertidumbre e incompletitud. Luego, en la Sección 7.2, se mencionarán algunos trabajos vinculados a la generación automática de bases

de conocimiento para formalismos de argumentación estructurada, esto en relación al generador de programas PreDeLP presentado en el Capítulo 4. Por último, y en relación a la plataforma P-DAQAP presentada en el Capítulo 6, en la Sección 7.3 se presentarán algunas herramientas que fueron diseñadas para analizar la estructura de diferentes tipos de razonamiento en argumentación estructurada.

- Finalmente, en el Capítulo 8 se presentan las conclusiones y los trabajos futuros.

1.6. Sumario

En este capítulo introductorio se presentó uno de los principales problemas que tienen las bases de conocimiento y los modelos para razonar sobre ellas en dominios complejos. Se introdujo el formalismo DeLP3E como un modelo para dar respuesta a los problemas mencionados, señalando como principal motivación el estudio de técnicas de aproximación para computar las respuestas exactas en dicho formalismo. También se listaron las contribuciones de esta tesis y los trabajos científicos surgidos de la misma. Finalmente, se presentó la organización de este documento.

Capítulo 2

Conceptos preliminares

En este capítulo se presentarán brevemente los conceptos centrales para el desarrollo de esta tesis. En la Sección 2.1 se comenzará con una descripción de los formalismos de argumentación y su aplicación en el dominio de la ciberseguridad. Luego, en la Sección 2.2 se hará una revisión sobre la relación entre la argumentación y los enfoques probabilísticos (argumentación probabilística), luego, en la Sección 2.3 se presentará una breve descripción de los modelos probabilísticos gráficos, en particular, redes bayesianas. Por último, en la Sección 2.4, presentaremos el formalismo DeLP3E detallando cada uno de sus componentes acompañado con un ejemplo de aplicación en ciberseguridad.

2.1. Argumentación en Inteligencia Artificial

En los últimos años, los problemas en el dominio de la seguridad de la información crecieron tanto en número como en variedad y complejidad [PTSM21]. Esto generó un mayor interés en aplicar modelos de Inteligencia Artificial (IA) orientados a la detección tanto de riesgos de seguridad simples como de ciberataques sofisticados, que cuenten con la capacidad para modelar información de naturaleza compleja (variedad de formatos, incompleta, inconsistente, de naturaleza cualitativa, entre otros). En la actualidad, existen muchas aplicaciones en las que la IA tiene un impacto positivo; sin embargo, son bien conocidas sus limitaciones al momento de ofrecer explicaciones o una trazabilidad del proceso llevado a cabo para generar una salida (esto es particularmente notorio en el área de aprendizaje automatizado) [GA19]. Dada la naturaleza de estos problemas, es clara la

necesidad de comprender los procesos de toma de decisión de estos sistemas, lo cual se conoce generalmente como IA explicable (XAI, por sus siglas en inglés). En este sentido, los usuarios de herramientas basadas en IA no solo necesitan hacer un uso efectivo de ellas, sino que también necesitan apoyo que les permita comprender cómo el sistema llegó a una conclusión. Es aquí donde la Teoría de la Argumentación como formalismo para representar y procesar conocimiento de dominios complejos es una herramienta que ofrece grandes ventajas.

Tradicionalmente, la argumentación ha sido un tema de interés interdisciplinario en áreas como la filosofía, psicología, ciencias de la comunicación, lingüística, leyes, economía, y otras ciencias no técnicas. Esto cambió con el surgimiento de los agentes inteligentes autónomos y los sistemas multiagente, donde el objetivo es que un computador decida y actúe en lugar de un ser humano, lo que ocasionó que la argumentación se convierta en un tópico de investigación en ciencias de la computación, particularmente en la comunidad de IA.

La *argumentación* es una disciplina que forma parte del área de razonamiento no monótono en la cual la representación del conocimiento se realiza a través de la especificación de *argumentos*. Cada uno de estos argumentos da soporte a una *conclusión* a partir de un conjunto de *premisas*. Un formalismo argumentativo brinda la posibilidad de representar conocimiento en contradicción, dado que un argumento puede estar en conflicto con otro argumento. La ventaja de un formalismo con esas características es la tolerancia a la inconsistencia. Es decir, es posible generar conclusiones válidas a partir de conocimiento que puede ser parcialmente inconsistente, ya que el proceso argumentativo decidirá qué información prevalece frente a una situación de conflicto. Dicho proceso consiste en *razonar* a partir de la construcción y evaluación de argumentos que soportan conclusiones contradictorias. Esto último es objeto de estudio de la *Argumentación Rebatible*.

La argumentación rebatible es una formalización del razonamiento rebatible [SL92], donde el concepto principal es el de argumento. En particular, un argumento para una conclusión C constituye una pieza de razonamiento tentativa que un agente inteligente está dispuesto a aceptar para explicar C. Si el agente adquiere luego nueva información, la conclusión C junto con el argumento que la soporta podrían quedar invalidados. En un sistema argumentativo rebatible la validez de una conclusión C será garantizada cuando exista un argumento que brinde una justificación válida para C. Este proceso involucra al construcción de un argumento $\mathcal{A}$, para C, que no se encuentre derrotado. En este sentido,

para verificar si el argumento $\mathcal{A}$ está derrotado, se construyen contraargumentos que son posibles derrotadores de $\mathcal{A}$. Como estos derrotadores son argumentos, se debe verificar que no estén a su vez derrotados, y así siguiendo. De esta manera, se modela el proceso de razonamiento en el cual se producen y se evalúan argumentos a favor y en contra de una conclusión para verificar la garantía de dicha conclusión [SL92].

Los formalismos argumentativos crean modelos para representar situaciones del mundo real. Cada uno de estos modelos posee diferentes niveles de abstracción, dependiendo del dominio de estudio para el cual fue creado. Por ejemplo, el *Marco Argumentativo Abstracto* propuesto por Phan Minh Dung en [Dun95], se define como un par compuesto por un conjunto de argumentos y una relación binaria que representa las relaciones de ataque entre argumentos. Aquí, un argumento es una entidad abstracta cuyo rol está determinado únicamente por sus relaciones con otro argumento. Para determinar qué argumento puede sobrevivir al conflicto, se necesita un método sistemático bien definido; tales métodos formales para identificar los resultados de los conflictos para cualquier marco de argumentación se denominan *semánticas de argumentación*. En la literatura existen dos enfoques principales para la definición de semánticas de argumentación: el enfoque *basado en etiquetas* y el enfoque *basado en extensiones* [BCG11]. La idea subyacente al enfoque basado en *etiquetas* es la de asignar a cada argumento una etiqueta. Una posible elección para el conjunto de etiquetas es: *adentro*, *afuera*, e *indeciso*, donde la etiqueta *adentro* significa que el argumento es *aceptado*, *afuera* significa que el argumento es *rechazado*, e *indeciso* significa que si el argumento es aceptado o rechazado es una cuestión de opinión. La idea del enfoque basado en *extensiones* es la identificación de conjuntos de argumentos, llamados *extensiones*, que pueden sobrevivir juntos al conflicto y, por lo tanto, representar colectivamente una postura razonable que podría tomar un razonador autónomo.

Si se busca una formalización más detallada de la estructura interna de los argumentos que la disponible en la argumentación abstracta, se puede recurrir a la *argumentación estructurada*. En la literatura existen algunos formalismos que se basan en esta idea; algunos de ellos son ABA [Ton14], ASPIC+ [MP14], Programación Lógica Rebatible [GS04] (DeLP, por sus siglas en Inglés), y la Programación Deductiva [BGH$^+$14]. En particular, DeLP ofrece un sistema de razonamiento computacional que usa un mecanismo basado en la argumentacón para obtener respuestas a partir de una base de conocimiento usando un lenguaje de programación lógica extendido con reglas rebatibles. Esta combinación genera un sistema computacionalmente efectivo junto con un modelo de razonamiento

similar al utilizado por los humanos que facilita su uso en aplicaciones del mundo real. La herramienta que presentamos en este capítulo se basa en *DeLP* y *PreDeLP*, más la extensión para manejar el modelo del entorno y la función de anotación; por otro lado, nos centramos en el enfoque basado en *extensiones* para calcular las extensiones de los grafos de Dung generados a partir de programas DeLP.

En un escenario ideal, los argumentos se construyen sólo a partir de hechos o premisas generalmente aceptadas. Pero, como ya mencionamos anteriormente, existen dominios donde es necesario representar y manipular información incierta, incompleta y hasta contradictoria. Para estos casos, una teoría lógica argumentativa más expresiva debería considerar presunciones, premisas y hechos con un grado de incertidumbre asociado, a partir de los cuales se puedan construir argumentos valuados cuantitativamente [Hae09]. Un área que estudia este enfoque en particular es la *argumentación probabilística*.

2.2. Argumentación Probabilística

Como se menciona anteriormente, la argumentación está inherentemente impregnada de incertidumbre. Uno de los conceptos centrales del razonamiento rebatible y, por lo tanto, de la argumentación es lo falible de la percepción, que nos obliga a ser capaces de razonar incluso con información incompleta y estar preparados para retractarnos de nuestras conclusiones ante nuevos datos. Esto se complica aún más al aplicar la argumentación en situaciones de la vida real, donde la incertidumbre puede surgir como resultado del contexto en el que se usa la argumentación, los tipos de agentes que están involucrados, los tipos de argumentos que se usan, entre otros. Así como existen múltiples fuentes de incertidumbre en la argumentación, existen en la literatura múltiples propuestas para modelar este tipo de razonamiento.

Uno de los enfoques más desarrollados es el de la *argumentación probabilística*, el cual ofrece medios para cuantificar el nivel de incertidumbre presente en el dominio a tratar. Si bien el uso de la teoría de la probabilidad como un medio para modelar aspectos inciertos de la argumentación tiene sus puntos cuestionables [Par99], los desarrollos actuales en el campo mostraron su capacidad en escenarios reales [Ver14].

En la argumentación existen varios ejemplos que pueden verse beneficiados al incorporar probabilidades durante el proceso de razonamiento: cuando la incertidumbre surge

como consecuencia de la ambigüedad (una manera de imprecisión en el lenguaje utilizado en los argumentos); cuando se trabaja con argumentos del mundo real presente en el lenguaje natural, ya que generalmente son entimemas; o cuando la incertidumbre inherente se refiere al grado en que un agente cree o no en un argumento, o sus premisas, o su conclusión. En la vida real, las personas a menudo tienden a confiar o estar de acuerdo con ciertas cosas solo hasta cierto punto, y la presentación de argumentos en contra puede debilitar esta creencia en lugar de conducir al rechazo absoluto de un argumento determinado. Esto ejemplos, aunque sencillos, son bastantes comunes y reflejan la ambigüedad y la incertidumbre que están presentes en la vida diaria. También representan desafíos que cualquier aplicación de la argumentación, en particular una que involucre a agentes humanos, debe abordar [HPP+21].

Existen varios enfoques para modelar dichas situaciones usando argumentación probabilística. Gran parte de lo trabajos en esta área adoptan los sistemas de argumentación abstracta como la base para la extensión probabilística [LON11, Thi12, Hun12, FFP13]. En la argumentación abstracta, a diferencia de la argumentación estructurada (que es el enfoque adoptado en esta tesis), no se considera nada acerca de la composición interna de cada argumento, sino que se centra en el estudio de las relaciones de ataque entre los argumentos. Por otro lado, en la argumentación estructurada, se definen los argumentos y sus relaciones en términos de la estructura interna asumida por estos argumentos y, a menudo, ofrecen las herramientas para construirlos a partir de una base de conocimiento subyacente o una fuente de datos [BH08, MP14, Ton14, GS04].

Considerando los sistemas de argumentación abstracta, existen varios enfoques para incorporar información probabilística. En [Hun13], por ejemplo, los autores discuten dos enfoques: el enfoque de *constelaciones* y el enfoque *epistémico*. En el enfoque de constelaciones, la incertidumbre está en la topología del grafo (se consideran grafos de argumentación probabilísticas como una extensión de los grafos de argumentación abstracta) por lo que ciertos argumentos o relaciones aparecen en dicho grafo solo con una probabilidad dada. Este enfoque es útil cuando un agente no está seguro de qué argumentos y ataques conoce otro agente, o si la ambigüedad o imprecisión de los argumentos genera incertidumbre en la estructura del grafo. En el enfoque epistémico, la topología del grafo de argumentos es fija, pero existe incertidumbre en cuanto al grado en que se cree en cada argumento. Este método se puede aprovechar cuando un agente no está seguro de su propia opinión o de la de otro agente sobre los argumentos, si una situación dada requiere

una forma detallada de juzgar los argumentos, o si el razonamiento percibido de un agente dado escapa a la semántica de argumentación clásica. Los resultados computacionales de estos dos enfoques se detallan en la Sección 7.1 del Capítulo 7.

Por otro lado, en [Hae09] y [SSM+16], los autores presentan enfoques para usar información probabilística en sistemas de argumentación estructurada para razonar bajo incertidumbre. En [Hae09] se propone una teoría de argumentación probabilística donde la credibilidad (o "peso") de los argumentos se mide en base a valores probabilísticos. En ese trabajo, se consideran las premisas como fuentes atómicas de incertidumbre, y que depende en concreto del problema o modelo sí son independientes o no. Dichas premisas se representan como *variables probabilísticas*, y al restringir los valores de dichas variables se conforman *presunciones*, a su vez, la combinación de presunciones (denominados *escenarios*) permite la construcción de argumentos. Este proceso de construcción de argumentos a partir de presunciones inciertas se conoce generalmente como razonamiento *basado en presunciones* o *hipotético* [Bon88, KM93, LL89]. El objetivo general en la argumentación probabilística que se presenta en [Hae09] es evaluar cuantitativamente la verdad o falsedad de una proposición a través de un proceso de inferencia argumentativo. La parte cualitativa de la información disponible o evidencia, a partir de la cual se construyen los argumentos, se denomina *base de conocimiento*, y la proposición a evaluar se denomina *hipótesis*. La base de conocimiento y la hipótesis se expresan a través de sentencias lógicas en un lenguaje apropiado. Para una hipótesis dada, el primer paso es construir los argumentos y contraargumentos. Cada argumento (contraargumento) ofrece una prueba lógica que soporta la verdad (falsedad) de la hipótesis, en el sentido en que la hipótesis (el complemento de la hipótesis) se convierte en una consecuencia lógica a partir de la agregación del argumento a la base de conocimiento. La combinación de argumentos y contraargumentos naturalmente puede conducir a una contradiccón lógica. Aquí es donde la argumentación probabilística propone una solución precisa de cómo tratar esta contradicción. Una vez que la parte cualitativa del análisis está completa (generación de argumentos, contraargumentos y conflictos), lo que sigue es la parte cuantitativa para asignar los valores probabilísticos a cada argumento y así poder resolver las situaciones de conflicto. Como solución general, la teoría de la argumentación probabilística propone una medida no aditiva denominada *grado de soporte* y su contraparte denominada *grado de posibilidad*, la primera mide la credibilidad de la hipótesis en base al total de las probabilidades asociadas a los argumentos que la soportan, mientras que la segunda lo hace en base a las probabilidades asociadas a los contraargumentos de la hipótesis. Esto puede

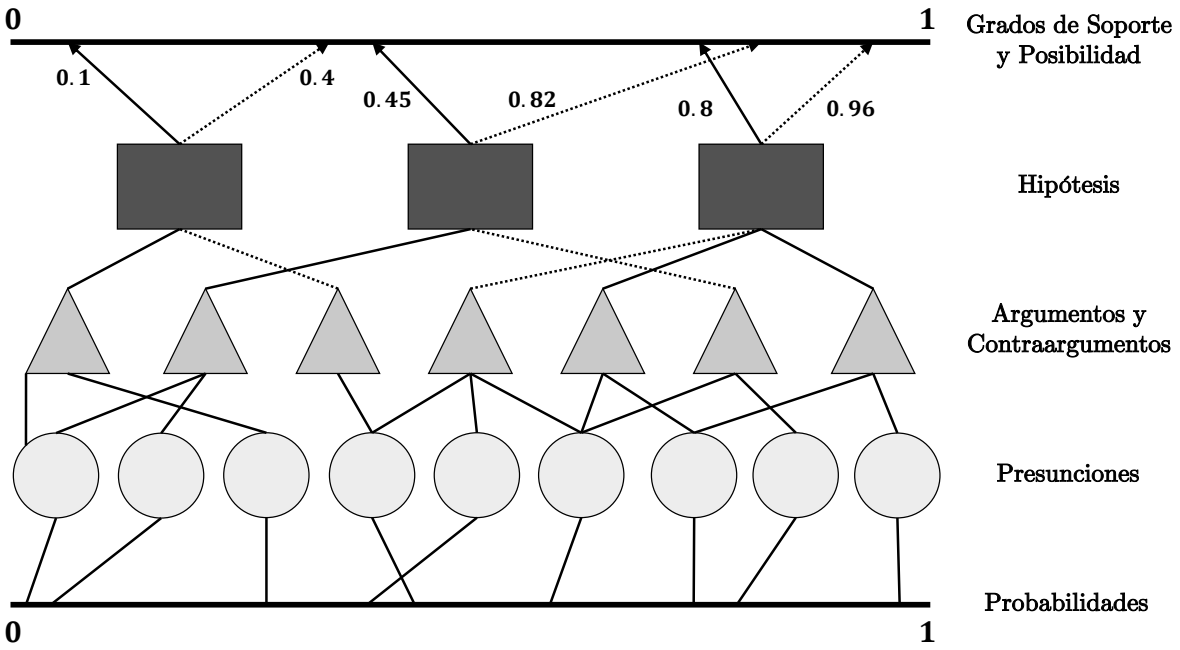


Figura 2.1: Las relaciones entre presunciones y argumentos, argumentos e hipótesis (arco sólido), contraargumento e hipótesis (arco punteado), hipótesis y grado de soporte (arco sólido), e hipótesis y grado de posibilidad (arco punteado). Figura reproducida del trabajo de Haenni en [Hae09].

verse en la Figura 2.1 tomada de [Hae09].

El modelo presentado en [SSM+16] combina la lógica probabilística de Nilsson [Nil86] y la programación lógica rebatible con presunciones [MGS12]. La lógica probabilística se utiliza para definir un *modelo del entorno* que captura el conocimiento específico del dominio que puede ser incierto. La programación lógica rebatible con presunciones se utiliza para definir un *modelo analítico* que contiene conocimiento básicos generales que pueden ser inconsistentes. Ambos modelos se combinan a través de una *función de anotación* que relaciona los elementos del modelo del entorno con el modelo analítico. El modelo que se presenta en este último trabajo se denomina DeLP3E y fue utilizado en el desarrollo de esta tesis. Una presentación completa y formal de este formalismo se presenta en la Sección 2.4. Una de las características más importante del modelo DeLP3E es que adopta una división para la base de conocimiento, esta división nos permite modelar, por un lado, todo el conocimiento que no es inherentemente probabilístico, es decir, el conocimiento incierto, y, por otro lado, el conocimiento asociado a eventos probabilísticos. Esto nos

permite tener una separación clara entre ambos modelos. Para poder desarrollar completamente este formalismo, en particular el modelo del entorno, es importante recordar algunas nociones sobre modelos probabilísticos.

2.3. Modelos Probabilísticos

Las representaciones basadas en modelos son un componente fundamental en muchos campos, en particular en aquellos donde los sistemas son complejos y que implican una cantidad significativa de incertidumbre. Recordemos que esta incertidumbre surge generalmente debido a las limitaciones en nuestra capacidad para observar el mundo, las limitaciones de nuestra capacidad para modelarlo y posiblemente debido al no determinismo propio del dominio. Debido a esta incertidumbre sobre el verdadero estado del mundo, es necesario que los sistemas de razonamiento consideren diferentes "posibilidades". Un enfoque es considerar cualquier estado posible del mundo, pero Desafortunadamente, rara vez ocurre que podamos eliminar por completo un estado como imposible dadas nuestras observaciones. Por otro lado, para obtener conclusiones significativas, necesitamos razonar no solo sobre lo que es posible, sino también sobre lo que es "probable".

El cálculo de la teoría de la probabilidad nos proporciona un marco formal para considerar múltiples resultados posibles y su probabilidad. Define un conjunto de posibilidades mutuamente excluyentes y exhaustivas, y le asocia a cada una de ellas un valor de probabilidad (un número entre 0 y 1, por lo que la probabilidad total de todas las posibilidades es 1). Este marco nos permite considerar opciones que son poco probables, pero no imposibles. Por otro lado, los sistemas complejos se caracterizan por la presencia de múltiples aspectos interrelacionados, muchos de los cuales están involucrados en las tareas de razonamiento. Los dominios de esos aspectos se pueden caracterizar en términos de un conjunto de "variables aleatorias", donde el valor de cada variable define una propiedad importante del dominio. El conjunto de posibles variables y sus valores es una decisión de diseño importante y depende en gran medida de las preguntas que deseemos responder sobre el dominio. De esta manera, la tarea consiste en razonar "probabilísticamente" sobre el valor de una o más de las varables, posiblemente teniendo en cuenta las observaciones sobre algunas otras. Para hacerlo utilizando el razonamiento probabilístico basado en principios, necesitamos construir una *distribución conjunta* sobre el espacio de posibles asignaciones de algunas variables aleatorias X. Este tipo de modelo nos permite responder

a una amplia gama de preguntas interesantes. Por ejemplo, podemos tener la observación de que una variable X_i toma el valor específico x_i, y preguntar, en la *distribución posterior* resultante, cuál es la distribución de probabilidad sobre los valores de otra variable X_j.

Los *modelos probabilísticos gráficos* utilizan una representación basada en grafos como base para codificar de forma compacta una distribución compleja en un espacio de grandes dimensiones. En esta representación gráfica, los nodos corresponden a las variables de nuestro dominio, y los arcos corresponden a interacciones probabilísticas directas entre ellas. Tal como se detalla en [KF09], hay una enfoque dual que se puede utilizar para interpretar la estructura de estos grafos. Por un lado, el grafo es una representación compacta de un conjunto de *independencias* que se mantienen en la distribución; estas propiedades toman la forma X es independiente de Y dado Z, para algún subconjunto de variables X, Y, Z. El otro enfoque es que el grafo define un "esqueleto" para representar de manera compacta una distribución de grandes dimensiones: en lugar de codificar la probabilidad de cada posible asignación a todas las variables de nuestro dominio, es posible "dividir"la distribución en "factores"más pequeños, cada uno sobre un espacio de posibilidades mucho menor. A partir de esto, se puede definir la distribución conjunta general como un producto de estos factores.

Este marco tiene muchas ventajas. En primer lugar, permiten escribir la distribución de forma tratable, incluso en el caso de que la representación explícita de la distribución conjunta sea muy grande grande. En segundo lugar, la misma estructura permite usar la distribución de manera efectiva para realizar "inferencias"; en particular, se proporcionan algoritmos para calcular la *probabilidad posterior* de algunas variables dada la evidencia de otras. Por último, este marco facilita la construcción , ya sea por un experto humando o de manera automática, de estos mismos modelos al "aprender" de los datos. Estos tres componentes (representación, inferencia y aprendizaje) son elementos críticos para la construcción de un sistema inteligente: es necesario contar con una representación declarativa que sea una codificación razonable del dominio a estudiar, se necesita usar esa representación de manera efectiva para responder a una amplia gama de preguntas que son de interés y, es necesario poder generar la distribución combinando el conocimiento del experto y los datos acumulados. Los modelos gráficos probabilísticos son uno de los pocos modelos que admiten las tres capacidades para una gama amplia de problemas.

Describimos brevemente una familia de representación gráfica de distribuciones denominada *Redes Bayesianas* (RB). Esta representación utiliza un grafo dirigido (donde las

aristas tienen un origen y un destino), tal como se muestra en la Figura 2.2. Utilizaremos esta representación para el modelo del entorno en DeLP3E.

Redes Bayesianas

Las redes bayesianas se basan en las mismas intuiciones que el modelo elemental de Bayes (*Naive Bayes*, en inglés) al explotar las propiedades de independencia condicional de la distribución para permitir una representación compacta y natural [KF09]. Sin embargo, no se limitan a representar distribuciones que satisfacen los supuestos de independencia fuerte implícitos en los modelos elementales. El concepto principal de la representación de una red bayesiana es un grafo acíclico (DAG, en inglés) G, cuyos nodos son las variables aleatorias del dominio y cuyas arcos corresponden, intuitivamente, a la influencia directa de un nodos sobre otro. De esta manera, una red bayesiana es un *grafo dirigido acíclico* donde:

- cada *nodo* representa una *variable aleatoria discreta*;
- si existe un *arco* entre el nodo X y el nodo Y, decimos que X es un *padre* de Y, y representa la *dependencia directa* entre X e Y;
- a cada nodo se le asigna una *distribución probabilística condicional* $P(X_i \mid Padres(X_i))$ que cuantifica el efecto de los nodos padre sobre la variable X_i;
- cada variable es *independiente* de sus no descendientes en el grafo, dado el estado de sus padres;
- la ausencia de un arco entre dos nodos expresa la ausencia de influencias causales entre las variables correspondientes, y la independencia probabilística (posiblemente condicional) entre ellas.

Por ejemplo, si considerando un dominio con 10 variables booleanas, tenemos una distribución conjunta de 1024 entradas. Modelando dicha distribución por medio de una red bayesiana, tenemos el grafo que se muestra en a Figura 2.2. Podemos ver este grafo como un mecanismo para muestrear, donde el valor de cada variable es seleccionado usando una distribución que depende solo de sus padres. Es decir, cada variable es una función estocástica de sus padres. Otro de los aspectos principales de una red bayesiana es un conjunto de *modelos de probabilidad local* que representan la naturaleza de la dependencia de cada variable con respecto a sus padres. En general, cada variable X está asociada con

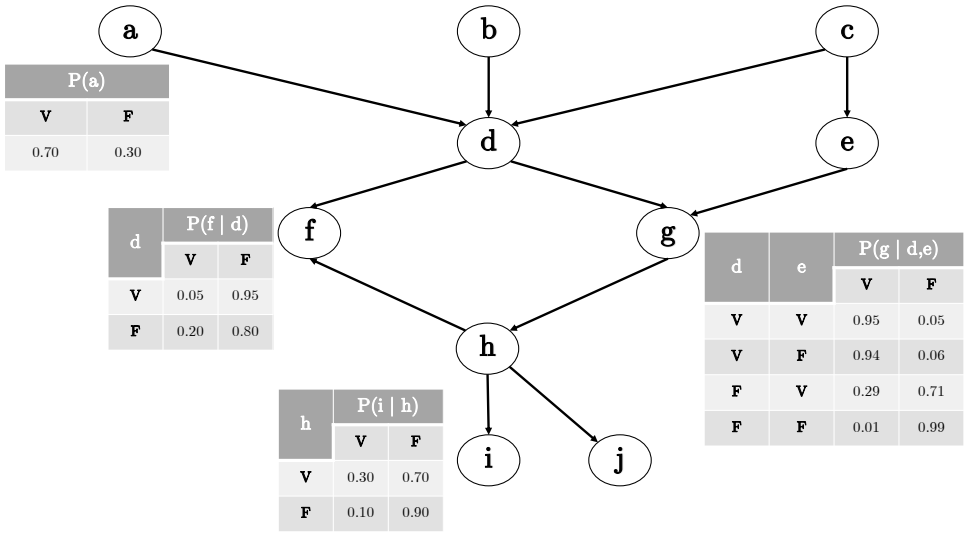


Figura 2.2: Ejemplo de Red Bayesiana.

una *distribución de probabilidad condicional* (CPT, por sus siglas en inglés) que especifica una distribución sobre los valores de X dada cada posible asignación conjunta de valores de sus padres en el modelo. Para un nodo sin padres, su CPT está condicionada por un conjunto vacío de variables. Por esto, se considera una CPT como una *distribución marginal*. En la Figura 2.2 también se muestran las tablas de probabilidad condicional de algunos nodos de la red (a, f, i y g). Una red bayesiana describe completamente una distribución, es decir, se sabe la probabilidad de cualquier conjunción de asignaciones de valores a cada variable: $P(X_1 = x_1 \wedge X_1 = x_2 \wedge \ldots X_n = x_n) = P(x_1, x_2, \ldots, x_n)$. Cada entrada en la distribución completa puede ser calculada a partir de la información presente en la red:

$$P(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | padres(X_i))$$

donde $padres(X_i)$ denota los valores de las variables en $Padres(X_i)$. Uno de los problemas más comunes a resolver dada una red bayesiana es computar la probabilidad de que un subconjunto de variables tengan valores dados (inferencia). Es a través de esta herramienta que podemos consultar la probabilidad de un subconjunto de variables que describen un mundo posible del dominio de estudio.

Con esta breve introducción a los modelos probabilísticos, y en particular a las redes bayesianas, podemos continuar presentando el formalismo DeLP3E.

2.4. Preliminares sobre DeLP3E

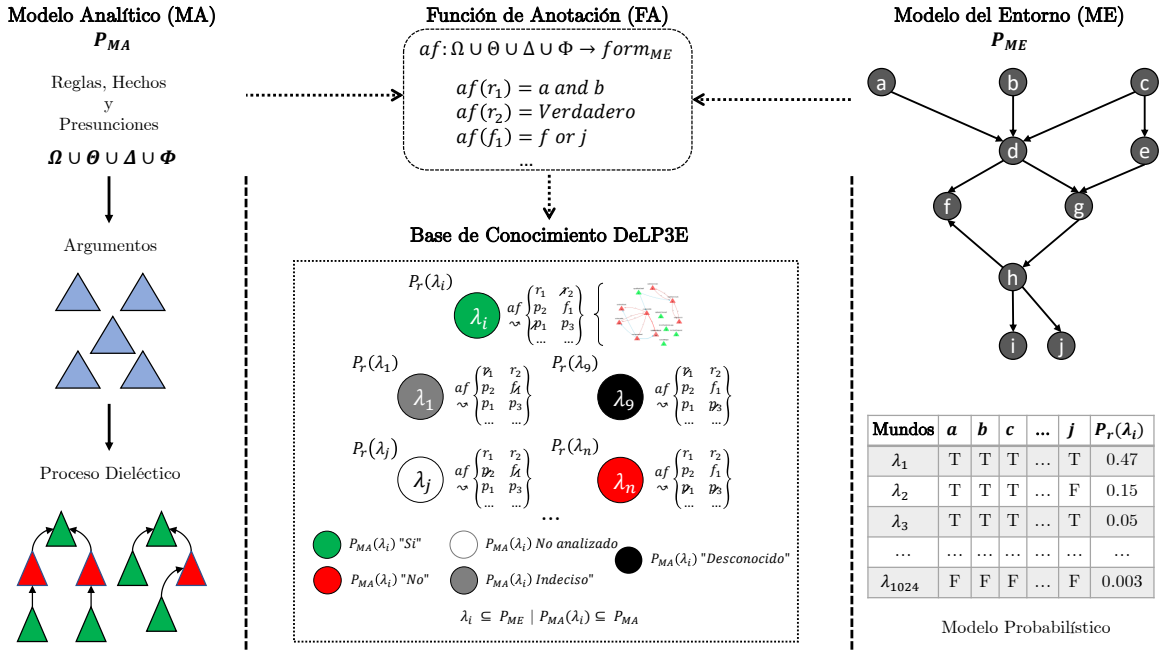
El formalismo DeLP3E está compuesto por dos modelos relacionados del mundo. El primero se denomina *modelo del entorno* (de ahora en adelante, "ME"), y es usado para describir conocimiento incierto acerca del dominio y que está sujeto a eventos probabilísticos. El segundo, denominado *modelo analítico* ("MA"), describe conocimiento del dominio que puede ser estricto o rebatible; esto será útil en el análisis de hipótesis que puedan explicar un fenómeno dado (que denominaremos "consultas").

En la Figura 2.3 se muestran los componente de un modelo DeLP3E. El MA está compuesto de un programa PreDeLP clásico [MGS12] (es decir, no probabilístico) que permite información contradictoria, dando al sistema la posibilidad de modelar explicaciones que compiten para una consulta dada. En general, el ME contiene conocimiento como evidencias, hechos inciertos, o conocimiento sobre agentes o sistemas. El MA, por otro lado, contiene conocimiento que puede o no ser estrictamente válido; sin embargo, no depende de eventos probabilísticos. Dividir el conocimiento entre MA y ME es una tarea de ingeniería del conocimiento, y para su adecuada resolución se requerirán decisiones de diseño; es importante notar que esta separación permite modelar diferentes tipos de incertidumbre – las reglas rebatibles y las presunciones pueden aprovecharse cuando no se tiene información probabilística, pero también es posible mantener una porción de la base de conocimiento como incierta.

A continuación se presentarán formalmente estos modelos, como así también cómo el conocimiento del MA puede anotarse con información del ME; estas anotaciones determinan las condiciones bajo las cuales las sentencias del MA pueden ser potencialmente verdaderas.

Lenguaje Básico

Se asume un conjunto de variables y constantes, denotadas con V y C, respectivamente. El lenguaje también contiene un conjunto n-ario de símbolos de predicados; el ME y MA usan conjuntos separados de símbolos de predicados, denotados como $\mathbf{P}_{ME}$ y $\mathbf{P}_{MA}$, respectivamente – sin embargo, los modelos pueden compartir variables y constantes. Un *término* está compuesto por una variable o una constante. Dados los términos $t_1, \ldots, t_n$ y un predicado n-ario p, $p(t_1, \ldots, t_n)$ se denomina un *átomo*; si $t_1, \ldots, t_n$ son constantes básicas, entonces se dice que el átomo está *instanciado* (a veces también llamado básico). El conjunto de todos los átomos instanciados del ME y el MA se denotan como $\mathbf{G}_{ME}$ y $\mathbf{G}_{MA}$,



Mundos	*a*	*b*	*c*	...	*j*	$P_r(\lambda_i)$
λ_1	T	T	T	...	T	0.47
λ_2	T	T	T	...	F	0.15
λ_3	T	T	T	...	T	0.05
...	...	...	...	...	...	...
λ_{1024}	F	F	F	...	F	0.003

Figura 2.3: Componentes del Modelo DeLP3E.

respectivamente; por último, $\mathbf{L}_{MA}$ denota el conjunto de todos los literales instanciados: $\{a \mid a \in \mathbf{G}_{MA}\} \cup \{\neg a \mid a \in \mathbf{G}_{MA}\}$.

Dado un conjunto de átomos instanciados, un *mundo* es cualquier subconjunto de átomos – aquellos que pertenecen al conjunto son *verdaderos* en el mundo, mientras que aquellos que no pertenecen son *falsos*. Por lo tanto, hay $2^{|\mathbf{G}_{ME}|}$ mundos posibles en el ME y $2^{|\mathbf{G}_{MA}|}$ mundos en el MA. Estos conjuntos se denotan con $\mathcal{W}_{ME}$ y $\mathcal{W}_{MA}$, respectivamente. Para evitar mundos que no modelan posibles situaciones dado un dominio en particular, se incluyen *restricciones de integridad* de la forma $\mathsf{oneOf}(\mathcal{A}')$, donde $\mathcal{A}'$ es un subconjunto de átomos instanciados. Intuitivamente, una restricción establece que cualquier mundo donde aparece más de un átomo del conjunto $\mathcal{A}'$ es inválido. Se utiliza $\mathbf{IC}_{ME}$ y $\mathbf{IC}_{MA}$ para denotar los conjuntos de restricciones de integridad para el ME y MA, respectivamente, y el conjunto de mundos que se adaptan a esas restricciones se denotan con $\mathcal{W}_{ME}(\mathbf{IC}_{ME})$ y $\mathcal{W}_{MA}(\mathbf{IC}_{MA})$, respectivamente.

Por último, las fórmulas lógicas se conforman a partir de la combinación de átomos usando los conectores tradicionales ($\wedge$, $\vee$, y $\neg$). Diremos que un mundo λ *satisface* una fórmula (f), escrito como $\lambda \models f$, si y sólo si:

1. Si f es un átomo, entonces $\lambda \models f$ si y sólo si $f \in \lambda$,

2. si $f = \neg f'$ entonces $\lambda \models f$ si y sólo si $\lambda \not\models f'$,

3. si $f = f' \wedge f''$ entonces $\lambda \models f$ si y sólo si $\lambda \models f'$ y $\lambda \models f''$ y,

4. si $f = f' \vee f''$ entonces $\lambda \models f$ si y sólo si $\lambda \models f'$ o $\lambda \models f''$

Usaremos la notación $form_{ME}, form_{MA}$ para denotar el conjunto de todas las fórmulas (instanciadas) posibles en el ME y MA, respectivamente; por último, se usará $básico_{MA}$ para denotar todas las posibles conjunciones o disyunciones de literales básicos del $\mathbf{L}_{MA}$, las cuáles serán referidas como *fórmulas básicas*.

Ejemplo 1 (Análisis de Intrusión en Redes) *El ejemplo presentado en esta sección toma el dominio del análisis de ciberamenazas; en particular, el análisis y detección de intrusión en redes. Usualmente, esta tarea cuenta con la ayuda de las plataformas de Gestión de Eventos y Seguridad de la Información (SIEM, por sus siglas en Inglés) que proporcionan continuamente alertas relacionadas con el estado de seguridad de la red y que son estudiadas por los analistas en el Centro de Operaciones de Seguridad (SOC, por sus siglas en Inglés). Estos analistas, a su vez, examinan las alertas para determinar si corresponden a eventos maliciosos. El estado final deseado es garantizar que las actividades no autorizadas se puedan manejar de manera eficaz aplicando las contramedidas adecuadas para evitar que lo problemas empeoren y resolver el incidente, eliminando la amenaza para la red [SSEJJD12]. Es importante remarcar que este tipo de análisis requiere un proceso mucho más impreciso que el razonamiento determinista [XLO+10], ya que, por ejemplo, no se conocen las decisiones del atacante, los ataques no siempre tienen éxito, y las observaciones de los analistas son limitadas.*

Antes de presentar los modelos del entorno y analítico de nuestro ejemplo, primero vamos a mostrar los predicados de los conjunto $\mathbf{P}_{ME}$ y $\mathbf{P}_{MA}$ y sus respectivos significados, estos pueden verse en el cuadro 2.1. Como se muestra en dicho cuadro, algunos predicados forman parte del MA y otros del ME. Por ejemplo, los predicados que describen el comportamiento del sistema bajo ciertas circunstancias son parte del modelo analítico. Por otro lado, el modelo del entorno contiene los predicados asociados a eventos inciertos.

$\mathbf{P}_{ME}$

Variable	Predicado	Significado
a	*alertaFirewall*	Alertas de intrusión desde el firewall
b	*alertaSOC*	Alertas de intrusión de los analistas del SOC
c	*procolosVul*	Se están utilizando protocolos desactualizados
d	*accesoRed*	Los atacantes obtuvieron acceso a la red
e	*httpdVul*	El servicio httpd que se ejecutan en el servidor web es vulnerable
f	*archivosCompr*	Los archivos del sistema están comprometidos
g	*privilegiosEjec*	Los atacantes obtuvieron privilegios de ejecución en el servidor web
h	*instalacionMalSpy*	Los atacantes instalan software malicioso (malware) y espía (spyware) en el servidor web
i	*infectanRed*	Los atacantes infectaron la red
j	*puertasTraseras*	Los atacantes abrieron puertas traseras

$\mathbf{P}_{MA}$

Predicado	Significado
errorRegSis	Error detectado en los registros del sistema
vulnerable	El sistema está en un estado vulnerable
comprometido	El sistema está comprometido
bajoAtaque	El sistema está bajo ataque
personalCalificado	El personal está entrenado para manejar situaciones peligrosas que puedan comprometer el sistema
servidorWebVul	El servidor web es vulnerable
httpDesactualizado	El servicio httpd que se ejecuta en el servidor está desactualizado
consultasSosp	Se registraron consultas sospechosas
firewallActualizado	El firewall del sistema está actualizado

Cuadro 2.1: Predicados del ME (arriba) y MA (abajo).

2.4.1. Modelo del Entorno

En este modelo se representa el conocimiento del dominio que está sujeto a eventos probabilísticos. Algunos de los modelos que pueden usarse son Redes Bayesianas, Redes Lógicas de Markov, extensiones de la lógica de primer orden como la lógica probabilística de Nilsson o hasta una función de distribución de probabilidad como la de la Definición 1.

Definición 1 (Modelo Probabilístico) *Dado los conjuntos* $\mathbf{P}_{ME}$*,* V*, y* C*, y los conjuntos correspondientes* $\mathbf{G}_{ME}$ *y* $\mathcal{W}_{ME}$*, un modelo probabilístico* $\mathcal{P}_{ME}$ *es cualquier función* $Pr : \mathcal{W}_{ME} \rightarrow [0, 1]$ *tal que* $\sum_{\lambda \in \mathcal{W}_{ME}} \Pr(\lambda) = 1$*.*

En esta tesis asumimos que contamos con un modelo probabilístico definido sobre $\mathbf{G}_{ME}$, el cual representa una distribución de probabilidad sobre $\mathcal{W}_{ME}$.

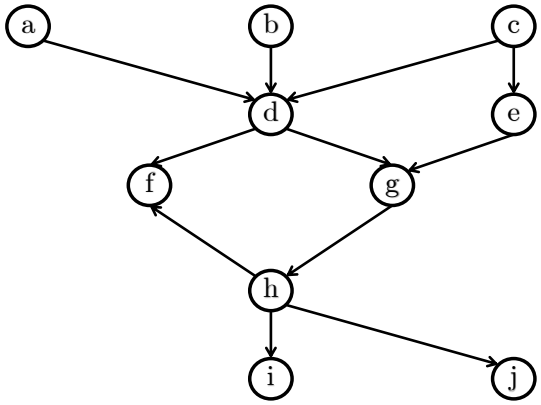
Continuando con el Ejemplo 1, utilizaremos un *grafo de ataque* determinístico para codificar la causalidad lógica asociada a los eventos del sistema (un grafo que ilustra los posibles ataques de múltiples etapas, típicamente presentando las relaciones lógicas de causalidad entre múltiples privilegios y configuraciones [XLO+10]). Esta es una herramienta muy útil para entender los eventos asociados a la seguridad, siempre que se pueda representar *adecuadamente* la incertidumbre inherente al proceso de razonamiento. Una herramienta muy utilizada para manejar esta incertidumbre son las Redes Bayesianas (RB) [MKRC19, FWSJ08, GEYF19]. Utilizaremos el mismo modelo para el ME del ejemplo.

En la Figura 2.4 (arriba) mostramos la estructura del grafo que usamos para nuestro ME. Cada uno de los predicados del ME se representa con un nodo en el grafo (para este ejemplo, son todos booleanos) y los arcos representan la relación de causalidad entre los predicados. Esta lógica puede representarse mediante una disposición adecuada de la estructura del grafo y mediante las tablas de probabilidad condicional (TPC).

La red bayesiana que se muestra en la Figura 2.4 (arriba) describe la relación entre los predicados del ME; con el fin de tener una presentación más clara, omitimos presentar las tablas de probabilidad condicional y nos limitamos a mostrar la distribución de probabilidad completa P_r sobre algunos mundos de $\mathcal{W}_{ME}$, ya que en total existen 1024 (2^{10}). Esta distribución puede verse en la Figura 2.4 (abajo). De esta manera, la probabilidad de que todos los predicados sean verdaderos (un mundo en particular, en este caso λ_1) y, por lo tanto, los atacantes infectaron la red (predicado i) y abrieron puertas traseras (predicado j) es igual a $0,47$. De esta manera, terminamos de presentar el modelo del entorno; a continuación presentamos el modelo analítico.

2.4.2. Modelo Analítico

DeLP3E adopta un marco de argumentación estructurada [SR09] para el MA. Mientras que el ME contiene información probabilística a cerca del estado del dominio, el AM debe



$\mathcal{W}_{ME}$	a	b	c	$\ldots$	j	$P_r(\lambda_i)$
λ_1	T	T	T	...	T	0.47
λ_2	T	T	T	...	F	0.15
λ_3	T	T	T	...	T	0.05
...	...	...	...	...	...	...
λ_{1024}	F	F	F	...	F	0.003

Figura 2.4: Red Bayesiana del ME y Distribución de Probabilidad sobre $\mathcal{W}_{ME}$.

permitir un tipo diferente de información; en particular, debe ser capaz de representar conocimiento contradictorio. Este enfoque permite la creación de *argumentos* que pueden competir entre sí para llegar a una conclusión con respecto a una consulta determinada. Esto se conoce como un *proceso dialéctico*, donde los argumentos se derrotan entre sí en función de un *criterio de comparación*. Como resultado de este proceso, algunos argumentos están *garantizados*, mientras que otros son *derrotados*. El razonamiento basado en la argumentación fue propuesto y estudiado en profundidad como una forma natural de gestionar información inconsistente – su fortaleza radica en el hecho de que se parece mucho a la forma en la que los humanos resuelven disputas (considerando, por ejemplo, cómo se deciden las condenas en los juicios). Otra característica muy importante en los marcos de argumentación estructurada es que, una vez que se llega a una conclusión, también se alcanza una explicación de *cómo se llegó* a dicha conclusión, como así también información del por qué un argumento fue garantizado. A continuación, primero presentamos los conceptos básicos del marco de argumentación subyacente utilizado, y luego, detallamos el modelo analítico (MA).

Programación Lógica Rebatible con Presunciones (PreDeLP)

PreDeLP, introducido en [MGS12], es un formalismo que combina programación lógica con argumentación rebatible. Mencionaremos los conceptos básicos de PreDeLP, para una presentación completa referimos al lector a [GS04, MGS12].

El formalismo contiene varios constructores diferentes: hechos, presunciones, reglas estrictas, y reglas rebatibles. Los hechos son declaraciones que son siempre verdaderas, mientras que las presunciones son declaraciones que pueden o no ser verdaderas. Las reglas estrictas establecen consecuencias lógicas (similar a la implicación en la lógica de primer orden, aunque la semántica no es exactamente la misma ya que la contrapositiva de una regla estricta no es necesariamente válida). Mientras que las reglas estrictas, como los hechos, son siempre verdaderos, las reglas rebatibles especifican consecuencias lógicas que pueden asumirse como verdaderas cuando no existe información contradictoria disponible. Estos componentes se utilizan para la construcción de *argumentos*, y juntos conforman un programa PreDeLP.

Formalmente, usamos la notación $\mathcal{P} = (\Theta, \Omega, \Phi, \Delta)$ para denotar un programa PreDeLP, donde:

- Ω es el conjunto de reglas estrictas de la forma $L_0 \leftarrow L_1, \ldots, L_n$, donde L_0 es un literal instanciado y $\{L_i\}_{i>0}$ es un conjunto de literales instanciados;
- Θ es el conjunto de hechos, escritos como átomos;
- Δ es el conjunto de reglas rebatibles de la forma $L_0 \prec L_1, \ldots, L_n$, donde L_0 es un literal instanciado y $\{L_i\}_{i>0}$ es un conjunto de literales instanciados, y
- Φ es el conjunto de presunciones, las cuáles se escriben como reglas rebatibles sin cuerpo.

Por simplicidad, nos referiremos con $\mathcal{P}_{MA}$ al conjunto que corresponde con la unión de estos componentes.

Es importante recordar que todos los átomos en el MA deben formarse con un predicado del conjunto $\mathbf{P}_{MA}$, y como se permite la *negación fuerte* (*i.e.*, , la negación clásica como en la lógica de primer orden) en la cabeza de las reglas, esto puede usarse para representar conocimiento contradictorio.

$$
\begin{aligned}
\Theta: \quad & \theta_1 = errorRegSis \\
& \theta_2 = httpdDesactualizado \\
& \theta_3 = firewallActualizado \\
& \theta_4 = consultasSosp \\
\Omega: \quad & \omega_1 = vulnerable \leftarrow comprometido \\
& \omega_2 = bajoAtaque \leftarrow comprometido, errorRegSis \\
\Phi: \quad & \phi_1 = comprometido \prec \\
& \phi_2 = personalCalificado \prec \\
\Delta: \quad & \delta_1 = bajoAtaque \prec comprometido \\
& \delta_2 = servidorWebVul \prec httpdDesactualizado \\
& \delta_3 = \sim vulnerable \prec personalCalificado \\
& \delta_4 = servidorWebVul \prec consultasSosp \\
& \delta_5 = \sim servidorWebVul \prec personalCalificado, firewallActualizado
\end{aligned}
$$

Figura 2.5: $\mathcal{P}_{MA}$ del Ejemplo 1.

En la Figura 2.5 presentamos el programa $\mathcal{P}_{MA}$ del ejemplo 1. Este programa codifica conocimiento básico del dominio; por ejemplo, la regla estricta ω_2 establece que si se encuentran errores en los registros del sistema (*errorRegSis*) y el sistema está comprometido (*comprometido*), entonces el sistema está bajo ataque (*bajoAtaque*). La regla rebatible δ_2 establece que si el servicio httpd que se ejecuta en el servidor está desactualizado (*httpdDesactualizado*), entonces se asume que el servidor web es vulnerable (*servidorWebVul*).

Argumentos

Dada una consulta en forma de un átomo instanciado, el objetivo es derivar argumentos a favor y en contra de su validez – las derivaciones siguen el mismo mecanismo de la programación lógica [Llo87], y vamos a denotarlas con el símbolo "$\vdash$". Diremos que una derivación es "estricta" si solo usa hechos o reglas estrictas; de lo contrario, diremos que una derivación es "rebatible". De la misma manera, diremos que un literal es derivado estrictamente (rebatiblemente) si la derivación es estricta (rebatible). Por último, diremos que un argumento es "fáctico" si no se usan presunciones en su derivación.

Dado que las cabezas de las reglas pueden contener una negación fuerte, es posible derivar rebatiblemente literales contradictorios a partir de un programa. Para el tratamiento del conocimiento contradictorio, PreDeLP incorpora un formalismo de argumentación re-

batible que permite identificar las piezas de información que están en conflicto y, a través de un proceso dialéctico que explicaremos más adelante, decidir qué información prevalece como garantizada. Este proceso dialéctico consiste en la creación y evaluación de argumentos, definido formalmente a continuación.

Definición 2 (Argumento) *Un argumento $\langle \mathcal{A}, L\rangle$ para un literal L es un par formado por el literal y un conjunto (posiblemente vacío) del MA ($\mathcal{A} \subseteq \mathcal{P}_{MA}$) que provee una prueba mínima para L y que cumple los siguientes requerimientos:*

1. *L es derivado rebatiblemente por $\mathcal{A}$;*
2. *$\Omega \cup \Theta \cup \mathcal{A}$ no es inconsistente; y*
3. *$\mathcal{A}$ es un conjunto minimal de $\Delta \cup \Phi$ que satisface 1 y 2, denotado como $\langle \mathcal{A}, L\rangle$.*

El literal L se denomina la conclusión soportada por el argumento, y $\mathcal{A}$ es el soporte del argumento. Un argumento $\langle \mathcal{B}, L\rangle$ es un subargumento de $\langle \mathcal{A}, L'\rangle$ si $\mathcal{B} \subseteq \mathcal{A}$. Un argumento $\langle \mathcal{A}, L\rangle$ es presuntivo si $\mathcal{A} \cap \Phi$ es no vacío. También usaremos $\Omega(\mathcal{A}) = \mathcal{A} \cap \Omega$, $\Theta(\mathcal{A}) = \mathcal{A} \cap \Theta$, $\Delta(\mathcal{A}) = \mathcal{A} \cap \Delta$, y $\Phi(\mathcal{A}) = \mathcal{A} \cap \Phi$.

Esta definición difiere de la presentada en [SL92], donde se introduce DeLP, ya que se incluyen las reglas estrictas y los hechos como parte de los argumentos. Esto es así ya que en un modelo DeLP3E las reglas estrictas y hechos usados para construir un argumento en particular *pueden ser verdaderos sólo en algunos mundos del ME*. Por lo tanto, el conjunto de hechos y reglas estrictas no tiene que ser necesariamente consistente en un modelo DeLP3E. En la siguiente subsección presentaremos la manera de asociar porciones del MA con mundos del ME.

Continuando con nuestro ejemplo, algunos argumentos que pueden construirse a partir de $\mathcal{P}_{MA}$ son:

$$\langle \mathcal{A}_1, servidorWebVul\rangle, \text{ with } \mathcal{A}_1 = \{\delta_2, \theta_2\}$$
$$\langle \mathcal{A}_2, servidorWebVul\rangle, \text{ with } \mathcal{A}_2 = \{\delta_4, \theta_4\}$$
$$\langle \mathcal{A}_3, \sim servidorWebVul\rangle, \text{ with } \mathcal{A}_3 = \{\delta_5, \phi_2, \theta_3\}.$$

Dado un argumento $\langle \mathcal{A}_1, L_1 \rangle$, un contraargumento es un argumento que lo contradice. Se dirá que un argumento $\langle \mathcal{A}_2, L_2 \rangle$ *contraargumenta* o *ataca* a $\langle \mathcal{A}_1, L_1 \rangle$ en un literal L', si y sólo si, existe un subargumento $\langle \mathcal{A}, L'' \rangle$ de $\langle \mathcal{A}_1, L_1 \rangle$ tal que el conjunto $\Omega(\mathcal{A}_1) \cup \Omega(\mathcal{A}_2) \cup \Theta(\mathcal{A}_1) \cup \Theta(\mathcal{A}_2) \cup \{L_2, L''\}$ es inconsistente. Un *derrotador propio* de un argumento $\langle \mathcal{A}, L \rangle$ es un contraargumento que – por algún criterio – es considerado mejor que $\langle \mathcal{A}, L \rangle$; si los dos argumentos son incomparables de acuerdo a ese criterio, el contraargumento se denomina *derrotador por bloqueo*. Una característica importante de PreDeLP es que el criterio de comparación de argumentos es modular, y, por lo tanto, es posible aplicar el criterio más apropiado teniendo en cuenta la representación del dominio; el criterio que se utiliza por defecto en la programación lógica rebatible clásica (de la cual deriva PreDeLP) es el de *especificidad generalizada* [SGCS03], aunque se requiere una extensión de este criterio para considerar argumentos que utilizan presunciones [MGS12]. A continuación presentaremos la definición de este criterio, para luego utilizarla en la adaptación que considera las presunciones en el cuerpo de los argumentos.

Definición 3 (Preferencia de Argumentos en DeLP) *Dado un programa PreDeLP* $\mathcal{P}_{MA} = (\Theta, \Omega, \Phi, \Delta)$ *y* $\mathcal{F}$ *el conjunto de todos los literales que tienen una derivación rebatible a partir de* $\mathcal{P}_{MA}$*. Un argumento* $\langle \mathcal{A}_1, L_1 \rangle$ *es* preferido a $\langle \mathcal{A}_2, L_2 \rangle$*, denotado como* $\mathcal{A}_1 \succ_{PS} \mathcal{A}_2$ *si:*

(1) *Para todo* $H \subseteq \mathcal{F}$, $\Omega(\mathcal{A}_1) \cup \Omega(\mathcal{A}_2) \cup H$ *es consistente: si existe una derivación para* L_1 *de* $\Omega(\mathcal{A}_2) \cup \Omega(\mathcal{A}_1) \cup \Delta(\mathcal{A}_1) \cup H$*, y no existe una derivación para* L_1 *de* $\Omega(\mathcal{A}_1) \cup \Omega(\mathcal{A}_2) \cup H$*, entonces hay una derivación para* L_2 *de* $\Omega(\mathcal{A}_1) \cup \Omega(\mathcal{A}_2) \cup \Delta(\mathcal{A}_2) \cup H$*; y*

(2) *existe al menos un conjunto* $H' \subseteq \mathcal{F}$*, donde* $\Omega(\mathcal{A}_1) \cup \Omega(\mathcal{A}_2) \cup H'$ *es consistente, tal que existe una derivación para* L_2 *de* $\Omega(\mathcal{A}_1) \cup \Omega(\mathcal{A}_2) \cup H' \cup \Delta(\mathcal{A}_2)$*, no existe una derivación para* L_2 *de* $\Omega(\mathcal{A}_1) \cup \Omega(\mathcal{A}_2) \cup H'$*, y no existe una derivación para* L_1 *de* $\Omega(\mathcal{A}_1) \cup \Omega(\mathcal{A}_2) \cup H' \cup \Delta(\mathcal{A}_1)$*.*

Intuitivamente, el principio de especificidad establece que, en presencia de dos líneas de argumentación contradictorias sobre una proposición, aquella que utiliza la mayor cantidad de la información disponible es la más convincente. La siguiente extensión de este criterio para considerar argumentos presuntivos fue introducida en [MGS12].

Definición 4 (Preferencia de Argumentos Presuntivos) *Dado un programa Pre-DeLP* $\mathcal{P}_{MA} = (\Theta, \Omega, \Phi, \Delta)$*, un argumento* $\langle \mathcal{A}_1, L_1 \rangle$ *es* preferido a $\langle \mathcal{A}_2, L_2 \rangle$*, denotado como* $\mathcal{A}_1 \succ \mathcal{A}_2$ *si se cumple alguna de las siguientes condiciones:*

(1) $\langle \mathcal{A}_1, L_1 \rangle$ *y* $\langle \mathcal{A}_2, L_2 \rangle$ *son ambos fácticos y* $\langle \mathcal{A}_1, L_1 \rangle \succ_{PS} \langle \mathcal{A}_2, L_2 \rangle$.

(2) $\langle \mathcal{A}_1, L_1 \rangle$ *es un argumento fáctico y* $\langle \mathcal{A}_2, L_2 \rangle$ *es un argumento presuntivo.*

(3) $\langle \mathcal{A}_1, L_1 \rangle$ *y* $\langle \mathcal{A}_2, L_2 \rangle$ *son argumentos presuntivos, y*

   (*a*) $\Phi(\mathcal{A}_1) \subsetneq \Phi(\mathcal{A}_2)$ *o,*

   (*b*) $\Phi(\mathcal{A}_1) = \Phi(\mathcal{A}_2)$ *y* $\langle \mathcal{A}_1, L_1 \rangle \succ_{PS} \langle \mathcal{A}_2, L_2 \rangle$.

Generalmente, si $\mathcal{A}, \mathcal{B}$ son argumentos con reglas X e Y, respectivamente, y $X \subset Y$, entonces $\mathcal{A}$ es más fuerte que $\mathcal{B}$. Esto también se mantiene cuando $\mathcal{A}$ y $\mathcal{B}$ usan presunciones P_1 y P_2, respectivamente, y $P_1 \subset P_2$.

Es importante remarcar que este criterio *no es transitivo* [WS14], y por lo tanto no debe asumirse para definir un orden sobre un conjunto de argumentos. Esto no representa un problema en DeLP3E, ya que el criterio se utiliza para comparar argumentos de a pares y así determinar cuál prevalece.

A partir de estas relaciones de ataque se generan secuencias de argumentos denominadas *líneas de argumentación*, donde cada argumento derrota a su predecesor. Para evitar situaciones indeseables, como por ejemplo circularidad en las líneas, en DeLP una *línea de argumentación* es *aceptable* si satisface ciertas restricciones [GS04]. Un literal L está *garantizado* si existe un argumento no derrotado $\mathcal{A}$ que soporta L.

Un argumento $\langle \mathcal{A}, L \rangle$ puede tener varios derrotadores. Por lo tanto, pueden generarse varias líneas de argumentación a partir de $\langle \mathcal{A}, L \rangle$, produciendo una estructura de árbol. El árbol se construye a partir del conjunto de todas las líneas de argumentación que tienen como raíz el argumento inicial. En un árbol de dialéctica, cada nodo (excepto la raíz) representa un derrotador de su padre, y las hojas corresponden a argumentos sin derrotadores. Cada rama del árbol desde la raíz hasta una hoja corresponde a una línea de argumentación aceptable diferente. Un árbol de dialéctica proporciona una estructura para considerar todas las posibles (máximas) líneas de argumentación aceptables que se pueden generar para decidir si un argumento es derrotado. Este árbol se denomina *dialéctico* porque representa un análisis dialéctico (en el sentido de proveer razones a

favor y en contra de una postura) exhaustivo para el argumento en su raíz. Para un argumento dado $\langle \mathcal{A}, L \rangle$, su árbol de dialéctica se denota como $\mathcal{T}(\langle \mathcal{A}, L \rangle)$.

Dado un literal L y un argumento $\langle \mathcal{A}, L \rangle$, para determinar si un literal L está garantizado o no, todo nodo en el árbol de dialéctica $\mathcal{T}(\langle \mathcal{A}, L \rangle)$ se marca recursivamente como "D"(*derrotado*) o "U"(*no derrotado*), obteniendo un árbol de dialéctica etiquetado $\mathcal{T}^*(\langle \mathcal{A}, L \rangle)$ de la siguiente manera:

1. Todas las hojas en $\mathcal{T}^*(\langle \mathcal{A}, L \rangle)$ son marcadas como "U", y
2. Dado $\langle \mathcal{B}, L_q \rangle$ un nodo interno de $\mathcal{T}^*(\langle \mathcal{A}, L \rangle)$. Entonces $\langle \mathcal{B}, L_q \rangle$ será marcado como "U" si y sólo si todos los hijos de $\langle \mathcal{B}, L_q \rangle$ están marcados como "D". El nodo $\langle \mathcal{B}, L_q \rangle$ será marcado como "D" si y sólo si tiene al menos un hijo marcado como "U".

Dado un argumento $\langle \mathcal{A}, L \rangle$ de $\mathcal{P}_{MA}$, si la raíz de $\mathcal{T}^*(\langle \mathcal{A}, L \rangle)$ está marcada con "U", diremos que $\mathcal{T}^*(\langle \mathcal{A}, L \rangle)$ *garantiza* L y que L está *garantizado* por $\mathcal{P}_{MA}$. Existe un requisito adicional cuando los argumentos en el árbol dialéctico contienen presunciones – la conjunción de todas las presunciones utilizadas en los niveles pares (resp. impares) debe ser consistente. Esto puede dar lugar a múltiples árboles para un literal dado, ya que potencialmente puede haber diferentes argumentos que hagan suposiciones contradictorias.

Es posible extender la idea de un árbol de dialéctica a un *bosque de dialéctica*. Para un literal dado L, un bosque de dialéctica $\mathcal{F}(L)$ consiste de un conjunto de árboles de dialéctica para todos los argumentos de L. Un bosque de dialéctica etiquetado es el conjunto de todos los árboles de dialéctica etiquetados y se denota como $\mathcal{F}^*(L)$. Por lo tanto, para un literal L, diremos que está *garantizado* si existe al menos un argumento para ese literal en el bosque de dialéctica $\mathcal{F}^*(L)$ que está etiquetado como "U", *no garantizado*, si existe al menos un argumento para el literal $\neg L$ en el bosque de dialéctica $\mathcal{F}^*(\neg L)$ que está etiquetado con "U", e *indeciso* en cualquier otro caso. Nos referiremos a los estados garantizado, no garantizado o indeciso de L como los "estados de garantía" de L, y en algunas ocasiones nos referiremos al estado "garantizado" como "Sí" y al estado "no garantizado" como "No".

2.4.3. El formalismo DeLP3E

Este formalismo, originalmente presentado en [SSM+16], es el resultado de combinar el modelo del entorno con el modelo analítico (que denotamos con $\mathcal{P}_{ME}$ y $\mathcal{P}_{MA}$, respecti-

$$
\begin{aligned}
af(\theta_1) &= \quad verdadero \\
af(\theta_2) &= \quad httpdVul \\
af(\theta_3) &= \quad \neg protocolosVul \\
af(\theta_4) &= \quad alertaFirewall \\
af(\omega_1) &= \quad verdadero \\
af(\omega_2) &= \quad accesoRed \vee privilegiosEjec \\
af(\phi_1) &= \quad accesoRed \\
af(\phi_2) &= \quad verdadero \\
af(\delta_1) &= \quad verdadero \\
af(\delta_2) &= \quad httpdVul \\
af(\delta_3) &= \quad verdadero \\
af(\delta_4) &= \quad alertaFirewall \\
af(\delta_5) &= \quad \neg alertaFirewall \wedge \neg alertaSOC
\end{aligned}
$$

Figura 2.6: Función de Anotación del Ejemplo 1.

vamente). Intuitivamente, dado el $\mathcal{P}_{MA}$, cada elemento de $\Omega \cup \Theta \cup \Delta \cup \Phi$ sólo se mantiene en ciertos mundos del conjunto $\mathcal{W}_{ME}$ – es decir, estos elementos están sujetos a eventos probabilísticos. Cada elemento de $\Omega \cup \Theta \cup \Delta \cup \Phi$ está asociado a una fórmula sobre $\mathbf{G}_{ME}$ (usando conjunción, disyunción, y negación) – el conjunto de todas esas fórmulas se denota con $form_{ME}$. La noción de *función de anotación* asocia elementos de $\Omega \cup \Theta \cup \Delta \cup \Phi$ con elementos de $form_{ME}$.

Definición 5 (Función de Anotación) *Una función de anotación es cualquier función de la forma* $fa : \Omega \cup \Theta \cup \Delta \cup \Phi \rightarrow form_{ME}$. *El conjunto de todas las funciones de anotación se denota con* $[fa]$.

En algunas ocasiones usaremos la notación de conjuntos de pares $(f, af(f))$ para denotar las funciones de anotación. En la Figura 2.6 se muestra la función de anotación para el Ejemplo 1. Usando esta función de anotación, podemos descomponer el $\mathcal{P}_{MA}$ en subprogramas y consultar el estado de un literal en cada uno de ellos. Entonces, por ejemplo, en el mundo λ_{1024} de nuestro ejemplo todos los elementos del ME son falsos. A partir de estos valores podemos determinar cuáles de estas anotaciones son verdaderas; aquellos elementos de $\mathcal{P}_{MA}$ cuyas fórmulas son verdaderas generan el programa $\mathcal{P}_{MA}(\lambda_{1024})$, y podemos

asignar la probabilidad de este mundo a los literales garantizados por este subprograma. A continuación, mostramos algunos ejemplos de subprogramas.

$$\lambda_{1024} = \{\} \qquad \lambda_1 = \{a, b, c, ..., h, i, j\}$$

$$P_{AM}(\lambda_{1024}) : \begin{Bmatrix} \theta_1, \theta_3, \omega_1, \\ \phi_2, \delta_1, \delta_3, \\ \delta_5 \end{Bmatrix} \quad P_{AM}(\lambda_1) : \begin{Bmatrix} \theta_1, \theta_2, \theta_4, \\ \omega_1, \omega_2, \phi_1, \\ \phi_2, \delta_1, \delta_2, \\ \delta_3, \delta_4 \end{Bmatrix}$$

$$\lambda_{1000} = \{f, g\} \qquad \lambda_{50} = \{a, b, c, d, g, h, i\}$$

$$P_{AM}(\lambda_{1000}) : \begin{Bmatrix} \theta_1, \theta_3, \omega_1, \\ \omega_2, \phi_2, \delta_1, \\ \delta_3, \delta_5 \end{Bmatrix} \quad P_{AM}(\lambda_{50}) : \begin{Bmatrix} \theta_1, \theta_4, \omega_1, \\ \omega_2, \phi_1, \phi_2, \\ \delta_1, \delta_3, \delta_4 \end{Bmatrix}$$

Definición 6 (Programa DeLP3E) *Dado un modelo del entorno $\mathcal{P}_{ME}$, un modelo analítico $\mathcal{P}_{MA}$, y una función de anotación fa, un programa DeLP3E es de la forma $\mathcal{P} = (\mathcal{P}_{ME}, \mathcal{P}_{MA}, fa)$. El conjunto de todos los posibles programas se denota con $[\mathcal{P}]$*

De ahora en adelante, dado un programa DeLP3E $\mathcal{P} = (\mathcal{P}_{ME}, \mathcal{P}_{MA}, fa)$ y $\lambda \in \mathcal{W}_{ME}$, usaremos la notación $\mathcal{P}_{MA}(\lambda) = \{f \in \mathcal{P}_{MA} \text{ t.q. } \lambda \models fa(f)\}$. Esto genera una vista de subprogramas, tal como mostramos a continuación. Por tanto, la forma más directa de considerar las consecuencias de un programa DeLP3E es considerar lo que sucede en cada mundo de $\mathcal{W}_{ME}$; es decir, la relación de derrota entre argumentos depende del estado actual del mundo (ME).

Definición 7 (Existencia de un Argumento en un Mundo) *Dado un program DeLP3E $\mathcal{P} = (\mathcal{P}_{ME}, \mathcal{P}_{MA}, fa)$, diremos que un argumento $\langle \mathcal{A}, L \rangle$ existe en un mundo $\lambda \in \mathcal{W}_{ME}$ si $\forall c \in \mathcal{A}, \lambda \models af(c)$.*

La noción de existencia se extiende para líneas de argumentación, árboles de dialéctica, y bosques de dialéctica de la manera esperada (por ejemplo, una línea de argumentación existe en λ si y sólo si todos los argumentos que conforman dicha línea existen en λ).

La idea de un árbol de dialéctica también se extiende en referencia a los mundos; por lo tanto, para un mundo $\lambda \in \mathcal{W}_{ME}$, el árbol de dialéctica (resp. árbol etiquetado)

inducido por λ es denotado con $\mathcal{T}_\lambda\langle\mathcal{A}, L\rangle$ (resp., $\mathcal{T}^*_\lambda\langle\mathcal{A}, L\rangle$). Para esto se requiere que todos los argumentos y derrotadores que aparecen en estos árboles existan en λ. De la misma manera, se extiende esta noción para los bosques de dialéctica (denotados como $\mathcal{F}_\lambda(L)$ y $\mathcal{F}^*_\lambda(L)$, respectivamente). A partir de estos conceptos, es posible presentar la noción de *escenario de garantía*.

Definición 8 (Escenario de Garantía) *Sea $\mathcal{P} = (\mathcal{P}_{ME}, \mathcal{P}_{MA}, fa)$ un programa DeLP3E y un literal L formado por un átomo instanciado de $\mathbf{G}_{MA}$, un mundo $\lambda \in \mathcal{W}_{ME}$ se dice escenario de garantía para L (denotado como $\lambda \vdash_{war} L$) si existe un bosque de dialéctica $\mathcal{F}^*_\lambda(L)$ en el cual L está garantizado y $\mathcal{F}^*_\lambda(L)$ existe en λ.*

La idea de un escenario de garantía se usa para definir formalmente la derivación en DeLP3E. El conjunto de mundos del ME donde el literal L del MA *debe* ser verdadero es exactamente el conjunto de escenarios de garantía – estos son los mundos "necesarios":

$$nec(L) = \{\lambda \in \mathcal{W}_{ME} \mid (\lambda \vdash_{\mathsf{war}} L)\}$$

Por otro lado, el conjunto de mundos del ME donde el literal L del MA *puede* ser verdadero son los siguientes – estos son los mundos "posibles":

$$poss(L) = \{\lambda \in \mathcal{W}_{ME} \mid \lambda \nvdash_{\mathsf{war}} \neg L\}.$$

Usaremos la notación $for(\lambda) = \bigwedge_{a\in\lambda} a \wedge \bigwedge_{a\notin\lambda} \neg a$ para denotar la fórmula que tiene a λ como su único modelo. Esta noción puede extenderse para conjuntos de mundos: $for(W) = \bigvee_{\lambda\in W} for(\lambda)$. A partir de estos conceptos, la derivación en DeLP3E se define de la siguiente manera:

Definición 9 (Derivación en DeLP3E) *Sea $\mathcal{P} = (\mathcal{P}_{ME}, \mathcal{P}_{MA}, fa)$ un programa DeLP3E, un literal L del MA y un intervalo de probabilidad $p \in [\ell, u]$, se dice que a partir de $\mathcal{P}$ se deriva L con probabilidad $p \in [\ell, u]$ si la distribución de probabilidad* Pr *producida por $\mathcal{P}_{ME}$ es tal que $\ell \leq \sum_{\lambda\in nec(L)} \Pr(\lambda)$ y $\sum_{\lambda\in poss(L)\lambda} \Pr(\lambda) \leq u$.*

El procedimiento por fuerza bruta para computar este intervalo de probabilidad consiste en recorrer cada uno de los mundos del ME y, por cada uno de ellos, consultar el estado del literal, y en base a su estado de garantía, actualizar el valor ℓ o u según corresponda. Este procedimiento se describe a continuación:

1. Inicializar $\ell = 0$ y $u = 1$.
2. Para cada mundo λ_i, computar su correspondiente subprograma del AM $\mathcal{P}_{MA}(\lambda_i)$.
   a) Si el literal consultado está garantizado en ese subprograma, entonces actualizar $\ell \leftarrow \ell + \Pr(\lambda_i)$.
   b) De lo contrario, si la *negación* del literal está *garantizada*, actualizar $u \leftarrow u - \Pr(\lambda_i)$.
3. Retornar $[\ell, u]$.

Es importante remarcar que, dado que el número de mundos en $\mathcal{W}_{ME}$ es *exponencial* en el número de variables aleatorias en el ME, computar este procedimiento será intratable salvo para instancias pequeñas. Sin embargo, es posible obtener una aproximación *sana* seleccionando un subconjunto de mundos de $\mathcal{W}_{ME}$ y ejecutando el mismo procedimiento. En el Capítulo 3 se desarrolla un análisis más completo sobre este punto, como así también se presentan diferentes técnicas de muestreo para aproximar ese intervalo de probabilidad asociado a un literal en una base de conocimiento DeLP3E.

2.5. Sumario

En este capítulo se presentan los conceptos preliminares para esta tesis. En particular, se describen los formalismos de argumentación y la manera de extenderlos para trabajar con el razonamiento probabilístico, luego, se introducen brevemente los modelos probabilísticos gráficos, en particular las redes bayesianas, y por último, se presenta y desarrolla el formalismo DeLP3E acompañado con un caso de uso en el dominio de la ciberseguridad. En el siguiente capítulo se estudia el problema de computar las respuestas exactas en el formalismo DeLP3E y se proponen dos enfoques para aproximar dichas respuestas.

Capítulo 3

Hacia un familia de algoritmos de aproximación a consultas en DeLP3E

En este capítulo se presentará el estudio del problema de computar el intervalo de probabilidad exacto para un literal en particular, un conjunto de métricas para cuantificar la complejidad de cada componente del formalismo DeLP3E (ME, MA y Función de Anotación) y, por último, una familia de algoritmos y enfoques para aproximar el valor exacto del intervalo de probabilidad basados en las métricas definidas anteriormente. En la Sección 3.1 se comenzará presentando la intratabilidad inherente al problema de computar el intervalo de probabilidad para responder a una consulta; luego, en la Sección 3.2, se definirán un conjunto de métricas que nos ayudará a clasificar los componentes del modelo DeLP3E según diferentes criterios. En la Sección 3.3, se presentará un conjunto de algoritmos para aproximar el valor del intervalo de probabilidad teniendo en cuenta la complejidad de los componentes de la base de conocimiento según el valor de las métricas presentadas.

3.1. Intratabilidad

Dar respuesta a una consulta en DeLP3E consiste en computar el intervalo de probabilidad asociado al literal consultado. Esto es, por ejemplo, para un literal L computar el intervalo $p \in [\ell, u]$ donde la distribución de probabilidad Pr producida por el $\mathcal{P}_{ME}$ es tal que $\ell \leq \sum_{\lambda \in nec(L)} \Pr(\lambda)$ y $\sum_{\lambda \in poss(L)\lambda} \Pr(\lambda) \leq u$. Como se mencionó al final del capítulo

anterior, el procedimiento para computar este intervalo consiste en recorrer cada uno de los mundos del ME (Modelo del Entorno) y, por cada uno de ellos, consultar el estado del literal L, y, en base a su estado de garantía, actualizar el valor ℓ o u según corresponda.

En esta sección, listaremos brevemente los resultados computacionales presentados en [SSF16] y [AGP$^+$21a] en relación al procedimiento para dar respuesta a una consulta en DeLP3E. Comenzamos señalando que la complejidad del problema de decidir el estado de garantía de un literal en un programa PreDeLP clásico es coNP-hard (consecuencia directa de lo demostrado en [AGP$^+$21a] para DeLP). Como consecuencia de este resultado, tenemos que decidir el estado de garantía de un literal con su intervalo de probabilidad asociado tiene esta cota inferior de complejidad, ya que es necesario consultar por el literal en cada subprograma PreDeLP de los mundos del $\mathcal{P}_{ME}$ para calcular su intervalo exacto.

Bajo estos resultados, en [SSF16] se analiza la complejidad del problema de decidir la el estado de garantía de un literal en DeLP3E bajo la suposición simplificadora de que la consulta en PreDeLP puede hacerse en tiempo polinomial; lamentablemente, aun bajo esta suposición, el problema permanece intratable (#P-hard). En ese mismo trabajo, también se analizó la complejidad suponiendo que las probabilidades asociadas a los mundos del $\mathcal{P}_{ME}$ podían computarse en tiempo polinomial, lo cuál sigue arrojando como resultado #P-hard como complejidad para el problema en PreDeLP.

Estos resultados muestran la complejidad inherente, aún bajo ciertas suposiciones, para dar respuesta a una consulta en un modelo DeLP3E. En la siguiente sección, abordamos esta intratabilidad a través de técnicas de aproximación que se basan en información de cada uno de los modelos de la base de conocimiento.

3.2. Técnicas de aproximación y métricas de la base de conocimiento

Como se menciona en la sección anterior, computar el intervalo de probabilidad exacto en un tiempo razonable no es posible para instancias grandes. Para estos casos, una alternativa es aplicar alguna técnica que nos permite aproximar, con un valor aceptable, el intervalo exacto. Estas técnicas consisten en muestrear el espacio de soluciones, que en nuestro caso son los mundos en $\mathcal{P}_{ME}$, y, a partir de estas muestras, actualizar el intervalo

de probabilidad aproximado. Aquí podemos definir y aplicar diferentes heurísticas y procedimientos para elegir qué mundos muestrear, por ejemplo, aquellos que tengan un valor de probabilidad mayor (para analizar la mayor cantidad de masa probabilística en menor cantidad de muestras) o aquellos que generen subprogramas del $\mathcal{P}_{MA}$ que sean más fáciles de consultar (para intentar realizar una cobertura de bajo costo).

Antes de comenzar a estudiar cómo aplicar una técnica de muestreo, recordemos brevemente cómo está formada la base de conocimiento DeLP3E y cuáles son los escenarios que nos interesan analizar. En una base de conocimiento DeLP3E tenemos tres componentes, y uno de ellos relaciona a los otros dos. Por un lado, tenemos el modelo analítico ($\mathcal{P}_{MA}$), formado por reglas que codifican el conocimiento del dominio en un lenguaje lógico formal. Por otro lado, tenemos al modelo del entorno ($\mathcal{P}_{ME}$), el cual representa los eventos probabilísticos de nuestro dominio y su correspondiente función de distribución de probabilidad. El tercer componente, encargado de vincular los otros dos, relaciona cada regla del $\mathcal{P}_{MA}$ con fórmulas lógicas formadas con los eventos del $\mathcal{P}_{ME}$ y se denomina función de anotación (fa). De esta forma, y desde un punto de vista del $\mathcal{P}_{ME}$, cada mundo representa un escenario posible de razonamiento en base a las reglas del $\mathcal{P}_{MA}$ cuyas fórmulas pueden satisfacerse con los estados de los eventos del $\mathcal{P}_{ME}$ que caracterizan ese mundo en particular. Esto puede verse en la Figura 3.1, donde cada círculo representa un mundo posible λ_i del $\mathcal{P}_{ME}$ y $\mathcal{P}_{MA}(\lambda_i)$ representa el subconjunto de reglas (subprograma) del $\mathcal{P}_{MA}$ cuyas fórmulas de anotación son verdaderas (por ejemplo, en el mundo λ_5 todas las fórmulas asociadas a las reglas del $\mathcal{P}_{MA}$ son verdaderas salvo la asociada a la regla ϕ_1).

Supongamos entonces que consultamos por un literal L; para computar su intervalo de probabilidad exacto tenemos que obtener los escenarios de garantía tanto para el literal L (para el límite inferior ℓ) como para su complemento $\sim L$ (para el límite superior u). Para esto es necesario construir el correspondiente subprograma asociado a cada uno de los mundos del $\mathcal{P}_{ME}$ y consultar en cada uno de ellos el estado del literal. Una vez que tenemos todos los subprogramas creados y todas las consultas ejecutadas, tendríamos un escenario como el de la Figura 3.1. Por último, lo que nos interesa sumar son las probabilidades asociadas a los mundos λ_i donde se garantiza el literal (pintados de verde) y aquellos donde se garantiza el complemento del literal (pintados de rojo), es decir, los escenarios de garantía del literal y su complemento respectivamente. De esta forma, el intervalo exacto es $[\ell = \sum P_r(\lambda_i \text{ verde}), u = 1 - \sum P_r(\lambda_i \text{ rojo})]$.

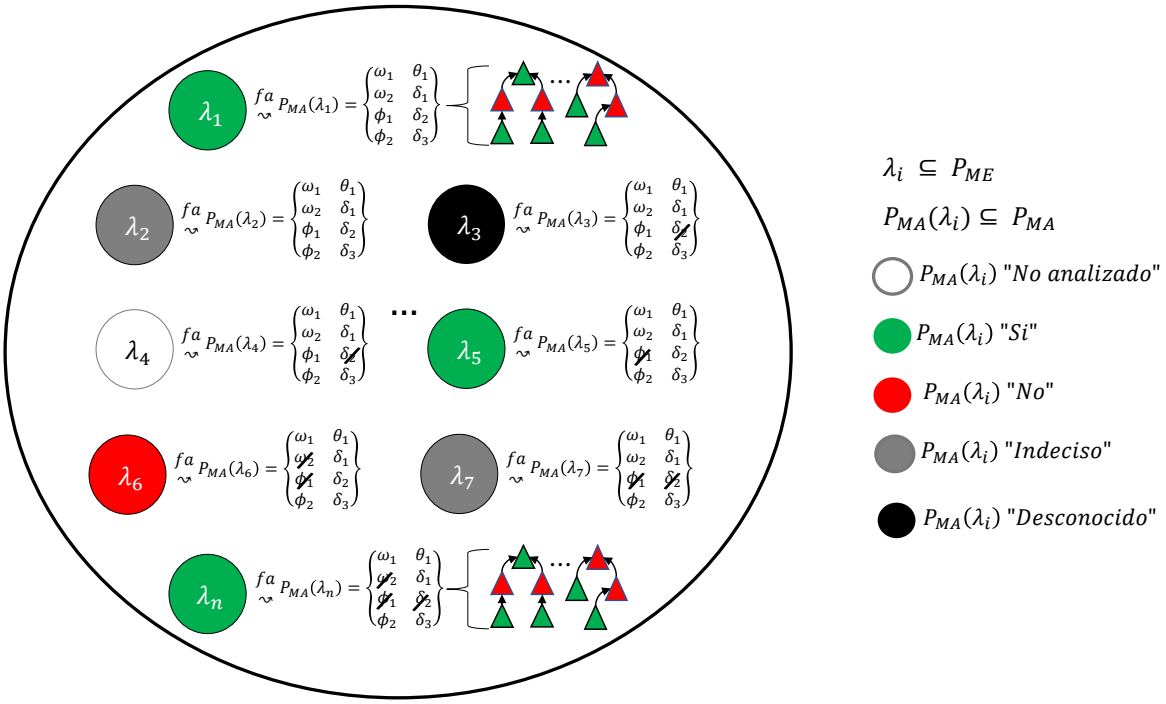


Figura 3.1: Mundos de $\mathcal{P}_{ME}$.

Teniendo en cuenta esto, aplicar una técnica de muestreo consiste en elegir mundos λ_i al azar, o siguiendo algún criterio, y construir su subprograma correspondiente para luego consultar por el estado del literal y así sumar la probabilidad del límite que corresponda. La forma de elegir estos mundos y la cantidad a elegir dependerá de las características de los modelos y de la información disponible que tengamos de cada uno de ellos. Esto se verá en detalle más adelante en esta misma sección.

Esta técnica de muestreo se basa en los mundos del $\mathcal{P}_{ME}$; sin embargo, y teniendo en cuenta la manera de relacionar los modelos de la base de conocimiento, también es posible muestrear a partir de los subprogramas del $\mathcal{P}_{MA}$. Muestrear por subprogramas consiste en dividir el universo de todos los posibles subprogramas en regiones formadas por los mundos que los generan. Es decir, lo que se muestrea son subprogramas del MA y se consulta por el literal en ellos; una vez que se tiene el estado del literal, se consulta por la probabilidad que suman todos los mundos que generan ese subprograma. Cada subprograma puede ser generado por múltiples mundos, ya que las fórmulas de la función de anotación pueden tener múltiples modelos que las satisfacen. De esta manera, podemos considerar dos tipos de muestreo: *basado en mundos* y *basado en subprogramas*. Estos dos enfoques pueden verse en la Figura 3.2. También es importante remarcar que en ambos

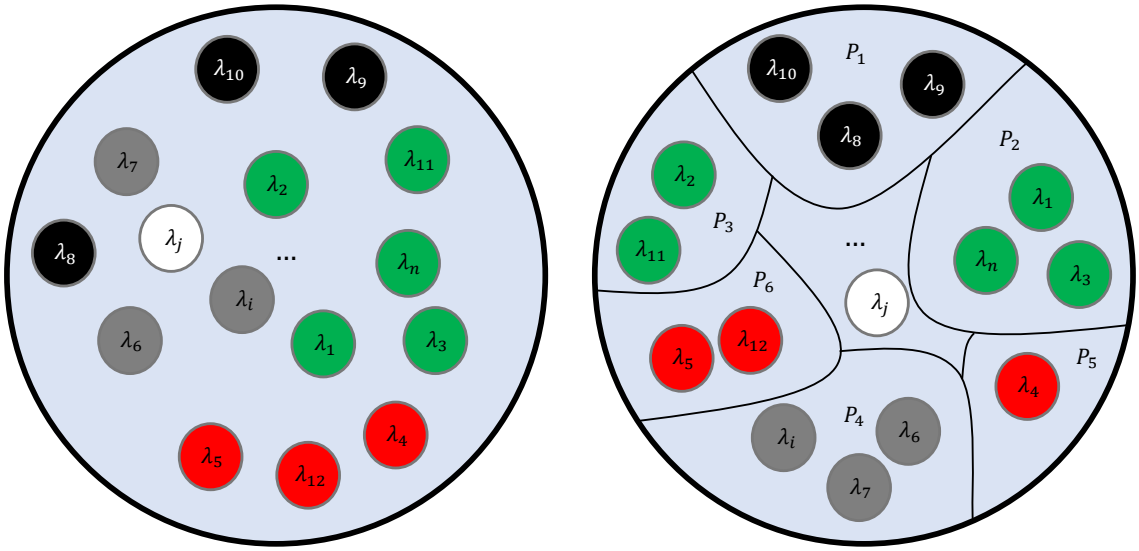


Figura 3.2: Enfoque de muestreo basado en mundos λ_i (izquierda) y basado en subprogramas P_i (derecha).

tipos de muestreo el estado del literal puede ser "indeciso" o "desconocido" y, dada la naturaleza incompleta del proceso de muestreo, parte de la masa de probabilidad quedará inexplorada. A continuación se explicarán en detalle ambas técnicas de muestreo.

Muestreo por mundos

El Algoritmo 3.1 muestra el proceso para ejecutar un muestreo por mundos. El tamaño de la muestra o el número de mundos a muestrear puede estar determinado por un porcentaje de $\mathcal{W}_{ME}$ o un tiempo definido (en este caso se muestrea hasta agotar dicho tiempo). El algoritmo que se detalla a continuación muestrea un número de mundos W_s (línea 8). Luego, por cada mundo, primero verificamos que no sea un mundo repetido (línea 11), ya que el muestreo es con sustitución y, en ese caso, consultamos por su valor de probabilidad (línea 13) y generamos su correspondiente subprograma del $\mathcal{P}_{MA}$ (línea 15). Aquí podemos aplicar una optimización al algoritmo: dado que es posible generar el mismo subprograma a partir de mundos diferentes, una vez generado el subprograma podemos verificar si no fue previamente analizado (línea 17), es decir, si ya conocemos el estado del literal en ese programa. En caso de ser un programa no conocido, se consulta

por el estado del literal y se almacena dicho programa y estado computado (líneas 19 y 20), luego, según la respuesta obtenida, se actualiza el intervalo con la probabilidad asociada al mundo λ_i (líneas 21 a 26). De esta manera, si el subprograma ya fue analizado, nos evitamos realizar la misma consulta y simplemente actualizamos el intervalo con la probabilidad asociada al mundo muestreado (líneas 28 y 29). En caso de repetir un mundo muestreado, simplemente se cuenta como repetido (línea 32). Finalmente, se retorna el intervalo de probabilidad aproximado (línea 33). El procedimiento de la línea 9 es el encargado de seleccionar un mundo del conjunto $\mathcal{W}_{ME}$, aquí es posible muestrear de manera aleatoria, a partir de la función de distribución de probabilidad del $\mathcal{P}_{ME}$, o aplicar diferentes técnicas de selección.

Muestreo por subprogramas

En el caso del muestreo por subprogramas, el esquema general del algoritmo no difiere mucho del muestreo por mundos. Aquí, el tamaño de la muestra puede definirse de la misma manera que en el muestreo por mundos (porcentaje o tiempo); sin embargo, es importante remarcar que los porcentajes no son comparables entre los dos tipos de muestreo, ya que la cantidad de mundos en cada caso puede ser muy diferente. Esto último es así ya que, como se mencionó previamente, un programa puede ser generado por múltiples mundos, por lo que muestrear un 20 % de los programas posibles en el $\mathcal{P}_{MA}$ puede requerir consultar la probabilidad de un conjuntos de mundos que no representa el 20 % de los mundos posibles en el $\mathcal{P}_{ME}$. En relación al número de programas posibles, este difiere del valor determinado por $2^{|\mathbf{G}_{MA}|}$, ya que al muestrear en base al número de reglas existentes en el $\mathcal{P}_{MA}$ es posible obtener programas inconsistentes: supongamos el programa que se muestra en la Figura 3.3, donde varios de sus elementos están anotados con *verdadero* (es decir, esas reglas están presentes en todos los mundos posibles). Es evidente que el número posible de programas es 2, uno donde la regla ϕ_s es válida, y otro donde no lo es. Sin embargo, al considerar sólo el número de reglas, existen 2^{10} programas posibles. Es por esto que el proceso para muestrear subprogramas parte de un conjunto que se computa previamente, y que se considera en el tiempo requerido del algoritmo. Dicho conjunto está formado por los programas que son posibles construir a partir del valor de sus anotaciones. La construcción de ese conjunto consiste evaluar todas las anotaciones con los posibles valores de las variables usadas en dichas anotaciones (es decir, el conjunto de mundos relacionados a esas anotaciones). Dicho procedimiento genera el conjunto de

Algoritmo 3.1: Muestreo por mundos

Datos:
L: Literal a consultar
W_s: Número de mundos a muestrear, con $W_s < 2^{|\mathbf{G}_{ME}|}$
$\mathcal{W}_{ME}$: Mundos posibles en $\mathcal{P}_{ME}$
$\mathcal{P} = (\mathcal{P}_{ME}, \mathcal{P}_{MA}, fa)$: Programa DeLP3E
Resultado: Intervalo aproximado $[\ell', u']$

```
1  inicio
2      // inicializar los valores del intervalo
3      ℓ' ⟵ 0
4      u' ⟵ 1
5      // estructuras para almacenar mundos y subprogramas muestreados
6      known_worlds ← {}
7      known_programs ← {}
8      mientras W_s > 0 hacer
9          λ_i ← seleccionar un mundo de W_ME
10         // verificar si no fue analizado previamente
11         si λ_i ∉ known_worlds entonces
12             // consultar por la probabilidad del mundo muestreado
13             P_r(λ_i) ← get_prob(λ_i, P_ME)
14             // construir el correspondiente subprograma
15             P_MA(λ_i) ← gen_program(λ_i, P_MA, fa)
16             // verificar si no fue analizado previamente
17             si P_MA(λ_i) ∉ known_programs entonces
18                 // consultar por el estado de L en P_MA(λ_i)
19                 query_status ← query(L, P_MA(λ_i))
20                 known_programs ∪ [P_MA(λ_i), query_status]
21                 // actualizar intervalo de probabilidad aproximado
22                 si query_status = SI entonces
23                     // P_MA(λ_i) ⊢_war L
24                     ℓ' ← ℓ' + P_r(λ_i)
25                 sino, si query_status = NO entonces
26                     // P_MA(λ_i) ⊢_war ∼L
27                     u' ← u' − P_r(λ_i)
28             sino
29                 query_status ← get_status(known_programs, P_MA(λ_i))
30                 // igual a pasos 21 a 26
31         sino
32             // Contar como mundo repetido
33         W_s ← W_s − 1.
34     devolver [ℓ', u']
```

$$
\begin{aligned}
af(\theta_1) &= \quad verdadero \\
af(\theta_2) &= \quad verdadero \\
af(\theta_3) &= \quad verdadero \\
af(\omega_1) &= \quad verdadero \\
af(\omega_2) &= \quad verdadero \\
af(\phi_1) &= \quad verdadero \\
af(\phi_2) &= \quad var_{12} \\
af(\delta_1) &= \quad verdadero \\
af(\delta_2) &= \quad verdadero \\
af(\delta_3) &= \quad verdadero
\end{aligned}
$$

Figura 3.3: Ejemplo de programa anotado

programas posibles, y es a partir de ese conjunto que se muestrean los subprogramas. El Algoritmo 3.2 detalla el procedimiento general: primero se construye el conjunto de subprogramas posibles (línea 6) del $\mathcal{P}_{MA}$; se muestrea un número P_s de subprogramas (línea 10); luego, por cada uno de estos subprogramas, se verifica que no sea un subprograma que ya fue muestreado previamente (línea 13), ya que el muestro también es con sustitución y, en ese caso, se procede a construir la expresión que genera ese subprogramas a partir de las variables del $\mathcal{P}_{ME}$, es decir, el subconjunto de mundos que generan el subprograma muestreado (línea 15). Dicha expresión consiste en la conjunción de las anotaciones de los elementos del subprograma y la conjunción de la negación de las anotaciones de los elementos que no forman parte del subprograma muestreado:

$$for(\gamma_i) = \bigwedge_{f \in \gamma_i} af(f) \wedge \bigwedge_{f \notin \gamma_i} \neg af(f),$$

donde $f \in \Omega \cup \Theta \cup \Delta \cup \Phi$ que forma parte del subprograma muestreado γ_i y *af(f)* corresponde a la fórmula anotada de *f*. Una vez que tenemos construida la expresión, se procede a consultar por el estado del literal en el subprograma (línea 17) y a almacenar dicho subprograma (línea 18); en este caso no es necesario almacenar el estado del literal, ya que no se tiene en cuenta un subprograma repetido en el proceso de muestreo. Luego, se consulta por los modelos que satisfacen la expresión del subprograma (línea 20), dichos modelos son las asignaciones de valores para las variables que satisfacen la expresión, es decir, los mundos del $\mathcal{P}_{ME}$ que generan el subprograma. Esto último es necesario para consultar por la probabilidad que acumulan esos mundos, y así, poder actualizar el inter-

valo de probabilidad aproximado según corresponda el resultado de la consulta. Para esto, se toma cada modelo que satisface la expresión (línea 23) y se analiza si no fue contabilizado previamente (ya que puede haber mundos repetidos debido a la misma asignación de valores para un subconjunto de variables del $\mathcal{P}_{ME}$, y el no controlarlos implica sumar la misma probabilidad varias veces), en ese caso, se lo almacena (línea 28) y se consulta por su valor de probabilidad (línea 30) para sumarlo al total de la probabilidad que acumula el subprograma (línea 31). Finalmente, y según el estado de la consulta, se actualiza el límite correspondiente del intervalo con la suma de las probabilidades de todos los mundos que generan el subprograma (líneas 33 a 38). Por último, se retorna el intervalo de probabilidad aproximado (línea 42).

Una vez que tenemos el intervalo aproximado, por cualquier tipo de muestreo, es necesario computar qué tan cercano es éste al intervalo exacto. A continuación presentamos una métrica para realizar esa tarea.

Métrica de calidad

Para determinar qué tan buena es la aproximación conseguida a través del muestreo, definimos la siguiente métrica de calidad.

Definición 10 (Métrica de Calidad) *Dado un intervalo de probabilidad $i_1 = [a, b]$, la siguiente métrica mide la calidad de una aproximación sana $i_2 = [c, d]$ (es decir, se asume que $[a, b] \subseteq [c, d]$):*

$$Q_{i_1}(i_2) = \frac{1 - (d - c)}{1 - (b - a)}$$

Intuitivamente, esta métrica calcula la masa de probabilidad que se *descarta* por un intervalo en relación con otro; observar que el valor resultante es siempre un número real entre 0 y 1. En general, usaremos el resultado del algoritmo para computar el intervalo exacto en el numerador y una aproximación en el denominador.

Una vez que tenemos detalladas las técnicas de muestreo y una métrica para determinar la calidad de las aproximaciones, continuamos con el análisis de los modelos de la base de conocimiento para poder medir su complejidad y así poder optar por una técnica de muestreo en particular.

Algoritmo 3.2: Muestreo por subprogramas

Datos:
L: Literal a consultar
P_s: Número de programas a muestrear, con $P_s < 2^{|\mathbf{G}_{MA}|}$
$\mathcal{W}_{MA}$: Programas en $\mathcal{P}_{MA}$
$\mathcal{P} = (\mathcal{P}_{ME}, \mathcal{P}_{MA}, fa)$: Programa DeLP3E
Resultado: Intervalo aproximado $[\ell', u']$

inicio
  // inicializar los valores del intervalo
  $\ell' \longleftarrow 0$
  $u' \longleftarrow 1$
  // computar los subprogramas posibles
  $poss\_progs \leftarrow$ buscar todos los subprogramas posibles en $\mathcal{W}_{MA}$
  // estructuras para almacenar mundos y subprogramas muestreados
  $known\_programs \leftarrow \{\}$
  $known\_worlds \leftarrow \{\}$
  mientras $P_s > 0$ **hacer**
    $\gamma_i \leftarrow$ seleccionar un subprograma de $poss\_progs$
    // verificar si no fue analizado previamente
    si $\gamma_i \notin known\_programs$ **entonces**
      // construir la expresión que genera el subprograma γ_i
      $\gamma_i\_expression \leftarrow \bigwedge_{f \in \gamma_i} af(f) \wedge \bigwedge_{f \notin \gamma_i} \neg af(f)$
      // consultar por el estado de L en γ_i
      $query\_status \leftarrow query(L, \gamma_i)$
      $known\_programs \cup \gamma_i$
      // consultar por todos los modelos de la expresión que generan el subprograma
      $\gamma_i\_models \leftarrow get\_models(\gamma_i\_expression, \mathcal{P}_{ME})$
      // para almacenar la probabilidad de todos los mundos que generan el subprograma
      $\gamma_i\_prob \leftarrow 0{,}0$
      para cada $model$ en γ_i_models **hacer**
        // un modelo es una asignación de valores para las variables (un mundo de $\mathcal{P}_{ME}$)
        $\lambda_i \leftarrow model$
        // verificar si no es un mundo cuya probabilidad ya fue sumada
        si $\lambda_i \notin known\_worlds$ **entonces**
          $known\_worlds \cup \lambda_i$
          // consultar por la probabilidad del mundo
          $P_r(\lambda_i) \leftarrow get\_prob(\lambda_i, \mathcal{P}_{ME})$
          $\gamma_i\_prob \leftarrow \gamma_i\_prob + P_r(\lambda_i)$
      // actualizar intervalo de probabilidad aproximado
      si $query\_status =$ SI **entonces**
        // $\gamma_i \vdash_{\mathsf{war}} L$
        $\ell' \leftarrow \ell' + \gamma_i\_prob$
      sino, si $query\_status =$ NO **entonces**
        // $\gamma_i \vdash_{\mathsf{war}} \sim L$
        $u' \leftarrow u' - \gamma_i\_prob$
    sino
      // Contar como subprograma repetido
    $P_s \leftarrow P_s - 1$
  devolver $[\ell', u']$

Modelo	Atributo	Métrica	Observable	Aproximación
MA	Tamaño	$\#Reglas$	Sí	-
		$\#Hechos\_Pres$	Sí	-
	Complejidad	$MDDL$	-	Sí
		h	-	Sí
		t	-	Sí
		τ	-	Sí
ME	Tamaño	$\#Variables$	Sí	-
		$\#PGM\_Arcs$	Sí	-
	Complejidad	PGM_TW	-	Sí
		Ent	-	En algunos escenarios
FA	Tamaño	$\%AF\_An$	Sí	-
	Complejidad	AF_Com	Sí	-

Cuadro 3.1: Métricas de la base de conocimiento

Métricas de la base de conocimiento

Con el objetivo de definir una familia de algoritmos de muestreo para aproximar la respuesta a una consulta en el modelo DeLP3E de la manera más eficiente y efectiva posible, debemos considerar qué métricas e información disponible tenemos para nuestra base de conocimiento. El Cuadro 3.1 muestra un conjunto de tales métricas, organizadas con respecto al componente al que se aplican, y el atributo que miden. Para cada una de ellas también se analiza si es posible calcularla de manera tratable (columna *observable*) o aproximarse.

Para cada modelo de la base de conocimiento podemos centrarnos en dos atributos principales: *tamaño* y *complejidad*. Para el modelo MA, $\#Reglas$ y $\#Hechos\_Pres$ se refiere al número total de reglas y hechos más presunciones diferentes en el programa PreDeLP, respectivamente. Con respecto al componente de complejidad, tenemos las siguiente métricas:

- *MDDL*: Longitud promedio de la derivación rebatible, es decir, el número de reglas rebatibles presentes en cualquier argumento construido a partir del programa.

- h: Longitud máxima promedio de las líneas de argumentación (secuencia de argumentos donde cada elemento de la secuencia derrota a su predecesor).
- t: Número promedio de líneas de argumentación diferentes que surgen a partir de un argumento.
- τ: Número promedio de árboles de dialéctica.

Para el ME, en general éste se especifica con modelos probabilísticos gráficos (PGMs [KF09], por sus siglas en inglés); por lo tanto, definimos algunas métricas específicas a este tipo de modelo. Para otro tipo de modelos, deberán definirse métricas específicas que se ajusten a cada caso. Para el ME, $\#\mathit{Variables}$ se refiere al número de variables aleatorias en el modelo (por ejemplo, nodos en una red bayesiana), mientras que $\#\mathit{PGM\_Arc}$ se refiere al número de arcos en un modelo probabilístico gráfico. Con respecto al componente de complejidad, tenemos la siguientes métricas:

- $\mathit{PGM\_TW}$: *treewidth* del PGM. Intuitivamente, este valor mide qué tan cerca está el modelo probabilístico gráfico de tener una estructura de árbol; si bien se pueden calcular aproximaciones para esta métrica [KF09], el valor exacto no se puede calcular de manera tratable para modelos muy grandes.
- *Ent*: *entropía* del PGM. Este valor es inherente a la función de distribución de probabilidad conjunta subyacente al ME, y es posible aproximarla en algunos escenarios particulares [BDKR05].

Finalmente, para la función de anotación, una métrica para el atributo tamaño es $\%\mathit{FA\_An}$, que se refiere al porcentaje de reglas del MA que están anotadas. Para el atributo de complejidad, la métrica $\mathit{FA\_Com}$ se refiere a la complejidad de cada anotación en sí (es decir, qué operadores se usan, si se permite la combinación de operadores, etc).

Con los valores que arrojen estas métricas para cada uno de los modelos, vamos a tener una noción del tamaño y la complejidad de cada uno de ellos. Por ejemplo, para el caso del MA, podemos clasificar los programas en *simple*, *medio* y *complejo* según el tiempo que nos lleve consultar por el estado de sus literales en base al valor de sus métricas. Un ejemplo de esto se muestra en la Figura 3.4, donde cada caso se corresponde con un nivel de tamaño y complejidad. Para el caso del ME, y suponiendo que usamos redes

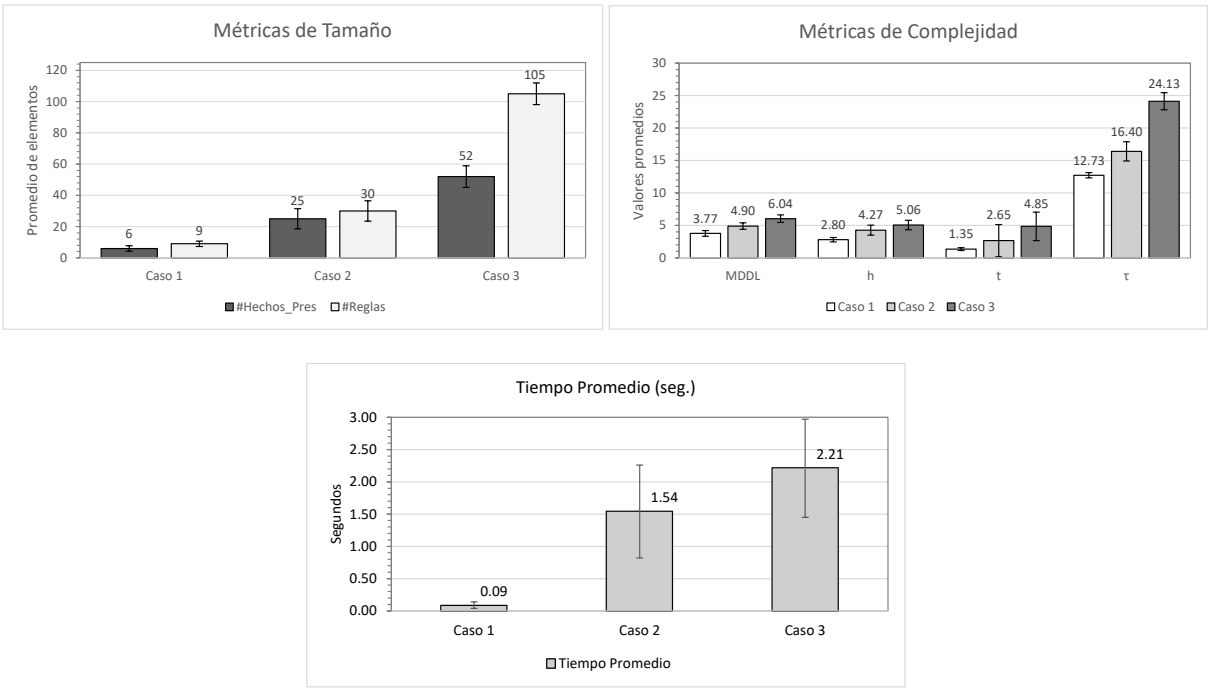
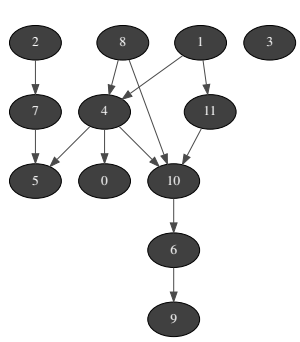


Figura 3.4: Ejemplo de Métricas para el MA (valores promedio sobre 1000 programas PreDeLP para cada clasificación).

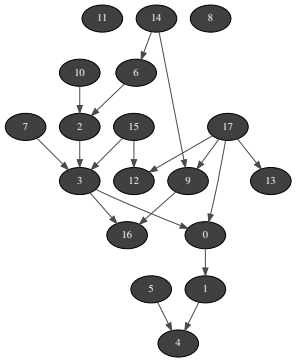
bayesianas, podemos tener modelos como los que se muestran en la Figura 3.5. Aquí podemos tener estructuras con diferente número de nodos, arcos, y valores de entropía, por ejemplo. Naturalmente, cuanto más compleja es la estructura, mayor es el costo de consultar el valor de probabilidad para un muestro en particular (un mundo del ME).

En cuanto a la función de anotación, la Figura 3.6 muestra tres ejemplos de anotaciones y sus respectivos valores de métricas. A la izquierda de la figura tenemos un modelo donde sólo se anotan los hechos y las presunciones del MA, con anotaciones simples usando sólo una variable del ME. En el centro, tenemos un caso donde se anota el 50 % del MA con anotaciones usando sólo un operador y dos variables del ME. Por último, a la derecha de la figura, tenemos un caso donde está anotado el 100 % del MA y se utilizan anotaciones complejas (múltiples operadores y tres variables del ME).

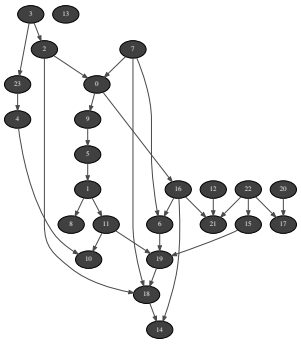
De esta manera, con la información que nos arrojen estas métricas, podemos definir un conjunto de algoritmos junto con una guía que nos indique cuál aplicar para aproximar la respuesta a una consulta de la manera más eficiente y en el menor tiempo. Esto lo detallamos en la siguiente sección.



#Variables	12
#PGM_Arcs	12
PGM_TW	2
Ent	0.9



#Variables	18
#PGM_Arcs	18
PGM_TW	2
Ent	0.6



#Variables	24
#PGM_Arcs	30
PGM_TW	2
Ent	0.5

Figura 3.5: Ejemplo de modelos del entorno usando redes bayesianas.

3.3. Familia de algoritmos de aproximación

En base a la información obtenida con las métricas definidas, podemos comenzar a explorar las posibles alternativas para derivar algoritmos de aproximación basados en técnicas de muestreo. Por ejemplo, considere un escenario en el que tenemos 10 elementos en el MA ($\#Reglas + \#Hechos\_Pres$) y 50 variables en el ME ($\#Variables$), y AF_Comp es bajo (solo se utiliza una variable del $\mathcal{P}_{ME}$, es decir, son fórmulas sin operadores). Aquí, podemos aproximar el intervalo de probabilidad muestreando *subprogramas*, ya que el número de subprogramas será menor en comparación con el número de mundos ($2^{10} < 2^{50}$), y las fórmulas usadas en las anotaciones son fáciles de evaluar (conjunción de variables o su negación). Por otro lado, si las anotaciones son complejas (se usa más de una variable, múltiples operadores y negaciones, por ejemplo $af(\omega_1) = a \vee b \rightarrow \neg c$) entonces el muestreo por mundos puede ser más simple (incluso si hay más elementos en el ME que reglas en el MA). Esto se debe a que el proceso para obtener el subprograma es simple (sustituir los valores de las variables aleatorias según el mundo muestreado y evaluar la fórmula de anotación, luego consultar el literal en el subprograma generado) en comparación con computar la probabilidad de los mundos que satisfacen esas fórmulas (expresión compleja en el ME). Aquí es importante notar que los ejemplos mencionados

$af(\theta_1) =$	var_1	$af(\theta_1) =$	$var_1 \vee var_{12}$	$af(\theta_1) =$	var_1
$af(\theta_2) =$	var_2	$af(\theta_2) =$	$verdadero$	$af(\theta_2) =$	var_2
$af(\theta_3) =$	var_3	$af(\theta_3) =$	$\sim var_3 \wedge var_4$	$af(\theta_3) =$	var_3
$af(\omega_1) =$	$verdadero$	$af(\omega_1) =$	$verdadero$	$af(\omega_1) =$	$var_1 \vee var_3$
$af(\omega_2) =$	$verdadero$	$af(\omega_2) =$	$verdadero$	$af(\omega_2) =$	$var_{12} \wedge var_1 \vee var_6$
$af(\phi_1) =$	var_2	$af(\phi_1) =$	var_2	$af(\phi_1) =$	var_2
$af(\phi_2) =$	var_{12}	$af(\phi_2) =$	var_{12}	$af(\phi_2) =$	var_{12}
$af(\delta_1) =$	$verdadero$	$af(\delta_1) =$	$verdadero$	$af(\delta_1) =$	$\sim var_4 \vee var_1$
$af(\delta_2) =$	$verdadero$	$af(\delta_2) =$	$verdadero$	$af(\delta_2) =$	$var_7 \wedge var_8$
$af(\delta_3) =$	$verdadero$	$af(\delta_3) =$	$var_5 \vee \sim var_2$	$af(\delta_3) =$	$\sim var_{10}$

Figura 3.6: Ejemplo de diferentes funciones de anotación.

anteriormente exponen una especie de asimetría entre MA y ME; a partir de un mundo solo es posible generar un único programa, pero un subprograma puede ser generado por un *conjunto de mundos* (ya que las fórmulas en las anotaciones pueden tener más de un modelo). Esto también puede verse en la Figura 3.2. Considere un caso en el que ME es una red bayesiana con valores altos de *treewidth* y *entropía*; lo que significa que la red está lejos de una estructura de árbol y la distribución de probabilidad subyacente tiene un alto grado de incertidumbre (entropía máxima). Esto conduce a un proceso de muestreo por mundos más lento, complejo y menos guiado; por lo tanto, no se recomienda un algoritmo para muestrear mundos aleatoriamente. En este caso es mejor decidir en un paso previo cuantos y qué mundos muestrear (por ejemplo, muestrear un 10 % de los mundos posibles a partir de la función de distribución de probabilidad de la red bayesiana), con el fin de optimizar los recursos.

Estos ejemplos muestran la variedad de enfoques posibles para aproximar la respuesta a consultas en DeLP3E. Uno de los objetivos de esta tesis es desarrollar un conjunto de criterios de decisión que permitan seleccionar el mejor algoritmo para el trabajo, contemplando las compensaciones inherentes entre el tiempo de ejecución (incluido el costo de calcular cualquier métrica aproximada que sea necesaria) y la precisión del resultado obtenido. Con este objetivo en mente, a continuación, presentamos un *árbol de decisión* basado en los posibles valores de las métricas, y que sirve de guía para determinar qué algoritmo de muestreo utilizar para aproximar el valor del intervalo de probabilidad para

una consulta dada.

Árbol de decisión para algoritmos de aproximación

A partir de la caracterización del tamaño y complejidad de los modelos en base a los valores que arrojen sus métricas, podemos definir un conjunto de criterios para determinar diferentes niveles de complejidad en un modelo DeLP3E. Este conjunto de criterios pueden representarse como ramas de un árbol, donde los nodos hoja determinarán el algoritmo de muestreo a elegir. Este árbol tiene como finalidad servir como una guía para la selección del algoritmo de aproximación en base al valor de las métricas y el costo de computarlas y/o aproximarlas.

El árbol que presentamos en esta sección se construyó en gran parte en base al resultado de un conjunto de experimentos que serán detallados en el próximo capítulo. Brevemente, se diseñaron generadores para cada uno de los modelos con el objetivo de construir escenarios de complejidad diferentes y ejecutar en cada uno de ellos los dos algoritmos de aproximación presentados; luego, se compararon los resultados según el valor de la métrica de calidad y el tiempo necesario para computar la aproximación. La decisión de presentar el árbol de decisión en esta sección consiste en mostrar el uso de los valores de las métricas previamente definidas, para detallar los valores de las mismas que son interesantes analizar con diferentes algoritmos de aproximación.

A continuación, presentaremos los puntos principales para comprender el diseño y la lectura del árbol; luego, en la Figura 3.7 se presenta el árbol final junto con sus correspondientes leyendas.

- Como es usual, los *rombos* representan una decisión en base al valor de la/s métricas que se encuentran en su interior. Aquí podemos agrupar algunas métricas para determinar, por ejemplo, el "tamaño" de un modelo en base a la suma de los valores de las métricas para el atributo tamaño. De esta forma, podemos utilizar:
  - $|MA| = \#Reglas + \#Hechos\_Pres$ o
  - $|ME| = \#Variables + \#PGM\_Arcs$
- Los *rectángulos* indican el valor de la/s métrica consultada.
- Los *círculos* o *cuadrados* indican el algoritmo de muestreo sugerido.

    - los círculos con *línea punteada* indican el algoritmo sugerido en base a toda la información obtenida hasta *ese punto*, y considerando el *peor* escenario posible para las métricas más complejas (cauteloso o pesimista). Esta sugerencia es para los casos donde no sea posible computar otras métricas más complejas.
    - los cuadrados con *línea punteada* indican el algoritmo sugerido en base a toda la información obtenida hasta *ese punto*, y considerando el *mejor* escenario posible para las métricas más complejas (crédulo u optimista). Esta sugerencia es para los casos donde no sea posible computar otras métricas más complejas.
    - los círculos con *línea sólida* indican el mejor algoritmo en base a *toda* la información que podemos obtener de los modelos hasta ese punto.

- El color *naranja* indica aquellas métricas que sólo pueden aproximarse y requieren un mayor costo computacional.

- La estructura del árbol pretende ofrecer una recomendación lo *antes posible*, es decir, recomendar un algoritmo de muestreo sin tener que consultar por los valores de *todas* las métricas. Esto permite recomendar un algoritmo para aquellos casos donde no sea posible, no se requiera, o no sea prioritario computar los valores para algunas métricas en particular.

- El árbol se diseñó para retrasar la decisión sobre aquellas métricas que sólo puedan aproximarse y que requieran un mayor costo computacional.

- Algunos subárboles se repiten en diferentes ramas, pero como éstos tienen estructuras simples, se optó por no usar conectores y repetir todo el subárbol. Esto se hizo para mejorar la lectura de la estructura y poder identificar cada rama del árbol de manera unívoca, y de esta forma, *justificar la recomendación del algoritmo* de manera organizada.

Como se mencionó anteriormente, los rombos de color naranja representan aquellas métricas que sólo pueden aproximarse, y, por lo tanto, requieren un costo computacional adicional. Un caso particular de estas métricas es la que corresponde a la complejidad del MA, ya que a diferencia de las métricas PGM_TW y Ent, para estas últimas existen métodos definidos para aproximar su valor [Klo94, BDKR05]. En el árbol se deja expresada esta métrica como una función $\Upsilon(MA)$ que depende de los valores de las cuatro métricas de complejidad presentadas anteriormente para este modelo ($MDDL$, h, t, y τ). Esto último

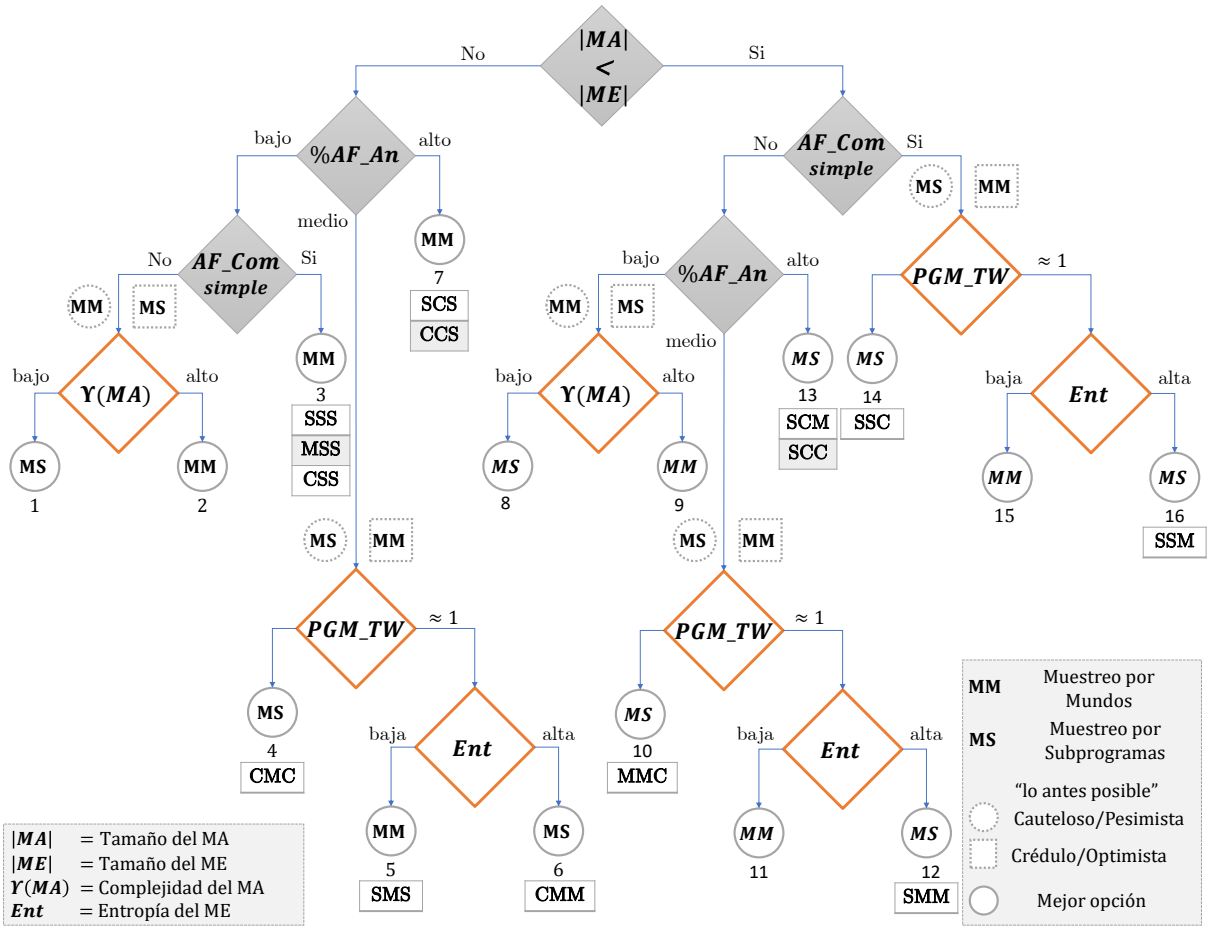


Figura 3.7: Árbol de decisión para determinar el mejor algoritmo de muestreo.

es útil para tener la opción de definir la complejidad del MA en base a diferentes criterios en base a sus métricas; por ejemplo, se podría definir que un "modelo analítico es complejo si $MDDL > 4$ y $t = 5$", indicando que el programa del MA genera argumentos que tienen en promedio más de 4 reglas en su cuerpo y que se generan 5 líneas de argumentación distintas por cada argumento.

Para leer e interpretar este árbol, se comienza analizando los valores de las métricas que son "observables"; a partir de esto, se tiene como recomendación un algoritmo de aproximación basado sólo en esos valores ("lo antes posible") o la opción de seguir bajando en el árbol obteniendo más información de los modelos y así tener una recomendación más guiada ("mejor opción"). A continuación presentamos algunos de los ejemplos más significativos de diferentes ramas o escenarios del árbol junto con su algoritmo recomendado y su correspondiente justificación; en cada caso, la numeración del escenario corresponde

a la utilizada para las hojas en la Figura 3.7.

- Escenario 1: Tenemos un caso donde $|ME| < |MA|$ (menor cantidad de mundos que subprogramas), tenemos un bajo porcentaje de anotaciones, esas anotaciones no son simples y la complejidad del MA es baja.
  - Recomendación: *Muestreo por Subprogramas*.
  - Justificación: Al tener pocas anotaciones y que utilizan varias variables y operadores, esta situación nos indica que cada subprograma agrupa un conjunto de mundos del ME; por lo tanto, con cada subprograma muestreado podemos aproximar el intervalo exacto a una razón mayor que si muestreamos mundo por mundo. Esto sumado a que cada subprograma tiene una complejidad baja, lo cual es una ventaja al momento de muestrear y consultar por el estado del literal.
- Escenario 4: Tenemos que $|ME| < |MA|$ (menor cantidad de mundos que subprogramas), un porcentaje de anotación medio y un $PGM\_TW \neq 1$.
  - Recomendación: *Muestreo por Subprogramas*.
  - Justificación: A pesar de tener menos mundos que subprogramas, la cantidad de anotaciones y la complejidad de la estructura del ME hacen que el proceso de muestreo por mundos sea más lento, ya que la estructura es compleja y el número de anotaciones a verificar no es bajo.
- Escenario 5: Tenemos que $|ME| < |MA|$ (menor cantidad de mundos que subprogramas), un porcentaje de anotación medio, $PGM\_TW \approx 1$ y entropía baja.
  - Recomendación: *Muestreo por Mundos*.
  - Justificación: En este caso, al tener una estructura cercana a un árbol, entropía baja y menor número de mundos que subprogramas, es más conveniente muestrear por mundos, independientemente de la complejidad de las anotaciones o la complejidad del MA. Esto es así ya que muestrear una estructura con estas características es más sencillo y, a la vez, obtenemos más información con un menor número de mundos consultados.

- Escenario 8: Tenemos que $|MA| < |ME|$ (menor cantidad de subprogramas que mundos), la complejidad de las anotaciones no es simple, el porcentaje de anotaciones es bajo y la complejidad del MA es baja.
  - Recomendación: *Muestreo por Subprograma.*
  - Justificación: Similar al escenario 1.
- Escenario 9: Tenemos que $|MA| < |ME|$ (menor cantidad de subprogramas que mundos), la complejidad de las anotaciones no es simple, el porcentaje de las anotaciones es bajo y la complejidad del MA es alta.
  - Recomendación: *Muestreo por Mundos.*
  - Justificación: En este caso, al tener un porcentaje bajo de anotaciones y una complejidad alta en el MA, esta complejidad se ve poco afectada por las anotaciones (ya que la mayoría son "verdaderas" en todos los mundos) y, por lo tanto, es conveniente retrasar la consulta al subprograma generado por el mundo muestreado. Esto es conveniente ya que nos permite controlar primero si se muestrea un mundo repetido o si ya se generó ese mismo subprograma antes (ya sabemos el estado del literal y no tenemos que consultar de nuevo).
- Escenario 14: Tenemos que $|MA| < |ME|$ (menor cantidad de subprogramas que mundos), anotaciones simples y una estructura compleja en el ME.
  - Recomendación: *Muestreo por Subprogramas.*
  - Justificación: En este caso el muestreo por subprogramas es la mejor opción, ya que hay menos candidatos para muestrear, las expresiones para consultar por la probabilidad de los mundos que generan los subprogramas muestreados se forma a partir de anotaciones simples, y porque muestrear mundos es más complejo por la estructura del ME.
- Escenario 16: Tenemos que $|MA| < |ME|$ (menor cantidad de subprogramas que mundos), anotaciones simples, $PMG\_TW \approx 1$ (estructura similar a un árbol) y entropía alta.
  - Recomendación: *Muestreo por Subprogramas.*
  - Justificación: Para este caso tenemos una estructura en el ME que es más sencilla de muestrear, pero al tener una entropía alta, muestrear por mundos

no tendrá un gran impacto en la aproximación en comparación con muestrear un subprograma que agrupe un conjunto de mundos y que además, existen menos candidatos para el muestreo ($\mathcal{W}_{MA} < \mathcal{W}_{ME}$).

Este conjunto de escenarios representan algunas situaciones que consideramos relevantes del total de posibles combinaciones a partir de los valores de las métricas. Algunos escenarios triviales no están especificados en el árbol; por ejemplo, si tenemos un escenario donde $|ME| < |MA|$, anotaciones simples, porcentaje de anotaciones bajo y entropía baja, entonces la mejor opción es el muestreo por mundos. Como se mencionó anteriormente, gran parte del proceso de construcción de este árbol fue informado por una batería de experimentos sobre diferentes configuraciones de complejidad para cada modelo de la base de conocimiento y aplicando los dos algoritmos de muestreo para cada configuración. Los detalles sobre la configuración y ejecución de este experimento se presentarán en el Capítulo 5, como así también sus resultados y conclusiones.

Con el objetivo de poder analizar los algoritmos de aproximación presentados sobre diferentes escenarios de complejidad, diseñamos y ejecutamos un conjunto de generadores que nos permiten crear modelos con valores de métricas regulables a través de ciertos parámetros. En el siguiente capítulo presentamos en detalle cada uno de estos generadores.

3.4. Sumario

En este capítulo se presentó el estudio sobre el problema de aproximar el intervalo de probabilidad exacto en el modelo DeLP3E, obteniendo como resultado final dos enfoques de aproximación, el *enfoque basado en mundos* y el *enfoque basado en subprogramas*. También se presentaron dos algoritmos de aproximación (los mismos implementan los enfoques mencionados anteriormente), y un métrica para determinar la calidad de la aproximación generada por esos algoritmos. Se definió un conjunto de métricas con el objetivo de poder cuantificar la complejidad y tamaño de cada componente de un modelo DeLP3E y, por último, se presentó un árbol de decisión que tiene como finalidad servir de guía para la selección del algoritmo de aproximación en base al valor de las métricas mencionadas y al costo de computarlas. En el siguiente capítulo se presentará el diseño y ejecución de tres generadores que nos permitirán crear modelos DeLP3E de diferentes complejidades, esto con la finalidad de poder crear un conjunto de pruebas para analizar los algoritmos de aproximación diseñados.

Capítulo 4

Generadores DeLP3E

En este capítulo presentamos un conjunto de generadores de bases de conocimiento DeLP3E que nos permite crear diferentes escenarios de complejidad de manera automática para luego ejecutar los algoritmos de aproximación propuestos. El objetivo principal de este capítulo es presentar el análisis, diseño, y la ejecución de tres algoritmos para crear, a partir de un conjunto de parámetros, cada uno de los tres componentes de una base de conocimiento DeLP3E. Una de las principales características de estos generadores es que nos permiten crear modelos cuyos valores de métricas pueden ser ajustables en base al valor de los parámetros de entrada que guían el proceso de generación. Este capítulo se organiza de la siguiente manera: en la Sección 4.1 se presenta una breve descripción de los motivos para crear estos generadores; en las Secciones 4.2, 4.3 y 4.4, se presentan el *Generador de Programas PreDeLP*, el *Generadores de Redes Bayesianas*, y el *Generador de Funciones de Anotación* respectivamente. En la sección correspondiente a cada generador, se detallan las decisiones de diseño, los parámetros, algoritmos, y ejemplos de modelos generados (también se muestra una comparativa de dichos modelos en base al valor de sus métricas). Finalmente, en la Sección 4.5, se presentan los conjuntos de modelos generados que serán usados en los experimentos del próximo capítulo.

4.1. Motivación

Luego de la definición de los algoritmos de aproximación en el capítulo anterior, se estudió la posibilidad de ponerlos a prueba bajo diferentes condiciones de tamaño y com-

plejidad para analizar su comportamiento y resultados. Dichas condiciones son determinadas por los valores de las métricas de cada uno de los modelos que conforman la base de conocimiento. Para poder llevar a cabo esto, era necesario contar con un conjunto de modelos de diferentes características y que podamos clasificarlos según su tamaño y complejidad.

A partir de esto, y ante la falta de un repositorio público que nos ofrezca estos datos, se decidió construir los generadores correspondientes para cada uno de los modelos y así poder crear nuestros propios conjuntos de datos para experimentar con ellos. Estos generadores tienen como único objetivo la creación automática de los componentes que forman parte de un modelo DeLP3E, teniendo en cuenta algunas restricciones y limitaciones que nos permitan manipularlos para la configuración del experimento a realizar. En particular, el objetivo es poder crear programas PreDeLP, redes bayesianas (para ser usadas como modelo probabilístico en el modelo del entorno) y funciones de anotación que relacionen los elementos de los otros dos modelos, y así, tener los componentes para crear modelos DeLP3E de diferentes características. Cada uno de estos modelos debe ser generado de tal manera que podamos hacer variar sus métricas de tamaño y complejidad, para luego poder crear modelos DeLP3E que se adapten a cada una de las líneas del árbol de decisión presentado en el Capítulo 3 (Figura 3.7). Para los propósitos de análisis de esta tesis, los modelos creados se clasifican según sus métricas en un esquema de configuración como X_{MA}, X_{ME}, X_{FA}, donde cada X_{Modelo} puede ser S (simple), M (medio) o C (complejo) para determinar el tamaño y complejidad del $Modelo$ en particular. Así, cada generador tendrá definido sus parámetros a través de los cuales podemos hacer variar las métricas de los modelos generados y, por lo tanto, poder clasificarlos como S, M o C. El resultado final de los modelos DeLP3E creados serán clasificados según este esquema de configuración; así, por ejemplo, un modelo DeLP3E *SMS* significa que su modelo analítico es simple, su modelo del entorno es de complejidad media y su función de anotación es de complejidad simple. Esto se detallará en la última sección de este capítulo, y será usado en los experimentos del próximo capítulo.

Además de nuestro objetivo en particular para los generadores presentados, también pretendemos que sean usados para otros experimentos y como una herramienta para la comunidad que desee trabajar con ellos, en particular con el generador de programas PreDeLP. Todos los generadores, como así también todos los conjuntos de modelos creados, están publicados en un repositorio en línea.

Generadores para modelos DeLP3E

En las siguientes secciones se presentarán cada uno de los tres generadores: *Generador de Programas PreDeLP* (GPP), *Generador de Redes Bayesianas* (GRB) y *Generador de Funciones de Anotación* (GFA). En particular, el GPP es el más complejo, ya que consiste en la creación de argumentos, derrotadores, y árboles de dialéctica que deben cumplir con ciertos parámetros de entrada. A diferencia del GPP, el GRB junto con el GFA son muchos más simples, dado que existen algunas herramientas para la creación de redes bayesianas y una función de anotación simplemente vincula elementos de los otros dos modelos ya creados a través de fórmulas lógicas de primer orden.

Los parámetros de cada generador sirven para definir un valor superior de alguno de los componentes del modelo a generar, es decir, los generadores hacen "lo mejor posible" para hacer cumplir los valores de los parámetros, pero no siempre es posible alcanzar esos valores. Esto permite manipular los valores de las métricas de cada modelo, sin tener una manipulación exacta de esos valores, ya que se busca generar los modelos con un cierto componente de aleatoriedad. Esto será explicado en detalle para cada generador en las siguiente secciones.

4.2. Generador de Programas PreDeLP (GPP)

Recordemos que un programa PreDeLP, denotado como $\mathcal{P} = (\Theta, \Omega, \Phi, \Delta)$, está formado por los siguientes conjuntos:

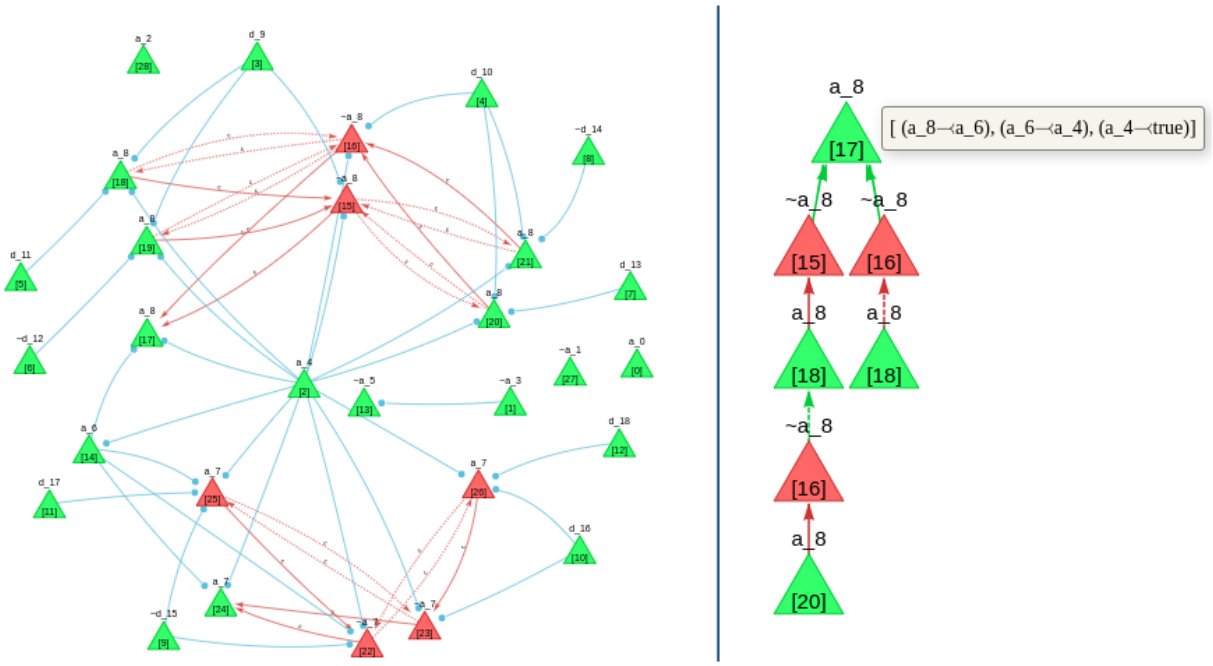
- Ω es el conjunto de reglas estrictas de la forma $L_0 \leftarrow L_1, \ldots, L_n$, donde L_0 es un literal instanciado y $\{L_i\}_{i>0}$ es un conjunto de literales instanciados;
- Θ es el conjunto de hechos, escritos como átomos o átomos negados;
- Δ es el conjunto de reglas rebatibles de la forma $L_0 -\!\prec L_1, \ldots, L_n$, donde L_0 es un literal instanciado y $\{L_i\}_{i>0}$ es un conjunto de literales instanciados, y
- Φ es el conjunto de presunciones, las cuáles se escriben como reglas rebatibles sin cuerpo.

La unión de todos estos conjuntos conforman un modelo $\mathcal{P}_{MA}$. Por lo tanto, el objetivo de nuestro generador es crear cada uno de estos conjuntos de manera que sus elementos

generen estructuras argumentales durante el proceso dialéctico [GS04], es decir, que los elementos creados estén vinculados según las relaciones de argumento, derrotador, argumento preferido, línea de argumentación y árboles de dialéctica. Un ejemplo de este tipo de programas se muestra en la Figura 4.1. Los grafos que se muestran en esta figura son generados con un módulo de P-DAQAP (plataforma que se presentará en detalle en el Capítulo 6). En ellos se muestran todas las estructuras que se generan a partir de un programa PreDeLP: los *argumentos* se representan con un triángulo con la conclusión en la parte superior y un identificador en su cuerpo; las relaciones de *derrota* son flechas de color rojo y los *subargumentos* se representan con arcos de color celeste; además, cada argumento se pinta de verde si su estado es "garantizado" y de rojo si es un argumento "derrotado" (basado en sus árboles de dialéctica). Este programa fue generado con el GPP (las reglas son el resultado de mapear las estructuras internas del generador a la sintaxis de PreDeLP), por lo que es posible apreciar que podemos construir programas que generan estructuras y relaciones variadas y, por lo tanto, con diferentes niveles de complejidad y tamaño. Al final de esta sección se presentará un repositorio de programas PreDeLP que está disponible para ser importado y usado en otros experimentos. Este repositorio es una colección de programas PreDeLP con diferentes valores de métricas.

Con respecto a la representación interna de cada componente del modelo, se optó por la siguiente sintaxis:

- *Átomo*: Cada átomo se representa con la letra "`a`", seguido de un identificador único `_Id` representado por un número natural.
- *Literal*: Un literal es un átomo o un átomo negado, es decir, `a_[Id]` | ∼`a_[Id]`.
- Θ: El conjunto de hechos del programa está formado por literales de la forma `a_[Id].` | ∼`a_[Id]`.
- Φ: Una presunción es un literal seguido del símbolo $-\!\prec$, es decir, por ejemplo, `a_[Id]` $-\!\prec$ | ∼`a_[Id]` $-\!\prec$.
- *Reglas*: Cada regla *estricta*(Ω) o *rebatible* (Δ) se representa como una lista de dos componentes: una *cabeza* (un literal), que denota el consecuente de la regla, y un *cuerpo* (lista de literales) que representa el antecedente de la regla. La distinción entre reglas estrictas y rebatibles se hará según la estructura donde se almacene cada regla (se generan dos estructuras, una para cada tipo de regla).



a_0 —≺	∼d_15 —≺	a_7 —≺ a_6.	a_7 —≺ ∼d_15, a_6, d_17.
∼a_3 —≺	d_16 —≺	∼a_8 —≺ d_9, a_4.	∼a_7 —≺ d_16, a_4.
a_4 —≺	d_17 —≺	∼a_8 —≺ d_10, a_4.	d_13 —≺
d_9 —≺	d_18 —≺	a_8 —≺ d_11, d_9, a_4.	∼d_14 —≺
d_10 —≺	∼a_1.	a_8 —≺ ∼d_12, d_9, a_4.	a_6 —≺ a_4.
d_11 —≺	a_2.	a_8 —≺ d_10, d_13, a_4.	a_8 —≺ a_6.
∼d_12 —≺	∼a_5 —≺ ∼a_3.	a_8 —≺ d_10, ∼d_14, a_4.	a_7 ← d_18, d_16, a_4.

Figura 4.1: Ejemplo de un programa PreDeLP generado con GPP.

Es importante notar que el *cuerpo* de una regla está formado por *literales* que a su vez son *cabezas* de otras reglas del programa, y así hasta alcanzar los hechos o presunciones. De esta forma, nos aseguramos que cada regla tenga su derivación en el programa. Con respecto a la distinción entre reglas, la misma consiste en almacenar en listas diferentes según el tipo de regla, por ejemplo, si tomamos la regla `a_8` —≺ `d_11, d_9, a_4.` del programa de la Figura 4.1, la misma se representa internamente como [`a_8`, [`d_11, d_9, a_4`]] en una lista *drule*, y cada elemento [`d_11, d_9, a_4`] de su cuerpo tiene su derivación en el programa. En resumen, la representación interna de cada componente se muestra en el Cuadro 4.1.

Componente	Representación
Átomo	a_[Id]
Literal	a_[Id] | ∼a_[Id]
Hecho	a_[Id]. | ∼a_[Id].
Presunción	a_[Id] –≺ | ∼a_[Id] –≺
Regla	[cabeza, cuerpo]

Cuadro 4.1: Componentes generados para el $\mathcal{P}_{MA}$.

4.2.1. Diseño

El diseño del GPP sigue un enfoque "de abajo hacia arriba" (bottom-up, en Inglés), es decir, se comienza creando los componentes básicos sobre los cuales se crearan las estructuras más complejas. Entonces, para la creación de los programas PreDeLP, comenzamos por los componentes básicos, que en nuestro caso son los *hechos* y las *presunciones* (ya que no dependen de ningún otro componente del modelo), que conforman lo que denominaremos la *base* de nuestro programa. Aquí, podemos definir el número *inicial* de elementos que queremos tener en nuestra base, como así también definir un criterio para elegir el tipo de elemento a crear. Una vez que tenemos creada nuestra base inicial, comenzamos a crear los argumentos utilizando la definición de *argumento completado* [Gar97], es decir, agregando a su cuerpo todas las reglas usadas en su cadena de derivación, salvo los hechos. Esto se muestra en la Definición 11. El Ejemplo 2 muestra un caso de argumento completado.

Definición 11 (Argumento Completado) *Sea $\mathcal{A}$ un argumento para el literal instanciado h, el argumento completado de $\mathcal{A}$, denotado $\mathcal{A}^c$, es un conjunto de reglas estrictas y reglas rebatibles formado por las reglas rebatibles de $\mathcal{A}$, más el subconjunto de reglas estrictas de Ω que fueron utilizadas para obtener la derivación de h a partir de $\Omega \cup \mathcal{A}$.*

Ejemplo 2 *En el programa formado $\Omega = \{h \leftarrow a; b \leftarrow d\}, \Theta = \{d, c\}, \Delta = \{a \prec b, c\}, \Phi = \{\emptyset\}$, $\mathcal{A} = \{a \prec b, c\}$ es un argumento para h. El argumento completado para $\mathcal{A}$ es $\mathcal{A}^c = \{h \leftarrow a; a \prec b, c; b \leftarrow d\}$.*

Por otro lado, los argumentos se organizan según un *nivel* en el programa; y este nivel está representado por un valor entero que indica el número máximo de reglas que

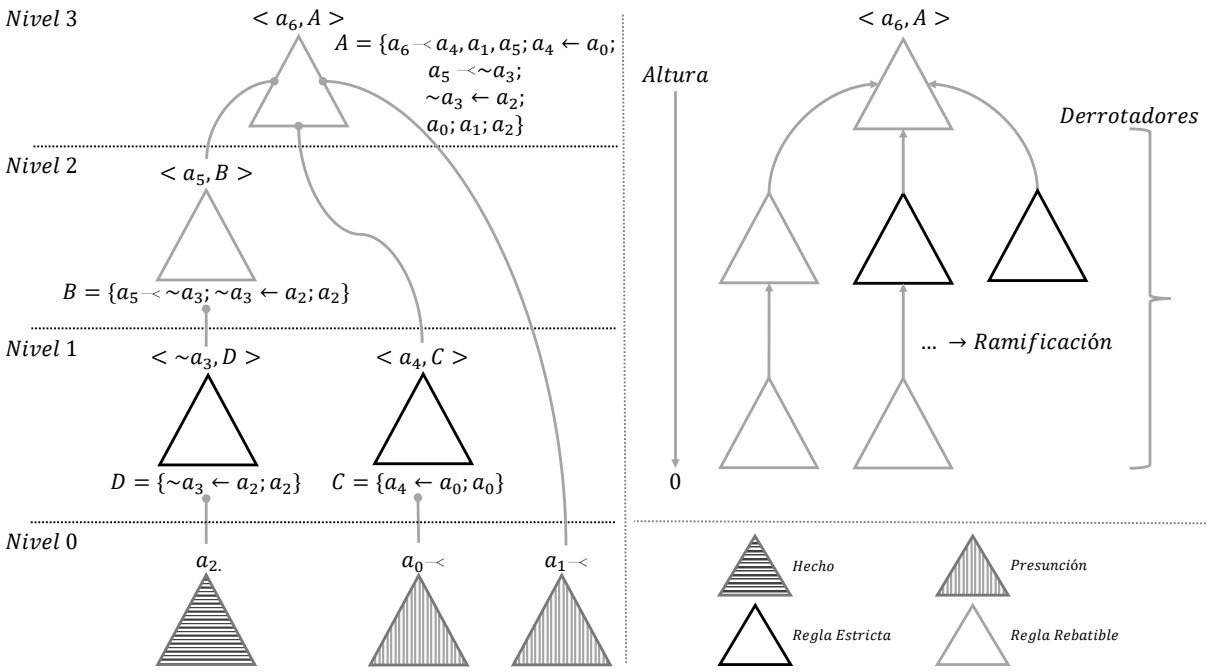


Figura 4.2: Diseño en niveles de los argumentos y árbol de dialéctica en el GPP.

se utilizan en su cadena de derivación hasta llegar a un elemento de la base. De esta manera, un argumento del nivel N, debe tener una regla del nivel $N-1$ y así hasta llegar a los componentes de la base (hechos y presunciones). Esto puede verse en la figura 4.2, donde el argumento $\langle a_5, \{a_5 -\prec \sim a_3; \sim a_3 \leftarrow a_2; a_2; \}\rangle$ es del nivel 2 ya que en su cadena de derivación tiene una regla del nivel 1 ($\sim a_3 \leftarrow a_2$), siendo a_2 del nivel 0. Los niveles se definieron para tener argumentos con diferente número de literales en su cuerpo y utilizar todos las reglas del programa, ya que cada argumento se construye en base a las reglas de los niveles inferiores. De esta manera, nos aseguramos que los componentes creados estén realmente vinculados y generen estructuras argumentales, además de evitar circularidad en las derivaciones.

Luego de crear todos los argumentos iniciales, se procede a crear los derrotadores para algunos de ellos. En particular, lo que se genera es un conjunto de árboles de dialéctica para cada argumento del nivel superior del programa. La decisión de crear derrotadores sólo para los argumentos del nivel superior se debe a que éstos son los que tienen un mayor número de puntos de ataque posibles (es decir, más literales en su cuerpo) y por lo tanto, es más sencillo y lógico que tengan derrotadores. Para crear los árboles de dialéctica, se define una altura y un número máximo de derrotadores por argumento; de esta manera,

por cada nivel del árbol se generan los derrotadores para los argumentos de ese nivel y así hasta llegar al último nivel del árbol (ver parte derecha de la figura 4.2). Se utilizó el criterio de *especificidad generalizada* [SGCS03] para la creación de los derrotadores, es decir, se crearon argumentos que son *estrictamente más específicos* que el argumento derrotado. Esto último se logró creando cada derrotador en base al *conjunto de activación no trivial* del argumento derrotado. Cada uno de estos aspectos del diseño serán explicados en detalle en las siguientes subsecciones. A continuación, se presentarán los parámetros que guiarán el proceso de generación.

4.2.2. Parámetros

Para generar un programa PreDeLP debemos tener en cuenta:

1. Número de elementos a generar: La cantidad de reglas, hechos y/o presunciones, argumentos, derrotadores, árboles de dialéctica, entre otros. Estos elementos son los que conforman el programa a generar, y son los que impactarán en el valor de las métricas.

2. Cómo generar cada elemento: Naturalmente, es necesario definir un procedimiento que nos garantice la correcta creación de cada elemento del programa.

3. La relación que debe existir entre los componentes del programa: Cada elemento que conforma el programa debe estar en relación con algún otro, debe existir para formar estructuras más generales o para permitir crear otros elementos.

Recordemos que nuestro objetivo es poder crear programas con diferentes tamaños y complejidades, lo cual medimos a través de las métricas definidas para ese modelo en particular; por lo tanto, el valor de los parámetros debe tener un impacto en el valor de las métricas. Las métricas para el MA son las siguientes: $\#Reglas$ (número total de reglas), $\#Hechos\_Pres$ (número de hechos y presunciones), $MDDL$ (longitud promedio de la derivación rebatibles de cualquier argumento), h (longitud máxima promedio de las líneas de argumentación), t (número promedio de líneas de argumentación que surgen a partir de un argumento) y τ (número promedio de árboles de dialéctica). Esto puede verse en el Cuadro 3.1 del Capítulo 3.

A continuación, se presentan los parámetros definidos para el GPP junto con su significado, tipo de valor, y las métricas del modelo sobre las que influyen.

- `BASE_SIZE` (`entero`): Mínimo número de hechos y presunciones para crear argumentos. Internamente, se considera al conjunto de hechos y presunciones como la "base" del programa. Este parámetro influye directamente sobre la métrica *#Hechos_Pres* del MA.
- `FACT_PROB` (`decimal`): Probabilidad de que un elemento de la base del programa sea un *hecho*. Este parámetro determina la cantidad de hechos en la base y, por lo tanto, también influye directamente en la métrica *#Hechos_Pres* del MA.
- `NEG_PROB` (`decimal`): Probabilidad para crear un átomo negado.
- `DRULE_PROB` (`decimal`): Probabilidad para crear una regla rebatible. Este parámetro influye en lás métricas *#Reglas*, *MDDL*, h y τ.
- `MAX_RULESPERHEAD` (`entero`): Máximo número de reglas con el mismo literal en su cabeza. Este valor influye en las métricas t y τ.
- `MAX_BODYSIZE` (`entero`): Máximo número de literales en el cuerpo de un argumento. Esta valor influye en las métricas *MDDL*, h y τ.
- `MIN_ARGSLEVEL` (`entero`): Mínimo número de argumentos distintos en un nivel del programa. Este valor influye sobre *#Reglas*.
- `LEVELS` (`entero`): Niveles del programa. El valor de un nivel indica el máximo número de reglas en la cadena de derivación de un argumento que pertenece a este nivel para alcanzar un elemento de la base. Este parámetro influye en la métricas *#Reglas*.
- `RAMIFICATION` (`entero`): Máximo número de derrotadores para un argumento. Este valor nos determina el número de líneas de argumentación diferentes que se generan de un argumento en particular (es por esto que se puede considerar como la "ramificación", ya que esas líneas conforman el árbol de dialéctica de un argumento). Este valor influye en las métricas t y h.
- `TREE_HEIGHT` (`entero`): Altura máxima de los árboles de dialéctica. Este valor influye en las métricas h y t.
- `INNER_PROB` (`decimal`): Probabilidad de que la relación de ataque sea de tipo "interno", es decir, la probabilidad de que los contraargumentos ataquen a conclusiones internas del argumento atacado.

- N_PROGRAMS (entero): Número de programas a generar.
- PREF_CRITERION: Criterio de preferencia entre argumentos a aplicar frente a situaciones de conflicto. Este criterio determina la manera de crear derrotadores para un argumento. En nuestro caso, utilizaremos el criterio de *especificidad generalizada*.

Existen restricciones y consideraciones sobre algunos parámetros. Por ejemplo, el tamaño inicial de la base del programa (BASE_SIZE), por sí mismo no nos indica el número *final* de hechos o presunciones a tener, ya que al momento de crear los argumentos y derrotadores podemos necesitar agregar nuevos elementos a la base para evitar circularidades y la reutilización de hechos o presunciones. Además, el parámetro FACT_PROB representa el volumen de hechos de la base, es decir, si tenemos FACT_PROB = 0,8, la base del programa estará formada inicialmente con un 80 % de hechos. Esto último también se cumple para los parámetros NEG_PROB, DRULE_PROB e INNER_PROB. Por otro lado, los parámetros N_PROGRAMS y PREF_CRITERION son más bien descriptivos, ya que el primero no influye en la creación de cada programa y el segundo indica que el criterio de preferencia para crear los derrotadores es *modular*; es decir, cada usuario puede especificar su criterio para decidir cuándo un argumento derrota a otro.

A partir de estos parámetros, la generación de programas PreDeLP consiste brevemente de los siguientes pasos:

1. Construir la base del programa.

   Se generan BASE_SIZE átomos (a_[Id]) y, por cada uno de ellos, se decide con FACT_PROB si se los agrega como hecho o como presunción al nivel 0 del programa. A su vez, cada literal será o no *negado* según NEG_PROB.

2. Construir todos los argumento recursivamente nivel por nivel.

   Comenzando desde el nivel 1, y hasta el nivel LEVELS−1, se crean MIN_ARGSLEVEL argumentos con conclusiones distintas y, por cada una de estas conclusiones (literales nuevos), se crea un número aleatorio entre 1 y MAX_RULESPERHEAD − 1 de reglas con ese literal en su cabeza. Estas últimas reglas también generan argumentos para el nivel que se está construyendo. Para la construcción del cuerpo de los argumentos, se genera un número aleatorio entre 1 y MAX_BODYSIZE y se usan los literales que se definieron en los niveles anteriores (el cuerpo del argumento del nivel N tiene que

tener al menos un literal definido en el nivel $N-1$). En caso de no tener los literales suficientes para construir el cuerpo del argumento, se crean nuevas reglas y literales para agregar en los niveles inferiores y así poder crear el argumento en cuestión.

3. Construir los árboles de dialéctica para cada argumento del nivel `LEVELS` del programa.

   Por cada argumento del nivel `LEVELS` se crea su árbol de dialéctica. Para esto, se comienza desde la raíz del árbol, el argumento en particular, y se genera un número entre 1 y `RAMIFICATION` de derrotadores. Luego, se repite el proceso por cada uno de estos derrotadores recursivamente hasta llegar a la altura 0 del árbol donde ya no se crean derrotadores (ver árbol de dialéctica de la figura 4.2). Para la construcción de un derrotador, se crea un argumento que soporta el complemento de la *conclusión* del argumento derrotado, y para la creación del cuerpo del derrotador, se utiliza el *conjunto de activación no trivial* del argumento derrotado *más* un literal nuevo del nivel 0 (aquí se utiliza la representación `d_[Id]`).

4. Transformar la representación interna del programa generado a la sintaxis de PreDeLP.

De esta manera, podemos establecer una relación entre los parámetros del GPP y las métricas del MA. Esto se muestran en el Cuadro 4.2, donde cada fila representa un parámetro y cada columna una métrica, así, la celda marcada con X significa que la variación en el valor de ese parámetro impacta en el valor de la métrica. Aquí podemos ver que una métrica puede estar relacionada con más de un parámetro y que no todos los parámetros tienen un impacto en alguna métrica, por ejemplo `NEG_PROB` e `INNER_PROB`; esto último es así ya que son valores que están relacionados a la definición de un literal y al tipo de derrota de un argumento respectivamente, y estos no influyen en el computo de ninguna métrica por sí mismos. Al final de esta sección se mostrará, por medio de algunos programas generados, cómo la variación de los parámetros impactan en el valor de las métricas computadas para esos programas. A continuación, se presentará en detalle cada uno de los algoritmos y las estructuras utilizadas para generar los programas.

Parámetros del GPP	Métricas del MA					
	#Hechos_Pres	*#Reglas*	*MDDL*	h	t	τ
BASE_SIZE	X					
FACT_PROB	X					
DRULE_PROB		X	X	X		X
MAX_RULESPERHEAD					X	X
MAX_BODYSIZE			X	X		X
MIN_ARGSLEVEL		X				
LEVEL		X				
RAMIFICATION				X	X	
TREE_HEIGHT				X	X	

Cuadro 4.2: Relación entre los parámetros del GPP y las métricas del MA.

4.2.3. Algoritmos y estructuras generales

La estructura principal consiste en un *diccionario*, es decir, un conjunto de elementos *clave* : *valor*. Aquí, *clave* representa el nivel del programa (el nivel 0 representa la base, es decir, el conjunto de hechos y presunciones) y *valor* es un diccionario formado por dos listas, *srule* o *drule*, una almacena las reglas estrictas, y la otra las reglas rebatibles para el nivel en particular. Esta estructura la denominamos `KB`, ya que representa la base de conocimiento del programa generado (Knowledge Base, en Inglés), y para acceder a cada nivel del programa, lo hacemos con `KB`[`nivel`][`tipo_regla`], donde *nivel* es un valor entero que representa un nivel en particular y *tipo_regla* indica la lista de reglas estrictas o rebatibles. Entonces, por ejemplo, la estructura `KB` final para el ejemplo de la figura 4.2 queda como:

$$
\begin{aligned}
&\texttt{KB} = \{\\
&0 : \{srule : [\texttt{a\_2}], drule : [\texttt{a\_0}, \texttt{a\_1}]\},\\
&1 : \{srule : [[\sim\texttt{a\_3}, [\texttt{a\_2}]], [\texttt{a\_4}, [\texttt{a\_0}]]], drule : []\},\\
&2 : \{srule : [], drule : [[\texttt{a\_5}, [\sim\texttt{a\_3}]]]\},\\
&3 : \{srule : [], drule : [[\texttt{a\_6}, [\texttt{a\_4}, \texttt{a\_1}, \texttt{a\_5}]]]\}\\
&\}
\end{aligned}
$$

Esta estructura es la que almacena toda la información del programa. Por otro lado, se identifica a cada literal nuevo con un valor numérico único, incrementando ese valor en 1 cada vez que se crea uno nuevo. Este valor numérico es accesible desde cualquier procedimiento y lo denotamos con `COUNT_LIT`, inicializado en 0; al igual que todos los valores de los parámetros definidos anteriormente, el valor de `COUNT_LIT` es accesible desde cualquier procedimiento interno del generador. De esta forma, se toma el valor de `COUNT_LIT` y se lo asocia al nuevo literal como su `Id` (`a_[Id]`). En relación al uso de parámetros que definen valores probabilísticos (`FACT_PROB`, `NEG_PROB`, `DRULE_PROB`) o valores máximos (`MAX_RULEPERHEAD`, `MAX_BODYSIZE`, `RAMIFICATION`), necesitamos generar números aleatorios para usar esos parámetros. En nuestro caso, usamos dos: *gen_random* y *gen_max(max)*. El primero genera un número aleatorio distribuido uniformemente del intervalo $[0, 1]$ y el segundo selecciona un valor del intervalo $[0, max]$, siguiendo también una distribución uniforme. Ambos algoritmos utilizan la librería `Numpy`[1] para simular la distribución de probabilidad y así generar o elegir el número a retornar.

Luego de estas presentaciones, estamos en condiciones de introducir los algoritmos principales para la generación de programas: *Generación de Literales Iniciales*, *Generación de la Base*, *Generación de Niveles*, *Generación de Argumentos* y *Generación de Árboles de Dialéctica*.

Generación de Literales Iniciales: Estos literales son creados al momento de definir un nuevo hecho, presunción, o regla para construir estructuras más complejas. El Algoritmo 4.1 (`gen_literal()`) muestra en detalle este procedimiento. Comenzamos creando un átomo `a` y le asignamos su identificador, es decir, el valor de `COUNT_LIT` en ese momento (línea 4). Luego, para determinar si negamos o no el átomo, generamos un número aleatorio (línea 6) y lo evaluamos con respecto al parámetro `NEG_PROB`; si es menor, el literal creado es el átomo negado, en otro caso, el literal es el átomo creado. Por último, retornamos el literal creado (línea 14).

Generación de la Base: El Algoritmo 4.2 (`gen_base()`) detalla este procedimiento. Para la generación de la base, se crean tantos literales como indica el parámetro `BASE_SIZE` (línea 4), luego, por cada literal creado, se genera un número aleatorio para determinar si el literal será agregado como un hecho o una presunción (línea 5), para lo cual comparamos el número generado con el valor del parámetro `FACT_PROB`; si es menor, agregamos el literal como hecho, en otro caso, agregamos el literal como una presunción (regla rebatible sin

[1] https://numpy.org/

Algoritmo 4.1: Crear un nuevo literal

```
1  def gen_literal()
       Resultado:
       literal: Nuevo literal creado
2      inicio
3          // primero creamos un nuevo átomo
4          atom ← "a_" + COUNT_LIT
5          // para determinar si negamos o no el nuevo átomo
6          polarity ← gen_random()
7          si polarity < NEG_PROB entonces
8              // negamos el átomo
9              literal ← ∼atom
10         sino
11             literal ← atom
12         fin
13         COUNT_LIT ← COUNT_LIT + 1
14         devolver literal
15     fin
16 fin
```

cuerpo). Estos literales son agregados al nivel 0 del programa, es decir, se *anexan* (append) a las listas de reglas que están en el nivel 0 de la KB.

Generación de Niveles: La generación de los niveles del programa se realiza de manera recursiva, comenzando por el nivel 1, y luego subiendo por los niveles hasta llegar a LEVELS. Esto se detalla en el Algoritmo 4.3 (gen_levels(*levels*)). En cada nivel, se ejecuta el procedimiento gen_arguments(*level*) (línea 7) para crear todos los argumentos usando reglas definidas en los niveles anteriores, es por esto que se comienza de los niveles inferiores. El procedimiento para crear los argumentos es el encargado de agregar las reglas en cada nivel del programa.

Generación de Argumentos: Este es uno de los algoritmos principales, ya que se encarga de crear las estructuras centrales del modelo, es decir, los argumentos. Esto se detalla en el Algoritmo 4.4 (gen_arguments(*level*)). Este procedimiento recibe como parámetro de entrada un valor entero que indica el nivel (*level*) que debe crear. El procedimiento comienza iterando sobre el valor de MIN_ARGSLEVEL para crear tantos argumentos como

Algoritmo 4.2: Generación de la Base

```
1  def gen_base()
       Resultado:
       KB[0]: La base del programa
2      inicio
3          para 1 hasta BASE_SIZE hacer
4              literal ← gen_literal()
5              fact_or_pres ← gen_random()
6              si fact_or_pres < FACT_PROB entonces
7                  // agregamos el nuevo literal como hecho
8                  KB[0][srule].append(literal)
9              sino
10                 // agregar el nuevo literal como presunción
11                 KB[0][drule].append(literal)
12             fin
13         fin
14     fin
15 fin
```

indique ese parámetro (línea 4). Aquí, es importante aclarar que sobrecargamos el concepto de *regla* con el de *argumento*, ya que al crear una regla lo que se está haciendo es crear un argumento también. Esto es así ya que el cuerpo de cada regla está formado por literales que son los consecuentes de las reglas de un nivel inferior y así hasta llegar a la base, lo que nos garantiza que existe una derivación para el consecuente de cada regla que creamos y, por lo tanto, cada regla creada constituye un argumento en sí misma (ya que consideramos en todo momento la noción de *argumento completado* [Gar97]). De esta manera, el procedimiento para generar argumentos consiste en crear una relación que asocie un *consecuente* (conclusión del argumento) con un *antecedente* (cuerpo del argumento) a través de una regla estricta o una regla rebatible. Para esto, se comienza creando el consecuente de la regla, o conclusión del argumento, generando un nuevo literal (línea 5). Luego, se define un número máximo de reglas que definen ese nuevo literal (línea 7) utilizando el parámetro `MAX_RULESPERHEAD`, es decir, el número máximo de argumentos que soportan la conclusión creada. Luego, se itera por el número de reglas a crear con el mismo consecuente (línea 8) y se crea el antecedente, o cuerpo del argumento, de cada una

Algoritmo 4.3: Generación de niveles

```
1  def gen_levels(levels)
       Datos: levels: Niveles del programa
       Resultado: KB: Todos los niveles del programa
2      inicio
3          si levels=1 entonces
4              gen_arguments(levels)
5          sino
6              gen_levels(levels − 1)
7              gen_arguments(levels)
8          fin
9      fin
10 fin
```

de ellas (línea 9). Una vez que se tiene el consecuente y el antecedente de la regla creada, se genera un número aleatorio y se lo compara con el valor del parámetro `DRULE_PROB`; si es menor, la regla será rebatible (línea 13), en otro caso, será estricta (línea 16). Por último, se agrega cada regla según su tipo al nivel *level* de la `KB`.

Generación del Cuerpo de un Argumento: Este procedimiento se detalla en el Algoritmo 4.5 (`gen_body_arg`(*level*, *conclusion*)). Este procedimiento recibe como parámetro *level* y *conclusion*: el primero indica el nivel al que tiene que pertenecer el argumento a crear y el segundo es la conclusión del argumento a crear. El objetivo final es crear una lista de literales que será el antecedente de una regla (cuerpo de un argumento). Para esto, se comienza definiendo el *tamaño* del antecedente de la regla (línea 5), es decir, el número de literales que darán soporte a la *conclusion* recibida como parámetro de entrada. Una vez que se tiene el número de literales, se selecciona *al menos* una regla del nivel anterior y se toma su consecuente para ser agregado al cuerpo (línea 7). Luego, se completa el cuerpo con los consecuentes de reglas elegidas al azar de los niveles 0 a `LEVEL` − 1 (línea 9 a 13). Al momento de seleccionar una regla para tomar su consecuente (procedimiento `choose_consequent`(*level*, *conclusion*)), se elige de aquellas del nivel *level* evitando las que tengan en su consecuente la *conclusion* o su complemento, esto para evitar argumentos inconsistentes y situaciones de circularidad. Por último, se retorna el cuerpo del argumento creado (línea 15).

Elección de una regla para formar el cuerpo de un argumento: Este proce-

Algoritmo 4.4: Creación de Argumentos

```
1  def gen_arguments(level)
       Datos: level: Nivel del programa a crear
       Resultado: KB[level]: Un nivel del programa
2      inicio
3          // crear MIN_ARGSLEVEL argumentos
4          para 1 hasta MIN_ARGSLEVEL hacer
5              conclusion ← gen_literal()
6              // máximo número de reglas para la conclusión
7              n_rules ← gen_max(MAX_RULESPERHEAD)
8              para 1 hasta n_rules hacer
9                  argument_body ← gen_body_arg(level,conclusion)
10                 drule_o_srule ← gen_random()
11                 si drule_o_srule < DRULE_PROB entonces
12                     // agregamos la regla rebatible al nivel level
13                     KB[level][drule].append([conclusion,argument_body])
14                 sino
15                     // agregamos la regla estricta al nivel level
16                     KB[level][srule].append([conclusion,argument_body])
17                 fin
18             fin
19         fin
20     fin
21 fin
```

dimiento se detalla en el Algoritmo 4.6 (`choose_consequent`(*level*, *parent_consequent*)). Este procedimiento selecciona una regla del nivel *level* de la `KB` y retorna su consecuente. Para esto, primero se filtran aquellas reglas que generan inconsistencias o circularidades; es decir, aquellas que tengan como consecuente al consecuente de la regla que estamos creando en el nivel superior, el cual es recibido en el parámetro *parent_consequent* (línea 4 y 5). Luego, para determinar si elegimos una regla estricta o rebatible, se genera un número aleatorio (línea 6) y se compara su valor con `DRULE_PROB`; si es menor, se elige una regla de entre las posibles reglas rebatibles (línea 9), en otro caso, se elige una regla de entre las posibles reglas estrictas (línea 17). En los casos donde tenemos que elegir una regla de un conjunto vacío (líneas 11 a 13 y 19 a 21), se crean nuevas reglas según el tipo

Algoritmo 4.5: Crear cuerpo de un argumento

```
1  def gen_body_arg(level, conclusion)
       Datos:
       level: Nivel al que tiene que pertenecer el argumento a crear
       conclusion: Conclusión del argumento
       Resultado:
       body: Lista con literales de los niveles inferiores a level
2      inicio
3          body ← [ ]
4          // para determinar el tamaño del cuerpo del argumento
5          body_size ← gen_max(MAX_BODYSIZE)
6          // elegir consecuente de una regla del nivel anterior
7          consequent_rule ← choose_consequent(level − 1, conclusion)
8          body.anexar(consequent_rule)
9          para 1 hasta body_size − 1 hacer
10             // seleccionar un nivel inferior
11             select_level ← gen_max(level)
12             consequent_rule ← choose_consequent(select_level, conclusion)
13             body.append(consequent_rule)
14         fin
15         devolver body
16     fin
17 fin
```

indicado y se las agrega a la KB en el nivel 0. Por último, se retorna el consecuente de la regla elegida o creada (línea 24).

Generación de los Árboles de Dialéctica: Como se mencionó anteriormente, sólo se crearán árboles de dialéctica para los argumentos del último nivel del programa. Para detallar este procedimiento, primero vamos a especificar algunos procedimientos menores que son usados en la generación de los árboles; luego, debemos citar algunas definiciones que serán necesarias para la comprensión del algoritmo en general. Dichos procedimientos son:

- `arg_completed`(*argument*): Este procedimiento retorna el *argumento completado* del argumento recibido como parámetro de entrada, es decir, toma todos los literales de

Algoritmo 4.6: Elegir la regla de un nivel y tomar su consecuente

```
1  def choose_consequent(level, parent_consequent)
       Datos:
       level: Nivel del cual se debe elegir una regla y tomar su consecuente
       parent_consequent: Consecuente de la regla del nivel level + 1
       Resultado:
       consequent: Consecuente de la regla elegida del nivel level
2      inicio
3          consequent ← ∅
4          drules_possible ← todas las reglas de KB[level][drule] que no tengan como consecuente a
             parent_consequent o su complemento
5          srules_possible ← todas las reglas de KB[level][srule] que no tengan como consecuente a
             parent_consequent o su complemento
6          drule_or_srule ← gen_random()
7          si drule_or_srule < DRULE_PROB entonces
8              si drules_possible es no vacío entonces
9                  consequent ← tomar el consecuente de una regla al azar de drules_possible
10             sino
11                 literal ← gen_literal()
12                 KB[0][drule].append(literal)
13                 consequent ← literal
14             fin
15         sino
16             si srules_possible es no vacío entonces
17                 consequent ← tomar el consecuente de una regla al azar de srules_possible
18             sino
19                 literal ← gen_literal()
20                 KB[0][srule].append(literal)
21                 consequent ← literal
22             fin
23         fin
24         devolver consequent
25     fin
26 fin
```

su cuerpo y los sustituye por las reglas que los definen, sean estrictas o rebatibles (salvo los hechos).

- `arg_conclusion`(*arg_completed*): Este procedimiento retorna la conclusión de un argumento completado que recibe como parámetro de entrada.

- `is_defeasible`(*arg_completed*): Es verdadero si el argumento completado de entrada es un argumento que puede ser derrotado, es decir, si no está construido usando sólo reglas estrictas y hechos.

Luego de estas presentaciones, es necesario recordar y citar algunas definciones: *Criterio de especificidad, Conjunto de literales básicos, Conjunto de activación* y *Conjunto de activación no trivial*.

Recordemos que cuando dos conclusiones complementarias pueden ser derivadas rebatiblemente de un programa PreDeLP, resulta necesario definir un criterio de inferencia, a fin de que sólo una de las conclusiones tenga éxito, cuyo argumento será el argumento *derrotador*. Para esto, es necesario utilizar un criterio de comparación entre los argumentos que sustentan dichas conclusiones. La comparación de argumentos es un problema abierto, y existen diferentes propuestas que intentan solucionarlo. Una de ellas es el *criterio de especificidad*, el cual fue descripto por Poole en [Poo85], y separadamente por Loui en [Lou86]. Algunos sistemas de razonamiento rebatible han utilizado este criterio para comparar sólo reglas rebatibles (ver [Nut92]), y otros para comparar argumentos enteros (ver [SL92]). Aquí también lo usaremos para comparar argumentos completos.

El objetivo de esta introducción es detallar el procedimiento a aplicar para generar un derrotador utilizando el criterio de especificidad generalizada; es decir, cómo construir un argumento que, al ser comparado con otro argumento en conflicto, salga como vencedor del conflicto y constituya un derrotador. Esta es la tarea principal en la creación de los árboles de dialéctica. Para esto, utilizaremos algunas técnicas y conceptos definidos en [Gar97], comenzando por la definición del criterio de especificidad en términos de *conjuntos de activación*.

La Definición 12, muestra el criterio de especificidad entre argumentos del sistema de argumentación rebatible definido en [SL92]. En dicho sistema, $\Delta^{\downarrow}$ representa el conjunto de reglas rebatibles instanciadas, y $\mathcal{K}$ al conjunto de reglas no rebatibles ($\mathcal{K} = \mathcal{K}_G \cup \mathcal{K}_P$), distinguiendo con $\mathcal{K}_P$ al conjunto particular (subconjunto de hechos instanciados) y con $\mathcal{K}_G$ al conocimiento general (reglas sin instanciar). El símbolo "$\mid\sim$" se utiliza para denotar la derivación rebatible.

Definición 12 (Especificidad [SL92]) *Sea* $L = \{l : l$ *es un literal instanciado y* $\mathcal{K} \cup \Delta^{\downarrow} \mid\sim l\}$. *El argumento* $\mathcal{A}$ *para* h_1 *es estrictamente más específico que el argumento* $\mathcal{B}$ *para* h_2, *si y solo si:*

1. *para todo conjunto* $C \subseteq L$
   si $\mathcal{K}_G \cup C \cup \mathcal{A} \mid\sim h_1$ *(*C *activa* $\mathcal{A}$*) y* $\mathcal{K}_G \cup C \nvdash h_1$, *(activación no trivial) entonces* $\mathcal{K}_G \cup C \cup \mathcal{B} \mid\sim h_2$ *(*C *activa* $\mathcal{B}$*).*

2. *existe un conjunto* $C' \subseteq L$ *tal que:*
   $\mathcal{K}_G \cup C' \cup \mathcal{B} \mid\sim h_2$ *(*C' *activa* $\mathcal{B}$*),* $\mathcal{K}_G \cup C' \nvdash h_2$*, (activación no trivial) y* $\mathcal{K}_G \cup C' \cup \mathcal{A} \mid\approx h_1$ *(*C' *no activa* $\mathcal{A}$*).*

Si un argumento $\mathcal{A}$ es estrictamente más específico que $\mathcal{B}$, se denotará $\mathcal{A} \succ \mathcal{B}$. En el caso que $\mathcal{A} \not\succ \mathcal{B}$ y que $\mathcal{B} \not\succ \mathcal{A}$, se dirá que $\mathcal{A}$ y $\mathcal{B}$ son incomparables, y se denotará con $\mathcal{A} \not\approx \mathcal{B}$.

Si utilizamos la Definición 12 para aplicar el criterio de especificidad, hay que considerar todos los subconjuntos de L. El conjunto L contiene *todos* los literales que se pueden derivar rebatiblemente del programa y, si tiene n elementos, entonces habrá 2^n conjuntos para considerar. El principal problema es que se están considerando gran cantidad de conjuntos de literales que no están relacionados con los argumentos en comparación, siendo irrelevantes durante el proceso de decidir qué argumento prevalece. García [Gar97] presenta una definición equivalente a la Definición 12 pero con un costo computacional menor, ya que se focaliza únicamente sobre los literales de los argumentos en comparación. A continuación, se citarán algunos conceptos introducidos en [Gar97] para presentar dicha versión de especificidad (que será la que utilizamos en esta tesis para la creación de los derrotadores).

Definición 13 (Conjunto de literales básicos de $\mathcal{A}^c$) *Sea* $\mathcal{A}^c$ *un argumento completado (definición 11), el conjunto de literales básicos de* $\mathcal{A}^c$ *denotado* $Lit(\mathcal{A}^c)$*, es el conjunto de literales que aparecen en los antecedentes y consecuentes de toda cláusula de* $\mathcal{A}^c$*.*

Definición 14 (Conjunto de activación) *Sea* $\mathcal{A}^c$ *un argumento completado, y sea* $Lit(\mathcal{A}^c)$ *el conjunto de literales correspondientes. Un subconjunto* $U \subseteq Lit(\mathcal{A}^c)$ *es un conjunto de activación de* $\mathcal{A}^c$*, si* U *junto con* $\mathcal{A}^c$ *derivan rebatiblemente a* h *(i.e.,* $U \cup \mathcal{A}^c \mid\sim h$*), y además* U *es minimal con respecto a la inclusión de conjuntos (i.e.,* $\nexists\, U' \subseteq U$ *tal que* $U' \cup \mathcal{A}^c \mid\sim h$*). Se denotará con* $Act(\mathcal{A}^c)$*, al conjunto de todos los conjuntos de activación de* $\mathcal{A}^c$*.*

Los conjuntos de activación serán los encargados de reemplazar a los subconjuntos de C de L que se utilizan en la definición 12. La diferencia entre considerar sólo los conjuntos de activación en lugar de todos los conjuntos de L, puede verse en el ejemplo presentado en [Gar97] y citado a continuación:

Ejemplo 3 *Considerando el siguiente programa, donde* Π *es el conjunto de hechos y reglas estrictas y* Δ *el conjunto de reglas rebatibles:*

$$\Pi = \left\{ \begin{array}{cc} a \leftarrow b,c & c \leftarrow f,g \\ z \leftarrow h & x \leftarrow h \\ y \leftarrow h & h \\ i & j \\ k & g \end{array} \right\} \qquad \Delta = \left\{ \begin{array}{cc} b \prec d,e & d \prec h,i \\ f \prec j,k & t \prec x \\ u \prec y & \end{array} \right\}$$

Sea $\mathcal{A}^c$ *el argumento completado para el literal "a":*

$$\mathcal{A}^c = \{a \leftarrow b,c; b \prec d,e; d \prec h,i; c \leftarrow f,g; f \prec j,k\}$$

En este caso $L = \{a,b,c,d,e,f,g,h,i,j,k,u,t,x,y,z\}$*. Por lo tanto existen* 2^{16} *subconjuntos para considerar, además* $Lit(\mathcal{A}^c) = \{a,b,c,d,e,f,g,h,i,j,k\}$*, lo cual sólo bajaría el número de subconjuntos a 2048. Sin embargo, solamente hay 10 conjuntos de activación para considerar:*

$$\left\{ \begin{array}{ccccc} \{a\} & \{b,c\} & \{d,e,c\} & \{h,i,e,c\} & \{h,i,e,f,g\} \\ \{h,i,e,j,k,g\} & \{d,e,f,g\} & \{d,e,j,k,g\} & \{b,f,g\} & \{b,j,k,g\} \end{array} \right\}$$

Es importante remarcar que dado un argumento completado $\mathcal{A}^c$ para una meta h, la Definición 14 no excluye la posibilidad de que un conjunto de activación U derive *trivialmente* la meta h (*i.e.*, , $U \cup \mathcal{K}_G \vdash h$). En el ejemplo presentado, los conjuntos $\{b,c\}$, $\{a\}$ y $\{b,f,g\}$ son conjuntos de activación que trivialmente derivan la conclusión h. Este tipo de conjuntos no se tienen en cuenta en la definición de especificidad, ya que la condición "$U \cup \mathcal{K}_G \nvdash h$", lo prohíbe. Es por esto que eliminar dichos conjuntos sería más útil. Para esto se definen los conjuntos de activación no trivial.

Definición 15 (Conjuntos de activación no trivial) *Dado un argumento completado* $\mathcal{A}^c$ *para la meta* h*, se dirá que* U *es un conjunto de activación no trivial de* $\mathcal{A}^c$*, si* U *es un conjunto de activación de* $\mathcal{A}^c$*, y* $U \cup \mathcal{K}_G \nvdash h$*. Se denotará con Act-NT(*$\mathcal{A}^c$*) al conjunto de todos los conjuntos de activación no triviales de* $\mathcal{A}^c$*.*

El Algoritmo 4.7 indica cómo calcular todos los conjuntos de activación no triviales de un argumento completado. Aquí, para determinar si un conjunto de activación es trivial o no, se observa si fue usada o no una regla rebatible (línea 7). Este procedimiento calcula todos los conjuntos de activación recorriendo el argumento completado.

Algoritmo 4.7: Construir conjuntos de activación no trivial [Gar97]

```
1  def get_ntact_sets(A^c, h)
       Datos:
       A^c: Argumento completado
       h: Conclusión del argumento
       Resultado:
       Act-NT(A^c): Conjunto de activación no trivial
2      inicio
3          Se inicializa una cola C con el par (h, trivial)
4          Act(A^c) y Act-NT(A^c) se inicializan en vacío
5          mientras C no esté vacía hacer
6              Sacar de C un par (conj, tipo)
7              Para cada literal l_i ∈ conj que sea consecuente de una regla r de A^c,
                 reemplazar en conj a l_i por los literales del antecedente de r, con lo cuál se
                 obtiene un nuevo conjunto de activación de c_i. El tipo de c_i será trivial
                 sólo si el tipo de conj es trivial y r es una CPE (regla estricta). En caso
                 contrario el tipo de c_i será no trivial
8              Todos lo nuevos conjuntos de activación c_i que no hallan sido previamente
                 expandidos, son agregados a C
9              El conjunto conj se agrega a Act(A^c). Si el conjunto conj es no trivial
                 entonces conj se agrega a Act-NT(A^c)
10         fin
11         devolver Act-NT(A^c)
12     fin
13 fin
```

Luego de estas presentaciones, se redefine el criterio de especificidad para que sólo tenga que considerar los conjuntos de activación no triviales.

Definición 16 (Especificidad [Gar97]) *Dado el argumento no vacío $\mathcal{A}$ para la meta h_1, y el argumento no vacío $\mathcal{B}$ para h_2, y $\mathcal{A}^c$, $\mathcal{B}^c$ sus respectivos argumentos completados. Se dirá que $\mathcal{A}$ es estrictamente más específico que $\mathcal{B}$ si y sólo si,*

1. *para todo conjunto $U \in$ Act-NT($\mathcal{A}^c$), se cumple que $U \cup \mathcal{K}_G \cup \mathcal{B}^c \mid\sim h_2$, y*
2. *existe un conjunto $U' \in$ Act-NT($\mathcal{B}^c$), se cumple que $U' \cup \mathcal{A}^c \mid\nsim h_1$.*

A partir de esta última definición de especificidad, el procedimiento para crear un argumento $\mathcal{A}$ con conclusión h_1 que derrote a un argumento $\mathcal{B}$ con conclusión h_2, se reduce a crear $\mathcal{A}$ tal que que todos los conjuntos de activación no trivial de $\mathcal{A}^c$, junto con las reglas estrictas del programa y $\mathcal{B}^c$, permiten derivar h_2 (condición 1 de especificidad) y garantizar que al menos uno de los conjuntos de activación no trivial de $\mathcal{B}^c$ junto con $\mathcal{A}^c$ no derivan h_1 (condición 2 de especificidad). Estas condiciones pueden cumplirse creando al argumento derrotador a partir de un conjunto de activación no trivial del argumento derrotado. Esto se detalla a continuación.

El procedimiento para generar un derrotador $\mathcal{A}$ para un argumento $\mathcal{B}$, consiste en unir uno de los conjuntos de activación no trivial de $\mathcal{B}^c$ con una presunción *nueva* de la base y utilizar el conjunto resultante para crear el cuerpo de $\mathcal{A}$. La conclusión del argumento derrotador será el complemento de la conclusión del argumento derrotado. Este procedimiento se detalla a continuación.

1. Construir el argumento completado $\mathcal{B}^c$ de $\langle \mathcal{B}, h_2 \rangle$.
2. Calcular Act-NT($\mathcal{B}^c$), es decir, los conjuntos de activación no trivial de $\mathcal{B}^c$.
3. Seleccionar un conjunto $\mathcal{D} \in$ Act-NT($\mathcal{B}^c$).
4. Crear una nueva presunción d y agregarla al programa.
5. Crear el cuerpo de $\mathcal{A}$ como $\mathcal{A} = \mathcal{D} \cup d$.
6. El argumento derrotador es $\langle \mathcal{A}, h_1 \rangle$.

De esta manera, podemos satisfacer las dos condiciones de la definición 16, ya que todos los conjuntos de activación no trivial de $\mathcal{A}^c$ (sólo tiene uno, $\mathcal{D} \cup d$), junto con las reglas estrictas del programa y $\mathcal{B}^c$ permiten derivar h_2 (ya que $\mathcal{D}$ por sí mismo es un conjunto de activación no trivial de $\mathcal{B}^c$), y existe un conjunto de Act-NT($\mathcal{B}^c$) que junto con $\mathcal{A}^c$ no derivan h_1 (dicho conjunto es $\mathcal{D}$). En nuestro caso, $h1$ es el complemento de h_2.

Una vez que tenemos el procedimiento para generar un derrotador para un argumento, podemos detallar el algoritmo para la creación de los árboles de dialéctica. Esto se detalla en el Algoritmo 4.8 (`gen_dialectical_trees`). Es importante remarcar que este procedimiento se invoca luego de crear todos los argumentos iniciales del programa, es decir,

Algoritmo 4.8: Crear Árboles de Dialéctica

```
1  def gen_dialectical_trees()
       Resultado: Árboles de Dialéctica para argumentos del nivel LEVELS
2      inicio
3          // tomar todas las reglas del nivel superior
4          root_arguments ← KB[LEVELS][drule] ∪ KB[LEVELS][srule]
5          para cada argument en root_arguments hacer
6              // construir el argumento completado
7              arg_completed ← arg_completed(argument)
8              si is_defeasible(arg_completed) entonces
9                  // tomar la conclusión del argumento
10                 conclusion ← arg_conclusion(arg_completed)
11                 // construir su conjunto de activación no trivial
12                 ntactsets ← get_ntact_sets(arg_completed,conclusion)
13                 // crear árbol de dialéctica
14                 gen_tree(conclusion,ntactsets,TREE_HEIGHT,RAMIFICATION)
15             fin
16         fin
17     fin
18 fin
```

todos los niveles ya tienen argumentos generados. El procedimiento comienza tomando todos los argumentos del último nivel del programa, es decir, todas las reglas rebatibles y estrictas del nivel `LEVELS`; estos serán los argumentos raíz de los árboles a crear (línea 4). Luego, por cada argumento raíz, se computa su correspondiente argumento completado (línea 7) y se verifica si es un argumento que puede ser derrotado; en caso afirmativo, se toma su conclusión (línea 10) y se computan sus conjuntos de activación no trivial (línea 12). Finalmente, se invoca el procedimiento para generar un árbol de dialéctica para un argumento en particular. Este último procedimiento se invoca en la línea 14 y recibe como parámetros de entrada la conclusión del argumento raíz (*conclusion*), los conjuntos de activación no trivial de dicho argumento (*ntactsets*), la altura del árbol (parámetro `TREE_HEIGHT`) y el número de derrotadores para el argumento raíz (parámetro `RAMIFICATION`).

El Algoritmo 4.9 detalla el procedimiento para generar un árbol de dialéctica para un

argumento raíz en particular. El proceso es recursivo, creando primero todos los derrotadores para el argumento raíz y bajando hasta llegar al último nivel del árbol, donde crea los argumentos que no tienen derrotadores (nodos hoja). Tomando como parámetro de entrada la altura actual del árbol (*height*), se consulta si la misma es igual a 1 (línea 3), y en caso afirmativo, se crean los derrotadres que serán los nodos hoja (línea 5). En caso de que la altura actual del árbol sea distinta de 1, se crean tantos derrotadores como indique el parámetro *ramification* (sólo en la primera llamada) y se itera por cada uno de ellos tomando a cada derrotador como la raíz del subárbol a construir, para lo cual se debe tomar la conclusión del derrotador (*conclusion*, línea 14), calcular sus conjuntos de activación no trivial (línea 15 y 16), generar un nuevo valor de ramificación (*ramification*, línea 17), y finalmente invocar de nuevo al procedimiento general `gen_tree` con los nuevos valores de parámetros y restando 1 al parámetro altura (*height*). De esta manera, el árbol se construye nivel a nivel hasta llegar a la altura 1, donde se crean los nodos hoja. Es importante remarcar que sólo los argumentos raíz, es decir, los argumentos del nivel superior del programa, tienen exactamente la cantidad de derrotadores que indica el parámetro `RAMIFICATION`, dado que para los demás derrotadores del árbol se genera un número entre 1 y `RAMIFICATION` (línea 17) para indicar cuántos derrotadores tendrá. Para crear el derrotador de un argumento con conclusión *conclusion* y conjuntos de activación no trivial *ntactsets*, se invoca el procedimiento `gen_defeater`(*conclusion*, *ntactsets*); para calcular los conjuntos de activación no trivial de un argumento, se invoca el procedimiento `get_ntact_sets` detallado en el Algoritmo 4.7.

Finalmente, el Algoritmo 4.10 detalla el procedimiento para generar el derrotador de un argumento. El procedimiento recibe como parámetro de entrada *conclusion*, que corresponde a la conclusión del argumento a derrotar, que a su vez será el *punto de contraargumentación*, y *ntactsets*, que corresponde a los conjuntos de activación no trivial del argumento a derrotar. Se comienza creando la conclusión del derrotador, es decir, el complemento de la conclusión del argumento derrotado (línea 4); luego se crea un nuevo literal y se agrega el mismo al nivel 0 del programa como una presunción (línea 8). Este nuevo elemento de la base será utilizado para crear el cuerpo del derrotador. A continuación, se crea el cuerpo del argumento derrotador utilizando un conjunto seleccionado al azar de *ntactsets* (línea 10) y agregando a dicho conjunto el literal creado anteriormente (línea 12). El conjunto resultante constituye el cuerpo del argumento derrotador. La conclusión y el cuerpo del derrotador se vinculan por medio de una regla rebatible y se anexa la misma al nivel `LEVELS` + 1 del programa (se almacena en este nivel sólo para tener

Algoritmo 4.9: Crear Árbol de Dialéctica para un argumento

```
1  def gen_tree(conclusion, ntactsets, height, ramification)
       Datos:
       conclusion: Conclusión del argumento raíz
       ntactsets: Conjuntos de activación no trivial del argumento raíz
       height: Altura del árbol a crear
       ramification: Número de derrotadores para el argumento raíz
       Resultado:
       Conjunto de derrotadores que forman el árbol de dialéctica de un argumento
2      inicio
3          si height = 1 entonces
4              para 1 hasta ramification hacer
5                  gen_defeater(conclusion, ntactsets)
6              fin
7          sino
8              defeaters ← [ ]
9              para 1 hasta ramification hacer
10                 defeater ← gen_defeater(conclusion, ntactsets)
11                 defeaters.append(defeater)
12             fin
13             para cada defeater en defeaters hacer
14                 conclusion ← arg_conclusion(defeater)
15                 arg_completed ← arg_completed(defeater)
16                 ntactsets ← get_ntact_sets(arg_completed,conclusion)
17                 ramification ← gen_max(ramification)
18                 gen_tree(conclusion,ntactsets,height − 1,ramification)
19             fin
20         fin
21     fin
22 fin
```

una mejor organización de los derrotadores creados). Por último, se retorna el argumento derrotador (línea 17).

Los algoritmos presentados constituyen los procedimientos principales para la generación de programas PreDeLP a partir de los parámetros definidos. De esta manera, el procedimiento general para crear un conjunto de modelos $\mathcal{P}_{MA}$ es el siguiente:

1. Iterar sobre el número de programas a crear (`N_PROGRMAS`).

Algoritmo 4.10: Crear derrotador

```
1  def gen_defeater(conclusion, ntactsets)
       Datos:
       conclusion: Punto de contraargumentación
       ntactsets: Conjuntos de activación no trivial del argumento a derrotar
       Resultado:
       [def_conclusion, [def_body]]: Argumento derrotador (una regla rebatible del nivel
       LEVELS + 1)
2      inicio
3          // se crea la conclusión del derrotador
4          def_conclusion ← complement(conclusion)
5          // se crea un literal para el cuerpo del derrotador
6          def_literal ← gen_literal()
7          // se suma el nuevo literal a la base
8          KB[0][drule].append(def_literal)
9          // se selecciona al azar un conjunto de ntactsets
10         def_ntactset ← choice_one(ntactsets)
11         // se crea el cuerpo del derrotador usando el conjunto
               seleccionado y agregando el nuevo literal
12         def_body ← def_ntactset.anexar(def_literal)
13         // se agrega el derrotador como regla rebatible en el nivel LEVELS
               + 1
14         KB[LEVELS + 1][drule].append([def_conclusion, [def_body]])
15     fin
16     // Retornar el derrotador creado
17     devolver [def_conclusion, [def_body]]
18 fin
```

1.1 `gen_base`(): Generar la *base* del programa.

1.2 `gen_levels`(`LEVELS`): Generar todos los niveles, es decir, crear todos los argumentos iniciales.

1.3 `gen_dialectical_trees`(): Generar los árboles de dialéctica para los argumentos del nivel superior del programa.

2. Recorrer la estructura `KB` y escribir cada regla con la sintaxis de PreDeLP.

En la siguiente subsección, se presentará un conjunto de modelos creados con este generador con el objetivo de mostrar las capacidades para generar modelos con diferentes niveles de tamaño y complejidad. En particular, se mostrará que la variación de los parámetros permite generar programas con diferentes valores de métricas.

4.2.4. Ejemplos de Modelos Generados

Esta subsección tiene como finalidad mostar las capacidades del generador detallado previamente en relación a la creación de programas PreDeLP que contengan reglas que permitan generar estructuras argumentales no triviales. Con este objetivo, se llevó a cabo un conjunto de pruebas para generar diferentes conjuntos de programas con diferentes valores de métricas, tanto en tamaño como en complejidad. Se generaron tres conjuntos de prueba (Caso 1, Caso 2 y Caso 3) a partir de diferentes valores de parámetros. Cada uno de estos conjuntos está formado por 1000 programas. Los resultados se detallan a continuación, separados según el atributo en el que se enfoquen (tamaño o complejidad).

Variación en el tamaño de los modelos: Se comenzará con la variación de las métricas de tamaño de un modelo $\mathcal{P}_{MA}$, es decir, las métricas *#Hechos_Pres* y *#Reglas*. Para obtener variaciones en las mismas, y teniendo en cuenta el Cuadro 4.2, se definieron valores para los parámetros requeridos agrupándolos en tres escenarios con el objetivo de hacer crecer el tamaño del programa. El resultado de estas pruebas puede verse en la Figura 4.3, en la misma se muestran los valores de los parámetros para cada escenario y el valor de las métricas en cada uno de ellos. Como podemos ver, en cada caso tenemos un aumento en el valor de las métricas conforme aumentamos el valor de los parámetros; a su vez, en el escenario 1 y 2, la diferencia entre las dos métricas no es significativa. En el escenario 3, hay una gran diferencia entre el número de reglas y el número de hechos y presunciones, esto se debe a que tenemos el doble de niveles (`LEVELS`), y el doble de argumentos distintos en cada uno de esos niveles (`MIN_ARGSLEVEL`) en comparación con los otros dos escenarios, lo que genera un mayor número de reglas necesarias. En estas pruebas, todos los parámetros relacionados a la generación de árboles de dialéctica son iguales a 1, ya que sólo nos interesa mostrar la variación en el tamaño de los programas; naturalmente, estos tamaños crecerán cuando se definan valores mayores para el número de derrotadores y altura de los árboles de diaéctica, ya que será necesario crear todos los derrotadores, y esto a su vez requiere la creación de nuevas reglas en el programa.

Parámetro	Caso 1	Caso 2	Caso 3
BASE_SIZE	5	20	40
FACT_PROB	5	50	80
DRULE_PROB	5	50	80
MIN_ARGSLEVEL	3	5	10
LEVELS	3	5	10

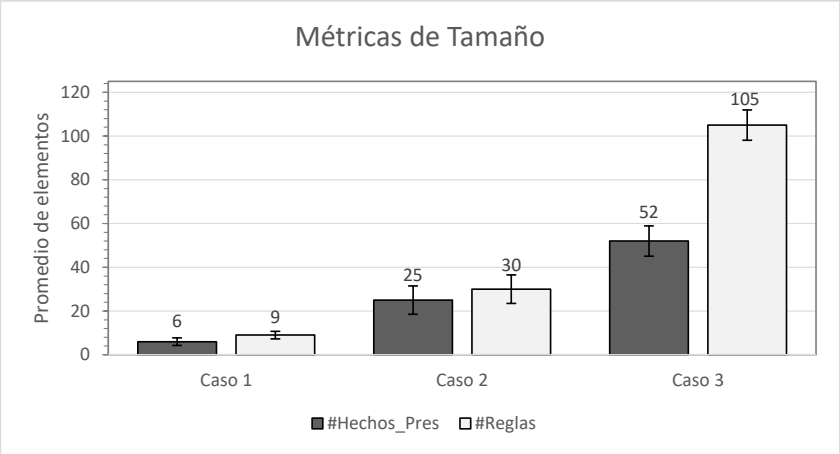


Figura 4.3: Parámetros para los tres escenarios (arriba). Métricas de tamaño para los tres escenarios (abajo).

A partir de estos resultados, podemos verificar que la variación de los parámetros nos permite generar programas con diferentes tamaños y, por lo tanto, poder generar un conjunto de programas con métricas de tamaño que sean de interés para ejecutar diferentes experimentos.

Variación de la complejidad de los modelos: Con respecto a las métricas de complejidad, es decir, las métricas $MDDL, h, t$ y τ, también se generaron tres conjuntos de programas y se computaron los valores de sus métricas. Además, se calculó el tiempo promedio para consultar un literal en los programas correspondientes a cada conjunto generado. El objetivo de estas pruebas, al igual que en las pruebas anteriores, es hacer crecer la complejidad de los modelos en cada escenario con respecto al anterior.

Para estas pruebas, los valores de los parámetros probabilísticos son constantes en los tres escenarios, ya que lo que nos interesa aquí es la variación de la complejidad de los modelos, la cual según el Cuadro 4.2 depende principalmente de los parámetros

Parámetro	Caso 1	Caso 2	Caso 3
BASE_SIZE	5	5	10
FACT_PROB	0.5	0.5	0.5
NEG_PROB	0.3	0.3	0.3
DRULE_PROB	0.8	0.8	0.8
MAX_RULESPERHEAD	1	1	1
MAX_BODYSIZE	2	3	4
MIN_ARGSLEVEL	4	4	4
LEVELS	2	3	3
RAMIFICATION	2	3	2
TREE_HEIGHT	2	2	3

Cuadro 4.3: Valores de parámetros para tres escenarios distintos.

MAX_BODYSIZE (esto nos dará un mayor valor en *MDDL*), MIN_ARGSLEVEL (impacta sobre el número de árboles de dialéctica, es decir, τ), RAMIFICATION (impacta sobre el número de líneas de argumentación, es decir, t) y TREE_HEIGHT (influye sobre la longitud de las líneas de argumentación, es decir, h). El valor de los parámetros para cada uno de los tres escenarios puede verse en el Cuadro 4.3.

Los resultados de estas pruebas pueden verse en la Figura 4.4. Aquí, podemos ver la variación de todas las métricas de complejidad para cada escenario, en particular aquellas que se refieren a los árboles de dialéctica (*MDDL*, h, t y τ). A su vez, en la misma figura, presentamos una comparativa de los tiempos promedio en segundos para consultar un literal de los programas de cada escenario, donde podemos ver cómo, conforme aumenta la complejidad del programa, también aumenta el tiempo para dar respuesta a una consulta en el mismo. De esta manera, podemos determinar que el conjunto de programas generados en el tercer caso es el más complejo, ya que sus programas generan muchas estructuras grandes y complejas y, por lo tanto, requiere mayor tiempo para computarlas y consultarlas. Todos los modelos generados están disponibles para su acceso en un repositorio público[2].

A partir de los resultados de estas pruebas, podemos verificar que el generador nos permite crear programas PreDeLP de diferente tamaño y complejidad. Usaremos este ge-

[2]https://github.com/marioa-l/DeLP-Gen

Casos	*MDDL*	*h*	*t*	*τ*	Tiempo (seg.)
Caso 1	3.77	2.80	1.35	12.73	0.09
Caso 2	4.90	4.27	2.65	16.40	1.54
Caso 3	6.04	5.06	4.85	24.13	2.21

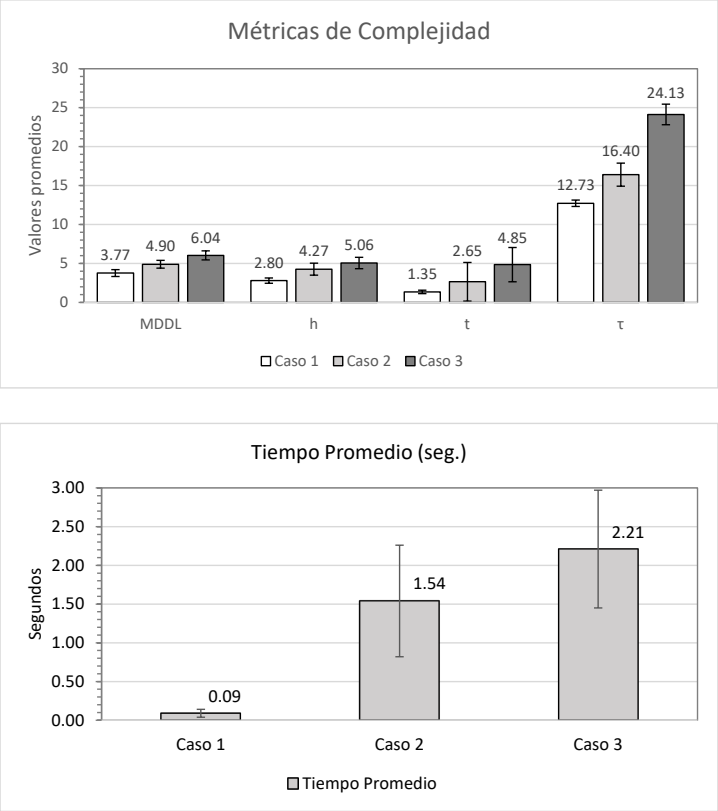


Figura 4.4: Variación de las métricas de complejidad en los tres escenarios.

nerador para crear modelos $\mathcal{P}_{MA}$ para luego usarlos en la generación de modelos DeLP3E. A continuación, detallaremos brevemente cómo generamos los otros dos componentes de un modelo DeLP3E, es decir, el modelo del entorno (solo vamos a generar un tipo de modelo probabilístico gráfico, en particular, redes bayesianas) y la función de anotación.

4.3. Generador de Redes Bayesianas (GRB)

Como se mencionó en el capítulo anterior, se utilizarán redes bayesianas para representar el conocimiento probabilístico del modelo del entorno. En esta sección, primero se

recordará la definición de estos modelos y sus componentes principales; luego, se presentará la definición de parámetros que guiarán la generación automática de estos modelos y finalmente, se detallarán los algoritmos que constituyen el generador automático de redes bayesianas (GRB). Además, también se mostrarán algunos ejemplos de conjuntos de modelos creados con el generador, donde cada uno de estos conjuntos será creado de tal manera que el valor de sus métricas de tamaño y complejidad sean diferentes y significativas.

Recordemos que una red bayesiana es un *grafo dirigido acíclico* [Pea88] donde:

- cada *nodo* representa una *variable aleatoria discreta*;
- si existe un *arco* entre el nodo X y el nodo Y, decimos que X es un *padre* de Y, y representa la *dependencia directa* entre X e Y;
- a cada nodo se le asigna una *distribución probabilística condicional* $P(X_i \mid Padres(X_i))$ que cuantifica el efecto de los nodos padre sobre la variable X_i;
- cada variable es *independiente* de sus no descendientes en el grafo, dado el estado de sus padres;
- la ausencia de un arco entre dos nodos expresa la ausencia de influencias causales entre las variables correspondientes, y la independencia probabilística (posiblemente condicional) entre ellas.

Una red bayesiana describe completamente una distribución, es decir, se sabe la probabilidad de cualquier conjunción de asignaciones de valores a cada variable: $P(X_1 = x_1 \wedge X_1 = x_2 \wedge \ldots X_n = x_n) = P(x_1, x_2, \ldots, x_n)$. Cada entrada en la distribución completa puede ser calculada a partir de la información presente en la red:

$$P(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | padres(X_i))$$

donde $padres(X_i)$ denota los valores de las variables en $Padres(X_i)$. Uno de los problemas más comunes a resolver dada una red bayesiana es computar la probabilidad de que un subconjunto de variables tengan valores dados (inferencia). Es a través de esta herramienta que podemos consultar la probabilidad de un mundo en particular en nuestro modelo del entorno.

Luego de esta breve introducción, continuamos detallando el generador de redes bayesianas. Los componentes principales a generar son los nodos, es decir, las variables aleatorias discretas, y las relaciones entre dichas variables. Luego, se crearán las *tablas de probabilidad condicional* para cada variable del modelo, lo que describirá la distribución de probabilidad del modelo generado. Los valores de las entradas de las tablas de probabilidad condicional serán asignados de tal manera que podamos producir diferentes valores de *entropía* para la distribución de probabilidad codificada en la red generada. A continuación, se detallará la creación de cada uno de estos componentes.

4.3.1. Diseño

Comenzamos detallando el proceso de creación de las redes bayesianas a partir de la creación de un *grafo dirigido acíclico* generado aleatoriamente. Recordemos que un grafo dirigido acíclico o DAG (Directed Acyclic Graph, en Inglés), es un grafo dirigido que no tiene ciclos, es decir, no existe un camino que empiece y termine en un mismo vértice. Para crear un grafo con estas características, utilizamos un algoritmo clásico de la librería `networkx` [HSS08][3], el cual consiste en crear un arco entre dos variables distintas seleccionadas al azar y agregar dicho arco al grafo resultante sólo si este no contiene ciclos; en caso de que el grafo con el nuevo arco genere un ciclo, se quita el arco y se selecciona otro par de variables distintas, y así hasta completar el número de arcos que debe tener el grafo. Una vez que tenemos creada la estructura de la red, creamos y modificamos las entradas de las tablas de probabilidad condicional de cada nodo (variable aleatoria) de la red. Esto último es para controlar el valor de la *entropía* inherente a la distribución de probabilidad codificada en la red bayesiana. Como medida de entropía usamos la definida por Shannon e introducida en [Sha48].

La entropía de Shannon es una medida de la aleatoriedad de una distribución, y es de gran importancia en estadística, teoría de la información y compresión de datos. Conocer la entropía de una fuente aleatoria puede ayudarnos a comprender los datos producidos por dicha fuente. Considerando una distribución discreta de tamaño n, sea $\mathrm{p} = \langle p_1, p_2, \ldots, p_n \rangle$ tal distribución y tal que $p_i \geq 0$, $\sum_{i=1}^{n} p_i = 1$, la entropía de dicha distribución está definida como [BDKR05]:

$$H(\mathrm{p}) = -\sum_{i=1}^{n} p_i \ \log \ p_i$$

[3] https://networkx.org/

Mientras menor sea este valor, menor es la aleatoriedad presente en la distribución de nuestro modelo (es decir, que la masa de probabilidad se distribuye en una menor cantidad de mundos, o que una mayor cantidad de mundos tiene probabilidad cero), lo que significa que muestrear a partir de la distribución de probabilidad será un proceso mucho más guiado que si se ejecuta un muestreo completamente aleatorio. En una red bayesiana con entropía baja, cada muestreo obtenido a partir de la distribución de probabilidad retornará aquellos escenarios que sean los más probables de todas las posibles combinaciones que modela la red. En nuestro caso, si ejecutamos un muestreo por mundos sobre una red con estas características, podemos obtener primero aquellos mundos que tengan mayor valor de probabilidad, y por lo tanto, los que modifiquen significativamente los límites del intervalo de probabilidad aproximado. Al tener acceso a todos los valores de probabilidad de todos los nodos de la red, podemos reducir el valor de la entropía al asignar valores de probabilidad altos a un conjunto de mundos en particular, lo que significa que la masa de probabilidad se distribuye en una menor cantidad de mundos haciendo a la función de distribución de probabilidad menos uniforme. Para asignar un valor a cada entrada de las tablas de probabilidad condicional de cada nodo (la selección de qué nodos modificar se verá más adelante en esta misma sección), primero se selecciona al azar entre los valores *Verdadero* y *Falso*, luego, le asignamos a ese valor una probabilidad en el intervalo $[\alpha, 1)$, donde α es un parámetro con valores en $[0, 1)$. Todo el tratamiento de las tablas de probabilidad condicional lo hacemos a través de una librería especializada en modelos probabilísticos gráficos denominada `pyAgrum`[4]. El procedimiento general para la creación de las redes bayesianas será detallado en las siguientes subsecciones.

4.3.2. Parámetros

Al igual que el GPP, este generador debe poder crear redes bayesianas cuyos valores de métricas podamos ajustar según el valor de sus parámetros. Recordemos que las métricas definidas para el modelo del entorno son: $\#\,Variables$ (número de variables aleatorias en el modelo), $\#PGM\_Arcs$ (número de arcos en el modelo), PGM_TW (medida de qué tan cerca está el modelo de tener una estructura de árbol) y Ent (*entropía*, valor inherente a la función de distribución de probabilidad conjunta del modelo). Por lo tanto, los parámetros a definir deben impactar sobre el valor de las métricas mencionadas. A continuación, se presentan dichos parámetros:

[4]https://pyagrum.readthedocs.io/

- `NODES` (*entero*): Número de variables aleatorias discretas que debe tener la red bayesiana. En nuestro caso, las variables toman valores booleanos, es decir, *Verdadero* o *Falso*.

- `ARCS` (*entero*): Número de arcos que debe tener la red bayesiana. Los arcos son dirigidos y unidireccionales, es decir, sólo puede existir un arco que conecte dos variables del modelo. Por otro lado, los arcos que se agreguen a la red no deben generar ciclos.

- α (*decimal*): Valor de probabilidad mínimo que puede tomar el valor de una variable del modelo. Se usará este valor para definir el intervalo $[\alpha, 1)$ del cual se tomará la probabilidad a asignar al valor de cada variable (*Verdadero* o *Falso*) en las entradas de su tabla de probabilidad condicional. Este parámetro es el que controla el valor de *entropía* de la red.

La relación entre los parámetros definidos y las métricas del ME pueden verse en el Cuadro 4.4. Para este generador en particular, el impacto de los parámetros sobre las métricas del modelo son más directas en comparación con el generador de programas PreDeLP, ya que aquí no tenemos demasiados parámetros y cada uno de ellos indica precisamente el tamaño o complejidad de cada componente que conforman el modelo final. Por ejemplo, el valor de las métricas de tamaño son precisamente el valor de los parámetros `NODES` y `ARCS`; con respecto al valor de PGM_TW, la misma depende de la relación entre el número de nodos y arcos, ya que, por ejemplo, en una red que tiene un mayor número de arcos que de nodos, es más probable que tengamos una estructura diferente a un árbol. En cuanto a la entropía, y como se mencionó previamente, mientras mayor sea el valor de α, menor será el valor de la entropía asociada, ya que se modificará la probabilidad del valor de una variable con un valor tomado del intervalo $[\alpha, 1)$. Esto se detalla en el procedimiento *Adaptar las Tablas de Probabilidad Condicional* de la siguiente subsección.

A partir de estos parámetros, la generación de redes bayesianas consiste brevemente de los siguiente pasos:

1. Crear un grafo dirigio acíclico con tantas variables como indique `NODES` y tantos arcos como indique `ARCS`:

   a) Crear una estructura grafo G

Parámetros del GRB	**Métricas del ME**			
	#Variables	*#PGM_Arcs*	*PGM_TW*	*Ent*
`NODES`	X		X	
`ARCS`		X	X	
α				X

Cuadro 4.4: Relación entre los parámetros del GRB y las métricas del ME.

b) Agregar `NODES` variables a G

c) Mientras existan arcos para agregar a G (`ARCS` es distinto de 0):

1) Seleccionar dos variables X e Y distintas del grafo y agregar el arco (X, Y) a G

2) Si el grafo G resultante es acíclico, restar 1 a `ARCS`; si no, remover el último arco agregado

2. Para cada nodo del grafo G:

a) Tomar todos los padres del nodo y generar todas las posibles combinaciones de valores para esos nodos (estas combinaciones serán las entradas de las TPC)

b) Por cada posible combinación de valores de sus nodos padres:

1) Tomar un valor de probabilidad p aleatorio del intervalo $[\alpha, 1)$

2) Elegir $Verdadero$ o $Falso$

3) Crear la entrada de la TPC del nodo correspondiente al valor de sus padres asignando p al valor elegido en el paso anterior y $1 - p$ al complemento.

De esta manera, podemos establecer una relación entre los parámetros del GRB y las métricas del ME. Las mismas pueden verse en el cuadro 4.4. Al final de esta sección se mostrará, por medio de algunos modelos generados, cómo la variación de los parámetros impactan en el valor de las métricas de dichos modelos. A continuación, se presentarán en detalle los algoritmos para generar las redes bayesianas.

4.3.3. Algoritmos y estructuras generales

Para la generación de una red bayesiana utilizamos dos algoritmos: *Generador de Redes Bayesianas* y *Adaptar CPT*. El primero es el encargado de generar la estructura

de la red, y luego, invocar al segundo para que éste modifique el valor de las tablas de probabilidad condicional de todos los nodos de la red. A continuación, se detallará cada uno de ellos.

Generador de Redes Bayesianas: Este algoritmo es el encargado de crear la estructura de la red, es decir, el grafo dirigido acíclico. Este procedimiento se detalla en el algoritmo 4.11 (`gen_bn`(*nodes*, *arcs*,α)). Se comienza creando una estructura grafo G por medio del constructor de la librería `networkx` (línea 4); luego, agrega al grafo tantos nodos como indique el parámetro *nodes* (línea 5 a 7). Una vez que tenemos todos los nodos en la estructura, se comienza agregando los arcos entre ellos; para esto, se seleccionan dos variables distintas al azar (línea 11 a 15) y se crea el arco que las une, y luego se agrega este arco al grafo (línea 17). Si el grafo resultante no contiene un ciclo, se resta 1 a la cantidad de arcos que debe tener la estructura (línea 20), en caso contrario, se elimina el último arco agregado a la red (línea 23). Este procedimiento de selección de variables, agregar arcos y controlar ciclos, se repite hasta que la red tenga una cantidad de arcos iguales a *arcs* (línea 9). Por último, una vez que tenemos creada la red, se adaptan los valores de las tablas de probabilidad condicional de todos los nodos (línea 28) a través del método `adapt_cpt`(G,α).

Adaptar las Tablas de Probabilidad Condicional: El objetivo de este procedimiento es modificar las entradas de las tablas de probabilidad condicional (TPC) de cada nodo de la red para reducir el valor de la entropía asociada. A continuación, se discuten los detalles para derivar este procedimiento.

Para reducir el valor de la entropía es necesario que la masa probabilística se agrupe en un subconjunto de mundos en particular. Para esto, es necesario definir un valor alto de probabilidad en las variables que describen ese subconjunto de mundos; como tenemos acceso y es posible modificar esos valores, lo que resta es seleccionar los nodos cuyas tablas de probabilidad condicional serán modificadas. Para determinar los nodos que serán modificados, y el valor de probabilidad a usar, se analizaron cinco alternativas:

- C1: Modificar los nodos *sin padres*.
- C2: Modificar los nodos que tengan sólo *dos padres*.
- C3: Modificar los nodos que tengan *más de dos padres*.
- C4: Modificar los nodos que tengan al menos un *nodo hijo*.

Algoritmo 4.11: Crear Red Bayesiana

```
1  def gen_bn(nodes, arcs,α)
       Datos:
       nodes: Nodos de la red bayesiana (variables aleatorias del modelo)
       arcs: Arcos de la red
       α: Valor para regular la entropía de la red
       Resultado: Red Bayesiana con sus tablas de probabilidad adaptadas
2      inicio
3          // se crea una estructura de grafo con networkx
4          G ← networkx.new_graph()
5          para i=1 hasta nodes hacer
6              // agregamos cada nodo a la estructura del grafo
7              G.add_node(i)
8          fin
9          mientras arcs > 0 hacer
10             // seleccionamos dos nodos distintos al azar
11             a ← gen_max(nodes − 1)
12             b ← a
13             mientras b == a hacer
14                 b ← gen_max(nodes − 1)
15             fin
16             // agregamos un arco entre los dos nodos seleccionados
17             G.add_arc(a, b)
18             // se controlar que el grafo resultante no contenga ciclos
19             si networkx.is_dag(G) entonces
20                 arcs ← arcs − 1
21             sino
22                 // en caso de que contenga un ciclo, se elimina el último
                       arco agregado
23                 G.remove_arc(a, b)
24             fin
25         fin
26     fin
27     // se adaptan las tablas de probabilidad condicional de cada nodo
28     adapt_cpt(G,α)
29     devolver G
30 fin
```

α	Nodos	Cantidad de Nodos y Arcos de la RB					
		5		10		15	
		TGA	TA	TGA	TA	TGA	TA
0.70	C1	3.71	2.72	7.14	5.22	10.99	8.80
	C2	3.20	3.16	7.41	6.31	11.43	9.17
	C3	3.63	3.60	7.32	6.71	11.07	10.87
	C4	3.75	2.04	7.19	4.21	10.90	5.86
	C5	3.62	**1.12**	6.98	**2.33**	11.30	**3.76**
0.90	C1	3.43	2.57	7.80	5.23	11.17	7.91
	C2	3.45	3.26	7.35	6.72	11.09	9.69
	C3	3.73	3.61	7.48	6.50	10.95	10.42
	C4	3.62	1.55	6.91	3.45	11.26	5.45
	C5	3.45	**0.64**	7.18	**1.43**	11.32	**2.06**
0.99	C1	3.81	2.56	7.13	4.68	10.72	6.95
	C2	3.57	3.29	7.08	5.88	11.01	9.52
	C3	3.53	3.54	7.53	6.87	11.34	10.02
	C4	3.78	1.40	7.27	2.70	10.93	4.65
	C5	3.47	**0.21**	7.36	**0.30**	11.20	**0.50**

Cuadro 4.5: Reducción de Entropía modificando las TPC de los nodos de la red

- C5: Modificar *todos los nodos de la red*.

Los valores de probabilidad a usar para cada valor posible de los nodos fueron 0,7, 0,9 y 0,99, ya que para valores inferiores a estos, la distribución resultante era uniforme, y por lo tanto, con un mayor valor de entropía; se realizaron las pruebas en redes con 5, 10 y 15 nodos, y se comparó la entropía resultante entre redes con TPC adaptadas (TA, tabla adaptada) y redes con TPC generadas aleatoriamente (TGA, tabla generada aleatoriamente). Para ejecutar estar pruebas, primero se seleccionaron aquellos nodos que cumplan con las condiciones que nos interesaban, y luego se ejecutó una variación del procedimiento 4.12 que sólo considera el conjunto de nodos seleccionados y modifica sus TPC con un valor de α en particular (0,7, 0,95 o 0,99). Los resultados de dichas pruebas pueden verse en el Cuadro 4.5. Como podemos ver, la mejor alternativa es modificar las TPC de todos los nodos de la red (C5); además, cuanto mayor sea el valor de α, menor

valor de entropía tendrá la red resultante.

De esta manera, el procedimiento final para reducir la entropía toma todos los nodos de la red y adapta sus correspondientes TPC con un valor elegido al azar del intervalo $[\alpha, 1]$. Este procedimiento puede verse en el algoritmo 4.12 (`adapt_cpt`(G,α)). Se comienza tomando todos los nodos de la red (línea 4); luego, por cada nodo, se toman todos sus padres (línea 7) y se procede a crear cada una de las entradas de su tabla de probabilidad condicional. Para generar cada entrada de la TPC de un nodo con padres, primero se generan todas las posibles combinaciones de valores de sus nodos padre (línea 10); por ejemplo, si el nodo tiene dos padres $p1$ y $p2$, las combinaciones son $\{(V,V),(V,F),(F,V),(F,F)\}$, siendo cada combinación una entrada de la TPC del nodo. Luego, por cada combinación de valores, se genera un valor de probabilidad al azar del intervalo $[\alpha, 1]$ (línea 13), se calcula su complemento (línea 14), y se selecciona qué valor posible del nodo se va a modificar con la probabilidad generada (línea 16). Si se debe modificar el valor $Verdadero$ de la TPC del nodo, se crea la entrada para la combinación de los valores de los nodos padre y se usa el valor de probabilidad elegido (línea 19); si se debe modificar el valor $Falso$ de la TPC del nodo, se crea la entrada pero se invierten los valores de probabilidad a asignar (línea 21). Por último, se agrega la entrada de la TPC creada a la red (línea 24). En el caso que el nodo seleccionado no tenga padres, simplemente se crea la única entrada de su TPC con los valores de probabilidad seleccionados (línea 27 a 30) y se agrega dicha tabla a la red. El procedimiento $G(node)$.create_cpt($entry$, $entry\_table$) crea la entrada para los valores de los nodos padre especificado por $entry$ (combinación de valores de los nodos padres) y la asignación de probabilidades de $entry\_table$ (un valor de probabilidad para el valor $Verdadero$ y uno para el valor $Falso$). Por otro lado, el procedimiento $G(node)$.create_cpt($entry\_table$) crea la única entrada en la tabla del nodo (sin padres) con la asignación de probabilidades de $entry\_table$.

4.3.4. Ejemplos de Modelo Generados

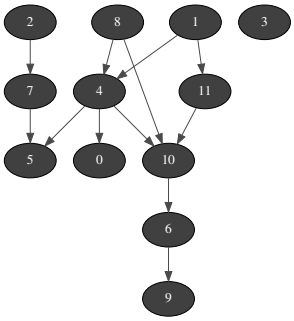
En la Figura 4.5 se muestran 3 ejemplos de modelos creados. Cada red bayesiana se muestra junto con el valor de los parámetros usados para su generación (arriba de la red), y el valor de sus métricas (abajo de la red). Los valores de Ent y PGM_TW se computaron a través de la librería `networkx`. Todos los modelos generados están disponibles para su acceso en un repositorio público[5].

[5]https://github.com/marioa-l/exp-delp3e

Algoritmo 4.12: Adaptar Tablas de Probabilidad Condicional

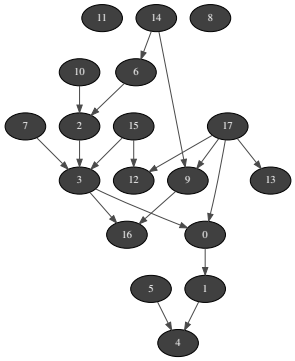
```
1  def adapt_cpt(G, α)
       Datos:
       G: Estructura de la red bayesiana (nodos y arcos)
       α: Valor para definir el intervalo [α, 1].
       Resultado: Red Bayesiana con sus TPC adaptadas
2      inicio
3          // se toman todos los nodos de la red
4          nodes ← get_nodes(G)
5          para cada node de nodes hacer
6              // se toman todos los nodos padres de cada nodo
7              parents ← get_parents(node)
8              si parents es no vacío entonces
9                  // calcular las entradas de la TPC del nodo en base a sus
                       padres
10                 table_entries ← gen_combinations(parents)
11                 para cada entry de table_entries hacer
12                     // generamos el valor de probabilidad usando el α
13                     probability ← gen_random(α,1)
14                     prob_complement ← 1− probability
15                     // seleccionar el valor del nodo a modificar
16                     selected_value ← choice_one(verdadero, falso)
17                     si selected_value = verdadero entonces
18                         // se crea el valor de la entrada de la tabla
19                         entry_table ← {True : probability, False : prob_complement}
20                     sino
21                         entry_table ← {True : prob_complement, False : probability}
22                     fin
23                     // se agrega la entrada entry en la TPC del nodo
24                     G(node).create_cpt(entry, entry_table)
25                 fin
26             sino
27                 // el nodo seleccionado no tiene nodos padres
28                 // se repiten los pasos 12 a 22
29                 // se agrega la única entrada en la TPC del nodo
30                 G(node).create_cpt(entry_table)
31             fin
32         fin
33     fin
34 fin
```

Parámetros	
NODES	12
ARCS	12
α	0.97



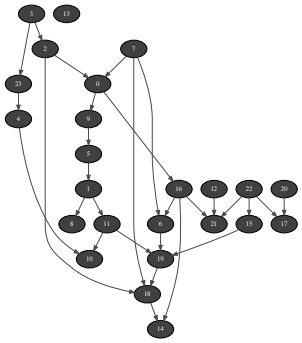
Métricas	
$\#Variables$	12
$\#PGM\_Arcs$	12
PGM_TW	2
Ent	0.9

Parámetros	
NODES	18
ARCS	18
α	0.75



Métricas	
$\#Variables$	18
$\#PGM\_Arcs$	18
PGM_TW	2.18
Ent	0.6

Parámetros	
NODES	24
ARCS	30
α	0.50



Métricas	
$\#Variables$	24
$\#PGM\_Arcs$	30
PGM_TW	2.83
Ent	0.5

Figura 4.5: Ejemplo de modelos ME usando redes bayesianas

4.4. Generador de Funciones de Anotación (GFA)

Una vez que tenemos definidos y desarrollados los generadores para los modelos analíticos y del entorno, el último componente a generar, y que vincula los otros dos modelos, es la *función de anotación*. Recordemos que el formalismo DeLP3E combina un modelo de entorno y uno analítico ($\mathcal{P}_{ME}$ y $\mathcal{P}_{MA}$, respectivamente). Intuitivamente, dado el $\mathcal{P}_{MA}$, cada elemento de $(\Theta, \Omega, \Phi, \Delta)$ sólo se aplica en ciertos mundos del conjunto $\mathcal{W}_{ME}$. Esta vinculación entre $\mathcal{P}_{MA}$ y $\mathcal{P}_{ME}$ se obtiene a partir de una asociación entre los elementos del modelo analítico (reglas, hechos y presunciones) y fórmulas lógicas construidas a partir de las variables del modelo del entorno (usando conjunción, disyunción, y negación). De esta manera, aquellos elementos del $\mathcal{P}_{MA}$ que son verdaderos o valen en un mundo en particular del $\mathcal{P}_{ME}$, son aquellos cuyas fórmulas lógicas son verdaderas a partir de los valores que describen el mundo en particular.

En esta sección se definirá y presentará un generador de funciones de anotación a partir del modelo analítico (MA) y del modelo del entorno (ME). Este generador tomará un subconjunto de los elementos del MA que reciba como entrada, y le asignará a cada uno de ellos una fórmula lógica construida usando un conjunto de conectores y variables del ME. Los elementos del MA y los conectores y variables del ME que serán usados se detallarán en la próxima subsección. De esta manera, con el desarrollo de este último generador, tenemos todas las herramientas para generar modelos DeLP3E completos con diferentes tamaños y complejidades.

4.4.1. Diseño

El proceso de creación de la función de anotación comienza seleccionando los componentes del MA que serán anotados con las fórmulas lógicas; aquí, podemos seleccionar según el tipo de conocimiento que modelan (reglas estrictas, reglas rebatibles, hechos o presunciones) o según un porcentaje del programa, es decir, por ejemplo, anotar el 50 % del programa (se seleccionarán los componentes de manera aleatoria). Una vez que tenemos seleccionados los componentes a anotar, se crea una fórmula lógica por cada uno de ellos a partir de un conjunto de variables del ME y un conjunto de conectores. Los conectores pueden ser conjunción ($\wedge$), disyunción ($\vee$) y negación ($\neg$). El procedimiento para generar las anotaciones permite usar múltiples variables del ME y operadores lógicos para crear cada fórmula, en nuestro caso en particular, usaremos un máximo de 3 variables, ya que con este número podemos complejizar las anotaciones de una manera significativa para el experimento a ejecutar; sin embargo, como se detalla en las próxima subsecciones, el procedimiento para generar las anotaciones puede seleccionar un conjunto mayor de variables. Por último, se asigna a cada componente del modelo analítico su correspondiente anotación (fórmula lógica). De esta manera, vinculamos el MA con el ME a partir de fórmulas lógicas.

4.4.2. Parámetros

Al igual que los generadores GPP y GRB, este generador debe poder crear funciones de anotación cuyos valores de métricas podamos ajustar según el valor de sus parámetros. Recordemos que las métricas definidas para las funciones de anotación son: $\%FA\_An$, que se refiere al porcentaje de los componentes del MA que están anotados, y FA_Com que se

Parámetros del GFA	**Métricas de la *fa***	
	$\%FA\_An$	FA_Com
P_FA	X	X
EM_VARS		X

Cuadro 4.6: Relación entre los parámetros del GFA y las métricas de *fa*.

refiere a la complejidad de cada anotación en sí (es decir, el número y tipo de operadores que se usan, si se permite la combinación de operadores, etc). Así, el generador debe poder crear funciones que anoten diferentes porcentajes del programa y cada fórmula puede involucrar múltiples variables y conectores. Por lo tanto, los parámetros a definir deben impactar sobre el valor de las métricas mencionadas. A continuación, se presentan dichos parámetros.

- P_FA (*entero*): Porcentaje del programa a ser anotado. Si este valor es igual a 0, sólo se anotan los hechos del programa.
- EM_VARS (*entero*): Número de variables del ME a usar en cada fórmula. De este parámetro también depende el número de operadores a usar. En cada fórmula está permitido usar cualquiera de los tres operadores mencionados anteriormente ($\wedge, \vee, \neg$).

La relación entre los parámetros definidos y las métricas de una función de anotación pueden verse en el Cuadro 4.6. Aquí, los parámetros impactan de forman directa sobre el valor de las métricas, ya que P_FA es exactamente el porcentaje del programa que queremos anotar. Por otro lado, EM_VARS indica el número de variables a usar, y por lo tanto también influye sobre el número de operadores a usar, lo que indica que a mayor valor de este parámetro y el porcentaje de anotación, mayor complejidad tendrá la función de anotación.

A partir de estos parámetros, la generación de la función de anotación consiste brevemente de los siguientes pasos:

1. Tomar todos los elementos del MA y ME recibidos como entrada
2. Si P_FA es igual a 0, filtrar sólo los hechos del MA; en caso contrario, filtrar el porcentaje indicado por P_FA de todos los elementos del MA (seleccionando al azar)

3. Por cada elemento del MA filtrado, crear una fórmula lógica:

   a) Mientras `EM_VARS` sea distinto de 1:
   - Seleccionar una variable v_i del ME que no se haya preseleccionado.
   - Determinar si se niega o no la variable v_i.
   - Seleccionar un operador op_i del conjunto $\{\wedge, \vee, \neg\}$.
   - Expandir la fórmula lógica en creación con la expresión v_i op_i.
   - Restar 1 a `EM_VARS`.

   b) Seleccionar una variable v_i del ME que no se haya preseleccionado.

   c) Determinar si se niega o no la variable v_i.

   d) Expandir la fórmula lógica en creación con la expresión v_i.

   e) Asociar la fórmula lógica creada con el elemento filtrado del MA.

Al final de esta subsección se mostrará, por medio de algunos ejemplos generados, cómo la variación de estos parámetros impactan directamente sobre las métricas de las funciones de anotación generadas. A continuación, se presentará en detalle los algoritmos del GFA.

4.4.3. Algoritmos y estructuras generales

El procedimiento general para generar una función de anotación, es decir, anotar un modelo MA con fórmulas lógicas formadas con variables de un modelo ME, se detalla en el Algoritmo 4.13 `gen_fa`($\mathcal{P}_{MA}$, $\mathcal{P}_{ME}$, *P_FA*, *EM_VARS*). Este procedimiento comienza tomando todos los hechos, reglas y presunciones del MA recibido (línea 4), y todas las variables que forman parte de la red bayesiana del ME (línea 5). Luego, si el porcentaje del programa a anotar es igual a 0, se toman solo los hechos del programa (línea 8); en caso contrario, se toma un porcentaje igual a *P_FA* de los elementos del programa (línea 11). Estos elementos filtrados serán los que se anoten con las fórmulas generadas, es decir, son los elementos del MA que estarán sujetos a eventos probabilísticos. Por cada elemento filtrado, se genera una fórmula para ser asociada a dicho elemento (línea 15); para esto, se controla el valor del parámetro `EM_VARS`. Mientras `EM_VARS` sea distinto de 1, se selecciona una variable que no se haya usado para construir la anotación (línea 17), se determina si se niega o no (línea 19), se selecciona un operador (línea 20), y se expande

$af(\theta_1) = var_1$	$af(\theta_1) = var_1 \vee var_{12}$	$af(\theta_1) = \neg var_1 \wedge var_4 \vee var_9$
$af(\theta_2) = var_2$	$af(\theta_2) = verdadero$	$af(\theta_2) = var_5 \vee \neg var_7 \wedge var_1$
$af(\theta_3) = var_3$	$af(\theta_3) = \neg var_3 \wedge var_4$	$af(\theta_3) = var_2 \vee var_5 \wedge \neg var_{10}$
$af(\omega_1) = verdadero$	$af(\omega_1) = verdadero$	$af(\omega_1) = var_1 \vee \neg var_3 \vee var_2$
$af(\omega_2) = verdadero$	$af(\omega_2) = verdadero$	$af(\omega_2) = \neg var_{12} \vee var_1 \vee \neg var_6$
$af(\phi_1) = verdadero$	$af(\phi_1) = var_2$	$af(\phi_1) = \neg var_2 \wedge \neg var_7 \vee var_{16}$
$af(\phi_2) = verdadero$	$af(\phi_2) = var_{12}$	$af(\phi_2) = var_2 \wedge var_{11} \wedge \neg var_1$
$af(\delta_1) = verdadero$	$af(\delta_1) = verdadero$	$af(\delta_1) = var_5 \wedge var_{11} \vee var_{10}$
$af(\delta_2) = verdadero$	$af(\delta_2) = verdadero$	$af(\delta_2) = var_{19} \vee \neg var_9 \vee var_7$
$af(\delta_3) = verdadero$	$af(\delta_3) = var_5 \vee \neg var_2$	$af(\delta_3) = \neg var_{20} \wedge \neg var_1 \vee var_3$

Figura 4.6: Ejemplo de funciones de anotación generadas con el GFA.

la anotación con la variable y el operador seleccionado (línea 22). Por último, se agrega la última variable para terminar de formar la anotación y asociarla al elemento del programa (líneas 25 a 31). Es importante remarcar que aquellos elementos del MA que no hayan sido seleccionados para ser anotados, se anotan con la expresión *verdadero*, lo que implica que se mantienen en todos los mundos posibles del ME.

4.4.4. Ejemplos de modelos generados

En la Figura 4.6 se muestran tres modelos de funciones de anotación generadas con el GFA. Para este ejemplo, se usaron los elementos (Θ, Ω, Φ, Δ) del $\mathcal{P}_{MA}$ introducido en el Capítulo 2 y una red bayesiana con 20 variables ($var_1, var_2, \ldots, var_{20}$). Para el primer caso, se usaron los parámetros `P_FA` $= 0$ y `EM_VARS` $= 1$, es decir, solo se anotan los hechos del programa y se usa solo una variable como fórmula lógica, lo que produce la función de anotación que se muestra en la izquierda de la figura. Para la función de anotación del centro de la figura, se usaron los parámetros `P_FA` $= 50$ y `EM_VARS` $= 2$, es decir, se anota el 50 % del programa y se usan dos variables y un operador en cada fórmula lógica. Para el último caso, la función de anotación en la derecha de la figura, se usaron los parámetros `P_FA` $= 100$ y `EM_VARS` $= 3$, lo que genera anotaciones para el 100 % del programa y cada fórmula está formada por 3 variables y dos conectores. Cada una de estas funciones tiene una complejidad diferente, ya que al momento de verificar los elementos del MA que serán verdaderos en un mundo en particular, se debe verificar la validez de un número diferente de fórmulas y de complejidad variada (múltiples variables y conectores).

Algoritmo 4.13: Generar Funciones de Anotación

```
1  def gen_fa(P_MA, P_ME, P_FA, EM_VARS)
       Datos:
       P_MA: Modelo analítico, es decir, el conjunto de reglas, hechos y presunciones
       P_ME: Modelo del entorno, en nuestro caso, la red bayesiana
       P_FA: Porcentaje del modelo analítico a anotar
       EM_VARS: Número de variables del modelo del entorno a usar en cada anotación
       Resultado: P_MA anotado con fórmulas lógicas construidas con variables del P_ME
2      inicio
3          // se toman todos los elementos del P_MA y todas las variables del P_ME
4          am_elements ← reglas, hechos y presunciones del P_MA
5          em_variables ← todos los nodos de la red bayesiana del P_ME
6          si P_FA = 0 entonces
7              // solo se toman los hechos en am_elements
8              filtered_elems ← hechos en am_elements
9          sino
10             // se filtra un porcentaje de los elementos en am_elements
11             filtered_elems ← seleccionar al azar un porcentaje igual a P_FA de elementos de am_elements
12         fin
13         para cada element en filtered_elems hacer
14             // Se crea la anotación que será la fórmula lógica a asociar
15             annotation ←" "
16             mientras EM_VARS ≠ 1 hacer
17                 v_i ← Elegir una variable al azar del ME que no se haya preseleccionado
18                 // Determinar si negar o no la variables
19                 v_i ← v_i | ¬v_i
20                 op_i ← Elegir un operador al azar de {∧, ∨, ¬}
21                 // Expandir la anotación
22                 annotation ← annotation v_i op_i
23                 EM_VARS − 1
24             fin
25             v_i ← Elegir una variable al azar del ME que no se haya preseleccionado
26             // Determinar si negar o no la variables
27             v_i ← v_i | ¬v_i
28             // Expandir la anotación
29             annotation ← annotationv_i
30             // anotamos el elemento filtrado con la fórmula generada
31             element ← element : annotation
32         fin
33     fin
34 fin
```

4.5. Modelos Generados

Los generadores fueron creados para tener una herramienta que nos permita crear un conjunto de modelos DeLP3E de diferentes tamaños y complejidades. El objetivo final es poder crear modelos que puedan clasificarse según el valor de sus métricas. Particularmente, en esta tesis usaremos un esquema sencillo que clasifique a cada modelo en "simple", "medio" o "complejo". De esta manera, podemos crear modelos que se adapten a las líneas del árbol de decisión del capítulo 3, y así, poder ejecutar los algoritmos de aproximación para diferentes configuraciones y analizar los tiempos, calidad de la métrica de aproximación, y cantidad de mundos consultados con el fin de determinar la mejor opción para aproximar el intervalo de probabilidad exacto. Esto se detallará con más profundidad en el próximo capítulo.

Para poder crear modelos según el esquema X_{MA}, X_{ME}, X_{FA} donde cada X_{Modelo} puede ser S (simple), M (medio) o C (complejo), se definieron valores para cada conjunto de parámetros correspondientes a cada generador. Así, contamos con los valores de los parámetros que nos generará, por ejemplo, un modelo analítico de complejidad "simple", o una red bayesiana de complejidad "media". Los valores de las métricas para cada modelo que generan esos parámetros definidos son las que nos permiten clasificar dichos modelos, estos valores fueron buscados a través de la variación de los parámetros ya que son los que consideramos significativos y nos permiten ejecutar experimentos en un tiempo razonable. Dichos valores para los parámetros fueron mostrados en las subsecciones correspondientes a los ejemplos de modelos generados de cada generador, y la clasificación de complejidad según las métricas obtenidas pueden verse en la Figura 4.7. Estos son los modelos que fueron usados en las pruebas empíricas que se detallan en el próximo capítulo. Podemos ver que se presenta un caso particular en la generación de las funciones de anotación, ya que la clasificación de su complejidad la determina directamente el valor de los parámetros; esto es así porque hay una relación directa entre el valor de los parámetros y sus métricas. A partir de estos valores, podemos generar modelos DeLP3E completos según el esquema X_{MA}, X_{ME}, X_{FA}, con lo cual podríamos generar un total de 27 combinaciones. Es importante remarcar de nuevo que estos valores fueron seleccionados de acuerdo a nuestras necesidades, limitaciones y objetivos, y que los algoritmos pueden ser utilizados para generar modelos que estén fuera de estos grupos identificados para los propósitos de esta Tesis. Todos los modelos generados están disponibles en un repositorio público[6].

[6]https://github.com/marioa-l/exp-delp3e

Métricas del MA

Complejidad	*#Reglas* + *#Hechos_Pres*	*MDDL*	*h*	*t*	τ	Tiempo (seg.)
Simple	15	3.77	2.80	1.35	12.73	0.09
Medio	55	4.90	4.27	2.65	16.40	1.54
Complejo	157	6.04	5.06	4.85	24.13	2.21

Métricas del ME

Complejidad	*#Variables*	*#PGM_Arcs*	*PGM_TW*	*Ent*
Simple	12	12	2	0.9
Medio	18	18	2.18	0.6
Complejo	24	30	2.83	0.5

Métricas de la *fa*

Complejidad	P_FA	EM_VARS
Simple	0	1
Medio	50	2
Complejo	100	3

Figura 4.7: Métricas de los modelos clasificados según complejidad.

4.6. Sumario

En este capítulo se presentó el diseño, análisis, y ejecución de tres generadores, cada uno orientado a crear cada modelo de una base de conocimiento DeLP3E. Dichos generadores nos permiten crear modelos con valores de métricas que podemos regular a través de sus parámetros, con lo cual podemos crear bases de conocimiento DeLP3E de complejidad variada y así crear diferentes escenarios para experimentar con los algoritmos de aproximación propuestos en el capítulo anterior. Por último, se presentó un conjunto de modelos clasificados según el valor de sus métricas, los cuales serán utilizados en el experimento a detallar en el próximo capítulo.

Capítulo 5

Evaluación Experimental

En este capítulo se presentará el diseño, ejecución, y análisis de los resultados de pruebas empíricas de los algoritmos de aproximación presentados sobre modelos DeLP3E de diferentes caracterśticas. Dichos algoritmos son los presentados en el Capítulo 3, y los modelos DeLP3E fueron generados a través de las herramientas presentadas en el Capítulo 4. El objetivo final de este capítulo es comparar los dos algoritmos de aproximación, el basado en mundos y el basado en subprogramas, sobre diferentes modelos DeLP3E para mostrar su comportamiento con respecto a diferentes criterios. Con esto, pretendemos validar y explorar alternativas sobre las diferentes líneas del árbol de decisión presentado al final del Capítulo 3 (ver Figura 3.7). Este capítulo se organiza de la siguiente manera: en la Sección 5.1 se detalla el diseño del experimento, se describe como se llevarán a cabo las pruebas y qué se evaluará en cada una de ellas; en la Subsección 5.1.1 se describe como se construyó el conjunto de pruebas y como está formado el mismo, y en la Subsección 5.1.2 se listan y detallan las métricas a evaluar para cada algoritmo en los escenarios de prueba; en la Sección 5.2 se mostrarán los resultados y el anális de los mismos y, finalmente, en la Sección 5.3 se presenta un resumen de lo detallado en este capítulo.

5.1. Diseño del Experimento

El experimento consiste en comparar los algoritmos de aproximación en diferentes escenarios según la métrica de calidad y el tiempo promedio requerido para computar el intervalo de probabilidad aproximado de un conjunto de literales. A su vez, como los

algoritmos deben consultar la probabilidad de un subconjunto de mundos, también se comparan según el número de consultas realizadas a la red bayesiana (el tiempo requerido para estas consultas está considerado como parte del tiempo promedio para computar la aproximación).

Para realizar las pruebas empíricas, primero se definió un conjunto de escenarios o configuraciones, cada una de las cuales modela un escenario DeLP3E diferente y puede ser mapeada a alguna de las líneas o ramas del árbol de decisión presentado en el Capítulo 3. Luego, sobre cada una de estas configuraciones, se ejecutan los algoritmos de aproximación para computar el intervalo de probabilidad aproximado de un conjunto de literales y comparar dichos intervalos con sus correspondientes valores exactos. Para analizar el desempeño de cada algoritmo, se comparan tres valores de salida para cada uno de ellos en cada configuración, los mismos se consideran las *métricas de desempeño* del algoritmo en dicha configuración. Los valores a analizar para cada algoritmo son: (i) *Métrica de Calidad*, que indica que tan buena es la aproximación obtenida; (ii) *Tiempo*, que indica el tiempo requerido para obtener la aproximación; y por último, (iii) *Consultas a la Red Bayesiana*, que determina el porcentaje de mundos que fueron consultados a la red bayesiana para obtener su valor de probabilidad. En base a estos resultados, se determinará el algoritmo que mejor se adapte teniendo en cuenta las condiciones del escenario de prueba.

La Figura 5.1 muestra la vista general del diseño del experimento. Primero, se crea el conjunto de pruebas formado por escenarios que describen modelos DeLP3E con diferentes estructuras usando los generadores GPP, GRB y GFA, es decir, escenarios que podemos clasificar según el esquema X_{MA}, X_{FA}, X_{ME} (programas PreDeLP, anotaciones y redes bayesianas de diferentes tamaños y complejidades). Luego, se computan los intervalos de probabilidad exactos de tres literales del MA de cada escenario (estos representan las consultas a realizar), dichos literales son elegidos según el nivel del programa PreDeLP al que pertenecen: se selecciona un literal L_1 del nivel 1, un literal L_2 de algún nivel intermedio, y un literal L_3 del último nivel del PreDeLP. Una vez que tenemos los valores exactos a aproximar, es decir, los intervalos $P_r(L_1) = [l_1, u_1]$, $P_r(L_2) = [l_2, u_2]$ y $P_r(L_3) = [l_3, u_3]$, se ejecutan los algoritmos de aproximación muestreando diferentes porcentajes de los mundos posible ($\mathcal{W}_{ME}$) o programas posibles ($\mathcal{W}_{MA}$) según el enfoque basado en mundos o el basado en subprogramas respectivamente. Cada algoritmo computará un intervalo de probabilidad aproximado para cada uno de los tres literales ($[l'_i, u'_i]$), luego, se computarán las métricas de desempeño: métrica de calidad de la aproximación, tiempo

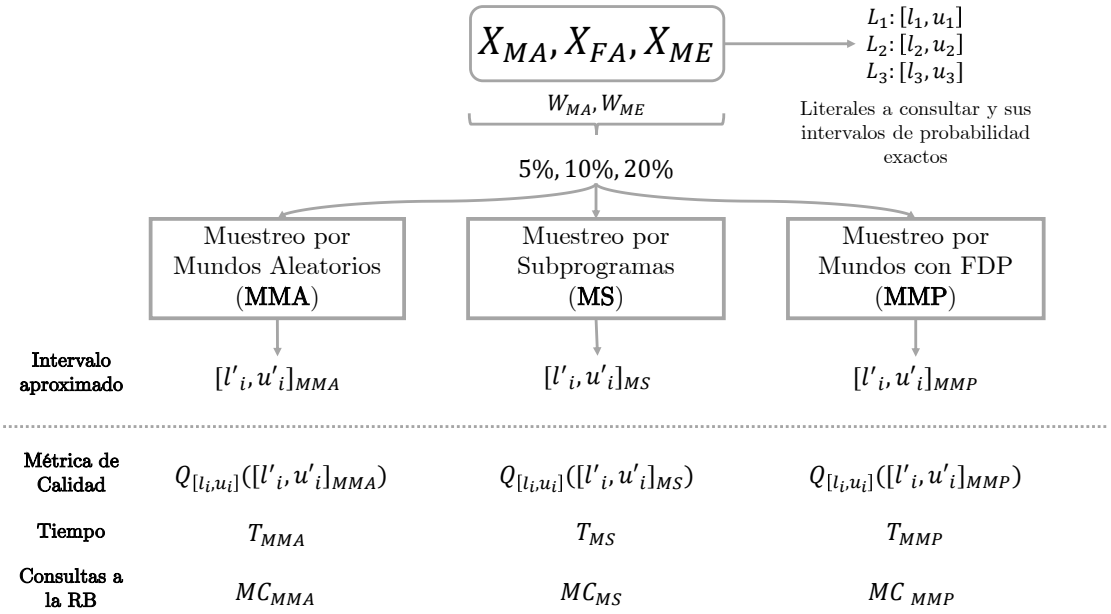


Figura 5.1: Diseño del experimento.

transcurrido, y el porcentaje de mundos que se consultan a la red bayesiana. Por último, se analizarán esos resultados para determinar el algoritmo que mejor se adapte al escenario de prueba X_{MA}, X_{FA}, X_{ME}.

5.1.1. Construcción del Conjunto de Pruebas

Para crear el conjunto de pruebas, se usaron los generadores y el conjunto de parámetros presentados en el capítulo anterior. Con esto, tenemos la capacidad de generar modelos $\mathcal{P}_{MA}$, $\mathcal{P}_{ME}$, y fa de diferentes tamaños y complejidades. Estas configuraciones modelan escenarios que, según la complejidad de cada modelo, pueden anotarse como X_{MA}, X_{FA}, X_{ME}, tal como se detallan en la Sección 4.5 del Capítulo 4. En la Figura 4.7 se muestran los valores de las métricas de cada modelo según la clasificación de complejidad. De esta manera, por ejemplo, el escenario *CMC* representa un modelo DeLP3E generado automáticamente compuesto por: un modelo analítico representado por un programa PreDeLP con 24 árboles de dialéctica en promedio y un tiempo de respuesta para conocer el estado de alguno de sus literales de 2.21 segundos en promedio, una función de anotación que asocia el 50 % del programa con fórmulas lógicas formadas con dos variables del modelo del entorno, y un modelo del entorno representado por una red bayesiana con

24 variables y un valor de entropía igual a 0.5. Los valores de las métricas que determinan el tamaño y complejidad de cada componente (por ejemplo, número y tamaño de los árboles de dialéctica o MDDL), fueron buscados a través de la variación de los parámetros de cada generador, ya que son los que consideramos significativos y nos permiten ejecutar experimentos en un tiempo razonable.

A partir del esquema X_{MA}, X_{FA}, X_{ME}, se pueden generar 27 configuraciones (considerando los casos en donde los 3 modelos son de igual complejidad); sin embargo, no todas se corresponden con alguna línea del árbol de decisión a analizar. Por esto, se decidió analizar sólo un subconjunto de las configuraciones posibles, aquellas que son las más representativas de una línea del árbol y las que mejor se diferencian entre sí, por ejemplo, las configuraciones *SSM* y *MSS*. A partir de las configuraciones seleccionadas, el conjunto de pruebas queda conformado por 14 escenarios. El Cuadro 5.1 detalla las características de cada uno de ellos, donde por cada escenario se muestra:

- **Mundos**: Número de mundos del ME.
- **Programas**: Número promedio de subprogramas únicos en el MA.
- **TCM**: Tiempo promedio en segundos para consultar por la probabilidad de un mundo, generar el subprograma correspondiente, y consultar por el estado de la consulta.
- **TE**: Tiempo promedio en horas para computar el intervalo exacto por medio del procedimiento de fuerza bruta.

De esta manera, a su vez, es posible corresponder cada escenario con alguna línea del árbol de decisión, por ejemplo, el escenario 9 cumple con todas las condiciones de la línea 13: se clasifica X_{FA} como C, lo que quiere decir que la complejidad de las anotaciones no es simple (3 variables y 2 operadores) y que el porcentaje de anotación es alto (el 100 %), el modelo analítico X_{MA} es S (simple), y el modelo del entorno X_{ME} es C, es decir, se cumple la condición $|MA| < |ME|$. La correspondencia entre escenario y línea del árbol final puede verse en la Figura 5.2.

A partir de los datos que se muestran en el Cuadro 5.1, es posible analizar la estructura de cada escenario según su complejidad. Por ejemplo, si comparamos los escenarios 4 y 5 (*SMS* y *SCS* resp.), podemos apreciar el impacto de la función de anotación en los

Id	Escenarios			Mundos	Programas	TCM	TE
	PAM	FA	PEM				
1	S	S	S	2^{12}	400	0.0074	0.008
2	M	S	S	2^{12}	3686	0.0662	0.075
3	C	S	S	2^{12}	3921	0.1048	0.119
4	S	M	S	2^{12}	210	0.0041	0.004
5	S	C	S	2^{12}	2131	0.0371	0.042
6	S	S	M	2^{18}	1548	0.0013	0.096
7	S	S	C	2^{24}	2781	0.0005	2.349
8	S	M	M	2^{18}	518	0.0008	0.060
9	S	C	C	2^{24}	219378	0.0016	7.704
10	S	C	M	2^{18}	30190	0.0136	0.991
11	M	M	C	2^{24}	1430268	0.0210	98.270
12	C	C	S	2^{12}	4095	0.0925	0.105
13	C	M	C	2^{24}	536896	0.023	16.385
14	C	M	M	2^{18}	140879	0.0572	4.166

Cuadro 5.1: Escenarios generados

tiempos TCM y TE, ya que tener anotaciones en el 50 % del programa genera un número menor de programa únicos en comparación a tener el 100 % del programa anotado, y al tener menos programas únicos, aumenta el número de programas repetidos, lo que permite evitar realizar consultas duplicadas. Todos los modelos generados están disponibles para su acceso en un repositorio público[1].

5.1.2. Métricas de Desempeño

Las métricas a computar y analizar para cada algoritmo en cada escenario son tres. Las mismas se detallan a continuación:

- Q (Métrica de Calidad): Indica qué tan buena es la aproximación conseguida a través del muestreo. Se utilizará la métrica de calidad de la Definición 10.

[1] https://github.com/marioa-l/exp-delp3e

- T (Tiempo Promedio): Tiempo promedio en segundos transcurridos desde que se realiza la consulta hasta que se obtiene el intervalo de probabilidad aproximado. Este tiempo considera todo el tiempo requerido para muestrear, consultar la probabilidad asociada, y consultar por el estado del literal, tanto en el muestreo por mundos como en el muestreo por subprogramas.

- MC (Porcentaje de Mundos Consultados): Indica el porcentaje de mundos que se consultan a la red bayesiana para obtener su probabilidad asociada. Recordar que el procedimiento de aproximación basado en mundos itera hasta muestrear un porcentaje del total, y según el método para muestrear (aleatorio o a partir de la función de distribución de probabilidad), pueden existir mundos que se muestrean más de una vez; sin embargo, se lleva un registro de los mundos repetidos, lo que nos evita tener que consultar por la probabilidad de mundos ya muestreados. En el caso del muestreo de subprogramas, es posible realizar una consulta definiendo los valores de un subconjunto de variables del modelo del entorno, lo que retornará la probabilidad total del conjunto de mundos que satisfacen esos valores de las variables definidas. De esta manera, si muestreamos el 20 % de los mundos en un escenario de 4096 mundos (12 variables en el ME), y tenemos una cantidad de mundos únicos igual a 400, esta métrica retornará el valor $9,76$. El mecanismo para realizar las consultas a la red bayesiana lo provee la librería `pyAgrum`.

Una vez detallado todo el diseño del experimento, el conjunto de pruebas, y las métricas a evaluar, en la siguiente sección se listan los resultados obtenidos.

5.2. Resultados: Desempeño de tres algoritmos en 14 escenarios

En esta sección se detallan los resultados obtenidos para cada uno de los escenarios. El entorno en el que se ejecutó el experimento es una máquina virtual con sistema operativo Debian 11, procesador Intel Core i7-4790 de 3.6GHz, y 8GB de memoria RAM.

Los resultados se listan de la siguiente manera:

- Figura 5.3: SSS, MSS y CSS.

- Figura 5.4: CMC, SMS y CMM.
- Figura 5.5: SCS, CCS y MMC.
- Figura 5.6: SMM, SCM y SCC.
- Figura 5.7: SSC y SSM.

Cada fila de una figura muestra los resultados de un escenario en particular. Los resultados se analizan por grupos de configuraciones y valores de métricas, así, por ejemplo, los resultados de la Figura 5.3 muestran que el algoritmo de aproximación MMP arroja los valores más altos para la métrica de calidad, en el menor tiempo, y realizando el menor número de consultas a la red bayesiana en comparación con los otros dos algoritmos. Es importante remarcar que, generalmente, es mejor muestrear mundos a partir de la función de distribución de probabilidad en escenarios con *entropía* baja, ya que con cada muestreo, se obtienen los mundos "más pesados" (o que concentran la mayor masa de probabilidad); además, como se detalla en el Algoritmo 3.1, al llevar un control de los mundos ya muestreados, se evita el recómputo cuando la mayor masa de probabilidad se acumula en un número reducido de mundos. Por otro lado, también es posible realizar un análisis definiendo previamente un *orden de importancia* para el valor de las métricas; de esta manera, si consideramos de mayor importancia la calidad de la aproximación, un algoritmo que realice un mayor número de consultas a la red bayesiana pero que consigue una mejor aproximación, puede ser elegido como la mejor opción frente a un algoritmo con menos consultas a la red bayesiana y una aproximación de baja calidad. La Figura 5.2 muestra el árbol de decisión final (es el mismo árbol que se muestra en la Figura 3.7, se lo presenta de nuevo sólo para facilitar su lectura en este capítulo), con todas sus líneas o ramas numeradas, las configuraciones elegidas para formar el conjunto de prueba que se corresponden con cada línea, y el algoritmo de aproximación que mejor se comporta en dicha configuración según diferentes criterios. Este árbol es el que se analiza y valida por medio de las pruebas empíricas. A continuación, se listarán los escenarios analizados agrupados según la línea del árbol que representan:

- Línea 3 (Fig. 5.3) [SSS,MSS,CSS]: En estos escenarios, el algoritmo MMP es el que mejor se comporta según sus métricas de desempeño. Podemos ver que obtiene la mejor métrica de aproximación, en el menor tiempo posible, y realizando el menor número de consultas a la red bayesiana.

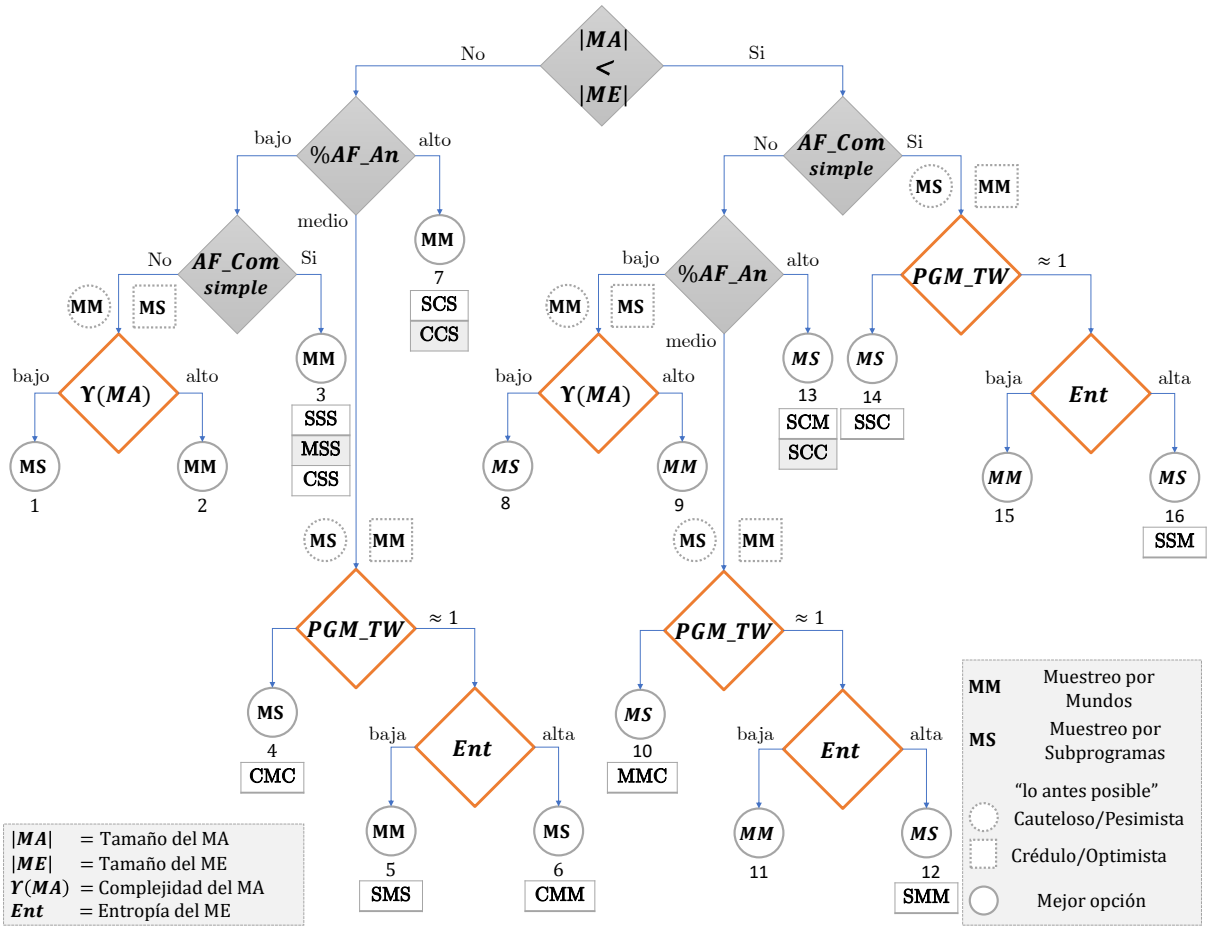


Figura 5.2: Árbol de Decisión y Escenarios

- Línea 4 (Fig. 5.4) [CMC]: En este caso, el algoritmo MS es el que menos consultas realiza a la red bayesiana, obteniendo una métrica de calidad cercana a 0.4 y en un tiempo similar al algoritmo MMA. Como el ME es complejo (C), lo mejor es evitar realizar muchas consultas a la red bayesiana, ya que esto impacta en el tiempo global del algoritmo.

- Línea 5 (Fig. 5.4) [SMS]: Similar a la línea 3, el algoritmo MMP es el que mejor se comporta según sus métricas de desempeño. Obtiene la mejor métrica de aproximación, en el menor tiempo posible, y realizando el menor número de consultas a la red bayesiana.

- Línea 6 (Fig. 5.4) [CMM]: Similar a la línea 4, el algoritmo MS obtiene una métrica de aproximación mejor que MMA y en un tiempo similar. Al tener un ME que no es simple (S), se trata de reducir el número de consultas a la RB.

- Línea 7 (Fig. 5.5) [SCS,CCS]: Al tener un número menor de mundos que subprogramas (y una red bayesiana con baja entropía), MMP es el algoritmo que mejor se comporta según sus tres métricas de desempeño (en los dos escenarios). Teniendo en cuenta la complejidad de la función de anotación (compleja), y como se muestra en el Cuadro 5.1, el número promedio de subprogramas para SCS es 2131 y para CCS es 4095, lo cual impacta en el tiempo requerido por el algoritmo MS; esto puede verse al comparar el tiempo de MS entre SCS y CCS.

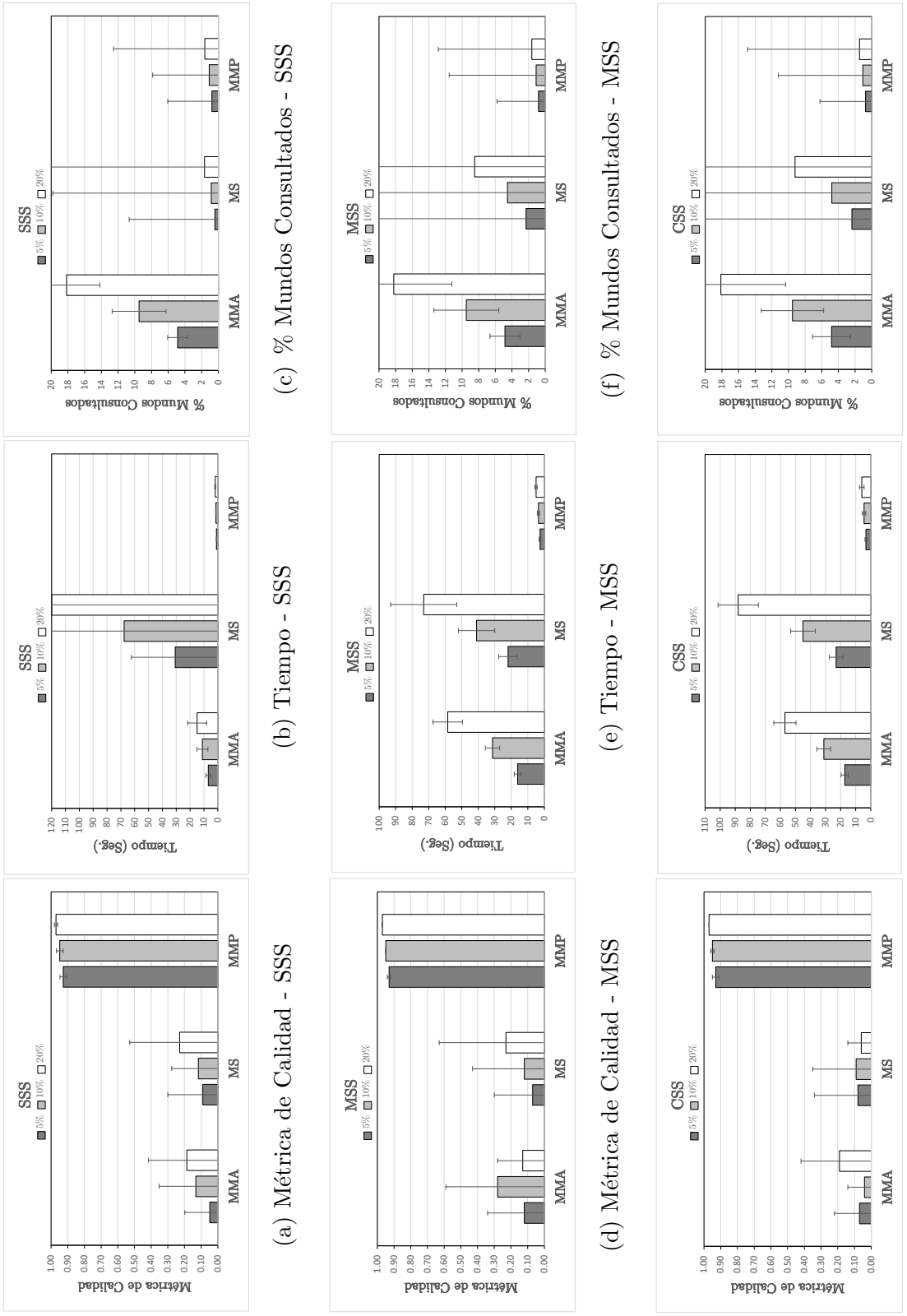
- Línea 10 (Fig. 5.5) [MMC]: El algoritmo MS arroja mejores métrias de aproximación, en un tiempo similar, y con un menor número de consultas a la RB en comparación con el algoritmo MMA. Al tener un ME que complejo (C), se trata de reducir el número de consultas a la RB.

- Línea 12 (Fig. 5.6) [SMM]: El algoritmo MS arroja mejores métricas de aproximación, en un tiempo mucho menor, y con un menor número de consultas a la RB en comparación con el algoritmo MMA. Al tener un ME que no es simple (M), se trata de reducir el número de consultas a la RB.

- Línea 13 (Fig. 5.6) [SCM,SCC]: Conforme aumenta la complejidad del ME, el algoritmo MS arroja mejores aproximaciones, en un menor tiempo, y con menor número de consultas a la RB en comparación con el algoritmo MMA. Al tener un ME que no es simple (M y C, resp.), se trata de reducir el número de consultas a la RB.

- Línea 14 (Fig. 5.7) [SSC]: El algoritmo MS arroja mejores métricas de aproximación, en un tiempo mucho menor, y con un menor número de consultas a la RB en comparación con el algoritmo MMA. En este escenario en particular, podemos dar prioridad a las métricas de tiempo y el número de consultas a la RB, ya que son las que mejoran en el algoritmo MS en comparación con los otros dos. Tener un ME complejo (C), se trata de reducir el número de consultas a la RB.

- Línea 16 (Fig. 5.7) [SSM]: En este escenario el tiempo requerido por MS es mayor en comparación con el muestreo por mundos, sin embargo, al muestrear el 20 % de los programas posibles obtenemos una métrica de aproximación cercana a 0.4; además, el porcentaje de mundos consultados a la RB es menor al 1 % de mundos posibles en el ME. Al tener un ME que no es simple (M), se trata de reducir el número de consultas a la RB.

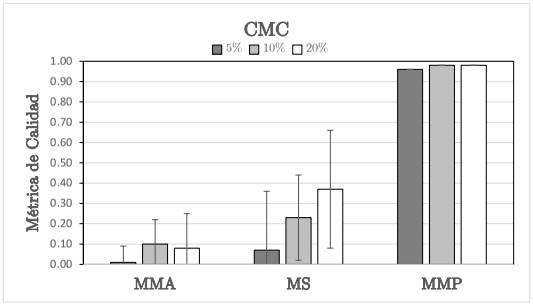


(a) Métrica de Calidad - SSS (b) Tiempo - SSS (c) % Mundos Consultados - SSS

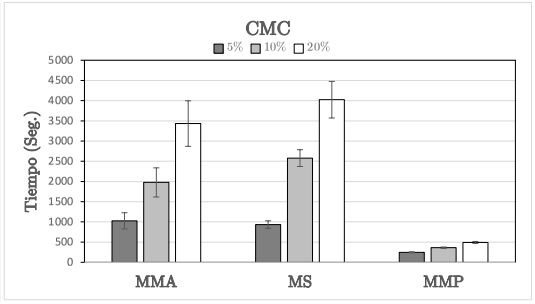
(d) Métrica de Calidad - MSS (e) Tiempo - MSS (f) % Mundos Consultados - MSS

(g) Métrica de Calidad - CSS (h) Tiempo - CSS (i) % Mundos Consultados - CSS

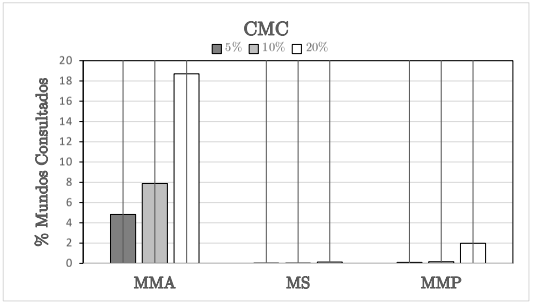
Figura 5.3: Resultados



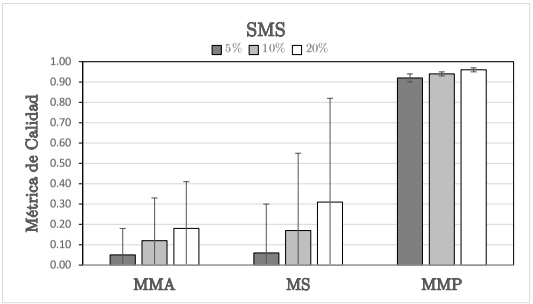
(a) Métrica de Calidad - CMC



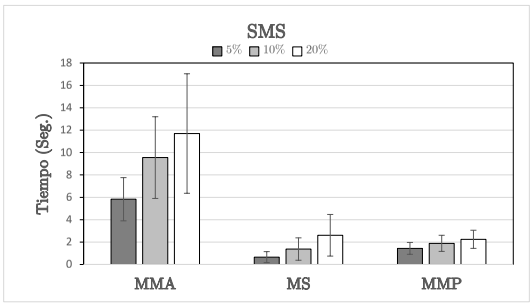
(b) Tiempo - CMC



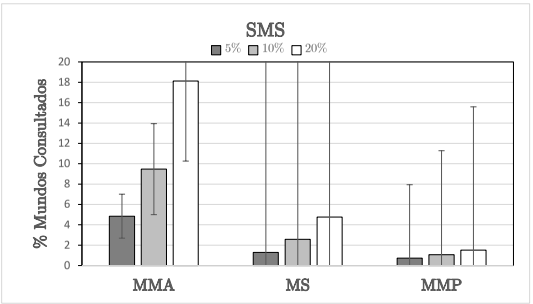
(c) % Mundos Consultados - CMC



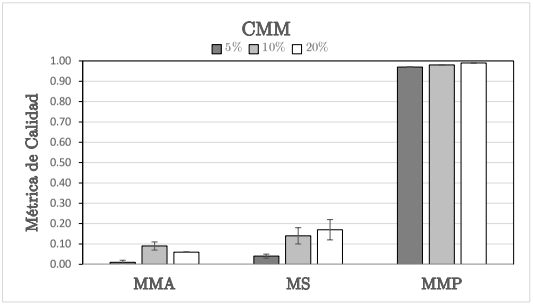
(d) Métrica de Calidad - SMS



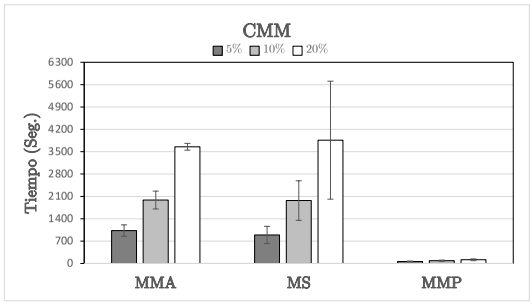
(e) Tiempo - SMS



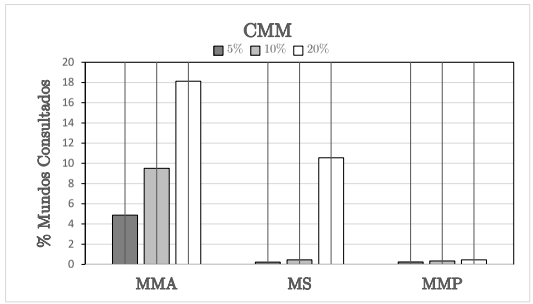
(f) % Mundos Consultados - SMS



(g) Métrica de Calidad - CMM

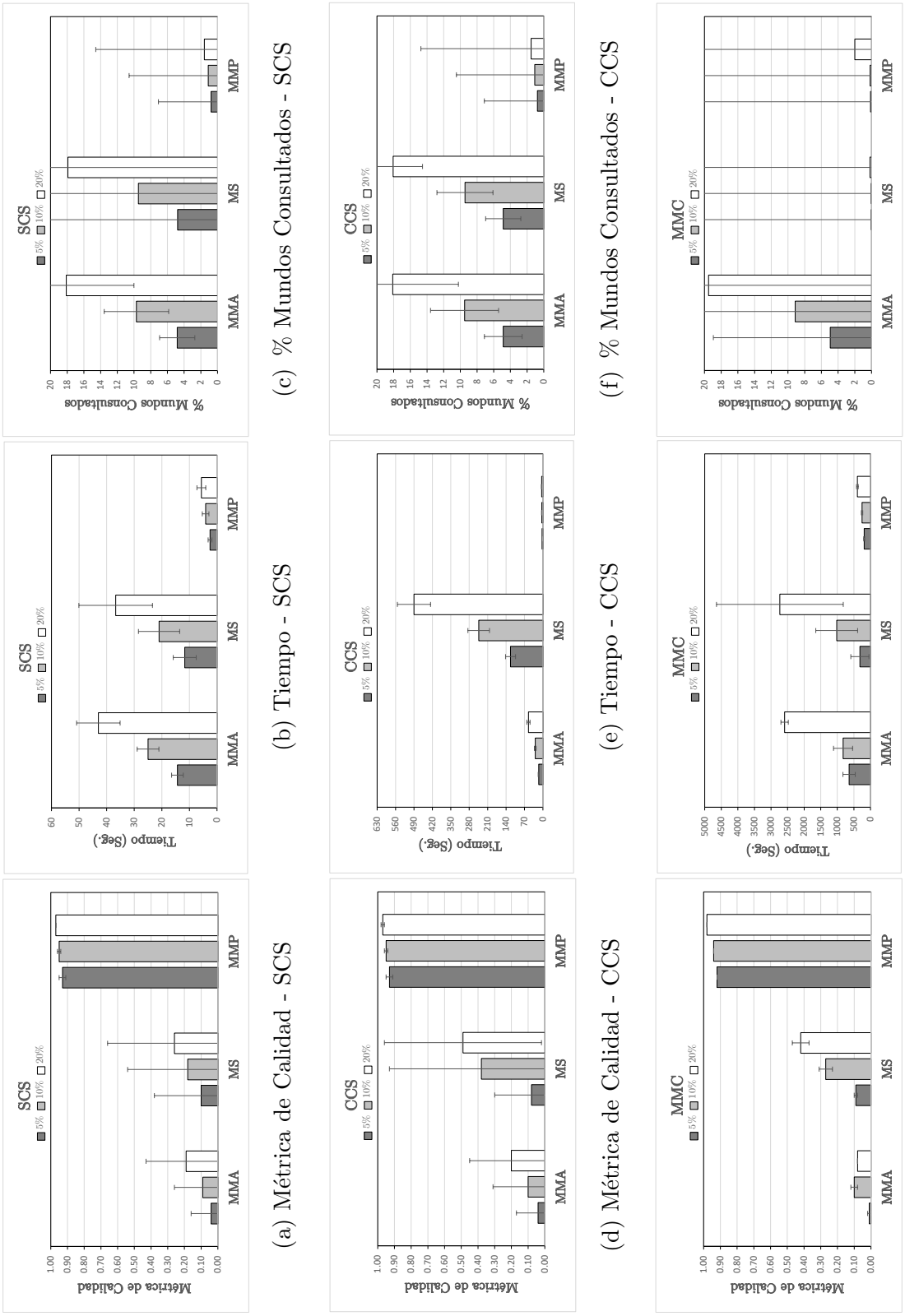


(h) Tiempo - CMM



(i) % Mundos Consultados - CMM

Figura 5.4: Resultados

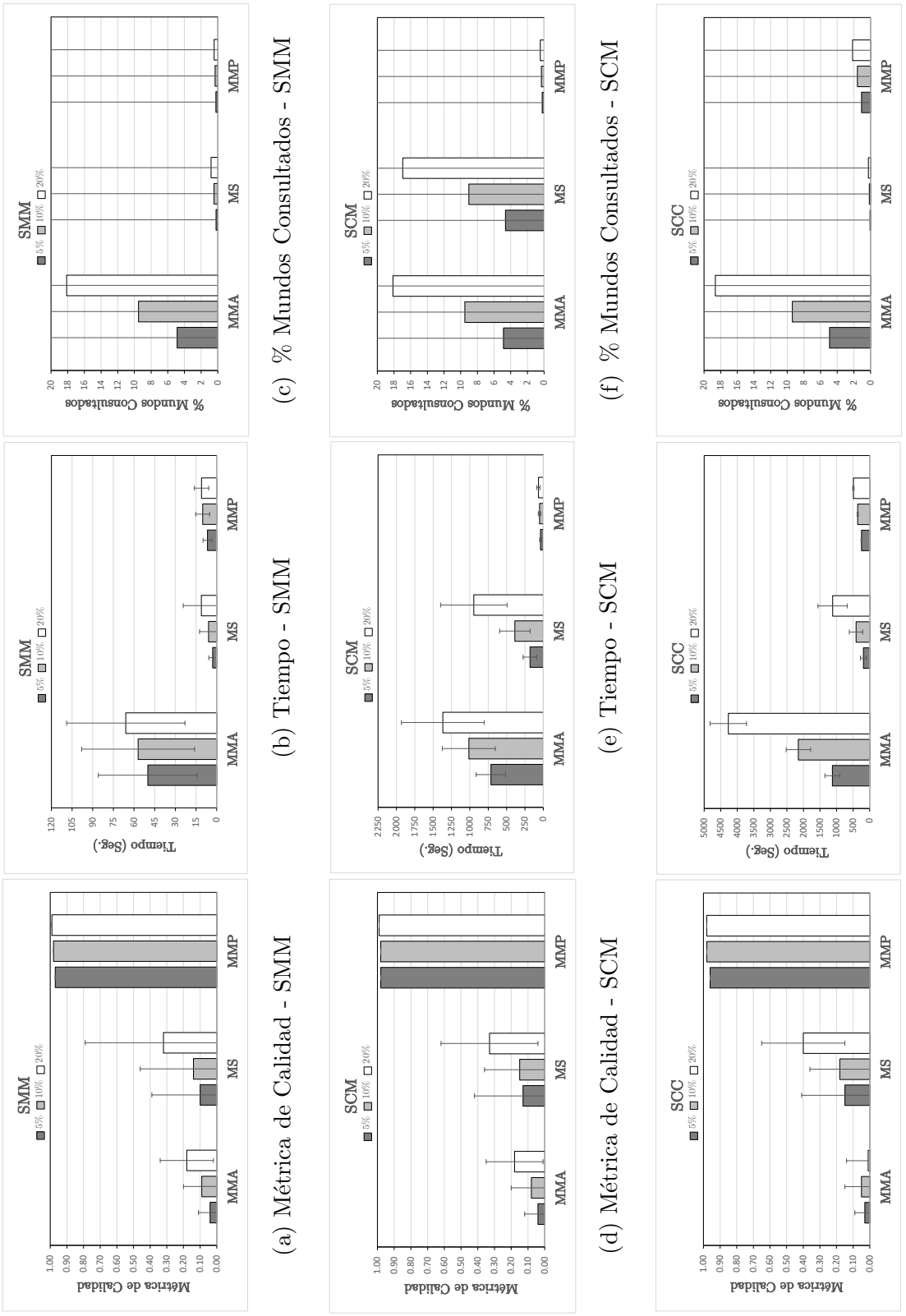


(a) Métrica de Calidad - SCS (b) Tiempo - SCS (c) % Mundos Consultados - SCS

(d) Métrica de Calidad - CCS (e) Tiempo - CCS (f) % Mundos Consultados - CCS

(g) Métrica de Calidad - MMC (h) Tiempo - MMC (i) % Mundos Consultados - MMC

Figura 5.5: Resultados

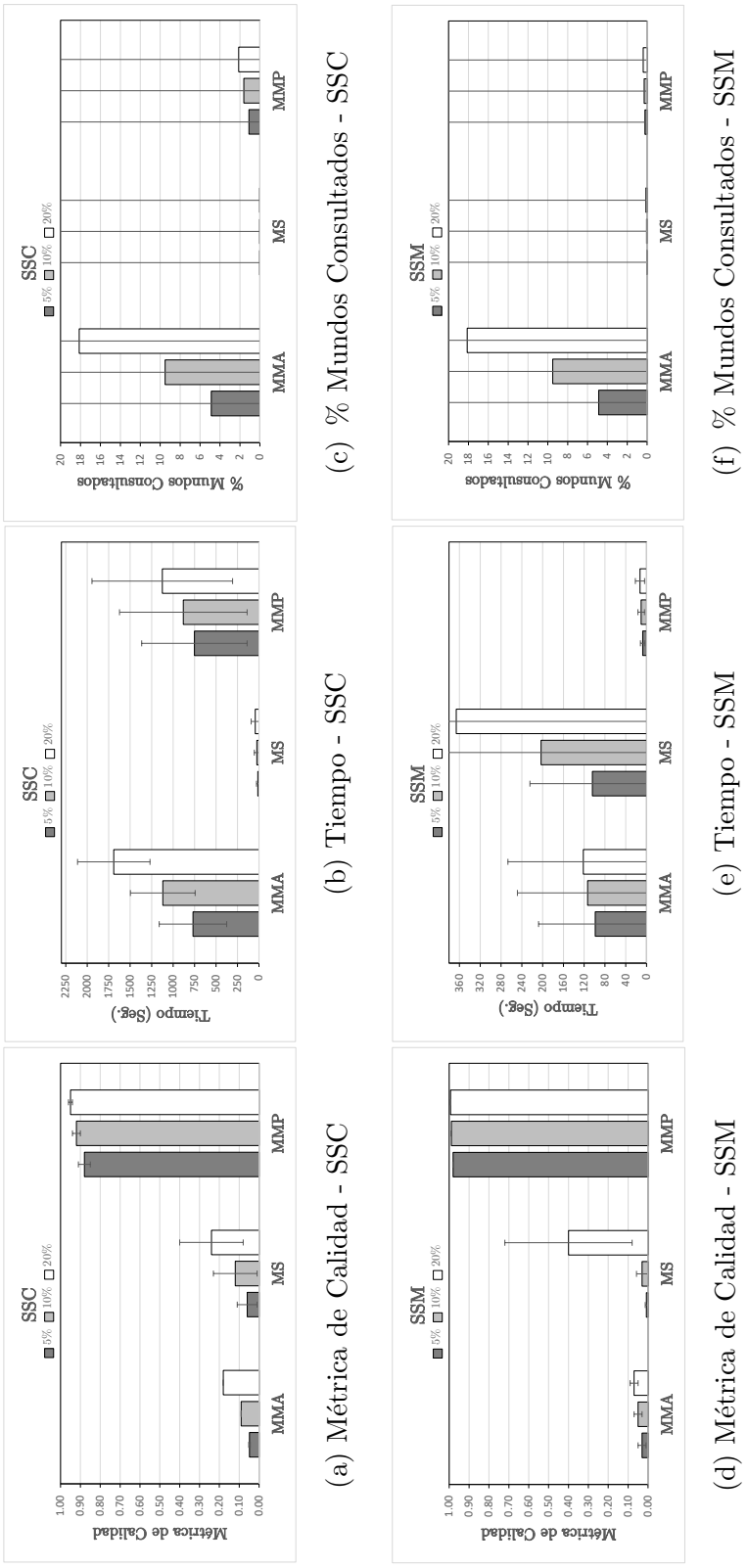


(a) Métrica de Calidad - SMM (b) Tiempo - SMM (c) % Mundos Consultados - SMM

(d) Métrica de Calidad - SCM (e) Tiempo - SCM (f) % Mundos Consultados - SCM

(g) Métrica de Calidad - SCC (h) Tiempo - SCC (i) % Mundos Consultados - SCC

Figura 5.6: Resultados



(a) Métrica de Calidad - SSC (b) Tiempo - SSC (c) % Mundos Consultados - SSC

(d) Métrica de Calidad - SSM (e) Tiempo - SSM (f) % Mundos Consultados - SSM

Figura 5.7: Resultados

5.3. Sumario

Para el desarrollo de este capítulo, se diseñó y ejecutó una batería de pruebas para comparar el desempeño de los algoritmos de aproximación presentados en el Capítulo 3. Se presentó el diseño del experimento, el objetivo final que motivó la ejecución del mismo, la creación del conjunto de pruebas, las métricas a analizar para comparar los algoritmos y, por último, los resultados de las pruebas. A partir del análisis de los resultados, se pudo completar y validar el árbol de decisión presentado en el Capítulo 3, el cual tiene la finalidad de servir como una guía para seleccionar el algoritmo de aproximación que mejor se adapte en base a la información disponible del dominio.

Capítulo 6

P-DAQAP: Plataforma para DeLP3E

En este capítulo se presenta P-DAQAP (Probabilistic Defeasible Argumentation Query Answering Platform), una plataforma web para *Respuesta a Consultas en Argumentación Rebatible Probabilística*, la cual ofrece una interfaz visual y un panel de control que facilita el análisis del proceso argumentativo llevado a cabo en la Progamación Lógica Rebatible (DeLP) [Gar97] y la argumentación probabilística a través del modelo DeLP3E [SSM+16]. El desarrollo de esta plataforma comenzó con la necesidad de contar con una herramienta que permita analizar de manera visual una base de conocimiento DeLP; luego, una vez que se desarrollaron y estudiaron los algoritmos de aproximación y los métodos de sampleo, se decidió extender la plataforma para poder trabajar con modelos DeLP3E. Dicha extensión a la plataforma se refleja en este capítulo a través de las funcionalidades que ofrece la misma, y el diseño de la interfaz de usuario o panel de control se muestra a través de bosquejos ("Web Mockups"). Con respecto al análisis del proceso argumentativo en DeLP, que se podría considerar como la sección de *argumentación rebatible clásica*, la herramienta representa a través de grafos la interacción de los argumentos generados a partir de un programa DeLP. Esto se realiza de dos maneras diferentes: la primera se centra en las estructuras obtenidas del programa DeLP, mientras que la segunda presenta las relaciones de derrota desde el punto de vista de los marcos argumentativos abstractos, con la posibilidad de calcular las extensiones utilizando la semántica de Dung (ver Sección 2.1 del Capítulo 2). Utilizando todos estos datos, la plataforma proporciona soporte para conocer el estado de los literales del programa de entrada.

El objetivo de este desarrollo es mostrar que una herramienta de este tipo puede resultar muy útil para el análisis de dominios complejos, como así también puede ser utilizada con fines educativos (mostrar los procesos argumentativos y probabilísticos dentro de DeLP y DeLP3E). A lo largo del capítulo se presentan casos de uso que muestran estas capacidades.

Este capítulo se organiza de la siguiente manera: en la Sección 6.1 se describe la motivación principal para el desarrollo de la plataforma; en la Sección 6.2 se presenta un caso de uso de la extensión DeLP3E para el análisis de ciberamenazas usando un conjunto de datos públicos; por último, en la Sección 6.3, se presenta la plataforma detallando su arquitectura, el análisis sobre el programa DeLP y los grafos de Dung, y las funcionalidades que representan la extensión DeLP3E.

6.1. Motivación

Cada día se desarrollan nuevas aplicaciones e investigaciones en Inteligencia Artificial (IA) en una amplia variedad de dominios, y en los últimos años el área de aprendizaje automático (ML) ha sido particularmente activa. Un problema inherente en esta área es el de la explicabilidad y la interpretabilidad, temas que no eran centrales en los primeros "auges de la IA" caracterizados por sistemas expertos y modelos basados en reglas. Los problemas subyacentes a este problema se encuentran dentro del dominio de la IA explicable (XAI, por sus siglas en inglés), que ahora es ampliamente reconocida como una característica crucial para el despliegue práctico de modelos de IA [ADRD$^+$20]. La importancia de este aspecto se puede apreciar señalando el programa "Inteligencia Artificial Explicable (XAI)" lanzado por la Agencia de Proyectos de Investigación Avanzados de Defensa (DARPA) [Gun17], que tiene como objetivo crear un conjunto de nuevas técnicas de inteligencia que permiten a los usuarios finales comprender, confiar y gestionar eficazmente la generación emergente de sistemas de inteligencia artificial [VM20]. El peligro es que los modelos complejos de caja negra (algunos de los cuales pueden estar formados por cientos de capas y millones de parámetros) [Cas16] se utilizan cada vez más para hacer predicciones importantes en contextos críticos, y estos modelos generan resultados que pueden no estar justificados o que simplemente no permiten explicaciones detalladas de su comportamiento [Gun17].

En este trabajo, nos centramos en la ciberseguridad como un ejemplo destacado de un contexto crítico en el que la disponibilidad de explicaciones que respalden la salida de un modelo son cruciales. La transparencia del modelo puede entenderse como la posibilidad de conocer los mecanismos a través de los cuales funciona y llega a sus resultados; cuando esto no es posible, los modelos correspondientes se describen típicamente como "cajas negras". Por lo tanto, la transparencia, junto con un procedimiento que involucre al humano (Human-in-the-Loop o HITL, por sus siglas en inglés), conduce a procesos de toma de decisiones más sólidas en cuyos resultados los usuarios pueden confiar [KLK20]. Aunque los sistemas basados en conocimiento son más aptos para implementar tales cualidades, también deben lidiar con información proveniente de múltiples fuentes heterogéneas con diferentes niveles de incertidumbre, ya sea debido a brechas en el conocimiento (incompletitud), sobreespecificación (inconsistencia) o por la incertidumbre inherente al contexto. En los dominios de la ciberseguridad, un claro ejemplo es la tarea del análisis de seguridad en tiempo real, que es un proceso complejo en el que se involucran muchos factores inciertos dado que los analistas deben lidiar con el comportamiento de diferentes actores y entidades (como atacantes, usuarios inofensivos, y herramientas de software de diversa sofisticación), la naturaleza dinámica de las vulnerabilidades y el hecho de que las observaciones del personal de defensa sobre posibles actividades de ataque son limitadas. El análisis de amenazas cibernéticas (CTA, por sus siglas en Inglés) [Als20] es un problema de inteligencia altamente técnica en el que un analista (humano) toma en consideración múltiples fuentes de información, con grados posiblemente variables de confianza o incertidumbre, con el objetivo de obtener información sobre los eventos de interés que pueden representar una amenaza para un sistema. Al crear herramientas de IA para ayudar en dicho proceso, los ingenieros de conocimiento enfrentan el desafío de aprovechar el conocimiento incierto de la mejor manera posible al resolver diferentes tipos de problemas [LSSS19].

Debido a la naturaleza de estos procesos de análisis, un sistema de razonamiento automatizado con capacidades humanas en el circuito sería el más adecuado para la tarea. Tal sistema debe ser capaz de lograr varios objetivos, entre los cuales distinguimos las siguientes capacidades principales [SSM+14]: (i) razonar acerca de la evidencia de una manera formal; (ii) considerar la evidencia asociada con la incertidumbre probabilística; (iii) considerar reglas lógicas que permitan que el sistema saque conclusiones basadas en ciertas piezas de evidencia y aplique iterativamente tales reglas; (iv) considerar piezas de información que pueden no ser compatibles entre sí, decidiendo cuáles son las más relevantes;

y (v) mostrar el estado real del sistema basado en las características descritas anteriormente, y proporcionar al analista la capacidad de comprender por qué una respuesta es correcta y cómo el sistema llega a esa conclusión (es decir, *explicabilidad e interpretabilidad*). El modelo que usamos en esta plataforma se basa en *razonamiento basado en la argumentación*; La razón principal detrás de esta elección es que tal enfoque, diseñado para imitar la forma en que los humanos llegan racionalmente a conclusiones al analizar argumentos a favor y en contra de ellos, es especialmente adecuado para adaptarse a las características mencionadas anteriormente, proporcionando naturalmente transparencia en sus mecanismos operativos.

La plataforma P-DAQAP que presentamos en este capítulo permite trabajar con el modelo DeLP3E [SSM+16], que se diseñó en función de los requisitos mencionados anteriormente. A modo de resumen, una base de conocimiento DeLP3E consta de dos partes: un *modelo analítico* (MA) y un *modelo del entorno* (ME), que representan diferentes aspectos de un dominio. El primer modelo contiene toda la información del dominio y el conocimiento que está disponible para el análisis: aquí podemos tener reglas, hechos o presunciones para representar el conocimiento del dominio. De este modelo podemos obtener conclusiones basadas en la creación y análisis de argumentos usando *programación lógica rebatible* [GS04]. Por otro lado, el modelo del entorno se utiliza para describir el conocimiento previo y es de naturaleza probabilística. A diferencia del MA, este modelo debe ser consistente, lo que simplemente significa que debe existir una distribución probabilística sobre los posibles estados del mundo que satisfaga todas las restricciones del modelo, así como los axiomas de la teoría de la probabilidad. En general, el ME contiene conocimiento como evidencia, informes de inteligencia o conocimiento sobre acciones, software y sistemas. Por lo tanto, hay dos tipos de incertidumbre que deben modelarse: la incertidumbre probabilística y la que surge del conocimiento rebatible. DeLP3E permite que ambas coexistan, como así también combinaciones de las dos, ya que las reglas y presunciones rebatibles (es decir, hechos rebatibles) también pueden anotarse con eventos probabilísticos.

La plataforma se basa en las características que ofrece DeLP3E para modelar el razonamiento en un dominio específico basado en las observaciones de eventos probabilísticos. Nuestro objetivo es diseñar una herramienta eficaz y eficiente que permita realizar tareas como:

- evaluar el estado del sistema bajo un conjunto particular de eventos observados (evidencia);
- derivar los escenarios más probables en base a escenarios contrafácticos (también conocido como razonamiento "what if", en inglés) y;
- aproximar el intervalo de probabilidad asociado con una consulta considerando todas las posibles eventualidades (mundos posibles).

Como veremos, estas tareas se pueden complementar naturalmente con la capacidad de comprender cómo el sistema llega a su conclusión a través de explicaciones basadas en los modelos MA y ME.

6.2. Caso de Uso: Análisis de Ciberamenazas con DeLP3E

En esta sección presentamos un caso de uso que aprovecha varios conjuntos de datos desarrollados y mantenidos por MITRE Corporation (una organización sin fines de lucro que trabaja con gobiernos, la industria y el mundo académico) y el Instituto Nacional de Estándares y Tecnología (NIST, por sus siglas en Inglés), una agencia dentro del gobierno de Estados Unidos[1]. La figura 6.1 muestra una descripción general de nuestro enfoque. Primero describimos los componentes básicos y luego mostramos cómo se especifican los componentes DeLP3E, junto con dos consultas para abordar problemas específicos en el dominio del análisis de amenazas cibernéticas.

El modelo ATT&CK es una base de conocimiento curada y un modelo orientado hacia el comportamiento adversarial en entornos de ciberseguridad; contiene información sobre las distintas fases de un ataque, así como las plataformas que son el objetivo más común. El modelo de comportamiento consta de varios componentes centrales:

- *Tácticas (Tactics)*, que denota las tácticas adversariales que son las metas a corto plazo durante un ataque.

[1]Conjuntos de datos MITRE: ATT&CK (https://attack.mitre.org), CAPEC (https://capec.mitre.org) y CWE (https://cwe.mitre.org). NIST administra la Base de Datos Nacional de Vulnerabilidades (NVD) (https://nvd.nist.gov) que contiene, entre otros, CVE y CPE.

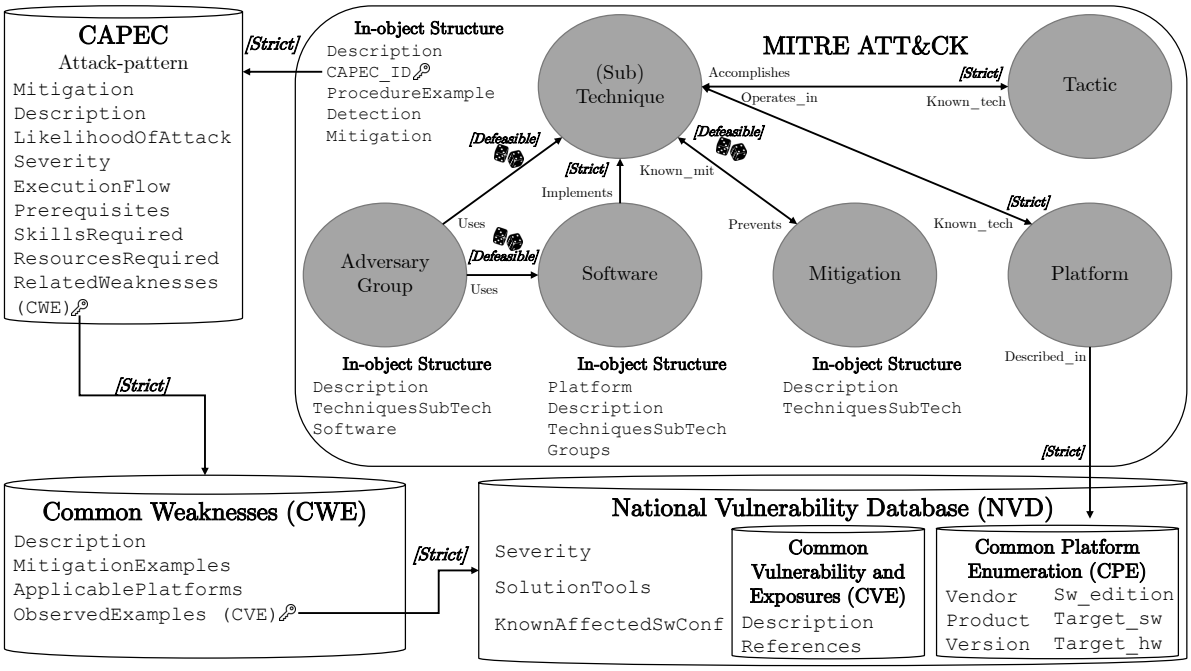


Figura 6.1: Mapeo de un conjunto de datos públicos sobre ciberseguridad a un modelo DeLP3E para el análisis de ciberamenazas.

- *Técnicas (Techniques)*, describiendo los medios por los cuales los adversarios logran objetivos tácticos.
- *Subtécnicas (Subtechniques)*, describiendo medios más específicos, a un nivel más bajo que las técnicas, por los cuales los adversarios logran objetivos tácticos.
- *Uso adversarial (Adversarial Group)* de técnicas documentadas, sus procedimientos y otros metadatos.

Los conjuntos de datos de apoyo brindan información sobre *patrones de ataque* (CAPEC), *tipos de debilidades* de software y hardware (CWE) y la *base de datos nacional de vulnerabilidades* (NVD). Este último es un depósito de datos muy valioso; aquí, distinguimos dos subconjuntos que incluyen datos sobre *vulnerabilidades* (CVE) y *plataformas* (CPE).

La Figura 6.1 muestra la información proporcionada por cada conjunto de datos y cómo se relacionan entre sí mediante claves foráneas. Por ejemplo, las técnicas de ataque incluidas en ATT&CK enlazan a entradas en CAPEC, que a su vez enlazan a CWE y NVD. Aumentamos esta estructura con dos características para derivar una base de conocimiento

Θ :
$\theta_1 = adv\_group(G)$
$\theta_3 = platform\_available(P)$
$\theta_2 = software(S)$
$\theta_4 = tech\_subtech(T\_ST)$

Ω :
$\omega_1 = accomp\_tactic(Tactic) \leftarrow tech\_subtech(T\_ST)$
$\omega_2 = op\_in\_platform(Platform) \leftarrow tech\_subtech(T\_ST)$
$\omega_3 = impl\_techsub(T\_ST) \leftarrow software(S)$
$\omega_4 = capec\_rel\_weaknesses(CWE\_List) \leftarrow capec\_id(T\_ST)$
$\omega_5 = cwe\_observed(CVE\_List) \leftarrow capec\_rel\_weaknesses(CWE\_List)$
$\omega_6 = nvd\_cve(Vuln\_info) \leftarrow cwe\_observed(CVE\_List)$
$\omega_7 = known\_techst(T\_ST) \leftarrow accomp\_tactic(T)$
$\omega_8 = known\_techst(T\_ST) \leftarrow platform\_available(P)$

Φ :
$\phi_1 = mitigation(M) \multimap$ $af(\phi_1) = e_1$
$\phi_2 = likelihoodAttack(CAPEC\_ID,\ Value) \multimap$ $af(\phi_2) = e_2$

Δ :
$\delta_1 = prev\_techsub(T\_ST) \multimap mitigation(M)$ $af(\delta_1) = e_3$
$\delta_2 = known\_mit(M) \multimap tech\_subtech(T\_ST)$ $af(\delta_2) = e_4$
$\delta_3 = tech\_in\_use(T\_ST) \multimap adv\_group(G)$ $af(\delta_3) = e_5$
$\delta_4 = soft\_in\_use(S) \multimap adv\_group(G)$ $af(\delta_4) = e_6$
$\delta_5 = pos\_threat(T\_ST,\ S) \multimap tech\_in\_use(T\_ST), soft\_in\_use(S)$ $af(\delta_5) = e_7$
$\delta_6 = \sim impl\_techsub(T\_ST) \multimap prev\_techsub(T\_ST)$ $af(\delta_6) = e_8$
$\delta_7 = intensify\_mit(M) \multimap known\_mit(M),\ tech\_in\_use(T\_ST),\ likelihoodAttack(T\_ST,\ high)$ $af(\delta_7) = e_9$

Figura 6.2: *Izquierda*: Programa PreDeLP que conforma el modelo analítico. *Derecha*: Función de Anotación.

DeLP3E. Primero, etiquetamos las conexiones entre conjuntos de datos (así como los componentes dentro de ATT&CK) con "[*strict*]" (estricto) o "[*defeasible*]" (rebatible), indicando el tipo de conocimiento que se codifica. Por ejemplo, los casos observados de una debilidad incluida en CWE están vinculados a los CVE incluidos en el NVD como estrictos, ya que se trata de un conocimiento bien establecido. Por otro lado, las estrategias de mitigación están vinculadas a las técnicas como conocimiento rebatible, ya que la relación entre las dos es de naturaleza tentativa. La segunda característica, que aparece en la figura como un pequeño icono que representa un par de dados, indica relaciones que están sujetas a *eventos probabilísticos*. En este caso de uso, todas las relaciones rebatibles se etiquetan de esta manera.

Usamos toda esta información para crear la función de anotación, los modelos analítico y del entorno y así, crear una base de conocimiento DeLP3E; un ejemplo pequeño se muestra en la Figura 6.2. En el lado izquierdo tenemos los elementos del MA, que se pueden usar para crear argumentos a favor y en contra de las conclusiones; por ejemplo:

$$\langle \mathcal{A}_1, \textit{tech\_in\_use(account\_discovery)} \rangle, \text{ con}$$
$$\mathcal{A}_1 = \{\delta_3, \theta_1(\textit{adv\_group}(\textit{apt}29))\}$$

$$\langle \mathcal{A}_2, {\sim}\textit{impl\_techsub(os\_credential\_dumping)} \rangle, \text{ con}$$
$$\mathcal{A}_2 = \{\delta_6, \delta_1(\textit{prev\_techsub(os\_credential\_dumping)}),$$
$$\phi_1(\textit{mitigation(credential\_access\_protection)})\}.$$

El primero indica que la técnica *account discovery* se está utilizando como técnica de ataque, ya que el grupo avanzado de amenazas persistentes 29 (APT29, también conocido como "Cozy Bear") está activo y se sabe que éste la utiliza. El último se refiere al uso de *credential access protection* como técnica de mitigación para evitar el uso de *OS credential dumping*. Este es un claro ejemplo de un argumento que implica incertidumbre, ya que la protección de acceso de credenciales no es un esfuerzo infalible; un ejemplo de esto es la conocida vulnerabilidad *Heartbleed* (CVE-2014-0160) que afectó las implementaciones de OpenSSL, dejándola abierta para que se filtren credenciales. En este ejemplo simple solo etiquetamos componentes del MA con eventos probabilísticos (e_1-e_9; los elementos sin anotación simplemente se etiquetan con *true*) y no se detalla cómo se relacionan en el ME. Un ejemplo podría ser simplemente asumir la independencia por pares (como se hace en muchos modelos de base de datos probabilísticos), o una red bayesiana.

Finalmente, presentamos dos consultas a las que volveremos en la sección 6.3.4:

- *pos_threat*($T1134, SO344$): ¿Cuál es la probabilidad de que se utilice la *manipulación del token de acceso* (técnica T1134) aprovechando el malware *Azorult* (ID de software SO344) para atacar nuestros sistemas?
- *intensify_mit*($M1026$): ¿Cuál es la probabilidad de que se implemente la *administración de cuentas con privilegios* (estrategia de mitigación M1026)? Tener en cuenta que M1026 mitiga T1134.

A partir de este simple ejemplo podemos notar que la herramienta es de gran utilidad para analizar dominios complejos. La misma permite realizar diferentes consultas, modelando la información argumental y probabilística en un solo modelo general y modular. En la sección relacionada a las funcionalidades DeLP3E (6.3.4), detallaremos las posibles consultas y la manera de mostrar los resultados al usuario.

6.3. La plataforma P-DAQAP

La plataforma web que presentamos consiste en una interfaz para visualizar mediante grafos la interacción de los argumentos generados a partir de un programa DeLP de entrada. Distinguimos tres secciones: la primera está dedicada al análisis de las estructuras obtenidas del programa DeLP (argumentos y derrotadores) y sus interacciones; la segunda analiza las relaciones de derrota en un grafo de Dung, y la tercera, es la extensión para trabajar con bases de conocimiento DeLP3E, es decir, con argumentación rebatible probabilística. A continuación presentamos un resumen de la arquitectura de la plataforma, luego nos enfocamos en los análisis de DeLP y Dung por separado y, por último, presentaremos las funcionalidades para trabajar con DeLP3E.

6.3.1. Arquitectura

La plataforma P-DAQAP se basa en una arquitectura cliente-servidor. Primero, detallaremos el flujo de trabajo de las secciones DeLP y argumentación abstracta (grafos de Dung): el cliente envía el programa DeLP de entrada para que lo procese el módulo DeLP que se aloja en un servidor; el módulo DeLP se encarga de generar los argumentos, las relaciones entre ellos (subargumento y derrotadores), y determinar el estado de cada argumento, para lo cual genera y analiza los árboles de dialéctica. Luego, los datos resultantes se envían al cliente en formato JSON donde se presentan gráficamente. La figura 6.3 presenta un esquema general de esta arquitectura.

Funcionamiento de P-DAQAP (DeLP y Argumentación Abstracta) (*Figura* 6.3) El usuario ingresa el programa DeLP para ser analizado junto con un criterio de preferencia (*prioridad entre reglas* o *especificidad generalizada*). Se realiza un análisis sintáctico y si el programa no contiene errores se envía al servidor web (1). Luego, el servidor web envía el programa y la instrucción que debe ejecutar el módulo DeLP alojado en el servidor de cómputo (A). El módulo DeLP se encarga de generar los argumentos y las relaciones entre ellos (subargumentos y derrotadores), y de retornar los datos en formato JSON al servidor web (B) para que, luego de verificar la respuesta, sea posible enviar estos datos al cliente y así poder graficarlos (2). Esto corresponde al análisis DeLP, en el que es posible obtener todos los argumentos generados y los árboles de dialéctica construidos durante el proceso argumentativo. Con respecto al análisis del grafo de Dung, la primera parte del

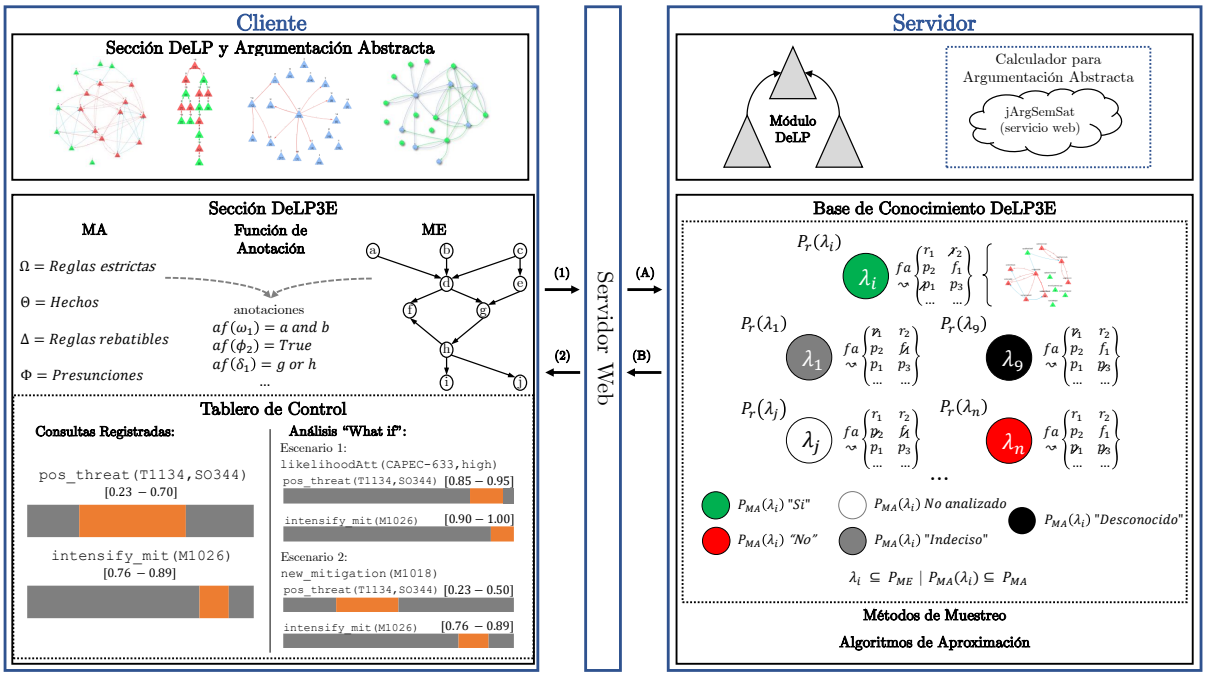


Figura 6.3: Arquitectura de la plataforma P-DAQAP.

flujo de trabajo es análoga a la de DeLP, la principal diferencia es que después de (2), el cliente envía los identificadores de los argumentos, los ataques y la semántica a calcular (1); en el servidor web se construye un archivo JSON para realizar una consulta a un calculador de semánticas (solver) a través de un servicio web (A) y luego, una vez que se obtiene una respuesta del solver (B), el servidor web verifica la respuesta y la envía al cliente (2).

Con respecto a la sección de DeLP3E, la Figura 6.3 muestra un bosquejo de la interfaz de usuario. Al igual que en la primera sección, la arquitectura se divide en dos módulos principales, el cliente y el servidor. En el lado del servidor se agregan tres submódulos que implementan el modelo de probabilidad (para el ME), métodos de muestreo y algoritmos de aproximación. Los dos últimos son los presentados y estudiados en los capítulos anteriores de esta tesis. A continuación, se detallará el flujo de trabajo centrado en las tareas DeLP3E en el orden de los pasos etiquetados en la interacción entre los dos módulos principales ($1 \rightarrow A \rightarrow B \rightarrow 2$). Este flujo de trabajo es de naturaleza iterativa e implementa el modelo HITL (Human-in-the-Loop).

Funcionamiento de P-DAQAP (DeLP3E) (*Figura* 6.3)

Cliente: El usuario carga una base de conocimiento DeLP3E (que involucra una especificación de MA, ME y la función de anotación) y especifica una *tarea* (Paso 1). Algunos ejemplos de tareas son tipos específicos de consultas: calcular todos los literales garantizados con sus intervalos de probabilidad o mostrar explicaciones para un par específico literal-intervalo.

Servidor: En este flujo de trabajo específico de DeLP3E, el servidor web envía el trabajo para que lo ejecute el módulo de argumentación probabilística (paso A); en el caso general, también podría enviarse al módulo de argumentación clásica. Este módulo del servidor genera luego las estructuras de datos asociadas con la base de conocimiento y ejecuta el trabajo, que involucra flujos de datos y control entre los diferentes submódulos.

Una vez que los resultados están disponibles, el servidor es responsable de devolver los datos de salida en formato JSON al servidor web (paso B). Tener en cuenta que, dada la naturaleza de los algoritmos de aproximación, un enfoque basado en algoritmos *anytime* puede ser adecuado, en el que los resultados se mejoran iterativamente y el usuario puede decidir cuándo detener el trabajo.

Cliente: El cliente recibe del servidor la respuesta a la solicitud de trabajo enviada en el Paso 1 y los datos se presentan al usuario. El analista ahora puede interactuar a través del tablero en respuesta a los resultados recibidos, por ejemplo, eligiendo hacer modificaciones a la base de conocimiento DeLP3E o modificando la consulta emitida en el primer paso.

6.3.2. El Análisis DeLP

Como se mencionó anteriormente, el análisis de los programas DeLP lo maneja el módulo principal DeLP, que toma un programa DeLP $\mathcal{P}$ y un criterio de preferencia como entradas y retorna un objeto JSON que contiene información sobre todos los argumentos que se pueden construir a partir de $\mathcal{P}$, las relaciones de derrota, el conjunto de literales garantizados como así también su correspondiente árbol de dialéctica etiquetado y las relaciones de subargumento entre argumentos. A partir de estos datos, se dibuja un grafo DeLP para mostrar los argumentos, subargumentos, relaciones de derrota y estado de cada argumento. Los elementos utilizados para representar los datos en el grafo se muestran en la Figura 6.4; en el grafo, los argumentos se representan como triángulos con la conclusión en la parte superior y un identificador en su cuerpo. También es posible visualizar el árbol de dialéctica generado para cada argumento y configurar diferentes vistas para el grafo

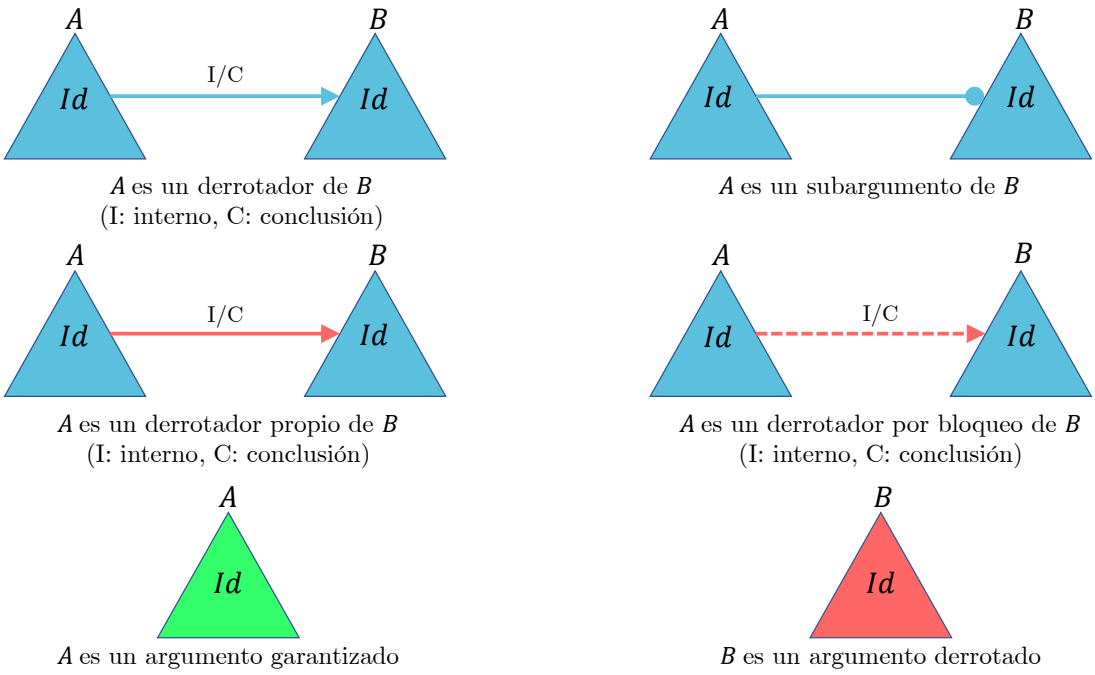


Figura 6.4: Elementos de un grafo DeLP.

DeLP. Además del programa DeLP a ser analizado, es opcional la especificación de un criterio de preferencia, el cual puede ser *especificidad generalizada* o *prioridad entre reglas*.

Ahora vamos a mostrar estas características con un ejemplo, comenzando con un grafo simple; considere el siguiente programa DeLP $\mathcal{P} = (\Pi, \Delta)$, donde Π es el conjunto de hechos y Δ es el conjunto de reglas rebatibles, y supongamos que usamos el criterio *especificidad generalizada*:

$$\Pi = \left\{ \begin{array}{lll} p & \leftarrow & t \\ t & & \\ z & & \end{array} \right\} \quad \Delta = \left\{ \begin{array}{lll} \sim a \prec y & y \prec x & x \prec z \\ y \prec p & a \prec w & w \prec y \\ \sim w \prec t & \sim x \prec t & x \prec p \end{array} \right\}$$

En la Figura 6.5 se muestra el grafo generado para este programa DeLP con la vista predeterminada (todas las opciones del panel de configuración están habilitadas). Aquí es posible ver los argumentos creados, cada uno con su identificador y la conclusión que respalda, sus estados de garantía y las relaciones de derrota. A partir de este grafo DeLP es posible analizar las estructuras y relaciones que se generan a partir de un programa de entrada. Para ver el conjunto de reglas que componen un argumento en particular, el usuario puede ubicar el puntero del ratón sobre el argumento en el grafo y se mostrará (en una ventana emergente) el conjunto de reglas que lo componen. Como se detalló ante-

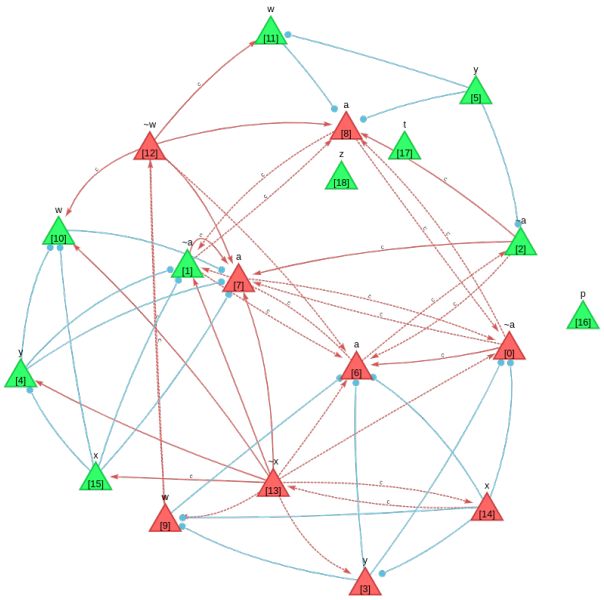


Figura 6.5: Grafo DeLP del ejemplo $\mathcal{P}$.

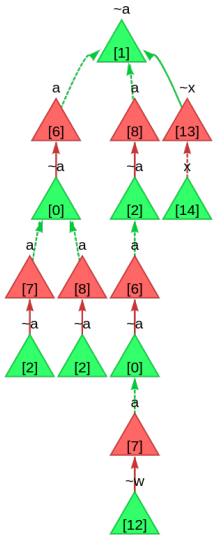


Figura 6.6: Árbol de Dialéctica.

riormente, también es posible visualizar el árbol de dialéctica asociado a cada argumento. En la Figura 6.6 se muestra el árbol de dialéctica del argumento con id 1; aquí también es posible distinguir entre las relaciones de derrota propia o por bloqueo.

A continuación presentaremos las diferentes vistas que el usuario puede aprovechar para obtener una visualización más clara del grafo DeLP.

Vistas

La herramienta tiene la capacidad de mostrar todos los argumentos generados a partir del programa DeLP, todas las relaciones de derrota y de subargumentos, y el estado de cada argumento en único grafo llamado *grafo DeLP*. De manera predeterminada, todos estos datos se grafican juntos sobre el grafo mencionado; sin embargo, dado que para programas más grandes esto puede generar grafos muy saturados y difíciles de analizar, desarrollamos un conjunto de vistas diferentes que se pueden configurar a través de un panel de configuraciones. Estas vistas permiten ocultar cierta información, reduciendo las conexiones u ocultando el estado final de los argumentos, con la finalidad de que el usuario pueda centrarse en el aspecto que más le importe. Hay cuatro vistas definidas; a continuación presentaremos cada una de ellas y el efecto producido por alguna de ellas en el grafo generado para el programa presentado en la Figura 6.5.

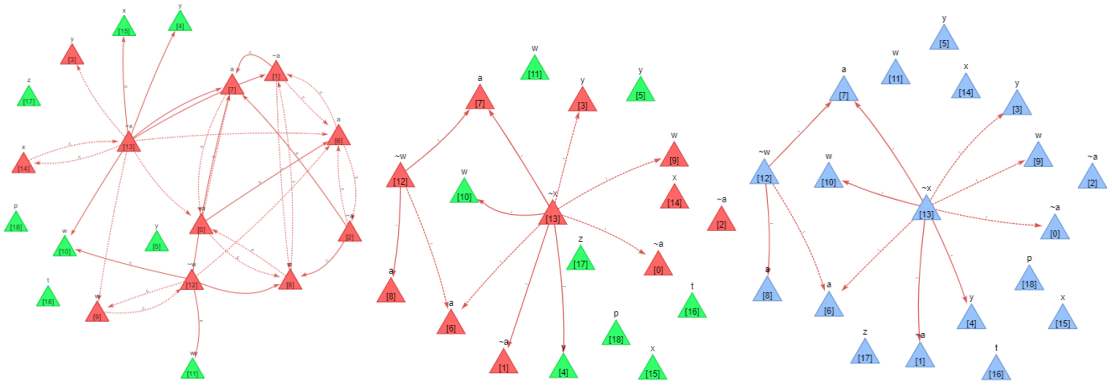


Figura 6.7: Grafo DeLP de $\mathcal{P} = (\Pi, \Delta)$ *Izquierda*: Ocultando los arcos de la relación de subargumento. *Centro*: Ocultando los arcos del tipo de ataque a la conclusión y la relación de subargumento. *Derecha*: Ocultando el tipo de ataque a la conclusión, la relación de subargumento, y el estado de los argumentos.

1. *Relación de argumento y subargumento*: Esta vista permite ocultar o mostrar la relación de subargumento, es decir, los arcos con un círculo en su final que representan que un argumento es un subargumento de otro (ver figura 6.7 a la izquierda).

2. *Tipos de derrota*: Permite diferenciar entre tipos de derrota (propias o por bloqueo), es decir, mostrar todos los arcos que tienen una flecha en su final y la misma forma.

3. *Tipo de ataque*: Con esta vista es posible mostrar u ocultar el tipo de ataque (representado por los arcos) de una relación de ataque. En la figura 6.7 (centro) se muestra el resultado de ocultar el tipo de ataque a la *conclusión* y la relación de subargumento en el grafo del programa DeLP $\mathcal{P}$.

4. *Estado de cada argumento*: Esta vista permite mostrar u ocultar el estado de cada argunento en el grafo. En la figura 6.7 (derecha) se muestra el resultado de ocultar el estado de los argumentos, la relación de subargumento, y el tipo de ataque a la conclusión sobre el grafo del programa DeLP $\mathcal{P}$.

Una de las configuraciones más "claras" para analizar programas DeLP que generan muchas relaciones de derrota se logra ocultando el tipo de ataque interno y mostrando la relación de subargumento. Con esta vista, es posible apreciar las relaciones sin sobrecargar el grafo ya que solo se mostrarán las derrotas a las conclusiones; naturalmente, esta claridad se obtiene ya que se eliminan los arcos que representan derrotas a conclusiones

internas. Esto permite concentrarse en los conflictos de forma más directa sin perder información, ya que los arcos eliminados se presentan de forma "implícita" por la relación de subargumento. Finalmente, también es posible combinar todas estas vistas.

6.3.3. El Análisis sobre el Grafo de Dung

Como se mencionó anteriormente, P-DAQAP también permite analizar el grafo DeLP generado al considerarlo como un marco de argumentación abstracta, que esencialmente es un grafo dirigido en el que los argumentos están representados por nodos y la relación de ataque está representada por arcos. También es posible calcular y mostrar diferentes semánticas (*grounded*, *estable*, *preferida* y *semistable*) sobre el grafo, como así también diferentes *intersecciones* de extensiones calculadas, esto último para dar soporte a enfoques cautelosos de respuesta a consultas.

Esta funcionalidad se proporciona ya que, dado el marco de argumentación abstracta, es posible analizar los conjuntos de argumentos que deben aceptarse mediante las *semánticas de argumentación*. Teniendo en cuenta que existen dos enfoques principales para la definición de semánticas de argumentación, es importante notar que, incluso cuando se usa un enfoque *basado en extensiones* o *basado en etiquetas*, la "respuesta final" con respecto a la justificación del argumento no se determina fácilmente. De hecho, hay varias opciones disponibles en cuanto a la derivación del estado de justificación a partir de un conjunto de extensiones o etiquetas. En un nivel básico, se pueden considerar dos alternativas muy simples para la noción de justificación: la justificación *escéptica (o cautelosa)* requiere que el argumento sea aceptado en *todas* las extensiones (o etiquetas), mientras que la justificación *crédula* requiere que el argumento sea aceptado en al menos una extensión (o etiqueta) [BCG11]. Intuitivamente, una actitud escéptica tiende a tomar decisiones menos comprometidas sobre la justificación de los argumentos; en otras palabras, un comportamiento escéptico tiende a dejar los argumentos en un estado de justificación "indeciso" y a aceptar (o rechazar) la menor cantidad de argumentos posibles, mientras que un comportamiento menos escéptico (o más crédulo) corresponde a una aceptación más amplia (o rechazo) de los argumentos: claramente, una justificación escéptica implica una justificación crédula. Además, es posible derivar una tercera justificación: un argumento es *no justificado* (o *rechazado*) si no está crédulamente justificado (y, por lo tanto, tampoco escépticamente justificado). También es importante señalar que en cualquier semántica de

estado único coinciden los enfoques escépticos y crédulos, por lo que un argumento solo puede ser aceptado o rechazado.

P-DAQAP permite trabajar con ambos enfoques para analizar qué conjuntos de argumentos pueden aceptarse. A continuación, mostramos cómo se calculan las semánticas para el programa DeLP de entrada.

Cálculo de Semánticas y Escepticismo: En DeLP, un ataque tiene éxito y se convierte en derrota solo cuando el argumento atacado no es preferido sobre el argumento que ataca. Luego, es esta relación de derrota la que se utiliza como relación binaria en el marco de Dung. Aquí, para crear el grafo se usan los componentes clásicos, es decir, los argumentos están representados por nodos (círculos) y las relaciones de ataque están representadas por arcos. A partir del grafo de Dung generado, es posible calcular las siguientes semánticas: *grounded*, *estable*, *preferido* y *semiestable*; las extensiones calculadas se grafican en otro grafo a la derecha del grafo de Dung. El cálculo de las semánticas se realiza mediante el solver *jArgSemSAT* [CVG16], el cual está disponible a través de un servicio web.

Debajo del grafo de Dung generado se muestran las semánticas que pueden calcularse; al seleccionar cualquiera de ellas se realiza la consulta al servicio web de *jArgSemSAT* y el resultado se muestra en otro grafo a la derecha del primero. En el segundo grafo, los nodos que representan al argumento que pertenece a alguna extensión de la semántica calculada se pintan de color verde. Todas las extensiones calculadas se enumeran debajo del segundo grafo. En las figuras 6.8 y 6.9 mostramos algunos ejemplos de semánticas calculadas a partir del programa DeLP $\mathcal{P} = (\Pi, \Delta)$.

De la misma manera, es posible calcular y mostrar las *intersecciones* de las extensiones mostradas. Una vez calculadas las intersecciones, se habilita un menú para seleccionar qué semántica (ahora para mostrar la intersección de sus extensiones) se desea mostrar en el segundo grafo. Además, mediante este menú, es posible calcular la *intersección de las intersecciones* de la semántica calculada y mostrarlas en el segundo grafo, lo que genera una respuesta aún "más cautelosa".

6.3.4. Funcionalidades DeLP3E

En esta sección comenzaremos detallando dos funcionalidades con respecto al caso de uso presentado en la Sección 6.2, las cuales se ilustran en el "Tablero de Control" de la figura 6.3, luego comentaremos algunas funcionalidades a ser implementadas en el futuro.

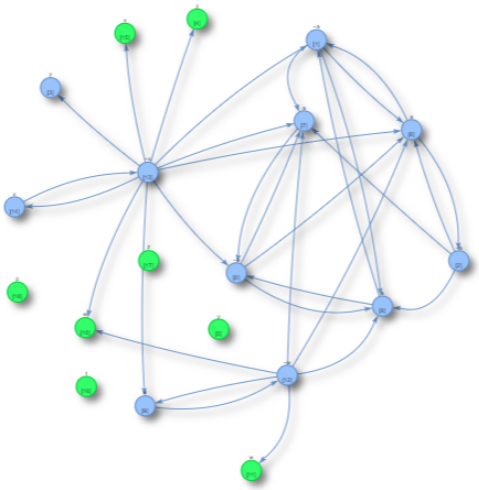


Figura 6.8: Semántica DeLP.

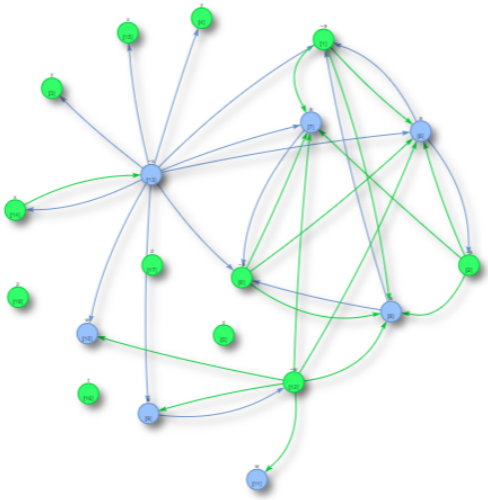


Figura 6.9: Semántica Estable.

Estado actual: *Consultas registradas*

Aquí, se establecen los valores de un subconjunto de variables del ME según el estado actual del sistema (evidencia observada). El analista registra un conjunto de consultas de interés para monitorear las probabilidades asociadas.

Por ejemplo, considere las consultas presentadas en la Sección 6.2; el usuario está interesado en monitorear una posible amenaza y el grado de aplicación de una estrategia de mitigación correspondiente. En la Figura 6.3 (abajo a la izquierda), podemos ver que en el estado actual, la consulta "$pos\_threat(T1134, SO344)$" está actualmente garantizada por la base de conocimiento con intervalo de probabilidad $[0{,}23, 0{,}7]$; este intervalo es bastante amplio, lo que apunta a una gran cantidad de incertidumbre y falta de resultados procesables. Por otro lado, la consulta "$intensify\_mit(M1026)$" arroja un intervalo de $[0{,}76, 0{,}89]$, que comunica una alta probabilidad de la necesidad de intensificar la estrategia de mitigación asociada con la técnica T1134.

Esta es una herramienta valiosa para los analistas, quienes pueden registrar consultas sobre estrategias de mitigación y técnicas de ataque de interés actual. Los resultados pueden informar, por ejemplo, los niveles de alerta de seguridad y las prioridades del esfuerzo de actualizaciones para los administradores del sistema. Como se detalló en el capítulo 5, las aproximaciones se pueden calcular siempre que el costo de obtener una respuesta exacta sea demasiado alto. En este caso, el sistema puede permitir al usuario

ingresar el número de muestras que se utilizarán o, dado un límite superior explícito en el tiempo disponible, decidir una estimación para el proceso de muestreo.

Escenarios contrafácticos ("What If")

Basado en la misma configuración que para las consultas registradas, el usuario puede estar interesado en realizar un razonamiento contrafáctico, también conocido como escenarios *"What If"*. En este caso, en lugar de tomar datos y configuraciones de variables del ME de observaciones directas, el sistema permite al usuario especificar escenarios como desee y muestra las probabilidades resultantes.

En la figura 6.3, ilustramos esta funcionalidad con las mismas consultas registradas que antes, mostrando cómo sus intervalos de probabilidad asociados cambian en dos escenarios diferentes. En el primero, el analista quiere saber cómo cambian las probabilidades en caso de que sea muy probable que una técnica se implemente con éxito según lo informado por CAPEC ($likelihoodAttack(CAPEC\text{-}633, high)$); el cambio más drástico se produce en la primera consulta, que ahora arroja una probabilidad entre 85 % y 95 %, mientras que la probabilidad de la otra consulta aumenta hasta el 90 %–100 %. Esto se debe a que CAPEC-633 se refiere a una técnica que, si se sabe que tiene una alta probabilidad de éxito, está directamente relacionada con la estrategia de mitigación M1026.

En el segundo escenario, el analista quiere saber cómo cambiarían las probabilidades si se agrega una nueva estrategia de mitigación ($new\_mitigation(M1018)$). Ahora, la consulta "$pos\_threat(T1134, SO344)$" se vuelve menos probable (23 %–50 %, ya que la nueva estrategia de mitigación ayuda a prevenir la técnica T1134), mientras que para este escenario, la respuesta a la otra consulta permanece sin cambios porque las dos estrategias de mitigación no están relacionadas.

Explicaciones basadas en reglas y escenarios más probables

Además de poder calcular la probabilidad exacta o aproximada de una consulta, es posible acompañar ese resultado con una *explicación* sobre cómo llegó el sistema a esa respuesta. Ahora discutimos dos enfoques que aún están en desarrollo para proporcionar tal información sobre el tipo de resultados presentados en las subsecciones anteriores. El primero se centra en el modelo probabilístico (ME), mientras que el segundo se centra en las reglas utilizadas para derivar las respuestas a las consultas (MA).

Escenarios más probables. Como una combinación de las dos funcionalidades anteriores, el sistema puede calcular un conjunto de k escenarios más probables dado el conjunto actual de observaciones; en la implementación actual, que utiliza redes bayesianas para especificar la distribución de probabilidad en el ME, este conjunto puede ser calculado por el módulo del modelo probabilístico simplemente devolviendo las *explicaciones más probables* (MPE, por sus siglas en Inglés) de la red bayesiana dada la evidencia actual en el ME. Luego, el resultado de este primer paso se puede combinar con el análisis contrafáctico descripto anteriormente y cada escenario se puede explorar teniendo en cuenta su probabilidad de ocurrencia y sus consecuencias.

Si bien este tipo de análisis se centra en el modelo probabilístico, conocer los escenarios más probables es un primer paso para explicar por qué una determinada consulta está relacionada con un determinado intervalo de probabilidad. Por ejemplo, un analista puede estar interesado en saber por qué el límite superior es más bajo de lo esperado, y una primera explicación sería que se le muestre un escenario de alta probabilidad en el que se implique la negación de la consulta. Si se necesitan más detalles, también se pueden derivar explicaciones analizando las reglas y argumentos involucrados en las derivaciones, como se analiza a continuación.

Explicaciones basadas en reglas. Además de analizar los escenarios más probables, otra posibilidad es mostrar los argumentos que sustentan la consulta en el subprograma generado por un escenario particular o conjunto de escenarios. Esto le proporciona al analista el conjunto de reglas, hechos y presunciones involucradas en la derivación, y precisamente qué papel desempeñaron, lo que puede resaltar la necesidad de revisar uno o más de estos componentes (considere, por ejemplo, hechos que provienen de una fuente de datos desactualizada que necesita ser actualizada o eliminada); un enfoque en esta dirección se informó recientemente en [BTG21]. Otro beneficio de los enfoques basados en reglas es que se pueden hacer más interpretables, por ejemplo, utilizando plantillas para traducir las reglas al lenguaje natural. Finalmente, como otra posibilidad más para derivar una explicación de las salidas del sistema, es posible mostrar al usuario conjuntos mínimos de elementos del ME (variables de la red bayesiana o mundos) que permiten la generación de argumentos de apoyo para la consulta, apuntando así a los elementos inciertos que juegan un papel en las derivaciones lógicas de interés.

Toda esta información proporciona a los analistas herramientas que les permiten aceptar con mayor confianza la respuesta obtenida o, por otro lado, revisar informaciones o

conocimientos que no se aplican a la situación actual.

6.4. Sumario

En este capítulo se presentó P-DAQAP, una plataforma web para facilitar el análisis del proceso argumentativo en DeLP y la argumentación probabilística a través del modelo DeLP3E. Se mostró, a través de casos de uso y ejemplos prácticos, que una herramienta de este tipo puede ser de gran utilidad para usuarios que requieran analizar dominios complejos y que tengan eventos probabilísticos asociados. Además de esa utilidad en entornos reales, también puede ser usada con fines educativos, ya que ofrece un conjunto de herramientas gráficas para facilitar la lectura y entendimiento de los procesos argumentales llevados a cabo en el formalismo DeLP. Uno de los objetivos principales de este capítulo fue el de mostrar una posible aplicación de los conceptos presentados en los capítulos anteriores para la implementación de una herramienta de software que pueda ser desplegada y utilizada.

Capítulo 7

Trabajos relacionados

En este capítulo se discutirán diferentes investigaciones y trabajos enfocados en distintas direcciones pero vinculadas en algún punto al trabajo desarrollado y presentado en esta tesis. Primero, en la Sección 7.1, se presentarán algunos trabajos que vinculan la *teoría de la argumentación* y la *teoría de la probabilidad* para modelar y razonar en dominios complejos con presencia de incertidumbre e incompletitud. Luego, en la Sección 7.2, se mencionarán algunos trabajos vinculados a la generación automática de bases de conocimiento para formalismos de argumentación estructurada, esto en relación al generador de programas PreDeLP presentado en el Capítulo 4. Por último, y en relación a la plataforma P-DAQAP presentada en el Capítulo 6, en la Sección 7.3 se presentarán algunas herramientas que fueron diseñadas para analizar la estructura de diferentes tipos de razonamiento en argumentación estructurada.

7.1. Argumentación probabilística

Como se mencionó en el Capítulo 2, existen en la literatura múltiples propuestas para modelar la incertidumbre e incompletitud a través de la argumentación y la teoría de la probabilidad. Dos de los enfoques más estudiados para argumentación abstracta son el enfoque de *constelaciones* y el enfoque *epistémico*. Para una lectura completa y detalla de estos dos enfoques, ver [Hun12]. Al igual que en DeLP3E, el uso del enfoque de constelaciones es costoso computacionalmente [FFP13]; sin embargo, existen avances en relación a técnicas de aproximación y razonamiento automatizado que pueden aplicarse [FFP13, FFF18, ACG+20]. En cuanto a las aplicaciones del enfoque de constelaciones,

se usa para modelar el grafo de argumentos entre oponentes en discusiones argumentativas [HT16], modificar y agregar diferentes perspectivas en un grafo de argumentos [HN20], y evaluaciones de planes [HKO15]. Para poder modelar la incertidumbre en la asignación de probabilidades, el enfoque de constelaciones fue extendido para soportar límites superiores e inferiores para dichas asignaciones a partir del uso de *credal sets* [ENT19].

El *Marco etiquetado para argumentación probabilística* (Labelling Framework for Probabilistic Argumentetion, en inglés) [RBG+18] es un marco que combina los dos enfoques mencionados anteriormente. Este marco, basado en una instanciación de lógica rebatible de la argumentación abstracta, modela tres representaciones de incertidumbre, donde cada representación induce a la siguiente. El marco es una combinación de dos enfoques, ya que trata tanto la incertidumbre sobre la topología de un marco de argumentación, como la incertidumbre resultante sobre si se acepta o no un argumento.

También se utilizan estos modelos para realizar o incluir *revisión de creencias*. En [Hun15] se propone la argumentación probabilística epistémica como modelo para el estado de creencias de los agentes durante un proceso de diálogo de persuasión. En resumen, el estado de creencias actual de un agente se representa como una distribución de probabilidad. Un agente puede plantear argumentos y aceptar o rechazar argumentos planteados por otros agentes. Dependiendo si un agente acepta o no un argumento, se actualiza el estado de creencias de dicho agente, modificando las probabilidades de determinadas creencias. En este mismo trabajo se proponen diferentes funciones de actualización.

En el mismo trabajo donde se presenta DeLP3E, se considera la revisión de creencias para la argumentación probabilística estructurada. La necesidad de la revisión de creencias en este modelos es que la lógica probabilística requiere suposiciones consistentes, incluso si la programación lógica rebatible no lo hace. Los autores estudian diferentes tipos de inconsistencias y formas de realizar la revisión de creencias sobre el modelo del entorno, el modelo analítico y la función de anotación. En particular, se proponen postulados para revisar el modelo analítico y la función de anotación similares a la literatura de revisión de creencias [Han97, FKRS12].

Por otro lado, también existen trabajos que investigan las relaciones entre las redes bayesianas y la argumentación probabilística. Las redes bayesianas se pueden usar para modelar razonamiento argumentativo [Vre04]. De igual manera, las redes bayesianas se pueden usar para capturar aspectos de la argumentación a través del modelo Carneades,

donde la propagación de la aplicabilidad de un argumento y la aceptabilidad del mismo se pueden expresar a través de tablas de probabilidad condicional [GGW10]. La argumentación también se puede utilizar para construir redes bayesianas [BR16, WBPR19]; como así también se pueden generar argumentos a partir de una red bayesiana [TMP+15]. Esto último implica construir argumentos que involucren un lenguaje basado en reglas para reflejar la estructura de la red.

7.2. Generadores de programas en argumentación estructurada

En relación a la generación automática de bases de conocimiento en argumentación estructurada, en [AGP+21b] se presenta un generador de programas DeLP y en [CT16] se introduce un algoritmo para generar modelos ABA [Ton14]. El motivo para desarrollar estos generadores fue el mismo que el de esta tesis: realizar pruebas exhaustivas en una *variedad de entornos*. Es importante remarcar la importancia de la variedad de entornos, ya que lo que se busca generar es un conjunto de programas que representen estructuras argumentales variadas (en tamaño y complejidad).

Comenzamos presentando el generador de programas DeLP propuesto en [AGP+21b]. En este trabajo, se generaron 9 series de programas DeLP agrupadas en tres conjuntos de diferentes tamaños en base al número de reglas que conforman cada programa y el número de literales en el cuerpo de las mismas. Aquí, para construir derivaciones finitas se asociaron las reglas y literales a un nivel definido como: i) el nivel de una regla r es igual al máximo nivel de los literales presentes en su cuerpo más 1 (las reglas que definen un hecho se consideran del nivel 1); ii) el nivel de un literal es igual al máximo nivel de las reglas que tienen al literal en su cabeza (los literales definidos por hechos solo tienen nivel 1). También se asume que el número de literales negados es igual al número de literales positivos multiplicado por un factor λ (en el experimento se asume $\lambda = 0{,}1$). Los parámetros de este generador son:

- N_F: El número de hechos en el programa $\mathcal{P}$.
- N_H (resp. $N_H * \lambda$): El número de literales positivos (resp. negativos) usados como cabeza de al menos una regla en $\mathcal{P}$.

- N_R: El máximo número de reglas que definen literales que están presentes en la cabeza de alguna regla.
- $N_S(\leq N_R)$: El máximo número de reglas estrictas que definen literales que están presentes en la cabeza de alguna regla.
- N_M: El máximo número de literales presentes en el cuerpo de una regla.
- K: El máximo nivel de los literales presentes en el programa.

El procedimiento comienza definiendo los conjuntos de literales positivos y negativos (algunos de estos pueden ser seleccionados para ser hechos). Luego, se selecciona aleatoriamene un número N_F de hechos entre literales positivos y negativos. Las funciones de selección están diseñadas para evitar la selección de dos literales complementarios o la selección del mismo literal dos veces. Luego, los hechos seleccionados se agregan al conjunto de hechos y reglas estrictas del programa. Después se procede a crear el conjunto de reglas rebatibles. Para esto último, el algoritmo genera reglas y avanza nivel por nivel en el programa. Primero, se define un número de literales que aparecerán en la cabeza de las reglas que tienen el mismo nivel. Luego, se crean los literales asociados a un nivel y las reglas que los definen. Por último, se retorna el programa formado por el conjunto de hechos y reglas estrictas, y el conjunto de reglas rebatibles. En este trabajo, no se detallan los análisis sobre las estructuras que generan los programas creados.

Con respecto al generador de programas ABA, el mismo toma como parámetro de entrada una tupla formada por $(N_s, N_a, N_{rh}, N_{rph}, N_{spb}, N_{apb})$, donde:

- N_s: El número total de sentencias en el programa.
- N_a: El número de suposiciones.
- N_{rh}: El número de sentencias distintas a ser usadas como cabezas de reglas.
- N_{rph}: El número de reglas por cabeza de regla distinta.
- N_{spb}: El número de sentencias por cuerpo de regla.
- N_{apb}: El número de suposiciones por cuerpo de regla.

A partir de dichos parámetros, se generan diferentes configuraciones sobre las cuales se ejecutan una serie de consultas. Se generaron cuatro series básicas de modelos ABA. En las tres primeras series, solo ser varía un parámetro; en la última serie se hace variar todos los parámetros de manera independiente. Los resultados reportados en este trabajo indican que los modelos generados contienen sentencias que, al ser consultadas según diferentes semánticas, arrojan respuestas del tipo: i) inmediatas, "sí" o "no"; ii) respuestas que no pudieron computarse porque la implementación de Prolog excedió los recursos o porque se excedió un límite de tiempo; y iii) respuestas que fueron computadas en tiempos mayores a 1 segundo, y que tenían estructuras de ataque anidadas entre el grafo de argumentos aceptables y los que lo atacaban. Los autores consideran que los parámetros definidos permiten generar modelos con sentencias deseables, ya que luego es posible estudiar el efecto de la variación de parámetros en la proporción de consultas que caen en las diferentes clases mencionadas.

7.3. Herramientas gráficas para argumentación

Recientemente, en la comunidad de investigación se han desarrollado nuevas herramientas de argumentación que son útiles para analizar la estructura de diferentes tipos de razonamiento. Se suele afirmar que estructurar y visualizar argumentos a través de gráficos es beneficioso y proporciona un aprendizaje más rápido, ya que la visualización de argumentos y las relaciones entre ellos se pueden ver más claramente en comparación con el texto sin formato [VvOPV08]. Además de las herramientas enfocadas a la visualización, también existen *sistemas de razonamiento automatizados* y *asistentes argumentales*. El primero realiza automáticamente el razonamiento sobre la base de la información en la base de conocimiento, mientras que el segundo ayuda a uno o más usuarios en la formulación, organización y presentación de argumentos [Ver03]. La principal diferencia entre ellos es que, mientras que los sistemas de razonamiento automatizados pueden realizar tareas de razonamiento para el usuario, los sistemas de asistencia de argumentos no razonan por sí mismos; el objetivo de los sistemas de asistencia no es *reemplazar* el razonamiento del usuario, sino *ayudar* al usuario en su proceso de razonamiento. La plataforma que presentamos en el Capítulo 6, tiene las características de un asistente argumental y un sistema de razonamiento automatizado, ya que por un lado permite la construcción automática de argumentos y el proceso argumentativo a partir de una base de conocimiento

y, por otro lado, presenta esa información a través de grafos de una manera clara para el usuario, permitiéndole analizar todo el proceso argumental. Además, es posible computar y mostrar las extensiones de las semánticas de Dung (grounded, estable, semistable, preferidas) para el grafo mostrado, como así también diferentes *intersecciones* de dichas extensiones.

Se han desarrollado herramientas de software que apoyan la construcción y visualización de los argumentos y estructuras argumentativas en varios formatos, como por ejemplo, tablas y grafos [VvOPV08]. Como resultado, existen algunas herramientas de visualización de argumentos [KBSC12], tales como: *Araucaria* [RR04], *Athena* [MR02], *Convince Me* [SR95], *Belvedere* [SWCP95], *Reason!Able* [Van02], y *Grafix* [CDL14]. Normalmente, algunas de estas herramientas producen diagramas compuestos por "cajas y flechas" en los que las premisas y las conclusiones se formulan como declaraciones. Éstas son representadas por nodos que se pueden unir mediante líneas para mostrar las inferencias, y las flechas se utilizan para indicar su dirección.

Algunas de estas herramientas (*Belvedere, Convince Me, and Reason!Able*) tienen en común que están orientadas a la educación y están diseñadas para enseñar pensamiento crítico y habilidades de discusión, probándose en entornos educativos. Sin embargo, existen algunas diferencias importantes entre ellas; por ejemplo, *Belvedere* y *Reason!Able* están totalmente diseñadas para asistir en la construcción y análisis de argumentos, mientras que *Convince Me* produce redes causales. *Reason!Able* [Van02] es un software educativo que admite el mapeo de argumentos para enseñar habilidades de razonamiento. El árbol de argumentos que se construye con esta aplicación está formado por conclusiones, razones y objeciones sobre un tópico en particular. Las razones y objeciones son objetos complejos que pueden descomponerse para mostrar el conjunto completo de premisas que los conforman. Otra herramienta es *Aracuria* [RR04], un programa para analizar argumentos que dan soporte a los usuarios para reconstruir y diagramar un argumento dado usando una interfaz de "seleccionar y arrastrar". El usuario mueve el texto del discurso que contiene un argumento como un archivo de texto a un cuadro en una ventana de la interfaz y luego resalta cada declaración (premisas y conclusiones). Cada declaración resaltada aparece como un cuadro de texto en otra ventana, y luego el usuario puede dibujar una flecha que representa cada inferencia desde un conjunto de premisas hasta una conclusión. El resultado es una cadena de argumentación que aparece como un diagrama de argumentos. Una vez que el argumento se ha diagramado completamente, se puede guardar para su

uso posterior en un formato de marcado de argumentos llamado *AML* (por sus siglas en Inglés). Aracuria fue la primera herramienta de visualización de argumentos en incorporar el uso de esquemas de argumentación.

Algunas de las herramientas más actuales para trabajar con argumentación y que ofrecen funciones de visualización son OVA+, opMAP y AVIZE. OVA+ [Ree14] (Visualización de argumentos en línea) es una interfaz en línea para el análisis de argumentos. La herramienta se creó como una respuesta al Formato de Intercambio de Argumentos [CMM+06] (AIF, por sus siglas en Inglés) – es una herramienta que permite aplicar los objetivos de AIF, es decir, la representación de argumentos y la posibilidad de intercambiar, compartir y reutilizar la estructura de los argumentos. La caracteística más importante es la posibilidad de guardar el análisis en formato AIF de manera local o en AIFdb [LBRS12] y agregarlo a un repositorio dedicado (creado anteriormente) para poder compartirlo de manera pública. OpMAP [BHMvS18] es una herramienta para visualizar espacios de opiniones multidimensionales a gran escala como mapas geográficos. Utiliza valores probabilísticos de justificación para calcular qué tan fuertemente dos opiniones son coherentes entre sí. En consecuencia, la muestra de opinión se representa como un grafo ponderado, llamado *grafo de opinión*, donde los nodos representan los vectores de opinión y los arcos muestran los valores de coherencia. Finalmente, AVIZE (Visualización y Evaluación de Argumentos) [GBR19] es una herramienta de diagramación de argumentos. Su objetivo es ayudar a los usuarios en la construcción y autoevaluación de argumentos del mundo real en el dominio de la política internacional. AVIZE ofrece un conjunto de esquemas de argumentos como bloques de construcción cognitivos para construir diagramas de argumentos.

En el próximo capítulo se presentarán las conclusiones de esta tesis, como así también los planes para trabajos futuros.

Capítulo 8

Conclusiones y trabajo futuro

Como se ha detallado a lo largo de esta tesis, el principal objetivo es el estudio de los mecanismos eficaces y eficientes para computar las respuestas en el formalismo DeLP3E. En particular, estudiar y proponer mecanismos que nos permitan aproximar las respuestas por medio de algoritmos que se basen en toda la información disponible de cada componente del modelo. Nos enfocamos en investigar este aspecto dado que, dar una respuesta exacta a una consulta en DeLP3E consiste en computar el intervalo de probabilidad asociado a la consulta realizada, y para esto es necesario recorrer cada escenario posible del ME; además, por cada uno de ellos se debe consultar por el estado de la consulta, y en base a dicho estado, actualizar los límites del intervalo. Dicho procedimiento, como se detalla en el Capítulo 3, no es aplicable en un tiempo razonable para instancias grandes. Es por esto que el objetivo principal para el desarrollo de esta tesis es el estudio y definición de técnicas, junto con una guía para seleccionar cúal de ellas aplicar, que nos permitan aproximar el valor del intervalo para una respuesta en base a toda la información disponible de cada componente del modelo DeLP3E. Esto se detalla en los Capítulo 1 y 2.

Para comenzar a avanzar hacia el objetivo establecido, en el Capítulo 3 se estudia en detalle el proceso de generar una respuesta a una consulta en un modelo DeLP3E, es decir, cómo computar el intervalo de probabilidad exacto para un literal en particular. Además, se introduce un conjunto de métricas para cuantificar la complejidad de cada componente del formalismo DeLP3E (ME, MA y Función de Anotación) y, por último, una familia de algoritmos y enfoques para aproximar el valor exacto del intervalo de probabilidad basados en las métricas definidas anteriormente. En la Sección 3.1 se describe la intratabilidad inherente al problema de computar el intervalo de probabilidad para responder a una

consulta, luego, en la Sección 3.2, se define un conjunto de métricas que nos ayudará a clasificar los componentes del modelo DeLP3E según diferentes criterios; por último, en la Sección 3.3, se presenta un conjunto de algoritmos para aproximar el valor del intervalo de probabilidad teniendo en cuenta la complejidad de los componentes de la base de conocimiento según el valor de las métricas presentadas.

En el Capítulo 4, y con la finalidad de poder evaluar el desempeño de los algoritmos propuestos en el Capítulo 3, se presentan tres generadores para crear modelos DeLP3E que nos permiten generar diferentes escenarios de complejidad de manera automática. Se presenta el análisis, diseño, y la ejecución de tres algoritmos para crear, a partir de un conjunto de parámetros, cada uno de los tres componentes de una base de conocimiento DeLP3E. Una de las principales características de estos generadores es que nos permiten crear modelos cuyos valores de métricas pueden ser ajustables en base al valor de los parámetros de entrada que guían el proceso de generación. En la Sección 4.5, se presentan los conjuntos de modelos generados que son usados en los experimentos del Capítulo 5.

En el Capítulo 5 se presenta el diseño, ejecución, y análisis de un conjunto de pruebas empíricas con el objetivo de comparar el comportamiento de los algoritmos de aproximación presentados sobre diferentes modelos DeLP3E. Dichos algoritmos son los presentados en el Capítulo 3, y los modelos DeLP3E fueron generados a través de las herramientas presentadas en el Capítulo 4. El objetivo es comparar los algoritmos de aproximación sobre diferentes modelos DeLP3E para analizar su comportamiento con respecto a diferentes criterios. Con esto, validamos y exploramos alternativas sobre la diferentes líneas del árbol de decisión presentado al final del Capítulo 3.

Como ejemplo de aplicación para las técnicas desarrolladas, en el Capítulo 6 se presenta P-DAQAP (Probabilistic Defeasible Argumentation Query Answering Platform), una plataforma web para *Respuesta a Consultas en Argumentación Rebatible Probabilística*, la cual ofrece una interfaz visual y un panel de control que facilita el análisis del proceso argumentativo llevado a cabo en la Progamación Lógica Rebatible (DeLP) [Gar97] y la argumentación probabilística a través del modelo DeLP3E [SSM+16]. El desarrollo de esta plataforma comenzó con la necesidad de contar con una herramienta que permita analizar de manera visual una base de conocimiento DeLP; luego, una vez que se desarrollaron y estudiaron los algoritmos de aproximación y los métodos de sampleo, se decidió extender la plataforma para poder trabajar con modelos DeLP3E. Dicha extensión a la plataforma se refleja en este capítulo a través de las funcionalidades que ofrece la misma. En la

Sección 6.1 se describe la motivación principal para el desarrollo de la plataforma; en la Sección 6.2 se presenta un caso de uso de la extensión DeLP3E para el análisis de ciberamenazas usando un conjunto de datos públicos; finalmente, en la Sección 6.3 se presenta la plataforma, aquí se detalla su arquitectura, el análisis sobre el programa DeLP y los grafos de Dung, y las funcionalidades que representan la extensión DeLP3E.

Finalmente, en el Capítulo 7 se mencionan y discuten algunos trabajos relacionados a los puntos principales de esta tesis.

Como contribución final del desarrollo de esta tesis, se han realizado diferentes trabajos de I+D en los que:

- Se definieron dos enfoques de aproximación: uno basado en mundos (muestreo a partir del ME) y otro basado en subprogramas (muestreo a partir del MA). También se presentaron dos algoritmos de aproximación (que implementan los enfoques mencionados), y una métrica para determinar la calidad de una aproximación obtenida.

- Se propuso un conjunto de métricas con el objetivo de poder cuantificar la complejidad y el tamaño de cada componente de un modelo DeLP3E.

- Se presentó un árbol de decisión que tiene como finalidad servir de guía para la selección del algoritmo de aproximación en base al valor de las métricas mencionadas y el costo de computarlas.

- Se desarrollaron generadores automáticos de modelos DeLP3E. Éstos tienen la capacidad de, a través de la configuración de sus parámetros, hacer variar las métricas de complejidad y tamaño de cada componente a generar. A partir de estas herramientas, se pudieron generar de una manera bien fundada conjuntos de prueba para evaluar los algoritmos de aproximación.

- Se llevaron a cabo experimentos para analizar el comportamiento de los algoritmos de aproximación.

- Se presentó P-DAQAP, una plataforma web para facilitar el análisis del proceso argumentativo y probabilístico de un modelo DeLP3E.

Trabajo futuro

En busca de continuar con el desarrollo y mejora de los procedimientos para computar respuestas en el formalismo de argumentación probabilística DeLP3E, como así también la construcción de un *benchmark* estructurado y parametrizado que permita generar y experimentar con instancias de diferentes características, resulta necesario llevar adelante más trabajo de experimentación e investigación con los componentes principales de estos modelos.

Con la intención de alcanzar ese objetivo, y como continuación del trabajo realizado en el desarrollo de esta tesis, se plantean los siguientes trabajos futuros:

- Continuar con las tareas de evaluación empírica con datos sintéticos comenzadas durante el desarrollo de esta tesis, considerando mayor diversidad de configuraciones y modelos probabilísticos. Esto nos permitirá expandir el estudio realizado de las técnicas de muestreo, como así también establecer los límites de la implementación actual en relación al costo computacional.

- Estudiar, diseñar e implementar algoritmos para reutilizar los resultados obtenidos durante el proceso de computar una respuesta en DeLP3E, para así evitar realizar cómputos redundantes frente a una situación de cambio en algún componente del modelo o una consulta modificada. Esto implica llevar a cabo un análisis acerca del proceso de cómputo, pero siguiendo un enfoque incremental que nos permita almacenar resultados parciales ya computados y estudiar cómo incorporar la nueva información evitando cómputos duplicados. Esto es un paso crucial para desarrollar implementaciones que puedan utilizarse y desplegarse en entornos reales que requieran respuestas rápidas, por ejemplo, en Sistemas de Procesamiento de Flujos, Eventos Complejos y Ciberseguridad, entre otros.

- Diseñar y construir un benchmark estructurado y parametrizado de modelos DeLP3E. Para esto, se requiere continuar con el estudio y desarrollo de cada uno de los tres generadores presentados y, en particular, el Generador de Programas PreDeLP (GPP). Esto implica abordar el estudio de los parámetros y las métricas establecidas para cada generador, con la finalidad de poder extenderlos para así tener un mayor control en la estructura generada de cada modelo, y arribar a un conjunto de instancias que será de valor para la comunidad de argumentación estructurada

probabilística como herramienta de comparación de desempeño de algoritmos de respuesta a consultas.

- En relación al punto anterior, también es posible estudiar la manera de aplicar el diseño y parametrización del GPP para trabajar con otros formalismos de razonamiento rebatible, por ejemplo, ABA y ASPIC, entre otros.
- Estudiar diferentes alternativas para proporcionar explicaciones que acompañen a las respuestas de las consultas ejecutadas. Esto es particularmente importante en los dominios de interés mencionados anteriormente, ya que un analista o experto en el dominio puede utilizar las explicaciones proporcionadas para corregir conocimiento incorrecto, tener una aceptación más fuerte de las conclusiones obtenidas y, además, llevar a cabo auditorias y análisis más detallados ya que se dispone de registros de mayor calidad del proceso que se lleva a cabo para computar las respuestas.

En relación a la plataforma P-DAQAP presentada en el Capítulo 6, es importante señalar que el desarrollo de herramientas para trabajar con teorías de la argumentación aún se encuentra principalmente en una fase experimental. Hay mucho que aprender y experimentar sobre la forma en que los argumentos y las relaciones entre ellos pueden presentarse de manera sensata y clara a los usuarios, y mucho más si se considera información probabilística. Por lo tanto, P-DAQAP es un trabajo en progreso, tanto en términos de funcionalidad como de experiencia de usuario. Un punto importante a abordar en este sentido es que la cantidad de información que se muestra puede ser bastante abrumadora, y es fundamental definir con precisión qué información se debe presentar al usuario, así como también la manera de ser presentada. En este sentido, actualmente continuamos experimentando con diferentes configuraciones para lograr una distribución más eficiente. Por lo tanto, una parte importante del trabajo actual y futuro en esta línea de investigación pasa por mejorar la herramienta y agregar funcionalidades como:

1. Exportar los grafos en formato AIF.
2. Diseñar una interfaz gráfica para introducir criterios de preferencia.
3. Diseñar una mejor interfaz gráfica para trabajar con el cálculo de semánticas.
4. Diseñar una librería externa que nos permita la distribución de la plataforma y, de esta manera, permitir que P-DAQAP sea importado por otras herramientas.

Finalmente, es esencial ofrecer las opciones para guardar, imprimir, y exportar los grafos y resultados generados en diferentes formatos.

Bibliografía

[ACG+20] Alfano, G., Calautti, M., Greco, S., Parisi, F., and Trubitsyna, I. Explainable acceptance in probabilistic abstract argumentation: Complexity and approximation. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020* (2020), D. Calvanese, E. Erdem, and M. Thielscher, Eds., pp. 33–43.

[ADRD+20] Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., et al. Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion 58* (2020), 82–115.

[AGP+21a] Alfano, G., Greco, S., Parisi, F., Simari, G. I., and Simari, G. R. Incremental computation for structured argumentation over dynamic delp knowledge bases. *Artificial Intelligence 300* (2021), 103553.

[AGP+21b] Alfano, G., Greco, S., Parisi, F., Simari, G. I., and Simari, G. R. Incremental computation for structured argumentation over dynamic delp knowledge bases. *Artif. Intell. 300* (2021), 103553.

[Als20] Alsmadi, I. *The NICE Cyber Security Framework: Cyber Security Management*. Springer Nature, 2020.

[BCG11] Baroni, P., Caminada, M., and Giacomin, M. An introduction to argumentation semantics. *The knowledge engineering review 26*, 4 (2011), 365–410.

[BDKR05] Batu, T., Dasgupta, S., Kumar, R., and Rubinfeld, R. The complexity of approximating the entropy. *SIAM J. Comput. 35*, 1 (2005), 132–150.

[BGH+14] Besnard, P., Garcia, A., Hunter, A., Modgil, S., Prakken, H., Simari, G., and Toni, F. Introduction to structured argumentation. *Arg. & Comp. 5*, 1 (2014), 1–4.

[BH08] Besnard, P., and Hunter, A. *Elements of Argumentation*. MIT Press, 2008.

[BHMvS18] Betz, G., Hamann, M., Mchedlidze, T., and von Schmettow, S. Applying argumentation to structure and visualize multi-dimensional opinion spaces. *Argument & Computation*, Preprint (2018), 1–18.

[Bon88] Bonner, A. J. A logic for hypothetical reasoning. In *Proceedings of the 7th National Conference on Artificial Intelligence, St. Paul, MN, USA, August 21-26, 1988* (1988), H. E. Shrobe, T. M. Mitchell, and R. G. Smith, Eds., AAAI Press / The MIT Press, pp. 480–484.

[BR16] Bex, F., and Renooij, S. From arguments to constraints on a bayesian network. In *Computational Models of Argument - Proceedings of COMMA 2016, Potsdam, Germany, 12-16 September, 2016* (2016), P. Baroni, T. F. Gordon, T. Scheffler, and M. Stede, Eds., vol. 287 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, pp. 95–106.

[BTG21] Buron Brarda, M. E., Tamargo, L. H., and García, A. J. Using argumentation to obtain and explain results in a decision support system. *IEEE Intelligent Systems 36*, 2 (2021), 36–42.

[Cas16] Castelvecchi, D. Can we open the black box of AI? *Nature News 538*, 7623 (2016), 20.

[CDL14] Cayrol, C., Doutre, S., and Lagasquie-Schiex, M. GRAFIX: a tool for abstract argumentation. In *Computational Models of Argument - Proceedings of COMMA 2014, Atholl Palace Hotel, Scottish Highlands, UK, September 9-12, 2014* (2014), S. Parsons, N. Oren, C. Reed, and F. Cerutti, Eds., vol. 266 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, pp. 453–454.

[CMM+06] Chesñevar, C. I., McGinnis, J., Modgil, S., Rahwan, I., Reed, C., Simari, G. R., South, M., Vreeswijk, G., and Willmott, S. Towards an argument interchange format. *Knowl. Eng. Rev. 21*, 4 (2006), 293–316.

[CT16] Craven, R., and Toni, F. Argument graphs and assumption-based argumentation. *Artif. Intell. 233* (2016), 1–59.

[CVG16] Cerutti, F., Vallati, M., and Giacomin, M. jargsemsat: an efficient off-the-shelf solver for abstract argumentation frameworks. In *Proc. KR* (2016).

[Dun95] Dung, P. M. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell. 77*, 2 (1995), 321–358.

[ENT19] Espinoza, M. M., Nieves, J. C., and Tacla, C. A. An imprecise probability approach for abstract argumentation based on credal sets. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 15th European Conference, ECSQARU 2019, Belgrade, Serbia, September 18-20, 2019, Proceedings* (2019), G. Kern-Isberner and Z. Ognjanovic, Eds., vol. 11726 of *Lecture Notes in Computer Science*, Springer, pp. 39–49.

[FFF18] Fazzinga, B., Flesca, S., and Furfaro, F. Credulous and skeptical acceptability in probabilistic abstract argumentation: complexity results. *Intelligenza Artificiale 12*, 2 (2018), 181–191.

[FFP13] Fazzinga, B., Flesca, S., and Parisi, F. On the complexity of probabilistic abstract argumentation. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013* (2013), F. Rossi, Ed., IJCAI/AAAI, pp. 898–904.

[FKRS12] Falappa, M. A., Kern-Isberner, G., Reis, M. D. L., and Simari, G. R. Prioritized and non-prioritized multiple change on belief bases. *J. Philos. Log. 41*, 1 (2012), 77–113.

[FWSJ08] Frigault, M., Wang, L., Singhal, A., and Jajodia, S. Measuring network security using dynamic bayesian network. In *Proceedings of the 4th*

ACM Workshop on Quality of Protection, QoP 2008, Alexandria, VA, USA, October 27, 2008 (2008), A. Ozment and K. Stølen, Eds., ACM, pp. 23–30.

[GA19] Gunning, D., and Aha, D. W. Darpa's explainable artificial intelligence (XAI) program. *AI Mag. 40*, 2 (2019), 44–58.

[Gar97] García, A. J. *La Programación en Lógica Rebatible: su definición teórica y computacional*. PhD thesis, Tesis de Maestría, Departamento de Ciencias de la Computación, Universidad Nacional del Sur, 1997.

[GBR19] Green, N. L., Branon, M., and Roosje, L. Argument schemes and visualization software for critical thinking about international politics. *Arg. & Comp. 10*, 1 (2019), 41–53.

[GEYF19] Graf, J., Eastwood, S., Yanushkevich, S. N., and Ferber, R. Risk inference models for security applications. In *Eighth International Conference on Emerging Security Technologies, EST 2019, Colchester, UK, July 22-24, 2019* (2019), IEEE, pp. 1–6.

[GGW10] Grabmair, M., Gordon, T. F., and Walton, D. Probabilistic semantics for the carneades argument model using bayesian networks. In *Computational Models of Argument: Proceedings of COMMA 2010, Desenzano del Garda, Italy, September 8-10, 2010* (2010), P. Baroni, F. Cerutti, M. Giacomin, and G. R. Simari, Eds., vol. 216 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, pp. 255–266.

[GS04] García, A. J., and Simari, G. R. Defeasible logic programming: An argumentative approach. *TPLP 4*, 1-2 (2004), 95–138.

[Gun17] Gunning, D. Explainable Artificial Intelligence (XAI). Defense Advanced Research Projects Agency (DARPA), 2017.

[Hae09] Haenni, R. Probabilistic argumentation. *J. Appl. Log. 7*, 2 (2009), 155–176.

[Hal05] Halpern, J. Y. *Reasoning about uncertainty*. MIT Press, 2005.

[Han97] Hansson, S. O. Semi-revision (invited paper). *J. Appl. Non Class. Logics 7*, 2 (1997).

[HKO15] Hung, N. D., Kaisaard, S., and Oo, S. H. P. Probabilistic argumentation for service restoration in power distribution networks. In *Recent Advances and Future Prospects in Knowledge, Information and Creativity Support Systems - Selected Revised Papers from the Tenth International Conference on Knowledge, Information and Creativity Support Systems (KICSS 2015), 12-14 November 2015, Phuket, Thailand* (2015), T. Theeramunkong, A. M. J. Skulimowski, T. Yuizono, and S. Kunifuji, Eds., vol. 685 of *Advances in Intelligent Systems and Computing*, Springer, pp. 54–68.

[HN20] Hunter, A., and Noor, K. Aggregation of perspectives using the constellations approach to probabilistic argumentation. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020* (2020), AAAI Press, pp. 2846–2853.

[HPP+21] Hunter, A., Polberg, S., Potyka, N., Rienstra, T., and Thimm, M. Probabilistic argumentation: A survey. *Handbook of Formal Argumentation 2* (2021).

[HSS08] Hagberg, A. A., Schult, D. A., and Swart, P. J. Exploring network structure, dynamics, and function using networkx. In *Proceedings of the 7th Python in Science Conference* (Pasadena, CA USA, 2008), G. Varoquaux, T. Vaught, and J. Millman, Eds., pp. 11–15.

[HT16] Hunter, A., and Thimm, M. Optimization of dialectical outcomes in dialogical argumentation. *Int. J. Approx. Reason. 78* (2016), 73–102.

[Hun12] Hunter, A. Some foundations for probabilistic abstract argumentation. In *Computational Models of Argument - Proceedings of COMMA 2012, Vienna, Austria, September 10-12, 2012* (2012), B. Verheij, S. Szeider, and S. Woltran, Eds., vol. 245 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, pp. 117–128.

[Hun13] Hunter, A. A probabilistic approach to modelling uncertain logical arguments. *Int. J. Approx. Reason. 54*, 1 (2013), 47–81.

[Hun15] Hunter, A. Modelling the persuadee in asymmetric argumentation dialogues for persuasion. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015* (2015), Q. Yang and M. J. Wooldridge, Eds., AAAI Press, pp. 3055–3061.

[KBSC12] Kirschner, P. A., Buckingham-Shum, S. J., and Carr, C. S. *Visualizing argumentation: Software tools for collaborative and educational sensemaking*. Springer Science & Business Media, 2012.

[KF09] Koller, D., and Friedman, N. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press, 2009.

[KLK20] Kuppa, A., and Le-Khac, N.-A. Black box attacks on explainable artificial intelligence (XAI) methods in cyber security. In *Proc. IJCNN* (2020), IEEE, pp. 1–8.

[Klo94] Kloks, T. *Treewidth, Computations and Approximations*, vol. 842 of *Lecture Notes in Computer Science*. Springer, 1994.

[KM93] Kohlas, J., and Monney, P. Probabilistic assumption-based reasoning. In *UAI '93: Proceedings of the Ninth Annual Conference on Uncertainty in Artificial Intelligence, The Catholic University of America, Providence, Washington, DC, USA, July 9-11, 1993* (1993), D. Heckerman and E. H. Mamdani, Eds., Morgan Kaufmann, pp. 485–491.

[LBRS12] Lawrence, J., Bex, F., Reed, C., and Snaith, M. AIFdb: Infrastructure for the argument web. In *COMMA* (2012), pp. 515–516.

[LGSS21] Leiva, M., García, A., Shakarian, P., and Simari, G. Probabilistic defeasible logic programming: Towards explainable and tractable query answering. *Electronic Proceedings in Theoretical Computer Science, EPTCS 345* (September 2021), 77–79. 37th International Conference on Logic Programming (Technical Communications), ICLP 2021 ; Conference date: 20-09-2021 Through 27-09-2021.

[LL89] Laskey, K. B., and Lehner, P. E. Assumptions, beliefs and probabilities. *Artif. Intell. 41*, 1 (1989), 65–77.

[Llo87] Lloyd, J. W. *Foundations of Logic Programming, 2nd Edition.* Springer, 1987.

[LON11] Li, H., Oren, N., and Norman, T. J. Probabilistic argumentation frameworks. In *Theorie and Applications of Formal Argumentation - First International Workshop, TAFA 2011. Barcelona, Spain, July 16-17, 2011, Revised Selected Papers* (2011), S. Modgil, N. Oren, and F. Toni, Eds., vol. 7132 of *Lecture Notes in Computer Science*, Springer, pp. 1–16.

[Lou86] Loui, R. Defeat among arguments: a system of defeasible inference. *Computational Intelligence* (1986), 100–106.

[LSG+19] Leiva, M. A., Simari, G. I., Gottifredi, S., García, A. J., and Simari, G. R. DAQAP: defeasible argumentation query answering platform. In *Flexible Query Answering Systems - 13th International Conference, FQAS 2019, Amantea, Italy, July 2-5, 2019, Proceedings* (2019), A. Cuzzocrea, S. Greco, H. L. Larsen, D. Saccà, T. Andreasen, and H. Christiansen, Eds., vol. 11529 of *Lecture Notes in Computer Science*, Springer, pp. 126–138.

[LSG20] Leiva, M. A., Simari, G. I., and Garcia, A. Towards effective and efficient approximate query answering in probabilistic delp (short paper). In *Proceedings of the Workshop on Advances In Argumentation In Artificial Intelligence 2020 co-located with the 19th International Conference of the Italian Association for Artificial Intelligence (AIxIA 2020), Online, November 25-26, 2020* (2020), B. Fazzinga, F. Furfaro, and F. Parisi, Eds., vol. 2777 of *CEUR Workshop Proceedings*, CEUR-WS.org, pp. 103–109.

[LSSS19] Leiva, M. A., Simari, G. I., Simari, G. R., and Shakarian, P. Cyber threat analysis with structured probabilistic argumentation. In *Proceedings of the 3rd Workshop on Advances In Argumentation In Artificial Intelligence co-located with the 18th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2019), Rende, Italy, November 19-22, 2019* (2019), F. Santini and A. Toniolo, Eds., vol. 2528 of *CEUR Workshop Proceedings*, CEUR-WS.org, pp. 50–64.

[MGS12] Martinez, M. V., García, A. J., and Simari, G. R. On the use of presumptions in structured defeasible reasoning. In *Computational Models of Argument - Proceedings of COMMA 2012, Vienna, Austria, September 10-12, 2012* (2012), B. Verheij, S. Szeider, and S. Woltran, Eds., vol. 245 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, pp. 185–196.

[MKRC19] Misuri, A., Khakzad, N., Reniers, G., and Cozzani, V. A bayesian network methodology for optimal security management of critical infrastructures. *Reliab. Eng. Syst. Saf. 191* (2019).

[MP14] Modgil, S., and Prakken, H. The *ASPIC*$^+$ framework for structured argumentation: a tutorial. *Argument Comput. 5*, 1 (2014), 31–62.

[MR02] Magnusson, C., and Rolf, B. Developing the art of argumentation-a software approach. In *International Conference on Argumentation* (2002).

[Nil86] Nilsson, N. J. Probabilistic logic. *Artif. Intell. 28*, 1 (1986), 71–87.

[Nut92] Nute, D. Basic defeasible logic. *Intensional logics for programming* (1992).

[Par99] Parsons, S. Cognitive carpentry: A blueprint for how to build a person by john l. pollock, MIT press, 1995, ISBN 0-262-16152-4, pp 377, £29.50. *Knowl. Eng. Rev. 14*, 1 (1999), 91–95.

[Par01] Parsons, S. *Qualitative methods for reasoning under uncertainty.* MIT Press, 2001.

[Pea88] Pearl, J. Bayesian networks.

[Poo85] Poole, D. On the comparison of theories: Preferring the most specific explanation. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence. Los Angeles, CA, USA, August 1985* (1985), A. K. Joshi, Ed., Morgan Kaufmann, pp. 144–147.

[PTSM21] Paredes, J., Teze, J. C., Simari, G. I., and Martinez, M. V. On the importance of domain-specific explanations in ai-based cybersecurity systems (technical report). *CoRR abs/2108.02006* (2021).

[RBG+18] Riveret, R., Baroni, P., Gao, Y., Governatori, G., Rotolo, A., and Sartor, G. A labelling framework for probabilistic argumentation. *Ann. Math. Artif. Intell. 83*, 1 (2018), 21–71.

[Ree14] Reed, M. J. J. L. C. Ova+: An argument analysis interface. In *Computational Models of Argument: Proceedings of COMMA* (2014), vol. 266, p. 463.

[RR04] Reed, C., and Rowe, G. Araucaria: Software for argument analysis, diagramming and representation. *Int. J. Artif. Intell. Tools 13*, 04 (2004), 961–979.

[SGCS03] Stolzenburg, F., García, A. J., Chesñevar, C. I., and Simari, G. R. Computing generalized specificity. *J. Appl. Non Class. Logics 13*, 1 (2003), 87–113.

[Sha48] Shannon, C. E. A mathematical theory of communication. *Bell Syst. Tech. J. 27*, 3 (1948), 379–423.

[SL92] Simari, G. R., and Loui, R. P. A mathematical treatment of defeasible reasoning and its implementation. *Artif. Intell. 53*, 2-3 (1992), 125–157.

[SR95] Schank, P., and Ranney, M. Improved reasoning with convince me. In *Conference Companion on Human Factors in Computing Systems* (1995), ACM, pp. 276–277.

[SR09] Simari, G. R., and Rahwan, I., Eds. *Argumentation in Artificial Intelligence*. Springer, 2009.

[SSEJJD12] Shameli-Sendi, A., Ezzati-Jivan, N., Jabbarifar, M., and Dagenais, M. Intrusion response systems: survey and taxonomy. *Int. J. Comput. Sci. Netw. Secur 12*, 1 (2012), 1–14.

[SSF16] Simari, G. I., Shakarian, P., and Falappa, M. A. A quantitative approach to belief revision in structured probabilistic argumentation. *Ann. Math. Artif. Intell. 76*, 3-4 (2016), 375–408.

[SSM+14] Shakarian, P., Simari, G. I., Moores, G., Parsons, S., and Falappa, M. A. An argumentation-based framework to address the attribution problem in cyber-warfare. In *Proc. CyberSecurity* (2014), ASE.

[SSM+16] Shakarian, P., Simari, G. I., Moores, G., Paulo, D., Parsons, S., Falappa, M. A., and Aleali, A. Belief revision in structured probabilistic argumentation. *AMAI 78*, 3-4 (2016), 259–301.

[SSR13] Shakarian, P., Shakarian, J., and Ruef, A. *Introduction to cyber-warfare: A multidisciplinary approach*. Newnes, 2013.

[SWCP95] Suthers, D., Weiner, A., Connelly, J., and Paolucci, M. Belvedere: Engaging students in critical discussion of science and public policy issues. In *Proc. WCAIE* (1995), Washington, DC, pp. 266–273.

[Thi12] Thimm, M. A probabilistic semantics for abstract argumentation. In *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012* (2012), L. D. Raedt, C. Bessiere, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, and P. J. F. Lucas, Eds., vol. 242 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, pp. 750–755.

[TMP+15] Timmer, S. T., Meyer, J. C., Prakken, H., Renooij, S., and Verheij, B. Explaining bayesian networks using argumentation. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty - 13th European Conference, ECSQARU 2015, Compiègne, France, July 15-17, 2015. Proceedings* (2015), S. Destercke and T. Denoeux, Eds., vol. 9161 of *Lecture Notes in Computer Science*, Springer, pp. 83–92.

[Ton14] Toni, F. A tutorial on assumption-based argumentation. *Argument Comput. 5*, 1 (2014), 89–117.

[Van02] Van Gelder, T. Argument mapping with reason! able. *The American Philosophical Association Newsletter on Philosophy and Computers 2*, 1 (2002), 85–90.

[Ver03] Verheij, B. Artificial argument assistants for defeasible argumentation. *Artif. Intell. 150*, 1-2 (2003), 291–324.

[Ver14] Verheij, B. Arguments and their strength: Revisiting pollock's anti-probabilistic starting points. In *Computational Models of Argument - Procee-*

dings of COMMA 2014, Atholl Palace Hotel, Scottish Highlands, UK, September 9-12, 2014 (2014), S. Parsons, N. Oren, C. Reed, and F. Cerutti, Eds., vol. 266 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, pp. 433–444.

[VM20] Viganò, L., and Magazzeni, D. Explainable security. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)* (2020), IEEE, pp. 293–300.

[Vre04] Vreeswijk, G. Argumentation in bayesian belief networks. In *Argumentation in Multi-Agent Systems, First International Workshop, ArgMAS 2004, New York, NY, USA, July 19, 2004, Revised Selected and Invited Papers* (2004), I. Rahwan, P. Moraitis, and C. Reed, Eds., vol. 3366 of *Lecture Notes in Computer Science*, Springer, pp. 111–129.

[VvOPV08] Van den Braak, S. W., van Oostendorp, H., Prakken, H., and Vreeswijk, G. A. A critical review of argument visualization tools: Do users become better reasoners? In *Proc. CMNA* (2008), pp. 67–75.

[WBPR19] Wieten, R., Bex, F., Prakken, H., and Renooij, S. Supporting discussions about forensic bayesian networks using argumentation. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Law, ICAIL 2019, Montreal, QC, Canada, June 17-21, 2019* (2019), ACM, pp. 143–152.

[WS14] Wirth, C., and Stolzenburg, F. David poole's specificity revised. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014* (2014), C. Baral, G. D. Giacomo, and T. Eiter, Eds., AAAI Press.

[XLO+10] Xie, P., Li, J. H., Ou, X., Liu, P., and Levy, R. Using bayesian networks for cyber security analysis. In *Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2010, Chicago, IL, USA, June 28 - July 1 2010* (2010), IEEE Computer Society, pp. 211–220.