Universidad Nacional del Sur

Tesis de Doctor en Ciencias de la Computación

Formalismos de argumentación en especificación de agentes autónomos.

Sebastian Gottifredi

Bahía Blanca Argentina

2012

Prefacio

Esta Tesis se presenta 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 otra. La misma contiene los resultados obtenidos en investigaciones llevadas a cabo en el ámbito del Departamento de Ciencias e Ingeniería de la Computación durante el período comprendido entre el 1 de abril de 2007 y el 12 de marzo de 2012, bajo la dirección del Dr. Alejandro J. García, Profesor Asociado del Departamento de Ciencias e Ingeniería de la Computación y del Dr. Guillermo R. Simari, Profesor Titular del Departamento de Ciencias e Ingeniería de la Computación.

.....

Sebastian Gottifredi
sg@cs.uns.edu.ar
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Bahía Blanca, 12 de marzo de 2012

UNIVERSIDAD NACIONAL DEL SUR
Secretaría General de Posgrado y Educación Continua

La presente tesis ha sido aprobada el .../.../..., mereciendo la calificación de .....(.....)

Agradecimientos

En primer lugar quiero agradecer a mis directores los Dres. Alejandro García y Guillermo Simari por su guía, dedicación, consejo, enseñanza y apoyo durante el desarrollo de esta tesis. Ellos me han brindado las herramientas necesarias y focalizado mi creatividad para poder madurar técnica y humanamente en este trabajo. Además quiero agradecerles por preocuparse por mi, tanto laboral como humanamente, y por abrirme las puertas para ser parte del LIDIA en todo sentido.

Luego quiero agradecer a todas las personas con las que compartí actividades de investigación durante estos cinco años. Entre ellos a Nicolas Rotstein, Mariano Tucat, Patrick Krümpelmann, Matthias Thimm, Gabrielle Kern-Iberner, Diego García y Paula Gonzalez. He aprendido y disfrutado mucho de las interacciones y el trabajo que hemos realizado en conjunto.

Agradezco a todos los integrantes del Departamento de Ciencias e Ingeniería de la Computación (DCIC) de la Universidad Nacional del Sur (UNS), por formarme como profesional, brindarme las herramientas para esta etapa doctoral, y hacerme sentir cómodo en mi lugar de trabajo. También quiero agradecer a todas las cátedras que de las que he sido parte por ayudarme en mi formación docente, la cual tuvo un fuerte impacto en la forma en que desarrollé esta tesis. En este sentido quiero agradecer especialmente a Diego Martínez, Laura Cobo, Jessica Carballido, Sebastián Escarza, Alejandro García y Luciano Tamargo.

Especialmente quiero agradecer a mi novia Noni, quien ha sido una de las grandes motivaciones para alcanzar los objetivos de mi vida. Ella ha sido mi compañera durante toda esta etapa, conteniéndome y ayudándome en los momentos duros, haciéndome disfrutar más los éxitos, y haciendo que todo este camino sea más lindo, más feliz. Además, quiero agradecerle por todas las discusiones técnicas y sus opiniones con respecto a varios de los trabajos que he realizado durante esta tesis.

Otro de los principales agradecimientos es para mi familia: mis padres, tíos, abuelos y hermana. En primer lugar por toda su dedicación en nuestra educación y por darme la oportunidad de vivir una infancia, adolescencia y adultez felices. En segundo lugar porque siempre motivaron mi creatividad y me enseñaron que uno tiene que intentar hacer las cosas bien.

Uno de los principales factores por los que he llegado a terminar esta etapa es que tuve la suerte de encontrar varios grupos de amigos fantásticos. Con ellos comparto gran parte de mis gustos, pasiones, sueños, hobbies y la vida. Por un lado, en especial quiero agradecer a mis amigos de SRI: José, Sebman, Tin, Christian, Gastón, Martín, Osky, Germán, Juli, David, Leo, Zorzi, El negro y Santi. Por otro lado a mis amigos de la universidad y la sala de becarios: Lucho, Mauro, Nico, Martín, Fer, Ana, Cristian, Seba, Diego, Tucky, Juli, Kcho, Ro, Ariel y Santi. Todos ellos son grandes responsables de que mi motivación y moral siempre se haya mantenido alta para alcanzar mis metas profesionales.

Quiero agradecer también a Tucky, Leo, Diego, Manu, Iñaki, Fer, Emi, Fede y Daniel, con quienes tuve la oportunidad de compartir la experiencia de participar en competencias relacionadas con los temas desarrollados en esta tesis. En especial, por darme la oportunidad de enseñar y aprender en conjunto cómo aplicar muchas de las herramientas que hemos estudiado, de manera práctica y en entornos de alta complejidad.

Por último quiero agradecer al Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) y la Universidad Nacional del Sur (UNS) por confiar en mí y darme el sustento requerido para poder llevar a cabo las investigaciones necesarias para el desarrollo de mi trabajo doctoral. Esta tesis no hubiera sido posible sin el apoyo de estas instituciones.

¡Gracias a todos!

Gotti

Resumen

En esta tesis se propone un lenguaje de programación de agentes cognitivos racionales y formalismos de argumentación basados en la noción de tipo de argumento. Estos formalismos argumentativos extenderán a aproximaciones existentes en la literatura de manera tal que permitan representar argumentos tipados y definir relaciones de conflicto, herencia y preferencia entre los tipos de argumento. En particular, el primero de estos formalismos se basará en los Marcos Argumentativos Abstractos, mientras que el segundo constituirá una extensión de la Programación en Lógica Rebatible (DeLP).

El lenguaje de programación de agentes propuesto en esta tesis permitirá la especificación declarativa de agentes a través de sus componentes mentales, y estará provisto de una semántica formal. A diferencia de otros lenguajes de la literatura, el lenguaje propuesto también permitirá especificar diferentes clases de metas y representar creencias y metas que pueden estar conflicto. Para tal fin, empleará los formalismos argumentativos desarrollados en esta tesis, en particular, utilizando los tipos de argumento para identificar los diferentes componentes mentales de un agente. Por lo tanto, un agente razonará con argumentos para sus percepciones, creencias, y las diferentes clases de metas que puede especificar para decidir, ante la presencia de conflictos, qué información prevalecerá.

Los formalismos argumentativos presentados en esta tesis propondrán un mecanismo novedoso y general para modelar sistemas con múltiples tipos. De esta manera, no sólo serán adecuados para modelar los tipos de argumento en el lenguaje de programación de agentes propuesto, sino que podrán ser utilizados para modelar tipos de argumento en otros dominios como diálogos argumentativos, sistemas con múltiples fuentes de información, o sistemas con información basada en valores. En esta tesis se abordará la definición, el análisis y las propiedades de estos formalismos argumentativos, así como también del lenguaje de programación de agentes propuesto.

Abstract

This thesis proposes a programming language for rational congnitive agents, along with argumentation formalisms based on the notion of argument type. These argumentation formalisms will extend already existing approaches in order to represent typed arguments and define conflict, inheritance and preference relations among argument types. In particular, the first formalism will be based on Abstract Argumentation Frameworks, whereas the second will be an extension of Defeasible Logic Programming (DeLP).

The proposed agent programming language will allow for a declarative specification of agents through their mental components, and will be provided of a formal semantics. Unlike other approaches proposed in the literature, this programming language will be able to consider different goal types and conflicting beliefs and goals. To accomplish this, it will use the argumentation formalisms developed in this thesis, particularly by identifying an agent's mental components through argument types. Therefore, an agent will reason with arguments for its perceptions, beliefs and goals to decide, in the presence of conflicts, which information prevails.

The argumentation formalisms developed in this thesis will constitute a novel and general mechanism for multi-typed systems modeling. Thus, they will be appropriate to model argument types in the proposed agent programming language, but they could also be used in other domains such as argumentation dialogues, multi-source information systems, and value-based information systems. This thesis will address the definition, analysis and properties of these argumentation formalisms and the proposed agent programming language.

Índice general

Capítulo 1

Introducción

En esta tesis se propone un lenguaje de programación de agentes cognitivos racionales llamado ArgAPL, y dos formalismos de argumentación basados en tipos. Estos nuevos formalismos extienden a los sistemas de argumentación existentes, permitiendo definir tipos de argumentos y luego especificar relaciones de conflicto, herencia y preferencia entre los tipos definidos. Como se explicará a continuación, el lenguaje ArgAPL permite la especificación de agentes a través de sus componentes mentales (como creencias y metas). Pero además, a diferencia de otros lenguajes propuestos en la literatura, ArgAPL permite especificar diferentes clases de metas y dispone de un lenguaje de representación de conocimiento que permite especificar creencias y metas que pueden estar en conflicto. Los formalismos argumentativos propuestos proveerán a los agentes especificados con ArgAPL la capacidad de razonar y decidir ante la presencia de información conflictiva. Esta tesis abordará la definición, el análisis y las propiedades de este nuevo lenguaje de programación de agentes, así como también de los nuevos formalismos argumentativos. Además, se incluirá la descripción de la semántica formal para el lenguaje ArgAPL.

Lenguajes para Programación de Agentes

En las últimas dos décadas ha habido un enorme desarrollo en el área de Agentes y Sistemas Multi-Agente. Una muestra de esto es que *AAMAS (Autonomous Agents and MultiAgent Systems)* se ha convertido en una de las conferencias más importantes de Ciencias de la Computación. Actualmente hay una aceptación muy grande al desarrollo de sistemas basados en agentes, y estos sistemas son generalmente implementados utilizando

lenguajes de programación de propósito general como Java, C o Prolog. No obstante, en la literatura se reconoce la importancia de contar con lenguajes dedicados, ya que facilitan la implementación de agentes mediante constructores para programar los agentes directamente en términos de lo establecido por las teorías formales de agentes. Aunque la propuesta de lenguajes específicos para la programación de agentes se ha mantenido a un ritmo constante en la literatura, aún no existe una aceptación completa de uno de ellos por parte de la comunidad. Probablemente esto se deba a que no existe un lenguaje de programación de agentes (APL, del inglés *agent programming language*) con la madurez necesaria, o que brinde herramientas específicas que aventajen claramente a los lenguajes de propósito general.

Entre las propuestas más destacas de APLs se encuentran AGENT-O, AgentSpeak(L), 3APL, 2APL, GOAL, Jason, JACK, Jadex e IMPACT. En el Capítulo 2 se incluye una breve reseña de la evolución de los APLs, y se mostrará que varios de estos lenguajes permiten programar a un agente mediante la especificación de sus componentes mentales (creencias, metas, etc). Esa clase de APLs provee características muy importantes, las cuales están aceptadas en la literatura: permiten especificaciones declarativas, proveen una semántica formal, y permiten el desarrollo de agentes verificables. Sin embargo, estos APLs presentan ciertos puntos débiles: no permiten representar información conflictiva para especificar sus componentes mentales, no suelen modelar diferentes tipos de metas, y los conflictos que surgen de la interacción entre los componentes mentales del agente generalmente se abordan mediante la remoción de información en lugar de la confrontación de razones a favor o en contra.

La contribución más importante de esta tesis es proponer un APL que mantiene las características deseables de otros APLs de la literatura (esto es, permite especificaciones declarativas, provee una semántica formal, y permite el desarrollo de agentes verificables); y además, propone una solución a los puntos débiles arriba mencionados utilizando un lenguaje de representación y un mecanismo de razonamiento basado en los nuevos formalismos argumentativos desarrollados en esta tesis.

Agentes y Argumentación

Una de las ventajas que presentan los formalismos de argumentación es que proveen un mecanismo de razonamiento automático que permite considerar información incompleta, contradictoria e incierta. En este tipo de razonamiento automático se consideran

argumentos que sustentan conclusiones, y un argumento puede estar en conflicto con otros argumentos. De esta forma, durante el proceso argumentativo todos los argumentos en conflicto son analizados para luego determinar qué conclusiones están aceptadas.

Desde el punto de vista teórico la aplicación de argumentación en el contexto de agentes inteligentes es ampliamente reconocida en la literatura de inteligencia artificial, y se han propuesto numerosos formalismos que aplican o combinan argumentación y agentes, por ejemplo: [BCD07, ADL08, RGS07, RA06]. Sin embargo, ninguno de los formalismos existentes propone un APL concreto, ni tampoco propone la aplicación de argumentación a un APL existente en la literatura. En esta tesis se proponen nuevos formalismos de argumentación que permiten la definición de un APL específico para programar agentes basados en componentes mentales, con lo cual se dispondrá de un nuevo APL que utiliza un formalismo de argumentación de manera concreta.

En esta tesis se pretende avanzar un paso más en la aplicación de argumentación al área de agentes, ya que ArgAPL permitirá construir argumentos a partir de la información utilizada para especificar los componentes mentales del agente. De esta forma, un agente ArgAPL razonará con argumentos que sustentan sus *percepciones, creencias, metas de logro* y *metas de mantenimiento*. Los argumentos construidos en ArgAPL podrán darse soporte unos a otros (como por ejemplo un argumento para una creencia motiva una meta) y también podrán estar en conflicto entre sí (como en el caso de contar con dos argumentos que sustentan metas incompatibles).

Por ejemplo, considere un agente cuya tarea es acomodar cajas en un depósito que cuenta con varias habitaciones. Suponga que el agente tiene un argumento que sustente la creencia de que la caja C_2 se halla en la habitación H_1, y otro argumento para establecer que la caja C_1 está dentro de C_2. Estos argumentos darán soporte al argumento que sustente que C_1 también se encuentra en la habitación H_1. Una meta de mantenimiento M para el agente podría ser que la caja C_1 deba permanecer en la habitación H_1 por cuestiones de temperatura. Una meta de logro L para el agente podría ser que la caja C_2 deba estar en la habitación H_2 para su uso. La meta L se considerará alcanzada cuando el agente tenga una creencia que indique que C_2 está en H_2. En este escenario, el argumento que indica que C_2 está en H_1 generará la necesidad de llevar a cabo un plan para lograr la meta L. Sin embargo, como C_1 está dentro de C_2 y el agente tiene una meta de mantenimiento M que obliga a dejar a C_1 en la habitación H_1, el argumento para la meta M estará en conflicto con el argumento que sustenta la meta de logro L.

Para resolver este conflicto podría ejecutarse previamente un plan para remover a C_1 del interior de C_2. Una vez concretado el plan para transportar a C_2, la caja C_2 se hallará en la habitación H_2. Por lo tanto, el argumento que sustenta que C_2 se encuentra en H_2 estará en conflicto con la meta de logro L, con lo cual el agente podrá descartar la meta L dado que ya la ha resuelto.

Los conflictos entre los argumentos cumplirán un rol clave al momento de modelar situaciones del mundo que el agente quiere mantener (metas de mantenimiento). Esto se debe a que los argumentos para este tipo de metas estarán en conflicto con aquellos argumentos para metas de logro que pongan en riesgo la condición que los primeros quieren mantener. El lenguaje ArgAPL proveerá un mecanismo de protección proactiva para las metas de mantenimiento del agente. Adicionalmente, el mecanismo argumentativo ayudará al agente a razonar con la información conflictiva que surja de la dinámica de su ejecución. Por ejemplo, un argumento para una meta de logro que ya fue alcanzada será derrotado por la creencia que indica esta situación. Los formalismos argumentativos propuestos en esta tesis serán un pilar fundamental para realizar inferencias en ArgAPL y así establecer cuáles son las conclusiones aceptadas para el agente.

El lenguaje 3APL es uno de los más desarrollados y más promisorios que se encuentra en la literatura. Es por esto que fue tomado como punto de partida para el desarrollo de ArgAPL. Además, a lo largo esta tesis se lo utilizará como referencia para contrastar con las características de ArgAPL. Como se detallará en el Capítulo 2, 3APL presenta características deseables para un lenguaje de programación de agentes cognitivos, pero también presenta limitaciones importantes.

Por ejemplo, considere un escenario donde un agente se encuentra en una ciudad y quiere viajar a otras ciudades para visitar a sus amigos. El viaje entre ciudades le insume energía al agente, y éste desea que su energía nunca se reduzca por debajo de las diez unidades. Por otra parte, los amigos del agente pueden mudarse de una ciudad a otra. De esta manera, las metas del agente estarán condicionadas por las creencias que determinan la ubicación de sus amigos. Además, dado que el agente no puede viajar a dos ciudades a la vez, estas metas podrán estar en conflicto. En 3APL no es posible proveer una implementación sencilla para el agente de este escenario. Esto se debe a que en 3APL no es posible expresar información conflictiva, las metas son incondicionales, y no se cuenta con metas de mantenimiento. Por lo tanto, para implementar el agente anteriormente descripto deberían simularse estos elementos faltantes mediante el uso de

otros componentes del lenguaje, lo cual es una tarea compleja. Además, la programación del agente dejaría de ser declarativa, lo cual constituye uno de los objetivos primordiales de este tipo de lenguajes.

En contraste, como se mencionó anteriormente, en ArgAPL será posible representar tanto creencias como metas de logro y metas de mantenimiento a través del uso de reglas que permitirán, entre otras cosas, expresar metas condicionales. Además, como ArgAPL permite representar información potencialmente contradictoria, en el escenario del párrafo anterior se podrá expresar que las metas del agente para ir a distintas ciudades están en conflicto. Por otra parte, la meta para mantener la energía del agente mayor a diez unidades estará en conflicto con las metas para ir a ciudades que impliquen dejar al agente con menos energía. Como se mostrará más adelante, el mecanismo argumentativo de ArgAPL permitirá al agente resolver los conflictos y decidir cuáles de estas metas perseguir finalmente.

Una característica deseable de un APL es que disponga de una semántica formal. En el Capítulo 6 se presentará una especificación formal para la semántica operacional de ArgAPL. Esta mostrará las transiciones de estado del agente a partir de la selección de planes que le permitan alcanzar sus metas y la posterior ejecución de las acciones de esos planes. Por otra parte, esta semántica mostrará formalmente cómo el agente utilizará el mecanismo argumentativo para elegir sus planes a partir de sus creencias y metas de logro, cómo protegerá sus metas de mantenimiento de planes y acciones que las amenacen, y cómo descartará los planes cuando sus metas de logro ya no sean factibles.

Sistemas Argumentativos Tipados

Para desarrollar un lenguaje de programación de agentes como ArgAPL es necesario contar con un sistema argumentativo capaz de identificar grupos de argumentos con ciertas características comunes, de manera que sea posible modelar las propiedades y conflictos de cada componente mental de un agente. Es decir, se necesita de un formalismo capaz de modelar el concepto de argumento tipado. En la literatura de argumentación el concepto de tipo de argumento siempre ha sido tratado de manera subyacente y fija al problema que se intenta resolver utilizando un formalismo argumentativo. Por lo tanto, ninguna de las propuestas existentes presenta una caracterización general y formal de la noción de tipo de argumento.

Es por este motivo que dentro de las contribuciones de esta tesis se proponen dos nuevos formalismos de argumentación que extienden a sistemas de argumentación existentes permitiendo definir tipos de argumentos y luego especificar relaciones de conflicto, herencia y preferencia entre los tipos definidos. Estos formalismos permitirán modelar los diferentes componentes mentales de un agente ArgAPL y las relaciones entre ellos.

Estos nuevos formalismos proponen un mecanismo novedoso y general para modelar sistemas con múltiples tipos de argumento. De esta manera, no sólo serán adecuados para modelar los tipos de argumento en ArgAPL, sino que también podrán ser utilizados para modelar tipos de argumento en otros dominios como diálogos argumentativos, sistemas con múltiples fuentes de información, sistemas con información basada en valores, o sistemas con información contextualizada.

Por ejemplo, considere un escenario en el que un cirujano y un anestesista están discutiendo acerca de cómo efectuar una operación. El conjunto de argumentos del cirujano $\{C_1 \dots C_n\}$ es tal que cada argumento comparte ciertas características y, por lo tanto, es considerado de tipo T_C. Adicionalmente, considere el conjunto de argumentos del anestesista $S_A = \{A_1 \dots A_m\}$ de tipo T_A. En general, los argumentos de tipo T_C propuestos por el cirujano serán preferidos a los argumentos de tipo T_A del anestesista. Suponga además que existe una especialización de T_A (llamada T_{AE}), correspondiente a los argumentos que se refieren al método de anestesia a utilizar durante la operación. El tipo T_{AE} está determinado por un subconjunto S_{AE} de los argumentos de tipo T_A ($S_{AE} \subset S_A$) y hereda ciertas propiedades de T_A. Claramente, en lo referente a métodos de anestesia los argumentos del anestesista serán preferidos a los argumentos del cirujano. Por lo tanto, en este contexto los argumentos de tipo T_{AE} serán preferidos a los de tipo T_C. Los formalismos argumentativos propuestos en esta tesis permitirán modelar estas situaciones, y a partir de ellas determinarán qué argumentos estarán aceptados.

Para presentar esta formalización, en primer lugar se desarrollarán los *marcos argumentativos de tipos múltiples* (MATM), los cuales serán construidos sobre la base de los *marcos argumentativos abstractos* [Dun95]. Esta aproximación permitirá especificar argumentos tipados y relaciones de conflicto, preferencia y herencia entre los distintos tipos de argumento del marco. El formalismo se abstraerá de la estructura interna de los argumentos y del origen de las relaciones entre los distintos tipos, para centrarse en el análisis de cómo estas relaciones repercutirán en la obtención del conjunto de argumentos del marco que serán finalmente aceptados.

Luego se introducirá T-DeLP, una extensión del formalismo argumentativo de programación en lógica rebatible DeLP [GS04]. T-DeLP presentará un modelo argumentativo completo para la argumentación basada en tipos de argumento. Es decir, cubrirá todas las etapas de un sistema argumentativo: construcción de argumentos, identificación de tipos, identificación de conflictos, generación de derrotas y cómputo de aceptabilidad; considerando durante todas estas etapas la noción de tipo de argumento. El estudio de estos temas resulta fundamental para esta tesis, ya que en ArgAPL el conocimiento se expresará mediante un lenguaje de representación concreto.

Los tipos en T-DeLP estarán relacionados a través de conflictos, herencia y preferencias. Por lo tanto, será necesario considerar un mecanismo de derivación más sofisticado que el utilizado por DeLP para la construcción de argumentos. A partir de las derivaciones en T-DeLP será posible identificar no sólo los argumentos obtenidos sino también sus tipos. Dadas las características de la derivación, se buscará identificar la versión más especializada de cada argumento de acuerdo a la relación de herencia. Estos constituirán los argumentos representativos de T-DeLP y resultarán fundamentales al momento de efectuar el razonamiento argumentativo. Una vez identificados los argumentos representativos del sistema, será necesario determinar las relaciones entre ellos para luego obtener los argumentos finalmente aceptados. Para tal fin se emplearán los marcos argumentativos de tipos múltiples propuestos en esta tesis.

De esta manera, ArgAPL se valdrá de T-DeLP como lenguaje de representación para la especificación de los componentes mentales de sus agentes, así como también de su mecanismo de razonamiento argumentativo. Los tipos de argumento serán utilizados en ArgAPL para identificar los conjuntos de percepciones, creencias y metas. Por lo tanto, las relaciones entre tipos de argumento provistas por T-DeLP permitirán modelar las relaciones entre los componentes mentales de un agente ArgAPL.

Finalmente, se desarrollarán dos extensiones para ArgAPL con respecto a T-DeLP. En la primer extensión se presentará un conjunto de acciones mentales para remover información de las bases de conocimiento de un agente ArgAPL, las cuales emplearán operadores de contracción especialmente definidos para T-DeLP. Por otra parte, la segunda extensión introducirá una técnica especializada de poda con el objetivo de reducir el tamaño de los árboles de dialéctica utilizados por T-DeLP para determinar las creencias y metas actuales de los agentes ArgAPL. Esta técnica de poda se basará en una heurística que será calculada de manera *offline* con respecto a la ejecución del agente.

1.1. Contribuciones

Las principales contribuciones de esta tesis pueden sintetizarse de la siguiente manera:

Lenguaje de programación de agentes con razonamiento argumentativo

Se propone ArgAPL, un lenguaje de programación de agentes basado en 3APL, el cual integrará el sistema argumentativo T-DeLP para modelar el razonamiento de sus agentes. ArgAPL constituye, en conjunto con T-DeLP, la principal contribución de esta tesis. Se mostrará cómo especificar las bases de creencias, metas de logro y metas de mantenimiento de un agente utilizado reglas y hechos tipados T-DeLP. De esta manera, se brindará a un agente ArgAPL la capacidad de representar metas y creencias potencialmente conflictivas. El tipo de los argumentos determinará a qué componente mental representan. Los conflictos que surjan entre los componentes mentales de un agente se modelarán a través de ataques entre argumentos de distinto tipo. Se utilizará el mecanismo argumentativo de T-DeLP para razonar con los argumentos conflictivos, para luego determinar cuáles serán las inferencias del agente en un estado en particular. Se probará que las inferencias de un agente ArgAPL serán consistentes, y que no tendrá una meta de logro que forme parte de sus creencias o que amenace a una meta de mantenimiento. Se formalizará la semántica operacional completa del lenguaje, en la que se especificará cómo las inferencias del mecanismo argumentativo son empleadas para determinar el accionar del agente. Adicionalmente, se mostrará cómo T-DeLP es utilizado para razonar con los conflictos que surjan de la dinámica de ejecución del agente. En particular, se mostrará que empleando T-DeLP como mecanismo de razonamiento será posible modelar agentes con compromiso de mundo-abierto con respecto a sus planes, y que los agentes protegerán sus metas de mantenimiento cancelando planes en ejecución o por ejecutar que pongan en riesgo la condición a mantener.

Marcos argumentativos abstractos de tipos múltiples

Se propone un marco argumentativo que brinda un tratamiento formal al concepto de tipo de argumento en el contexto de argumentación abstracta. En este marco será posible especificar tipos de argumento individuales, los cuales permitirán definir características especiales de un grupo de argumentos (ataques y preferencias), así como también relaciones de herencia, ataque y preferencia entre los tipos individuales. Estos

elementos constituirán en conjunto los marcos argumentativos de tipos múltiples. Se definirá la relación global de ataque entre argumentos de distinto tipo. Se formalizará la relación de preferencia global a través de los conceptos de preferencia interna y externa, los cuales contemplan la jerarquía de herencia entre los tipos de argumento. Finalmente, se mostrará cómo aplicar la semánticas de aceptabilidad clásicas a estos marcos.

Programación en lógica rebatible con tipos de argumento

Se propone T-DeLP, un sistema argumentativo concreto y basado en reglas, en el que los tipos de argumento constituyen la noción central del formalismo. T-DeLP representa una de las contribuciones principales de esta tesis. Este sistema extiende a DeLP [GS04] permitiendo asociar un tipo a los literales de hechos y reglas rebatibles. Asimismo, permitirá el modelado de herencia entre los tipos del sistema. Se presentará una relación generalizada de desacuerdo entre literales tal que contempla el concepto de tipo. Se definirá un mecanismo de derivación que considera los tipos de los argumento y la relación de herencia entre ellos. Se permitirá construir argumentos con sus tipos asociados, y los tipos serán determinados a partir de los tipos especificados en las reglas rebatibles y hechos utilizados para construir los argumentos. Se identificarán las versiones más especializadas de los argumentos, las cuales corresponderán a los argumentos representativos del sistema. Se mostrará que estos argumentos son únicos y que resultan suficientes para la correcta obtención de las inferencias del sistema. Se presentará una noción de ataque entre argumentos que contempla la relación de desacuerdo generalizada, así como también la relación de herencia entre tipos. Las derrotas entre los argumentos representativos se determinarán utilizando el formalismo introducido para los MATMs. Se presentarán dos alternativas para determinar los argumentos aceptables en T-DeLP: una empleando las semánticas de aceptabilidad presentadas para los MATMs, y otra a través de un procedimiento de prueba dialéctico como el utilizado en DeLP. Esta última alternativa será la adoptada por ArgAPL.

1.2. Publicaciones

A continuación se incluyen los artículos publicados como resultado de los trabajos llevados a cabo durante la realización de esta tesis. Para cada uno de estos trabajos se

detalla brevemente su contribución y se indica el capítulo de esta tesis con el cual se vinculan sus resultados.

- En los trabajos:
  - *"Query-Based Argumentation in Agent Programming"* [GGS10], publicado en *Lecture Notes in Computer Science vol. 6433*, y
  - *"Argumentation Systems and Agent Programming Languages"* [GGS09], publicado en *the Uses of Computational Argumentation, AAAI Fall Symposium 2009*,

  se presenta un lenguaje programación de agentes basado en 3APL que utiliza argumentación para razonar con sus metas y sus creencias, y la semántica operacional del modelo de ejecución de los agentes especificados en el lenguaje. Los resultados de estos artículos son la base del lenguaje de programación de agentes ArgAPL desarrollado en el Capítulo 6.

- En los trabajos:
  - *"Defeasible Knowledge and Argumentative Reasoning in 3APL agent Programming"* [GGS08], publicado en el *Twelfth International Workshop on Non-Monotonic Reasoning (NMR 2008)*, y
  - *"Agent Programming using Defeasible Argumentation for Knowledge Representation and Reasoning"* [GGS07], publicado en el *XIII Congreso Argentino de Ciencias de la Computación (CACIC 2007)*,

  se presenta una extensión de 3APL donde los agentes pueden utilizar una base de creencias representada en DeLP y utilizar su mecanismo argumentativo para razonar con las creencias. Los resultados de estos trabajos están reflejados en el formalismo desarrollado en el Capítulo 6.

- En los trabajos:
  - *"A BDI Architecture for High Level Robot Deliberation"* [GTC+10], publicado en la revista *Inteligencia Artificial vol.46*, y
  - *"An Argumentative Intentional Model for High Level Reasoning of Mobile Robots"* [GTGS09], publicado en el *X Argentine Symposium on Artificial Intelligence (ASAI 2009)*,

se presenta una arquitectura BDI para proveer a las aplicaciones con robots móviles capacidades de razonamiento deliberativo utilizando argumentación para determinar sus intenciones. Los resultados de estos trabajos están vinculados con el desarrollo del formalismo del Capítulo 6.

- En el trabajo *"Argument Types and Typed Argumentation Frameworks"* [GGS11], publicado en el *First International Workshop on the Theory and Applications of Formal Argumentation (TAFA 2011)*, se formaliza el concepto de tipo de argumento y los marcos argumentativos de múltiples tipos, se presentan las relaciones de preferencias, conflictos y herencia entre tipos de argumento y, se presenta una noción de derrota entre argumentos tipados. Los resultados de este trabajo están reflejados en los formalismos de los Capítulos 4 y 5.

- En el trabajo *"A Heuristics-Based Pruning Technique for Argumentation Trees"* [RGGS11], publicado en *Lecture Notes in Computer Science vol.6929*, se presenta un modelo para acelerar el cálculo de garantía en un sistema argumentativo a través de una técnica de poda heurística basada en la noción de fuerza argumental. Los resultados de este trabajo están plasmados en el Capítulo 7 para acelerar el cómputo de creencias y metas para el formalismo desarrollado en el Capítulo 6.

- En el trabajo *"On Influence and Contractions in Defeasible Logic Programming"* [GGK+11], publicado en *Lecture Notes in Computer Science vol.6645*, se presenta un conjunto de operadores de contracción para remover un literal de un sistema argumentativo, los cuales están basados en las nociones de derivación, garantía e influencia de este tipo de sistemas. Los resultados de este trabajo son utilizados en el Capítulo 7 para desarrollar un conjunto de acciones mentales especializadas para el lenguaje presentado en Capítulo 6.

1.3. Organización de la Tesis

Esta tesis estará organizada de la siguiente manera:

- En el Capítulo 2 se introducirán los conceptos fundamentales de los lenguajes de programación de agentes, con particular énfasis en una versión de 3APL que será la

base a partir de la cual se construirá el formalismo para la programación de agentes desarrollado en esta tesis.

- En el Capítulo 3 se introducirán los conceptos que caracterizan a los sistemas argumentativos mediante una estructura conceptual. Se presentarán los dos formalismos en los que se basarán los desarrollos argumentativos de esta tesis: los marcos argumentativos abstractos y la programación en lógica rebatible (DeLP).

- En el Capítulo 4 se propondrá un nuevo formalismo de argumentación abstracta para modelar el concepto de tipo de argumento. Allí se mostrará cómo las relaciones de preferencia, ataque y herencia entre los tipos de argumento afectan a la aceptabilidad final de los argumentos del sistema.

- En el Capítulo 5 se presentará T-DeLP, un nuevo sistema argumentativo completo basado en DeLP, el cual permitirá construir argumentos tipados e inferir las diferentes relaciones de tipo. Se mostrará cómo construir estos argumentos y cómo el sistema se apoyará en los modelos de aceptabilidad presentados en el Capítulo 4.

- En el Capítulo 6 se presentará ArgAPL, un lenguaje de programación de agentes que sigue la línea de 3APL y utiliza el formalismo de argumentación desarrollado en el Capítulo 5 como mecanismo de razonamiento interno. El lenguaje ArgAPL se formalizará presentando su sintaxis y semántica con especial énfasis en cómo representar y razonar con percepciones, creencias, metas de logro y metas de mantenimiento utilizando T-DeLP. Se mostrará cómo los argumentos tipados para cada componente mental determinarán las inferencias en un estado y cómo el mecanismo argumentativo impactará en la dinámica de ejecución del agente.

- En el Capítulo 7 se presentarán dos extensiones para ArgAPL con respecto a T-DeLP. En la primera se presentarán acciones mentales que consideran operadores de contracción basados en las diferentes nociones de inferencia provistas por T-DeLP. En la segunda extensión se mostrará cómo aplicar al mecanismo argumentativo de ArgAPL una técnica de poda basada en heurística que permite acelerar el cómputo de las creencias y metas de un agente en un estado particular.

- En el Capítulo 8 se presentarán conclusiones y trabajo a futuro.

Capítulo 2

Lenguajes de Programación de Agentes

El objetivo principal de este capítulo es introducir conceptos esenciales de los lenguajes de programación de agentes. En primera medida se introducirá cómo surgió y evolucionó el área de programación de agentes cognitivos, para luego remarcar la importancia del estudio formal de la semántica de los lenguajes en esta disciplina. Posteriormente, se presentará una versión simplificada del lenguaje de programación de agentes cognitivos 3APL, donde sólo se consideran los elementos relevantes para esta tesis. Para este lenguaje se mostrará cómo se especifican los agentes a través de sus componentes mentales, cómo estos componentes son utilizados en el razonamiento del agente, y cómo es la semántica operacional de sus componentes. Este lenguaje será la base para el formalismo de programación de agentes desarrollado en el Capítulo 6.

2.1. Introducción a los Lenguajes de Programación de Agentes

La presentación de un marco adecuado para la programación de agentes cognitivos racionales no es una tarea trivial. En primer lugar, es necesario determinar formalmente qué significa que un agente sea racional. Es decir, es necesario establecer una teoría que describa a los agentes racionales. Seguidamente, será necesario describir cómo las teorías de agentes racionales pueden implementarse en una unidad concreta de software. Esta

implementación, en general, puede alcanzarse de dos maneras: desarrollando una arquitectura para agentes racionales, o directamente diseñando un *lenguaje de programación dedicado*. La arquitectura correspondiente a la primer alternativa especificará cómo debe ser la estructura general del agente racional y, por lo tanto, podrá implementarse utilizando algún lenguaje de programación de propósito general. La otra posibilidad consiste en diseñar un lenguaje de programación dedicado que brinde constructores para programar los agentes racionales en términos de lo establecido por las teorías de agentes.

El estudio de las teorías, arquitecturas y lenguajes de programación de agentes es ampliamente reconocido en la comunidad de sistemas de agentes (ver [WJ95]). El origen de gran parte de los desarrollos e investigaciones en este área está relacionado con el modelo filosófico Creencias-Deseos-Intenciones (BDI, de *Belief-Desires-Intentions* en inglés) presentado en [Bra87]. El enfoque filosófico BDI está basado en el modelo intencional de [Den87]. La idea detrás del modelo intencional es que el comportamiento de un agente racional puede describirse atribuyendo creencias y deseos al agente, y asumiendo que el agente actuará buscando alcanzar sus metas teniendo en cuenta las creencias que tiene acerca del mundo.

En [Bra87] se establece que el contar con las creencias y los deseos de un agente no es suficiente para explicar y describir un comportamiento racional. En ese trabajo se expresa que existe otra noción esencial para que los agentes sean racionalmente prácticos: la intención. La visión adoptada es que un agente puede tener múltiples deseos, posiblemente conflictivos. Claramente, no es posible que el agente evalúe continuamente sus deseos para decidir cuál de ellos perseguir. Por lo tanto, en algún momento, el agente establecerá un subconjunto no conflictivo de deseos que buscará alcanzar. Los deseos elegidos, y las correspondientes acciones para posiblemente alcanzarlos, constituirán las intenciones del agente.

Las intenciones tienen la característica de estabilidad, en el sentido de que un agente no reconsiderará las intenciones adoptadas previamente, salvo en caso de que se presente un problema significativo. Adicionalmente, si un agente quiere adoptar nuevas intenciones, estas deberán ser coherentes (*i.e.*, no conflictivas) con las intenciones que posee actualmente. En consecuencia, las intenciones actuales de un agente formarán la estructura conocida como marco de admisibilidad.

La propuesta original de BDI sugiere una visión particular de los agentes racionales. Esta visión es la que ha formado las bases para el desarrollo de diversas lógicas para el

modelado de agente racionales [RJ90, RG91, vdHWvLBC90, RG98]. Esas lógicas basan sus propiedades y teoremas en las nociones de creencias, deseos e intenciones, pero también consideran nociones relacionadas como las de metas, anhelo, oportunidad, capacidad y compromiso. Todas estas nociones son comúnmente conocidas como aptitudes mentales o componentes mentales.

Luego de que las lógicas para BDI fueran propuestas, la primer aproximación hacia la idea de programar agentes racionales a través de sus actitudes mentales fue presentada por [Sho93]. Allí se propuso el lenguaje de programación de agentes AGENT-O, introduciendo el concepto de *programación orientada a agentes*. Paralelamente, surgieron las primeras arquitecturas BDI concretas y específicas para la implementación de agentes racionales [GL87, IGR92]. El sistema resultante de estas arquitecturas fue llamado Sistema de Razonamiento Procedural (PRS, del inglés *Procedural Reasoning System*).

Seguidamente se propuso el lenguaje de programación AgentSpeak(L) [Rao96], el cual se encuentra basado en la arquitectura PRS. Este lenguaje puede verse como una simplificación textual de PRS, y su principal motivación fue la inquietud de cómo relacionar los sistemas implementados con PRS con las lógicas BDI. Sin embargo, la relación entre los sistemas que implementan arquitecturas BDI y las lógicas BDI no fue resuelta en [Rao96], y actualmente es un problema que no ha sido resuelto por completo. No obstante, la propuesta de AgentSpeak(L) dio origen a un constante avance en el desarrollo e investigación de lenguajes de programación de agentes o APLs (por su abreviatura del inglés *agent programming languages*).

En particular, uno de los avances más significativos en el área de lenguajes programación de agentes fue 3APL (triple-APL, por su abreviatura del inglés *An Abstract Agent Programming Language*) [HdBvdHM99]. Este lenguaje es el principal lenguaje de programación de agentes en que se basarán los desarrollos de esta tesis. Al igual que AgentSpeak(L), 3APL está inspirado en la teoría BDI. Como se verá en la Sección 2.3, en este lenguaje los agentes se programan a través de sus componentes mentales, así como también mediante reglas de razonamiento práctico que vinculan los componentes mentales con los componentes operacionales del agente tales como planes y acciones. Este lenguaje permite especificaciones declarativas para los componentes mentales del agente, lo cual posibilita relacionarlas con las intuiciones descriptas en las teorías BDI. De esta manera, es posible obtener especificaciones procedurales para modelar el accionar del agente. Además, a diferencia de AgentSpeak(L) (y su plataforma Jason [BWH07]) donde las metas sólo son

eventos que activan la ejecución de un plan, en 3APL se permite una representación declarativa de las metas posibilitando su uso para decidir mediante reglas de razonamiento práctico qué plan será seleccionado. Otra característica notoria de 3APL es que es uno de los pocos lenguajes que describe cómo será la ejecución y razonamiento de sus agentes a partir de reglas de semántica formal. En la Sección 2.2 se profundizará acerca de la importancia de la semántica formal en este tipo de lenguajes de programación de agentes.

Existen otros lenguajes que siguen el espíritu de la programación de agentes racionales de [Sho93], los cuales adoptan un enfoque más alejado de AgentSpeak(L) o 3APL. Ejemplos de tales lenguajes son JACK [Win05] y Jadex [PBL05]. JACK es una extensión de JAVA [AGH00] que provee constructores especializados para la programación orientada a agentes, mientras que Jadex es un motor de razonamiento BDI implementado en JAVA que provee un entorno de ejecución y un conjunto de primitivas para la programación de los agentes. Sin embargo, a diferencia de 3APL, la semántica de estos lenguajes no está formalizada y no utilizan una semántica lógica para metas y creencias, por lo cual los agentes no pueden razonar acerca de cómo la aplicación de un plan repercute en sus metas. Además, estos lenguajes no formalizan una relación lógica entre metas y creencias, por lo que no pueden representar metas de manera declarativa (*i.e.*, indicar una situación del mundo que el agente quiere alcanzar).

En los comienzos del área de investigación y desarrollo en agentes, el término "programación de agentes" era asociado con la programación de agentes inspirados en la teoría BDI. Luego, con el crecimiento del área, surgieron otras aproximaciones a la programación de agentes. Entre ellas, trabajos que adoptan una aproximación mas algorítmica (en lugar de programática) de agentes, dado que están centrados en cuestiones de sistemas multi-agente tales como coordinación y negociación. Por otra parte, surgieron herramientas de programación de agentes que no se encuentran basadas en las teorías BDI. Por ejemplo, IMPACT [DZ05] es una herramienta especialmente basada en programación en lógica, la cual se focaliza en una plataforma multi-agente que permite trabajar con datos distribuidos y heterogéneos.

Para distinguir aquellas herramientas para programar agentes basadas en BDI de las que no lo son, en la comunidad suele utilizarse el término "programación de agentes BDI". Sin embargo, la relación entre estas herramientas y las teorías BDI no es del todo clara. Además, las nociones utilizadas en tales herramientas no corresponden precisamente a creencias, deseos e intenciones, sino a algunas variantes o especializaciones de las nociones

básicas. Generalmente, estas herramientas emplean nociones más operacionales como planes, metas, eventos, y capacidades; además de las nociones creencias, deseos e intenciones. Las nociones contempladas en este amplio espectro también son conocidas como nociones cognitivas. Por lo tanto, en esta tesis se utilizará el término "programación de agentes cognitivos" para hacer referencia a los lenguajes de programación como el que se propondrá en el Capítulo 6. Esta nomenclatura tiene el objetivo de enfatizar que el lenguaje está inspirado en las teorías BDI, pero que adicionalmente emplea nociones cognitivas.

En particular, las nociones cognitivas que se utilizarán en esta tesis son las de creencias, metas y planes. Básicamente, la idea es que, dadas sus creencias, un agente tratará de alcanzar sus metas ejecutando los planes apropiados. Por lo tanto, las creencias constituirán un componente informacional para el agente, las metas serán un componente motivacional, y los planes un componente procedural. Gran parte de los lenguajes de programación de agentes cognitivos cuentan con al menos un componente informacional y un componente procedural. El término "creencia" es el más común entre los componentes informacionales utilizados. Para el componente procedural suele emplearse una combinación del término "plan" y/o el término "intención". Finalmente, con respecto al componente motivacional, el termino "meta" suele aparecer más frecuentemente en la programación de agentes cognitivos que el término "deseo".

2.2. Semántica Formal

Los lenguajes de programación de agentes cognitivos constituyen uno de los pilares fundamentales de esta tesis, incluyendo en particular el estudio de su semántica formal. La investigación de la semántica formal de los lenguajes de programación concierne el estudio riguroso de los modelos computacionales producidos por los mismos. Para definir la semántica formal de un lenguaje de programación, los constructores del lenguaje son asociados a objetos en algún dominio de interpretación. De esta manera, la semántica formal provee una forma de especificación precisa de lo que significa ejecutar un programa producido por el lenguaje de programación.

La semántica formal puede contrastarse con las descripciones semánticas informales que suelen incluirse en manuales de referencia o estándares del lenguaje. Tales descripciones informales son más imprecisas por naturaleza, y se encuentran principalmente basadas en técnicas de implementación e intuiciones. El uso de la semántica formal como medio

para describir el significado de un lenguaje de programación tiene diversas ventajas por sobre las descripciones informales.

Una importante ventaja de contar con la semántica formal surge al utilizarla como herramienta de diseño para un lenguaje de programación, ya que permitirá detectar errores o situaciones problemáticas referentes a la semántica de manera temprana [Ten91]. En esta tesis se mostrarán distintas situaciones en las cuales el hecho de contar con la especificación de la semántica formal aporta precisión al momento de definir el lenguaje y, por lo tanto, ayuda a detectar cuestiones que en otro caso podrían haberse ignorado.

Otro beneficio de utilizar semántica formal es provee una base para la comparación de diferentes lenguajes. Sin una descripción precisa del significado del lenguaje, es realmente dificultoso efectuar afirmaciones sobre la relación exacta entre diferentes lenguajes. Dada la aparición lenguajes de programación de agentes cognitivos cada vez más sofisticados, el contar con este tipo de descripciones semánticas resulta esencial para determinar si constructores que parecen diferentes realmente lo son. De esta manera, el entendimiento de tales diferencias facilita la convergencia hacia un pequeño y especializado grupo de lenguajes de programación de agentes cognitivos.

En el contexto de los lenguajes de programación de agentes cognitivos, la semánticas formales son un prerrequisito para establecer la relación formal entre el lenguaje y las lógicas BDI. Como se mencionó en la sección anterior, esta fue una de las principales motivaciones que dieron origen a AgentSpeak(L) como una alternativa sobre la arquitectura PRS.

Adicionalmente, es importante notar que el uso de reglas de semántica formal para el diseño del lenguaje facilitaría la aceptación de los lenguajes de programación de agentes en el área de ingeniería de software. La razón de esta afirmación radica en que el uso de la semántica formal ayuda al mejor entendimiento de un lenguaje. De esta manera, contribuye a identificar la esencia del lenguaje, lo cual finalmente conduce al diseño un lenguaje más simple que permita capturar esa esencia. La claridad y simpleza de un lenguaje de programación son características que fomentan su uso en la práctica. Es por esto que el uso de semántica formal para la especificación de lenguajes de programación de agentes supone un avance significativo en su uso práctico.

En resumen, la semántica formal provee una base fundacional para el desarrollo de lenguajes de programación de agentes, motivo por el cual se eligió a 3APL como el lenguaje de programación en el que se basarán los resultados de esta tesis. A continuación se

verá que 3APL define una semántica formal completa para sus agentes. En particular, utilizando reglas de semántica para especificar cómo el agente pasa de un estado a otro, así como también para describir su modelo de razonamiento en un determinado estado.

2.3. El Lenguaje 3APL

En esta sección se estudiará una versión de 3APL, un lenguaje de programación de agentes que sigue el espíritu de las teorías BDI. Este lenguaje está basado en la programación de agentes cognitivos, permitiendo especificar un agente a través de un programas lógicos que representarán sus componentes mentales (*i.e.*, creencias, metas e intenciones). El uso de 3APL como lenguaje provee al usuario una forma muy intuitiva y simple de especificar agentes cognitivos racionales. El lenguaje permite especificar declarativamente las creencias (con un programa lógico) y las metas (con un conjunto de átomos), y cómo construir planes con las metas y creencias. Además, otra característica destacable de 3APL es que su modelo de ejecución está definido por reglas de semántica formal a través de un sistema de reglas de transición. Estas reglas definen cómo al aplicar una acción o regla de razonamiento el agente pasa de un estado a otro. Adicionalmente, la semántica formal es utilizada para determinar cómo el agente establecerá lo que cree o busca hacer en un estado particular.

El lenguaje 3APL es ampliamente aceptado en la literatura [BBD+06], y será el lenguaje en el cual se basarán los desarrollos presentados en el Capítulo 6 de esta tesis. Gran cantidad de lenguajes en la literatura utilizan a 3APL como base [dBHvdHM07, vRvdHM03, Das08, DvRDM03]. En general, estos lenguajes extienden ciertos componentes y simplifican o modifican otros, por lo que en esta tesis serán tratados como versiones de 3APL. Al igual que estos lenguajes, el formalismo que se desarrollará en el Capítulo 6 puede verse como una versión de 3APL. Por lo tanto, en esta sección se presentará una versión simplificada de 3APL, en la cual se consideran los elementos relevantes para la presentación del lenguaje y los desarrollos de esta tesis.

Adicionalmente, 3APL y sus variantes han sido exitosamente utilizados como herramienta educativa [Mey02], así como también para el desarrollo de sistemas multi-agentes complejos como los que se presentan en el Multi-Agent Programming Contest [BDD+10]. En particular, en la última edición de esta competencia los agentes del equipo ganador

fueron desarrollados utilizando GOAL [dBHvdHM07], uno de los lenguajes que puede considerarse como una versión de 3APL. Cabe destacar que el autor de esta tesis participó en tal competencia como integrante de un equipo, donde los agentes fueron implementados utilizando algunas de las nociones desarrolladas en esta tesis.

A continuación se introducirán los elementos de 3APL que conforman la descripción del lenguaje. En primera instancia se presentará la sintaxis, seguida de un sistema de transiciones para representar su semántica. Luego se introducirá cómo es el ciclo deliberativo de un agente, y por último se resaltarán ciertas características del lenguaje.

2.3.1. Sintaxis

Un agente 3APL está compuesto por los siguientes elementos:

- una base de creencias σ que representa cómo es el mundo para el agente,
- una base de metas γ que representa el conjunto de estados a los que quiere el agente quiere llegar,
- un conjunto de capacidades Cap que representa las acciones que el agente puede realizar,
- una base de planes Π que representa el plan que el agente está actualmente ejecutando, y
- un conjunto de reglas de razonamiento que es utilizado por el agente para elegir planes en base a sus metas o razonar sobre los planes actualmente en ejecución.

A continuación se explicará cómo especificar cada uno estos componentes, proveyendo una intuición más especifica de su funcionalidad para el agente.

Creencias

Las creencias en 3APL se representan a través de una base de creencias. Esta base es la encargada de describir la información que el agente tiene y puede inferir acerca del mundo en conjunto utilizando su propio conocimiento. Existen distintas aproximaciones para representar una base de creencias en 3APL (*e.g.* [HdBvdHM99, DvRDM03]). En

esta tesis se seguirá la aproximación de programación en lógica para su representación, ya que se asume como estándar en 3APL. Además, esta representación es la que mejor se adecua a los conceptos que se tratarán en esta tesis.

Una base de creencias en 3APL es un programa lógico. Estos programas estarán basados en un conjunto de reglas de átomos, y átomos con negación default. Estos últimos átomos estarán precedidos por un "not" y son usualmente llamados literales default.

Sea $\mathcal{L}$ un conjunto de átomos y $\mathcal{L}_{not} = \mathcal{L} \cup \{notA |\ A \in \mathcal{L}\}$ el conjunto de átomos y literales default (*i.e.*, átomos o átomos precedidos por "not"). Estos conjuntos representarán todos los posibles elementos que pueden aparecer en las reglas de una base de creencias, como se definirá a continuación.

Definición 2.1 (Base de Creencias 3APL) *La base de creencias de un agente 3APL, notada σ, es un conjunto de pares (Cabeza, Cuerpo) donde Cabeza $\in \mathcal{L}$ es un átomo, y cuerpo $\subseteq \mathcal{L}_{not}$ es un conjunto de literales default.*

Usualmente, siguiendo el estilo de programación en lógica, los elementos de σ son conocidos como *reglas*. Por lo tanto, un par $(A, \{A_1, \ldots, A_n\}) \in \sigma$ será se notará en forma de regla como sigue:

$$A \leftarrow A_1, \ldots, A_n$$

Además, aquellas reglas con cuerpo vacío son denominadas *hechos*, con lo que usualmente son denotadas únicamente a través de la cabeza.

Ejemplo 2.1 *Considere un dominio en el que un agente debe transportar cajas entre distintas habitaciones. Una caja puede estar en una habitación o en posesión del agente. En particular, las cajas que se encuentran en una habitación pueden estar apiladas. Para realizar esta tarea, el agente podrá levantar y dejar una caja, así como también ir de una habitación a otra. El agente sólo podrá levantar una caja, si esta no posee otra caja encima. Por lo tanto, para transportar una caja que no se encuentre libre, el agente deberá primero desapilar las cajas que se encuentren por encima de ella.*

Suponga un escenario en el que hay dos habitaciones (hab1 y hab2), tres cajas (caja1, caja2 y caja3), y el agente con identificador "yo". Además, el agente "yo" está en la habitación "hab2", las cajas "caja1" y "caja2" están en la habitación "hab2", la caja

"caja2" está apilada sobre la caja "caja1", y la caja "caja3" está en la habitación "hab1". Este escenario se encuentra descripto por la base de creencias σ_1:

$$\sigma_1 = \left\{ \begin{array}{ll} pos(yo, hab2) & sobre(caja2, caja1) \\ en(caja1, hab2) & en(caja3, hab1) \\ en(caja2, hab2) & libre(X) \leftarrow not\ sobre(Y, X) \end{array} \right\}$$

Note que la regla "$libre(X) \leftarrow not\ sobre(Y, X)$" denota que una caja estará en condiciones de ser levantada por el agente si no posee otra caja encima.

A partir de el Ejemplo 2.1 puede observarse que 3APL no contempla el concepto de literales en conflicto, lo cual es una limitación en el mecanismo de representación de conocimiento provisto por el lenguaje. En los Capítulos 4 y 5 se mostrarán formalismos argumentativos capaces de modelar el concepto de literales en conflicto, los cuales se utilizarán para representar el conocimiento en el lenguaje de programación de agentes desarrollado en el Capítulo 6.

Metas

Así como las creencias son utilizadas para representar lo que el agente conoce e infiere del mundo, las metas representan los elementos que el agente quiere conocer o inferir acerca del mundo. Es decir, un conjunto de metas denotará un estado del mundo al cual el agente quiere llegar. Para representar las metas en 3APL se utiliza la base de metas γ, la cual es un conjunto de átomos, formalmente:

Definición 2.2 (Base de Metas 3APL) *La base de metas de un agente 3APL, notada γ, es un conjunto tal que $\gamma \subseteq \mathcal{L}$.*

Ejemplo 2.2 *Considere el agente del escenario descripto en el Ejemplo 2.1, y suponga que el agente quiere que la caja "caja1" esté en la habitación "hab1", y que la caja "caja3" esté en la habitación "hab2". Esta situación es descripta por la base de metas γ_1:*

$$\gamma_1 = \left\{ \ en(caja1, hab1),\ en(caja3, hab2) \ \right\}$$

El tipo de metas manejado por 3APL es conocido como *metas de logro* y posee la característica de declaratividad. Es decir, las metas de logro son metas que expresan las condiciones del mundo a las que se quiere llegar, pero no especifican cómo hacerlo. En la literatura de agentes [vRDW08] se han estudiado otros tipos de metas declarativas, como por ejemplo las *metas de mantenimiento*. Estas metas representan situaciones del mundo que el agente quiere mantener (*i.e.*, información que no quiere dejar de creer). Por ejemplo, el dominio de los ejemplos 2.1 y 2.2, una meta de mantenimiento podría ser mantener $sobre(caja2, caja1)$ por lo cual, al tratar de cumplir sus metas de logro, el agente debería evitar quitar $sobre(caja2, caja1)$ de la base de creencias. Por lo tanto, note que las metas de mantenimiento podrían llegar a estar en conflicto con las metas de logro. El lenguaje de programación de agentes que se presentará en el Capítulo 6 contemplará tanto metas de logro como metas de mantenimiento, y considerará los conflictos que puedan surgir entre ellas.

Creencias y Metas Actuales

Las creencias y metas que poseerá un agente 3APL en un determinado momento serán llamadas creencias actuales y metas actuales [1]. Estos elementos pueden utilizarse para expresar que un agente tiene o no cierta creencia o meta. En particular, estas creencias y metas actuales serán empleadas para relacionar los diferentes componentes mentales de un agente 3APL, como se verá más adelante en esta sección.

Las creencias actuales y las metas actuales de un agente serán caracterizadas a través de los operadores **B** y **G** respectivamente (**B** de *belief* y **G** de *goal* en inglés). Estos operadores son un recurso meramente sintáctico para distinguir el origen de la información, y no constituyen operadores modales como los de las teorías BDI. A continuación se definirá la sintaxis de estas creencias y metas actuales.

Definición 2.3 (Sintaxis de creencias actuales y metas actuales) *Sea $a \in \mathcal{L}_{not}$. Una creencia actual para a se nota como $\mathbf{B}a$, y una meta actual para a como se nota como $\mathbf{G}a$. El conjunto de todas las creencias actuales de un agente será denotado como $\mathcal{L}_B$, y el conjunto de todas las metas actuales de un agente será denotado como $\mathcal{L}_G$.*

[1] Esta nomenclatura es utilizada en [GGS09, GGS10]. En la literatura también se utiliza la noción de fórmulas de creencias y metas para referenciar a los mismos conceptos.

A continuación, cuando se presenten los restantes componentes de un agente 3APL, se verá en los ejemplos cómo son utilizadas las creencias y metas actuales. Note que los operadores **B** y **G** no se pueden anidar, es decir, especificaciones como **BB**a y **BG**a no son válidas en el lenguaje 3APL. En la sección que describe la semántica formal del lenguaje se verá cómo estos constructores sintácticos son utilizados para representar las metas y creencias actuales de un agente 3APL.

Acciones Básicas

En 3APL un agente ejecutará planes con el objetivo de alcanzar sus metas. Estos planes, como se verá más adelante, serán composiciones de acciones básicas. A continuación se presentarán las acciones mentales utilizadas para modificar la base de creencias del agente, y las acciones externas que se utilizarán para interactuar con el entorno[2].

Acciones Mentales

Las acciones mentales permiten al agente realizar modificaciones sobre su base de creencias, es decir, modificar la información que posee acerca del mundo o cómo obtiene inferencias del mismo. Una acción mental consta de tres componentes: un nombre mediante el cual la acción será referenciada, una precondición que indica las creencias que se deben cumplir para que la regla sea aplicable, y un efecto que representa los cambios que se harán en la base de creencias. Los efectos se caracterizan con "$+$", representando elementos a agregar, y "$-$", representando elementos a remover de la base de creencias. Formalmente:

Definición 2.4 (Acción Mental 3APL) *Sea $\mathcal{L}$ el conjunto de todos los átomos. Una acción mental es una tupla $am = (\beta, N, E)$ tal que β es un conjunto de creencias actuales, $N \in \mathcal{L}$, y $E = \{-Y_1, \ldots, -Y_m, +X_1, \ldots, +X_n\}$ con $n \geq 0$, $m \geq 0$, y $X_i, Y_j \in \mathcal{L}$, donde los X_i e Y_j son elementos a agregar y remover de la base de creencias respectivamente. El conjunto de todas las acciones mentales es denotado como Cam.*

Ejemplo 2.3 *Como se mencionó en el Ejemplo 2.1, para transportar cajas de una habitación a otra el agente contará con las siguientes acciones:*

[2]Si bien 3APL también cuenta con acciones de comunicación y acciones de test, estas pueden ser fácilmente simuladas con las acciones mentales y las acciones externas.

- $(\{\mathbf{B}pos(yo, H), \mathbf{B}en(C, H), \mathbf{B}libre(C)\}, levantar(C), \{-en(C, H), +tiene(yo, C)\})$
- $(\{\mathbf{B}tiene(yo, C), \mathbf{B}pos(yo, H)\}, dejar(C), \{-tiene(yo, C), +en(C, H)\})$
- $(\{\mathbf{B}pos(yo, H_1)\}, ir(H_1, H_2), \{-pos(yo, H_1), +pos(yo, H_2)\})$

Informalmente, la primer acción mental establece que si el agente cree que está en la misma habitación H que la caja C y C está libre, entonces podrá "levantar" la caja C. El efecto de realizar esta acción será agregar a la base de creencias que el agente posee a C, y remover que C está en la habitación H. Por otra parte, la tercer acción mental expresa que para "ir" de la habitación H_1 a la habitación H_2, el agente debe creer que se encuentra en la habitación H_1. El efecto de ejecutar esta acción será remover de la base de creencias que la posición del agente es H_1, para agregar que su posición actual es H_2.

Acciones Externas

Estas acciones son utilizadas por un agente 3APL para producir cambios en el entorno en que se desenvuelve. La forma de estas acciones está directamente relacionada con la implementación del intérprete 3APL. Es por esto que en esta tesis serán tratadas simplemente como átomos, abstrayéndose de cómo se vinculan con la implementación (en particular, del mecanismo que utilizan para afectar al entorno).

Definición 2.5 (Acción Externa 3APL) *Sea $\mathcal{L}$ el conjunto de todos los átomos. Una acción externa E es tal que $E \in \mathcal{L}$. El conjunto de todas las acciones externas será notado como Cae.*

En el ejemplo presentado a lo largo del capítulo no se utilizarán acciones externas, ya que el agente emplea un modelo del mundo puramente interno (*i.e.*, el estado del mundo está completamente modelado en su base de creencias). Por lo tanto, en ese contexto el agente sólo requerirá del uso de acciones mentales para modificar el mundo. Sin embargo, en entornos dinámicos donde el agente sólo tiene información incompleta del mundo y para modificarlo necesita interactuar con un sistema externo, es necesario utilizar las acciones externas.

Finalmente, se notará como Cab al conjunto de todas las acciones básicas con las que contará un agente, es decir, $Cab = Cam \cup Cae$.

Planes

Un plan constituye el medio que posee un agente 3APL para alcanzar una meta. Básicamente, un plan es una secuencia de acciones básicas que posiblemente conduzcan a que la meta en cuestión se transforme en algo que el agente cree. A continuación se definirá formalmente la noción de plan.

Definición 2.6 (Plan 3APL) *Sea Cab el conjunto de acciones básicas. Un plan π para un agente 3APL es una secuencia $\pi = [a_1, \ldots, a_n]$ donde $n \geq 0$ y todo $a_i \in Cab$. El conjunto de todos los planes será notado como C_{Plan}.*

Esta estructura de planes podría ser extendida fácilmente para incorporar constructores de lenguajes de programación, como por ejemplo condicionales, ciclos o subplanes (siguiendo el estilo de 3APL estándar). Sin embargo, estas extensiones no resultan necesarias para las propuestas presentadas en esta tesis.

Reglas de Selección de Planes

Como se mencionó anteriormente, un agente 3APL tratará de alcanzar sus metas por medio de la ejecución de planes. Para permitir al desarrollador de agentes la posibilidad de especificar qué plan deberá ejecutar el agente para alcanzar una meta se emplean las *reglas de selección de planes*. Estas reglas vinculan una meta con un plan cuando se cumplen ciertas condiciones en la base de creencias. Este tipo de reglas constituye la pieza fundamental de razonamiento acerca de los planes en 3APL y fue introducido en [vRMdB06]. Sin embargo, en realidad estas reglas corresponden a una versión refinada de las reglas introducidas en [HdBvdHM00].

Definición 2.7 (Regla de Selección de Plan 3APL) *Sea C_{Plan} el conjunto de posibles planes. Una regla de selección de plan es una tupla (κ, β, π), usualmente notada como $\kappa \mid \beta \Rightarrow \pi$, donde κ una meta actual, β es un conjunto de creencias actuales, y $\pi \in C_{Plan}$.*

Intuitivamente, una regla $\kappa \mid \beta \Rightarrow \pi$ de este tipo, establece que el agente podrá elegir el plan π si cree en β y tiene a κ como meta. Normalmente, cuando se cumplen las condiciones impuestas por una regla de este tipo se dirá que es aplicable. Por lo tanto, un agente sólo ejecutará planes de reglas aplicables. De todos modos ,en la Sección 2.2 se estudiarán estas nociones con mayor profundidad.

Ejemplo 2.4 *Continuando con el escenario descripto en los ejemplos 2.1 y 2.2, el agente contará con las siguientes reglas de selección de planes:*

$\mathbf{G}en(C, H_1) \mid \{\mathbf{B}pos(yo, H_1), \mathbf{B}tiene(yo, C)\} \Rightarrow [dejar(C)]$

$\mathbf{G}en(C, H_1) \mid \{\mathbf{B}pos(yo, H_2), H_2 \neq H_1, \mathbf{B}tiene(yo, C)\} \Rightarrow [ir(H_2, H_1), dejar(C)]$

$\mathbf{G}en(C, H_1) \mid \{\mathbf{B}pos(yo, H_2), \mathbf{B}en(C, H_2)\} \Rightarrow [levantar(C), ir(H_2, H_1), dejar(C)]$

$\mathbf{G}en(C, H_1) \mid \{\mathbf{B}pos(yo, H_2), \mathbf{B}en(C, H_3), H_2 \neq H_3\} \Rightarrow [ir(H_2, H_3), levantar(C), ir(H_3, H_1), dejar(C)]$

La primera regla indica que si el agente tiene como meta que la caja C esté en la habitación H_1, y cree que él está en H_1 con C en su poder, entonces ejecutará el plan que consiste en dejar la caja C. La tercer regla expresa que si el agente quiere que la caja C esté en la habitación H_1, y cree que tanto él como la caja están en la habitación H_2, entonces deberá ejecutar un plan que consiste en levantar C, ir de la habitación H_2 a la habitación H_1, y finalmente dejar C en H_1.

Note que las reglas de selección de planes del agente no consideran el caso en que la caja que se desea transportar no se encuentre libre. A continuación se mostrará cómo las reglas de revisión de planes permitirán modelar esta situación.

Reglas de Revisión de Planes

En 3APL existe otro tipo de reglas de razonamiento, conocidas como reglas de revisión de planes. Estas reglas, a diferencia de las anteriores, sirven para rever un plan que está actualmente en ejecución. De esta manera, este tipo de reglas vincula un plan con otro plan, siempre y cuando se cumpla una condición a partir de la base de creencias del agente. Las reglas de revisión de planes fueron introducidas en [HdBvdHM99], donde fueron denotadas como reglas de razonamiento práctico.

Definición 2.8 (Regla de Revisión de Plan 3APL) *Sea C_{Plan} el conjunto de posibles planes. Una regla de revisión de plan es una tupla (π_e, β, π_c), usualmente notada como $\pi_e \mid \beta \rightsquigarrow \pi_c$, donde β un conjunto de creencias actuales, y $\pi_e, \pi_c \in C_{Plan}$.*

Intuitivamente, una regla de revisión de plan $\pi_e \mid \beta \rightsquigarrow \pi_c$ expresa que si se está ejecutando el plan π_e y el agente cree en β, entonces será posible reemplazar el plan π_e por el plan π_c.

Ejemplo 2.5 *Teniendo en cuenta que el agente de los ejemplos anteriores sólo puede levantar cajas que se encuentren libres, contará con la siguiente regla de revisión de plan:*

$$[levantar(C_1)] \mid \{\mathbf{B}sobre(C_2, C_1)\} \rightsquigarrow [levantar(C_2), dejar(C_2), levantar(C_1)]$$

Esta regla expresa que si el plan del agente consiste en efectuar la acción "levantar(C_1)" y cree que la caja C_2 está sobre la caja C_1 (con lo cual no es posible ejecutar esta acción, ya que no se cumple la precondición libre(C_1)), entonces tendrá un nuevo plan que implica primero colocar la caja C_2 en el piso de la habitación, y luego levantar la caja C_1.

Como se puede observar en el Ejemplo 2.5, las reglas de revisión de planes pueden utilizarse para reparar planes en los que falló la ejecución de alguna de las acciones. En la Sección 2.2 se mostrará cómo el uso de estas reglas permitirá reemplazar una porción de plan fallido por el plan indicado en una de estas reglas.

El Agente 3APL

Programar un agente en 3APL implica especificar una base de creencias inicial, un conjunto de metas iniciales, un conjunto de acciones básicas utilizables (conocidas como capacidades), y un conjunto de reglas de razonamiento (*i.e.*, reglas de selección y revisión de planes).

Definición 2.9 (Agente 3APL) *Un agente 3APL es una tupla $(\sigma_0, \gamma_0, Cap, SP, RP)$, donde σ_0 es una base de creencias, γ_0 es una base de metas, Cap es un conjunto de capacidades, SP es un conjunto de reglas de selección de plan, y RP es un conjunto de reglas de revisión de plan.*

Ejemplo 2.6 *El agente que se desenvuelve en el mundo de bloques puede ser definido mediante todos los elementos presentados previamente en los ejemplos 2.1, 2.2, 2.3, 2.4, y 2.5.*

Un agente 3APL contará con un mecanismo para determinar cómo la base de creencias será modificada en caso que el agente ejecute una acción mental. En la mayoría de los trabajos de 3APL [HdBvdHM99, DvRM05, DvRDM03] esto se modela mediante una

función abstracta que recibe la acción metal y la base de creencias actual, y retorna una nueva base de creencias modificada. En los últimos trabajos de 3APL [Das08] esta función se concretizó de manera trivial, dado que se emplea Prolog como lenguaje de representación. Básicamente, esta función de actualización, denotada $\mathcal{F}$, parte de los elementos de la pos-condición de la acción mental para agregar y remover de la base de creencias aquellos elementos antecedidos por el "+" y el "−" respectivamente.

Nótese que el agente no comienza comprometido con ningún plan inicial. La idea es que, a partir de sus metas iniciales, el agente elija sus primeros planes a través de las reglas de selección de planes. Esto se debe a que la Definición 2.9 es empleada para especificar la situación inicial del agente. Por lo tanto, en esta tesis se utilizará el término *especificación del agente* (o también programa de agente) para denotar a esta situación inicial, y el término agente (o estado del agente) para referenciar a la entidad computacional que ejecuta.

2.3.2. Semántica Formal en 3APL

En la sección anterior se definieron los componentes que son utilizados para especificar un agente. En esta sección se definirá qué significa ejecutar un agente, es decir, se especificará su semántica. En este contexto, deberá explicarse qué significa el uso de cada uno de los componentes mentales de un agente y cómo se relacionan entre sí.

Para explicar la semántica del agente se introducirá la noción de configuración. Las configuraciones serán utilizadas para representar el estado de los componentes mentales del agente en cada momento de su ejecución. Es decir, una configuración en un momento particular denotará una imagen de los componentes mentales en ese momento. Es así que las configuraciones de un agente estarán caracterizadas únicamente por aquellos componentes que pueden cambiar durante su ejecución. Por lo tanto, las reglas de razonamiento y las acciones básicas (presentes en la especificación inicial del agente) no formarán parte de una configuración. De esta manera, una configuración estará caracterizada por una base de creencias, una base de metas y, adicionalmente, una base de planes. Esta última indicará el plan y la meta que el agente está persiguiendo actualmente, o será especificada mediante ϵ en caso de que actualmente no posea un plan en ejecución.

Definición 2.10 (Configuración 3APL) *Una configuración de un agente 3APL es una tupla* (σ, γ, Π)*, donde* σ *es una base de creencias,* γ *es una base de metas, y* Π

es la base de planes constituida por el par (π, κ) *donde* π *es un plan y* κ *es una meta, o bien por* ϵ.

Inicialmente, un agente comenzará con una configuración caracterizada por las bases presentadas en la Definición 2.9 y una base de planes vacía. Formalmente:

Definición 2.11 *Sea* $A = (\sigma_0, \gamma_0, Cap, SP, RP)$ *la especificación de un agente 3APL. La configuración inicial de A será* $(\sigma_0, \gamma_0, \epsilon)$.

Ejemplo 2.7 *Considere el agente que se desenvuelve en el mundo de bloques correspondiente al Ejemplo 2.6. La configuración inicial de este agente contendrá la base de creencias del Ejemplo 2.1, la base de metas del Ejemplo 2.2 y una base de planes vacía.*

Razonamiento en 3APL

El modelo de razonamiento describe la semántica operacional de las creencias y las metas para un agente 3APL en un momento dado. Es decir, este modelo establece cómo el agente determinará en qué cree o qué busca lograr a partir de una configuración. Para esto se utilizarán las creencias y metas actuales. De esta manera, si una creencia actual $\mathbf{B}a$ es válida (o se infiere) en una configuración, querrá decir que el agente cree en a en esa configuración. Análogamente, y si $\mathbf{G}$b es válida en una configuración, significará que el agente tiene como meta a b en esa configuración.

Para determinar si $\mathbf{B}a$ es una creencia actual válida en una configuración, deberá analizarse si es posible inferir a a partir de la base de creencias de esa configuración. De manera similar, una meta actual $\mathbf{G}$c será válida en una configuración si se puede obtener c de la base de metas de esa configuración, y además no es inferible a partir de la base de creencias de tal configuración. Esto último se debe a la intuición de que el agente no debe tener una meta que cree ya alcanzada. Esta característica corresponde a las *metas de logro*, el tipo de metas con las que cuenta 3APL.

A continuación, en la definición de la semántica de las creencias actuales y las metas actuales, se capturarán las intuiciones enunciadas en los párrafos anteriores.

Definición 2.12 (Semántica de creencias y metas actuales) *Sea* $C = (\sigma, \gamma, \Pi)$ *una configuración de agente,* $\vdash$ *la relación de deducción estándar de programación en*

lógica sobre los elementos de $\mathcal{L}$, $\mathbf{B}a$ *una creencia actual,* $\mathbf{G}c$ *una meta actual,* β *un conjunto de creencias actuales y* κ *un conjunto de metas actuales. La semántica de las creencias actuales y metas actuales se define respectivamente como:*

$$
\begin{array}{lcl}
(\sigma,\gamma,\Pi)\vdash_S \mathbf{B}a & \Leftrightarrow & \sigma \vdash a \\
(\sigma,\gamma,\Pi)\vdash_S \beta & \Leftrightarrow & \textit{Para todo } \mathbf{B}a_i \textit{ en } \beta \textit{ vale que } (\sigma,\gamma,\Pi)\vdash_S \mathbf{B}a_i
\end{array}
$$

$$
\begin{array}{lcl}
(\sigma,\gamma,\Pi)\vdash_S \mathbf{G}c & \Leftrightarrow & \gamma \vdash c \textit{ y } \sigma \nvdash c \\
(\sigma,\gamma,\Pi)\vdash_S \kappa & \Leftrightarrow & \textit{Para todo } \mathbf{G}c_j \textit{ en } \kappa \textit{ vale que } (\sigma,\gamma,\Pi)\vdash_S \mathbf{G}c_j
\end{array}
$$

Note que, de acuerdo a lo establecido por la Definición 2.12, si $(\sigma,\gamma,\Pi)\vdash_S \mathbf{B}a$, entonces a es una creencia actual válida en la configuración (σ,γ,Π). Análogamente, si $(\sigma,\gamma,\Pi)\vdash_S \mathbf{G}c$, entonces c es una meta actual válida en la configuración (σ,γ,Π).

Como se mencionó anteriormente, las creencias actuales y las metas actuales representan los elementos que el agente utilizará para analizar si tiene una creencia o meta determinada. En particular, estas creencias y metas actuales son utilizadas en acciones, reglas de selección de planes y reglas de revisión de planes. Por lo tanto, la semántica de creencias y metas actuales resultará importante para explicar cómo es la semántica operacional de estos otros componentes.

Reglas de Transición

Habiendo definido cómo el agente razona acerca de sus creencias y metas en una configuración determinada, es posible presentar la semántica de ejecución de un agente 3APL. Esta semántica operacional se definirá a través de un sistema de transiciones [Plo04]. Un sistema de transiciones para un lenguaje de programación consiste de un conjunto de axiomas y reglas de transición que determinarán cómo el sistema pasa de un estado a otro. Por lo tanto, una transición en un agente será la transformación de una configuración en otra configuración, que además corresponderá a un paso computacional individual.

Una regla de transición en 3APL estará compuesta por tres elementos: una condición bajo la cual la regla de transición es aplicable y dos configuraciones (una de origen y otra de destino), las cuales describen cómo el agente cambia de configuración al aplicar la regla de transición. Por ejemplo, una regla de transición T con condición C, configuración origen O y configuración destino D, se notará como:

$$T = \frac{\text{condición C}}{\text{configuración origen O}\rightarrow\text{configuración destino D}}$$

En las reglas de transición que se presentarán a continuación se asumirá que se cuenta con un conjunto SP de reglas de selección de plan, con un conjunto RP de reglas de revisión de plan, y con un conjunto Cap de acciones básicas, todos obtenidos a partir de la especificación del agente (Definición 2.9), además de la función de actualización de la base de creencias $\mathcal{F}$ que se mencionó anteriormente.

Semántica de Acciones Básicas

Las acciones mentales actualizan la base de creencias del agente a través de la función $\mathcal{F}$. Esta función, como se mencionó anteriormente, retornará la base de creencias que resulta de aplicar una acción mental a una base de creencias. Las metas que se alcancen luego de la ejecución de esta acción deberán ser removidas de la base de metas. Luego de la ejecución de una acción, esta deberá ser removida del plan correspondiente. Por último, la ejecución de una acción será posible siempre y cuando se cumplan las precondiciones de la acción y la meta para la cual el plan (que contiene la acción) fue concebido siga valiendo.

Definición 2.13 (Transición de Acciones Mentales) *Sea $am = (\beta, N, E)$ una acción mental y $C_O = (\sigma, \gamma, ([am, a_1, \ldots, a_n], \kappa))$ una configuración, con $n \geq 0$. La transición producida por ma en C_O se define como:*

$$\frac{C_O \vdash_S \beta \ \wedge \ C_O \vdash_S \kappa \ \wedge \ \mathcal{F}(\sigma, am) = \sigma'}{(\sigma, \gamma, ([am, a_1, \ldots, a_n], \kappa)) \rightarrow (\sigma', \gamma', ([a_1, \ldots, a_n], \kappa))}$$

donde $\gamma' = \{\phi |\ \phi \in \gamma \ \wedge \ (\sigma', \gamma', ([a_1, \ldots, a_n], \kappa)) \nvdash_S \mathbf{B}\phi\}$

Ejemplo 2.8 *Considere la configuración $C_O = (\sigma_1, \gamma_1, ([ir(hab2, hab1), levantar(caja3), ir(hab1, hab2), dejar(caja3)], \mathbf{G}en(caja3, hab1)))$ para el agente presentado en los ejemplos anteriores, donde σ_1 es la base de creencias del Ejemplo 2.1 y γ_1 es la base de metas del Ejemplo 2.2. Dado que $C_O \vdash_S \{\mathbf{B}en(yo, hab2)\}$, que $C_O \vdash_S \mathbf{G}en(caja3, hab1)$, que $\mathcal{F}(ir(hab2, hab1), \sigma_1) = \sigma_2 = (\sigma_1 \cup \{pos(yo, hab2)\}) \setminus \{pos(yo, hab2)\}$, y que $\gamma_2 = \gamma_1$, se podrá aplicar la regla de transición para la acción mental "$ir(hab2, hab1)$", llevando al agente a la configuración $C_D = (\sigma_2, \gamma_2, ([levantar(caja3), ir(hab1, hab2), dejar(caja3)], \mathbf{G}en(caja3, hab1)))$.*

Como se mencionó en la Sección 2.3.1, las acciones externas en 3APL serán átomos que cuando son ejecutados representarán que el agente interactúa con su entorno. Normalmente estas acciones son utilizadas en combinación con las acciones mentales, ya que podrían llegar a pasar información para actualizar la base de creencias. Aún así, los autores de 3APL han tratado estas acciones de manera abstracta, tomando un enfoque diferente en las distintas versiones del lenguaje. Dada esta situación, se asumirá la existencia de una función *Call* que ejecutará una acción externa en un ambiente adecuado.

Definición 2.14 (Ejecución de Acción Externa) *Sea ae una acción externa y* $C_O = (\sigma, \gamma, ([e, a_1, \ldots, a_n], \kappa))$ *una configuración. La transición producida por ae en* C_O *se define como:*

$$\frac{Call(ae) \ \wedge \ C_O \vdash_S \kappa}{(\sigma, \gamma, ([ae, a_1, \ldots, a_n], \kappa)) \rightarrow (\sigma, \gamma, ([a_1, \ldots, a_n], \kappa))}$$

Como se mencionó anteriormente, el ejemplo utilizado a lo largo de este capítulo no hace uso de acciones externas. Sin embargo, existen entornos dinámicos en los que un agente necesita interactuar con un sistema externo para ejecutar sus acciones y poder así conocer cómo los efectos de estas acciones afectarán al mundo y, por lo tanto, a sus creencias. Es por esto que, en entornos con tales características, un agente necesitará utilizar acciones externas.

Por ejemplo, en un escenario como el del *Multi-Agent Contest* [BDD+10], los agentes comunican sus acciones a un servidor encargado de manipular el mundo, siendo este servidor el que efectivamente ejecuta las acciones en el mundo. Si bien el agente puede predecir en cierto grado cual será el efecto de su accionar en el mundo, no puede asegurar que una acción tendrá éxito, ni cómo se combinará con el accionar de otros agentes. Por lo tanto, en este contexto, para saber definitivamente cual será el efecto de su accionar en el mundo, el agente deberá solicitar al servidor una percepción del mundo luego de aplicar la acción. Los agentes 3APL, si bien proveen el mecanismo para modelar las acciones externas, se abstraen demasiado de todas estas cuestiones. El lenguaje de programación de agentes desarrollado en el Capítulo 6 considerará una representación de acciones externas más rica, así como también un modelo para considerar las percepciones.

Semántica de Reglas de Razonamiento

A continuación se analizará la transición para las reglas de selección de plan y las reglas de revisión de plan. Una regla de selección de plan hará que un plan pase a ejecutarse

si se dan ciertas condiciones. En primer lugar, la regla de selección de plan debe ser aplicable, es decir, tanto la meta que pregona como sus precondiciones tienen que valer en la configuración actual del agente. Por otra parte, un agente podrá en ejecución un plan a partir de las reglas de selección de planes, siempre y cuando no haya un plan actual en ejecución. La intuición detrás de esta restricción es que el agente elegirá un nuevo plan únicamente cuando haya finalizado de ejecutar uno anterior, o cuando el plan actual ya no sea válido. En caso de cumplirse estas condiciones, el plan correspondiente a la regla de selección de plan aplicable pasará a ser el plan actual en la base de planes.

Definición 2.15 (Transición de Reglas de Selección de Planes) *Sea* $\kappa \mid \beta \Rightarrow \pi$ *una regla de selección de plan y* $C_O = (\sigma, \gamma, \epsilon)$ *una configuración. La transición producida por* $\kappa \mid \beta \Rightarrow \pi$ *en* C_O *se define como:*

$$\frac{C_O \vdash_S \beta \quad \wedge \quad C_O \vdash_S \kappa}{(\sigma, \gamma, \epsilon) \rightarrow (\sigma, \gamma, (\pi, \kappa))}$$

Ejemplo 2.9 *Considere el agente que se desenvuelve en el dominio del Ejemplo 2.1, cuya configuración inicial* $C_I = (\sigma_1, \gamma_1, \epsilon)$ *es la presentada en el Ejemplo 2.7. En esta configuración las siguientes reglas de selección de planes son aplicables:*

$$\mathbf{G}en(C, H_1) \mid \{\mathbf{B}pos(yo, H_2), \mathbf{B}en(C, H_2)\} \Rightarrow [levantar(C), ir(H_2, H_1), dejar(C)]$$

$$\mathbf{G}en(C, H_1) \mid \{\mathbf{B}pos(yo, H_2), \mathbf{B}en(C, H_3), H_2 \neq H_3\} \Rightarrow [ir(H_2, H_3), levantar(C), ir(H_3, H_1), dejar(C)]$$

La primer regla es aplicable dado que $C_I \vdash_S \mathbf{G}en(caja1, hab1)$, $C_I \vdash_S \{\mathbf{B}pos(yo, hab2), \mathbf{B}en(caja1, hab2)\}$ *y la base de planes de* C_I *es vacía. Por lo tanto, al aplicar la regla de transición a esta regla de selección de planes la configuración resultante será* $C_2 = (\sigma_1, \gamma_1, ([levantar(caja1), ir(hab2, hab1), dejar(caja1))], \mathbf{G}en(caja1, hab1)))$. *Por otra parte, la segunda regla es aplicable dado que* $C_I \vdash_S \mathbf{G}en(caja3, hab2)$, $C_I \vdash_S \{\mathbf{B}pos(yo, hab2), \mathbf{B}en(caja3, hab1)\}$ *y la base de planes de* C_I *es vacía. Por lo tanto, al aplicar la regla de transición a esta regla de selección de planes la configuración resultante será* $C_{2'} = (\sigma_1, \gamma_1, ([ir(hab2, hab1), levantar(caja3), ir(hab1, hab2), dejar(caja3))], \mathbf{G}en(caja3, hab1)))$. *Adicionalmente, note que la configuración* $C_{2'}$ *es la utilizada como configuración inicial en el Ejemplo 2.8.*

Note que, como se puede observar en el Ejemplo 2.9, el agente podría tener más de una regla de selección de plan aplicable. En ese caso, el agente debe elegir una de ellas utilizando alguna estrategia, como por ejemplo preferir el plan más corto, o tratar de alcanzar las metas más valiosas. En las implementaciones de 3APL [DvRM05] las reglas de selección de plan se ordenan según como han sido especificadas, y ese orden establece cuáles se intentarán aplicar primero. Aun así, el formalismo de 3APL maneja este criterio de selección de manera abstracta y modular.

De manera similar a las reglas de selección de planes, las reglas de revisión de planes podrán producir una transición cuando sean aplicables. Estas reglas serán aplicables cuando, en la configuración actual, el agente crea en la precondición de la regla y además el plan de la configuración actual tenga como prefijo al plan a revisar por la regla de revisión de planes. Al aplicar la regla, se llegará a una nueva transición donde el prefijo de plan es reemplazado por el plan en el cuerpo de la regla. Por lo tanto, para definir esta transición se utilizará la función Prefijo, la cual dados dos planes retornará $\top$ si el primero es un prefijo del segundo y $\bot$ en caso contrario. Por ejemplo, una regla $[a, b] \mid \top \rightsquigarrow [c]$ puede ser aplicada en el contexto de un plan [a,b,c], ya que no posee precondiciones y $\mathsf{Prefijo}([a, b], [a, b, c]) = \top$, con lo cual se obtendrá el plan [c,c].

Definición 2.16 (Transición de Reglas de Revisión de Planes) *Sea* $[a_1, \ldots, a_n] \mid \beta \rightsquigarrow [c_0, \ldots, c_k]$ *una regla de revisión de plan con* $n \geq 1$, $k \geq 0$, *y sea* $C_O = (\sigma, \gamma, ([a_1, \ldots, a_n, a_{n+1}, \ldots, a_m], \kappa))$ *una configuración con* $m \geq n$. *La transición producida por* $[a_1, \ldots, a_n] \mid \beta \rightsquigarrow [c_0, \ldots, c_k]$ *en* C_O *se define como:*

$$\frac{C_O \vdash_S \beta \ \wedge \ C_O \vdash_S \kappa \ \wedge \ \mathit{Prefijo}([a_1, \ldots, a_n], [a_1, \ldots, a_n, a_{n+1}, \ldots, a_m])}{(\sigma, \gamma, ([a_1, \ldots, a_n, a_{n+1}, \ldots, a_m], \kappa)) \rightarrow (\sigma, \gamma, ([c_0, \ldots, c_k, a_{n+1}, \ldots, a_m], \kappa))}$$

Ejemplo 2.10 *Considere que el agente se encuentra en la configuración* $C_2 = (\sigma_1, \gamma_1, ([levantar(caja1), ir(hab2, hab1), dejar(caja1)]), \mathbf{G}en(caja1, hab1)))$ *del Ejemplo 2.9. Note que la regla:*

$$[levantar(C_1)] \mid \{\mathbf{B}sobre(C_2, C_1)\} \rightsquigarrow [levantar(C_2), dejar(C_2), levantar(C_1)]$$

es aplicable en C_2. *Esto se debe a que* $C_2 \vdash_S \mathbf{B}sobre(caja2, caja1)$, $C_3 \vdash_S \mathbf{G}en(caja1, hab1)$ *y el plan* $[levantar(caja1)]$ *es un prefijo del plan* $[levantar(caja1), ir(hab2, hab1), dejar(caja1)]$ *actualmente en ejecución. En consecuencia, el resultado de aplicar esta regla llevará al agente a la siguiente configuración* $C_3 = (\sigma_1, \gamma_1, ([levantar(caja2), dejar(caja2), levantar(caja1), ir(hab2, hab1), dejar(caja1)], en(caja1, hab1)))$.

Con esta última transición se presentaron todas las transiciones para los componentes especificados en el agente. Sin embargo, es necesario utilizar otras transiciones especiales para representar qué ocurre cuando, por ejemplo, la meta del plan que se está ejecutando actualmente deja de ser válida, o cuando el agente intenta ejecutar un plan vacío.

En primer instancia, si por alguna razón el agente está intentando ejecutar un plan en una configuración tal que la meta para la cual fue concebido ya no es válida en esa configuración, el plan debe ser descartado. En consecuencia, al descartar un plan, la base de planes pasa a estar vacía.

Definición 2.17 (Transición Fallo Meta) *Sea $C_O = (\sigma, \gamma, (\pi, \kappa))$ una configuración. La transición producida en caso de que $C_O \not\models_S \kappa$ se define como:*

$$\frac{C_O \not\models_S \kappa}{(\sigma, \gamma, (\pi, \kappa)) \rightarrow (\sigma, \gamma, \epsilon)}$$

Nótese que esta transición es aplicable siempre que la meta del plan actual deje de valer, sin importar cuáles sean las acciones que conforman el plan. Adicionalmente, es importante distinguir el concepto de plan descartado del concepto de plan bloqueado. Como se mencionó anteriormente, un plan descartado es aquel para el cual la meta que busca alcanzar ya no es válida, ya sea porque fue alcanzada previamente o porque fue descartada por alguna otra razón. En contraste, un plan bloqueado es un plan aún válido, que por alguna razón no puede continuar su ejecución hasta que la situación del mundo cambie o el plan sea reparado. En 3APL no se considerarán reglas de transición especiales para estas últimas situaciones, ya que el estado del agente no cambiará por ellas.

Si ocurre la situación en que un agente 3APL cuenta con un plan vacío en ejecución, esto implica probablemente ya ejecutó todas sus acciones y, por lo tanto, debe removerlo de la base de planes. Por lo tanto, el agente pasará a estar en un estado en el que no posee un plan por ejecutar.

Definición 2.18 (Transición plan vacío) *Sea $C_O = (\sigma, \gamma, ([], \kappa))$ una configuración. La transición producida cuando en C_O hay un plan vacío se define como:*

$$\frac{\top}{(\sigma, \gamma, ([], \kappa)) \rightarrow (\sigma, \gamma, \epsilon)}$$

2.3.3. Ciclo Deliberativo

En las secciones previas se introdujo cómo es posible especificar un agente en 3APL, qué componentes tiene, y cómo interactúan entre sí. Además, se mostró a través del sistema de transiciones cómo estos componentes afectan la ejecución de un agente, es decir, qué implica utilizarlos. En particular, se mostró que un agente ejecutará planes y aplicará las distintas reglas de razonamiento para modificar su base de planes (ya sea agregando un plan o revisando uno existente). Sin embargo, es necesario contar con una guía para determinar qué regla de transición aplicar en cada momento, es decir, establecer el orden en que se utilizarán los distintos componentes del agente. Por ejemplo, asuma que se tiene el plan $[a]$ en la base de planes del agente. Suponga además que, por una parte, hay una acción mental aplicable en esa configuración y, por otra parte, hay una regla de revisión de planes también aplicable. ¿Qué debe hacer el agente?. Más aún, suponga la situación en que el agente no tiene ningún plan en ejecución en la configuración actual, y posee dos reglas de selección de plan aplicables. ¿Qué plan pasará a ejecutarse?.

Es por esto que el agente necesita una estrategia para determinar cómo usar sus componentes. Usualmente, esto se conoce como ciclo deliberativo del agente. Este ciclo puede verse como un programa deliberativo que determina qué operaciones de deliberación deben aplicarse y cuándo. Por ejemplo, podría programarse para determinar cómo son elegidas las metas de selección de planes, o cuándo se aplican las reglas de revisión de planes. Una característica de este ciclo en 3APL es que también es modular o, equivalentemente, programable. En [DdBDM03] se presenta un lenguaje concreto para programar el ciclo deliberativo de los agentes 3APL.

En la implementación de 3APL [DvRM05] se presenta un estrategia concreta para el ciclo deliberativo. Esta estrategia se describe de la siguiente manera:

1. Si es posible, elegir la primer regla de selección de planes aplicable de acuerdo al orden en que fueron especificadas. Si existe una regla con estas características poner su plan en ejecución.

2. Si es posible, elegir la primer regla de revisión de planes aplicable de acuerdo al orden en que fueron especificadas. Si existe una regla con estas características poner su plan en ejecución.

3. Si existe un plan no vacío en ejecución, ejecutar su primer acción. En caso contrario remover el plan.

4. Ir a 1.

2.4. Resumen

En este capítulo se presentaron los conceptos básicos de los lenguajes programación de agentes cognitivos. Se introdujo cómo surgió y evolucionó el área de este tipo de lenguajes, y se indicó sobre qué tipo de lenguajes de programación de agentes se contextualizan las propuestas de esta tesis. Adicionalmente, se remarcó la importancia del estudio de la semántica formal de este tipo de lenguajes.

Como base teórica para el desarrollo de los formalismos de esta tesis se presentó el lenguaje de programación de agentes cognitivos 3APL. Para este lenguaje se mostró cómo especificar un agente a través de sus componentes mentales y un conjunto de reglas de razonamiento que permiten vincular aptitudes mentales con elementos operacionales tales como planes y acciones. Luego se presentó la semántica formal del lenguaje, en la cual se caracteriza cómo el agente determina cuáles son sus inferencias en un estado mental, y cómo pasa de un estado a otro al aplicar reglas de razonamiento o acciones. Finalmente, se mostró cómo construir un ciclo deliberativo completo para los agentes 3APL, el cual se basa en la semántica formal previamente definida.

En el próximo capítulo se introducirán los conceptos básicos de los sistemas argumentativos, para luego presentar los formalismos argumentativos basados en tipos que forman parte de las contribuciones de esta tesis. Estos formalismos constituirán el mecanismo de representación de conocimiento y razonamiento de los agentes especificados mediante el lenguaje de programación de agentes presentado en el Capítulo 6.

Capítulo 3

Conceptos Básicos de Argumentación

En este capítulo se introducirán los conceptos que caracterizan a los sistemas argumentativos. Esto se realizará a través de la estructura conceptual identificada en [PV02]. Adicionalmente se presentarán los dos sistemas argumentativos en los cuales se basarán los formalismos desarrollados en esta tesis: el sistema argumentativo para los marcos argumentativos abstractos propuesto en [Dun95], y el sistema argumentativo de la programación en lógica rebatible, también conocido como DeLP (por su abreviatura del inglés), presentado en [GS04]. Se mostrarán las principales características de estos sistemas, identificando con qué elementos de la estructura conceptual cuentan.

3.1. Introducción

Argumentación es una forma de razonamiento en la cual, para una afirmación determinada, se examinan explícitamente las justificaciones presentadas y la resolución de los posibles conflictos entre ellas. En este tipo de razonamiento, una afirmación es aceptada o rechazada según el análisis de los argumentos a su favor y en su contra. La forma en que los argumentos y las justificaciones para una afirmación son considerados permite definir un tipo de razonamiento automático, en el cual puede haber información contradictoria, incompleta, e incierta.

De esta manera, la argumentación constituye una área de estudio de especial interés en el ámbito de la Inteligencia Artificial (ver *e.g.* [RS09]), principalmente, porque permite razonar con información incompleta e incierta, y permite manejar inconsistencia en los sistemas basados en conocimiento. Este tipo de razonamiento es particularmente atractivo para la toma de decisiones, y dentro del área de Inteligencia Artificial existe particular interés en abordar este tipo de problemas.

El estudio de la argumentación ha sido abordado desde diferentes enfoques: a nivel lógico puede considerarse como una forma de modelar inferencia rebatible, y a nivel dialógico como una forma de interacción entre agentes. Esta forma de razonamiento ha sido ampliamente estudiada en disciplinas como la filosofía [Tou03], y desde los años '70 es posible encontrar estudios en Ciencias de la Computación que han contribuido notablemente a la noción de argumento. A mediados de los años '80 comenzaron los desarrollos del área de argumentación desde un punto de vista computacional, donde los argumentos son explícitamente construidos y comparados como medios para resolver problemas en una computadora. En la última década, la argumentación ha evolucionado como un atractivo paradigma para conceptualizar el razonamiento de sentido común (ver [CML00] y [PV02]). Esto produjo como resultado la formalización de diferentes frameworks de argumentación abstracta como [Dun95], [BGG05], y [BG09], entre otros; y de Sistemas Argumentativos Basados en Reglas (SABR) como [PS97], [AK07], [DKT06] y [GS04]. En consecuencia, esto permitió el desarrollo de diversas aplicaciones del mundo real basadas en argumentación. En el último tiempo, el campo de aplicación de la argumentación se ha expandido velozmente, en gran parte debido a los avances teóricos, pero también gracias a la demostración exitosa de su uso práctico en un gran número de dominios de aplicación, tales como el razonamiento legal [PS02], la ingeniería del conocimiento [CRL00], los sistemas multi-agente [PSJ98, RA06, RGS07, GGS08, GGS10], y el e-government [ABCM05], entre muchos otros.

A continuación se presentará una estructura conceptual que describe los principales elementos con los que se puede caracterizar un sistema argumentativo. Luego se presentarán los dos formalismos argumentativos que son utilizados como base de esta tesis: los marcos argumentativos abstractos [Dun95] y la programación en lógica rebatible o DeLP [GS04].

3.1.1. Una estructura conceptual para los sistemas argumentativos

En [PV02] se identifica una estructura conceptual dentro de la cual pueden caracterizarse la mayoría de los sistemas argumentativos existentes. De acuerdo a este marco conceptual, generalmente, un sistema de argumentación rebatible presenta los siguientes cinco elementos (en algunos casos de manera implícita):

- **Lenguaje lógico de representación:** Los sistemas argumentativos suelen estar apoyados en un lenguaje lógico subyacente, que es utilizado para representar la información acerca del dominio en que se basará la argumentación. Asociada a este lenguaje lógico se define una noción de consecuencia lógica, la cual es central para la construcción de los argumentos (siguiente elemento de la estructura conceptual). Algunos sistemas argumentativos adoptan una lógica particular [GS04, BH01a], mientras que otros dejan la lógica subyacente parcialmente o completamente sin especificar [Dun95]. Estos últimos pueden ser instanciados con diferentes lógicas alternativas, por lo que son considerados más bien marcos que sistemas [Pra10].

- **Construcción de argumentos:** Un argumento constituirá una prueba a partir de la lógica subyacente y el operador de consecuencia lógica mencionado en el elemento anterior de la estructura conceptual. En general, en la literatura un argumento se construye a partir de una secuencia de prueba (o derivación) [PS97, GS04]. A nivel de representación, un argumento suele especificarse como un par premisas-conclusión, dejando implícito el hecho de que existe una prueba para la conclusión a partir de las premisas en la lógica subyacente [GS04]. Algunos formalismos se abstraen de cómo es la construcción y la estructura interna de los argumentos. El sistema de Dung [Dun95] es un ejemplo de estos formalismos, ya que se abstrae tanto de la lógica subyacente como de la estructura interna de los argumentos. Dung trata la noción de argumento como una entidad abstracta que su sistema recibe de antemano.

- **Ataques entre argumentos:** Claramente, argumentación presupone desacuerdo en algún sentido. Esto se captura mediante la noción de conflicto entre argumentos, también llamada contra-argumentación o ataque. El la literatura existen tres reconocidos tipos de conflicto. El primero de ellos, conocido como *rebutting attack* (o ataque por refutación), ocurre cuando dos argumentos poseen conclusiones contradictorias. Un segundo tipo de conflicto, conocido como *assumption attack* (o ataque

a una suposición), se presenta cuando un argumento involucra una suposición de no existencia de prueba (es decir, sustenta su conclusión bajo la suposición de que una fórmula determinada no sea probable, empleando algún tipo de negación default provista por el lenguaje lógico subyacente), y otro argumento prueba la conclusión que fue asumida como no probable por el primero. El tercer tipo de conflicto, conocido como *undercutting attack*, ocurre cuando un argumento cuestiona una regla de inferencia utilizada por otro argumento. Generalmente, los tipos de conflicto arriba mencionados son inferidos a través de las características del lenguaje lógico subyacente. Sin embargo, en los sistemas como [Dun95], donde hay abstracción del lenguaje, se asume que los conflictos entre argumentos se encuentran preestablecidos de antemano.

- **Derrotas entre argumentos:** Para determinar qué argumento será aceptado entre un par de argumentos en conflicto es necesario determinar cuál de ellos está derrotado. Por lo tanto, la noción de derrota constituye en la evaluación comparativa de pares de argumentos en conflicto para establecer si un determinado ataque tiene éxito o no. Esta noción de *ataque exitoso* se formaliza mediante una relación binaria entre argumentos, comúnmente denominada derrota, y definida como *ataca y es tanto o más fuerte*.

  En la comunidad de argumentación no hay un consenso establecido acerca de qué criterio utilizar para evaluar los argumentos. Aún así, uno de los criterios más populares en Inteligencia Artificial es el criterio de especificidad generalizada. Este criterio prefiere argumentos basados en información más específica, valiéndose únicamente de la estructura lógica interna del argumento y abstrayéndose de la información del dominio. Sin embargo, otras propuestas sugieren que la información acerca del dominio suele ser la herramienta principal para evaluar los argumentos. Por esta razón, muchos sistemas argumentativos se encuentran parametrizados con respecto al criterio de comparación [GS04], el que se espera sea provisto por el usuario y esté relacionado con el dominio de aplicación.

- **Argumentos aceptables** El objetivo de un sistema argumentativo consiste en determinar qué argumentos serán finalmente aceptados. Para esto será necesario analizar las derrotas que haya entre los argumentos. No obstante, no alcanza con sólo considerar las derrotas directas al argumento bajo análisis, sino que también es necesario considerar las derrotas para sus derrotadores, las derrotas para los

derrotadores de sus derrotadores, y así sucesivamente. Por ejemplo, un argumento $\mathcal{A}$ que está derrotado por otro argumento $\mathcal{B}$ estará aceptado si existe un argumento $\mathcal{C}$ que derrota a $\mathcal{B}$, y $\mathcal{C}$ no está derrotado.

Existen muchas variantes acerca de cómo calcular la aceptabilidad de los argumentos de un sistema argumentativo. En particular, estas se pueden dividir en dos categorías: las aproximaciones declarativas y las aproximaciones procedurales. En las aproximaciones declarativas se proveen ciertas condiciones que un conjunto de argumentos aceptables debe cumplir, sin especificar cómo se computarán tales conjuntos. Estos métodos son normalmente conocidos como semánticas de aceptabilidad, y poseen una aproximación extensional en sus definiciones. Por otra parte, las aproximaciones procedurales brindan un procedimiento para determinar si un argumento determinado es aceptable o no. En general, estos métodos suelen simular el juego argumentativo que ocurre en una discusión, y se modela a través de árboles de argumentos conocidos como árboles de dialéctica.

3.2. Marcos Argumentativos Abtractos

En esta sección se presentarán los marcos argumentativos abstractos presentados por Phan Minh Dung en [Dun95]. A diferencia de los sistemas argumentativos concretos (*e.g.*, DeLP [GS04]) que caracterizan todos los elementos de la estructura conceptual introducida en la Sección 3.1.1, los marcos argumentativos abstractos sólo se concentran en el último elemento de dicha estructura, es decir, en determinar qué argumentos son aceptables. Por lo tanto, estos marcos asumen la existencia de un conjunto de argumentos y una relación de derrota entre ellos, abstrayéndose así de cómo estos elementos son determinados. Cabe destacar que esta característica permite que los elementos sin especificar se pueden instanciar de distintas formas, dando así origen a un grupo de formalismos diferentes.

Dada la simpleza y gran nivel de abstracción de estos marcos, en la actualidad son utilizados como base teórica para la gran mayoría de los avances en área de argumentación, especialmente en aquellos que presentan nuevas técnicas para determinar argumentos aceptables [BG09]. Adicionalmente, como las técnicas de aceptabilidad presentadas para estos marcos, en especial las presentadas en [Dun95], son ampliamente aceptadas es común que nuevos formalismos argumentativos más concretos (*i.e.* que contemplan más elementos

de la estructura conceptual) busquen caracterizar marcos argumentativos abstractos. De esta manera, los nuevos sistemas podrán utilizar todas las nociones de aceptabilidad y los resultados desarrollados para los marcos argumentativos abstractos. En particular, se verá que los formalismos argumentativos desarrollados en estas tesis, en parte, siguen este enfoque.

Por estas razones, se han presentado muchas extensiones sobre los marcos argumentativos abstractos, como por ejemplo los marcos argumentativos basados en preferencias [AC02], los marcos argumentativos basados en valores [BC03], o los marcos argumentativos bipolares [ACLSL08]. En particular, los marcos argumentativos basados en múltiples tipos [GGS11] desarrollados en el Capítulo 4 de esta tesis también pueden verse como una extensión de los marcos argumentativos abstractos.

3.2.1. Especificación del marco argumentativo

La noción principal de la teoría de Dung es la de marco argumentativo, que se define como un par ordenado compuesto por un conjunto de argumentos y una relación binaria de derrota[1] Un argumento es una entidad abstracta indivisible cuyo único rol está determinado por su relación con otros argumentos. La relación de derrota se define como un conjunto de pares de argumentos, donde cada par representa una derrota de la primera componente sobre la segunda.

Definición 3.1 (Marco Argumentativo Abstracto) *Un marco argumentativo abstracto (MA) es una tupla* $MA = (\mathsf{AR}, \rightarrow)$*, donde* AR *es un conjunto finito de argumentos y* $\rightarrow \subseteq (\mathsf{AR} \times \mathsf{AR})$ *es la relación de derrota sobre* AR*.*

En general, los argumentos en AR para un marco argumentativo MA son denotados a través letras mayúsculas caligráficas ($\mathcal{A}, \mathcal{B}, \ldots, \mathcal{A}_j, \mathcal{C}_i, \ldots$). No se hará ninguna referencia a la lógica subyacente necesaria para la representación y construcción de estos argumentos ya que, como se mencionó anteriormente, este formalismo se abstrae de su estructura interna. Dados dos argumentos $\mathcal{A}$ y $\mathcal{B}$, la relación de derrota entre ellos $(\mathcal{A}, \mathcal{B}) \in \rightarrow$, notado $\mathcal{A} \rightarrow \mathcal{B}$, representará que el argumento $\mathcal{A}$ derrota al argumento $\mathcal{B}$. Al igual que lo

[1] En el trabajo original [Dun95] se la llama relación de *ataque*. Sin embargo, los ataques en un marco argumentativo modelan el concepto que en esta tesis (y la gran mayoría de los sistemas argumentativos [PS97, Pra10, GS04, RS09]) se conoce como derrota, es decir, un ataque que resulta efectivo.

ocurrido con el conjunto de argumentos, los marcos argumentativos abstractos se abstraen de cómo se genera esta relación.

Los marcos argumentativos abstractos son, esencialmente, un grafo dirigido en el que los nodos son argumentos y los arcos representan las derrotas entre los argumentos. En la literatura suele hacerse referencia a este grafo como grafo de derrotas asociado al marco argumentativo. En adelante, dado un marco MA, se hará referencia a dicho grafo como grafo de derrotas asociado a MA o grafo de MA indistintamente.

Ejemplo 3.1 *Considere el marco argumentativo* $MA_1 = (AR_1, \rightarrow_1)$*, donde* $AR_1 = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{F}, \mathcal{G}\}$ *y* $\rightarrow_1 = \{(\mathcal{E}, \mathcal{A}), (\mathcal{F}, \mathcal{A}), (\mathcal{B}, \mathcal{E}), (\mathcal{C}, \mathcal{F}), (\mathcal{D}, \mathcal{G})\}$*. El grafo de derrotas asociado a* MA_1 *se presenta a continuación en la Figura 3.1.*

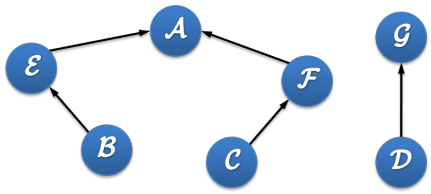


Figura 3.1: Grafo de derrotas para MA_1.

3.2.2. Semánticas de aceptabilidad en los marcos argumentativos abstractos

En un marco argumentativo abstracto, cuando un argumento derrota a otro, estos argumentos no deberían ser aceptados en conjunto. Por lo tanto, el estado de aceptabilidad de un argumento estará sujeto a una evaluación de sus derrotadores, de los derrotadores de sus derrotadores, y así sucesivamente. Esto se debe a que un argumento debería ser aceptado sólo si "sobrevive" de alguna manera a las derrotas que recibe o ser rechazado en otro caso. Este proceso de evaluación es definido por las semánticas de aceptabilidad.

En esta tesis se considerarán las cuatro semánticas de aceptabilidad propuestas originalmente por Dung [Dun95], las semánticas *completa, grounded, estable* y *preferida*, las cuales son ampliamente reconocidas y utilizadas en la literatura de argumentación [RS09].

En la literatura se presentan dos formas de definir las semánticas de aceptabilidad para estos marcos: el enfoque basado en asignación de estado y el enfoque basado en extensiones. El primero, presentado en [Pol94], define una semántica caracterizando asignaciones

de estado (*In, Out, Undecided*) para los argumentos a evaluar. El segundo enfoque, presentado en [Dun95], consiste en caracterizar declarativamente uno o varios conjuntos de argumentos, denominados extensiones, que se consideran colectivamente aceptados de acuerdo a la semántica adoptada. De esta manera, una extensión en el segundo enfoque es un conjunto de argumentos que intuitivamente son aceptables en conjunto bajo una determinada semántica. En esta tesis se seguirá el segundo enfoque para definir las semánticas, es decir, el enfoque basado en extensiones.

Para definir cómo se determinan las extensiones para estas semánticas será necesario introducir algunos conceptos preliminares. En primer lugar se presenta la noción de aceptabilidad. Dung postula que un argumento $\mathcal{A}$ es aceptable para un agente racional G si G puede defender a $\mathcal{A}$ de todas las derrotas que $\mathcal{A}$ recibe. Esta intuición se formaliza a través la siguiente definición.

Definición 3.2 (Argumento aceptable) *Sea $MA = (\mathsf{AR}, \rightarrow)$ un marco argumentativo abstracto. Un argumento $\mathcal{A}$ de AR es aceptable con respecto a un conjunto de argumentos $S \subseteq \mathsf{AR}$ si y solo si para cada argumento $\mathcal{B}$ de AR vale que si $\mathcal{B}$ derrota a $\mathcal{A}$, entonces $\mathcal{B}$ es derrotado por un argumento $\mathcal{C}$ tal que $\mathcal{C} \in S$.*

Por otra parte, sea cual sea la semántica adoptada, únicamente un conjunto de argumentos libre de conflictos entre sus argumentos será aceptable. Por lo tanto, es necesario definir la noción de libertad de conflictos para un conjunto de argumentos.

Definición 3.3 (Conjunto libre de conflictos) *Sea $MA = (\mathsf{AR}, \rightarrow)$ un marco argumentativo abstracto. Un conjunto de argumentos $S \subseteq \mathsf{AR}$ se dice libre de conflictos si y solo si no existen argumentos $\mathcal{A}$ y $\mathcal{B}$ en S tales que $\mathcal{A} \rightarrow \mathcal{B}$.*

Combinado estas dos nociones es posible definir el concepto de conjunto admisible. Básicamente, un conjunto admisible es aquel que, además de ser libre de conflictos, defiende a todos sus argumentos.

Definición 3.4 (Conjunto admisible) *Sea $MA = (\mathsf{AR}, \rightarrow)$ un marco argumentativo abstracto. Un conjunto de argumentos $S \subseteq \mathsf{AR}$ es admisible si y solo si S es un conjunto libre de conflictos y todo argumento $\mathcal{A}$ de S es tal que $\mathcal{A}$ es aceptable con respecto a S.*

Ejemplo 3.2 *Considere el marco argumentativo MA_1 presentado en el Ejemplo 3.1. Algunos de los conjuntos admisibles de MA_1 son: $\{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$, $\{\mathcal{B}\}$, $\{\mathcal{D}\}$, y $\{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}\}$. Mientras que $\{\mathcal{A}, \mathcal{B}\}$ no es admisible ya que, si bien es libre de conflictos, no defiende a $\mathcal{A}$ de la derrota de $\mathcal{F}$.*

El concepto de admisibilidad es esencial ya que permite definir la semántica completa, la cual es utilizada para definir las semánticas estable, preferida y grounded, en los mismos términos. Además, esta noción es importante dado cada extensión bajo alguna de estas semánticas es admisible. A continuación se detallan estas semánticas de acuerdo a como fueron introducidas por Dung en [Dun95], y posteriormente refinadas en [CA07].

En primer lugar se presentará la semántica completa, la cual se caracteriza a través de las extensiones completas. Estas extensiones básicamente consisten de conjuntos admisibles de argumentos, los cuales incluyen a todos los argumentos que defienden.

Definición 3.5 (Extensión Completa) *Sea $MA = (\mathsf{AR}, \rightarrow)$ un marco argumentativo abstracto. Un conjunto de argumentos $S \subseteq \mathsf{AR}$ es una extensión completa de MA si y solo si S es un conjunto admisible tal que para todo argumento $\mathcal{A}$ de AR, si $\mathcal{A}$ es aceptable con respecto a S entonces $\mathcal{A}$ está en S.*

Ejemplo 3.3 *Considere el marco argumentativo MA_1 del Ejemplo 3.1. Observe que $\{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}\}$ es la única extensión completa de MA_1. Por otra parte, por ejemplo, $\{\mathcal{B}, \mathcal{C}\}$ no es una extensión completa ya que, si bien es un conjunto admisible, no incluye a $\mathcal{A}$ (al cual defiende de las derrotas de $\mathcal{E}$ y $\mathcal{F}$) ni tampoco a $\mathcal{D}$ (al que defiende trivialmente ya que nadie lo derrota).*

Es posible que existan múltiples extensiones completas para un marco argumentativo abstracto. A continuación se mostrarán ejemplos de estas situaciones.

Ejemplo 3.4 *Considere los marcos argumentativos MA_2 y MA_3 caracterizados por los grafos de derrota de las figuras 3.2(a) y 3.2(b).*

(a) (b)

Figura 3.2: Grafos de derrotas para MA_2 y MA_3.

Note que en MA_2 las extensiones completas son $\emptyset$, $\{\mathcal{H}\}$ e $\{\mathcal{I}\}$, mientras que las extensiones completas de MA_3 son $\emptyset$, $\{\mathcal{J}, \mathcal{L}\}$ y $\{\mathcal{K}\}$.

La primera de las semánticas principales para los marcos argumentativos abstractos presentadas por Dung en [Dun95] es la semántica preferida. Esta semántica se identifica a través de las extensiones preferidas. Intuitivamente, una extensión preferida representa un conjunto de argumentos coherente y maximal, ya que debe ser libre de conflictos y clasificar como aceptados a la mayor cantidad posible de argumentos.

Definición 3.6 (Extensión preferida) *Sea $MA = (\mathsf{AR}, \rightarrow)$ un marco argumentativo abstracto. Un conjunto de argumentos $S \subseteq \mathsf{AR}$ es una extensión preferida de MA si y solo si S es una extensión completa maximal con respecto a la inclusión de conjuntos.*

Ejemplo 3.5 *Considere los marcos argumentativos MA_1, MA_2 y MA_3, presentados en los ejemplos 3.1 y 3.4. En MA_1 $\{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}\}$ es la única extensión preferida; en MA_2 $\{\mathcal{H}\}$ e $\{\mathcal{I}\}$ son las extensiones preferidas; mientras que en MA_3 las extensiones preferidas son $\{\mathcal{J}, \mathcal{L}\}$ y $\{\mathcal{K}\}$.*

Otra de las semánticas presentadas por Dung en [Dun95] es la semántica estable. Esta semántica es más restrictiva que la preferida, ya que impone que únicamente serán extensiones estables aquellas extensiones preferidas que derrotan a todos los argumentos fuera de la extensión.

Definición 3.7 (Extensión estable) *Sea $MA = (\mathsf{AR}, \rightarrow)$ un marco argumentativo abstracto. Un conjunto de argumentos $S \subseteq \mathsf{AR}$ es una extensión estable de MA si y solo si para todo argumento $\mathcal{B}$ en $\mathsf{AR} \setminus S$ existe un argumento $\mathcal{A}$ en S tal que $\mathcal{A}$ derrota a $\mathcal{B}$.*

En los marcos argumentativos de los ejemplos 3.1 y 3.4 note que las extensiones estables coinciden con las preferidas. Esto no es siempre así, más aún, Dung en [Dun95] demuestra que todas las extensiones estables son preferidas, pero que no todas las extensiones preferidas son estables.

Ejemplo 3.6 *Considere el marco argumentativo $MA_4 = (\{\mathcal{M}\}, \{(\mathcal{M}, \mathcal{M})\})$. En MA_4, $\emptyset$ es una extensión preferida. Sin embargo, $\emptyset$ no es una extensión estable de MA_4 ya que no derrota a $\mathcal{M}$.*

Finalmente, la última semántica presentada por Dung en [Dun95] es la semántica grounded. Esta semántica tiene la característica de ser la mas escéptica de las semánticas propuestas por Dung. Por lo tanto, corresponderá a la extensión más chica que contenga a todos los elementos que defiende.

Definición 3.8 (Semántica grounded) *Sea $MA = (AR, \rightarrow)$ un marco argumentativo abstracto. Un conjunto de argumentos $S \subseteq AR$ es la extensión grounded de MA si y solo si S es la extensión completa minimal con respecto a la inclusión de conjuntos.*

Ejemplo 3.7 *Considere los marcos argumentativos MA_5 y MA_6 caracterizados por los grafos de derrota de las figuras 3.3(a) y 3.3(b).*

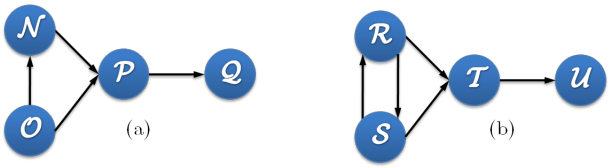


Figura 3.3: Grafos de derrotas para MA_5 y MA_6.

La única extensión completa de MA_5 es $\{\mathcal{O}, \mathcal{Q}\}$, mientras que en MA_6 las extensiones completas son $\emptyset$, $\{\mathcal{R},\mathcal{U}\}$ y $\{\mathcal{S},\mathcal{U}\}$. Note que la extensión grounded en MA_5 será $\{\mathcal{O}, \mathcal{Q}\}$, mientras que en MA_6 será $\emptyset$.

Si bien la semántica grounded es la mas escéptica, a diferencia de las otras semánticas caracteriza una única extensión siempre existente. Adicionalmente, otra característica interesante de esta semántica es que, como se muestra en [Dun95], la extensión grounded se puede computar utilizando una teoría de punto fijo, al igual que en numerosos sistemas de razonamiento no monótono.

Aplicando una semántica a un marco argumentativo abstracto es posible entonces determinar cuáles serán los argumentos aceptados. En particular, como se vio anteriormente, para algunos marcos argumentativos pueden obtenerse múltiples extensiones al aplicar ciertas semánticas. Por lo tanto, los argumentos de un marco argumentativo podrán estar crédulamente aceptados, escépticamente aceptados o rechazados.

Definición 3.9 *Sea $MA = (\mathsf{AR}, \rightarrow)$ un marco argumentativo y S una semántica de aceptabilidad. Un argumento $\mathcal{A}$ de AR estará:*

- *escépticamente aceptado con respecto a S, si pertenece a toda extensión obtenida al aplicar la semántica S a MA,*
- *crédulamente aceptado con respecto S, si pertenece a alguna extensión (pero no a todas) obtenida al aplicar la semántica S a MA, o*
- *rechazado con respecto a S, si no pertenece a ninguna extensión obtenida al aplicar la semántica S a MA.*

3.2.3. Marcos Argumentativos Abstractos con Preferencias

Como se mencionó al comienzo de la Sección 3.2, los marcos argumentativos abstractos han sido extendidos de diversas maneras. Una de ellas considera utilizar preferencias para determinar las derrotas entre los argumentos de un marco argumentativo [AC02, KvdT08, MGS08]. Es decir, en esta aproximación los marcos argumentativos, en lugar de caracterizar una relación de derrota entre los argumentos, introducen una relación de conflicto o ataque y una relación de preferencia entre los argumentos. Luego, la relación de derrota es inferida a partir de estas relaciones de ataque y preferencia. Esta extensión a los marcos argumentativos abstractos es de particular interés para esta tesis ya que, como se verá en el Capítulo 4, los marcos argumentativos basados en preferencias serán utilizados para construir los marcos argumentativos de tipos múltiples.

De manera similar a los marcos argumentativos abstractos de [Dun95], los marcos argumentativos basados en preferencias de [AC02, KvdT08, MGS08] no consideran todos los elementos de la estructura conceptual mostrada en la sección 3.1.1, sino que se concentran básicamente en los últimos dos elementos: determinar las derrotas entre los argumentos y establecer qué argumentos son aceptables. Por lo tanto, en estos marcos, tanto los argumentos como las relaciones de ataque y preferencia entre ellos se asumirán provistas, abstrayéndose de cómo se obtuvieron.

Definición 3.10 (Marco argumentativo basado en preferencias) *Un marco argumentativo basado en preferencias (MAP) es una tupla $MAP = (\mathsf{AR}, \rightharpoonup, \geq)$, donde AR es un conjunto finito de argumentos, $\rightharpoonup \subseteq \mathsf{AR} \times \mathsf{AR}$ es una relación de ataque entre argumentos , y $\geq \subseteq \mathsf{AR} \times \mathsf{AR}$ es una relación de preferencia entre argumentos.*

Como se mencionó anteriormente, a partir de los ataques y las preferencias en un marco argumentativo basado en preferencias es posible determinar la relación de derrota entre los argumentos del marco. Intuitivamente, un argumento $\mathcal{A}$ derrotará a otro argumento $\mathcal{B}$ si lo ataca y su ataque es efectivo, es decir, si $\mathcal{A}$ es tanto o más preferido a $\mathcal{B}$. Note que para cada par $(\mathcal{A}, \mathcal{B}) \in \geq$ se tiene que el argumento $\mathcal{A}$ es al menos tan preferido como el argumento $\mathcal{B}$, lo cual se nota como $\mathcal{A} \geq \mathcal{B}$. De manera similar, la existencia de un par $(\mathcal{A}, \mathcal{B}) \in \rightharpoonup$ implica que el argumento $\mathcal{A}$ ataca al argumento $\mathcal{B}$, y se nota como $\mathcal{A} \rightharpoonup \mathcal{B}$.

Definición 3.11 (Derrota en los MAPs) *Sea* $MAP = (\mathsf{AR}, \rightharpoonup, \geq)$ *un marco argumentativo basado en preferencias. La relación de derrota* $\rightarrow_P \subseteq \mathsf{AR} \times \mathsf{AR}$ *en* MAP *es tal que* $(\mathcal{A}, \mathcal{B}) \in \rightarrow_P$ *si y solo si* $\mathcal{A} \rightharpoonup \mathcal{B}$*, y si* $\mathcal{B} \geq \mathcal{A}$ *entonces* $\mathcal{A} \geq \mathcal{B}$*.*

Ejemplo 3.8 *Considere el marco argumentativo* $MAP_1 = (\ \{\mathcal{V}, \mathcal{X}, \mathcal{Y}\}, \{(\mathcal{X}, \mathcal{V}), (\mathcal{V}, \mathcal{X}), (\mathcal{X}, \mathcal{Y})\}, \{(\mathcal{V}, \mathcal{X})\})$*. En la Figura 3.4(a) se caracteriza el grafo con los argumentos y los ataques establecidos en* MAP_1*, mientras que en la Figura 3.4(b) se ilustran las derrotas entre estos argumentos, las cuales resultan de considerar la relación de preferencia en* MAP_1*,que establece que* $\mathcal{V} \geq \mathcal{X}$*.*

Figura 3.4: Grafo de ataques y grafo de derrotas para MAP_1.

Caracterizando la relación de derrota es posible determinar qué argumentos son aceptables en un marco argumentativo basado en preferencias. Para esto, en [KvdT08] se identifica el marco argumentativo abstracto que representa al marco argumentativo basado en preferencias, para luego aplicar las semánticas de aceptabilidad sobre el marco argumentativo abstracto, como se presentó en la Sección 3.2.2. Por lo tanto, a continuación se definirá cuál es el marco argumentativo abstracto representante para un marco argumentativo basado en preferencias.

Definición 3.12 (Marco argumentativo representante de un MAP) *Sea* $MAP = (\mathsf{AR}, \rightharpoonup, \geq)$ *un marco argumentativo basado en preferencias. Si* $\rightarrow_P$ *es la relación de derrota para* MAP*, entonces el marco argumentativo abstracto* MA *que representa a* MAP *se define como* $MA = (\mathsf{AR}, \rightarrow_P)$*.*

Ejemplo 3.9 *Considere el marco argumentativo basado en preferencias MAP_1 presentado en el Ejemplo 3.8. El marco argumentativo abstracto representante de MAP_1 será $MA = (\{\mathcal{V}, \mathcal{X}, \mathcal{Y}\}, \{(\mathcal{V}, \mathcal{X}), (\mathcal{X}, \mathcal{Y})\})$. Observe que a partir de MA las extensiones grounded, preferida y estable coinciden y corresponden al conjunto $\{\mathcal{V}, \mathcal{Y}\}$.*

3.3. Argumentación en la programación en lógica rebatible - DeLP

La Programación en Lógica Rebatible (DeLP) [GS04] es un formalismo de representación de conocimiento y razonamiento que combina resultados de la programación en lógica y la argumentación rebatible, que ha sido aplicado exitosamente en diferentes dominios concretos (*e.g.*, [GGS08, RGS07, GGS10, FEGS07, GCS08, SMCS08, CMS06]). A diferencia de los marcos argumentativos abstractos presentados en la sección anterior, el formalismo DeLP representa todos los elementos de la estructura conceptual para sistemas argumentativos presentada en la Sección 3.1.1. DeLP adopta como lenguaje lógico subyacente (el primer elemento de la estructura conceptual) una extensión de la programación en lógica, para incluir negación fuerte y representar información rebatible (además de estricta). A partir de estos elementos se construyen argumentos, se identifican conflictos (considerando la negación fuerte) entre los argumentos, se determinan las correspondientes derrotas, y se utilizan procedimientos de prueba dialéctica para determinar qué argumentos son aceptados.

En esta sección se presentará la versión DeLP que únicamente utiliza reglas rebatibles y hechos [2]. El motivo de esta elección radica en que esta versión de DeLP se ha mostrado adecuada para entornos dinámicos [CCS05, RGS07, GGS09, GGS10] como son los tratados en esta tesis, en especial, en agentes inteligentes. En particular, esta versión de DeLP será utilizada como base para el desarrollo del sistema argumentativo que emplea tipos de argumento desarrollado en el Capítulo 5.

Un programa lógico rebatible está formado por un conjunto de *hechos*, y *reglas rebatibles*, permitiendo estas últimas la representación de información tentativa. Toda conclusión del programa deberá ser sustentada por algún *argumento* que pueda construirse utilizando las reglas y los hechos del programa. Cuando se utilicen reglas rebatibles para

[2] En el trabajo original de DeLP [GS04] también se incluyen reglas estrictas.

derivar una conclusión C, esta conclusión será tentativa y podrá ser refutada por información que la contradiga. Así, un argumento que sustenta una conclusión C podrá ser atacado por otros argumentos *derrotadores* que lo contradigan y que puedan construirse a partir del programa. Dichos derrotadores podrán a su vez ser atacados, y así sucesivamente. Esto llevará a una estructura arbórea de derrotadores conocida como árbol de dialéctica. Entonces, para decidir cuándo una conclusión C puede aceptarse a partir de un programa lógico rebatible, se realizará un *análisis dialéctico*, construyendo estos árboles de dialéctica para los argumentos a favor y en contra de la conclusión C.

A continuación se introducirán los conceptos preliminares de la programación en lógica que son utilizados tanto por DeLP como por el formalismo desarrollado en el Capítulo 5. Luego se introducirá la sintaxis del lenguaje de los programas lógicos rebatibles de DeLP, los cuales permiten representar información tentativa y potencialmente contradictoria. Posteriormente se presentará un formalismo de *argumentación rebatible* para definir un procedimiento de prueba para DeLP.

3.3.1. Conceptos Preliminares

En esta sección se introducirán los conceptos y terminología estándar de lógica y programación en lógica, que son necesarios para presentar DeLP y el formalismo desarrollado en el Capítulo 5.

Definición 3.13 (Signatura) *Una signatura es una tupla $\sigma = \langle \mathcal{V}, Func, Pred \rangle$ donde $\mathcal{V}$, $Func$, y $Pred$ son conjuntos finitos que representan variables, funciones y predicados respectivamente.*

Dada una signatura σ, una función llamada "*aridad*" le asignará a cada elemento de $Func$ y $Pred$ un número entero positivo. Si $f \in Func$ y $aridad(f) = 0$, entonces f se denominará *constante*, por otra parte, si $p \in Pred$, y $aridad(p) = 0$, entonces p se denominará *proposición*.

Definición 3.14 (Alfabeto) *El alfabeto generado a partir de una signatura σ, consiste del conjunto de elementos miembros de la signatura, el símbolo de negación "$\sim$", y los símbolos de puntuación "(", ")", y ",".*

Definición 3.15 (Término) *Sea $\sigma = \langle \mathcal{V}, Func, Pred \rangle$ una signatura. Un término T, y respectivamente sus componentes $comp(T)$, se definen inductivamente como sigue:*

- *toda variable $V \in \mathcal{V}$, es un término, y $comp(V) = \{V\}$;*
- *toda constante $c \in Func$ ($aridad(c) = 0$), es un término, y $comp(c) = \{c\}$;*
- *si $f \in Func$, $aridad(f) = n$ con $n \geq 1$, y $t_1, \ldots, t_n$ son términos, entonces $f(t_1, \ldots, t_n)$ también es un término.*
  Se define $comp(f(t_1, \ldots, t_n)) = \{f\} \cup (\bigcup_{1 \leq i \leq n} comp(f_i))$.

Como se verá a continuación, en DeLP los términos no utilizarán variables, motivo por el cual se introducirá la siguiente definición.

Definición 3.16 (Término fijo) *Sea $\sigma = \langle \mathcal{V}, Func, Pred \rangle$ una signatura y T un término. T será un término fijo si no contiene variables. Esto es, $comp(T) \cap \mathcal{V} = \emptyset$.*

Definición 3.17 (Átomo) *Sea $\sigma = \langle \mathcal{V}, Func, Pred \rangle$ una signatura, y $t_1, \ldots, t_n$ términos. Si $p \in Pred$ con $aridad(p) = n$, entonces $p(t_1, \ldots, t_n)$ es un átomo. Un átomo fijo es un átomo $p(t_1, \ldots, t_n)$, donde todos sus términos $t_1, \ldots, t_n$ son fijos.*

Definición 3.18 (Literal) *Un literal L es un átomo "A" o un átomo negado "$\sim A$", donde "$\sim$" representa la negación fuerte. Un literal L se dirá negativo si es un átomo negado, y positivo en caso contrario. Un literal fijo es un átomo fijo o un átomo fijo negado.*

Los literales podrán ser átomos negados utilizando el símbolo "$\sim$" de la *negación fuerte*. Note que este tipo de negación no debe confundirse con la negación por falla (o negación default) utilizada en la Programación en Lógica, y que normalmente se denota con el símbolo "not".

Definición 3.19 (Complemento de un literal) *Sea L un literal y A un átomo. El complemento de L con respecto a la negación fuerte, denotado $\overline{L}$, se define de la siguiente manera:*

- *si $L = A$, entonces $\overline{L} = \sim A$;*
- *si $L = \sim A$, entonces $\overline{L} = A$.*

3.3.2. Representación de conocimiento en DeLP

Un programa lógico rebatible estará compuesto por *hechos* y *reglas rebatibles*, cuya sintaxis se define a continuación.

Definición 3.20 (Hecho) *Un hecho es un literal fijo L. Esto es, un átomo fijo, o un átomo fijo negado.*

Definición 3.21 (Regla Rebatible) *Una regla rebatible es un par ordenado, denotado "Cabeza $-\prec$ Cuerpo", donde el primer elemento, Cabeza, es un literal y el segundo elemento, Cuerpo, es un conjunto finito de literales. Una regla rebatible con cabeza L_0 y cuerpo $\{L_1, \ldots, L_n\}$ $(n \geq 0)$ se escribirá también como: $L_0 -\prec L_1, \ldots, L_n$.*

Definición 3.22 (Programa Lógico Rebatible)
Un programa lógico rebatible es un conjunto $\mathcal{P}$ de hechos no contradictorios y reglas rebatibles. En un programa $\mathcal{P}$ se identificará con Θ al conjunto de hechos, y con Δ al conjunto de reglas rebatibles. Por lo tanto, en algunas ocasiones se denotará a un programa lógico rebatible $\mathcal{P}$ con el par $\mathcal{P} = (\Theta, \Delta)$.

Pragmáticamente, los hechos se emplean para representar conocimiento seguro, libre de excepciones, mientras que las reglas rebatibles son reglas de inferencia que se utilizan para representar información tentativa, la cual puede ser cuestionada. En general, una regla rebatible $Cabeza -\prec Cuerpo$ se leerá como "razones para creer en $Cuerpo$ proveen razones para creer en $Cabeza$".

Ejemplo 3.10 *El programa lógico rebatible $\mathcal{P}_1$ presentado a continuación representa información sobre el mercado de valores.*

$$comprarAcciones(T) -\prec buenPrecio(T)$$
$$\sim comprarAcciones(T) -\prec buenPrecio(T), empresaRiesgosa(T)$$
$$empresaRiesgosa(T) -\prec enFusion(T)$$
$$empresaRiesgosa(T) -\prec conDeuda(T)$$
$$buenPrecio(acme)$$
$$enFusion(acme)$$

Note que, a diferencia de los hechos, los literales en las reglas rebatibles pueden tener variables. No obstante, DeLP emplea instancias fijas de estas reglas para determinar los literales que deriva. Una instancia fija para una regla es una versión de la regla tal que las variables son reemplazadas por términos fijos, donde se asume que las variables con el mismo nombre dentro de la regla representan el mismo elemento. Cada regla rebatible con variables constituye entonces un esquema que representa al conjunto de sus instancias fijas.

Definición 3.23 (Instancia fija de una regla rebatible) *Sea R una regla rebatible. Una instancia fija $R\sigma$ de R se obtiene reemplazando cada una de las variables de R por un término fijo, de forma tal que las variables de R con el mismo nombre sean reemplazadas consistentemente.*

A continuación se introducirá el concepto de *derivación rebatible*, con el cual se define qué literales pueden ser derivados utilizando las reglas de un programa lógico rebatible.

Definición 3.24 (Derivación Rebatible de un literal) *Sea $\mathcal{P} = (\Theta, \Delta)$ un programa y L un literal. Una derivación rebatible para L a partir de $\mathcal{P}$, consiste de una secuencia finita de literales fijos $L_1, L_2, \ldots, L_n = L$, tales que cada literal L_i pertenece a la secuencia porque:*

(a) L_i es un hecho, o

(b) existe en $\mathcal{P}$ una instancia fija $R_i\sigma$ de una regla rebatible R_i con cabeza L_i y cuerpo $B_1, B_2, \ldots, B_k$, donde todo literal B_j del cuerpo $(1 \leq j \leq k)$ es un elemento de la secuencia que precede a L_i.

Por ejemplo, a partir del programa del Ejemplo 3.10, es posible obtener una derivación rebatible para el literal "$comprarAcciones(acme)$", ya que existe la secuencia de literales: $buenPrecio(acme)$, $comprarAcciones(acme)$; que se obtiene utilizando el hecho "buenPrecio(acme)" y la regla rebatible "$comprarAcciones(acme) -\!\prec buenPrecio(acme)$". Además, observe que también es posible obtener una derivación para el literal "$\sim comprarAcciones(acme)$", ya que existe la secuencia de literales: $enFusion(acme), empresaRiesgosa(acme), buenPrecio(acme), \sim comprarAcciones(acme)$. Note entonces que es posible derivar literales complementarios a partir de un programa DeLP.

Este ejemplo muestra, que la noción de derivación rebatible no es conveniente como procedimiento de prueba de un literal. Por lo tanto, para decidir si un literal L está aceptado a partir de un programa, se realizará un *análisis global* más profundo que involucre toda la información relevante del programa.

En particular un *procedimiento de prueba* es el que proveerá a DeLP de un mecanismo de análisis global para decidir qué literales serán aceptados a partir de un programa. Este procedimiento de prueba está basado en el formalismo de *argumentación rebatible*, y permitirá definir *argumentos* para un literal y *contra-argumentos* para los argumentos obtenidos. Luego, un *análisis dialéctico* permitirá decidir cuando un literal está *aceptado*.

3.3.3. Construcción de argumentos en DeLP

En DeLP, un argumento para una conclusión (literal) h se caracteriza por un conjunto minimal y no contradictorio de reglas rebatibles que permiten derivar rebatiblemente h. Observe que esta noción captura las intuiciones del segundo elemento de la estructura conceptual presentada en la Sección 3.1.1.

Definición 3.25 (Argumento) *Sea h un literal fijo y $\mathcal{P} = (\Theta, \Delta)$ un programa lógico rebatible. Un argumento para h es un par $\langle \mathcal{A}, h \rangle$, donde $\mathcal{A}$ es un conjunto de reglas rebatibles de Δ tal que:*

1. *existe una derivación rebatible para h a partir de $\Theta \cup \mathcal{A}$,*
2. *$\Theta \cup \mathcal{A}$ es no contradictorio, y*
3. *$\mathcal{A}$ es minimal, es decir, no existe un subconjunto propio $\mathcal{A}'$ de $\mathcal{A}$ tal que $\mathcal{A}'$ satisface las condiciones (1) y (2).*

Ejemplo 3.11 *A partir del programa del Ejemplo 3.10 es posible construir los argumentos $\langle \mathcal{A}_1, \sim comprarAcciones(acme) \rangle$, $\langle \mathcal{B}_1, empresaRiesgosa(acme) \rangle$, y $\langle \mathcal{A}_2, comprarAcciones(acme) \rangle$, donde:*

$$\mathcal{A}_1 = \left\{ \begin{array}{l} \sim comprarAcciones(acme) \prec buenPrecio(acme), empresaRiesgosa(acme) \\ empresaRiesgosa(acme) \prec enFusión(acme) \end{array} \right\}$$

$$\mathcal{B}_1 = \left\{ \; empresaRiesgosa(acme) \prec enFusión(acme) \; \right\}$$

$$\mathcal{A}_2 = \left\{ \; comprarAcciones(acme) \prec buenPrecio(acme) \; \right\}$$

Note que también $\langle \emptyset, buenPrecio(acme) \rangle$ *y* $\langle \emptyset, enFusion(acme) \rangle$ *son argumentos del programa* $\mathcal{P}_1$.

A partir del Ejemplo 3.11 se puede observar que en DeLP es posible construir argumentos cuyo conjunto de reglas rebatibles es vacío. Estos argumentos son conocidos como *argumentos vacíos* y corresponden a hechos del programa a partir del cual se construyen. DeLP asegura que siempre existirá un argumento vacío por cada hecho de un programa, y que si hay un argumento vacío para un literal h, entonces no habrá ningún argumento para $\overline{h}$.

Un argumento en DeLP puede contener otros argumentos, o subargumentos. Básicamente, un subargumento es un argumento cuyas reglas rebatibles están contenidas en otro argumento.

Definición 3.26 (SubArgumento) *Un argumento* $\langle \mathcal{B}, q \rangle$ *es un subargumento de* $\langle \mathcal{A}, h \rangle$ *si y solo si* $\mathcal{B} \subseteq \mathcal{A}$.

Ejemplo 3.12 *Considere los argumentos presentados en el Ejemplo 3.11. El argumento* $\langle \mathcal{B}_1, empresaRiesgosa(acme) \rangle$, *es un subargumento de* $\langle \mathcal{A}_1, \sim comprarAcciones(acme) \rangle$, *y no es un subargumento* $\langle \mathcal{A}_2, comprarAcciones(acme) \rangle$.

3.3.4. Conflictos en DeLP

En esta sección se determinará bajo qué condiciones los argumentos de DeLP están en conflicto. Por lo tanto, se identificarán los conceptos caracterizados por el tercer elemento de la estructura conceptual presentada en la Sección 3.1.1.

Dos literales complementarios representan información estrictamente contradictoria. En DeLP se dice que estos literales están en *desacuerdo*[3]. De esta manera, dos argumentos estarán en conflicto o se atacarán cuando sustenten conclusiones en desacuerdo. Además, un argumento también estará en conflicto con otro si sustenta una conclusión que esté en desacuerdo con la conclusión de alguno de los subargumentos del otro argumento. Estas intuiciones se capturan a continuación bajo la noción de contra-argumento.

[3]En la versión de DeLP que se presenta en esta sección, el desacuerdo se produce únicamente entre literales complementarios. Por otra parte, en la propuesta original de DeLP, para determinar el desacuerdo se considera todo el conocimiento estricto, incluidas las reglas estrictas.

Definición 3.27 (Contra-argumento o ataque) *Sean $\langle \mathcal{A}_1, h_1 \rangle$ y $\langle \mathcal{A}_2, h_2 \rangle$ argumentos obtenidos a partir de un programa lógico rebatible $\mathcal{P}$. El argumento $\langle \mathcal{A}_1, h_1 \rangle$ contra-argumenta o ataca al argumento $\langle \mathcal{A}_2, h_2 \rangle$ en el literal h, si y solo si existe un subargumento $\langle \mathcal{A}, h \rangle$ de $\langle \mathcal{A}_2, h_2 \rangle$ tal que h y h_1 están en desacuerdo. El argumento $\langle \mathcal{A}, h \rangle$ se llama subargumento de desacuerdo, y el literal h será el punto de contra-argumentación.*

Ejemplo 3.13 *Considere los argumentos del Ejemplo 3.11. El argumento $\langle \mathcal{A}_1, \sim comprarAcciones(acme) \rangle$ es un contra-argumento para $\langle \mathcal{A}_2, comprarAcciones(acme) \rangle$. Note que, en este caso, el subargumento de desacuerdo es el mismo argumento $\langle \mathcal{A}_2, comprar\_acciones(acme) \rangle$, y el punto de contra-argumentación es "$comprarAcciones(acme)$". Observe además que, recíprocamente, $\langle \mathcal{A}_2, comprarAcciones(acme) \rangle$ es un contra-argumento para $\langle \mathcal{A}_1, \sim comprarAcciones(acme) \rangle$.*

En la Definición 3.27 puede ocurrir que el subargumento $\langle \mathcal{A}, h \rangle$ sea el propio argumento $\langle \mathcal{A}_2, h_2 \rangle$, y en este caso se dirá que $\langle \mathcal{A}_1, h_1 \rangle$ contra-argumenta o ataca *directamente* a $\langle \mathcal{A}_2, h_2 \rangle$. Por otra parte, puede ocurrir que $\langle \mathcal{A}, h \rangle$ sea un subargumento propio de $\langle \mathcal{A}_2, h_2 \rangle$, en cuyo caso el ataque se producirá en un punto h interno de $\langle \mathcal{A}_2, h_2 \rangle$, y se dirá que $\langle \mathcal{A}_1, h_1 \rangle$ contra-argumenta o ataca *internamente* a $\langle \mathcal{A}_2, h_2 \rangle$.

A continuación, en la Figura 3.5 se introducirá una notación gráfica que se utilizará en esta tesis, y con ella se ilustrarán los argumentos del Ejemplo 3.11 junto con los respectivos ataques presentados en el Ejemplo 3.13. La notación gráfica utilizada en esta figura caracteriza a los argumentos mediante triángulos, y a los ataques mediante flechas con líneas a trazos. Además, los triángulos dentro de los triángulos mayores denotan subargumentos. La información en el vértice superior de los triángulos corresponde a la conclusión del argumento (subargumento), la información dentro del triángulo corresponde a los literales utilizados para construir el argumento, y el símbolo $-\!\prec$ representa la conexión establecida entre estos literales a través de las reglas rebatibles utilizadas por el argumento. Los argumentos vacíos se denotan solo a través del literal que concluyen.

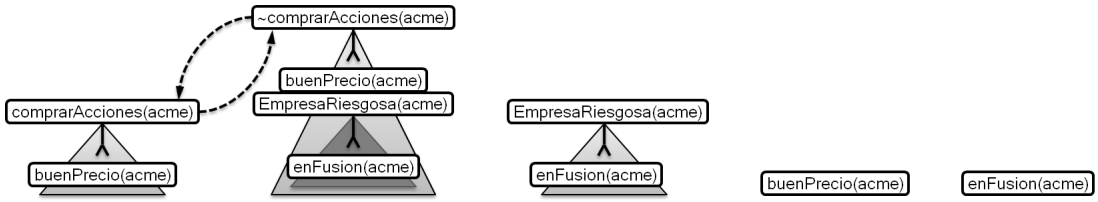


Figura 3.5: Argumentos y ataques del programa DeLP $\mathcal{P}_1$.

Note que los argumentos vacíos nunca tendrán contra-argumentos. Esto se debe a que no será posible construir argumentos para literales que estén en desacuerdo con los hechos, ya que en ese caso no serían argumentos válidos por estar en contradicción con la parte estricta del programa.

La noción de contra-argumentación establece cuándo un argumento es atacado por otro, capturando la noción de conflicto entre argumentos a través de la negación fuerte. Sin embargo, siempre que un argumento $\mathcal{A}$ contra-argumenta a otro argumento $\mathcal{B}$, existe un subargumento de $\mathcal{B}$ que es un contra-argumento para $\mathcal{A}$. Esta simetría no permite indicar cuándo un argumento se impone sobre el otro. Para esto, es indispensable una forma de comparar argumentos, de manera de poder determinar cuándo un argumento es mejor que otro.

3.3.5. Comparación y derrota de argumentos en DeLP

Como se mostró en la estructura conceptual de la Sección 3.1.1, para poder establecer una relación de derrota entre los argumentos de un sistema argumentativo es necesario utilizar un mecanismo de evaluación entre argumentos. En DeLP esta comparación se modela a través de un criterio de comparación de argumentos, el cual es modular[4]. Este criterio establecerá cuándo un argumento es tanto o más preferido que otro argumento. Por lo tanto, como la comparación entre argumentos puede definirse de diversas formas, en lo que resta de esta sección se asumirá que existe un orden parcial "$\succ$"que permite distinguir cuándo un argumento es mejor que otro. Por ejemplo, si $\langle \mathcal{A}_1, h_1 \rangle$ es mejor que $\langle \mathcal{A}, h \rangle$, se lo denotará como "$\langle \mathcal{A}_1, h_1 \rangle \succ \langle \mathcal{A}, h \rangle$".

[4]Se puede utilizar cualquier orden parcial sobre los argumentos del programa, por lo cual es posible diseñar el más adecuado para el problema que se intenta modelar.

Si bien en DeLP el criterio de comparación es modular, existen varios criterios concretos [FEGS08, GS04] para comparar argumentos. En esta tesis, cuando sea necesario, se utilizará el criterio conocido como *especificidad generalizada* desarrollado en [SGCS03]. Al utilizar este criterio, un argumento será preferido a otro siempre y cuando sea más especifico que este otro. A continuación se presentará este criterio en DeLP.

Especificidad Generalizada

En esta sección se presentará el criterio de comparación llamado *especificidad generalizada* desarrollado en [SGCS03], el cual está basado en la noción de especificidad definida en [Poo85, SL92]. Informalmente, un argumento $\langle \mathcal{A}_1, h_1 \rangle$ será mas especifico que otro argumento $\langle \mathcal{A}_2, h_2 \rangle$, si se cumple que $\langle \mathcal{A}_1, h_1 \rangle$ usa mayor información que $\langle \mathcal{A}_2, h_2 \rangle$, o $\langle \mathcal{A}_1, h_1 \rangle$ es más directo (usa menos reglas) que $\langle \mathcal{A}_2, h_2 \rangle$Ṗor ejemplo, $\langle \{\sim a -\prec b, c\}, \sim a \rangle$ sería mas específico que $\langle a -\prec b, a \rangle$ porque usa más información, mientras que $\langle \{a -\prec b\}, a \rangle$ sería más específico que $\langle \sim a -\prec d, d -\prec b, \sim a \rangle$ por ser más directo.

Definición 3.28 (Especificidad generalizada) *Sea $\mathcal{P}$ un programa lógico rebatible y sea $lit(\mathcal{P})$ el conjunto de literales fijos que es posible derivar a partir de $\mathcal{P}$. Un argumento $\langle \mathcal{A}_1, h_1 \rangle$ es estrictamente más específico que un argumento $\langle \mathcal{A}_2, h_2 \rangle$ si y solo si:*

1. *para cada $H \subseteq lit(\mathcal{P})$ tal que $H \cup \mathcal{A}_1 \mid\sim h_1$, con $h_1 \notin H$, vale que $H \cup \mathcal{A}_2 \mid\sim h_2$, y*
2. *existe $H' \subseteq lit(\mathcal{P})$ tal que $H' \cup \mathcal{A}_2 \mid\sim h_2$, con $h_2 \notin H'$, y $H' \cup \mathcal{A}_1 \not\mid\sim h_1$.*

Ejemplo 3.14 *Considere los argumentos del Ejemplo 3.11. Note que el argumento $\langle \mathcal{A}_1, \sim comprarAcciones(acme) \rangle$ es más específico que el argumento $\langle \mathcal{A}_2, comprarAcciones(acme) \rangle$.*

Derrotas en DeLP

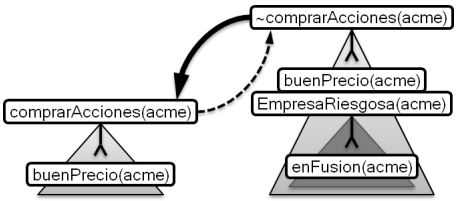
Si $\langle \mathcal{A}_1, h_1 \rangle$ es un contra-argumento para $\langle \mathcal{A}_2, h_2 \rangle$, es necesario analizar si el ataque de $\langle \mathcal{A}_1, h_1 \rangle$ sobre $\langle \mathcal{A}_2, h_2 \rangle$ es lo suficientemente fuerte como para derrotarlo. Como caso particular, $\langle \mathcal{A}_1, h_1 \rangle$ puede atacar directamente a $\langle \mathcal{A}_2, h_2 \rangle$ en el punto h_2, con lo cual deberían compararse los argumentos $\langle \mathcal{A}_1, h_1 \rangle$ y $\langle \mathcal{A}_2, h_2 \rangle$. En general, si $\langle \mathcal{A}_1, h_1 \rangle$ ataca indirectamente a $\langle \mathcal{A}_2, h_2 \rangle$ en el punto h, siendo $\langle \mathcal{A}, h \rangle$ el subargumento de desacuerdo

de $\langle \mathcal{A}_2, h_2 \rangle$, se deben comparar el argumento atacante $\langle \mathcal{A}_1, h_1 \rangle$ y el subargumento de desacuerdo $\langle \mathcal{A}, h \rangle$.

Definición 3.29 (Derrotador) *Sean $\langle \mathcal{A}_1, h_1 \rangle$ y $\langle \mathcal{A}_2, h_2 \rangle$ dos argumentos. El argumento $\langle \mathcal{A}_1, h_1 \rangle$ es un derrotador para el argumento $\langle \mathcal{A}_2, h_2 \rangle$ en el literal h, si y solo si existe un subargumento $\langle \mathcal{A}, h \rangle$ de $\langle \mathcal{A}_2, h_2 \rangle$ tal que $\langle \mathcal{A}_1, h_1 \rangle$ contra-argumenta a $\langle \mathcal{A}_2, h_2 \rangle$ en el literal h, y vale que:*

- *$\langle \mathcal{A}_1, h_1 \rangle$ es mejor que $\langle \mathcal{A}, h \rangle$ de acuerdo al criterio de comparación utilizado ($\langle \mathcal{A}_1, h_1 \rangle \succ \langle \mathcal{A}, h \rangle$), denotando que el argumento $\langle \mathcal{A}_1, h_1 \rangle$ es un derrotador propio para el argumento $\langle \mathcal{A}_2, h_2 \rangle$; o*
- *$\langle \mathcal{A}_1, h_1 \rangle$ no es mejor que $\langle \mathcal{A}, h \rangle$, ni $\langle \mathcal{A}, h \rangle$ es mejor que $\langle \mathcal{A}_1, h_1 \rangle$ de acuerdo al criterio de comparación utilizado ($\langle \mathcal{A}_1, h_1 \rangle \not\succ \langle \mathcal{A}, h \rangle$, y $\langle \mathcal{A}, h \rangle \not\succ \langle \mathcal{A}_1, h_1 \rangle$), denotando que el argumento $\langle \mathcal{A}_1, h_1 \rangle$ es un derrotador por bloqueo para el argumento $\langle \mathcal{A}_2, h_2 \rangle$.*

Ejemplo 3.15 *Considere los argumentos del Ejemplo 3.11. Si se utiliza especificidad generalizada como criterio de comparación, el argumento $\langle \mathcal{A}_1, \sim comprarAcciones(acme) \rangle$ es un derrotador propio para el argumento $\langle \mathcal{A}_2, comprarAcciones(acme) \rangle$. Esta situación ses ilustrada debajo, donde la flecha gruesa denota la derrota y la flecha con línea a trazos denota el ataque no exitoso.*



3.3.6. Aceptabilidad en DeLP

Como se presentó en el último elemento de la estructura conceptual introducida en la Sección 3.1.1, la finalidad de cualquier sistema argumentativo es determinar qué argumentos son aceptados. En DeLP, además, los argumentos aceptados determinarán qué literales son inferibles a partir de un programa. Por lo tanto, para decidir si es posbile inferir un literal h a partir de un programa $\mathcal{P}$, es necesario analizar la totalidad de los argumentos, tanto a favor como en contra de h, que pueden construirse a partir de $\mathcal{P}$.

Por ejemplo, si a partir de un programa $\mathcal{P}$ se obtiene un argumento $\langle \mathcal{A}_0, h_0 \rangle$ que no tiene derrotadores, entonces un agente que disponga de $\mathcal{P}$ para razonar, podrá "creer" en el literal h_0. Sin embargo, podría ocurrir que $\langle \mathcal{A}_0, h_0 \rangle$ posea un derrotador $\langle \mathcal{A}_1, h_1 \rangle$, con lo cual existirán razones para invalidar la creencia del agente en h_0, como puede observarse en el Ejemplo 3.15. Sin embargo, también puede ocurrir que el argumento $\langle \mathcal{A}_1, h_1 \rangle$ tenga a su vez un derrotador $\langle \mathcal{A}_2, h_2 \rangle$, reinstaurando así la creencia del agente en h_0.

La secuencia de interacciones arriba mencionada puede ir más lejos, y a su vez puede existir un derrotador $\langle \mathcal{A}_3, h_3 \rangle$ para $\langle \mathcal{A}_2, h_2 \rangle$ que vuelve a invalidar la creencia del agente en h_0, y así siguiendo. Esto da origen a una secuencia de argumentos $[\langle \mathcal{A}_0, h_0 \rangle, \langle \mathcal{A}_1, h_1 \rangle, \langle \mathcal{A}_2, h_2 \rangle \langle \mathcal{A}_3, h_3 \rangle, \ldots]$ conocida como *línea argumentativa*, donde cada elemento de la secuencia es un derrotador de su predecesor.

Definición 3.30 (Línea argumentativa) *Sea $\mathcal{P}$ un programa lógico rebatible y $\langle \mathcal{A}_1, h_1 \rangle$ un argumento obtenido a partir de $\mathcal{P}$. Una* línea argumentativa *para $\langle \mathcal{A}_1, h_1 \rangle$, es una secuencia de argumentos de $\mathcal{P}$ denotada $\Lambda = [\langle \mathcal{A}_1, h_1 \rangle, \langle \mathcal{A}_2, h_2 \rangle, \langle \mathcal{A}_3, h_3 \rangle, \ldots]$, donde para cada elemento de la secuencia $\langle \mathcal{A}_i, h_i \rangle$, el siguiente elemento $\langle \mathcal{A}_{i+1}, h_{i+1} \rangle$ es un derrotador para $\langle \mathcal{A}_i, h_i \rangle$.*

En una línea argumentativa para un argumento $\langle \mathcal{A}_1, h_1 \rangle$ los argumentos en posiciones impares son denominados argumentos de soporte o Pro (es decir, que están a favor de $\langle \mathcal{A}_1, h_1 \rangle$), y los argumentos en posiciones pares son denominados argumentos de interferencia o Con (es decir, que están en contra de $\langle \mathcal{A}_1, h_1 \rangle$).

Para evitar secuencias de argumentos indeseables que pueden representar cadenas de razonamiento falaces, en DeLP se introduce el requerimiento de que una línea argumentativa debe ser aceptable. Es decir, para que una línea argumentativa sea aceptable debe ser finita, no debe tener argumentos repetidos, no debe contener dos derrotas por bloqueo consecutivas, y los conjuntos de argumentos de soporte y de interferencia deben que ser internamente consistentes.

El requisito de que una línea argumentativa sea finita está directamente relacionado con el evitar introducir argumentos repetidos en la línea. Es decir, evitar lo que se denomina como argumentación circular [GS04]. Básicamente, se pueden dar dos situaciones que llevan a argumentación circular. La primer situación, ilustrada en la Figura 3.6(a), ocurre cuando un argumento completo es reintroducido para defenderse a sí mismo. El otro

caso, ilustrado en la Figura 3.6(b), se presenta cuando un subargumento de desacuerdo es reintroducido en la línea para defender al argumento que lo contiene.

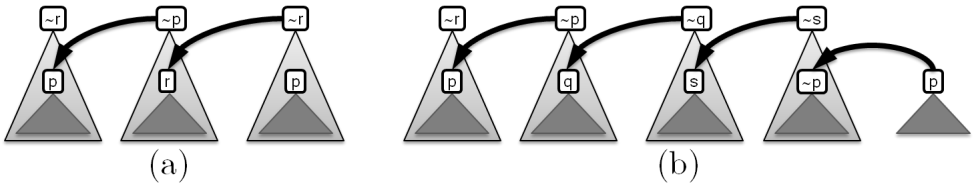


Figura 3.6: Argumentación circular.

Claramente, estas situaciones conllevan a procesos argumentativos falaces y posiblemente infinitos, motivo por el cual deben ser evitadas en una línea argumentativa aceptable. Para evitar esta situación se requerirá que ningún argumento en una línea argumentativa sea un subargumento de un argumento que aparece previamente en la línea.

El hecho de que una línea aceptable no pueda contener dos derrotas por bloqueo consecutivas busca impedir que un bloqueo reinstaure un argumento que ha sido derrotado por bloqueo. Esto captura la intuición de que el sistema no debería sustentar la inferencia de un literal simplemente porque tiene más argumentos a su favor.

Por último, otra situación importante a considerar sobre una línea argumentativa, es que tanto los argumentos de soporte como los de interferencia sean en conjunto consistentes. Esto es con el objetivo de evitar la situación falaz de defender a un argumento con otro argumento con el que es contradictorio.

Definición 3.31 (Línea argumentativa aceptable) *Una línea argumentativa* $\Lambda = [\langle \mathcal{A}_1, h_1 \rangle, \ldots, \langle \mathcal{A}_i, h_i \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle]$ *será aceptable si:*

1. Λ *es una secuencia finita;*
2. *el conjunto* Λ_S*, de argumentos de soporte de* Λ *es consistente, y el conjunto* Λ_I *de argumentos de interferencia de* Λ *también es consistente;*
3. *ningún argumento* $\langle \mathcal{A}_k, h_k \rangle$ *de* Λ *es un subargumento de un argumento* $\langle \mathcal{A}_i, h_i \rangle$ *que aparece previamente en* Λ $(i < k)$*; y*
4. *para todo argumento* $\langle \mathcal{A}_i, h_i \rangle$ *tal que* $\langle \mathcal{A}_i, h_i \rangle$ *es un derrotador por bloqueo de* $\langle \mathcal{A}_{i-1}, h_{i-1} \rangle$*en* Λ*, si existe* $\langle \mathcal{A}_{i+1}, h_{i+1} \rangle$ *en* Λ*, entonces* $\langle \mathcal{A}_{i+1}, h_{i+1} \rangle$ *es un derrotador propio de* $\langle \mathcal{A}_i, h_i \rangle$*.*

Finalmente, observe que siguiendo la Definición 3.31, una línea argumentativa aceptable podría no ser exhaustiva. Es decir, podrían no estar considerándose argumentos que influyen en la evaluación del argumento en discusión. Por lo tanto, en DeLP sólo se considerarán líneas argumentativas aceptables exhaustivas.

Definición 3.32 (Línea argumentativa aceptable exhaustiva) *Sea $\mathcal{P}$ programa DeLP y $\Lambda = [\langle \mathcal{A}_1, h_1\rangle, \ldots, \langle \mathcal{A}_i, h_i\rangle, \ldots, \langle \mathcal{A}_n, h_n\rangle]$ una línea argumentativa aceptable. Se dirá que Λ es una línea argumentativa aceptable exhaustiva si y solo si no existe un argumento $\langle \mathcal{B}, k\rangle$ en $\mathcal{P}$ tal que $[\langle \mathcal{A}_1, h_1\rangle, \ldots, \langle \mathcal{A}_i, h_i\rangle, \ldots, \langle \mathcal{A}_n, h_n\rangle, \langle \mathcal{B}, k\rangle]$ es una línea argumentativa aceptable.*

Árboles de Dialéctica

Claramente, puede existir más de un derrotador para un determinado argumento construido a partir de un programa DeLP. La presencia de múltiples derrotadores para un argumento produce una ramificación de líneas de argumentación, dando origen a un árbol de derrotadores que se denomina *árbol de dialéctica*. En este árbol, cada camino desde la raíz hasta una hoja corresponde a una línea argumentativa aceptable exhaustiva.

Definición 3.33 (Árbol de Dialéctica) *Sea $\mathcal{P}$ un programa DeLP y $\langle \mathcal{A}_0, h_0\rangle$ un argumento de $\mathcal{P}$. Un árbol de dialéctica de $\langle \mathcal{A}_0, h_0\rangle$, notado como $\mathcal{T}\langle \mathcal{A}_0, h_0\rangle$, es un árbol donde los nodos son argumentos de tal $\mathcal{P}$ y los arcos son derrotas entre estos argumentos, tal que:*

- *$\langle \mathcal{A}_0, h_0\rangle$ es la raíz de $\mathcal{T}\langle \mathcal{A}_0, h_0\rangle$;*
- *si $\Lambda = [\langle \mathcal{A}_0, h_0\rangle, \langle \mathcal{A}_1, h_1\rangle, \langle \mathcal{A}_2, h_2\rangle, \ldots, \langle \mathcal{A}_n, h_n\rangle]$ es una rama de $\mathcal{T}\langle \mathcal{A}_0, h_0\rangle$ (camino desde la raíz hasta una hoja), entonces Λ es una línea argumentativa aceptable exhaustiva para $\langle \mathcal{A}_0, h_0\rangle$; y*
- *no existen dos nodos hermanos $\mathcal{N}$ y $\mathcal{N}'$ en $\mathcal{T}\langle \mathcal{A}_0, h_0\rangle$ tales que $\mathcal{N} = \mathcal{N}'$.*

Ejemplo 3.16 *Considere el siguiente programa lógico rebatible:*

$a \prec b$	$\sim d \prec k$	$\sim b \prec c, f$	$\sim f \prec i$
$b \prec c$	e	$f \prec g$	i
c	$\sim b \prec e$	g	$\sim h \prec k$
$\sim b \prec c, d$	$h \prec j$	k	
$d \prec g$	j	$\sim f \prec g, h$	

A partir de este programa es posible construir un argumento $\langle \mathcal{A}, a \rangle$*, con* $\mathcal{A} = \{(a \prec b), (b \prec c)\}$*. Si se utiliza especificidad generalizada como criterio de comparación de argumentos, es posible construir tres derrotadores para* $\langle \mathcal{A}, a \rangle$ *que lo atacan indirectamente en el literal b:*

- $\langle \mathcal{B}_1, \sim b \rangle$*, con* $\mathcal{B}_1 = \{(\sim b \prec c, d), (d \prec g)\}$
- $\langle \mathcal{B}_2, \sim b \rangle$*, con* $\mathcal{B}_2 = \{(\sim b \prec c, f), (f \prec g)\}$
- $\langle \mathcal{B}_3, \sim b \rangle$*, con* $\mathcal{B}_3 = \{(\sim b \prec e)\}$

El argumento $\langle \mathcal{B}_1, \sim b \rangle$*, a su vez, tiene un derrotador* $\langle \mathcal{C}_1, \sim d \rangle$ *con* $\mathcal{C}_1 = \{(\sim d \prec k)\}$*. El argumento* $\langle \mathcal{B}_2, \sim b \rangle$ *tiene dos derrotadores:* $\langle \mathcal{C}_2, \sim f \rangle$ *con* $\mathcal{C}_2 = \{(\sim f \prec i)\}$*, y* $\langle \mathcal{C}_3, \sim f \rangle$ *con* $\mathcal{C}_3 = \{(\sim f \prec g, h), (h \prec j)\}$*. Finalmente, el argumento* $\langle \mathcal{C}_3, \sim f \rangle$ *tiene el derrotador* $\langle \mathcal{D}_1, \sim h \rangle$*, donde* $\mathcal{D}_1 = \{(\sim h \prec k)\}$*. Entonces existen cuatro líneas de argumentación aceptables exhaustivas para* $\langle \mathcal{A}, a \rangle$*:*

- $\Lambda_1 = [\langle \mathcal{A}, a \rangle, \langle \mathcal{B}_1, \sim b \rangle, \langle \mathcal{C}_1, \sim d \rangle]$
- $\Lambda_2 = [\langle \mathcal{A}, a \rangle, \langle \mathcal{B}_2, \sim b \rangle, \langle \mathcal{C}_2, \sim f \rangle]$
- $\Lambda_3 = [\langle \mathcal{A}, a \rangle, \langle \mathcal{B}_2, \sim b \rangle, \langle \mathcal{C}_3, \sim f \rangle, \langle \mathcal{D}_1, \sim h \rangle]$
- $\Lambda_4 = [\langle \mathcal{A}, a \rangle, \langle \mathcal{B}_3, \sim b \rangle]$

Teniendo en cuenta estas líneas argumentativas, se puede construir el árbol de dialéctica $\mathcal{T}\langle \mathcal{A}, a \rangle$ *para* $\langle \mathcal{A}, a \rangle$*, el cual se ilustra en la Figura 3.7 a continuación.*

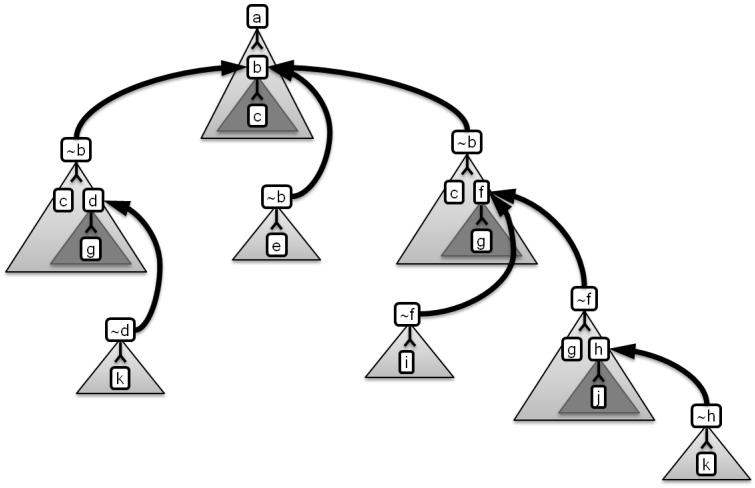


Figura 3.7: Árbol de dialéctica para $\langle \mathcal{A}, a \rangle$.

Como puede observarse en la Figura 3.7, los nodos hoja del árbol de dialéctica corresponden a argumentos no derrotados. En contraste, un nodo interno que tiene como hijo a un nodo hoja corresponderá a un argumento derrotado. Siguiendo con este análisis desde las hojas hacia la raíz, los nodos en un árbol de dialéctica pueden marcarse como **D** *derrotado*, o **U** *no derrotado (del inglés* undefeated*)*. A continuación se especifica cómo se lleva a cabo el procedimiento de marcado para un árbol de dialéctica.

Definición 3.34 (Árbol de dialéctica marcado DeLP) *Sea $\mathcal{T}\langle \mathcal{A}, h \rangle$ un árbol de dialéctica para un argumento $\langle \mathcal{A}, h \rangle$. El árbol de dialéctica marcado para $\mathcal{T}\langle \mathcal{A}, h \rangle$, notado $\mathcal{T}^*\langle \mathcal{A}, h \rangle$, se obtiene marcando cada nodo $\mathcal{N}$ del árbol como* **D** *(derrotado) o* **U** *(no derrotado) de acuerdo al siguiente criterio:*

- *si $\mathcal{N}$ es un nodo hoja, entonces $\mathcal{N}$ es marcado como* **U***;*
- *si $\mathcal{N}$ es un nodo interno y tiene algún hijo marcado como* **U***, entonces $\mathcal{N}$ es marcado como* **D***; o*
- *si $\mathcal{N}$ es un nodo interno y todos sus hijos son marcados como* **D***, entonces $\mathcal{N}$ es marcado como* **U***.*

A continuación, en la Figura 3.8, se presenta el marcado para el árbol de dialéctica de la Figura 3.7.

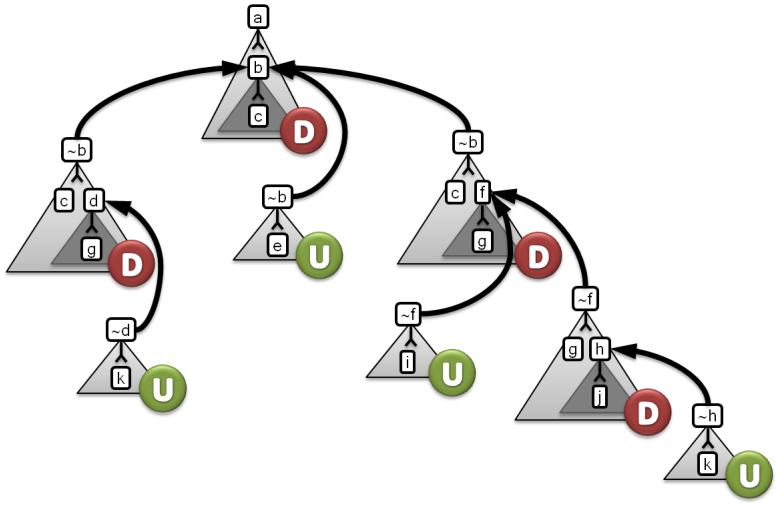


Figura 3.8: Árbol de dialéctica marcado para $\langle \mathcal{A}, a \rangle$.

Un árbol de dialéctica marcado $\mathcal{T}^*\langle \mathcal{A}, h \rangle$ representa el análisis dialéctico en el cual todos los argumentos construibles a partir de un programa $\mathcal{P}$ son considerados a fin de decidir el estado de un argumento $\langle \mathcal{A}, h \rangle$. Por lo tanto, si un argumento Por lo tanto, el estado de un argumento $\langle \mathcal{A}, h \rangle$ será "derrotado" si la raíz de $\mathcal{T}^*\langle \mathcal{A}, h \rangle$ queda marcada con **D**, o "garantizado" si la raíz tiene como marca **U**.

Definición 3.35 (Argumento garantizado) *Sea $\mathcal{P}$ un programa DeLP y $\langle \mathcal{A}, h \rangle$ un argumento construible a partir de $\mathcal{P}$. Se dirá que $\mathcal{P}$ garantiza $\langle \mathcal{A}, h \rangle$ si y solo si el argumento $\langle \mathcal{A}, h \rangle$ se encuentra marcado como* **U** *en $\mathcal{T}^*\langle \mathcal{A}, h \rangle$.*

Claramente, si hay un argumento garantizado, esto implica que hay una garantía para inferir su conclusión. Por lo tanto, en DeLP un literal estará garantizado si es la conclusión de un argumento garantizado. Los literales garantizados son entonces los literales inferidos bajo el procedimiento de prueba dialéctica en DeLP.

Definición 3.36 (Literales garantizados) *Sea $\mathcal{P} = (\Theta, \Delta)$ un programa lógico rebatible y h un literal. El literal h está garantizado si y solo si existe un argumento $\langle \mathcal{A}, h \rangle$ para h y $\langle \mathcal{A}, h \rangle$ está garantizado en $\mathcal{P}$.*

3.4. Resumen

En este capítulo se presentaron los principales conceptos que caracterizan a los sistemas argumentativos. Para esto se mostró la estructura conceptual identificada en [PV02]. Se mostró y se describió cada uno de los cinco elementos de esta estructura: lenguaje de representación, construcción de argumentos, conflictos entre argumentos, derrotas entre argumentos, y determinación de argumentos aceptables.

En particular, se presentaron los dos sistemas argumentativos en los cuales se basan gran parte de los desarrollos de esta tesis. En primer lugar se presentaron los marcos argumentativos abstractos [Dun95], caracterizando y enfatizando su esencia abstracta. Se mostraron las diferentes semánticas de aceptabilidad para estos marcos, y cómo son usadas para determinar los argumentos aceptados para el sistema. Adicionalmente, se presentaron los marcos argumentativos basados en preferencias, una extensión de los marcos argumentativos abstractos en la cual se consideran ataques y preferencias entre argumentos. Estos últimos resultan de importancia para el formalismo abstracto de argumentación basada en tipos de argumento desarrollado en el Capítulo 4 de esta tesis.

Luego se presentó DeLP [GS04], un sistema argumentativo concreto basado en reglas que combina argumentación rebatible y programación en lógica. Se mostró cómo este sistema contempla todos los elementos de la estructura conceptual presentada en este capítulo. Por lo tanto, se mostró cómo DeLP representa cada uno de los elementos de esta estructura: el lenguaje de representación está dado por los programas DeLP, los cuales están compuestos por hechos y reglas rebatibles; los argumentos se construyen a partir de la noción de derivación rebatible; los conflictos entre argumentos se determinan a partir del concepto de desacuerdo entre literales; las derrotas surgen de los conflictos y un criterio de comparación modular entre argumentos; y la aceptabilidad de los argumentos se determina mediante un proceso de prueba dialéctica. Finalmente, DeLP constituye el formalismo fundacional empleado para desarrollar el sistema argumentativo concreto basado en tipos de argumento que se presentará en el Capítulo 5 de esta tesis.

Capítulo 4

Marcos Argumentativos Abstractos Basados en Tipos

En este capítulo se definirá un marco de argumentación abstracta que permite representar argumentos con tipos, y establecer relaciones sobre los tipos de argumento. En el formalismo propuesto cada tipo será identificado con un conjunto de argumentos abstractos y ciertas relaciones entre ellos. Adicionalmente, se presentarán un conjunto de relaciones de tipo para establecer los posibles ataques, preferencias y herencia entre los distintos tipos de argumentos. Los tipos individuales en conjunto con las relaciones entre tipos caracterizarán un Marco Argumentativo de Tipos Múltiples. Este marco abstracto permitirá evaluar la aceptabilidad de sus argumentos tipados considerando la información de los tipos y sus relaciones. Además de su formalización, se ilustrarán su comportamiento y especificación mediante ejemplos.

4.1. Introducción

En los últimos años se han presentado una gran cantidad de propuestas para modelar argumentación basadas en marcos abstractos [Dun95], usando lógica clásica [BH01b], y usando programación en lógica [GS04, Pra10]. En general, este área se centra en el estudio de las estructuras de los argumentos, la interacción entre los argumentos, y cuál será el resultado formal luego del proceso argumentativo. Es por esto que argumentación puede ser utilizada como mecanismo de razonamiento práctico, para resolver problemas

de toma de decisiones, y como mecanismo de negociación y diálogo entre agentes (ver *e.g.*, [BFKIT10, BH09]).

En la literatura del área, especialmente en aquellos trabajos en los que se utilizan los mecanismos argumentativos para resolver algún problema, se consideran distintos tipos de argumentos. Estos argumentos son empleados para representar distintas clases o fuentes de información, o para distinguir argumentos que sirven para distintos propósitos. Por lo tanto, el concepto de "tipo" de argumento es inherente al estudio computacional en argumentación. Por ejemplo, en [AP05] se utilizan argumentos de explicación, argumentos de recompensa y argumentos de amenaza para modelar negociación mediante diálogos; y en las propuestas de razonamiento práctico [ADL08, RGS07, GGS10] se identifican argumentos para creencias, metas y planes con el objetivo de representar los diferentes componentes mentales de un agente. Sin embargo, el concepto de tipo siempre se ha tratado de manera subyacente y fija al problema que se intenta resolver utilizando argumentación.

La idea de tipo de argumento consiste en caracterizar un grupo de argumentos que comparten ciertos atributos (*e.g.*, su estructura, conflictos o preferencias) y, por lo tanto, estos atributos los distinguen de otros grupos de argumentos. En las propuestas arriba mencionadas cada tipo de argumento y sus características son definidos específicamente para el dominio o problema particular para el cual el formalismo es concebido (*e.g.*, sistemas de diálogos, razonamiento práctico, razonamiento legal, etc.). Es decir, el concepto de tipo de argumento siempre ha sido subyacente al formalismo y, por lo tanto, ninguna de las propuestas existentes presenta una caracterización general y formal de la noción de tipo de argumento.

En este capítulo, se buscará dar un tratamiento formal al concepto de tipo de argumento en el contexto de argumentación abstracta. En consecuencia, la contribución de este capítulo será proveer una formalización que permita especificar tipos de argumentos individuales (llamados Marcos Argumentativos de Tipo Simple o MATSs), los cuales permiten definir las características especiales de un grupo de argumentos (*e.g.*, ataques y preferencias) y relaciones entre los marcos argumentativos simples (*e.g.*, herencia, preferencias y ataques), que en conjunto constituirán un Marco Argumentativo de Tipos Múltiples o MATMs. Por ejemplo, considere un escenario en el cual un conjunto de argumentos $\{A_1 \dots A_n\}$ es tal que cada argumento comparte ciertas características y, por lo tanto, es considerado de tipo T_A. Adicionalmente, considere otro conjunto de argumentos $S_B = \{B_1 \dots B_m\}$ de tipo T_B, tal que en el dominio a ser modelado los argumentos de

tipo T_A son preferidos a los argumentos de tipo T_B. Suponga además que existe una especialización de T_B (llamada T_C) dada por un subconjunto S_C de los argumentos de T_B ($S_C \subset S_B$) que hereda ciertas propiedades de T_B, y es tal que los argumentos de tipo T_C son preferidos a los de tipo T_A. Como se mostrará en este capítulo, este tipo de relaciones entre argumentos tipados y entre tipos de argumentos podrán ser representadas utilizando los marcos argumentativos de tipos múltiples.

El capítulo está organizado de la siguiente manera: En la Sección 4.2 se describirá cómo representar tipos individuales utilizando marcos argumentativos con preferencias. Luego en la Sección 4.3 se mostrará cómo establecer conflictos, preferencias y herencia entre tipos. Seguidamente, en la Sección 4.4, se presentarán los Marcos Argumentativos de Tipos Múltiples, así como también el mecanismo para determinar ataques, preferencias y derrota en este contexto. En la Sección 5 se mostrará cómo evaluar los argumentos de estos marcos. Por último, se concluirá el capítulo y se discutirá el trabajo relacionado.

4.2. Marcos Argumentativos de Tipos Simples

Como se vio en el Capítulo 3, un marco argumentativo abstracto clásico está compuesto por un conjunto de argumentos y una relación de derrota entre ellos. Cuando un argumento derrota a otro, estos no pueden ser aceptados en simultáneo. Por lo tanto, debe llevarse acabo un análisis de la relación de derrota para así determinar qué argumentos serán finalmente aceptados.

Los marcos argumentativos de tipos múltiples (o MATMs) propuestos en este capítulo también contendrán argumentos y una relación de derrota. Sin embargo, los argumentos pertenecerán a marcos individuales, los cuales representarán tipos simples. Siguiendo las ideas de [AC02], en estos marcos individuales no se representará una relación de derrota directa sino que se establecerán ataques y preferencias entre los argumentos que los representan. Estos tipos simples podrán relacionarse entre ellos por medio de preferencias entre tipos, conflictos y herencia. Por lo tanto, la relación de derrota se construirá considerando todos estos elementos, y será la que determine qué argumentos son finalmente aceptados.

A continuación se introducirá un mecanismo para representar tipos individuales en un marco argumentativo de tipos múltiples. Estos tipos de argumento simples serán referidos como marcos argumentativos de tipo simple, y estarán basados en los marcos

argumentativos con preferencias de [AC02, KvdT08, MGS08]. Básicamente, un tipo simple estará representado por tres elementos: un conjunto de argumentos y dos relaciones entre estos últimos.

Definición 4.1 (Marco Argumentativo de Tipo Simple) *Un Marco Argumentativo de Tipo Simple (MATS) es una tupla* $T = (AR, \rightsquigarrow, \succeq)$*, donde* AR *es un conjunto finito de argumentos,* $\rightsquigarrow \subseteq AR \times AR$ *es una relación de ataque entre argumentos, y* $\succeq \subseteq AR \times AR$ *es una relación de preferencia entre argumentos.*

Dado un MATS T, se utilizarán las funciones $\mathsf{Args}(T), \mathsf{Atts}(T), \mathsf{Prefs}(T)$ para obtener los argumentos, los ataques y las preferencias de T respectivamente.

En este capítulo se tratará a los argumentos como entidades abstractas, al igual que en [Dun95, AC02], y serán denotados con letras mayúsculas caligráficas. No se hará ninguna referencia a la lógica subyacente necesaria para la representación y construcción de estos argumentos, ya que este formalismo se abstrae de su estructura interna[1].

La relación de ataque entre dos argumentos $(\mathcal{A}, \mathcal{B}) \in \rightsquigarrow$, notado $\mathcal{A} \rightsquigarrow \mathcal{B}$, representará que un argumento $\mathcal{A}$ ataca a un argumento $\mathcal{B}$, es decir, que $\mathcal{A}$ está en conflicto con $\mathcal{B}$. Al igual que lo ocurrido con la estructura de los argumentos, el formalismo desarrollado en este capítulo se abstraerá de cómo se genera esta relación de ataque. Por otra parte, la relación de preferencia $\succeq$ es introducida en estos marcos para evaluar los argumentos, modelando un criterio de preferencia basado en una medida de fuerza. Nuevamente, como en el caso de los argumentos y ataques, esta relación se asumirá provista.

Ejemplo 4.1 *Considere un escenario en el cual un agente está analizando si comprar una casa en el barrio A de la ciudad CT. Para tomar una decisión el agente debe considerar argumentos de dos diferentes tipos:*

T_1) *argumentos de un periódico de CT que tiene noticias e información acerca de la seguridad de la ciudad, y*

T_2) *argumentos de un sitio web de la ciudad CT donde los habitantes publican comentarios sobre los suburbios.*

[1] En el Capítulo 5 se presentará un formalismo en el cual los argumentos tipados sí tienen estructura interna, y utilizará los mecanismos desarrollados en este capítulo para computar la aceptabilidad de los argumentos.

El tipo T_1 (diario local) tiene los siguientes argumentos:

- *Los artículos del diario local están usualmente basados en rumores sin fundamento (argumento $\mathcal{R}$).*
- *El diario local publicó un artículo, llamado N, comentando que el crimen se ha incrementado en el barrio A de la ciudad CT (argumento $\mathcal{C}$).*
- *El artículo N está basado en una entrevista con el jefe de la policía de CT (argumento $\mathcal{P}$).*

Estos argumentos serán caracterizados con el tipo de argumento representado a través del MATS $T_1 = (\{\mathcal{C}, \mathcal{R}, \mathcal{P}\}, \{\mathcal{R} \rightsquigarrow \mathcal{C}, \mathcal{P} \rightsquigarrow \mathcal{R}\}, \{\mathcal{P} \succeq \mathcal{R}\})$. Note que la relación de preferencia en este MATS denota que el agente favorece la fuente de información del articulo N, al conocimiento general de que las noticias están basadas en rumores.

El tipo T_2 (sitio web) tiene los siguientes argumentos:

- *El barrio B es la mejor opción en cuanto a inversión en toda la ciudad CT (argumento $\mathcal{B}$).*
- *Las casas del barrio A están sobre valuadas (argumento $\mathcal{A}$).*
- *El barrio A es considerado un barrio seguro, con lo cual es deseable comprar una casa allí (argumento $\mathcal{S}$).*

Este tipo de argumento será caracterizado utilizando el MATS $T_2 = (\{\mathcal{B}, \mathcal{A}, \mathcal{S}\}, \{\mathcal{B} \rightsquigarrow \mathcal{S}, \mathcal{A} \rightsquigarrow \mathcal{S}\}, \{\mathcal{S} \succeq \mathcal{B}, \mathcal{S} \succeq \mathcal{A}\})$, donde tanto el argumento $\mathcal{A}$ como el argumento $\mathcal{B}$ atacan al argumento $\mathcal{S}$, pero este último es preferido a los otros dos.

Adicionalmente, el agente considera más confiable a aquella información proveniente del diario local que a la originada por el sitio web. Por lo tanto, esto denota que el agente tendrá una preferencia por los argumentos de tipo T_1 por sobre los argumentos de tipo T_2. Sin embargo, dado que esta preferencia involucra distintos tipos, no puede ser capturada en el contexto de un MATS. Como se verá en la Sección 4.3.1 esto se modelará mediante la relación de preferencia entre tipos.

En un marco argumentativo abstracto clásico, como el presentado en [KvdT08], la relación de preferencia y la relación de ataque son utilizadas en conjunto para determinar si un ataque es efectivo o inefectivo. Es decir, para establecer si un ataque se convierte en una *derrota* o no. Por lo tanto, intuitivamente, en el Ejemplo 4.1 se tendría que $\mathcal{P}$ *derrota* a $\mathcal{R}$. Sin embargo, en el contexto de argumentación con múltiples tipos no es suficiente considerar únicamente las relaciones de un tipo simple en forma aislada dado que, como se mostrará en la siguiente sección, estos tipos pueden interactuar entre sí. Por ejemplo, si un argumento de tipo T_1 ataca a un argumento de tipo T_2, como se mencionó en el Ejemplo 4.1, será necesario considerar la relación de preferencia entre tipos para determinar si el ataque es efectivo.

4.3. Relaciones de tipo entre los MATSs

En el contexto de argumentación con múltiples tipos de argumentos es esperable que los tipos individuales interactúen entre sí a través de distintas relaciones. En este capítulo se presentarán relaciones para expresar ataques, preferencia y herencia entre tipos de argumento. Estas relaciones de tipo generarán cierta dependencia entre los tipos individuales, que posteriormente será utilizada para determinar qué argumentos serán finalmente aceptados en el marco argumentativo de múltiples tipos.

4.3.1. Preferencias entre tipos de argumento

Considere la situación presentada en los ejemplos previos. Si el agente tuviese alguna preferencia sobre las fuentes de información sería deseable poder expresar algún tipo de preferencia entre los tipos de argumento. Como se mencionó anteriormente, esta preferencia ayudará a determinar si los ataques entre argumentos de distintos tipos son efectivos o no. Para modelar estas nociones se utilizará la relación de preferencia entre tipos de argumento.

Definición 4.2 (Relación de preferencia entre tipos de argumento) *Sea C_T un conjunto de MATSs. Una relación de preferencia entre tipos de argumento $>_T$ será una relación no reflexiva, no simétrica, definida entre los elementos de C_T.*

Si $(T_1, T_2) \in >_T$ ser dirá que el tipo T_1 es preferido al tipo T_2, y también se notará como $T_1 >_T T_2$.

Ejemplo 4.2 *Considere la situación descripta en el ejemplo 4.1. Allí se menciona que el agente prefiere la información del diario local a la originada por el sitio web, por lo tanto, prefiere los argumentos de tipo T_1 a los de tipo T_2. Esta situación será representada a través de una relación de preferencia entre tipos que establece que $T_1 >_T T_2$.*

4.3.2. Ataques entre argumentos de distinto tipo

Usualmente, cuando se cuenta con diferentes tipos de argumento, es deseable permitir ataques entre los tipos individuales. Por ejemplo, en el contexto de programación de agentes es esperable que un argumento para las metas de mantenimiento (situaciones del mundo que se quieren mantener) ataque a los argumentos de tipo metas de logro que amenacen con violar las condiciones a mantener. Entonces, la noción de ataque entre tipos de argumento permitirá que ciertos argumentos de un tipo ataquen a ciertos argumentos de otro tipo. Para representar este concepto se empleará una función que dados dos MATSs retorna los ataques entre los argumentos de cada uno.

Definición 4.3 (Función de ataque entre argumentos de distintos tipos) *Dados dos MATSs T_1 y T_2, una función de ataque entre argumentos de distintos tipos $\mathsf{TAtt}(T_1, T_2)$ retorna un conjunto C de pares $(\mathcal{A}, \mathcal{B})$ tal que $\forall(\mathcal{A}, \mathcal{B}) \in C$, o bien $\mathcal{A} \in \mathsf{Args}(T_1)$ y $\mathcal{B} \in \mathsf{Args}(T_2)$, o $\mathcal{A} \in \mathsf{Args}(T_2)$ y $\mathcal{B} \in \mathsf{Args}(T_1)$.*

Ejemplo 4.3 *Considere el escenario presentado en el Ejemplo 4.1. En esa situación, los argumentos $\mathcal{C}$ tipo T_1, expresando que hay crimen en el barrio A, y $\mathcal{S}$ de tipo T_2, expresando que barrio A es seguro, se atacan mutuamente. Por lo tanto, esto será representado a través de la función de ataque entre tipos de la siguiente manera: $\mathsf{TAtt}(T_1, T_2) = \{(\mathcal{C}, \mathcal{S}), (\mathcal{S}, \mathcal{C})\}$. Note que, si hubiera algún otro ataque entre T_1 y T_2, estaría especificado en $\mathsf{TAtt}(T_1, T_2)$.*

La principal motivación detrás de esta función es capturar aquellos conflictos que surgen de la integración de diferentes tipos de argumentos, la cual podría tener una connotación semántica con respecto a tales tipos de argumento. De esta manera, por ejemplo,

se permitirá capturar los conflictos entre argumentos para metas de mantenimiento y metas de logro como se mostrará en el Capítulo 6. Por lo tanto, la función T_C provee un mecanismo para expresar explícitamente este tipo de relaciones.

Intuitivamente, introduciendo las relaciones de preferencia entre tipos y ataques entre argumentos de distintos tipos, sería posible concluir que el argumento $\mathcal{C}$ de tipo T_1 *derrota* al argumento $\mathcal{S}$ de tipo T_2. Sin embargo, aún es posible definir formalmente el concepto de *derrota* en el contexto de argumentación multi-tipo, ya que será necesario analizar la relación de herencia entre tipos de argumento.

4.3.3. Herencia entre tipos de argumento

Otra relación interesante que se presentará entre tipos de argumento es la relación de *herencia*. Esta relación es utilizada para determinar un vínculo "es-un" entre tipos de argumento, donde se identificarán tipos base y subtipos. El subtipo (o tipo heredado) contendrá un subconjunto de los argumentos del tipo base y podrá extender o sobreescribir las relaciones individuales del tipo (*i.e.*, ataque y preferencias del MATS).

Definición 4.4 (Relación de herencia entre tipos de argumento) *Sea C_T un conjunto de MATSs, $\multimap\!\triangleright\!\multimap$ es una relación de herencia entre tipos de argumento sobre C_T, si y solo si $\multimap\!\triangleright\!\multimap$ es reflexiva, no simétrica, transitiva y $\forall T_i, T_j \in C_T$ vale que si $T_i \multimap\!\triangleright\!\multimap T_j$ entonces $\mathsf{Args}(T_i) \subseteq \mathsf{Args}(T_j)$.*

Si $(T_1, T_2) \in \multimap\!\triangleright\!\multimap$ se dirá que T_1 hereda de T_2 o que T_1 es un subtipo T_2, y también se notará como $T_1 \multimap\!\triangleright\!\multimap T_2$. De manera similar, se dirá que T_1 es un descendiente de T_2 y que T_2 es un ancestro de T_1. Adicionalmente, se dirá también que T_1 es un tipo más específico[2] o mas especializado que T_2.

Una de las principales motivaciones por las cuales se presenta esta relación, es el permitir la representación de tipos especializados. Es decir, utilizando la relación de herencia y la relación de preferencia entre tipos será posible hacer que argumentos de un tipo especializado S sean preferidos a argumentos de otros tipos, aún cuando argumentos del tipo base de S no lo fueren. A continuación, se mostrará esta situación a través de un ejemplo.

[2]Cabe resaltar que decir que un tipo es más específico que otro no tiene ninguna relación con el concepto de especificidad utilizado para la comparación de argumentos en DeLP.

Ejemplo 4.4 *Considere la situación descripta en los ejemplos 4.1, 4.3, y 4.2, y suponga ahora que el argumento $\mathcal{S}$, expresando que el barrio A es seguro, corresponde a la opinión de un amigo muy confiable que vive allí. El agente prefiere los argumentos de los amigos confiables por sobre otros argumentos. Sin embargo, este argumento aún sigue siendo de tipo T_2, el cual no es preferido a los argumentos de tipo T_1. Por lo tanto, los argumentos del amigo confiable constituirán un nuevo tipo de argumento especializado T_3 que heredará de T_2 y será preferido a T_1. Esto será representado mediante el MATS T_3 $=(\{\mathcal{S}\}, \emptyset, \emptyset)$ y las relaciones $T_3 \multimap\!\triangleright\!\!- T_2$ y $T_3 >_T T_1$.*

Observe que en la Definición 4.4 no se incluye ninguna restricción sobre las relaciones de ataque y preferencia de los MATSs relacionados a través de herencia. Esto se debe a que un subtipo podrá sobreescribir o establecer nuevas relaciones sobre los argumentos heredados. Adicionalmente, note que a partir del Ejemplo 4.4 el argumento $\mathcal{S}$ pertenece a dos MATSs T_2 y T_3. En la siguiente sección se presentará el impacto de esta situación cuando se defina la relación de derrota.

4.4. Marcos Argumentativos de Tipos Múltiples

Una vez introducidos el modelo representacional para tipos de argumento individuales y la forma en que se relacionan estos tipos, será posible definir formalmente los marcos argumentativos de múltiples tipos. Básicamente, estos marcos consistirán de un conjunto de tipos argumentativos simples y las tres relaciones de tipo entre los tipos de ese conjunto.

Definición 4.5 (Marco Argumentativo de Tipos Múltiples) *Un Marco Argumentativo de Tipos Múltiples (MATM) es una tupla $MT = (C_T, \multimap\!\triangleright\!\!-, \mathsf{TAtt}, >_T)$, donde C_T es un conjunto de marcos argumentativos de tipo simple, $\multimap\!\triangleright\!\!-$ es una relación de herencia entre tipos de argumento definida sobre los elementos de C_T, TAtt es una función de ataque entre tipos de argumento definida sobre los elementos de C_T, y $>_T$ es una relación de preferencia de tipos de argumento entre los elementos de C_T.*

Dado un MATM $MT = (C_T, \multimap\!\triangleright\!\!-, \mathsf{TAtt}, >_T)$ la función $\mathsf{MArgs}(MT)$ retornará los argumentos de todos los MATSs en MT, es decir, $\mathsf{MArgs}(MT) = \bigcup \mathsf{Args}(T_i), T_i \in C_T$.

Ejemplo 4.5 *Para modelar la situación completa descripta en los ejemplos anteriores de este capítulo se utilizará un MATM. Este MATM será* $MT_B = (C_T, \multimap\!\triangleright\!\multimap, \mathsf{TAtt}, >_T)$*, donde* $C_T = \{T_1, T_2, T_3\}$ *con* T_1*,* T_2*, y* T_3 *los MATSs presentados en los ejemplos 4.1 y 4.4,* $\multimap\!\triangleright\!\multimap$ *es la relación de herencia presentada en el Ejemplo 4.4,* TAtt *es la función de ataque entre tipos de argumento presentada en el Ejemplo 4.3, y por último la relación de preferencia entre tipos* $>_T$ *es la presentada en el Ejemplo 4.2.*

Dado que los MATSs están relacionados a través de una relación de herencia en el MATM al cual pertenecen, es posible que un argumento pueda pertenecer a más de un tipo. Por lo tanto, se utilizará la función Tipos que dado un argumento de un MATM retornará los tipos a los cuales está asociado.

Definición 4.6 (Función de Tipos) *Sea MT un MATM y* $\mathcal{A}$ *un argumento tal que* $\mathcal{A} \in \mathsf{MArgs}(MT)$*. La función de tipos* $\mathsf{Tipos}(A, MT)$ *retornará un conjunto* A_T *de MATSs tal que para todo* $T_i \in A_T$ *vale que* $\mathcal{A} \in \mathsf{Args}(T_i)$*.*

Como se mencionó en secciones previas de este capítulo, utilizando la información de un MATM correspondiente a sus tipos y sus relaciones será posible determinar la aceptabilidad de sus argumentos. Para lograr esto será necesario determinar una relación de derrota entre los argumentos del marco completo. Recuerde que para establecer la existencia de una *derrota* será necesario encontrar qué ataques son efectivos teniendo en cuenta las preferencias. Por lo tanto, para decidir si un argumento $\mathcal{A}$ de un tipo arbitrario *derrota* a un argumento $\mathcal{B}$ de posiblemente otro tipo, debería haber un ataque de $\mathcal{A}$ a $\mathcal{B}$ y, además, $\mathcal{A}$ debería ser al menos tan preferido como $\mathcal{B}$. De esta manera, será necesario definir cómo se establecen las relaciones de ataque y preferencia globales a todos los argumentos que forman parte de un MATM.

4.4.1. Ataques en un MATM

Básicamente, hay dos posibilidades de que exista un ataque entre dos argumentos de un MATM: si se atacan en alguno de los MATSs a los que pertenecen, o si hay un ataque entre ellos establecido por la función de ataque entre tipos.

Definición 4.7 (MATM - Ataques) *Sea* $MT = (C_T, \multimap\!\triangleright\!\multimap, \mathsf{TAtt}, >_T)$ *un MATM tal que* $\mathcal{A}, \mathcal{B} \in \mathsf{MArgs}(MT)$*. Se dirá que* $\mathcal{A}$ *ataca a* $\mathcal{B}$ *en MT, notado* $\mathcal{A} \rightharpoonup \mathcal{B}$*, si y solo*

si existe T_i *en* C_T *tal que* $(\mathcal{A}, \mathcal{B}) \in \mathsf{Atts}(T_i)$*, o* $(\mathcal{A}, \mathcal{B}) \in \mathsf{TAtt}(T_A, T_B)$ *para cualquier* $T_A \in \mathsf{Tipos}(A, MT)$ *y* $T_B \in \mathsf{Tipos}(B, MT)$*.*

Por ejemplo, considere el MATM MT_B del Ejemplo 4.5, representando el escenario del agente que tiene que decidir si comprar una casa en el barrio A. En la Figura 4.1 se presenta un gráfico que ilustra los ataques en MT_B, donde los rectángulos representan los MATSs de MT_B, los círculos corresponden a los argumentos, y las flechas delgadas son los ataques obtenidos.

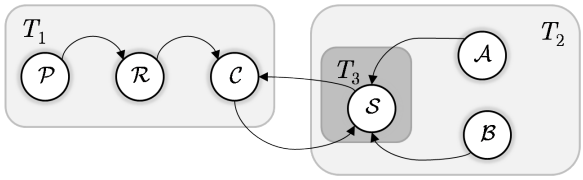


Figura 4.1: Ataques en el MATM MT_B.

4.4.2. Preferencias en un MATM

La relación de preferencia en el contexto de un MATM requiere un análisis más profundo que el realizado para la relación de ataque. Esto se debe a que esta relación está principalmente vinculada con la dependencia entre los tipos generada por la relación de herencia.

Intuitivamente, las preferencias establecidas para los tipos más especializados deberían *sobrescribir* las preferencias establecidas para los tipos menos especializados. Esto es, un argumento debería ser preferido a otro argumento si hay una preferencia establecida en (entre) algún tipo, y esta preferencia no es sobrescrita en algún tipo más especializado. Por lo tanto, será necesario analizar la relación de herencia para determinar si un argumento es preferido a otro en el contexto de un MATM.

La relación de herencia en los MATM conduce a una red de herencia [HTT90, TK93] entre sus tipos. Este tipo de redes denotan un grafo dirigido acíclico, donde los nodos son tipos y los arcos dirigidos son las relaciones de herencia entre ellos. Para mostrar que la relación de herencia en los MATMs conduce a una red de herencia, se demostrará que la relación de herencia es no circular.

Proposición 4.1 *Sea* $MT = (C_T, \multimap\triangleright\multimap, \mathsf{TAtt}, >_T)$ *un MATM, y* $T_i, T_j \in C_T$ *tales que* $T_i \multimap\triangleright\multimap T_j$*. Entonces,* $\nexists T_k \in C_T$*,* $T_k \neq T_i$*,* $T_k \neq T_j$*, tal que* $T_j \multimap\triangleright\multimap T_k$ *y* $T_k \multimap\triangleright\multimap T_i$*.*

Prueba : Asuma que $\exists T_i, T_j, T_k \in C_T$ *tales que* $T_i \multimap\triangleright\multimap T_j$*,* $T_k \multimap\triangleright\multimap T_i$ *y* $T_j \multimap\triangleright\multimap T_k$*. Entonces, por transitividad* $T_i \multimap\triangleright\multimap T_i$*, lo cual es una contradicción ya que la relación de herencia debe ser no reflexiva.* □

Por lo tanto, dadas las características de la relación $\multimap\triangleright\multimap$, será posible construir un grafo dirigido acíclico en base a la herencia entre tipos de argumento en un MATM. Este grafo será llamado $\multimap\triangleright\multimap$ grafo, y será tal que sus nodos serán los MATSs y sus arcos estarán determinados por la relación $\multimap\triangleright\multimap$ del MATM asociado. Estos grafos serán utilizados para explicar cómo se obtienen las preferencias para los argumentos en el contexto de los MATMs.

Para determinar las preferencias de los argumentos en los MATMs es necesario considerar dos casos: cuando las preferencias están establecidas dentro de un MATS, o cuando las preferencias entre los argumentos están establecidas a través de la relación de preferencia entre tipos. A continuación se analizarán ambas situaciones y se proveerán definiciones para cada uno de estos casos.

Preferencia Interna

Como se presentó en la Definición 4.1, un MATS puede definir preferencias entre sus argumentos. Sin embargo, la relación de herencia en un MATM permite a un subtipo de un MATS redefinir las preferencias entre los argumentos establecidas por sus ancestros. Por ejemplo, considere el MATM de la Figura 4.2, donde $T_c \multimap\triangleright\multimap T_b \multimap\triangleright\multimap T_a$ y los argumentos $\mathcal{X}$ e $\mathcal{Y}$ pertenecen a los tres tipos de argumento. A continuación se analizará la relación de preferencia entre $\mathcal{X}$ e $\mathcal{Y}$ en el contexto de este MATM. El MATS T_a establece que $\mathcal{X} \succeq \mathcal{Y}$, pero esta preferencia es sobrescrita por la preferencia establecida en T_b, dado que T_b es un tipo de argumento más especializado que T_a. Por otra parte, si bien T_c es un tipo de argumento más especializado que T_b, T_c no establece ninguna preferencia entre $\mathcal{X}$ e $\mathcal{Y}$. Por lo tanto, en este MATM el argumento $\mathcal{Y}$ debería ser preferido al argumento $\mathcal{X}$.

La situación mostrada en la Figura 4.2 ilustra la noción de *preferencia interna*, donde la preferencia entre dos argumentos es establecida dentro de un MATS T, pero puede ser sobrescrita por MATSs más especializados que hereden de T. Es decir, un argumento $\mathcal{N}$

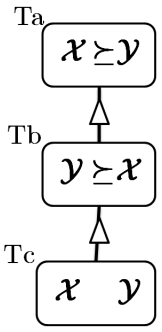


Figura 4.2: Preferencias redefinidas y herencia.

será internamente preferido a otro argumento $\mathcal{M}$, si para cualquier MATS que establece que $\mathcal{M}$ es preferido a $\mathcal{N}$, existe un MATS más especializado que establece lo contrario. Formalmente:

Definición 4.8 (Preferencia interna) *Sea* $MT = (C_T, \multimap\!\triangleright\!\multimap, \mathsf{TAtt}, >_T)$ *un MATM y* $\mathcal{A}$,$\mathcal{B}$ *dos argumentos de* $\mathsf{MArgs}(MT)$. *El argumento* $\mathcal{A}$ *es internamente preferido al argumento* $\mathcal{B}$ *en MT, notado* $\mathcal{A} >_I \mathcal{B}$, *si y solo si:*

- *existe un MATS* T_i *en* C_T *tal que* $\mathcal{A} \succeq \mathcal{B}$ *está en* T_i, *y*
- *para todo MATS* T_j *de* C_T *tal que* $\mathcal{B} \succeq \mathcal{A}$ *está en* T_j, *existe un* T_k *en* C_T *tal que* $T_k \multimap\!\triangleright\!\multimap T_j$ *y* $\mathcal{A} \succeq \mathcal{B}$ *está en* T_k.

La Definición 4.8 sigue el espíritu de la agregación de preferencias en el contexto de argumentación presentado en [APP00]. Esto se debe a que la preferencia interna captura la intuición de que en los marcos argumentativos puede haber varios órdenes de preferencia entre los argumentos, y estos órdenes son ordenados o elegidos por otra relación. En este caso, los múltiples ordenes están dados por los MATSs y el orden o selección sobre estos órdenes está determinado por la relación de herencia.

Ejemplo 4.6 *Considere los MATMs ilustrados en las figuras 4.3 (a), (b) y (c). A continuación se analizará la relación de preferencia interna entre los argumentos de estos MATMs.*

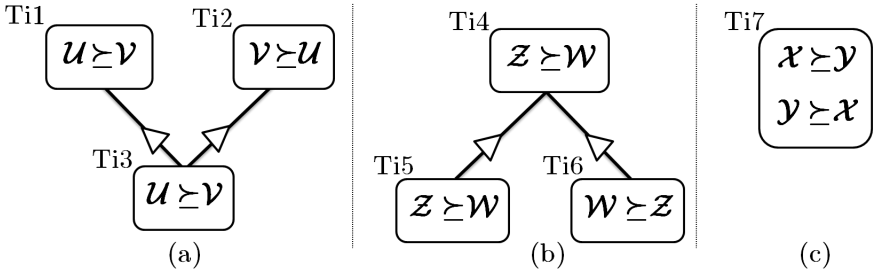


Figura 4.3: Preferencias internas en los MATMs.

Note que en el MATM de la Figura 4.3(a) $\mathcal{U}$ *es internamente preferido a* $\mathcal{V}$*, porque existe un tipo* $Ti2$ *que establece* $\mathcal{V} \succeq \mathcal{U}$*, pero* $Ti3$ *que hereda de* $Ti2$ *establece* $\mathcal{U} \succeq \mathcal{V}$*. En el MATM de la Figura 4.3(b) se da que* $\mathcal{Z} \not\succ_I \mathcal{W}$ *y* $\mathcal{W} \not\succ_I \mathcal{Z}$*, ya que al analizar si* $\mathcal{Z}$ *es internamente preferido a* $\mathcal{W}$*, existe el tipo* $Ti6$ *que establece* $\mathcal{W} \succeq \mathcal{Z}$ *y no existe ningún tipo que herede de Ti6 y diga lo contrario; por otra parte, al analizar si* $\mathcal{Z}$ *es internamente preferido a* $\mathcal{W}$ *ocurre una situación análoga con el tipo* $Ti5$*. Similar es el caso de la Figura 4.3(c), donde se da que* $\mathcal{X} \not\succ_I \mathcal{Y}$ *e* $\mathcal{Y} \not\succ_I \mathcal{X}$*, ya que en* $Ti7$ *se establece que* $\mathcal{X}$ *e* $\mathcal{Y}$ *son igual de preferidos y no hay ningún tipo que herede de* $Ti7$ *y establezca una preferencia por uno o por otro.*

Una propiedad que cumple la relación de preferencia interna es que, si $\mathcal{A}$ es internamente preferido a $\mathcal{B}$, entonces existirá un conjunto de MATSs más especializados que establecen que $\mathcal{A}$ es preferido a $\mathcal{B}$, y no habrá ningún tipo que herede de ellos y establezca lo mismo o lo contrario. Esta propiedad se encuentra formalizada en la siguiente proposición.

Proposición 4.2 *Sea* $MT = (C_T, \multimap, \mathsf{TAtt}, >_T)$ *un MATM y* $\mathcal{A}$,$\mathcal{B}$ *dos argumentos de* $\mathsf{MArgs}(MT)$*. Si* $\mathcal{A} >_I \mathcal{B}$ *entonces existe un* T_i *en* C_T *que establece* $\mathcal{A} \succeq \mathcal{B}$ *y no existe un* $T_j \in C_T$ *tal que* $T_j \multimap T_i$ *y* T_j *establece* $\mathcal{B} \succeq \mathcal{A}$ *o* $\mathcal{A} \succeq \mathcal{B}$*.*

Prueba : *Por Definición 4.8 de preferencia interna, y dado que la Proposición 4.1 enuncia que la relación de herencia es no circular, entonces existe algún* T_i *en* C_T *que establece* $\mathcal{A} \succeq \mathcal{B}$*, y no existe ningún* T_k *en* C_T *tal que* $T_k \multimap T_i$ *y* T_k *establezca lo mismo que* T_i*. Luego para* T_i *no existirá un* T_j *en* C_T *tal que* $T_j \multimap T_i$ *y que establezca* $\mathcal{B} \succeq \mathcal{A}$*, ya que sino* $\mathcal{A}$ *no sería internamente preferido a* $\mathcal{B}$*.* □

El conjunto de todos los MATSs que cumplen esta propiedad para $\mathcal{A} >_I \mathcal{B}$ será llamado $IT(\mathcal{A}, \mathcal{B})$. La importancia de los tipos de este conjunto radica en que son los que finalmente rectifican que $\mathcal{A}$ es internamente preferido a $\mathcal{B}$, ya que no habrá otro tipo más especializado que diga lo contrario. Este conjunto será útil para definir la preferencia global entre dos argumentos de un MATM, como se verá más adelante.

Definición 4.9 (Conjunto IT) *Sea* $MT = (C_T, -\triangleright-, \mathsf{TAtt}, >_T)$ *un MATM y* $\mathcal{A}$,$\mathcal{B}$ *dos argumentos de* $\mathsf{MArgs}(MT)$ *tales que* $\mathcal{A} >_I \mathcal{B}$ *en MT. El conjunto* $IT(\mathcal{A}, \mathcal{B})$ *para* $\mathcal{A}$, $\mathcal{B}$, *es tal que todo* $T_i \in IT(\mathcal{A}, \mathcal{B})$ *es un tipo que verifica la Proposición 4.2.*

Preferencia Externa

Otra situación que debe considerarse al momento de definir la relación de preferencia en un MATM ocurre cuando hay herencia y preferencias establecidas entre tipos de argumentos. Por ejemplo, considere la situación ilustrada en la Figura 4.4. A continuación se analizará la preferencia entre $\mathcal{W}$ y $\mathcal{Z}$ en el contexto del MATM. Se cuenta con tres MATSs T_d, T_e y T_f tales que $T_e -\triangleright- T_d$, y la relación de preferencia entre tipos de argumento es la ilustrada en la figura. Aún cuando T_f es preferido a T_d, el argumento $\mathcal{W}$ también pertenece a un tipo más especializado (T_e), el cual es preferido ante T_f. Por lo tanto, en este caso $\mathcal{W}$ debería ser preferido a $\mathcal{Z}$. Note que en la situación presentada en el Ejemplo 4.4 ocurre algo similar.

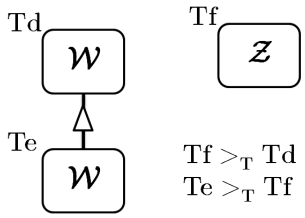


Figura 4.4: Preferencias entre tipos de argumento y herencia.

Este ejemplo ilustra la otra forma de preferencia en los MATM, la *preferencia externa*. En este caso, la preferencia determinada por una relación de preferencia entre tipos puede ser sobrescrita por una preferencia entre tipos involucrando a un tipo más especializado. Luego, se dirá que un argumento $\mathcal{N}$ será externamente preferido a otro argumento $\mathcal{M}$, si para cada tipo T_M asociado a $\mathcal{M}$ tal que es preferido a un tipo asociado a $\mathcal{N}$, existe un

tipo más especializado de $\mathcal{N}$ tal que es preferido a T_M o alguno de sus tipos descendientes (que también sea tipo de $\mathcal{M}$). Formalmente:

Definición 4.10 (Preferencia externa) *Sea* $MT = (C_T, \multimap\!\triangleright\!\!-, \mathsf{TAtt}, >_T)$ *un MATM y* $\mathcal{A}$,$\mathcal{B}$ *dos argumentos de* $\mathsf{MArgs}(MT)$ *tales que* $\mathsf{Tipos}(\mathcal{A}, MT) = Ts_A$ *y* $\mathsf{Tipos}(\mathcal{B}, MT) = Ts_B$. *El argumento* $\mathcal{A}$ *es externamente preferido al argumento* $\mathcal{B}$, *notado* $\mathcal{A} >_E \mathcal{B}$, *si y solo si:*

- *existen un tipo* $T_1 \in Ts_A$ *y un tipo* $T_2 \in Ts_B$ *tales que* $T_1 >_T T_2$, *y*
- *para todo* $T_j \in Ts_B$, $T_i \in Ts_A$ *tales que* $T_j >_T T_i$, *existen* $T_k \in Ts_A$ *y* $T_l \in Ts_B$ *tales que* $T_k \multimap\!\triangleright\!\!- T_i$, $T_l \multimap\!\triangleright\!\!- T_j$ *y* $T_k >_T T_l$.

Ejemplo 4.7 *Considere los MATMs ilustrados en las figuras 4.5 (a), (b) y (c). A continuación se analizará la relación de preferencia externa entre los argumentos de estos MATMs.*

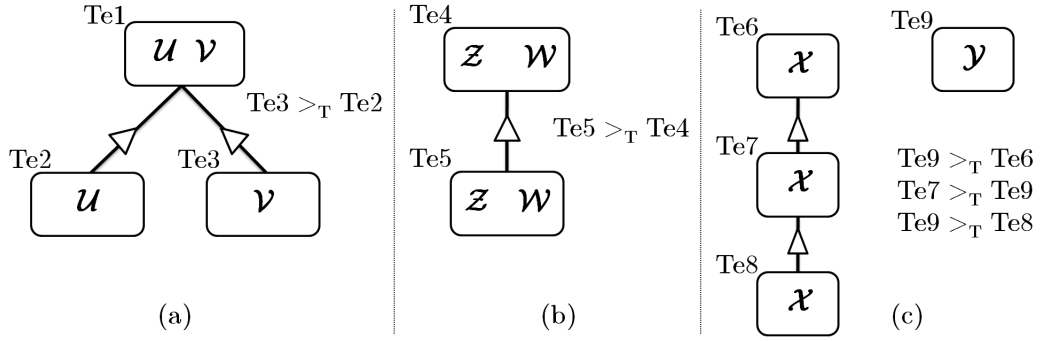
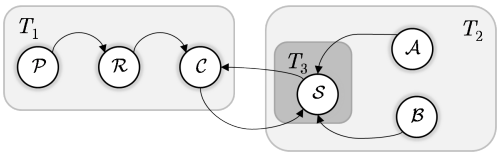


Figura 4.5: Preferencias externas en los MATMs.

Note que en el MATM de la Figura 4.3(a) $\mathcal{V}$ *es externamente preferido a* $\mathcal{U}$, *porque el tipo* $Te3$ *al cual* $\mathcal{V}$ *pertenece es preferido al tipo* $Te2$ *de* $\mathcal{U}$, *y no existe ningún tipo que herede de ellos y establezca lo contrario. En el MATM de la Figura 4.3(b) se da que* $\mathcal{Z} \not>_E \mathcal{W}$ *y* $\mathcal{W} \not>_E \mathcal{Z}$ *ya que, por ejemplo, al analizar si* $\mathcal{Z}$ *es externamente preferido a* $\mathcal{W}$, *existe el tipo* $Te5$ *de* $\mathcal{Z}$ *que es preferido al tipo* $Te4$ *de* $\mathcal{W}$, *pero* $Te5$ *es también un tipo de* $\mathcal{W}$ *que es preferido al tipo* $Te4$ *de* $\mathcal{Z}$ *y no hay ningún tipo que herede de ellos y establezca alguna preferencia. En el caso de la Figura 4.3(c)* $\mathcal{Y} >_E \mathcal{X}$, *ya que si bien existe el tipo* $Te7$ *de* $\mathcal{X}$ *que es preferido al tipo* $Te9$ *de* $\mathcal{Y}$, *el tipo* $Te9$ *es preferido a* $Te8$ *que es un descendiente de* $Te7$, *y no existe otro tipo descendiente que establezca lo contrario.*

Note que entre los argumentos $\mathcal{S}$ y $\mathcal{C}$ del Ejemplo 4.5 se presenta una situación clara de preferencias externas, teniendo en cuenta que $T_1 >_T T_2$ y $T_3 >_T T_1$.



Recuerde que como se muestra arriba el argumento $\mathcal{S}$ pertenece a los tipos T_1 y T_3, mientras que el argumento $\mathcal{C}$ pertenece al tipo T_2. Si bien, el tipo T_2 es preferido al tipo T_1, el tipo T_3 (que hereda del tipo T_1) es preferido al tipo T_2. Por lo tanto, dado que $\mathcal{C}$ no pertenece a ningún otro tipo tal que sea preferido a un tipo de $\mathcal{S}$, se tiene que $\mathcal{S} >_E \mathcal{C}$.

De manera similar a la preferencia interna, la preferencia externa puede identificar un conjunto de MATSs más especializados que determinan finalmente la preferencia externa entre dos argumentos. Es decir, si $\mathcal{A}$ es externamente preferido a $\mathcal{B}$ existirá al menos un MATS de $\mathcal{A}$ que es preferido a uno de $\mathcal{B}$, ninguún MATS de $\mathcal{B}$ será preferido a ese MATS de $\mathcal{A}$, y no habrá otro MATS más especializado de $\mathcal{A}$ que cumpla esas condiciones. Esto se encuentra formalizado en la siguiente proposición.

Proposición 4.3 *Sea* $MT = (C_T, -\triangleright-, \mathsf{TAtt}, >_T)$ *un MATM y* $\mathcal{A}$,$\mathcal{B}$ *dos argumentos de* $\mathsf{MArgs}(MT)$ *tales que* $\mathsf{Tipos}(\mathcal{A}, MT) = Ts_A$ *y* $\mathsf{Tipos}(\mathcal{B}, MT) = Ts_B$. *Si* $\mathcal{A} >_E \mathcal{B}$ *entonces existe un* T_i *en* Ts_A *tal que* $T_i >_T T_j$ *para algún* T_j *en* Ts_B, *y para todo* T_k *en* Ts_A *tal que* $T_k -\triangleright- T_i$ *no existe ningún* T_l *en* Ts_B *tal que* $T_l >_T T_k$ *o* $T_k >_T T_l$.

Prueba : *Por Definición 4.10 de preferencia externa, y dado que la Proposición 4.1 establece que la relación de herencia es no circular, entonces existe algún* T_i *en* Ts_A *tal que* $T_i >_T T_j$ *para algún* T_j *en* Ts_B *y, además, no hay ningún* T_k *en* Ts_A *tal que* $T_l >_T T_m$ *con* T_m *en* Ts_B. *Luego, no existirá ningún* T_l *en* Ts_B *tal que* $T_l >_T T_k$ *con* $T_k -\triangleright- T_i$ *y* T_k *en* Ts_A, *ya que sino* $\mathcal{A}$ *no sería externamente preferido a* $\mathcal{B}$. □

El conjunto de todos los MATSs que cumplen esta propiedad para $\mathcal{A} >_E \mathcal{B}$ será llamado $ET(\mathcal{A}, \mathcal{B})$. La importancia de los tipos de este conjunto es que son los que finalmente ratifican que $\mathcal{A}$ es externamente preferido a $\mathcal{B}$, ya que no habrá otro tipo más especializado que diga lo mismo o lo contrario. Este conjunto será útil para definir la preferencia global entre dos argumentos de un MATM, como se verá más adelante.

Definición 4.11 (Conjunto ET) *Sea* $MT = (C_T, \multimap, \mathsf{TAtt}, >_T)$ *un MATM y* $\mathcal{A}$,$\mathcal{B}$ *dos argumentos de* $\mathsf{MArgs}(MT)$ *tales que* $\mathcal{A} >_E \mathcal{B}$ *en MT. El conjunto* $ET(\mathcal{A}, \mathcal{B})$ *para* $\mathcal{A}$, $\mathcal{B}$, *es tal que todo* $T_i \in ET(\mathcal{A}, \mathcal{B})$ *es un tipo que verifica la Proposición 4.3.*

Preferencia Global en MATM

Las nociones formalizadas en las definiciones 4.8 y 4.10 siguen el principio de anticipación (*preemption* en inglés) presentado en la literatura de redes de herencia [HTT90]. La intuición detrás de este concepto radica en que las relaciones basadas en tipos más especializados reemplazan a las relaciones basadas en tipos menos especializados. Ambas definiciones siguen este principio al indicar que la preferencia entre dos argumentos se establece en un tipo, si y solo si no es sobrescrita por una preferencia que indique lo contrario en tipos más especializados.

Para poder definir la relación de preferencia global de los argumentos de un MATM será necesario relacionar las preferencias interna y externa. En la siguiente tabla se muestra cómo las distintas combinaciones de preferencias externas e internas influyen al momento de determinar la preferencia global de un argumento $\mathcal{A}$ por sobre otro argumento $\mathcal{B}$.

Preferencia Externa	Preferencia Interna	Preferencia Global
$\mathcal{A} >_E \mathcal{B}$	$\mathcal{A} >_I \mathcal{B}$	$\mathcal{A} >_{MT} \mathcal{B}$
$\mathcal{A} \not>_E \mathcal{B}$	$\mathcal{A} \not>_I \mathcal{B}$	$\mathcal{A} \not>_{MT} \mathcal{B}$
$\mathcal{A} >_E \mathcal{B}$	$\mathcal{B} \not>_I \mathcal{A}$	$\mathcal{A} >_{MT} \mathcal{B}$
$\mathcal{B} \not>_E \mathcal{A}$	$\mathcal{A} >_I \mathcal{B}$	$\mathcal{A} >_{MT} \mathcal{B}$
$\mathcal{A} >_E \mathcal{B}$	$\mathcal{B} >_I \mathcal{A}$	Analizar los tipos de $ET(\mathcal{A}, \mathcal{B})$ y $IT(\mathcal{B}, \mathcal{A})$
$\mathcal{B} >_E \mathcal{A}$	$\mathcal{A} >_I \mathcal{B}$	Analizar los tipos de $ET(\mathcal{B}, \mathcal{A})$ y $IT(\mathcal{A}, \mathcal{B})$

Si un argumento $\mathcal{A}$ es externamente e internamente preferido a otro argumento $\mathcal{B}$, claramente $\mathcal{A}$ será preferido a $\mathcal{B}$ en el MATM. En el caso completamente opuesto, donde $\mathcal{A}$ no es ni internamente ni externamente preferido a $\mathcal{B}$, $\mathcal{A}$ no será preferido a $\mathcal{B}$ en el MATM. Por otra parte, cuando $\mathcal{A}$ es externamente preferido a $\mathcal{B}$ y no ocurre que $\mathcal{B}$ es internamente preferido a $\mathcal{A}$, entonces $\mathcal{A}$ será globalmente preferido a $\mathcal{B}$. Cuando $\mathcal{A}$ es internamente preferido a $\mathcal{B}$ y $\mathcal{B}$ no es externamente preferido a $\mathcal{A}$ la situación es análoga. Entonces, cuando solo uno o los dos tipos de preferencias establecen que $\mathcal{A}$ es preferido sobre $\mathcal{B}$, claramente $\mathcal{A}$ será globalmente preferido a $\mathcal{B}$.

Note que el análisis en el caso en que las preferencias se contraponen es más complejo. Considere el MATM de la Figura 4.6. En este escenario puede observarse que los argumentos $\mathcal{U}$ y $\mathcal{V}$ pertenecen al MATS $T1$, el cual establece $\mathcal{U} \succeq \mathcal{V}$. Además, $\mathcal{U}$ también pertenece al MATS $T2$ que hereda de $T1$, y $\mathcal{V}$ pertenece al MATS $T3$ que también hereda de $T1$. En particular, estos dos tipos son tales que $T3 >_T T2$.

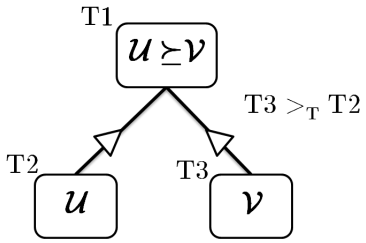


Figura 4.6: Preferencias internas y externas contrapuestas.

Por lo tanto, observe que en esta situación, por una parte, $\mathcal{U}$ será internamente preferido a $\mathcal{V}$ ya que se establece una preferencia $\mathcal{U} \succeq \mathcal{V}$ en $T1$ y no hay ningún otro MATSs que lo contradiga. Por otra parte, note que $\mathcal{V}$ será externamente preferido a $\mathcal{U}$, ya que existe un tipo de $\mathcal{V}$ que es preferido a un tipo de $\mathcal{U}$ (T3 $>_T$ T2), pero no viceversa. Se tiene entonces que, a pesar de que $\mathcal{U}$ sea internamente preferido a $\mathcal{V}$, la preferencia externa de $\mathcal{V}$ sobre $\mathcal{U}$ se establece en un par de tipos más especializados que el tipo que establece la preferencia interna. Por lo tanto, en este contexto es esperable que $\mathcal{V}$ sea globalmente preferible a $\mathcal{U}$.

La situación ilustrada en el párrafo anterior se generaliza de la siguiente manera. Básicamente, si un argumento $\mathcal{A}$ es externamente preferido a otro argumento $\mathcal{B}$ pero $\mathcal{B}$ es internamente preferido a $\mathcal{A}$, entonces $\mathcal{A}$ será globalmente preferido a $\mathcal{B}$ sólo si para todo tipo en $IT(\mathcal{B}, \mathcal{A})$ existen un subtipo $ET(\mathcal{A}, \mathcal{B})$ tal que establece la preferencia externa.

El caso inverso a la situación descripta en el párrafo anterior resulta análogo. Es decir, el caso en que $\mathcal{A}$ sea internamente preferido $\mathcal{B}$ y $\mathcal{B}$ sea externamente preferido a $\mathcal{A}$, pero los tipos de $IT(\mathcal{A}, \mathcal{B})$ son subtipos de los de $ET(\mathcal{B}, \mathcal{A})$. En este caso, $\mathcal{A}$ debería ser globalmente preferido $\mathcal{B}$. Teniendo en cuenta esta situación y las descriptas en los párrafos anteriores, a continuación se presentará la definición de preferencia global en el contexto de un MATM.

Definición 4.12 (Preferencia global) *Sea* $MT = (C_T, \multimap, \mathsf{TAtt}, >_T)$ *un MATM y* $\mathcal{A}$,$\mathcal{B}$ *dos argumentos de* $\mathsf{MArgs}(MT)$. *El argumento* $\mathcal{A}$ *es globalmente preferido al argumento* $\mathcal{B}$ *en MT, notado* $\mathcal{A} >_{MT} \mathcal{B}$, *si y solo si:*

- $\mathcal{A} >_E \mathcal{B}$ *y* $\mathcal{B} \not> _I \mathcal{A}$;
- $\mathcal{A} >_I \mathcal{B}$ *y* $\mathcal{B} \not> _E \mathcal{A}$;
- $\mathcal{A} >_E \mathcal{B}$, $\mathcal{B} >_I \mathcal{A}$, *y para todo tipo* T_i *en* $IT(\mathcal{B}, \mathcal{A})$ *existe un tipo* T_j *en* $ET(\mathcal{A}, \mathcal{B})$ *tal que* $T_j \multimap T_i$; *o*
- $\mathcal{A} >_I \mathcal{B}$, $\mathcal{B} >_E \mathcal{A}$, *y para todo tipo* T_i *en* $ET(\mathcal{B}, \mathcal{A})$ *existe un tipo* T_j *en* $IT(\mathcal{A}, \mathcal{B})$ *que* $T_j \multimap T_i$.

Por ejemplo, note que en el MATM del Ejemplo 4.5, se darán las siguientes preferencias globales: $\mathcal{S} >_{MT} \mathcal{A}$, $\mathcal{S} >_{MT} \mathcal{B}$, $\mathcal{P} >_{MT} \mathcal{R}$, $\mathcal{R} >_{MT} \mathcal{C}$, y $\mathcal{S} >_{MT} \mathcal{C}$. Además, observe que la relación de preferencia global no establece ninguna predilección entre preferencias externas e internas.

4.4.3. Derrotas en un MATM

Habiendo definido las nociones de ataque y preferencia para los argumentos de los diferentes tipos dentro de un MATM, es posible definir la relación de derrota entre argumentos. En este capítulo se seguirá la noción de derrota similar a la presentada en [KvdT08], en la cual un argumento derrota a otro si lo ataca y es preferido ante él.

Definición 4.13 (Derrota) *Sea MT un MATM. Un argumento* $\mathcal{A}$ *derrota a otro argumento* $\mathcal{B}$ *(*$\mathcal{A}$ *es un derrotador de* $\mathcal{B}$*), notado* $\mathcal{A} \rightarrow \mathcal{B}$, *si y solo si* $\mathcal{A} \rightharpoonup \mathcal{B}$, *y* $\mathcal{A} >_{MT} \mathcal{B}$ *o* $\mathcal{B} \not> _{MT} \mathcal{A}$.

Ejemplo 4.8 *Considere el MATM presentado en el Ejemplo 4.5 y los ataques entre sus argumentos ilustrados en la Figura 4.1. En la Figura 4.7 se presenta un grafo que ilustra los ataques que resultan en derrota en el Ejemplo 4.5. Las flechas gruesas representan derrotas y las flechas delgadas representan los ataques que no terminaron siendo derrotas.*

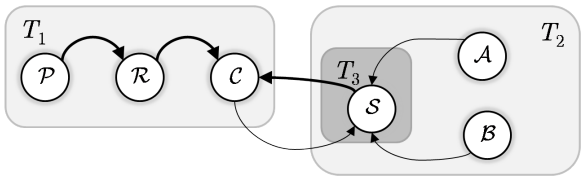


Figura 4.7: Derrotas en el MATM MT_B.

Por lo tanto en este MATM la relación de derrota será tal que $\rightarrow = \{(\mathcal{P},\mathcal{R}),(\mathcal{R},\mathcal{C}),(\mathcal{S},\mathcal{C})\}$.

En la próxima sección, las nociones de aceptabilidad clásicas [Dun95, BG09] serán aplicadas a los MATMs. De esta manera se podrá analizar qué argumentos quedan aceptados en un MATM, en relación a los conceptos introducidos anteriormente en este capítulo.

4.5. Semánticas de Aceptabilidad Basadas en Tipos

Dado que los argumentos en un MATM pueden derrotarse entre sí, no pueden ser aceptados en simultáneo. Por lo tanto, el estado de aceptabilidad de un argumento estará sujeto a una evaluación. Esto se debe a que un argumento debería ser aceptado sólo si "sobrevive"de alguna manera a las derrotas que recibe, o será rechazado en otro caso. Este proceso de evaluación está definido por las semánticas de aceptabilidad.

En este capítulo se adoptará el enfoque de las semánticas basadas en extensiones presentado en el Capítulo 3 (ver [BG09] para más detalles). Una extensión es un conjunto de argumentos que intuitivamente son aceptables en conjunto. En la literatura se ha propuesto una gran cantidad de semánticas diferentes. Esta tesis, como fue mostrado en el Capítulo 3, solo se concentrará en las ampliamente aceptadas semánticas grounded, preferida y estable, propuestas en [Dun95].

Al igual que como se presentó para los marcos argumentativos basados en preferencias en el Capítulo 3, para calcular las extensiones correspondientes a estas semánticas en un MATM se presentará su marco argumentativo abstracto asociado. Es decir, en lugar de definir nuevamente todas estas semánticas en torno a los argumentos y la relación de derrota de un MATM, se caracterizará el marco argumentativo abstracto que representa

al MATM, para luego aplicar la semánticas ya definidas en el Capitulo 3 directamente sobre este marco argumentativo abstracto.

Definición 4.14 (Marco argumentativo asociado de un MATM) *Sea* $MT = (C_T, \rightarrow\!\!\triangleright\!\!-, \mathsf{TAtt}, >_T)$ *un MATM y* $\rightarrow$ *la relación de derrota para los argumentos en* MArgs *(MT). El marco argumentativo* $AF = (\mathsf{MArgs}(MT), \rightarrow)$ *es el marco argumentativo asociado a MT.*

Ejemplo 4.9 *Considere el MATM* MT_B *presentado en el Ejemplo 4.5 y la relación de derrota para este MATM mostrada en el Ejemplo 4.8 el marco argumentativo asociado a* MT_B *será* $AF_B = (\{\mathcal{P}, \mathcal{R}, \mathcal{C}, \mathcal{S}, \mathcal{A}, \mathcal{B}\}, \{(\mathcal{P}, \mathcal{R}), (\mathcal{R}, \mathcal{C}), (\mathcal{S}, \mathcal{C})\})$.

De esta manera, los argumentos aceptados bajo una semántica en el marco argumentativo abstracto asociado a un MATM serán los argumentos aceptados en el MATM bajo esa semántica.Dado un MATM MT, su marco argumentativo asociado AF y una semántica s, un argumento $\mathcal{A}$ es escépticamente aceptado en MT si pertenece a todas las extensiones determinadas por s en AF, $\mathcal{A}$ es crédulamente aceptado en MT si pertenece a alguna (no todas) extensión determinada por s en AF, y $\mathcal{A}$ es rechazado en MT si no pertenece a ninguna extensión de s en AF.

Ejemplo 4.10 *Considere el MATM* MT_B *presentado en el Ejemplo 4.5 y su marco argumentativo asociado* AF_B *presentado en el Ejemplo 4.9. En* AF_B *se obtendrá una única extensión completa, preferida y estable que también coincide con la extensión grounded:* $\{\mathcal{P}, \mathcal{S}, \mathcal{A}, \mathcal{B}\}$. *Por lo tanto, los argumentos* $\mathcal{P}, \mathcal{S}, \mathcal{A}, \mathcal{B}$ *serán aceptados en* MT_B *y los argumentos* $\mathcal{R}, \mathcal{C}$ *serán rechazados en* MT_B.

El hecho de que sea posible caracterizar un marco argumentativo asociado para un MATM es muy importante, ya que además de permitir utilizar las semánticas de aceptabilidad, permite aprovechar todos los resultado existentes y futuros sobre estos marcos. Como se mencionó en el Capítulo 3 los marcos argumentativos abstractos constituyen hoy en día uno de los elementos más utilizados para el desarrollo de nuevos avances en argumentación. Otro resultado importante de esta caracterización es que los MATM pueden considerarse como una forma "más concreta" de representar a un marco argumentativo, esto es, una forma de determinar el origen de los argumentos y derrotas del marco.

4.6. Trabajo Relacionado

En términos del formalismo específico que presentó en este capítulo, no existe ningún trabajo que aborde la formalización del concepto de tipo de argumento de manera explícita y general. Como se mencionó en secciones de este capítulo, varios trabajos del área de argumentación utilizan la idea de tipo de argumento, pero el modelo representacional que emplean está fijo al dominio o problema para el cual desarrollan el sistema argumentativo. En contraste, el formalismo introducido en este capítulo presenta un modelo general y modular en cual es posible representar múltiples tipos de argumento que pueden interactuar entre sí. Por lo tanto, en este formalismo, la representación de tipos y sus relaciones no están restringidas a ningún dominio en particular.

Un ejemplo de la situación mencionada en el párrafo anterior ocurre en el sistema argumentativo para negociación y diálogos presentado en [AP05]. En ese sistema, a partir de una base de conocimiento, se construyen argumentos de amenaza, de recompensa y de explicación para utilizar en una negociación. Esos argumentos tipados se relacionan entre sí ya que se podrán atacar, y además los argumentos de explicación serán preferidos por sobre los argumentos de los otros dos tipos. Además, cada uno de estos tipos de argumento posee, internamente, sus preferencias y conflictos. Claramente, es posible modelar este escenario mediante el uso de MATMs, en cuyo caso se utilizarán tres MATSs: T_E para representar argumentos de explicación, T_A para los argumentos de amenaza y T_R para los argumentos de recompensa, donde cada MATS capturará los conflictos y preferencias internas de estos tipos. La relación de preferencia entre tipos del MATM establecerá que los argumentos de explicación son preferidos a los de amenaza y a los de recompensa, esto es, $T_E{>_T}T_A$ y $T_E{>_T}T_R$. La función de ataque entre tipos de argumento TAtt modelará los conflictos entre estos tres tipos. Por ejemplo, $(\mathcal{A}, \mathcal{B}) \in \mathsf{TAtt}(T_E, T_A)$ si $\mathcal{A}$ está en desacuerdo con alguna de las premisas que llevan a motivar la amenaza representada por $\mathcal{B}$. De esta manera se obtiene el MATM $M_{ND} = (\{T_E, T_A, T_R\}, \emptyset, \mathsf{TAtt}, \{(T_E, T_A), (T_E, T_R)\})$, el cual puede ser utilizado para determinar los argumentos aceptables de una escenario como el presentado en [AP05]. Otro grupo de trabajos que sigue este patrón es el agentes BDI que utilizan razonamiento argumentativo [ADL08, RGS07]. En estos trabajos se utilizan argumentos de creencias, deseos e intenciones para modelar los componentes mentales de los agentes BDI. En esta tesis, en el Capítulo 6, se verá claramente cómo modelar lo representado por dichos trabajos utilizando los MATMs en conjunto con el formalismo que se presentará en el Capítulo 5.

Si bien no existen trabajos que traten el concepto de tipo de argumento como lo hace el formalismo desarrollado en este capítulo, existen algunos formalismos en los que este concepto podría simularse. Un ejemplo de tales formalismos lo constituyen los marcos argumentativos basados en valores, presentados en [BC03]. En la argumentación basada en valores cada argumento puede tener asociado un conjunto de valores que pregona. Este modelo se presenta utilizando marcos argumentativos abstractos, denominados Marcos Argumentativos basados en Valores (MAV), los cuales contienen, además de argumentos y ataques, un conjunto de valores que los argumentos pueden pregonar, una función que mapea argumentos a valores, y una audiencia que determina un orden de preferencia entre estos valores. En términos de la comparación, es posible considerar a los valores como tipos de argumentos, si bien dista de su espíritu. Por lo tanto, de esta manera un argumento tendrá ciertos tipos asociados utilizando la función que mapea argumentos a valores, de forma similar al formalismo desarrollado en este capítulo. Sin embargo, los MAV no proveen un mecanismo general para modelar los conflictos y preferencias internos entre argumentos de un mismo tipo, sino que sólo establece preferencias entre los valores, lo cual sería similar a la relación de preferencias entre tipos de los MATMs. Por otra parte, los MAV no proveen ninguna herramienta para representar herencia o especialización. No obstante, esta comparación no es totalmente justa ya que el propósito de los valores en los MAV no es representar tipos de argumento. Por lo tanto, agregar los elementos que los distancian de los MATM podría no ser adecuado para la semántica de valores.

Por otra parte, con respecto al formalismo en sí, se mencionó que la definición de preferencia interna sigue el espíritu de la agregación de preferencias en el contexto de argumentación presentado en [APP00]. En ese trabajo se presenta un marco argumentativo abstracto donde es posible establecer múltiples preferencias entre los argumentos, proveyéndose además un orden completo entre estas preferencias. Este orden es utilizado para determinar cuál de las preferencias se debe aplicar en cada caso. En el contexto de los MATMs, cada MATS establece una preferencia entre sus argumentos y la relación de herencia entre los MATSs implícitamente impone un orden entre estas relaciones. Sin embargo, este orden no es completo, dado que pueden existir varios caminos de herencia en el $-\triangleright-$ grafo donde la preferencia para dos argumentos está definida.

Como se mostró y mencionó a lo largo del capítulo, los ejemplos y aplicaciones mostrados para los MATMs podrían ser modelados por otros sistemas, particularmente, mediante los marcos argumentativos abstractos. Como se presentó en la sección de aceptabilidad, es

posible pasar de un MATM a su marco argumentativo abstracto asociado. Sin embargo, además de apuntar a formalizar el concepto de tipo de argumento, los MATMs buscan proveer un mecanismo más concreto que los marcos argumentativos abstractos para representar escenarios con argumentos de múltiples tipos que interactúan entre sí, o dominios en los que se requiere razonamiento no monótono y la información puede ser tipificada.

4.7. Conclusiones

En este capítulo se presentó una formalización para el concepto de tipo de argumento en el contexto de los marcos argumentativos abstractos. Se introdujo un modelo formal en el que los tipos de argumento son representados utilizando marcos argumentativos abstractos con preferencias, y se relacionan entre sí a través de ataques, preferencias y herencia entre tipos. Estos tipos individuales en conjunto con sus relaciones componen un marco argumentativo de tipos múltiples o MATM.

Se decidió tratar a los MATSs y a los MATMs como entidades separadas, ya que esto permite manejar a los tipos individuales y sus propiedades internas de manera más clara. Adicionalmente, este modelo contribuye a una mejor modularización de los conceptos, así como también a una especificación directa de la relación de herencia.

Para definir adecuadamente la relación de derrota para estos marcos de múltiples tipos fue necesario considerar las relaciones entre y dentro de los distintos tipos de argumento. En particular, fue necesario estudiar cómo la relación de herencia afectaba a estos tipos, especialmente al momento de determinar la relación de preferencia entre argumentos. Utilizando la relación de derrota y los argumentos de un MATM se mostró cómo determinar cuáles de ellos son aceptables bajo las semánticas de aceptabilidad clásicas presentadas en [Dun95]. Asimismo, se mostró que un MATM puede traducirse a un marco argumentativo abstracto asociado, heredando así todas las propiedades de estos modelos ampliamente utilizados en la comunidad argumentativa.

De esta manera, los MATMs proveen una forma adecuada de representar argumentos con diferentes tipos. La posibilidad que otorgan los MATMs para representar relaciones entre los diferentes tipos permite modularizar las interacciones entre argumentos de tipos diferentes. Por ejemplo, esto se visualiza con las preferencias, permitiendo expresar la preferencia directamente entre los tipos de argumento, sin tener que especificar la preferencia

individual entre cada par de argumentos de diferente tipo. Un ejemplo adicional de esta situación se verá en el Capítulo 6 cuando se presente la preferencia entre los diferentes componentes mentales de un agente. Por otra parte, la relación de herencia introduce un concepto interesante en los sistemas argumentativos, permitiendo modelar situaciones donde un tipo de argumento es una especialización de otro, y permitiendo que todas las relaciones establecidas en un tipo base sean heredadas por los argumentos de un subtipo. Por ejemplo, esto permite modelar, como se verá en el Capítulo 6, situaciones en las que las percepciones de un agente son también creencias.

Capítulo 5

Argumentación Rebatible Basada en Tipos

En este capítulo se presenta T-DeLP, un formalismo en el cual los argumentos poseen una estructura concreta, que a su vez es utilizada para determinar su tipo. Para tal objetivo, se extenderá el lenguaje de DeLP, permitiendo que reglas rebatibles y hechos contengan tipos. Esto llevará a rever las nociones de derivación y argumento, contemplando las diferentes relaciones entre los tipos. La existencia de herencia entre tipos permitirá contar con varias versiones de un argumento, por lo que se introducirá la noción de argumento representativo. Estos argumentos serán los representantes en T-DeLP y serán empleados para determinar cuales son las inferencias del sistema. Por último, se mostrará cómo las nociones presentadas en el capítulo anterior pueden utilizarse para determinar qué argumentos están garantizados en T-DeLP.

5.1. Introducción y Motivación

En el capítulo anterior se presentó un formalismo basado en argumentación abstracta que permite modelar diferentes tipos de argumentos y sus relaciones. Ese sistema se abstraía del origen y de la estructura de sus argumentos, así como también de cómo se determinan sus tipos, para concentrarse en cómo considerar las relaciones entre argumentos y tipos al momento de determinar qué argumentos aceptar. En este capítulo se buscará presentar un formalismo que complemente al del capítulo anterior, es decir, que

permita explicitar cómo se construyen los argumentos (dándoles una estructura interna), y a partir de esta construcción determinar a qué tipos pertenecen. Para abordar estos objetivos se presentará una extensión de DeLP, llamada T-DeLP (por su sigla en inglés Typed-DeLP).

En primer lugar, la motivación principal de T-DeLP será brindar un aproximación que, en conjunto a lo visto en el capítulo anterior, presenta un modelo argumentativo completo para la argumentación basada en tipos de argumento. Es decir, cubrirá todas las etapas de un sistema argumentativo: construcción de argumentos, identificación de tipos, identificación de conflictos, generación de derrotas y cómputo de aceptabilidad; considerando durante todas estas etapas la noción de tipos de argumento. Por otra parte, el estudio de estos temas resulta fundamental para esta tesis, ya que en los lenguajes de programación de agente el conocimiento se expresa mediante un lenguaje de representación concreto. Por lo tanto, si se utilizará argumentación basada en tipos, es esencial determinar cómo se construirán los argumentos a partir de ese conocimiento y cómo se identificarán los tipos de estos argumentos. También resulta interesante estudiar cómo se construyen los argumentos, dado que la mayoría de los sistemas argumentativos que involucran tipos se construyen a partir de conocimiento concreto. Por último, es interesante contar con un lenguaje concreto que permita especificar información conflictiva a la cual se la pueda relacionar con una estructura de tipos o un dominio, ya que se tendrán más medios para determinar qué información prevalece ante los conflictos.

Siguiendo esta motivación, en T-DeLP el concepto de tipo de argumento tendrá un rol protagónico. Claramente, los tipos de los argumentos se determinarán a partir de su estructura interna. Por lo tanto, y dado que será una extensión de DeLP, en T-DeLP se podrán adjuntar tipos a los literales en los hechos y reglas rebatibles de un programa. Estos tipos asociados serán utilizados por la máquina de inferencia para determinar los tipos de los argumentos que los involucren. Esto difiere al objetivo de la programación en lógica tipada [KW91], donde esencialmente se busca establecer un sistema de tipos a programas lógicos para encontrar posibles errores de tipo (aun así habrá ciertas nociones en común).

Los tipos en T-DeLP, al igual que en el formalismo del capítulo anterior, estarán relacionados a través de una relación de herencia. Por lo tanto, será necesario considerar un mecanismo más sofisticado que la igualdad sintáctica para construir las derivaciones. Además, en T-DeLP ciertos tipos podrán propagarse a través de las reglas rebatibles. Esto

permitirá utilizar una regla con tipos generales para inferir instancias de sus literales pero con tipos más específicos, si estos últimos son propagables. Adicionalmente, se espera que si es posible derivar un literal para un tipo, dicho literal sea también derivable para todos sus tipos ancestros (con respecto a la relación de herencia). Todo esto hará que en T-DeLP sea necesario considerar un concepto de derivación más complejo que en DeLP. Para tal objetivo se adaptarán los conceptos de conformidad [KW91] en la derivación T-DeLP. A partir de las derivaciones consistentes que cumplan estas características será posible identificar no sólo los argumentos, sino también sus tipos. En particular, estos tipos serán los tipos que reflejen la derivación en la cual están sustentados los argumentos.

Dadas las características que deberá poseer la derivación en el contexto de T-DeLP, será posible generar distintas versiones de un mismo argumento. Cada versión corresponderá a un tipo de argumento distinto, el cual se encontrará relacionado con los tipos de las demás versiones del argumento a través de herencia. Por lo tanto, en T-DeLP será esencial identificar los *argumentos representativos*. Es decir, de todas las versiones para un mismo argumento, identificar aquella que contenga la mayor información con respecto a sus tipos, siendo así la versión representante del argumento. En este sentido, será fundamental mostrar que T-DeLP puede trabajar únicamente con argumentos representativos, para así evitar los posibles problemas que puedan surgir al utilizar dos versiones de un mismo argumento.

Una vez identificados los argumentos representativos, para poder determinar cómo se derrotan entre ellos y así cuales quedan aceptados, será necesario determinar cómo se atacan y cuáles son las preferencias entre ellos. Aun así, será necesario considerar las preferencias entre los tipos y la relación de herencia entre ellos. Por lo tanto, para determinar la relación de derrota entre estos argumentos, se aprovecharán los marcos argumentativos de múltiples tipos. De este modo, al identificar esta relación y los argumentos representativos, se podrá aplicar cualquier semántica de aceptabilidad. En particular, se mostrará cómo aplicar un procedimiento de prueba dialéctico basado en el de DeLP.

En resumen, T-DeLP extiende a DeLP en los siguiente aspectos:

- permite representar literales con tipos asociados;
- permite modelar herencia entre los tipos;
- posee una relación generalizada de desacuerdo entre literales tal que contempla el concepto de tipo;

- utiliza un mecanismo de derivación que contempla los tipos y sus relaciones;
- sus argumentos tienen tipos asociados;
- los argumentos se caracterizan también a través de los hechos en los que se basan;
- identifican los argumentos representativos, los cuales identifican las versiones con los tipos más especializados para un argumento;
- presenta una noción de subargumentación que toma en cuenta los argumentos representativos;
- los ataques entre argumentos consideran la relación de desacuerdo generalizada y la herencia entre tipos;
- las derrotas se determinan utilizando el formalismo para los MATM presentado en el capítulo anterior; y
- para determinar los argumentos garantizados puede utilizar un proceso de prueba dialéctica, así como también las semánticas de aceptabilidad de [Dun95].

Este capítulo está organizado de la siguiente manera. En la Sección 5.2 se presentará cómo especificar todos los componentes que hacen a un programa T-DeLP. En la Sección 5.3 se indicará cómo son las derivaciones en T-DeLP, cómo se construyen los argumentos a partir de ellas, cómo se obtienen los tipos de estos últimos, y se identificará qué argumentos son representativos. En la Sección 5.4 se mostrará cómo combinar los MATM para calcular acpetabilidad en T-DeLP, para lo cual se definirá el concepto de ataque y preferencias entre argumentos tipados; además, se presentará un procedimiento de prueba dialéctico para T-DeLP. Por último, en las secciones 5.5 y 5.6 se presentarán los trabajos relacionados y las conclusiones del capítulo.

5.2. Representación de Conocimiento en T-DeLP

En esta sección se presentará cómo se especifica el conocimiento en T-DeLP. En particular se indicará cómo extender los componentes básicos de un programa DeLP para que contemple tipos. Además, se introducirán los nuevos elementos relacionados con los tipos que contendrá un programa T-DeLP.

En este capítulo extenderemos los literales clásicos utilizados en la literatura para que contengan tipos. Estos literales extendidos contendrán información referente al tipo de información al cual pertenecen y serán llamados *literales tipados*. Es decir, serán literales que tendrán asociado un identificador del tipo al que pertenecen. El conjunto $\mathbb{T}$ caracterizará a todos los identificadores de tipo utilizables por estos literales tipados. Asumiremos que el conjunto $\mathbb{T}$ siempre contendrá al identificador τ. El tipo τ es es un tipo especial que, como veremos más adelante, corresponde al tipo ancestro de todos los tipos. Además, será utilizado como el tipo más general.

Definición 5.1 (Literal Tipado) *Sean L un literal y $T \in \mathbb{T}$ un identificador de tipo, L^T es un literal tipado de tipo T. Llamaremos $\mathbb{L}^{\mathbb{T}}$ al conjunto de todos los literales tipados.*

Para simplificar la notación, siempre que no se indique un tipo para un literal, asumiremos que su tipo es τ. Es decir, es posible especificar un literal tipado L^{τ} simplemente como L. Por otra parte, al igual que en DeLP, los literales tipados podrán contener variables, la cuales serán identificadas con letras mayúsculas. En particular, los literales que no contengan variables serán denotados como *literales tipados fijos*.

Note que el concepto de tipo asociado a un literal se expresa en un meta-nivel con respecto a la especificación del literal. Otra posibilidad de representación hubiese sido expresar el tipo mediante un argumento (parámetro) del literal. Sin embargo, esto traería complicaciones para modelar adecuadamente el comportamiento de herencia (como se verá más adelante). Además, con esa alternativa sería necesario identificar qué argumentos del literal son de tipo y cuáles no lo son, lo que traería complicaciones al lenguaje de representación.

Ejemplo 5.1 *Considere un escenario similar al descripto en el capítulo anterior, donde un agente debe decidir si comprar una casa en el barrio A. El agente puede obtener información de los diarios y de una página web. Además, cierta información de la página web proviene de amigos. Entonces, para modelar estas fuentes de información, se identifican los siguientes tres tipos:*

- *Noticias (abreviado N), representando información obtenida de las noticias;*
- *Web (abreviado W), que corresponde a información obtenida a través de un post en un foro de la web; y*

- *AmigosWeb (abreviado A), asociado a información obtenida a través de amigos confiables en el foro de la web.*

Por lo tanto, en este el contexto el conjunto $\mathbb{T}$ *sería* $\{\tau, Noticias, Web, AmigosWeb\}$. *A lo largo del capítulo se utilizarán las abreviaturas para estos tipos a fin de facilitar la lectura de reglas y argumentos. En consecuencia, se tratará a* $\mathbb{T}$ *en este escenario como* $\{\tau, N, W, A\}$.

Los literales tipados se utilizarán para representar la información disponible. Así, $articulo(a1, d)^N$ *representará que hay un artículo* $a1$ *del diario* d*, y* $tema(crimen(bA), a1)^N$ *que el artículo* $a1$ *habla de crimen en el barrio* A*, siendo ambos literales de tipo Noticias.*

Por otra parte, respecto a la información web, el literal tipado $caro(bA)^W$ *representa que el barrio* A *es caro, e* $inversion(bB)^W$ *que el barrio* B *es una buena fuente de inversión. Luego se utilizará* $\sim crimen(bA)^A$ *para representar que un amigo había puesto en el foro de la web que en el barrio* A *no hay crimen. Finalmente, este agente podría utilizar el literal comprar(X) de tipo* τ *para representar su decisión a cerca de comprar en un barrio determinado, o el literal seguro(X), también de tipo* τ*, para determinar si un barrio es seguro o no.*

El Ejemplo 5.1 utiliza los identificadores de tipo para representar fuentes de información asociadas a los literales. Existen diversos usos que se le pueden dar a los tipos, como por ejemplo dominios, valores, modos, contextos, propósitos, patrones de razonamiento, o características comunes. A continuación se presentarán dos ejemplos que muestran el uso de tipos en otros escenarios.

Ejemplo 5.2 *Considere un agente que quiere comprar un automóvil. Para tomar esta decisión el agente contará con las características del automóvil, donde cada característica estará asociada a un valor: seguridad, costo, utilidad, o confort. Estos valores serán representados a través de identificadores de tipos. Entonces, el literal* $airbag(c4)^{seguridad}$ *representa que el auto* $c4$ *tiene airbag y por lo tanto promueve el valor seguridad. Mientras que el literal* $aire(c4)^{confort}$ *representa que el* $c4$ *tiene aire acondicionado y promueve el confort.*

Ejemplo 5.3 *Considere el escenario de programación de agentes deliberativos. En este medio es importante identificar el componente mental al que pertenece cada pieza de información, ya que en base a esto su significado será diferente. En particular, considere como componentes mentales creencias y metas*[1]*. Así un literal* $en(caribe, agente2)^{creencia}$ *denota que el agente cree que el "agente2" se encuentra en el caribe, mientras que* $en(caribe, yo)^{meta}$ *denota que el propio agente (que se representa a sí mismo como "yo") tiene como meta estar en el caribe.*

De manera similar a DeLP, el conocimiento en T-DeLP será especificado mediante el uso de programas T-DeLP. A partir de estos programas se construirían argumentos, se determinarán sus tipos y se decidirá cuáles de ellos quedan aceptados.

5.2.1. Hechos y Reglas Tipados

Siguiendo el esquema de los sistemas argumentativos basados en reglas [GS04, Pra10], los programas T-DeLP contendrán *hechos* y *reglas rebatibles*. Los hechos, al igual que en DeLP, representan conocimiento seguro, libre de excepciones. Por otra parte, las reglas rebatibles se emplearán para representar información tentativa. Tanto hechos como reglas rebatibles se construirán a partir de literales tipados, por lo tanto, en T-DeLP se llamarán hechos tipados y reglas rebatibles tipadas. Como es usual en la programación en lógica, a partir de estos hechos y reglas se podrá inferir información y estas inferencias constituirán los argumentos. Como se verá más adelante, los tipos asociados a los literales tendrán un rol fundamental en T-DeLP, ya que serán los que finalmente establezcan los tipos que quedarán asociados a los argumentos.

Definición 5.2 (Hecho Tipado) *Un hecho tipado es un literal tipado fijo.*

Ejemplo 5.4 *Considerando el conjunto de identificadores de tipo* $\{\tau, N, W, A\}$ *presentados en el Ejemplo 5.1, tendremos los siguientes hechos tipados que expresan:*

[1] En el Capítulo 6, se mostrará cómo utilizar estos identificadores de tipo para modelar formalmente diferentes componentes mentales de especificados para un agente a través de un lenguaje de programación de agentes.

$articulo(a1, cronica)^N$	*hay un artículo de nombre a1 en el diario crónica*
$tema(a1, crimen(bA))^N$	*el artículo a1 expresa que hay crimen en el barrio A*
$caro(bA)^W$	*el barrio A es caro*
$inversion(bB)^W$	*el barrio B es una buena fuente de inversión*
$inversion(bA)^W$	*el barrio A es una buena fuente de inversión*
$\sim crimen(bA)^A$	*en el barrio A no hay crimen*

Definición 5.3 (Regla Rebatible Tipada) *Una regla rebatible tipada es un par ordenado (Cabeza,Cuerpo), donde Cabeza es un literal tipado y Cuerpo es conjunto de literales tipados.*

Usualmente una regla rebatible tipada $(L_H, \{L_1, \ldots, L_n\})$ será notada $L_H -\!\prec L_1, \ldots, L_n$. Dada una regla rebatible tipada $R = L_H -\!\prec L_1, \ldots, L_n$, la función $\mathsf{Cabeza}(R)$ devolverá la cabeza de R, y la función $\mathsf{Cuerpo}(R)$ devolverá el cuerpo de R. Además, la función $\mathsf{Tipo}(R)$ devolverá el conjunto de tipos de R.

Generalmente, los literales tipados en el cuerpo de estas reglas representan restricciones respecto al tipo de la información que se utilizará para activarlas. Así también, los tipos de los literales en la cabeza de las reglas representarán el tipo que producirían las inferencias. Estos conceptos se explicarán más adelante, cuando presentemos el proceso de inferencia en T-DeLP.

Ejemplo 5.5 *Continuando con el escenario del agente comprador, se identificarán diferentes grupos de reglas rebatibles, correspondientes a las diferentes fuentes de información que posee el agente. La primera regla expresa que hay razones, a través de las noticias, para creer que un barrio no es seguro, si hay un artículo Art en el diario D enunciando que hay crimen en el barrio X,*

$$(1)\ \sim seguro(X)^N -\!\prec articulo(Art, D)^N, tema(Art, crimen(X))^N$$

La siguiente regla, relacionada con la información establecida por la web, expresa que hay razones para creer que un barrio X es seguro si hay algún post en la web indicando que en el barrio X no hay crimen.

$$(2)\ seguro(X)^W -\!\prec \sim crimen(X)^W$$

Por último, el agente también contará con un conjunto de reglas generales que utilizará para tomar sus decisiones sobre comprar en un lugar o no. Estas reglas utilizarán literales de tipo τ.

(3) $comprar(X) -\!\prec seguro(X)$
(4) $comprar(X) -\!\prec inversion(X)$
(5) $\sim comprar(X) -\!\prec \sim seguro(X)$
(6) $\sim comprar(X) -\!\prec caro(X)$

Note que, a diferencia de los hechos tipados, los literales en las reglas rebatibles tipadas pueden tener variables. No obstante, al igual que DeLP en, T-DeLP sólo emplea instancias fijas de estas reglas. Como se vio en el Capítulo 3, una instancia fija para regla es una versión de la regla tal que las variables son reemplazadas por términos fijos, donde se asume que las variables con el mismo nombre dentro de la regla representan el mismo elemento. Cada regla rebatible tipada con variables es un esquema que representa al conjunto de sus instancias fijas.

Definición 5.4 (Instancia fija de una regla rebatible tipada) *Sea R una regla rebatible tipada. Una instancia fija $R\sigma$ de R se obtiene reemplazando cada una de las variables de R por un término fijo, de forma tal que las variables de R con el mismo nombre sean reemplazadas consistentemente.*

Como se puede observar en el Ejemplo 5.5, al igual que en DeLP, las reglas rebatibles tipadas serán elementos centrales al momento de inferir información a partir de una especificación T-DeLP. Aun así, estas reglas tienen una connotación más profunda que las reglas rebatibles en DeLP. En T-DeLP las reglas no sólo representan una herramienta de inferencia de literales, sino que también son un patrón para determinar de qué tipo serán las inferencias que se obtengan. Por ejemplo, considere la regla $(comprar(X) -\!\prec inversion(X))$ del ejemplo anterior. En esta regla todos los literales son de tipo base o τ. Por lo tanto, es esperable que cuando se utilice esta regla sea posible realizar una inferencia de tipo base para $comprar(bB)$, dado que $inversion(bB)^W$. Sin embargo, en T-DeLP esta regla también podrá ser utilizada para inferir $comprar(bB)^W$. Para poder presentar estos mecanismos con mayor precisión será necesario presentar las diferentes relaciones que pueden existir entre los tipos en T-DeLP.

5.2.2. Relaciones entre los Tipos

Como se vio en el capítulo anterior, una de las nociones más importantes al momento de representar tipos de argumento es la representación de las diferentes relaciones que pueden surgir entre ellos. En ese capítulo se presentaron 3 relaciones entre tipos de argumento: Herencia, Conflicto y Preferencia. Estas relaciones esencialmente influirán al momento de determinar qué argumentos serán aceptados en el sistema.

En T-DeLP se mostrarán las respectivas versiones de estas relaciones, pero teniendo en cuenta que el concepto de tipo afecta a varios elementos subyacentes a los argumentos en sí. Por ejemplo, las relaciones de herencia y conflictos entre tipos determinarán qué derivaciones son posibles a partir de un conjunto de hechos y reglas rebatibles tipados. En consecuencia, antes poder presentar la forma en que se construyen y relacionan los argumentos tipados, se definirán las diferentes relaciones entre los tipos de argumentos.

Como se mencionó anteriormente, los tipos asociados a los literales que denotarán los literales tipados son los tipos que llevarán a identificar los tipos de los argumentos. Siendo esto así, las relaciones entre tipos de argumentos estarán definidas en el contexto de estos identificadores de tipo.

Relación de Herencia

En primera instancia se presentará la relación de herencia entre tipos. Al igual que en el capítulo anterior, la herencia permitirá establecer una relación "es-un" entre los tipos involucrados en un sistema T-DeLP. La principal diferencia es que en T-DeLP los tipos forman parte del lenguaje de representación, siendo parte de las reglas rebatibles y hechos a partir de los literales tipados. Por lo tanto, la relación de herencia no sólo afectará a los argumentos, sino que dará información adicional sobre los literales de una especificación T-DeLP.

Definición 5.5 (Relación de Herencia) *Sea $\mathbb{T}$ el conjunto de todos los identificadores de tipo. Una relación $\rhd : \mathbb{T} \times \mathbb{T}$ será una relación de herencia entre tipos para $\mathbb{T}$ si y solo si $\rhd$ es una relación, reflexiva, antisimétrica y transitiva, y además para todo $T_i \in \mathbb{T}$ vale que $(T_i, \tau) \in \rhd$, y si $(T_i, T_j) \in \rhd$ y $(T_i, T_k) \in \rhd$ entonces $(T_j, T_k) \in \rhd$ o $(T_k, T_j) \in \rhd$.*

Observe que la relación de herencia en T-DeLP, a diferencia de la relación de herencia para el formalismo del capitulo anterior, no soporta herencia múltiple. Esto se debe, como

se mostrará más adelante, a que la herencia múltiple impide la adecuada propagación de tipos a través de las reglas rebatibles en una derivación. Adicionalmente, si T-DeLP permitiese herencia múltiple, no podrían asegurarse ciertas propiedades importantes de su mecanismo de derivación.

Si $(T_1, T_2) \in \rhd$ se dirá que T_1 hereda de T_2 o que T_1 es un subtipo T_2, y también se notará como $T_1 \rhd T_2$. De manera similar, se dirá que T_1 es un descendiente de T_2 y que T_2 es un ancestro de T_1. Adicionalmente, se dirá también que T_1 es un tipo más específico que T_2[2]. Por simplicidad, en los ejemplos presentados a lo largo del capítulo se omitirán aquellos pares de la relación de herencia obtenidos a partir de las características de transitividad y reflexividad de esta relación.

La relación de herencia es una noción fundamental para determinar qué literales son inferibles a partir de un conjunto de reglas rebatibles y hechos tipados. Claramente, a nivel semántico, el hecho de que un tipo herede de otro significa que todos los elementos del primero también serán parte del segundo. Por lo tanto, dado un literal tipado L^T, y una relación de herencia que establece que $T \rhd T_p$, el literal L^T podría aplicarse en cualquier contexto donde se requiera L^{T_p}. Esto se debe a que la intuición de la relación de herencia establece que si un literal es de tipo T también lo será de tipo T_p. En consecuencia, la relación de herencia permitirá que donde sea necesario un literal tipado se pueda utilizar cualquier literal tipado cuyo literal coincida y el tipo sea un tipo descendiente del tipo requerido. Estas nociones se formalizarán más adelante, cuando se introduzca el concepto de derivación.

Ejemplo 5.6 *Como fue mencionado en el ejemplo 5.1, el tipo A relacionado con la información del amigo de la web, es información de la web. Por lo tanto tendremos que* $A \rhd W$. *En este contexto, por ejemplo, si se desea obtener el literal* $\sim crimen(bA)^W$, *pueden utilizarse derivaciones para* $\sim crimen(bA)^W$ *así como también derivaciones para* $\sim crimen(bA)^A$.

Una propiedad importante que cumple una relación de herencia en T-DeLP es que es no circular. Al igual que como se mostró en el Capítulo 4, esto permite construir un grafo dirigido acíclico en el cual los nodos son tipos, los arcos están determinados por la relación de herencia entre los tipos.

[2]Cabe resaltar que decir que un tipo es más específico que otro no tiene ninguna relación con el concepto de especificidad utilizado para la comparación de argumentos en DeLP.

Proposición 5.1 *Si $\rhd$ es una relación de herencia, entonces $\rhd$ es no circular.*

Prueba : Asuma que $\rhd$ es circular. Por lo tanto, sin pérdida de generalidad, existen T_1, T_2 y T_3 tales que $T_1 \rhd T_2$, $T_2 \rhd T_3$, y $T_3 \rhd T_1$. Considerando que por Definición 5.5 $\rhd$ es transitiva tenemos entonces que $T_3 \rhd T_2$, lo cual es un absurdo dado que, por la misma definición, $\rhd$ es no simétrica.

Como se verá más adelante, está propiedad será útil para mostrar otras propiedades de los argumentos y derivaciones en T-DeLP. Esto se debe a que, dado un conjunto de tipos relacionados a través de esta relación, se puede asegurar que habrá un subconjunto de tipos que no heredan de ninguno y otros que heredan de todos.

Relación de Desacuerdo

Como se mencionó en el capítulo anterior, en DeLP dos literales están en desacuerdo si uno es el complemento del otro. En contrapartida, en T-DeLP la relación de desacuerdo será más general. Dos literales complementarios del mismo tipo estarán en desacuerdo, pero en caso de ser de distinto tipo no necesariamente será así . Por ejemplo, considere el escenario de programación de agentes mencionado en el Ejemplo 5.3 en el cual un agente posee un literal tipado $en(caribe, yo)^{meta}$ representando que $en(caribe, yo)$ es una meta, y otro literal $\sim en(caribe, yo)^{creencia}$ representado que $\sim en(caribe, yo)$ es una creencia; claramente, estos literales no estarán en desacuerdo (más aún, uno podría ser la justificación del otro). Por otra parte, podrá ocurrir que dos literales tipados estén en desacuerdo a pesar de no ser complementarios. Por ejemplo, en el mismo escenario, los literales $en(caribe, yo)^{creencia}$ y $en(antartida, yo)^{creencia}$ deberían estar en desacuerdo ya que el agente no puede creer que está en dos lugares simultáneamente.

Para poder modelar las intuiciones de desacuerdo mencionadas en el párrafo anterior, a continuación se definirá la relación de desacuerdo. Esta relación podrá especificar pares de literales que estarán en desacuerdo, donde en particular estarán relacionados todos los pares literales complementarios del mismo tipo.

Definición 5.6 (Relación de Desacuerdo) *Sea $\mathbb{L}^{\mathbb{T}}$ el conjunto de todos los literales tipados, una relación $\otimes : \mathbb{L}^{\mathbb{T}} \times \mathbb{L}^{\mathbb{T}}$ es una relación de desacuerdo entre literales tipados si y solo si es una relación simétrica, y para todo $L^T \in \mathbb{L}^{\mathbb{T}}$ se tiene que $(L^T, \overline{L}^T) \in \otimes$.*

Si $(L_1^U, L_2^V) \in \otimes$, se dirá que L_1^U esta en desacuerdo con L_2^V y viceversa. También se puede decir que L_1^U esta en conflicto con L_2^V, y se puede notar $L_1^U \otimes L_2^V$.

Ejemplo 5.7 *Continuando con el ejemplo del agente que quiere decidir si comprar una casa, se dará que* $seguro(X)^A \otimes \sim seguro(X)^N$, *y* $comprar(bA)^Y \otimes comprar(bB)^X$ *donde X e Y pueden ser cualquier tipo entre* $\{A, W, N, \tau\}$ *. Note que la última relación representa que el agente no puede comprar una casa en el barrio A y en el barrio B a la vez, sea cual fuera el identificador de tipo de la información.*

Preferencias entre tipos

Al igual que en el formalismo presentado en el capítulo anterior, en T-DeLP se podrá especificar una relación de preferencia entre tipos. En este caso, la relación se dará entre identificadores de tipo.

Definición 5.7 (Preferencia entre Tipos) *Sea* $\mathbb{T}$ *el conjunto de todos los identificadores de tipo. Una relación* $>> : \mathbb{T} \times \mathbb{T}$ *será una relación de preferencia entre tipos si y solo si* $>>$ *es una relación no reflexiva.*

Ejemplo 5.8 *Continuando con el escenario en que el agente que debe decidir si comprar la casa en el barrio A, las fuentes de información están caracterizadas por tres identificadores de tipo que determinaran el tipo de sus argumentos. Estos tipos son N para las noticias, W para información de la web y A para información de un amigo en la web. El agente confía más en la información de las noticias que en la de la web, pero confía mas en la de la de sus amigos que en la de las noticias y la de la web. Por lo tanto, tendrá una relación de preferencia* $>> = \{(N, W), (A, N), (A, W)\}$.

Más adelante se mostrará cómo esta relación se puede vincular con el formalismo presentado en el capitulo anterior para computar la aceptabilidad de los argumentos en T-DeLP.

Conjunto de tipos Propagables

Como se mencionó anteriormente cuando se presentaron las reglas rebatibles tipadas, estas reglas no sólo permiten inferir los literales tipados que se encuentran en su cabeza,

sino que también pueden ser utilizadas como plantillas para realizar derivaciones de literales de tipos descendientes. Si por ejemplo se tiene la regla $seguro(X)^W \prec \sim crimen(X)^W$, es posible utilizarla tanto para el tipo W como para los descendientes de W, como por ejemplo A (ver Ejemplo 5.6). Este concepto se denomina *propagación de tipos*, y en la sección que introduce la noción de derivación se mostrará cómo y bajo qué condiciones una regla puede aprovechar este concepto.

A pesar de que la propagación de tipos no es una relación especifica entre tipos, esta noción está vinculada a la relación de herencia. En primer lugar, una especificación de T-DeLP estará caracterizada por un conjunto de tipos que son propagables. Además, este conjunto estará asociado a una relación de herencia ya que, si un tipo es propagable, sus ancestros también lo serán. Esto último se debe a que si se concluye algo de tipo T porque este fue propagado, la conclusión también deberá ser de todos los tipos ancestros de T y, por lo tanto, estos tipos también deben ser propagados.

Definición 5.8 (Conjunto de Tipos Propagables) *Sea $\mathbb{T}$ el conjunto de identificadores de Tipo y $\rhd$ una relación de herencia definida sobre $\mathbb{T}$. Un conjunto de tipos propagables para $\rhd$ es un conjunto $\mathcal{TP} \subseteq \mathbb{T}$ tal que si $T \in \mathcal{TP}$ entonces para todo T' tal que $T \rhd T'$ vale que $T' \in \mathcal{TP}$.*

Ejemplo 5.9 *Considere $\mathbb{T} = \{\tau, T_1, T_2, T_3, T_4\}$ y una relación $\rhd$ para $\mathbb{T}$ tal que $T_1 \rhd \tau$, $T_2 \rhd T_1$, $T_3 \rhd T_1$, $T_4 \rhd T_2$. Si se busca que T_2 sea un tipo propagable, entonces $\mathcal{TP} = \{\tau, T_1, T_2\}$.*

Claramente, si un tipo no está en el conjunto de tipos propagables entonces no se podrá propagar. Adicionalmente, observe que si al menos un tipo es propagable entonces τ será propagable. De todos modos, más adelante cuando se formalice la noción de derivación en T-DeLP, se estudiará el uso de este concepto con mayor profundidad.

Signatura

Una signatura en T-DeLP es la unidad que contiene todas las relaciones de tipo presentadas, y que caracterizará a un programa T-DeLP.

Definición 5.9 (Signatura) *Sea* $\mathbb{T}$ *el conjunto de todos los identificadores de tipo. Una signatura T-DeLP es una tupla* $\Gamma = (\mathbb{T}_\Gamma, \rhd, \otimes, >>, \mathcal{TP})$ *donde* $\mathbb{T}_\Gamma \subseteq \mathbb{T}$*,* $\rhd$ *es una relación de herencia definida sobre* $\mathbb{T}_\Gamma$*,* $\otimes$ *es una relación de desacuerdo definida sobre* $\mathbb{T}_\Gamma$*,* $>>$ *una relación de preferencia entre tipos definida sobre los elementos de* $\mathbb{T}_\Gamma$ *y* $\mathcal{TP}$ *es un conjunto de tipos propagables para* $\mathbb{T}_\Gamma$*.*

Dada una Signatura $\Gamma = (\mathbb{T}_\Gamma, \rhd, \otimes, >>, \mathcal{TP})$, las funciones $\mathsf{FTipos}(\Gamma)$, $\mathsf{FHerencia}(\Gamma)$, $\mathsf{FPref}(\Gamma)$, $\mathsf{FDesacuerdo}(\Gamma)$ y $\mathsf{FTPropagable}(\Gamma)$ devolverán los conjuntos determinados por $\mathbb{T}_\Gamma$, $\rhd$, $>>$, $\otimes$, y $\mathcal{TP}$ respectivamente.

Ejemplo 5.10 *En el escenario desarrollado a lo largo del capítulo se contará con la signatura* $\Gamma = (\{N, W, A, \tau\}, \rhd, \otimes, >>, \{N, W, A, \tau\})$*, donde* $\rhd$ *es la relación de herencia de tipos presentada en el Ejemplo 5.6,* $>>$ *la relación de preferencia presentada en el Ejemplo 5.8 y* $\otimes$ *es la relación de desacuerdo entre literales clasificados presentada en el Ejemplo 5.7. En particular, todos los tipos provistos en la signatura serán propagables.*

5.2.3. Programas T-DeLP

Utilizando los elementos anteriormente definidos se identificará la unidad básica a partir de la cual se realizarán las inferencias. Al igual que en las aproximaciones de programación en lógica, el conocimiento será especificado a través de un programa T-DeLP. Este programa, básicamente, será un conjunto de hechos y reglas rebatibles tipadas, en conjunto con una signatura que debe hacer referencia tanto a los tipos de los hechos como a los de las reglas rebatibles.

Definición 5.10 (Programa T-DeLP) *Un programa T-DeLP es una tupla* (Γ, Θ, Δ) *donde,* Θ *es un conjunto de hechos tipados,* Δ *es un conjunto de reglas rebatibles tipadas y* Γ *es un una signatura tal que los tipos de literales tipados en* Θ *y* Δ *están incluidos en* $\mathsf{FTipos}(\Gamma)$*, y no existen* L_1^U *y* L_2^V *en* Θ *tales que* $(L_1^U, L_2^V) \in \mathsf{FDesacuerdo}(\Gamma)$*.*

Ejemplo 5.11 *Para modelar el escenario que se ha desarrollado a lo largo de la tesis, se utilizará el programa T-DeLP* $\mathcal{P}^\Gamma = (\Gamma, \Theta, \Delta)$*, donde* Θ *y* Δ *son los conjuntos de hechos y reglas rebatibles tipados presentados en los ejemplos 5.4 y 5.5:*

$$\Theta \qquad\qquad \Delta$$

$$\left\{\begin{array}{l} articulo(a1, cronica)^N \\ tema(a1, crimen(bA))^N \\ caro(bA)^W \\ inversion(bB)^W \\ inversion(bA)^W \\ \sim crimen(bA)^A \end{array}\right\} \quad \left\{\begin{array}{l} \sim seguro(X)^N -\!\!< articulo(Art, D)^N, tema(Art, crimen(X))^N \\ seguro(X)^W -\!\!< \sim crimen(X)^W \\ comprar(X) -\!\!< seguro(X) \\ comprar(X) -\!\!< inversion(X) \\ \sim comprar(X) -\!\!< \sim seguro(X) \\ \sim comprar(X) -\!\!< caro(X) \end{array}\right\}$$

y $\Gamma = (\mathbb{T}_\Gamma, \rhd, \otimes, >>, \mathcal{TP})$ *es la signatura es la presentada en el Ejemplo 5.10, donde* $\mathbb{T}_\Gamma = \{N, W, A, \tau\}$, $\rhd = \{(N, \tau), (W, \tau), (A, W)\}$, $\otimes$ *es la relación de desacuerdo presentada en el Ejemplo 5.7,* $>> = \{(A, W), (N, W), (A, N)\}$, *y* $\mathcal{TP} = \{N, W, A, \tau\}$.

5.3. Construcción de Argumentos Tipados en T-DeLP

En esta sección se introducirá cómo se construyen los argumentos en T-DeLP. Con tal fin, en primer lugar se estudiará cómo derivar literales tipados, teniendo en cuenta la relación de herencia y los tipos propagables. Luego se presentará cómo construir argumentos tipados a partir de las derivaciones obtenidas. Aun así, el concepto central de esta sección serán los *argumentos representativos*. Como se verá en la sección existirán distintas versiones de un mismo argumento, y el argumento representativo será aquel que agrupe las características de todas estas versiones.

5.3.1. Derivación

Como se vio en el Capítulo 3, el conocimiento en DeLP se representa a través de un programa (Θ, Δ) caracterizado por un conjunto de hechos Θ y un conjunto de reglas rebatibles Δ. En T-DeLP se utiliza un programa (Γ, Θ, Δ) que cuenta con hechos tipados Θ y reglas rebatibles tipadas Δ, y una signatura Γ que contiene información de los tipos utilizados en estos hechos y reglas. En DeLP un literal L es derivable a partir de un programa (Θ, Δ) si existe una secuencia de literales $L_1, \ldots, L_n$ donde $L_n = L$; además, cada L_i en la secuencia es un hecho o L_i *es igual* a la cabeza de una regla y todos los literales del cuerpo de la regla *son iguales* a literales que aparecen antes de L_i en la secuencia. En T-DeLP no es posible utilizar el concepto de "igualdad" como en DeLP,

dado que es necesario considerar los tipos de los literales. Por ejemplo, considere los siguientes programas T-DeLP:

$\mathcal{P}_1^\Gamma$	$\mathcal{P}_2^\Gamma$	$\mathcal{P}_3^\Gamma$		$\mathcal{P}_4^\Gamma$	
$a \prec b$	$a \prec b^{t1}$	$a \prec b^{t1}$	t2 $\rhd$ t1	$a \prec b^{t1}$	t2 $\rhd$ t1
$b \prec c$	$b^{t2} \prec c^{t3}$	$b^{t2} \prec c^{t3}$	t5 $\rhd$ t4	$b^{t2} \prec c^{t3}$	t5 $\rhd$ t4
$b \prec e$	$b^{t1} \prec e^{t4}$	$b^{t1} \prec e^{t4}$		$b^{t1} \prec e^{t4}$	$\mathcal{TP} = \{\tau, \text{t1}\}$
c	c^{t3}	c^{t3}		c^{t3}	
e	e^{t5}	e^{t5}		e^{t5}	

En el programa $\mathcal{P}_1^\Gamma$ todos los literales son del mismo tipo (el tipo τ), y por lo tanto hay derivaciones para a, b (a través de dos reglas distintas), c y e. Note que en este caso, el programa $\mathcal{P}_1^\Gamma$ es como un programa DeLP.

En contraste, observe el programa $\mathcal{P}_2^\Gamma$, donde existen literales con distintos tipos. En este programa es esperable derivar únicamente c^{t3} y e^{t5} por ser hechos, y b^{t2} por la segunda regla. Por otra parte, b^{t1} no debería ser derivable ya que, por más que e^{t5} sea un hecho y se cuente con la regla ($b^{t1} \prec e^{t4}$), e^{t4} y e^{t5} son de distinto tipo. Una situación similar ocurre para a considerando b^{t1} y b^{t2}.

Considere ahora el programa $\mathcal{P}_3^\Gamma$ que posee los mismos hechos y reglas rebatibles que $\mathcal{P}_2^\Gamma$, pero además introduce que $t2 \rhd t1$ y $t5 \rhd t4$. Intuitivamente, si $t5$ hereda de $t4$, entonces todo literal de tipo $t5$ también es de tipo $t4$. Por lo tanto, por más que e^{t5} y e^{t4} tengan distinto tipo, e^{t5} debería *conformar* con e^{t4} para así permitir la derivación de b^{t1}. De esta manera, siguiendo esta intuición, a partir del programa $\mathcal{P}_3^\Gamma$ se podría derivar a, además de c^{t3}, e^{t5} y b^{t2} que ya eran derivables en $\mathcal{P}_2^\Gamma$.

Finalmente, considere el programa $\mathcal{P}_4^\Gamma$. Note que a diferencia de $\mathcal{P}_3^\Gamma$ incorpora que el tipo $t1$ es propagable (y también el tipo τ, dado que $t1$ hereda de τ). Por lo tanto, sería interesante poder propagar $t1$ en aquellas derivaciones en que se utilice una regla rebatible tal que los literales tipados empleados para activar la regla sean de tipo $t1$ y la cabeza de la regla sea de un tipo que ancestro de $t1$. En este caso, como b^{t1} es derivable, sería posible propagar el tipo $t1$ a a en la regla ($a \prec b^{t1}$), permitiendo así derivar a^{t1}.

Como puede observarse a partir de los ejemplos en T-DeLP, para construir una derivación no es suficiente contar con la igualdad sintáctica de literales (como en el caso de DeLP), sino que también se tendrán en cuenta los tipos asociados a los literales y la

relación de herencia entre los tipos. En T-DeLP el concepto básico para construir una derivación es la noción de *conformidad* [KW91]. Esta noción puede visualizarse claramente en el programa $\mathcal{P}_3^\Gamma$ entre los literales e^{t5} y e^{t4}. La conformidad será el mecanismo utilizado por T-DeLP en una derivación, y reemplazará a la igualdad sintáctica empleada en DeLP.

Definición 5.11 (Conformidad) *Sea $\mathcal{P}^\Gamma = (\Gamma, \Theta, \Delta)$ un programa T-DeLP. Diremos que un literal tipado L_1^U conforma con otro literal tipado L_2^V en $\mathcal{P}^\Gamma$, notado como $L_1^U \circledcirc L_2^V$, si y solo si $L_1 = L_2$ y $U \triangleright V \in \mathsf{FHerencia}(\Gamma)$.*

Ejemplo 5.12 *Sea $\mathcal{P}^\Gamma$ el programa T-DeLP del Ejemplo 5.11, dado que $A \triangleright W \triangleright \tau$ tendremos por ejemplo que $\sim crimen(bA)^A$ conforma con $\sim crimen(bA)^W$, $inversion(bB)^W$ conforma inversion(bB), $caro(bA)^W$ conforma con caro(bA), y $seguro(bA)^A$ conforma con seguro(bA). Por otra parte, observe que por ejemplo $caro(bA)^N$ no conforma con $caro(bA)^W$ ya que sus tipos no están relacionados, y caro(bA) no conforma con $caro(bA)^W$ ya que el tipo del primero no hereda del segundo (es un supertipo).*

Cabe destacar que, al estar basada en la relación de herencia, la conformidad es una relación no simétrica entre literales tipados. Una excepción a esto se da en el caso de que se trate exactamente del mismo literal tipado, es decir, igual literal e igual tipo. Además, la conformidad es una relación transitiva.

Proposición 5.2 *Si $\circledcirc$ es una relación de conformidad para los elementos de un programa T-DeLP $\mathcal{P}^\Gamma$, entonces $\circledcirc$ es una relación reflexiva y transitiva.*

Prueba : (transitividad) Se debe probar que si $L^U \circledcirc L^V$ y $L^V \circledcirc L^W$ entonces $L^U \circledcirc L^W$. Por definición de conformidad vale que $U \triangleright V$ y $V \triangleright W$. Además, por definición $\triangleright$ es una relación transitiva, por lo tanto, $U \triangleright V$ y entonces $L^U \circledcirc L^W$. □

(reflexividad) Considere L^T un literal tipado cualquiera. Por Definición de herencia vale que $T \triangleright T$, con lo cual $L^T \circledcirc L^T$. □

Como se mencionó anteriormente, la conformidad de literales permitirá definir las inferencias en T-DeLP. Además de diferenciarse de DeLP en la forma en que se enlazan los elementos para obtener una derivación, en T-DeLP una derivación contendrá un tipo, es decir, el tipo bajo el cual es posible derivar el literal. Intuitivamente, la derivación de un literal tipado en T-DeLP puede ocurrir en tres situaciones: cuando un hecho (o

una presunción) conforma con el literal tipado; cuando la cabeza de una regla rebatible conforma con el literal tipado y todos los literales tipados del cuerpo de la regla son derivables; o cuando el literal tipado conforma con la cabeza de una regla rebatible, todos los literales tipados del cuerpo de la regla son derivables, y además permiten propagar el tipo del literal tipado en cuestión.

Definición 5.12 (Derivación Rebatible Tipada) *Sea* $\mathcal{P}^\Gamma = (\Gamma, \Theta, \Delta)$ *un programa T-DeLP y* L^T *un literal tipado. Existe una derivación rebatible tipada para* L^T *a partir de* $\mathcal{P}^\Gamma$*, notado* $\mathcal{P}^\Gamma \mid\sim L^T$*, si y solo si existe una secuencia finita de literales tipados* $S = [L_1^{T_1}, \ldots, L_n^{T_n}]$*, donde* $L_n^{T_n} = L^T$*, tal que para todo* $L_i^{T_i}$ *en la secuencia se cumple una de las siguientes condiciones:*

1. *existe* $L_i^M \in \Theta$ *tal que* $L_i^M \circledcirc L_i^{T_i}$*;*
2. *existe una instancia fija* $R\sigma$ *para una regla* $R \in \Delta$ *tal que* $\mathsf{Cabeza}(R\sigma) \circledcirc L_i^{T_i}$*, y para todo* $B_j^{D_j}$ *en* $\mathsf{Cuerpo}(R\sigma)$ *existe un* $B_j^{T_k}$ *que aparece en la secuencia antes de* $L_i^{T_i}$ *y es tal que* $B_j^{T_k} \circledcirc B_j^{D_j}$*; o*
3. *existe una instancia fija* $R\sigma$ *para una regla* $R \in \Delta$ *con* $\mathsf{Cuerpo}(R\sigma) \neq \emptyset$ *y un tipo* $T_i \in \mathsf{FTPropagable}(\Gamma)$ *tales que* $L_i^{T_i} \circledcirc \mathsf{Cabeza}(R\sigma)$*, y para todo* $B_j^{D_j}$ *en* $\mathsf{Cuerpo}(R\sigma)$ *existe un* $B_j^{T_k}$ *que aparece en la secuencia antes de* $L_i^{T_i}$ *y es tal que* $B_j^{T_k} \circledcirc B_j^{D_j}$ *y* $T_k \rhd T_i$*.*

Si $\mathcal{P}^\Gamma \mid\sim L^T$ también se dirá que hay una derivación de tipo T para L a partir de $\mathcal{P}^\Gamma$. Además, la secuencia finita de literales que caracteriza a una derivación $\mathcal{P}^\Gamma \mid\sim L^T$ es llamada secuencia de derivación de $\mathcal{P}^\Gamma \mid\sim L^T$.

Siguiendo el primer ítem de la Definición 5.12, claramente, un hecho será derivable; pero también serán derivables todos los literales tipados cuyo tipo sea un ancestro (con respecto a la relación de herencia). Por ejemplo, siguiendo el programa T-DeLP del Ejemplo 5.11 dado que $A \rhd W \rhd \tau$ y tenemos el hecho tipado $\sim crimen(bA)^A$, serán derivables los literales $\sim crimen(bA)^A$, $\sim crimen(bA)^W$ y $\sim crimen(bA)^\tau$. Es decir, existirá una derivación rebatible para un literal tipado si existe un hecho con el cual *conforme*.

El segundo ítem implica que puede existir una inferencia para un literal tipado si existe una regla rebatible tipada tal que su cabeza conforme con ese literal. Es decir, que la cabeza posea un literal con un subtipo del elemento a derivar. En este caso debe

existir una derivación rebatible tipada para todos los literales tipados del cuerpo de esta regla. Para ilustrar esto, considere por ejemplo el programa T-DeLP del Ejemplo 5.11. Teniendo en cuenta la regla $\sim seguro(X)^N \prec articulo(Art, D)^N, tema(Art, crimen(X))^N$ y los hechos tipados $articulo(a1, cronica)^N$ y $tema(a1, crimen(bA))^N$, serán derivables tanto $\sim seguro(bA)^N$ como $\sim seguro(bA)^\tau$.

La situación especificada por el tercer ítem enmarca el caso en que un literal tipado se deriva aplicando la propagación de tipos. Esto permite que una regla con literales de cierto tipo T se pueda utilizar para derivar literales con tipos descendientes de T, que en particular sean propagables. Por ejemplo, considere la regla $\sim comprar(X) \prec caro(X)$ y el hecho $caro(bA)^W$ del Ejemplo 5.11. En este caso, utilizando esta regla y considerando que $W \in \mathcal{TP}$ será posible derivar $\sim comprar(bA)^W$.

Como se ve en la Definición 5.12, para realizar una propagación es necesario que se cumplan tres condiciones. En primer lugar, el tipo del literal a derivar debe ser propagable. En segundo lugar, el literal tipado a derivar debe conformar con la cabeza de una regla rebatible, permitiendo utilizar una regla cuya cabeza tenga un tipo ancestro del tipo del literal tipado a derivar. Por último, deben existir derivaciones que conformen con los elementos del cuerpo de la regla y que promuevan el tipo propagable. Es decir, los tipos de los literales que conforman con los elementos del cuerpo de la regla utilizada deben ser descendientes del tipo del literal que originalmente se quiere derivar.

Ejemplo 5.13 *Considere el programa T-DeLP* $\mathcal{P}^\Gamma = (\Theta, \Delta, \Gamma)$ *del Ejemplo 5.11. Dado que* $\mathsf{FTPropagable}(\Gamma) = \{A, W, N, \tau\}$ *a partir de* $\mathcal{P}^\Gamma$ *se puede obtener una derivación para* $comprar(bA)^A$ *a partir de la secuencia de derivación* $[\sim crimen(bA)^A, seguro(bA)^A, comprar(bA)^A]$. *Note que esta derivación se obtiene a través de las reglas* $seguro(X)^W \prec \sim crimen(X)^W$ *y* $comprar(X) \prec seguro(X)$, *en conjunto con el hecho* $\sim crimen(bA)^A$, *y teniendo en cuenta que el tipo A es propagable.*

En la Figura 5.1 se presenta un esquema detallado de cómo se realiza esta derivación del Ejemplo 5.13. En primera medida se aplica la primer alternativa de la definición de derivación, seleccionando el hecho $\sim crimen(bA)^A$ *como primer elemento de la secuencia. Luego, observe que es posible incluir en la secuencia* $seguro(bA)^A$, *dado que antes aparece* $\sim crimen(bA)^A$ *y a través de la regla rebatible* $seguro(X)^W \prec \sim crimen(X)^W$ *es posible propagar A. Note que la propagación de A es posible porque A pertenece al conjunto de tipos propagables,* $\sim crimen(bA)^A$ *conforma con* $\sim crimen(X)^W$, *y* $seguro(bA)^A$ *conforma*

con $seguro(X)^W$. *Finalmente, siguiendo un razonamiento análogo al anterior, es posible añadir el literal* $comprar(bA)^A$ *a la secuencia. En esta figura es posible visualizar cómo conforman los literales en la secuencia con los cuerpos de las reglas rebatibles, y cómo se propaga el tipo A (los círculos que encierran a A en las flechas que se dirigen a las cabezas de las reglas). Adicionalmente, note que los cuadros sombreados contienen los literales tipados que formarán la derivación para* $comprar(bA)^A$.

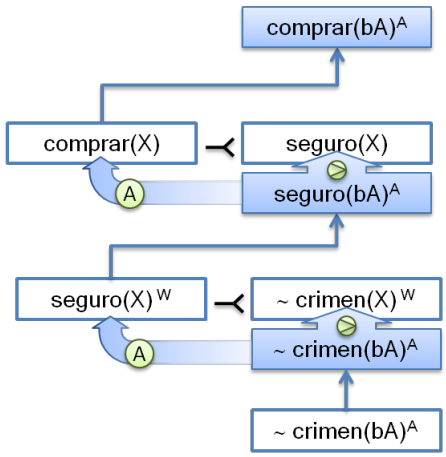


Figura 5.1: Derivación detallada para $comprar(bA)^A$.

Como se puede ver en el ejemplo anterior, la propagación utilizada por el mecanismo de derivación permite aprovechar reglas definidas para tipos ancestros, permitiendo así propagar tipos descendientes. A continuación se mostrarán las derivaciones para el ejemplo del agente que quiere comprar su casa en un barrio, que aprovechan el concepto de propagación.

Ejemplo 5.14 *Considere el programa T-DeLP* $\mathcal{P}^\Gamma = (\Gamma, \Theta, \Delta)$ *del Ejemplo 5.11, donde:*

$$\Theta = \left\{ \begin{array}{l} articulo(a1, cronica)^N \\ tema(a1, crimen(bA))^N \\ caro(bA)^W \\ inversion(bB)^W \\ inversion(bA)^W \\ \sim crimen(bA)^A \end{array} \right\} \quad \Delta = \left\{ \begin{array}{l} \sim seguro(X)^N -\!\prec articulo(Art, D)^N, tema(Art, crimen(X))^N \\ seguro(X)^W -\!\prec \sim crimen(X)^W \\ comprar(X) -\!\prec seguro(X) \\ comprar(X) -\!\prec inversion(X) \\ \sim comprar(X) -\!\prec \sim seguro(X) \\ \sim comprar(X) -\!\prec caro(X) \end{array} \right\}$$

Teniendo en cuenta que $\mathsf{FTPropagable}(\Gamma) = \{A, W, N, \tau\}$ *se pueden obtener, entre otras, derivaciones para:*

- $comprar(bA)^A$ *por* $[\sim crimen(bA)^A, seguro(bA)^A, comprar(bA)^A]$;
- $comprar(bB)^W$ *por* $[inversion(bB)^W, comprar(bB)^W]$;
- $\sim comprar(bA)^N$ *por* $[articulo(a1, cronica)^N, tema(a1, crimen(bA))^N, \sim seguro(bA)^N, \sim comprar(bA)^N]$; *y*
- $\sim comprar(bA)^W$ *por* $[caro(bA)^W, \sim comprar(bA)^W]$

Observe que las reglas para decidir si comprar o no en un barrio son ampliamente aprovechadas por el mecanismo de propagación de tipos. De no contar con la propagación de tipos, en este escenario sería necesario definir cada una de las reglas acerca de si comprar o no comprar para cada tipo que las pudiera utilizar. Por lo tanto, la propagación brinda un mecanismo para agrupar patrones de razonamiento común para un conjunto de tipos descendientes en una regla rebatible con tipos ancestros.

El mecanismo de derivación de T-DeLP captura la intuición de que si hay razones para creer en un literal bajo un tipo, es esperable que haya razones para creer en ese literal bajo los ancestros de ese tipo. Por lo tanto, si se deriva un literal con un tipo T también podrá derivarse a ese literal con los ancestros de T. Por ejemplo, considere el programa T-DeLP $\mathcal{P}^\Gamma = (\Gamma, \Theta, \Delta)$ del Ejemplo 5.11. Para este programa, a partir del hecho $\sim crimen(bA)^A$ y la regla $seguro(X)^W \prec \sim crimen(X)^W$, es posible derivar: $seguro(bA)^A$, con la secuencia de derivación $[\sim crimen(bA)^A, seguro(bA)^A]$; $seguro(bA)^W$, con la secuencia de derivación $[\sim crimen(bA)^A, seguro(bA)^W]$; y $seguro(bA)^\mathcal{T}$, con la secuencia de derivación $[\sim crimen(bA)^A, seguro(bA)^\mathcal{T}]$. La siguiente proposición muestra que esta propiedad es siempre satisfecha en T-DeLP.

Proposición 5.3 *Sea* $\mathcal{P}^\Gamma = (\Theta, \Delta, \Gamma)$ *un programa T-DeLP. Si* $\mathcal{P}^\Gamma \mid\sim L^T$ *entonces* $\mathcal{P}^\Gamma \mid\sim L^{T'}$ *para todo T' ancestro de T.*

Prueba : Si $\mathcal{P}^\Gamma \mid\sim L^T$ *por Definición 5.12 se puede construir una secuencia S, donde* L^T *es el último elemento de S por alguna de las siguientes razones:*

- *Existe un hecho tipado* L^H *que conforma con* L^T*, con lo cual* $H \triangleright T$*. Dado que por hipótesis* $T \triangleright T'$*, tenemos entonces que* $H \triangleright T'$*, por lo cual* L^H *conforma con* $L^{T'}$*. Por lo tanto, por Definición 5.12,* $\mathcal{P}^\Gamma \mid\sim L^{T'}$. □

- *Existe una instancia fija $R\sigma$ de una regla de Δ tal que $\mathsf{Cabeza}(R\sigma) = L^A$ conforma con L^T y para cada B^D en $\mathsf{Cuerpo}(R\sigma)$ hay un B^C en S tal que B^C conforma con B^D. Entonces $A \triangleright T$, y como por hipótesis $T \triangleright T'$ entonces $A \triangleright T'$. Por lo tanto L^A conforma con $L^{T'}$. Por lo tanto, por Definición 5.12, $\mathcal{P}^\Gamma \mid\sim L^{T'}$. □*

- *T es un tipo propagable y existe una instancia fija $R\sigma$ de una regla de Δ tal que L^T conforma con $L^A = \mathsf{Cabeza}(R\sigma)$ y para cada B^D en $\mathsf{Cuerpo}(R\sigma)$ hay un B^C en S tal que B^C conforma con B^D y $C \triangleright T$. Dado que L^T conforma con L^A vale que $T \triangleright A$, y como $T \triangleright T'$ entonces por Definición 5.5 vale que o $T' \triangleright A$ o $A \triangleright T'$. Entonces:*

  - *Si $T' \triangleright A$, entonces $L^{T'}$ conforma con L^A, y como $C \triangleright T$ y $T \triangleright T'$ entonces $C \triangleright T'$. Además, como T es propagable y $T \triangleright T'$ entonces T' es propagable. Luego por Definición 5.12, $\mathcal{P}^\Gamma \mid\sim L^{T'}$. □*
  - *Si $A \triangleright T'$, L^A conforma con $L^{T'}$. Por lo tanto, por Definición 5.12, $\mathcal{P}^\Gamma \mid\sim L^{T'}$.□*

Observe que esta propiedad vale porque la relación de herencia en T-DeLP no permite herencia múltiple. Si lo permitiese, el tercer caso de la prueba no sería válido, y por lo tanto no se podrían propagar los tipos adecuadamente. Como se verá mas adelante, esta propiedad es importante al momento de considerar los tipos de los argumentos en T-DeLP.

Un resultado adicional que surge a partir de una derivación, es que todos los literales en la secuencia de derivación son literales derivables bajo los tipos con que aparecen en la secuencia.

Proposición 5.4 *Sea S una secuencia de derivación para L^T a partir de un programa T-DeLP $\mathcal{P}^\Gamma$. Si B^U aparece en S entonces $\mathcal{P}^\Gamma \mid\sim B^U$.*

Prueba : Suponga que $\mathcal{P}^\Gamma$ no deriva B^U. Entonces no existe en $\mathcal{P}^\Gamma$ una secuencia de derivación que termina en B^U. Sea S' la subsecuencia de S que termina B^U, entonces S' no sería una secuencia de derivación. Por lo tanto, por Definición 5.12 como S' es una subsecuencia de S, S tampoco sería una secuencia de derivación, lo cual contradice la hipótesis. □

5.3.2. Conflictos

Anteriormente en la Sección 5.2.2 se presentó la noción de desacuerdo entre literales tipados. Esta noción determina cuándo dos literales de determinados tipos no puede

coexistir, *i.e.* no pueden aceptarse como inferencias en conjunto. Si bien en T-DeLP es posible derivar literales en desacuerdo, el mecanismo argumentativo será el encargado de determinar cuáles de estos literales son los que prevalecen. Aun así, los argumentos deben constituir una unidad de razonamiento consistente en conjunto con la información estricta del programa. Dado que los argumentos son conjuntos de reglas rebatibles tipadas, es importante identificar si un conjunto de estas reglas es conflictivo, es decir, si a partir de ellas es posible derivar literales que están en desacuerdo.

En DeLP para determinar si un conjunto de reglas rebatibles es conflictivo se analiza si ese conjunto permite derivar dos literales complementarios (en desacuerdo). En T-DeLP no es posible utilizar esta noción, ya que es necesario considerar la relación de herencia de los literales tipados y la mecánica de derivación. Por ejemplo, considere el siguiente programa T-DeLP en el contexto de programación de agentes:

$$\Theta = \left\{ \; en(alaska, yo)^{creencia} \; \right\} \Delta = \left\{ \begin{array}{l} en(caribe, yo)^{meta} -\!\prec \sim en(caribe, yo)^{creencia} \\ \sim en(caribe, yo)^{creencia} -\!\prec en(alaska, yo)^{creencia} \end{array} \right\}$$

donde $\rhd = \{(\text{creencia},\tau)(\text{meta},\tau)\}$ y $\otimes$ sólo contiene los desacuerdos para los literales complementarios del mismo tipo. Observe que a partir de Δ y Θ es posible derivar, entre otros, los literales $\sim en(caribe, yo)^{creencia}$, $en(caribe, yo)^{meta}$, $\sim en(caribe, yo)^{\tau}$ y $en(caribe, yo)^{\tau}$. Si se utilizara la noción de conjunto conflictivo de DeLP, el conjunto de reglas Δ sería un conjunto conflictivo porque permite derivar $\sim en(caribe, yo)^{\tau}$ y $en(caribe, yo)^{\tau}$, los cuales están en desacuerdo. Claramente, es esperable que Δ no sea un conjunto conflictivo ya que $\sim en(caribe, yo)^{creencia}$ es una justificación para $en(caribe, yo)^{meta}$, y en este dominio sería deseable que Δ constituya un argumento para $en(caribe, yo)^{meta}$. Más aun, considere que los conjuntos de hechos y reglas rebatibles tipados del programa anterior son $\Theta = \{ \; en(caribe, yo)^{meta}, \sim en(caribe, yo)^{creencia} \; \}$ y $\Delta = \emptyset$. En este escenario si se emplea la noción de conjunto conflictivo de DeLP, el conjunto vacío de hechos sería conflictivo.

El problema de utilizar la noción de conjunto conflictivo de DeLP en T-DeLP surge al comparar los desacuerdos producidos por todas las derivaciones de distinto tipo para un literal. En el ejemplo del párrafo anterior, el desacuerdo se producía en el tipo τ, pero no en los tipos más especializados para las derivaciones de esos literales. En ese ejemplo note que, si bien $\sim en(caribe, yo)^{\tau}$ y $en(caribe, yo)^{\tau}$ están en desacuerdo, $\sim en(caribe, yo)^{creencia}$ y $en(caribe, yo)^{meta}$ no lo están. En general, es deseable considerar únicamente los desacuerdos de los tipos más específicos ya que, como se muestra en

ejemplo del párrafo anterior, un desacuerdo especificado para un tipo ancestro puede ser descartado en un tipo descendiente. Por lo tanto, en T-DeLP, para definir un conjunto conflictivo sólo se analizarán los literales derivados con el tipo más específico posible.

Definición 5.13 (Derivación más específica) [3] *Sea* $\mathcal{P}^\Gamma = (\Gamma, \Theta, \Delta)$ *un programa T-DeLP. Una derivación mas específica para un literal* L *a partir de* $\mathcal{P}^\Gamma$*, notado* $\mathcal{P}^\Gamma|\sim {}_E L^T$*, es una derivación* $\mathcal{P}^\Gamma|\sim L^T$ *tal que no existe* T' *en* $\mathsf{FTipos}(\Gamma)$ *tal que* $\mathcal{P}^\Gamma|\sim L^{T'}$ *y* $T' \triangleright T$*.*

Ejemplo 5.15 *Si se considera el programa del Ejemplo 5.11, la derivación más específica para el literal* $comprar(bA)$ *es de tipo* A*.*

Las derivaciones más específicas permiten determinar cuál es el tipo *real* de un literal. Por lo tanto, un conjunto de reglas rebatibles y hechos tipados será conflictivo para un programa T-DeLP, si combinado con la información estricta y basándose en la signatura, hay derivaciones más específicas para dos o más literales tipados que están en desacuerdo. Formalmente:

Definición 5.14 (Conjunto Conflictivo) *Sea* $\mathcal{P}^\Gamma = (\Gamma, \Theta, \Delta)$ *un programa T-DeLP. Un conjunto de reglas rebatibles tipadas* $\Delta_C \subseteq \Delta$ *será conflictivo con respecto a* $\mathcal{P}^\Gamma$ *si* $(\Gamma, \Theta, \Delta_C)|\sim {}_E L_1^U, L_2^V$*, tales que* $(L_1^U, L_2^V) \in \mathsf{FDesacuerdo}(\Gamma)$*.*

Ejemplo 5.16 *Considere el programa T-DeLP del Ejemplo 5.11. En este programa el conjunto con reglas* $C = \{(seguro(X)^W -\!\prec \sim crimen(X)^W), (comprar(X) -\!\prec seguro(X)), (\sim comprar(X) -\!\prec caro(X))\}$ *es un conjunto conflictivo ya que* $(\Gamma, \Theta, C)|\sim {}_E comprar(bA)^A, \sim comprar(bA)^W$ *los cuales están en desacuerdo.*

Una propiedad fundamental que debe cumplir la noción de conjunto conflictivo es que $\emptyset$ no sea un conjunto conflictivo. Recuerde que la noción de conjunto conflictivo de DeLP no cumple esta propiedad en T-DeLP.

Proposición 5.5 *Sea* $\mathcal{P}^\Gamma$ *un programa T-DeLP. El conjunto* $\emptyset$ *no es un conjunto conflictivo para* $\mathcal{P}^\Gamma$*.*

[3] Nuevamente, este concepto está asociado a la relación de herencia de T-DeLP y no al criterio de comparación *especificidad generalizada* de DeLP.

Prueba : *Suponga que* $\emptyset$ *es un conjunto conflictivo para* $\mathcal{P}^{\Gamma} = (\Gamma, \Theta, \Delta)$. *Entonces* $(\Gamma, \Theta, \emptyset) \sim_{E} L_1^{C_1}, L_2^{C_2}$, *tales que* $(L_1^{C_1}, L_2^{C_2}) \in$ FDesacuerdo(Γ). *Como el conjunto de reglas rebatibles es vacío entonces, por las definiciones 5.13 y 5.12,* $L_1^{C_1}$ *y* $L_2^{C_2}$ *son dos hechos tipados de* Θ. *Por lo tanto* $\mathcal{P}^{\Gamma}$ *no es un programa T-DeLP, lo cual contradice la hipótesis.*□

5.3.3. Argumentos Tipados

Como se vio en la sección anterior, a partir de un programa T-DeLP es posible derivar literales que, no obstante, pueden estar en desacuerdo. Por lo tanto, la derivación no es suficiente para determinar qué información será aceptada a partir de un programa T-DeLP. Para determinar si un literal tipado es finalmente aceptado en T-DeLP, al igual que en DeLP, se construirán argumentos para ese literal tipado. De esta manera, la aceptación del literal dependerá de la aceptación de los argumentos que lo sustentan, para lo cual se analizarán todos los contra argumentos para los argumentos obtenidos.

En DeLP un argumento para un literal es un conjunto minimal y no conflictivo de reglas rebatibles tales que, junto al conocimiento estricto, permiten derivar el literal. En T-DeLP, dado que hay literales tipados, un argumento estará por el conjunto de hechos tipados y reglas tipadas que utiliza para derivar al literal tipado que concluye. Además, los argumentos en T-DeLP estarán caracterizados por un conjunto de tipos, los cuales determinarán con qué otros argumentos están en conflicto y, en caso de estarlo, cuáles son preferidos.

Dada la naturaleza relación de herencia, en T-DeLP se podrán obtener varias versiones del mismo argumento pero con distintos tipos. Por lo cual, más adelante se mostrará cómo identificar los *argumentos representativos* de entre estos argumentos. Un argumento representativo será aquel que agrupe todas las características de las distintas versiones del argumento.

Un argumento para un literal tipado L^T en T-DeLP será un conjunto minimal de reglas rebatibles tipadas y hechos tipados, tales que en conjunto permiten derivar L^T. Adicionalmente, como se mencionó anteriormente, un argumento debe ser una pieza de información consistente con respecto al programa a partir del cual se construye. Por lo tanto, su conjunto de reglas rebatibles tipadas debe ser no conflictivo.

Definición 5.15 (Argumento) *Sea L^T un literal tipado y $\mathcal{P}^\Gamma = (\Gamma, \Theta, \Delta)$ un programa T-DeLP. Un argumento para L^T es un par $\langle \mathcal{A}, L^T \rangle$ con $\mathcal{A} \subseteq \Theta \cup \Delta$, $\Theta_\mathcal{A} = \mathcal{A} \cap \Theta$ y $\Delta_\mathcal{A} = \mathcal{A} \cap \Delta$, tal que:*

1. $(\Theta_\mathcal{A}, \Delta_\mathcal{A}, \Gamma) \mid\sim L^T$,
2. $\Delta_\mathcal{A}$ *no es un conjunto conflictivo con respecto a* $\mathcal{P}^\Gamma$, *y*
3. $\mathcal{A}$ *es minimal. Es decir, no existe un subconjunto propio* $\mathcal{A}'$ *de* $\mathcal{A}$ *tal que* $\mathcal{A}'$ *satisface las condiciones (1) y (2).*

Note que se espera que el conjunto de reglas rebatibles tipadas del argumento no sea conflictivo con respecto al programa $\mathcal{P}^\Gamma$, el cual incluye todos los hechos tipados. Esto se debe a que, al igual que en DeLP, se espera que un argumento sea consistente con respecto a la parte estricta del programa que lo genera.

Como se muestra en la Definición 5.15, dado un argumento $\langle \mathcal{A}, L^T \rangle$, la componente $\mathcal{A}$ será tratada como un conjunto de hechos tipados y reglas rebatibles. Cuando sea necesario identificar individualmente hechos y reglas de un argumento $\langle \mathcal{A}, L^T \rangle$, se utilizarán los conjuntos $\Theta_\mathcal{A}$ y $\Delta_\mathcal{A}$ respectivamente. Más abajo, en la Observación 5.1 se explicará el por qué de la inclusión de los hechos tipados en los argumentos T-DeLP.

Ejemplo 5.17 *Considere el programa del Ejemplo 5.11 y la derivación mostrada en la Figura 5.1 para el literal $comprar(bA)^A$. Observe que $\mathcal{A}_1 = \{\sim crimen(bA)^A, (seguro(X)^W \prec \sim crimen(X)^W), (comprar(X) \prec seguro(X))\}$ permite derivar $comprar(bA)^A$, y en particular es el conjunto mínimo que lo permite. Por lo tanto, $\langle \mathcal{A}_1, comprar(bA)^A \rangle$ será un argumento de este programa T-DeLP. A continuación se muestran algunos otros argumentos que se pueden construir a partir de este programa:*

(1) $\langle \{\sim crimen(bA)^A, (seguro(bA)^W \prec \sim crimen(bA)^W)\}, seguro(bA)^W \rangle$
(2) $\langle \{\sim crimen(bA)^A, (seguro(X)^W \prec \sim crimen(bA)^W)\}, seguro(bA)^A \rangle$
(3) $\langle \{\sim crimen(bA)^A, (seguro(X)^W \prec \sim crimen(bA)^W)\}, seguro(bA) \rangle$
(4) $\langle \{\sim crimen(bA)^A\}, \sim crimen(bA)^A \rangle$
(5) $\langle \{\sim crimen(bA)^A\}, \sim crimen(bA)^W \rangle$
(6) $\langle \{\sim crimen(bA)^A\}, \sim crimen(bA) \rangle$
(7) $\langle \{inversion(bB)^W, (comprar(bB) \prec inversion(bB))\}, comprar(bB)^W \rangle$
(8) $\langle \{inversion(bB)^W, (comprar(bB) \prec inversion(bB))\}, comprar(bB) \rangle$
(9) $\langle \{inversion(bB)\}^W, inversion(bB)^W \rangle$
(10) $\langle \{inversion(bB)\}^W, inversion(bB) \rangle$

Sin embargo note que, por ejemplo, no es posible construir un argumento para $comprar(bB)^A$ *yq que no hay una derivación para ese literal a partir del programa T-DeLP. Por otra parte, note que si* $\mathcal{A}_n = \{\sim crimen(bA)^A$, $popular(cronica)^N$, $seguro(bA)^W \prec \sim crimen(bA)^W\}$, $\langle \mathcal{A}_n, seguro(bA)^A \rangle$ *no es un argumento porque* $\mathcal{A}_n$ *no es minimal.*

Observación 5.1 *Una cuestión notable que diferencia a T-DeLP de DeLP, es que los argumentos en T-DeLP están caracterizados también por el conocimiento estricto que utilizan. En DeLP, como fue presentado en el Capítulo 3, no es necesario incluir este conocimiento como parte de los argumentos, dado que si existe más de una derivación estricta utilizada en la construcción de un argumento, la conclusión del argumento será la misma. En T-DeLP, al contar con tipos y propagación de tipos a través de las reglas, es posible crear argumentos diferentes que comparten las reglas rebatibles utilizadas pero no la parte estricta.*

Ejemplo 5.18 *Considere que se tienen los hechos tipados* b^{t1} *y* b^{t2}*, la regla rebatible tipada* $(a \prec b)$ *y los tipos* $t1$ *y* $t2$ *no están relacionados, pero ambos son propagables. En este contexto, en T-DeLP se construirán dos argumentos distintos para* a $\langle \{(a \prec b), b^{t1}\}, a^{t1} \rangle$ *y* $\langle \{(a \prec b), b^{t2}\}, a^{t2} \rangle$*, los cuales se diferencian en el hecho en el cual están basados.*

Note que de no estar presente la parte estricta, no se podría determinar cómo estos argumentos llegan a conclusiones diferentes. Aun así, como se puede ver en el Ejemplo 5.17, existen argumentos que difieren en su conclusión pero comparten los hechos y las reglas en los cuales están basados. Sin embargo, estos argumentos tienen ciertas características que los hacen especiales, ya que en realidad corresponden a versiones de un único argumento conocido como *argumento representativo*. Esta clase de argumentos será estudiada en profundidad en la Sección 5.3.4.

Todo argumento que no utilice reglas rebatibles en su construcción, es decir, aquel que sea construible a partir de un hecho tipado del programa, será llamado argumento estricto. Por ejemplo, los argumentos $\langle \{inversion(bB)^W\}, inversion(bB)^W \rangle$ y $\langle \{\sim crimen(bA)\}^A, \sim crimen(bA)^A \rangle$ son algunos de los argumentos estrictos del ejemplo anterior. Nótese que los argumentos estrictos están constituidos por un único hecho tipado. Adicionalmente, observe que los argumentos estrcitos se corresponden con los argumentos vacíos de DeLP presentados en el Capítulo 3.

Un subargumento en T-DeLP es un argumento que forma parte de la activación un argumento que lo contiene. Por lo tanto, un subargumento está constituido por un subconjunto de las reglas rebatibles del argumento que lo contiene, y tiene la propiedad de que su conclusión es parte de la secuencia de derivación del argumento que lo contiene.

Definición 5.16 (Subargumento) *Sean $\langle \mathcal{A}, L^U \rangle$ y $\langle \mathcal{B}, M^V \rangle$ argumentos. $\langle \mathcal{B}, M^V \rangle$ es un subargumento de $\langle \mathcal{A}, L^U \rangle$ si y solo si $\mathcal{B} \subseteq \mathcal{A}$ y M^V aparece en la secuencia de derivación de L^U a partir de $\mathcal{A}$.*

Ejemplo 5.19 *Considere los argumentos presentados en el Ejemplo 5.17. Note que, por ejemplo, el argumento (4) es un subargumento de argumento (2), y los argumentos (4) y (5) pueden ser subargumentos del argumento (1). Por otra parte, el argumento (5) no es un subargumento de (2), porque $\sim crimen(bA)^W$ no permite derivar $seguro(bA)^A$ (debe aparecer $\sim crimen(bA)^A$).*

La definición de argumento presentada más arriba define cómo es un argumento estructuralmente, es decir, qué elementos lo componen. En T-DeLP los argumentos adicionalmente tendrán asociado un número de tipos que dependen de la conclusión que el argumento sostiene. Para esto, se presentará la siguiente función que retornará un conjunto con todos los tipos asociados a un argumento.

Definición 5.17 (Función de Tipo de Argumento) *Sea $\mathcal{P}^\Gamma = (\Gamma, \Theta, \Delta)$ un programa T-DeLP, Args el conjunto de todos los argumentos de $\mathcal{P}^\Gamma$, y $\langle \mathcal{A}, L^T \rangle \in \mathsf{Args}$. La función de tipo de argumento $\mathsf{FTA} : \mathsf{Args} \rightarrow 2^{\mathsf{FTipos}(\Gamma)}$, se define declarativamente como:*

- $T \in \mathsf{FTA}(\langle \mathcal{A}, L^T \rangle)$
- *Si $D \in \mathsf{FTA}(\langle \mathcal{A}, L^T \rangle)$ y $D \triangleright D'$ entonces $D' \in \mathsf{FTA}(\langle \mathcal{A}, L^T \rangle)$*

Note que, a partir de esta definición, los tipos de un argumento serán el tipo de su conclusión y todos sus ancestros. A continuación se presentarán ejemplos de esta función.

Ejemplo 5.20 *Considere los argumentos del Ejemplo 5.17 en el contexto del programa T-DeLP presentado en el Ejemplo 5.11. Debajo se listan los resultados para la función de tipo de argumento, para los argumentos según como fueron numerados en ese ejemplo:*

$\mathsf{FTA}((1)) = \{\tau, W\}$	$\mathsf{FTA}((6)) = \{\tau\}$
$\mathsf{FTA}((2)) = \{\tau, W, A\}$	$\mathsf{FTA}((7)) = \{\tau, W\}$
$\mathsf{FTA}((3)) = \{\tau\}$	$\mathsf{FTA}((8)) = \{\tau\}$
$\mathsf{FTA}((4)) = \{\tau, W, A\}$	$\mathsf{FTA}((9)) = \{\tau, W\}$
$\mathsf{FTA}((5)) = \{\tau, W\}$	$\mathsf{FTA}((10)) = \{\tau\}$

Un resultado importante a resaltar con respecto a los argumentos de T-DeLP, es que todos sus tipos están relacionados a través de la relación de herencia. Es decir, un argumento no tendrá dos o más tipos que no estén directamente relacionados a través de la relación de herencia.

Proposición 5.6 *Sea $\mathcal{P}^\Gamma$ un programa T-DeLP. Si $\langle \mathcal{A}, L^T \rangle$ es un argumento de $\mathcal{P}^\Gamma$, entonces para todo T_i y T_j en $\mathsf{FTA}(\langle \mathcal{A}, L^T \rangle)$ vale que $T_i \rhd T_j$ o $T_j \rhd T_i$.*

Prueba : Se tiene que T está en $\langle \mathcal{A}, L^T \rangle$ por Definición 5.17, y dado que por Definición 5.5 la relación de herencia $\rhd$ es transitiva, se tiene entonces que $T \rhd T_i$ y $T \rhd T_j$. Luego, por Definición 5.5 vale que o $T_i \rhd T_j$ o $T_j \rhd T_i$. □

Note que esta propiedad muestra cierta característica de los argumentos en T-DeLP que no se presentaba en los argumentos de los Marcos Argumentativos de Tipos Múltiples del capítulo anterior. En los MATM un argumento podía pertenecer a cualquier tipo, por más que estos no estén relacionados.

Otra cuestión importante a destacar es que la propiedad 5.6 implica que un argumento tendrá un único tipo más específico, que será el del literal tipado que concluye. Como se presentará en la próxima subsección, esto ayudará a identificar los argumentos representativos en T-DeLP.

Proposición 5.7 *Sea $\mathcal{P}^\Gamma$ un programa T-DeLP. Si $\langle \mathcal{A}, L^T \rangle$ es un argumento de $\mathcal{P}^\Gamma$, entonces para todo T' en $\mathsf{FTA}(\langle \mathcal{A}, L^T \rangle)$ tal que $T \neq T'$, vale que $T \rhd T'$.*

Prueba : Directa por Definición 5.17. □

5.3.4. Argumentos Representativos

Considere los siguientes argumentos, donde $t \rhd t1$, $l \neq r$, $\mathcal{A} \neq \mathcal{B}$ y q no está relacionado con t ni con $t1$.

(1) $\langle \mathcal{A}, l^{t1} \rangle$ (2) $\langle \mathcal{A}, l^{t} \rangle$ (3) $\langle \mathcal{A}, l^{q} \rangle$
(4) $\langle \mathcal{A}, r^{t} \rangle$ (5) $\langle \mathcal{B}, l^{t} \rangle$

Asuma que (1) es un argumento siempre válido. En DeLP, como fue visto en el capítulo 3, si no se consideran los tipos de los literales y (1) es válido, entonces (2),(3) y (5) podrían ser argumentos válidos también, mientras que (4) no ya que, a pesar de usar $\mathcal{A}$, está concluyendo otro literal distinto de l. En T-DeLP, como se tiene en consideración a los tipos, la situación es diferente. Si (1) es un argumento válido en T-DeLP, entonces (2) y (5) también pueden ser argumentos válidos. Por el contrario, (3) no es un argumento ya que, si bien comparte $\mathcal{A}$ con (1), concluye un literal tipado cuyo tipo no está relacionado con el de (1). Por último, (4) tampoco sería un argumento por la misma razón que en DeLP.

En particular, considere los argumentos (1),(2) y (5). Claramente, en T-DeLP (5) es un argumento diferente a (1) y (2) ya que, si bien llega a la misma conclusión que (1), el conjunto de reglas y hechos que utiliza para llegar a esa conclusión es diferente. Por otra parte, la situación entre (1) y (2) es diferente. Tanto (1) como (2) son versiones del mismo argumento, pero que identifican los diferentes tipos que puede tener ese argumento. En esta sección se estudiarán estos argumentos, y en particular se mostrará cómo identificar el más representativo entre todas las versiones.

Considere los siguientes argumentos del Ejemplo 5.17, para el agente que tiene que decidir si compra una casa en el barrio A, en conjunto con sus respectivos tipos presentados en el Ejemplo 5.20:

(1) $\langle \{\sim crimen(bA)^A, (seguro(bA)^W \prec \sim crimen(bA)^W)\}, seguro(bA)^W \rangle$

(2) $\langle \{\sim crimen(bA)^A, (seguro(bA)^W \prec \sim crimen(bA)^W)\}, seguro(bA)^A \rangle$

(3) $\langle \{\sim crimen(bA)^A, (seguro(bA)^W \prec \sim crimen(bA)^W)\}, seguro(bA) \rangle$

Observe que los tres argumentos coinciden en los conjuntos de reglas y hechos que utilizan. Por lo tanto, coinciden en el literal que concluyen (pero no en el tipo). Por otra parte, note que los tipos del argumento *(1)* están incluidos en los del argumento *(2)*, y los del argumento *(3)* están incluidos en los del argumento *(1)* (y por lo tanto también en los de *(2)*), es decir, $\mathsf{FTA}(\textit{(3)}) \subseteq \mathsf{FTA}(\textit{(1)}) \subseteq \mathsf{FTA}(\textit{(2)})$. Esto se debe a que *(1)*, *(2)* y *(3)* son todas versiones del mismo argumento, cada una para un tipo distinto relacionado a través herencia. Sin embargo, esta versiones no son equivalentes, siendo una de ellas

la más representativa. Esta versión del argumento es la captura más características ya que incluye a todos los tipos que puede tener el argumento, motivo por el cual se la denominará versión representativa del argumento.

Definición 5.18 (Argumento representativo) *Sea* $\mathcal{P}^\Gamma$ *un programa T-DeLP y* $\langle \mathcal{A}, L^T \rangle$ *un argumento de* $\mathcal{P}^\Gamma$*. Diremos que* $\langle \mathcal{A}, L^T \rangle$ *es un argumento representativo en* $\mathcal{P}^\Gamma$ *si para todo* $\langle \mathcal{A}, L^{T'} \rangle$ *de* $\mathcal{P}^\Gamma$ *con* $T' \neq T$*, vale que* $\mathit{FTA}(\langle \mathcal{A}, L^{T'} \rangle) \subset \mathit{FTA}(\langle \mathcal{A}, L^T \rangle)$.

Ejemplo 5.21 *Considere el programa T-DeLP* $\mathcal{P}^\Gamma = (\Gamma, \Theta, \Delta)$ *del Ejemplo 5.11, donde:*

$$\Theta \left\{ \begin{array}{l} articulo(a1, cronica)^N \\ tema(a1, crimen(bA))^N \\ caro(bA)^W \\ inversion(bB)^W \\ inversion(bA)^W \\ \sim crimen(bA)^A \end{array} \right\} \qquad \Delta \left\{ \begin{array}{l} \sim seguro(X)^N -\!\!< articulo(Art, D)^N, tema(Art, crimen(X))^N \\ seguro(X)^W -\!\!< \sim crimen(X)^W \\ comprar(X) -\!\!< seguro(X) \\ comprar(X) -\!\!< inversion(X) \\ \sim comprar(X) -\!\!< \sim seguro(X) \\ \sim comprar(X) -\!\!< caro(X) \end{array} \right\}$$

$\mathbb{T} = \{A, W, N, \tau\}$, $A \triangleright W$, $W \triangleright \tau$, $N \triangleright \tau$, $\otimes$ *es la relación de desacuerdo presentada en el Ejemplo 5.7, y todos los tipos de* $\mathbb{T}$ *son propagables. Los argumentos representativos de* $\mathcal{P}^\Gamma$ *se muestran en la Figura 5.2 a continuación.*

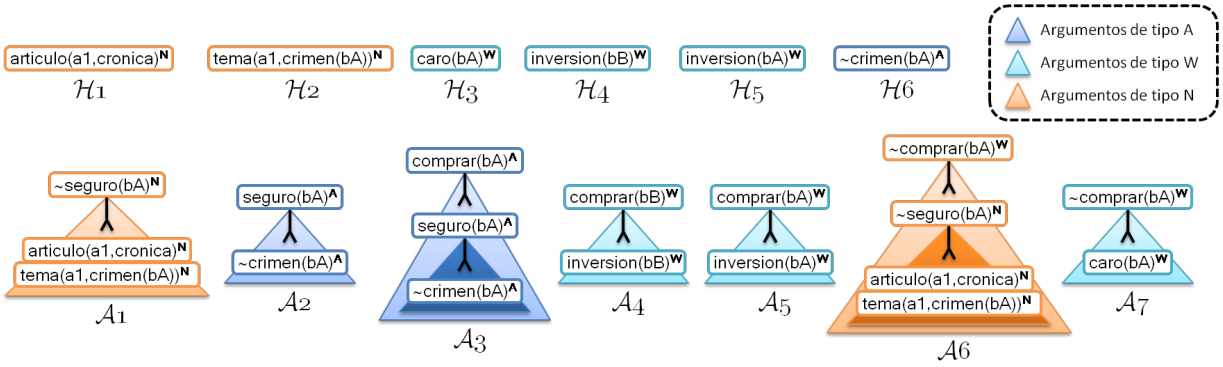


Figura 5.2: Argumentos representativos para el programa del Ejemplo 5.11.

Una propiedad que cumplen todos los argumentos representativos es que la derivación involucrada en su concepción es una derivación más específica para la conclusión del argumento.

Proposición 5.8 *Sea $\mathcal{P}^{\Gamma} = (\Gamma, \Theta, \Delta)$ un programa T-DeLP. Si $\langle \mathcal{A}, L^T \rangle$ es un argumento representativo de $\mathcal{P}^{\Gamma}$ con $\mathcal{A} = \Theta_{\mathcal{A}} \cup \Delta_{\mathcal{A}}$, entonces $(\Theta_{\mathcal{A}}, \Delta_{\mathcal{A}}, \Gamma) \mid\sim L^T$ es una derivación más específica para L.*

Prueba : Suponga que $(\Theta_{\mathcal{A}}, \Delta_{\mathcal{A}}, \Gamma) \mid\sim L^T$ no es la derivación más específica para L en $(\Theta_{\mathcal{A}}, \Delta_{\mathcal{A}}, \Gamma)$. Entonces por Definición 5.13 de derivación mas específica, existe un Te, tal que $Te \neq T$, $Te \rhd T$ y $(\Theta_{\mathcal{A}}, \Delta_{\mathcal{A}}, \Gamma) \mid\sim L^{Te}$. Entonces por Definición 5.15 de argumento, $\langle \Theta_{\mathcal{A}} \cup \Delta_{\mathcal{A}}, L^{Te} \rangle$ es un argumento en $\mathcal{P}^{\Gamma}$. Luego, por Definición 5.17 de función de tipo de argumento, Te está en $\mathsf{FTA}(\langle \Theta_{\mathcal{A}} \cup \Delta_{\mathcal{A}}, L^{Te} \rangle)$, pero por Proposición 5.7 Te no está en $\mathsf{FTA}(\langle \Theta_{\mathcal{A}} \cup \Delta_{\mathcal{A}}, L^T \rangle)$ ya que $Te \rhd T$. Entonces $\mathsf{FTA}(\langle \Theta_{\mathcal{A}} \cup \Delta_{\mathcal{A}}, L^{Te} \rangle) \not\subseteq \mathsf{FTA}(\langle \Theta_{\mathcal{A}} \cup \Delta_{\mathcal{A}}, L^T \rangle)$, y se tiene que $\langle \Theta_{\mathcal{A}} \cup \Delta_{\mathcal{A}}, L^T \rangle$ no es un argumento representativo, lo cual contradice la hipótesis. □

La característica más importante que poseen los argumentos representativos es que son únicos. Es decir, no existe ningún otro argumento representativo que utilice las mismas reglas rebatibles tipadas y hechos tipados, y concluya el mismo literal pero con tipo diferente. Esta característica es demostrada por el siguiente Teorema.

Teorema 5.1 *Sea $\mathcal{P}^{\Gamma} = (\Theta, \Delta, \Gamma)$ un programa T-DeLP. Si $\langle \mathcal{A}, L^T \rangle$ es un argumento representativo de $\mathcal{P}^{\Gamma}$, entonces no existe otro argumento representativo de $\mathcal{P}^{\Gamma}$ $\langle \mathcal{A}, L^S \rangle$ tal que $S \neq T$.*

Prueba : Suponga que existe un argumento representativo $\langle \mathcal{A}, L^S \rangle$ con $S \neq T$. Por Definición 5.18 de argumento representativo $\mathsf{FTA}(\langle \mathcal{A}, L^N \rangle) \subset \mathsf{FTA}(\langle \mathcal{A}, L^S \rangle)$ para todo $N \neq S$. En particular, $\mathsf{FTA}(\langle \mathcal{A}, L^T \rangle) \subset \mathsf{FTA}(\langle \mathcal{A}, L^S \rangle)$, con lo cual $\mathsf{FTA}(\langle \mathcal{A}, L^S \rangle) \not\subseteq \mathsf{FTA}(\langle \mathcal{A}, L^T \rangle)$. Luego, $\langle \mathcal{A}, L^T \rangle$ no es un argumento representativo, lo cual contradice la hipótesis. □

Adicionalmente, una propiedad que cumplen los argumentos representativos es que el tipo del literal que concluyen es un descendiente de todos los tipos de los argumentos no representativos para ese mismo literal.

Proposición 5.9 *Sea $\mathcal{P}^{\Gamma} = (\Theta, \Delta, \Gamma)$ un programa T-DeLP. Si $\langle \mathcal{A}, L^T \rangle$ es un argumento representativo de $\mathcal{P}^{\Gamma}$, entonces para todo argumento $\langle \mathcal{A}, L^S \rangle$ con $S \neq T$ vale que $T \rhd S$.*

Prueba : Si $\langle \mathcal{A}, L^T \rangle$ es un argumento representativo, entonces por el Teorema 5.1 $\langle \mathcal{A}, L^S \rangle$ con $S \neq T$ no es un argumento representativo. Entonces, por Definición 5.18 de argumento representativo vale que $\mathsf{FTA}(\langle \mathcal{A}, L^S \rangle) \subset \mathsf{FTA}(\langle \mathcal{A}, L^T \rangle)$. Luego por Definición 5.17

de función de tipo de argumento S está en $\mathsf{FTA}(\langle \mathcal{A}, L^S \rangle)$, con lo cual S está también en $\mathsf{FTA}(\langle \mathcal{A}, L^T \rangle)$. Luego por Proposición 5.7 vale que $T \rhd S$. □

Otra cuestión importante a destacar, es que si un argumento representativo se puede construir utilizando a un argumento no representativo como subargumento, entonces también se puede construir utilizando la versión más representativa de ese subargumento.

Teorema 5.2 *Sea $\langle \mathcal{A}, L^T \rangle$ un argumento representativo de un programa T-DeLP $\mathcal{P}^\Gamma$. Si $\langle \mathcal{B}, M^U \rangle$ es un subargumento de $\langle \mathcal{A}, L^T \rangle$, entonces existe un argumento representativo $\langle \mathcal{B}, M^{U'} \rangle$ en $\mathcal{P}^\Gamma$ tal que $\langle \mathcal{B}, M^{U'} \rangle$ también es subargumento de $\langle \mathcal{A}, L^T \rangle$.*

Prueba : Si $\langle \mathcal{B}, M^U \rangle$ es un subargumento de $\langle \mathcal{A}, L^T \rangle$, entonces por Definición 5.16 $\mathcal{B} \subseteq \mathcal{A}$ y M^U aparece en la secuencia de derivación S para L^T construida a partir de $\mathcal{A}$. Entonces se puede dar uno de los siguientes casos:

- *Si M^U no es el último elemento de S, entonces por Definición 5.12 de derivación M^U conforma con un literal tipado M^V del cuerpo de alguna regla de $\mathcal{A}$. Por lo tanto, se tiene que $U \rhd V$. Si existe un argumento representativo $\langle \mathcal{B}, M^{U'} \rangle$, entonces por Proposición 5.9 vale que $U' \rhd U$, y por Definición 5.5 de herencia se tiene que $U' \rhd V$, con lo cual $M^{U'}$ conforma con M^V. Luego, siendo que $M^{U'}$ es derivable a partir del mismo conjunto de reglas y hechos tipados que M^U, entonces $M^{U'}$ puede aparecer en lugar de M^U en la secuencia de derivación para L^T a partir de $\mathcal{A}$. Por lo tanto, $\langle \mathcal{B}, M^{U'} \rangle$ es un subargumento de $\langle \mathcal{A}, L^T \rangle$.* □
- *Si M^U es el último elemento de la secuencia S, entonces $U = T$, $M = L$ y $\mathcal{A} = \mathcal{B}$. Por lo tanto, $\langle \mathcal{B}, M^U \rangle$ es un argumento representativo y es un subargumento de $\langle \mathcal{A}, L^T \rangle$.* □

Ejemplo 5.22 *Considere un programa T-DeLP con una regla rebatible $a^{t1} \prec b^{t2}$, un hecho b^{t3} y la relación de herencia $t3 \rhd t2$. Claramente, $\mathcal{A} = \langle \{a^{t1} \prec b^{t2}\}, a^{t1} \rangle$ es un argumento representativo, note que $\mathcal{B}' = \langle \{b^{t2}\}, b^{t2} \rangle$ es un subargumento para $\mathcal{A}$, pero $\mathcal{B}'$ no es representativo. Sin embargo, observe que el argumento $\mathcal{B} = \langle \{b^{t3}\}, b^{t3} \rangle$, que es el argumento representativo de $\mathcal{B}'$, también es un subargumento de $\mathcal{A}$.*

En T-DeLP es importante considerar únicamente las versiones representativas de los argumentos al momento de determinar cuáles de ellos son aceptables. Si se consideran

todas la versiones de un argumento para determinar su aceptabilidad, estas versiones podrían utilizarse como defensa entre sí ante ataques de otros argumentos, lo cual llevaría a un proceso argumentativo falaz. Además, si se considerasen todas las versiones de un argumento podrían surgir conflictos erróneos con otros argumentos. Intuitivamente, dos argumentos estarán en conflicto si sus conclusiones están en desacuerdo. Como se mostró anteriormente en la subsección de conflictos, si no se considera la versión mas específica de los literales en desacuerdo se pueden generar conflictos incorrectos. Asimismo, los conflictos entre argumentos se pueden producir a nivel de subargumento, lo que resalta la importancia de utilizar subargumentos representativos.

Los resultados obtenidos en esta sección muestran que es posible utilizar a los argumentos representativos como representantes de las distintas versiones de un argumento. Esto se debe a que son únicos, y a que efectivamente cada uno de ellos *representa* a todas las versiones del mismo argumento, siendo la versión del argumento que agrupa a todos los tipos posibles.

Finalmente, una última propiedad notacional correspondiente a los argumentos representativos es que se los puede identificar unívocamente mediante su conjunto de reglas y hechos rebatibles tipados. Esto es, si se tiene el argumento representativo $\langle \mathcal{A}, L^T \rangle$, se lo puede identificar unívocamente simplemente indicándolo como $\mathcal{A}$. Esto es posible gracias al Teorema 5.1 que asegura que los argumentos representativos son únicos. Por lo tanto, a lo largo de esta Tesis será usual que al hacer referencia a un argumento representativo sólo se haga mención a su conjunto de reglas y hechos tipados.

5.4. Computando Aceptabilidad en T-DeLP

En las secciones anteriores se ha introducido cómo es la especificación de conocimiento en T-DeLP y cómo a partir de este conocimiento se construyen argumentos tipados representativos. Los argumentos representativos, como se mostró en la sección anterior, constituyen individualmente inferencias consistentes para la conclusión que sustentan. Al igual que en DeLP y en los sistemas argumentativos en general, en T-DeLP pueden existir argumentos en conflicto. Es por esto que para determinar qué argumentos prevalecen ante estos conflictos se realizará un análisis de aceptabilidad.

Para determinar qué argumentos representativos son aceptables en T-DeLP se construirá un Marco Argumentativo de Múltiples Tipos (MATM), considerando los todos los

argumentos representativos y sus relaciones. Por lo tanto, como se vio en el Capítulo 4, este marco argumentativo resultante proveerá el soporte necesario para computar cuáles de estos argumentos representativos son aceptables.

Existen diversos beneficios que justifican la construcción de un MATM para computar aceptabilidad en T-DeLP, en lugar de calcularla directamente (como hacen la gran mayoría de las versiones de DeLP). En primer lugar, como se vio en el Capítulo 4, los MATMs son marcos generales y abstractos a los cuales es posible aplicar cualquier tipo de semántica de aceptabilidad de las presentadas en la literatura de marcos argumentativos [Dun95, DKT06]. Si se siguiera una aproximación directa, sólo se proveería de una única semántica de aceptabilidad. En segundo lugar, el formalismo de los MATMs provee un mecanismo suficientemente sofisticado como para considerar el concepto de tipo de argumento, especialmente al momento de determinar derrotas entre argumentos tipados.

Como se vio en el Capítulo 4 los MATMs proveen un marco para determinar la aceptabilidad de sus argumentos tipados al establecer una relación de derrota entre ellos. Esta relación se construye a partir de los conflictos y preferencias individuales entre argumentos identificadas para cada tipo de argumento, en conjunto con las relaciones de ataque, herencia y preferencia entre los *tipos* de los argumentos. Por lo tanto, si se busca construir un MATM para computar la aceptabilidad de los argumentos representativos de un programa T-DeLP, será necesario identificar cómo se determinar estos elementos a partir del programa. Más concretamente, será necesario caracterizar:

1. cada uno de los tipos individualmente para crear los Marcos Argumentativos de Tipo Simple (MATS), lo cual implica:
   a) identificar todos los argumentos de un tipo,
   b) identificar los conflictos entre estos argumentos, y
   c) determinar las preferencia entre estos argumentos;
2. la relación de herencia entre los tipos;
3. la relación de preferencia entre los tipos; y
4. los conflictos entre argumentos de distintos tipos, para determinar la función de ataques entre tipos.

Hasta el momento, partiendo de un programa T-DeLP $\mathcal{P}^{\Gamma} = (\Gamma, \Theta, \Delta)$ es posible obtener relaciones de herencia y preferencia entre tipos de argumento a partir de Γ, y es posible construir los argumentos representativos identificando los tipos a los cuales pertenecen. Por lo tanto, aun resta identificar los conflictos entre argumentos de un mismo tipo y entre argumentos de distinto tipo, y las preferencias entre argumentos de un mismo tipo.

A continuación, en la siguientes subsecciones, se mostrará cómo se determinan los conflictos o *ataques* en T-DeLP, así también como un mecanismo modular para establecer las preferencias entre argumentos de un mismo tipo. Seguidamente se especificará cómo construir los marcos argumentativos para representar los tipos individualmente, para posteriormente integrarlos al contexto completo del MATM.

5.4.1. Ataques entre Argumentos Representativos

Como se vio en las secciones anteriores, a partir de un programa T-DeLP es posible realizar derivaciones que llevan a conflictos. De esta manera, dado que los argumentos representativos se construyen a partir de derivaciones, es posible que ciertos argumentos estén en conflicto.

A nivel de argumentos los conflictos son usualmente conocidos como ataques [PV02]. Como se vio en los capítulos anteriores, es esencial determinar los ataques entre argumentos ya que si dos argumentos se atacan entre sí no pueden ser aceptados en simultáneo. En T-DeLP un ataque de un argumento representativo $\mathcal{A}$ a otro $\mathcal{B}$ se dará cuando $\mathcal{B}$ tenga un subargumento cuya conclusión este en desacuerdo con la conclusión de $\mathcal{A}$.

Definición 5.19 (Ataque) *Sean $\langle \mathcal{A}, L^T \rangle$ y $\langle \mathcal{B}, M^U \rangle$ argumentos representativos de un programa T-DeLP $\mathcal{P}^{\Gamma} = (\Gamma, \Theta, \Delta)$. Se dice que el argumento $\langle \mathcal{A}, L^T \rangle$ ataca al argumento $\langle \mathcal{B}, M^U \rangle$ si y solo si existe $\langle \mathcal{C}, N^V \rangle$ un subargumento representativo de $\langle \mathcal{B}, M^U \rangle$ tal que $L^T \otimes N^V$.*

Observe que el desacuerdo que genera el ataque sigue las intuiciones presentadas en la sección 5.3.2. Es decir, este desacuerdo se produce entre los literales tipados más especializados posibles ya que, al ser las conclusiones de argumentos representativos, provienen de derivaciones más específicas.

Ejemplo 5.23 *Considere los siguientes argumentos representativos construidos a partir del Ejemplo 5.11:*

- $\langle \{inversion(bB)^W, (comprar(bB) -\prec inversion(bB))\}, comprar(bB)^W \rangle$
- $\langle \{\sim crimen(bA)^A, (seguro(bA)^W -\prec \sim crimen(bA)^W), (comprar(bA) -\prec seguro(bA))\}, compar(bA)^A \rangle$

Observe que el primer argumento ataca al segundo (y viceversa) ya que $compar(bA)^A$ *y* $comprar(bB)^W$ *están en desacuerdo.*

Otra cuestión a notar con respecto a la Definición 5.19 es que, como se puede ver en el ejemplo anterior, el argumento para $comprar(bA)^A$ ataca al argumento para $comprar(bB)^W$ directamente (el subargumento atacado es el argumento en sí mismo). Por lo tanto, es posible identificar dos situaciones de ataque entre argumentos: el *ataque directo* cuando un argumento ataca a otro directamente a su conclusión, y el *ataque interno* que ocurre cuando un argumento ataca a un subargumento propio del argumento atacado.

Una propiedad importante que cumplen los ataques en T-DeLP es que si un argumento representativo $\mathcal{A}$ ataca a otro argumento representativo $\mathcal{B}$, siempre existirá un argumento representativo $\mathcal{C}$ que ataque a $\mathcal{A}$. Esto se da porque la relación de desacuerdo es simétrica. Por lo tanto, si un argumento ataca directamente a otro, este último atacará al primero también. Ahora bien, si se trata de un ataque interno, el argumento que es subargumento del argumento atacado también atacará al atacante.

Proposición 5.10 *Sean* $\langle \mathcal{A}, L^T \rangle$ *y* $\langle \mathcal{B}, M^U \rangle$ *argumentos representativos de un programa T-DeLP* $\mathcal{P}^\Gamma = (\Gamma, \Theta, \Delta)$*. Si* $\langle \mathcal{A}, L^T \rangle$ *ataca a* $\langle \mathcal{B}, M^U \rangle$*, entonces existirá* $\langle \mathcal{C}, N^V \rangle$ *un argumento representativo en* $\mathcal{P}^\Gamma$ *tal que* $\langle \mathcal{C}, N^V \rangle$ *ataca directamente a* $\langle \mathcal{A}, L^T \rangle$*.*

Prueba : *Si* $\langle \mathcal{A}, L^T \rangle$ *ataca directamente a* $\langle \mathcal{B}, M^U \rangle$ *entonces* $L^T \otimes M^U$ *Por lo tanto,* $\langle \mathcal{B}, M^U \rangle$ *ataca directamente a* $\langle \mathcal{A}, L^T \rangle$*. Si* $\langle \mathcal{A}, L^T \rangle$ *ataca internamente a* $\langle \mathcal{B}, M^U \rangle$ *entonces, por Definición 5.19 de ataque, existe un argumento representativo* $\langle \mathcal{C}, N^V \rangle$ *en* $\mathcal{P}^\Gamma$ *tal que es un subargumento de* $\langle \mathcal{B}, M^U \rangle$ *y* $L^T \otimes N^V$*. Luego,* $\langle \mathcal{C}, N^V \rangle$ *ataca directamente a* $\langle \mathcal{A}, L^T \rangle$*.* □

Otra propiedad que cumplen los ataques en T-DeLP es que si existe un argumento básico (argumentos que sólo utilizan hechos) representativo, entonces no existirá ningún argumento que lo ataque. Esto se debe a que este argumento está formado únicamente por conocimiento estricto y este conocimiento está libre de conflictos.

Proposición 5.11 *Sea un programa T-DeLP* $\mathcal{P}^\Gamma = (\Gamma, \Theta, \Delta)$ *y* $\langle \mathcal{A}, L^T \rangle$ *un argumento representativo de* $\mathcal{P}^\Gamma$*. Si* $\mathcal{A}$ *es tal que* $\mathcal{A} \cap \Delta = \emptyset$*, entonces no existe un argumento representativo* $\langle \mathcal{B}, M^U \rangle$ *de* $\mathcal{P}^\Gamma$ *tal que* $\langle \mathcal{B}, M^U \rangle$ *ataca a* $\langle \mathcal{A}, L^T \rangle$*.*

Prueba : Suponga que existe un argumento representativo $\langle \mathcal{B}, M^U \rangle$ *tal que ataca a* $\langle \mathcal{A}, L^T \rangle$*. Como* $\mathcal{A} \cap \Delta = \emptyset$ *entonces el único subargumento de* $\langle \mathcal{A}, L^T \rangle$ *es* $\langle \mathcal{A}, L^T \rangle$*. Entonces, por Definición 5.19 de ataque,* $L^T \otimes M^U$*. Luego como* $\langle \mathcal{B}, M^U \rangle$ *es un argumento construible a partir de* $\mathcal{P}^\Gamma$*, entonces* $(\mathcal{B} \cap \Delta)$ *es un conjunto no conflictivo respecto a* $\mathcal{P}^\Gamma$*. Por lo tanto,* $L^T \notin \Theta$*, lo cual contradice la hipótesis.* □

5.4.2. Preferencias entre Argumentos Tipados

En la Sección 5.2.2 se mostró que un programa T-DeLP cuenta con una relación de preferencia entre sus tipos. Por lo tanto, solo resta identificar las preferencias entre los argumentos de un mismo tipo. Para esto T-DeLP, de manera similar a DeLP, se valdrá de un conjunto de criterios de comparación externos.

En el Capítulo 3 se mostró que DeLP utiliza un criterio de comparación para determinar si un argumento era preferido a otro. En T-DeLP se seguirá una aproximación similar. La principal diferencia radica en que T-DeLP contará con un conjunto de criterios de comparación, cada uno relacionado con un tipo del programa.

Dado que, al igual que en DeLP, estos criterios son modulares y modificables, serán manejados de manera abstracta a través de la función de comparación de argumentos tipados. Esta función recibirá dos argumentos representativos y un tipo, y devolverá el argumento preferido entre ambos bajo ese tipo.

Definición 5.20 (Función de comparación de argumentos tipados) *Sea* $\mathcal{P}^\Gamma = (\Gamma, \Theta, \Delta)$ *un programa T-DeLP y* AR *el conjunto de todos los argumentos representativos de* $\mathcal{P}^\Gamma$*. La función de comparación de argumentos tipados* $\mathsf{FComp} : \mathsf{FTipos}(\Gamma) \times \mathsf{AR} \times \mathsf{AR} \rightarrow \mathsf{AR}$ *es tal que si* $\mathcal{A}, \mathcal{B} \in \mathsf{AR}$ *y* $T \in \mathsf{FTipos}(\Gamma)$*, entonces:*

$$\mathsf{FComp}(T, \mathcal{A}, \mathcal{B}) = \begin{cases} \mathcal{A} & \text{si } \mathcal{A} \text{ es preferido a } \mathcal{B} \text{ en } T \\ \mathcal{B} & \text{si } \mathcal{B} \text{ es preferido a } \mathcal{A} \text{ en } T \\ \emptyset & \text{en otro caso} \end{cases}$$

Note que en caso que esta función retorne $\emptyset$ pueden ocurrir diversas situaciones. Por ejemplo, puede implicar que los argumentos son incomparables, que tienen igual fuerza o que no haya preferencias establecidas. También observe que FComp (T,$\mathcal{A}$,$\mathcal{B}$) = FComp (T,$\mathcal{B}$,$\mathcal{A}$).

En particular, cuando sea necesario para los ejemplos, en el resto del capitulo se utilizara como FComp el criterio de especificad generalizada de DeLP. El criterio se comporta de la misma manera que el criterio presentado en el Capitulo 2 para DeLP, solamente que ignora los tipos de los literales tipados involucrados.

Habiendo presentado cómo se construyen los argumentos representativos, cómo se determinan sus ataques, sus preferencias y las preferencias entre sus tipos, es posible proceder a mostrar cómo se determina cuáles de ellos serán aceptables. Como se verá en la siguiente sección, todos estos elementos permitirán caracterizar un MATM, el cual proveerá los mecanismos adecuados para identificar los argumentos aceptables.

5.4.3. Marcos Argumentativos para los Tipos Simples

En el Capítulo 4 se mostró que los MATMs están constituidos por un conjunto de Marcos Argumentativos de Tipo Simple (MATS). Cada uno de estos MATSs representa un tipo individual dentro del MATM. Por lo tanto, para construir el MATM que representa un programa T-DeLP, en primer lugar será necesario identificar los MATSs que surgen de los argumentos representativos y tipos del programa T-DeLP.

Como se vio en el capítulo anterior los MATSs están caracterizados por un conjunto de argumentos y dos relaciones entre ellos: una relación de ataque y una relación de preferencia. Intuitivamente, cada tipo de un programa T-DeLP constituirá un MATS. En consecuencia, si un argumento representativo es de un tipo T, ese argumento pertenecerá al MATS que representa a ese tipo. Por ejemplo, considere los argumentos representativos para el programa T-DeLP del Ejemplo 5.11 ilustrados en la Figura 5.2. El argumento $\langle \mathcal{A}_3, comprar(bA)^A \rangle$, con $\mathsf{FTA}(\langle \mathcal{A}_3, comprar(bA)^A \rangle) = \{A, W, \tau\}$ pertenecerá a los MATS de A, W y τ, mientras que el argumento $\langle \mathcal{A}_6, \sim comprar(bA)^N \rangle$ con $\mathsf{FTA}(\langle \mathcal{A}_6, \sim comprar(bA)^N \rangle) = \{N, \tau\}$ pertenecerá a los MATS de N y τ. Luego las relaciones de ataque y preferencia internas a cada MATS se determinarán a partir de los ataques entre sus argumentos y la función de comparación de argumentos tipados.

Definición 5.21 (Marco Argumentativo de un Tipo T-DeLP)
Sea $\mathcal{P}^{\Gamma} = (\Gamma, \Theta, \Delta)$ *y* $T \in \mathsf{FTipos}(\Gamma)$. *El MATS* MA_T *será un Marco Argumentativo de tipo* T *a partir del programa* $\mathcal{P}^{\Gamma}$ *si y solo si es tal que verifica las siguientes condiciones:*

- *si existe un argumento representativo* $\mathcal{A}$ *de* $\mathcal{P}^{\Gamma}$ *con* $T \in \mathsf{FTA}(\mathcal{A})$, *entonces* $\mathcal{A} \in \mathsf{Args}(MA_T)$;
- *si* $\mathcal{B}, \mathcal{C} \in \mathsf{Args}(MA_T)$ *y* $\mathcal{B}$ *ataca a* $\mathcal{C}$ *en* $\mathcal{P}^{\Gamma}$ *entonces* $(\mathcal{B}, \mathcal{C}) \in \mathsf{Atts}(MA_T)$; *y*
- *si* $\mathcal{D}, \mathcal{E} \in \mathsf{Args}(MA_T)$ *y* $\mathsf{FComp}(T, \mathcal{D}, \mathcal{E}) = \mathcal{D}$ *entonces* $(\mathcal{D}, \mathcal{E}) \in \mathsf{Prefs}(MA_T)$.

El conjunto de los MATS de todos los tipos de un programa T-DeLP $\mathcal{P}^{\Gamma}$ será llamado $CMA_{\mathcal{P}^{\Gamma}}$. Note que este conjunto tendrá tantos elementos como elementos haya en $\mathsf{FTipos}\ (\Gamma)$.

Ejemplo 5.24 *Considere los siguientes argumentos representativos del programa T-DeLP* $\mathcal{P}^{\Gamma}$ *del Ejemplo 5.11, los cuales se ilustran en el Ejemplo 5.21. En la Figura 5.3 se ilustran los MATSs resultantes de* $\mathcal{P}^{\Gamma}$, *destacando los argumentos y ataques en cada MATS. En particular, los rectángulos con las letras A, W, N y* τ *representan los MATS, los triángulos etiquetados representan los argumentos, y las flechas punteadas dobles representan que el ataque es bidireccional.*

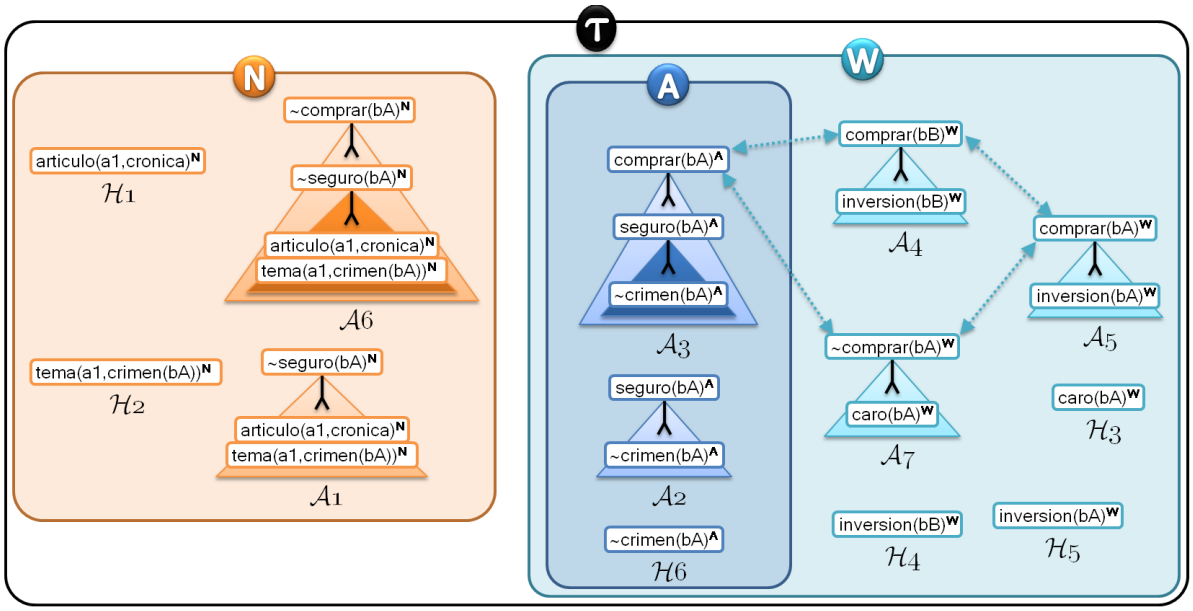


Figura 5.3: MATSs para los argumentos representativos del programa del Ejemplo 5.11.

Note que en la Figura 5.3 se identifican cuatro MATSs, uno para cada tipo de $\mathcal{P}^{\Gamma}$*. Además, observe que los argumentos pueden pertenecer a más de un MATS, por ejemplo, el argumento* $\langle \mathcal{A}_3, comprar(bA)^A \rangle$ *pertenece a los MATS A, W y* τ*. Esto puede visualizarse en la Figura 5.3 dado que los MATSs a los que pertenece el argumento se encuentran superpuestos en ese lugar. Más adelante, cuando se presente el MATM para este programa, se verá que esta superposición modela la relación de herencia entre los tipos. Adicionalmente, como se puede ver en la figura, sólo hay ataques internos entre los argumentos del MATS* W*. Finalmente, note que no hay preferencias internas establecidas entre los argumentos de ningún MATS.*

5.4.4. MATM a partir de T-DeLP

Dado un conjunto de MATS, la relación de herencia y las preferencias entre los tipos, para poder construir el MATM que representa un programa T-DeLP sólo resta determinar los ataques entre argumentos de distintos tipos. Para esto, al igual que con los ataques internos de los MATSs, se utilizará el concepto de ataque entre argumentos representativos, considerando únicamente aquellos ataques entre argumentos de diferentes tipos.

Definición 5.22 (Función de ataque entre argumentos de distinto tipo) *Sea* $\mathcal{P}^\Gamma = (\Theta, \Delta, \Gamma)$ *un programa T-DeLP, y* AR *el conjunto de sus argumentos representativos. La función de ataque entre argumentos de distinto tipo para* $\mathcal{P}^\Gamma$*, definida como* $\mathsf{TAtt}_{\mathcal{P}^\Gamma} : \mathsf{FTipos}(\Gamma) \times \mathsf{FTipos}(\Gamma) \rightarrow 2^{\mathsf{AR}\times\mathsf{AR}}$ *es tal que* $(\mathcal{B}, \mathcal{C}) \in \mathsf{TAtt}_{\mathcal{P}^\Gamma}(T_1, T_2)$ *si y solo si* $T_1 \neq T_2$, $\mathcal{B}, \mathcal{C} \in \mathsf{AR}$, $T_1 \in \mathsf{FTA}(\mathcal{B})$, $T_2 \in \mathsf{FTA}(\mathcal{C})$, *y* $\mathcal{B}$ *ataca a* $\mathcal{C}$ *en* $\mathcal{P}^\Gamma$.

Ejemplo 5.25 *Considere los MATSs presentados en el Ejemplo 5.24 para el programa T-DeLP* $\mathcal{P}^\Gamma$ *del Ejemplo 5.11. La función de ataque entre argumentos de distinto tipo para* $\mathcal{P}^\Gamma$ *se ilustra a continuación en la Figura 5.4 mediante el uso de flechas a trazos.*

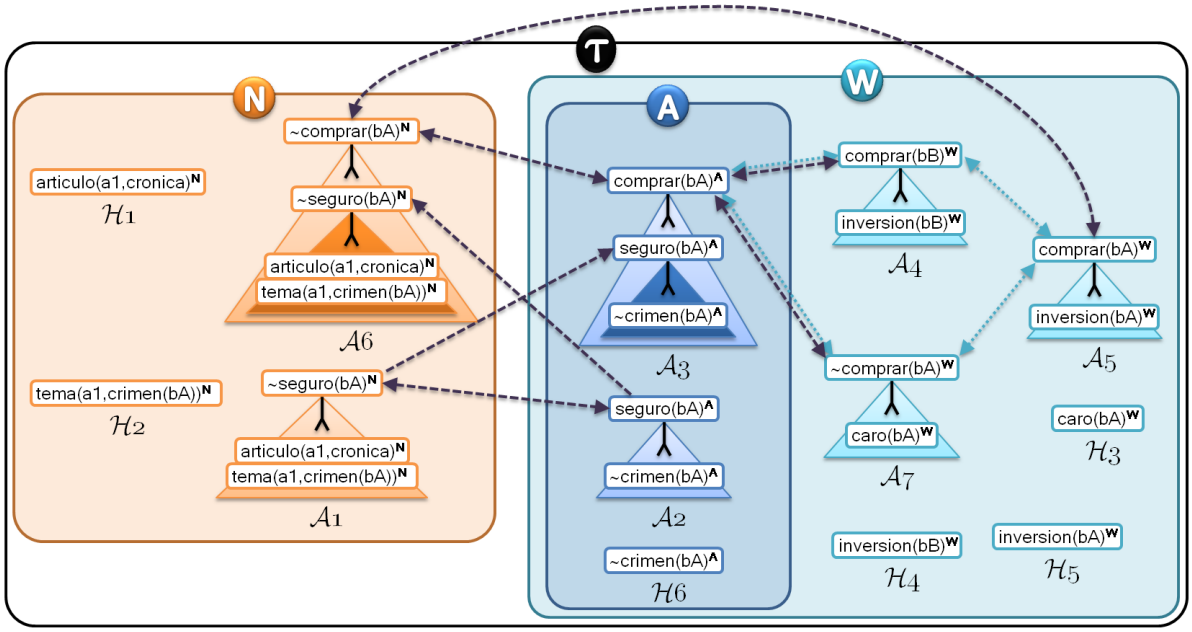


Figura 5.4: Función de ataque entre argumentos de distinto tipo para el programa del Ejemplo 5.11.

Note que no todos los ataques son bidireccionales, por ejemplo, puede verse que tanto el ataque de $\mathcal{A}_1$ *a* $\mathcal{A}_3$ *como el ataque de* $\mathcal{A}_2$ *a* $\mathcal{A}_6$ *son ataques internos. Adicionalmente, note que existen ataques que ocurren tanto internamente en un MATS como a través de la función de ataque entre argumentos de distintos tipos. Por ejemplo, el ataque entre* $\mathcal{A}_3$ *y* $\mathcal{A}_4$ *ocurre en el MATS para el tipo* W*, así como también a través de la función de ataque entre argumentos de distintos tipos, dado que* $\mathcal{A}_3$ *es también de tipo* A *y* $\mathcal{A}_4$ *es de tipo* W.

Observe que la función presentada en la Definición 5.22 es una función de ataque entre argumentos de distinto tipo en el contexto de los MATM (ver Sección4.3.2 del Capítulo 4). Por lo tanto, utilizando esta última definición será posible presentar formalmente cómo es el MATM utilizado para computar la aceptabilidad de los argumentos representativos de un programa T-DeLP.

Definición 5.23 (MATM representativo) *Sea* $\mathcal{P}^\Gamma = (\Gamma, \Theta, \Delta)$ *un programa T-DeLP,* $\mathsf{TAtt}_{\mathcal{P}^\Gamma}$ *la función de ataque entre argumentos de distinto tipo para* $\mathcal{P}^\Gamma$*, y* $CMA_{\mathcal{P}^\Gamma}$ *conjunto de los MATSs para los tipos de* $\mathcal{P}^\Gamma$*. El MATM representativo para* $\mathcal{P}^\Gamma$*, notado* $MTAM_{\mathcal{P}^\Gamma}$ *es tal que* $MATM_{\mathcal{P}^\Gamma} = (CMA_{\mathcal{P}^\Gamma}, \mathsf{TAtt}_{\mathcal{P}^\Gamma}, \mathsf{FHerencia}(\Gamma), \mathsf{FPref}(\Gamma))$.

Ejemplo 5.26 *Considere el MATM representativo* $M_{\mathcal{P}^\Gamma}$ *para el programa* $\mathcal{P}^\Gamma$ *del Ejemplo 5.11.* $M_{\mathcal{P}^\Gamma} = (CMA_{\mathcal{P}^\Gamma}, \mathsf{TAtt}_{\mathcal{P}^\Gamma}, \mathsf{FHerencia}(\Gamma), \mathsf{FPref}(\Gamma))$*, donde* $CMA_{\mathcal{P}^\Gamma}$ *es el conjunto de los MATSs para los tipos A, W, N y* τ *presentados en el Ejemplo 5.24,* $\mathsf{TAtt}_{\mathcal{P}^\Gamma}$ *es la función ilustrada en el la Figura 5.4 del Ejemplo 5.25,* $\mathsf{FHerencia}(\Gamma)$ *es tal que* $A \rhd W$*,* $W \rhd \tau$ *y* $N \rhd \tau$*, y* $\mathsf{FPref}(\Gamma))$ *es tal que* $N >> W$*,* $A >> N$ *y* $A >> W$*. Note que la Figura 5.4 captura todos los componentes de* $M_{\mathcal{P}^\Gamma}$*, a excepción de las preferencias entre los tipos. En particular, observe que la relación de herencia es capturada en tal figura mediante de la superposición de los rectángulos que representan a los MATSs para los diferentes tipos.*

Como se vio en el capítulo anterior, a partir de un MATM es posible especificar un Marco Argumentativo clásico [Dun95], en cual es posible aplicar cualquiera de las semánticas de aceptabilidad presentadas en la literatura de argumentación. Con tal objetivo, se mostró que es necesario construir una la relación de derrota entre los argumentos. Esta relación busca capturar aquellos ataques que son efectivos, con lo cual un argumento derrotará a otro si lo ataca y es preferido ante el argumento atacado. En particular, recuerde que la preferencia entre dos argumentos es establecida analizando las preferencias indicadas en los MATSs a los que pertenecen estos argumentos (*i.e.* preferencia interna), y las preferencias entre los tipos de los argumentos (*i.e.* preferencia externa).

En los MATMs que se presentaron en el capítulo anterior se asume que los argumentos son entidades abstractas, libres de estructura interna. En un MATM que se construye a partir de un programa T-DeLP, los argumentos tendrán una estructura interna, la cual repercutirá al momento de determinar las derrotas entre argumentos.

Ejemplo 5.27 *Considere un programa T-DeLP donde:*

$$\Theta = \left\{ \begin{array}{c} c \\ d \end{array} \right\} \qquad \Delta = \left\{ \begin{array}{l} a^{t1} -\!\prec b^{t3} \\ b^{t3} -\!\prec c \\ \sim b^{t2} -\!\prec d \end{array} \right\}$$

Adicionalmente, la relación de herencia es sólo la obtenida por defecto (t1,t2 y t3 heredan del tipo base τ), la relación de preferencia entre tipos es tal que t3 >> t2 >> t1, y $b^{t3} \otimes \sim b^{t2}$. Note que a partir de este programa es posible construir los siguientes argumentos representativos:

$$\langle \mathcal{B}_1, a^{t1} \rangle = \langle \{(a^{t1} -\!\prec b^{t3}), (b^{t3} -\!\prec c), c\}, a^{t1} \rangle$$
$$\langle \mathcal{B}_2, b^{t3} \rangle = \langle \{(b^{t3} -\!\prec c), c\}, b^{t3} \rangle$$
$$\langle \mathcal{B}_3, \sim b^{t2} \rangle = \langle \{(\sim b^{t2} -\!\prec d), d\}, \sim b^{t2} \rangle$$

Observe que el argumento $\langle \mathcal{B}_3, \sim b^{t2} \rangle$ ataca a $\langle \mathcal{B}_1, a^{t1} \rangle$ en el subargumento $\langle \mathcal{B}_2, b^{t3} \rangle$, ya que $b^{t3} \otimes \sim b^{t2}$. En consecuencia, ese ataque estará en el MATM representativo para este programa. Además, siguiendo la noción de preferencia de tipos para el MATM, observe que $\langle \mathcal{B}_3, \sim b^{t2} \rangle$ es externamente preferido a $\langle \mathcal{B}_1, a^{t1} \rangle$ porque t2>>t1. Luego se tiene que $\langle \mathcal{B}_3, \sim b^{t2} \rangle$ derrota a $\langle \mathcal{B}_1, a^{t1} \rangle$.

Note que la derrota es mostrada en el Ejemplo 5.27 es contra-intuitiva ya que, si bien $t2>>t1$, el ataque de $\langle \mathcal{B}_3, \sim b^{t2} \rangle$ a $\langle \mathcal{B}_1, a^{t1} \rangle$ se produce en $\langle \mathcal{B}_2, b^{t3} \rangle$ que es de tipo $t3$, el cual es preferido ante $t2$. Como se puede observar, la situación indeseada se presenta debido a que la derrota se establece a partir de la preferencia entre el argumento atacante y el argumento atacado, en lugar de considerar el argumento atacante y el subargumento del argumento atacado. Por lo tanto, para estos MATM se presentará una versión diferente de derrota, la cual considera las preferencias entre el argumento atacante y el subargumento del argumento atacado.

Definición 5.24 (Derrota MATM representativo) *Sea $MATM_{\mathcal{P}^\Gamma} = (CMA_{\mathcal{P}^\Gamma}$, $\mathsf{TAtt}_{\mathcal{P}^\Gamma}$, $\mathsf{FHerencia}(\Gamma)$, $\mathsf{FPref}(\Gamma))$ el MATM representativo de un programa T-DeLP $\mathcal{P}^\Gamma$. Un argumento representativo $\mathcal{A}$ derrota a otro argumento representativo $\mathcal{B}$, notado $\mathcal{A} \rightarrow_{\mathcal{P}^\Gamma} \mathcal{B}$, si y solo si $\mathcal{A} \rightharpoonup \mathcal{B}$, con $\mathcal{A}$ atacando a $\mathcal{B}$ en un subargumento representativo $\mathcal{C}$, y $\mathcal{A} >_{MT} \mathcal{C}$ o bien $\mathcal{A} \not>_{MT} \mathcal{C}$ y $\mathcal{C} \not>_{MT} \mathcal{A}$*[4].

[4]Las relaciones $\rightharpoonup$ y $>_{MT}$ de ataque y preferencia global respectivamente, son aquellas definidas para los MATMs en el Capítulo 4 de esta tesis.

Ejemplo 5.28 *Considere el MATM representativo $M_{\mathcal{P}^\Gamma}$ para el programa T-DeLP $\mathcal{P}^\Gamma$ del Ejemplo 5.11, el cual fue presentado en el Ejemplo 5.26. La Figura 5.5 a continuación ilustra mediante flechas gruesas las derrotas que surgen entre los argumentos representativos de $\mathcal{P}^\Gamma$.*

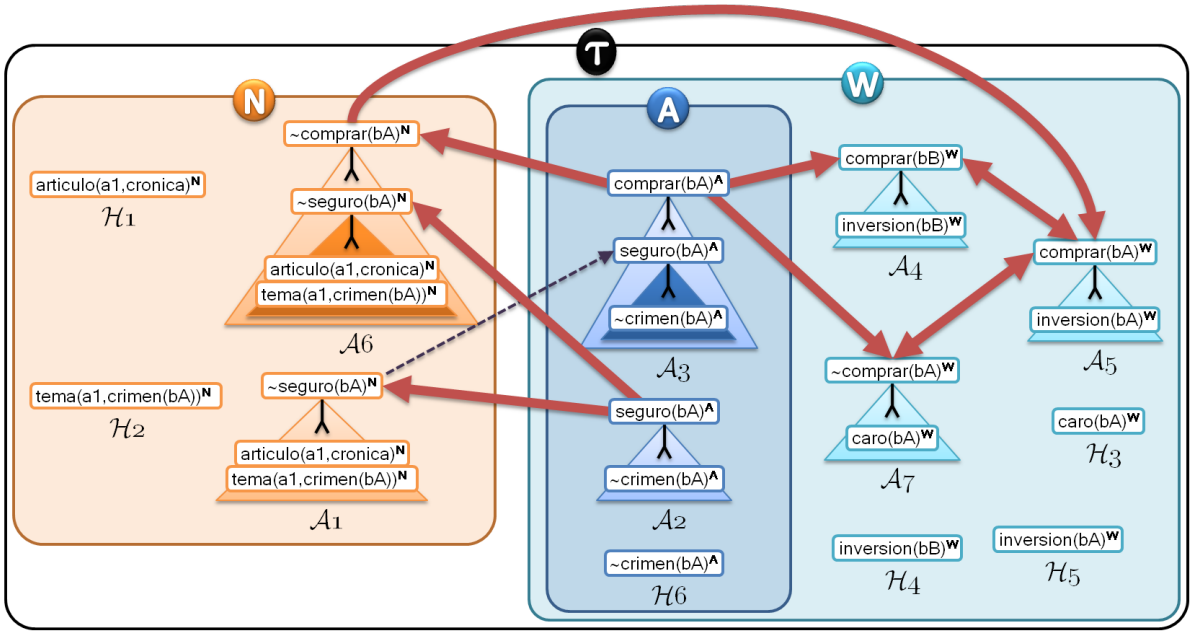


Figura 5.5: Derrotas para programa T-DeLP del Ejemplo 5.11.

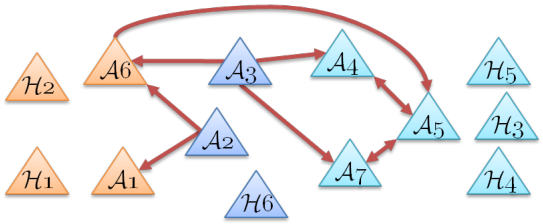
Utilizando esta noción de derrota en conjunto con los argumentos de estos MATMs será posible aplicar las semánticas de aceptabilidad presentadas en los capítulos 3 y 4. En particular, como se mostró en el capítulo anterior, para todo MATM existe un marco argumentativo abstracto [Dun95] asociado, sobre el cual es posible aplicar las semánticas de aceptabilidad. Por lo tanto, si un argumento es aceptado bajo una semántica en el marco abstracto asociado, también será aceptado en el MATM correspondiente, y por lo tanto en el programa T-DeLP respectivo, bajo esa semántica.

Definición 5.25 (Argumento representativo aceptado) *Sea $\mathcal{P}^\Gamma = (\Gamma, \Theta, \Delta)$ un programa T-DeLP, $MATM_{\mathcal{P}^\Gamma}$=$(CMA_{\mathcal{P}^\Gamma}$, $\mathsf{TAtt}_{\mathcal{P}^\Gamma}$, $\mathsf{FHerencia}(\Gamma)$, $\mathsf{FPref}(\Gamma))$ el MATM representativo de $\mathcal{P}^\Gamma$ y S una semántica de aceptabilidad. El argumento $\mathcal{A}$ es:*

- *escépticamente aceptado en $\mathcal{P}^\Gamma$ bajo S si y solo si $\mathcal{A}$ pertenece a toda extensión de $(\mathsf{MArgs}(MATM_{\mathcal{P}^\Gamma}), \rightarrow_{\mathcal{P}^\Gamma})$;*

- *crédulamente aceptado en* $\mathcal{P}^\Gamma$ *bajo* S *si y solo si* $\mathcal{A}$ *pertenece a alguna extensión de* $(\mathsf{MArgs}(MATM_{\mathcal{P}^\Gamma}), \rightarrow_{\mathcal{P}^\Gamma})$*; o*
- *rechazado en* $\mathcal{P}^\Gamma$ *bajo* S *si y solo si* $\mathcal{A}$ *no pertenece a ninguna extensión de* $(\mathsf{MArgs}(MATM_{\mathcal{P}^\Gamma}), \rightarrow_{\mathcal{P}^\Gamma})$.

Ejemplo 5.29 *Continuando con el escenario presentado para el programa T-DeLP* $\mathcal{P}^\Gamma$ *del Ejemplo 5.11, su MATM representativo y las derrotas entre sus argumentos, se obtiene el siguiente marco argumentativo* $(\mathsf{MArgs}(M_{\mathcal{P}^\Gamma}), \rightarrow_{\mathcal{P}^\Gamma})$*, el cual es un resumen de la Figura 5.5:*



Considere la semántica grounded presentada en el Capítulo 3. Se puede observar que, a partir de este marco argumentativo, los argumentos que pertenecen a la extensión grounded son $\{\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4, \mathcal{H}_5, \mathcal{H}_6, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_5\}$*. Estos serán entonces los argumentos escépticamente (y crédulamente, dado que hay una única extensión) aceptados. Los argumentos rechazados bajo esta semántica son* $\{\mathcal{A}_1, \mathcal{A}_4, \mathcal{A}_6, \mathcal{A}_7\}$.

Dados los argumentos aceptados será posible determinar finalmente qué literales tipados se pueden inferir a partir de un programa T-DeLP. Básicamente, si un argumento está aceptado, entonces su conclusión será un literal tipado inferido a partir del programa. Claramente, en tal sentido sólo se utilizarán los argumentos escépticamente aceptados ya que, si se considerasen los crédulos, sería posible inferir literales en desacuerdo.

Definición 5.26 (Literal Tipado Inferido) *Sea* $\mathcal{P}^\Gamma$ *un programa T-DeLP,* L^T *un literal tipado y* S *una semántica de aceptabilidad. El programa* $\mathcal{P}^\Gamma$ *infiere* L^T *bajo* S*, notado* $\mathcal{P}^\Gamma \vdash_S L^T$*, si y solo si existe un argumento representativo* $\langle \mathcal{A}, L^T \rangle$ *tal que es escépticamente aceptado en* $\mathcal{P}^\Gamma$ *bajo* S*. En otro caso* $\mathcal{P}^\Gamma \nvdash_S L^T$.

Ejemplo 5.30 *Continuando con el Ejemplo 5.29 los literales inferidos serán* $\{articulo(a1, cronica)^N$, $tema(a1, crimen(bA))^N$, $caro(bA)^W$, $inversion(bB)^W$, $inversion(bA)^W$, $\sim crimen(bA)^A$, $seguro(bA)^A$, $comprar(bA)^A$, $comprar(bA)^W$ $\}$.

Este modelo de aceptabilidad para los argumentos de T-DeLP sigue la mecánica extensional. Como se vio en capítulos anteriores, esta mecánica no es siempre la más adecuada debido a su costo computacional, y dado que se aleja de la naturaleza dialéctica de argumentación. En la siguiente sección se mostrará una aproximación dialéctica similar a la de DeLP, la cual se valdrá de varios conceptos introducidos hasta hasta el momento.

5.4.5. Un Proceso de prueba dialéctico para T-DeLP - MATM

En esta sección se mostrará cómo determinar las inferencias de un programa T-DeLP a partir de un procedimiento de prueba dialéctico. Este enfoque, como se mostró en el Capítulo 3, es el que sigue DeLP, y es ampliamente adoptado en la literatura en la caracterización de procedimientos de prueba para la argumentación, destacándose además los trabajos de [DKT06, DMT07, DBC03, VP00]. En general, un procedimiento de prueba dialéctico consiste en la simulación de todas las líneas de discusión entre un proponente y un oponente enfrentados. El proponente comienza proponiendo un argumento $\mathcal{A}$ a determinar si es aceptado, y luego tanto oponente como proponente van presentando sucesivamente argumentos derrotadores en respuesta a los argumentos de su contrincante. Como se vio en el Capítulo 3, estas discusiones generan árboles de argumentos y derrotas. Finalmente, $\mathcal{A}$ quedará aceptado o rechazado luego de analizar la discusión de manera exhaustiva.

En primer lugar se presentará la noción formal de *línea argumentativa* en T-DeLP. Al igual que en DeLP, una línea argumentativa representa una línea de discusión entre el proponente y oponente acerca de un determinado argumento $\mathcal{A}$. Concretamente, es una secuencia de argumentos (comenzando con $\mathcal{A}$) donde cada argumento derrota a su predecesor en la línea, tal que cada argumento que aparece en una posición impar (entre ellos $\mathcal{A}$) se considera como presentado por el proponente (y por lo tanto, a favor de $\mathcal{A}$), y cada argumento en una posición par se considera presentado por el oponente (y por lo tanto, en contra de $\mathcal{A}$).

Definición 5.27 (Línea argumentativa T-DeLP) *Sea $\mathcal{P}^\Gamma$ un programa T-DeLP, $M_{\mathcal{P}^\Gamma}$ su MATM representativo y $\mathcal{A}$ un argumento representativo de $\mathcal{P}^\Gamma$. Una línea argumentativa para $\mathcal{A}$ será una secuencia de argumentos representativos $\lambda = [\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n]$, con $\mathcal{A}_1 = \mathcal{A}$, tal que:*

- *todo* $\mathcal{A}_i \in \mathsf{MArgs}(M_{\mathcal{P}^\Gamma})$, *y*
- $\mathcal{A}_i \rightarrow_{\mathcal{P}^\Gamma} \mathcal{A}_{i-1}$ *para todo* $1 < i \leq n$.

Note que, a diferencia de DeLP, se empleará el MATM representativo de un programa T-DeLP para obtener la relación de derrota que es utilizada para conectar los argumentos de una línea de argumentación. Por ejemplo, considerando los argumentos y derrotas ilustrados en la Figura 5.5, una línea argumentativa podría ser $[\mathcal{A}_5, \mathcal{A}_6, \mathcal{A}_2]$.

Al igual que en DeLP, las líneas argumentativas pueden ser sujeto de falacias, por lo que únicamente se considerarán aquellas líneas catalogadas como *aceptables*. En particular, como en DeLP, básicamente se pueden presentar cuatro situaciones que conllevan a líneas falaces: los argumentos de soporte no son concordantes, los argumentos de interferencia no son concordantes, existe argumentación circular, y existen bloqueos simultáneos. Estos conceptos son similares a los que se vieron para DeLP en Capítulo 3. La concordancia se consigue evaluando si los argumentos de soporte y interferencia independientemente no conducen a conjuntos conflictivos, el doble bloqueo (o bloqueo simultáneo) se evita analizando las derrotas simétricas, y la argumentación circular se evita impidiendo que se reintroduzcan subargumentos representativos como elementos de la línea.

Definición 5.28 (Línea argumentativa aceptable T-DeLP) *Sea* $\mathcal{P}^\Gamma$ *un programa T-DeLP y* $\lambda = [\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n]$ *una línea argumentativa de* $\mathcal{P}^\Gamma$. *Se dirá que* λ *es una línea argumentativa aceptable en* $\mathcal{P}^\Gamma$ *si y solo si:*

- *el conjunto de los argumentos en posiciones impares de la línea es no conflictivo;*
- *el conjunto de los argumentos en posiciones pares de la línea es no conflictivo;*
- *ningún argumento* $\mathcal{A}_i$ *de* λ *es tal que es un subargumento representativo de un argumento* $\mathcal{A}_k$ *que aparece previamente en* λ $(k < i)$; *y*
- *para todo* $\mathcal{A}_i$, *si* $\mathcal{A}_i \rightarrow_{\mathcal{P}^\Gamma} \mathcal{A}_{i-1}$, $\mathcal{A}_{i-1} \rightarrow_{\mathcal{P}^\Gamma} \mathcal{A}_i$ *y existe* $\mathcal{A}_{i+1}$ *en* λ, *entonces* $\mathcal{A}_i \not\rightarrow_{\mathcal{P}^\Gamma} \mathcal{A}_{i+1}$.

Observe que una línea argumentativa aceptable puede no ser exhaustiva. De esta manera, podrían no estar considerándose argumentos representativos que influyen en la evaluación del argumento en discusión. Por lo tanto, en T-DeLP solo se considerarán líneas argumentativas aceptables exhaustivas.

Definición 5.29 (Línea argumentativa aceptable exhaustiva T-DeLP) *Sea* $\mathcal{P}^{\Gamma}$ *programa T-DeLP y* $\lambda = [\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n]$ *una línea argumentativa aceptable en* $\mathcal{P}^{\Gamma}$. *Se dirá que* λ *es una línea argumentativa exhaustiva en* $\mathcal{P}^{\Gamma}$ *si y solo si no existe un argumento representativo* $\mathcal{B}$ *de* $\mathcal{P}^{\Gamma}$ *tal que* $[\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n, \mathcal{B}]$ *es una línea argumenta aceptable en* $\mathcal{P}^{\Gamma}$.

Al igual que en DeLP, para poder determinar si un argumento representativo en T-DeLP prevalece finalmente en este modelo, será necesario analizar todas las líneas argumentativas aceptables y exhaustivas que lo tienen como primer elemento. Es decir, se deberán analizar todos los derrotadores del argumento, los derrotadores de sus derrotadores y así sucesivamente, modelando así una discusión exhaustiva sobre el argumento analizado. La estructura resultante de tal análisis será un árbol de derrotas y argumentos conocido como *árbol de dialéctica*. Estos árboles tiene la característica de estar enraizados en el argumento a analizar, y son tales que cada línea argumentativa para ese argumento corresponde a una rama del árbol.

Definición 5.30 (Árbol de dialéctica T-DeLP) *Sea* $\mathcal{P}^{\Gamma}$ *un programa T-DeLP y* $\mathcal{A}$ *un argumento representativo de* $\mathcal{P}^{\Gamma}$. *Un árbol de dialectica de* $\mathcal{A}$ *en* $\mathcal{P}^{\Gamma}$, *notado como* $\mathcal{T}_{\mathcal{A}}$, *es tal que:*

- $\mathcal{A}$ *es la raíz de* $\mathcal{T}_{\mathcal{A}}$;
- *si* $\lambda = [\mathcal{A}, \ldots, \mathcal{A}_n]$ *es una rama de* $\mathcal{T}_{\mathcal{A}}$ *(camino desde la raiz hasta una hoja), entonces* λ *es una línea argumentativa aceptable exhaustiva para* $\mathcal{A}$ *en* $\mathcal{P}^{\Gamma}$; *y*
- *no existen dos nodos hermanos* $\mathcal{N}$ *y* $\mathcal{N}'$ *en* $\mathcal{T}_{\mathcal{A}}$ *tales que* $\mathcal{N} = \mathcal{N}'$.

Observe, que al igual que en DeLP, la última condición del árbol de dialéctica consiste en tratar de factorizar lo más posible las líneas argumentativas. Es decir, esta condición busca fusionar los segmentos superiores comunes de las distintas líneas argumentativas de un árbol.

Ejemplo 5.31 *Considere el programa T-DeLP presentado el Ejemplo 5.11, sus argumentos representativos ilustrados en la Figura 5.2, y las derrotas entre ellos presentadas en el Ejemplo 5.28. A continuación, en la Figura 5.6, se presenta el árbol de dialéctica para el argumento representativo* $\langle \mathcal{A}_5, comprar(bA)^W \rangle$.

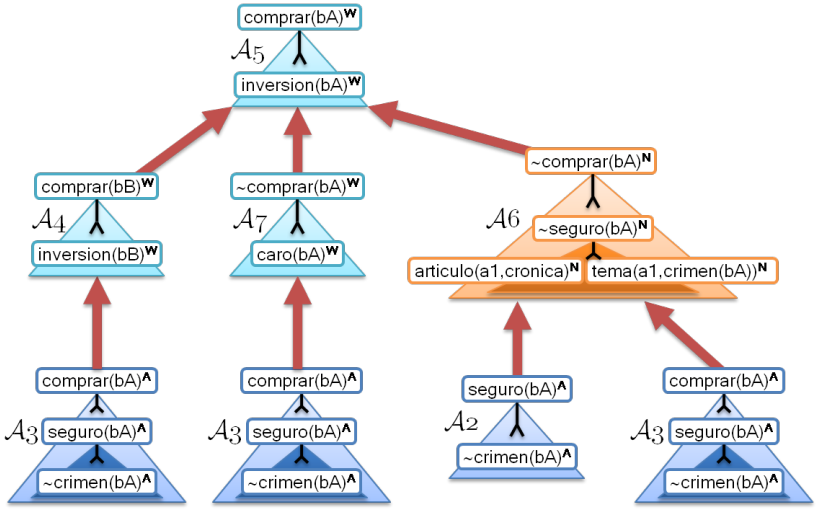


Figura 5.6: Árbol de dialéctica T-DeLP para el argumento $\mathcal{A}_5$ del Ejemplo 5.11.

Note que, si bien el argumento $\mathcal{A}_5$ derrota al argumento $\mathcal{A}_4$, no es posible introducirlo en el árbol ya que sino no se obtendría una línea argumentativa aceptable, dado que $\mathcal{A}_5$ aparece previamente en la línea (es derrotado por $\mathcal{A}_4$).

Finalmente, para evaluar estado de un argumento en T-DeLP se utilizará un árbol de dialéctica marcado. Básicamente, este árbol es un árbol de dialéctica en el que los nodos se van marcando como derrotados y no derrotados, comenzando con las hojas (trivialmente no derrotadas) y propagando el marcado hasta el argumento de la raíz.

Definición 5.31 (Árbol de dialéctica marcado T-DeLP) *Sea $\mathcal{T}_\mathcal{A}$ un árbol de dialéctica para un argumento representativo $\mathcal{A}$ de un programa T-DeLP. El árbol de dialéctica marcado para $\mathcal{A}$, notado $\mathcal{T}^*_\mathcal{A}$, se obtiene marcando cada nodo $\mathcal{N}$ del árbol como* **D** *(derrotado) o* **U** *(no derrotado) de acuerdo al siguiente criterio:*

- *si $\mathcal{N}$ es un nodo hoja, entonces $\mathcal{N}$ es marcado como* **U***;*
- *si $\mathcal{N}$ es un nodo interno y tiene algún hijo marcado como* **U***, entonces $\mathcal{N}$ es marcado como* **D***; o*
- *si $\mathcal{N}$ es un nodo interno y todos sus hijos son marcados como* **D***, entonces $\mathcal{N}$ es marcado como* **U***.*

Siguiendo el modelo de DeLP, en T-DeLP un argumento representativo estará garantizado (*i.e.* aceptado bajo el procedimiento de prueba dialéctico) si y solo si existe un árbol de dialéctica marcado enraizado en ese argumento, tal que la raíz está marcada como **U**.

Definición 5.32 (Argumento garantizado T-DeLP) *Sea $\mathcal{P}^\Gamma$ un programa T-DeLP y $\mathcal{A}$ un argumento representativo de $\mathcal{P}^\Gamma$. Se dirá que $\mathcal{P}^\Gamma$ garantiza a $\mathcal{A}$ si y solo si existe $\mathcal{T}^*_\mathcal{A}$ en $\mathcal{P}^\Gamma$ tal que $\mathcal{A}$ está marcado como* **U** *en $\mathcal{T}^*_\mathcal{A}$.*

Observe que en esta sección se utiliza la noción de garantía para identificar los elementos aceptados a partir del procedimiento de prueba dialéctico. Esta diferencia en la nomenclatura busca distinguir de manera sencilla cuando se habla de aceptabilidad a nivel abstracto utilizando el modelo extensional para las semánticas de aceptabilidad, y cuando se utiliza el modelo de procedimiento de prueba.

Ejemplo 5.32 *Considere el árbol de dialéctica $\mathcal{T}_{\mathcal{A}_5}$ ilustrado en la Figura 5.6 para el argumento representativo $\langle \mathcal{A}_5, comprar(bA)^W \rangle$ del programa T-DeLP $\mathcal{P}^\Gamma$ del Ejemplo 5.11. El árbol de dialéctica marcado resultante para $\mathcal{A}_5$ se presenta en la Figura 5.7 a continuación.*

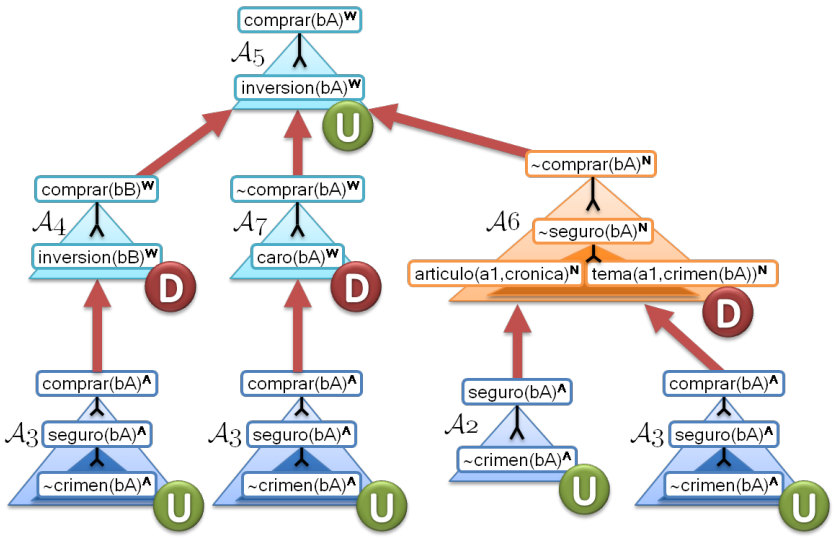


Figura 5.7: Árbol de dialéctica marcado T-DeLP para el argumento $\mathcal{A}_5$ del Ejemplo 5.11.

Observe que, como la marca para la raíz $\langle \mathcal{A}_5, comprar(bA)^W \rangle$ del árbol es **U**, *se tiene que $\langle \mathcal{A}_5, comprar(bA)^W \rangle$ será un argumento garantizado en $\mathcal{P}^\Gamma$.*

Una propiedad fundamental que cumple un programa T-DeLP en cuanto a los argumentos que garantiza, es que no garantizará simultáneamente argumentos conflictivos. Es decir, si T-DeLP garantiza a un argumento, no garantizará a los argumentos que este derrota, ni a los que lo derrotan. Para demostrar formalmente que esta propiedad se cumple en T-DeLP será necesario presentar primero un conjunto de resultados auxiliares.

Proposición 5.12 *Sean $\mathcal{A}$ y $\mathcal{B}$ dos argumentos representativos de un programa T-DeLP $\mathcal{P}^{\Gamma}$. Si $\mathcal{A} \rightarrow_{\mathcal{P}^{\Gamma}} \mathcal{B}$ y $\mathcal{A}$ es un argumento garantizado en $\mathcal{P}^{\Gamma}$, entonces $\mathcal{A}$ está marcado como* **U** *en el árbol de dialéctica marcado $\mathcal{T}^{*}_{\mathcal{B}}$.*

Prueba : Dado que $\mathcal{A}$ está garantizado, entonces $\mathcal{A}$ está marcado con **U** *en el árbol de dialéctica marcado $\mathcal{T}^{*}_{\mathcal{A}}$. Si $\mathcal{A}$ no tiene derrotadores en $\mathcal{T}^{*}_{\mathcal{A}}$, entonces en $\mathcal{T}^{*}_{\mathcal{B}}$ también está marcado como* **U**, *ya que allí tampoco posee derrotadores.*

Considere ahora el caso en que $\mathcal{A}$ posee derrotadores en $\mathcal{T}^{*}_{\mathcal{A}}$. Para que $\mathcal{A}$ no esté marcado como **U** *en $\mathcal{T}^{*}_{\mathcal{B}}$ debe ocurrir que no aparezca en $\mathcal{T}^{*}_{\mathcal{B}}$ uno de los argumentos $\mathcal{C}$ de soporte para $\mathcal{A}$ que está marcado como* **U** *en $\mathcal{T}^{*}_{\mathcal{A}}$. Esto podría suceder por cuatro razones:*

- *el argumento derrotado por $\mathcal{C}$ en $\mathcal{T}^{*}_{\mathcal{A}}$ no aparece en $\mathcal{T}^{*}_{\mathcal{B}}$. De esta manera, si $\mathcal{C}$ no aparece en $\mathcal{T}^{*}_{\mathcal{B}}$, entonces no afecta al marcado de $\mathcal{A}$ en ese árbol;*
- *$\mathcal{C}$ genera un conjunto conflictivo con algún argumento de interferencia en la línea a la que pertenece en $\mathcal{T}^{*}_{\mathcal{B}}$. Esto no puede ocurrir porque de ser así también hubiese ocurrido en $\mathcal{T}^{*}_{\mathcal{A}}$, y por lo tanto, no aparecería en ese árbol. Recuerde que $\mathcal{A}$ es un derrotador de $\mathcal{B}$, por lo tanto, no hay argumentos nuevos en $\mathcal{T}^{*}_{\mathcal{B}}$ con los cuales $\mathcal{C}$ pueda generar un conjunto conflictivo;*
- *$\mathcal{C}$ no aparece $\mathcal{T}^{*}_{\mathcal{B}}$ en por introducir un doble bloqueo. Esto no es posible, ya que de ser así el doble bloqueo se hubiese introducido también en $\mathcal{T}^{*}_{\mathcal{A}}$; o*
- *$\mathcal{C}$ es un subargumento representativo de alguno de los argumentos de la línea argumentativa en $\mathcal{T}^{*}_{\mathcal{B}}$. Este caso puede originarse por dos situaciones:*
  - *$\mathcal{C}$ es subargumento de algún otro argumento de la línea distinto de $\mathcal{B}$, lo cual no es posible porque de ser así $\mathcal{C}$ no podría haber aparecido tampoco en $\mathcal{T}^{*}_{\mathcal{A}}$, ya que sería un subargumento de un argumento de su línea en ese árbol; o*

- *$\mathcal{C}$ es un subargumento de $\mathcal{B}$, y como $\mathcal{C}$ derrota a algún argumento $\mathcal{D}$ en $\mathcal{T}^*_{\mathcal{A}}$, se tiene que $\mathcal{C}$ y $\mathcal{D}$ son un conjunto conflictivo. Por lo tanto, $\mathcal{B}$ y $\mathcal{D}$ serían un conjunto conflictivo (ya que $\mathcal{C} \subseteq \mathcal{B}$). Esto implica que $\mathcal{D}$ no pueda aparecer en $\mathcal{T}^*_{\mathcal{B}}$, con lo cual la marca de $\mathcal{A}$ en $\mathcal{T}^*_{\mathcal{B}}$ no se vería afectada.*

Por lo tanto, en cualquier caso, $\mathcal{A}$ quedará marcado como **U** *en $\mathcal{T}^*_{\mathcal{B}}$.* □

Lema 5.1 *Sea $\mathcal{P}^\Gamma$ un programa T-DeLP y $\mathcal{A}$ un argumento representativo de $\mathcal{P}^\Gamma$. Si $\mathcal{P}^\Gamma$ garantiza a $\mathcal{A}$, entonces no existe un argumento representativo $\mathcal{B}$ tal que $\mathcal{P}^\Gamma$ garantiza a $\mathcal{B}$ y $\mathcal{B} \rightarrow_{\mathcal{P}^\Gamma} \mathcal{A}$ o $\mathcal{A} \rightarrow_{\mathcal{P}^\Gamma} \mathcal{B}$.*

Prueba : *Suponga que existe un argumento representativo $\mathcal{B}$ tal que está garantizado en $\mathcal{P}^\Gamma$ y $\mathcal{B} \rightarrow_{\mathcal{P}^\Gamma} \mathcal{A}$. Entonces, por Proposición 5.12, $\mathcal{B}$ aparece marcado como* **U** *en el árbol $\mathcal{T}^*_{\mathcal{A}}$. Por lo tanto, $\mathcal{A}$ no estaría garantizado, lo cual contradice la hipótesis.* □

Suponga ahora que existe un argumento representativo $\mathcal{B}$ tal que está garantizado en $\mathcal{P}^\Gamma$ y $\mathcal{A} \rightarrow_{\mathcal{P}^\Gamma} \mathcal{B}$. Entonces, dado que $\mathcal{B}$ está garantizado, $\mathcal{A}$ está marcado como **D** *en $\mathcal{T}^*_{\mathcal{B}}$, lo cual es un absurdo por Proposición 5.12.* □

Este resultado es de suma importancia ya que, como se verá a continuación, será utilizado para demostrar que T-DeLP no garantiza literales tipados en desacuerdo. Para esto, primero se definirá cuándo un literal tipado está garantizado en T-DeLP.

Claramente, si hay un argumento garantizado a partir de un programa T-DeLP, implica que hay una garantía para inferir su conclusión. Por lo tanto, en T-DeLP se dirá que un literal tipado está garantizado si es la conclusión de un argumento garantizado. Los literales tipados garantizados corresponderán entonces a los literales inferidos bajo el procedimiento de prueba dialéctico.

Definición 5.33 (Literal Garantizado) *Sea $\mathcal{P}^\Gamma$ un programa T-DeLP y L^T un literal tipado. Se dirá que $\mathcal{P}^\Gamma$ garantiza a L^T, notado $\mathcal{P}^\Gamma \vdash_W L^T$, si y solo si existe un argumento representativo $\langle \mathcal{A}, L^T \rangle$ en $\mathcal{P}^\Gamma$ tal que $\langle \mathcal{A}, L^T \rangle$ está garantizado en $\mathcal{P}^\Gamma$.*

El resultado más importante acerca de un programa T-DeLP es que no garantizará literales en desacuerdo. Es decir, el conjunto de literales inferidos por un programa T-DeLP a través del procedimiento de prueba dialéctico será consistente.

Teorema 5.3 *Sea* $\mathcal{P}^\Gamma = (\Gamma, \Theta, \Delta)$ *un programa T-DeLP y* L^T *un literal tipado. Si* $\mathcal{P}^\Gamma \vdash_W L^T$ *y* $(L^T, M^U) \in \mathsf{FDesacuerdo}(\Gamma)$*, entonces* $\mathcal{P}^\Gamma \nvdash_W M^U$*.*

Prueba : *Suponga que existe un literal tipado* M^U *garantizado en* $\mathcal{P}^\Gamma$ *tal que* $(L^T, M^U) \in \mathsf{FDesacuerdo}(\Gamma)$*. Entonces existe un argumento* $\langle \mathcal{B}, M^U \rangle$ *garantizado en* $\mathcal{P}^\Gamma$*. Luego, como* $(L^T, M^U) \in \mathsf{FDesacuerdo}(\Gamma)$ *entonces* $\langle \mathcal{B}, M^U \rangle \rightarrow_{\mathcal{P}^\Gamma} \langle \mathcal{A}, L^T \rangle$ *o* $\langle \mathcal{A}, L^T \rangle \rightarrow_{\mathcal{P}^\Gamma} \langle \mathcal{B}, M^U \rangle$*, para cualquier argumento* $\mathcal{A}$ *para* L^T*. Entonces, por Lema 5.1,* $\mathcal{P}^\Gamma$ *no garantiza a ningún argumento* $\langle \mathcal{A}, L^T \rangle$*. Por lo tanto,* $\mathcal{P}^\Gamma$ *no garantiza a* L^T*, lo cual contradice la hipótesis.* □

De manera similar a DeLP, en T-DeLP se asegura que todos los hechos tipados de un programa están garantizados. Esto se muestra mediante la siguiente proposición.

Proposición 5.13 *Sea* $\mathcal{P}^\Gamma = (\Gamma, \Theta, \Delta)$ *un programa T-DeLP y* L^T *un literal tipado. Si* L^T *es un hecho de* Θ*, entonces* $\mathcal{P}^\Gamma \vdash_W L^T$*.*

Prueba : *Como* L^T *es un hecho, por definiciones 5.12, 5.15 y 5.18, existe un argumento representativo* $\langle \{L^T\}, L^T \rangle$*. Luego, dado que por Proposición 5.11* $\{L^T\} \cap \Delta = \emptyset$*, no existe ningún argumento que ataque a* $\langle \{L^T\}, L^T \rangle$*. Por lo tanto, no existirá argumento que derrote a* $\langle \{L^T\}, L^T \rangle$*. Luego,* $\langle \{L^T\}, L^T \rangle$ *es un argumento garantizado en* $\mathcal{P}^\Gamma$*, por lo cual* $\mathcal{P}^\Gamma \vdash_W L^T$*.* □

5.5. Trabajo Relacionado

Como se mencionó en el capítulo anterior, en la actualidad no existen otras aproximaciones de argumentación que modelen el concepto de tipo de argumento de forma explícita y general. Esto aplica también para los sistemas argumentativos con argumentos concretos como T-DeLP. Si bien existen algunas aproximaciones que identifican diferentes tipos dentro los posibles argumentos con los que trabajan, estos tipos están fijos al dominio del problema que resuelven. Adicionalmente, en algunos de estos trabajos es posible que argumentos de diferentes tipos sean estructuralmente diferentes (como por ejemplo, argumentos que soportan dos conclusiones).

En [CS07] se propone un sistema argumentativo basado en etiquetas. Estos trabajos combinan argumentación con la aproximación de los sistemas de etiquetas estudiados en [Gab96]. Si bien en T-DeLP los tipos se asemejan a las etiquetas por la forma en que

se asignan a los literales y por como son propagados, T-DeLP y el sistema argumentativo de [CS07] son completamente diferentes. En primer lugar, en ese sistema las etiquetas son utilizadas para diferenciar entre conocimiento estricto y rebatible. Por otra parte, el objetivo del uso de etiquetas en [CS07] es más sofisticado que el de los identificadores de tipo en T-DeLP. Allí las etiquetas son utilizadas para modelar todo el proceso argumentativo: derivación, construcción de argumentos, construcción de árboles de dialéctica y cómputo de garantía. De esa manera, al combinar las etiquetas con diferentes reglas de inferencia se permite generar todos estos conceptos. Es decir, en [CS07] las etiquetas constituyen el elemento fundamental para desarrollar el cómputo argumentativo. En contrapartida, en T-DeLP es diferente, ya que los identificadores de tipo son simplemente utilizados para identificar a qué tipo pertenece cada literal (y no una regla rebatible), y para inferir qué tipo tendrá finalmente un argumento. Adicionalmente, la forma en que se construyen los argumentos tipados y se determina su aceptabilidad en T-DeLP, está basada en DeLP [GS04] y en los marcos argumentativos abstractos [Dun95]. Aun así, en [CS07] es posible aplicar un sistema de etiquetas por encima del sistema argumentativo. En este sentido, resultaría interesante contar con una aproximación que aplique los conceptos de argumentación basada en tipos a estos sistemas usando una aproximación basada en etiquetas.

Como se mencionó en la introducción de este capítulo, existen varias aproximaciones que combinan programación en lógica con tipos [KW91, HTT90]. En T-DeLP se utilizaron algunos de los conceptos desarrollados para estas aproximaciones, como por ejemplo conformidad, herencia de tipos y propagación. Sin embargo, en particular no se aplicó a T-DeLP ninguno de esto sistemas por completo, ya que los objetivos son completamente diferentes y, por lo tanto, las soluciones también son diferentes. En T-DeLP se busca proveer un mecanismo para construir argumentos tipados en el que los tipos se infieren a través la construcción de los argumentos. En estas otras aproximaciones el objetivo es proveer a la programación en lógica de un modelo de sistema de tipos en el sentido de los lenguajes de programación [Seb99]. Por lo tanto, el análisis de estos trabajos se centra esencialmente en los mecanismos que asignan tipos a los distintos términos para luego realizar chequeos entre los argumentos de los predicados, considerando los diferentes tipos de polimorfismo soportados por el sistema de tipos. Claramente, en estas aproximaciones se busca resolver un problema, si bien diferente, mucho más complejo (con respecto al tratamiento de tipos) que el abordado en este capítulo. Aquí no se busca realizar chequeos de tipo para los argumentos, sino que se busca determinar qué tipos tendrán los

argumentos.

5.6. Conclusiones

En este capítulo se presentó T-DeLP, un sistema argumentativo que extiende a DeLP para contemplar la noción de tipo de argumento. En conjunto con el formalismo presentado en el capítulo anterior para los marcos argumentativos de tipos múltiples, conforma un sistema argumentativo completo para tratar con la problemática asociada a la representación y cómputo de argumentos tipados. Para esto fue necesario desarrollar un formalismo capaz de construir argumentos tipados a partir de un conjunto de reglas y hechos, identificar cuáles de estos argumentos eran representativos y, por último, combinar este formalismo con los resultados obtenidos en el capítulo anterior para determinar la aceptabilidad de sus argumentos.

Presentar un formalismo como T-DeLP implicó varios desafíos. En primera medida, proponer un lenguaje de representación a partir del cual se puedan construir argumentos e inferir sus tipos. Para esto se extendió el concepto de literal permitiendo que tenga un tipo asociado, de manera que las reglas rebatibles y hechos también tengan tipos asociados. Luego fue necesario desarrollar un modelo de derivación capaz de contemplar las interacciones y características de estos tipos. Para ello se presentó una derivación que contempla el concepto de conformidad, una noción clásica en la literatura de sistemas de tipo. De este modo, la derivación de T-DeLP es capaz, no sólo de derivar un literal bajo un tipo, sino también bajo todos sus tipos ancestros. Adicionalmente, esta derivación admite la propagación de ciertos tipos permitiendo así que, por ejemplo, unas reglas rebatibles con cierto tipo sean aprovechadas para realizar derivaciones con tipos más específicos. Todo esto le brinda una gran flexibilidad a los programas T-DeLP ya que, por ejemplo, si se especifica una regla rebatible tal que todos sus literales son de un tipo más general, esta regla podrá utilizarse para derivar el literal de su cabeza bajo cualquier tipo descendiente que cumpla con las condiciones para ser propagado.

A partir de esta noción de derivación fue posible presentar el concepto de argumento tipado en T-DeLP. Básicamente, el tipo de un argumento es el tipo bajo el cual se deriva su conclusión. Si bien es un concepto directo, las características de la derivación llevaron a realizar un análisis más profundo para construir e identificar los argumentos que finalmente se obtendrán a partir de un programa T-DeLP. Esto se debió, principalmente, a que es

posible generar distintas versiones del mismo argumento pero con diferente tipo, tales que esos tipos se encuentran relacionados a través de herencia. Por lo tanto, fue necesario presentar una noción central para T-DeLP: la noción de *argumento representativo*. Este tipo de argumentos son aquellos que engloban todas las características de las diferentes versiones del mismo argumento. Se mostró que estos argumentos existen, son únicos, y que el sistema será completo considerando únicamente estos argumentos.

Para determinar qué argumentos representativos son aceptados, y por lo tanto qué literales están garantizados en T-DeLP, fue necesario determinar una relación de derrota entre los argumentos, es decir, se debió identificar qué argumentos no pueden aceptarse en conjunto. Para ello se aprovecharon los MATMs presentados en el capítulo anterior, ya que brindan un formalismo lo suficientemente sofisticado como para poder generar la relación de derrota entre argumentos a partir de todas las relaciones entre argumentos y tipos. Por lo tanto, dado un programa T-DeLP se mostró cómo generar su MATM asociado para así obtener las derrotas, y luego poder aplicar cualquier semántica de aceptabilidad de las estudiadas en la literatura. Esto último le provee a T-DeLP un marco de aplicación muy amplio, ya que cualquier semántica de aceptabilidad desarrollada para un marco argumentativo abstracto podrá también ser aplicada a T-DeLP.

Finalmente, se presentó un modelo de inferencia para T-DeLP basado en un procedimiento de prueba dialéctico que extiende al modelo de DeLP. Este modelo aprovecha la relación de derrota que se obtiene del MATM asociado, y determina si un literal está garantizado o no a partir de un programa T-DeLP. Con este modelo y el modelo basado en los MATMs, T-DeLP cuenta con dos aproximaciones computacionales diferentes para determinar la aceptabilidad de sus argumentos. Así, con todos estos elementos, T-DeLP presenta un sistema argumentativo completo y flexible, capaz de lidiar con las problemáticas que involucran la construcción, interacción y aceptabilidad de argumentos con diferentes tipos.

Capítulo 6

Programación de Agentes basados en Argumentación Tipada

En este capítulo se presentará la integración del sistema argumentativos basado en tipos T-DeLP con un lenguaje de programación de agentes basado en 3APL. El lenguaje resultante será llamado ArgAPL, en el cual percepciones, creencias, metas de logro y metas de mantenimiento serán especificadas utilizado reglas y hechos tipados T-DeLP. Por lo tanto, a diferencia de 3APL, los agentes en este lenguaje podrán representar información potencialmente conflictiva en sus componentes mentales, y el mecanismo de razonamiento argumentativo decidirá qué prevalece. Así, cada componente mental determinará argumentos de un mismo tipo, los cuales capturarán las propiedades representacionales del componente. El poder expresivo de T-DeLP permitirá modelar y razonar adecuadamente con las interacciones y conflictos entre estos componentes mentales. Adicionalmente, se caracterizará un conjunto de capacidades que permitirán al agente modificar dinámicamente el conocimiento almacenado en sus componentes mentales. Por último, se presentará la semántica formal y una estrategia de deliberación.

6.1. Introducción

En el Capítulo 2 se mostró que los lenguajes de programación de agentes cognitivos han recibido una atención especial recientemente [BBD+06], [DvRM05], [BWH07]. Entre las aproximaciones más destacadas se encuentran 3APL [HdBvdHM99, DvRM05],

Jason [BWH07], y GOAL [dBHvdHM07]. Estos lenguajes proveen características como especificaciones declarativas, semánticas formales, y desarrollo de agentes verificables. Sin embargo, ninguno de ellos utiliza argumentación como mecanismo de razonamiento de sus agentes.

En la literatura del área de inteligencia artificial, el uso de argumentación es ampliamente reconocido como mecanismo de razonamiento en agentes inteligentes [BCD07]. Desde los comienzos del desarrollo de agentes racionales a través de modelo *Belif-Desires-Intentions* (BDI) [RG91], se ha considerado el uso de razonamiento rebatible para manejar el modelo de decisión intencional. Luego, [PSJ98] dio un paso hacia adelante en esta dirección y apuntó las ventajas del uso de argumentación en agentes BDI. Actualmente, varios trabajos [ADL08, RGS07, RA06] presentan contribuciones importantes en este área, conectando aproximaciones argumentativas con arquitecturas BDI. Estos artículos muestran cómo algunos componentes mentales pueden beneficiarse con el uso de argumentación para razonar con información conflictiva. Sin embargo, como se mostró en el Capítulo 2, en el campo de programación de agentes, a pesar de que la deliberación es un tema importante, es necesario establecer una semántica de ejecución para el lenguaje. Es decir, en este área es importante explicitar reglas que muestren cómo las acciones y decisiones del agente afectarán a sus componentes mentales y, en consecuencia, su razonamiento futuro. Los trabajos en argumentación y BDI mencionados anteriormente proponen arquitecturas BDI en lugar de lenguajes de programación de agentes (APLs, por su sigla en inglés *agent programming languages*). Por lo tanto, se enfocan principalmente en el soporte argumentativo para la deliberación, y no en la semántica de ejecución.

El objetivo principal de este capítulo es presentar un lenguaje de programación de agentes, ArgAPL, en el que la argumentación sea el principal pilar para el razonamiento de sus agentes. Es decir, que el mecanismo argumentativo no sólo permita razonar al agente sobre qué metas perseguir (deliberación), sino que también ayude al agente a razonar con información conflictiva que surge de la dinámica de su ejecución. Por ejemplo, se buscará que el mecanismo argumentativo ayude al agente a detectar cuándo una meta que está persiguiendo ya fue alcanzada, motivo por el cual no tiene sentido seguir con ella.

Para esto se utilizará T-DeLP que, además de permitir especificaciones declarativas (un requisito para este tipo de lenguajes), provee el concepto de tipo de argumento que resultará de gran utilidad para los objetivos de ArgAPL. Así, un agente en este lenguaje

tendrá argumentos para percepciones, creencias, metas de logro y metas de mantenimiento, los cuales interactuarán y estarán en conflicto entre sí. Continuando con la intuición del párrafo anterior, un argumento para una creencia "a" debería derrotar a un argumento que sustenta una meta de logro "a", dado que esto implica que la meta ya ha sido alcanzada.

Más aun, aprovechando las ventajas del razonamiento argumentativo y la identificación de tipos de argumento provista por T-DeLP, se buscará modelar conceptos avanzados como la representación de metas de mantenimiento tanto reactivas como proactivas. En particular, este tema ha sido dejado de lado en lenguajes como 3APL o Jason debido a la complejidad de los conflictos que generan estas metas, especialmente cuando se busca protegerlas proactivamente. Por lo tanto, los argumentos para metas de mantenimiento atacarán otras metas cuyos planes pongan en riesgo la condición que quieren mantener.

En conclusión, en este capítulo se mostrará ArgAPL, un lenguaje de programación de agentes basado en 3APL, el cual utilizará T-DeLP como mecanismo de razonamiento deliberativo y dinámico, donde sus principales contribuciones son las siguientes:

- representa la percepción de manera explícita y modular;
- contempla la representación de metas de mantenimiento, además de las metas de logro;
- admite la representación de metas condicionales (tanto de mantenimiento como de logro);
- utiliza los identificadores de tipo de T-DeLP para identificar las reglas de razonamiento de cada componente mental;
- permite representar metas y creencias conflictivas;
- se construyen argumentos para determinar las inferencias de cada componente mental, donde el tipo asociado a cada argumento corresponde al componente metal que representa;
- aprovecha el mecanismo argumentativo para razonar con la información conflictiva de cada componente mental;
- emplea el mecanismo argumentativo para razonar sobre los conflictos que surgen de la dinámica del agente;

- utiliza el mecanismo argumentativo para asegurar proactividad con respecto a la protección de las metas de mantenimiento del agente; y
- en todos los proceso argumentativos del agente los tipos constituyen un elemento esencial al momento de determinar qué argumentos están en conflicto y cuáles de ellos prevalecen.

El capítulo estará estructurado de la siguiente manera. En primer lugar se presentará un dominio de ejemplo, el cual será utilizado para ejemplificar los diferentes constructores del lenguaje. Luego, en la Sección 6.2, se presentará cómo especificar un agente ArgAPL. En particular, se mostrará cómo representar creencias, metas de logro, metas de mantenimiento, acciones, planes y reglas de selección de plan. Seguidamente, en la Sección 6.3, se presentará la semántica de ArgAPL. Allí se mostrará el modelo de razonamiento del agente, construyendo un programa T-DeLP para determinar cuáles son las creencias y metas de un agente en un estado particular. Adicionalmente, en esa sección se mostrarán las reglas de transición de estado para un agente ArgAPL que determinarán su semántica operacional. Por último, se presentará trabajo relacionado seguido de conclusiones.

6.1.1. Dominio de Ejemplo

En esta sección se presentará un dominio para el cual se construirán agentes ArgAPL. Diferentes escenarios en este dominio ayudarán a mostrar cómo se especifican los agentes de este lenguaje y cómo es su dinámica.

Considere que se busca implementar un agente que se encuentra en un dominio en el cual existen diversas islas que pueden llegar a contener tesoros. El objetivo del agente es recolectar la mayor cantidad de tesoros posibles. El agente puede estar en una sola isla a la vez, en la cual puede buscar un tesoro, y también puede viajar entre las islas. Tanto los viajes como las búsquedas de tesoros implican un gasto de energía para el agente, por lo tanto, debe ser cuidadoso ya que si se queda sin energía se desmayará y se le caerán todos los tesoros ya recolectados. Para recuperar su energía el agente dispondrá de un refugio en una de las islas, a donde podrá ir a descansar y reponer así sus fuerzas para futuras aventuras. Adicionalmente, es posible que haya otros agentes que también se encuentren recolectando tesoros por las islas. Si dos agentes se encuentran en la misma isla podrán

combatir, en cuyo caso el agente derrotado perderá una cantidad considerable de energía y, si se desmayase, el ganador podrá quedarse con todos sus tesoros.

Las islas estarán distribuidas en una grilla y los agentes podrán viajar directamente de una a otra, es decir, no necesitarán transitar por cada uno de los casilleros entre isla e isla. Sin embargo, la distancia entre estas islas (distancia *Manhattan* [Kra87]) determinará cuan costoso es el viaje para el agente.

El tiempo en este dominio transcurrirá en turnos. En el transcurso de un turno un agente podrá viajar de una a isla a otra, descansar, juntar un tesoro de la isla en la que se encuentra, o atacar a un agente que esté en la misma isla que él. Adicionalmente, en cada turno el agente recibirá información acerca de los agentes y tesoros que se encuentren en las islas que sean visibles dentro de su rango de visión. El rango de visión del agente estará determinado por un cuadrado centrado en la isla en la que se encuentra, cuya longitud de lado será de 5 casilleros. Además, será posible que en cada turno aparezcan nuevos tesoros en las islas.

Las acciones que el agente puede realizar le insumirán un gasto de energía, con excepción de las acciones para descansar y levantar un tesoro. En particular, atacar un agente consumirá una unidad de energía, mientras que viajar consumirá una cantidad de energía equivalente a la distancia entre las islas. Por otra parte, si un agente es atacado y el ataque tiene éxito, perderá entre 20 y 50 unidades de energía (a determinar aleatoriamente).

Ejemplo 6.1 *Considere el siguiente escenario para el agente "yo" presentado en la Figura 6.1. La figura ilustra que en este escenario hay cuatro islas:* $i1$, $i2$, $i3$, *e* $i4$*; que los tesoros* $t1$, $t2$, *y* $t4$ *están en las islas* $i2$, $i4$ *e* $i1$ *respectivamente; que el agente "yo" se encuentra en la isla* $i3$ *y cuenta con 95 unidades de energía; que el agente "*$ag1$*" está en la isla* $i4$*, posee el tesoro* $t3$ *y una energía de 25 unidades; que el agente "*$ag2$*" se halla en la isla* $i1$ *con una energía de 100 unidades; y que en la isla* $i1$ *existe un refugio.*

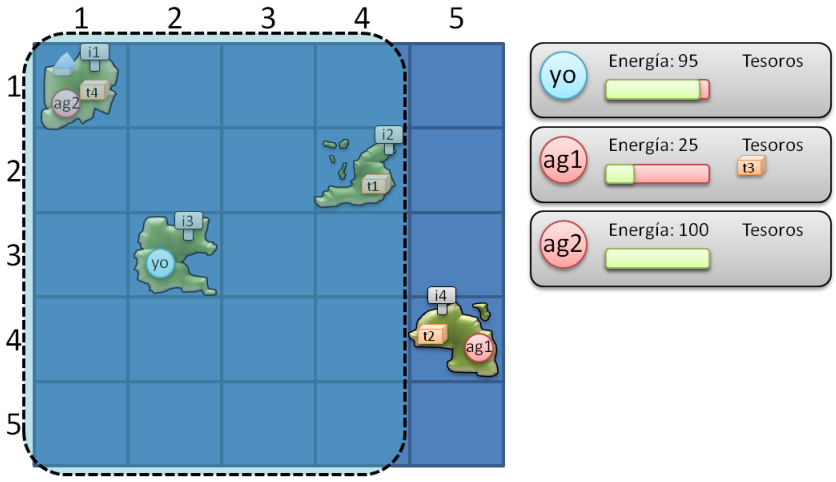


Figura 6.1: Escenario para el agente "*yo*" en el dominio descripto en la Sección 6.1.1.

Dado que el agente "yo" se encuentra en la isla i3, percibirá todos los tesoros y agentes que se hallen su isla y en aquellas islas que se sitúen a una distancia menor o igual que 3 (note que esta zona se enmarca en la Figura 6.1 mediante un rectángulo con borde punteado). Por lo tanto, el agente "yo" percibirá que se encuentra en la isla i3, su energía actual, que en las islas i1 e i2 hay tesoros, y que agente "ag2" está en la isla i1. Observe que, por encontrarse fuera de su rango de visión, el agente "yo"no ve que el agente "ag1" está en la isla i4, ni que hay un tesoro allí.

6.2. Especificación de Agentes ArgAPL

En esta sección se describirá la sintaxis de ArgAPL y se mostrará cómo especificar a los agentes ArgAPL. Estos agentes estarán conformados por cuatro componentes: una base de creencias, una base de metas, un conjunto de reglas de selección de planes, y un conjunto de acciones posibles. Las bases de creencias y de metas representarán los componentes mentales del agente y serán especificadas a través de programas T-DeLP. Estas bases de conocimiento serán utilizadas, como en 3APL, para computar las creencias y metas en cada ciclo deliberativo del agente. Las reglas de selección de planes se emplearán para generar planes con el objetivo de alcanzar las metas actuales del agente. Estos planes contendrán acciones que permitirán al agente interactuar con el ambiente o modificar sus bases de conocimiento.

Un agente ArgAPL podrá hacer referencia en su especificación a diferentes piezas de información, ya sea para representar lo que cree, lo que busca, o lo que quiere mantener. Esta información se representará a través de literales tipados. El tipo de estos literales corresponderá al del componente mental que está representando. En ArgAPL se utilizarán los tipos P (de *perceptions*) para representar las percepciones, B (de *beliefs*) para las creencias, G (de *goals*) para las metas, AG (de *achievement goals*) para las metas de logro, y MG (de *maintenance goals*) para las metas de mantenimiento. De esta manera, por ejemplo, un literal L^B representará que el agente cree en L, y un literal L^{AG} significará que el agente tiene como meta de logro a L. Durante el transcurso de esta sección se ilustrará cómo utilizar estos literales.

6.2.1. Especificación de Creencias

Como en la mayoría de los lenguajes de programación de agentes, la base de creencias es utilizada para representar información que el agente posee e infiere a cerca del mundo en el que desenvuelve, representado un componente informacional en el modelo del agente. Por lo tanto, la base de creencias permitirá determinar en qué literales el agente cree en un determinado momento, a partir de conocimiento que posee del dominio, las reglas de inferencia acerca de este conocimiento, y la información que percibe actualmente del mundo.

En 3APL la base de creencias del agente representa tanto conocimiento como percepciones actuales del mundo, y es usualmente especificada mediante un conjunto de reglas Prolog. En ArgAPL se hace una distinción entre la base de creencias y el conjunto de percepciones del agente. Esta distinción sigue el estilo de GOAL [HvR07], y está principalmente motivada por las consideraciones dinámicas que requiere el tratamiento de las percepciones en un agente, las cuales no son consideradas en 3APL. Si bien en ArgAPL estos componentes se representarán por separado, como se verá más adelante, estarán relacionados ya que las percepciones serán también creencias y, por lo tanto, la base de creencias podrá apoyarse en ellas para realizar sus inferencias.

Base de Creencias

En 3APL una base de creencias suele estar constituida por un programa Prolog o un lenguaje proposicional simplificado (Prolog sin variables). De esta manera, en 3APL el

conocimiento representado en la base de creencias debe ser consistente. En ArgAPL esto último no es necesario, ya que la base de creencias de un agente estará representada por un conjunto de reglas rebatibles tipadas. Por lo tanto, será posible representar creencias potencialmente conflictivas y, como se verá más adelante, el mecanismo argumentativo de T-DeLP permitirá razonar con este tipo de información.

Como se mencionó anteriormente, se utilizará el identificador de tipo B (del inglés, *beliefs*) para identificar a las creencias. Por consiguiente, las reglas de la base de creencias contendrán literales tipados de tipo B. De esta manera, los literales tipados de tipo B constituirán posibles creencias para un agente en ArgAPL.

Definición 6.1 (Base de Creencias) *Una base de creencias para un agente* ArgAPL *es un conjunto Δ^B tal que todo $R \in \Delta^B$ es una regla rebatible tipada de la forma: $L_h^B \prec L_1^B, \ldots, L_n^B$, con $n \geq 0$.*

Ejemplo 6.2 *Considere la base de creencias ${\Delta^B}_1$ para un agente que se desenvuelve en el dominio presentado en la Sección 6.1.1. Básicamente, esta base de creencias será utilizada para saber si una isla es promisoria y/o peligrosa, además de contener toda la información correspondiente a la distribución de las islas.*

$$\Delta_1^B = \left\{ \begin{array}{ll}
(1) & dist(I_o, I_d, D)^B \prec pos(I_o, X_o, Y_o)^B, pos(I_d, X_d, Y_d)^B, \\
 & \qquad D = |X_o - X_d| + |Y_o - Y_d| \\
(2) & promisoria(I)^B \prec en(I, tesoro(X))^B \\
(3) & promisoria(I)^B \prec en(I, Ag)^B, rival(Ag)^B, presaFacil(Ag)^B \\
(4) & \sim promisoria(I)^B \prec en(I, tesoro(X))^B, peligrosa(X)^B \\
(5) & peligrosa(I)^B \prec en(I, Ag)^B, rival(Ag)^B \\
(6) & \sim peligrosa(I)^B \prec en(I, Ag)^B, rival(Ag)^B, cansado(Ag)^B \\
(7) & cansado(Ag)^B \prec energia(Ag, E)^B, E < 20 \\
(8) & \sim cansado(Ag)^B \prec energia(Ag, E)^B, E >= 20 \\
(9) & en(i1, refugio)^B \prec \\
(10) & rival(Ag)^B \prec Ag \neq yo \\
(11) & masLejos(I_1, I_2, Ag)^B \prec en(I_o, Ag)^B, dist(I_o, I_1, D_1)^B, dist(I_o, I_2, D_2)^B, D_1 > D_2 \\
(12) & presaFacil(Ag)^B \prec rival(Ag)^B, tiene(Ag, tesoro(X))^B, cansado(Ag)^B \\
(13) & modo(defensivo)^B \prec
\end{array} \right\}$$

En la base de creencias ${\Delta^B}_1$, la regla (1) denota que hay razones para creer que la distancia entre las islas I_o e I_d es D (distancia Manhattan) si las coordenadas de las islas son

(X_o, Y_o) y (X_d, Y_d) respectivamente. Por otra parte, la regla número (2) denota que si el agente cree que en una isla I hay un tesoro X, entonces tendrá razones para creer que esa isla es un destino promisorio. La regla (4) expresa que el agente tiene razones para creer que la isla I no es promisoria si, por más que haya un tesoro allí, cree que es peligrosa. La regla (5) expresa que es posible creer tentativamente que la isla I será un lugar peligroso si hay otro agente que se encuentre en I. Observe que la regla (9) brinda información acerca de la ubicación del refugio para el agente. La regla (12) expresa que hay razones para creer que un agente será una presa fácil si tiene un tesoro y está cansado. La última regla expresa que el agente tiene razones para creer que se encuentra en modo defensivo.

Continuando con el Ejemplo 6.2, observe que, a diferencia de 3APL, un desarrollador de agentes podrá representar creencias potencialmente contradictorias utilizando las reglas rebatibles tipadas de T-DeLP. Esto se visualiza, por ejemplo, en las reglas *(5)* y *(6)*, y en las reglas *(7)* y *(8)*. Claramente, si estas reglas son empleadas para construir argumentos representativos para creencias, los argumentos obtenidos estarán en conflicto. Más adelante, cuando se presenten las nociones semánticas del lenguaje, se profundizarán estos conceptos.

Otra situación que cabe destacar es que existen diversos literales que aparecen en el cuerpo de las reglas rebatibles tipadas de la base de creencias del Ejemplo 6.2, pero que no son cabeza de ninguna de las reglas. Este es el caso, entre otros, de los literales "$energia(Ag, E)^B$" en la regla *(7)* y "$en(I, Ag)^B$" en la regla *(2)*. Esto se debe a que estos literales tipados, en el contexto de este dominio, constituirán percepciones para el agente. Es decir, corresponderán a literales que el agente podrá percibir directamente de su entorno.

Percepciones

Las percepciones de un agente ArgAPL estarán conformadas por un conjunto de hechos. Si bien las percepciones serán también creencias (indiscutibles), será necesario distinguirlas de los demás elementos de la base de creencias. Por lo tanto, como se mencionó anteriormente, los literales del conjunto de percepciones estarán caracterizados por el tipo P. De esta manera, los argumentos referentes a la percepción también serán de tipo P.

Definición 6.2 *Un conjunto de percepciones para un agente* ArgAPL *es un conjunto* Θ^P *tal que todo* $L \in \Theta^P$ *es una hecho tipado de la forma* L^P.

Ejemplo 6.3 *Considere el escenario del Ejemplo 6.1 para el agente "yo", en el dominio descripto en la Sección 6.1.1. Como se vio en la Figura 6.1, el agente "yo" se encuentra en la isla* $i3$ *y, por lo tanto, percibirá todos los tesoros y agentes en su isla y en aquellas islas situadas a no más de 3 celdas de distancia (note que esta zona se enmarca en la Figura 6.1 con el rectángulo de borde punteado). En* ArgAPL *el agente "yo" tendrá las siguientes percepciones* $\Theta_1^P = \{en(i3, yo)^P$, $energia(yo, 95)^P$, $en(i1, ag2)^P$, $energia(ag2, 100)^P$, $en(i1, tesoro(t4))^P$, $en(i2, tesoro(t1))^P$, $pos(i1, 1, 1)^P$, $pos(i2, 2, 4)^P$, $pos(i3, 3, 2)^P$ $\}$.

Observe que Θ_1^P *representa que el agente "yo", percibirá que está en la isla* $i3$ *y su energía es 95, que el agente "*$ag2$*" con una energía de 100 esta en la isla* $i1$*, que en las islas* $i1$ *e* $i2$ *hay tesoros, y finalmente las posiciones de las islas* $i1$*,* $i2$ *e* $i3$*. Note que, dado su rango de visión limitado, el agente "yo"no percibe que el agente "*$ag1$*" está en la isla* $i4$*, ni que hay un tesoro allí.*

A diferencia de la base de creencias el conjunto de percepciones no se encuentra especificado por el desarrollador, sino que es determinado por el entorno en que se desenvuelve el agente. En particular, la forma en que se actualiza este conjunto, integrando las nuevas percepciones a las viejas y manteniendo su consistencia, se verá más adelante cuando se presente la dinámica de ArgAPL.

Como se mencionó anteriormente, existe una relación entre las creencias y las percepciones de un agente ArgAPL. En los ejemplos 6.2 y 6.3 se puede observar cómo ciertas reglas de la base de creencias hacen referencia a literales que corresponden a percepciones. Esto se debe a que, como se verá más adelante, las percepciones serán creencias que poseen ciertas características particulares. Esta relación, entre otras, se presentará cuando se formalice el modelo de razonamiento de los agentes ArgAPL.

6.2.2. Especificación de Metas

Al igual que en los lenguaje de programación de agentes cognitivos, las metas en ArgAPL constituyen el componente motivacional de un agente. Como en tales lenguajes, ArgAPL contará una base de metas a partir de la cual será posible, en un determinado momento, establecer cuáles son las metas que el agente buscará alcanzar.

Como se mencionó en el Capítulo 2, 3APL sólo considera metas de logro, las cuales representan situaciones del mundo que el agente quiere alcanzar. En contraste, en ArgAPL se podrán representar dos tipos de metas: metas de logro y metas de mantenimiento. A diferencia de las metas de logro, las metas de mantenimiento denotan situaciones de el mundo que el agente quiere mantener.

Por otra parte, en 3APL las metas de un agente son un conjunto consistente de átomos que el agente perseguirá incondicionalmente. En ArgAPL las metas de un agente serán condicionales, es decir, el agente las perseguirá si se dan ciertas condiciones. Adicionalmente, estas metas podrán estar en conflicto (sean del mismo tipo o de tipos diferentes) y, como se verá más adelante, el mecanismo argumentativo de T-DeLP utilizado para el razonamiento en ArgAPL decidirá qué metas prevalecerán ante tales conflictos.

Para esto, como se verá a continuación, la base de metas en ArgAPL se especificará mediante dos bases: la base de metas de logro y la base de metas de mantenimiento. En particular, en ArgAPL se utilizarán los identificadores de tipo G (*goals*) para identificar a las metas en general, AG (*achievement goals*) para identificar a las metas de logro, y MG (*maintenance goals*) para identificar a las metas de mantenimiento. Por lo tanto, un literal L_1^G representará una meta en general, un literal L_2^{AG} corresponderá a una meta de logro, y un literal L_3^{MG} denotará una meta de mantenimiento.

Metas de Logro

Las metas de logro serán utilizadas para denotar situaciones del mundo que el agente alcanzar [HdBvdHM00, WPHT02, vRDM05, vRDW08]. Más concretamente, corresponderán a literales que el agente desea inferir como creencias a partir del conjunto de percepciones y su base de creencias. Estas metas son consideradas *metas declarativas* ya que hacen referencia a un estado, sin hacer mención a qué acciones permiten llegar a ese estado. Este tipo de metas es el más común en 3APL y sus variantes. En estos lenguajes las metas de logro son simplemente representadas a través de átomos, los cuales deben ser consistentes en forma conjunta. Por lo tanto, es normal caracterizar a estas metas como incondicionales.

En ArgAPL las metas de logro serán representadas a través de un conjunto de reglas rebatibles, tales que los literales tipados en sus cabezas determinarán las posibles metas de logro que tendrá el agente. Los literales que identifican a las metas de logro estarán

caracterizados por el identificador de tipo *AG* (de *achievement goal*). En el cuerpo de estas reglas podrán aparecer tanto creencias como también otras metas, y representarán las condiciones que justifican que el agente intente perseguir esa meta. Por lo tanto, se dirá que las metas de logro en ArgAPL siguen una política de adopción condicional.

Definición 6.3 (Base de metas de logro) *Una base de metas de logro para un agente* ArgAPL *es un conjunto* Δ^{AG} *tal que todo* $R \in \Delta^{AG}$ *es una regla rebatible tipada de la forma:* $L_h^{AG} -\prec L_1^{X_1}, \ldots L_n^{X_n}$*, con* $n \geq 0$ *y* $X_i \in \{B, G, AG, MG\}$.

Ejemplo 6.4 *Considere la base de metas de logro* ${\Delta^{AG}}_1$ *para un agente que se desenvuelve en el dominio presentado en la Sección 6.1.1. Esta base tendrá reglas para determinar cuándo un agente querrá o no estar en un lugar, y bajo qué condiciones querrá tener un tesoro.*

$$\Delta_1^{AG} = \left\{ \begin{array}{ll} (1) & en(I, yo)^{AG} -\prec promisoria(I)^B \\ (2) & \sim en(I, yo)^{AG} -\prec promisoria(I)^B, en(I_o, yo)^{AG}, masLejos(I, I_o, yo)^B \\ (3) & tiene(yo, tesoro(X))^{AG} -\prec en(I, tesoro(X))^B, en(I, yo)^B \\ (4) & modo(agresivo)^{AG} -\prec energia(yo, E)^B, E > 90 \\ (5) & modo(defensivo)^{AG} -\prec energia(yo, E)^B, E < 50 \\ (6) & atacarA(Ag)^{AG} -\prec en(I, yo)^B, en(I, Ag)^B, presaFacil(Ag)^B \end{array} \right\}$$

La primer regla denota que si una isla es promisoria, el agente tiene razones tentativas para querer estar allí. La segunda regla denota que hay razones para no querer estar en una isla que es promisoria, dado que el agente quiere estar en otra isla que le resulta más cercana. La tercer regla denota que si el agente esta en una isla en la que hay un tesoro, entonces posee razones para querer tener ese tesoro. Las reglas (4) y (5) denotan que el agente quiere estar en cierto modo, dependiendo del valor de su energía. La última regla expresa que el agente posee razones para querer atacar a otro agente, si se encuentra en la misma isla que él y además es una presa fácil.

Observe que en el cuerpo de las reglas rebatibles, por Definición 6.3 y como se puede ver en el Ejemplo 6.4, es posible hacer referencia a literales de diferentes tipos, entre ellos a las creencias. Más adelante, cuando se analice la dinámica del agente, se mostrará cómo combinar las bases de conocimiento y cómo establecer los respectivos desacuerdos y preferencias, para determinar qué metas de logro tiene el agente en un determinado momento.

Adicionalmente, note que la última regla del Ejemplo 6.4, a diferencia la demás, no necesariamente denota un estado del mundo al cual el agente buscará llegar, sino que también puede considerarse como una acción que el agente puede realizar. Por lo tanto, de esta manera será posible modelar fácilmente metas procedurales en ArgAPL.

Metas de mantenimiento

Las metas de mantenimiento, a diferencia de las metas de logro, son metas que representan un estado del mundo que el agente quiere mantener. Es decir, son metas que buscan proteger ciertas creencias del agente de determinadas acciones que puedan hacer que el agente deje de creer en ellas. Claramente, al igual que las metas de logro, estas metas son identificadas como declarativas ya que solo expresan qué estado del mundo es el que quieren mantener, sin especificar cómo lo hacen.

En general, en los lenguajes de programación de agentes las metas de mantenimiento suelen ser incondicionales y consistentes. Es decir, conforman conjunto de átomos consistente que el agente debería respetar durante toda su ejecución. En contraste, las metas de mantenimiento en ArgAPL serán condicionales y podrán estar en conflicto. Por lo tanto, al igual que las metas de logro, serán representadas utilizando reglas rebatibles tipadas. De esta manera, los literales en la cabeza de estas reglas serán las posibles metas de mantenimiento del agente, y tendrán asociado el identificador de tipo MG (*maintenance goal*). El cuerpo de estas reglas rebatibles solo estará integrado por creencias, ya que las metas de mantenimiento no pueden depender de otras metas [DHT06]. El conjunto de todas las reglas rebatibles para las metas de mantenimiento será conocido como la base de metas de mantenimiento.

Definición 6.4 (Base de metas de mantenimiento) *Una base de metas de mantenimiento para un agente* ArgAPL *es un conjunto* Δ^{MG} *tal que todo* $R \in \Delta^{MG}$ *es una regla rebatible tipada de la forma:* $L_h^{MG} \prec L_1^B, \ldots L_n^B$, *con* $n \geq 0$.

Ejemplo 6.5 *Considere la base de metas de mantenimiento* ${\Delta^{MG}}_1$ *para un agente que se desenvuelve en el dominio presentado en la Sección 6.1.1. Esta base tendrá reglas para intentar mantener la energía del agente alta, y para que se quede en islas promisorias si no se encuentra cansado.*

$$\Delta_1^{MG} = \left\{ \begin{array}{ll} (1) & \sim cansado(yo)^{MG} \prec \\ (2) & en(I, yo)^{MG} \prec en(I, yo)^B, promisoria(I)^B, \sim cansado(yo)^B \end{array} \right\}$$

Note que, como fue formalizado en la Definición 6.4 y se puede observar en el Ejemplo 6.5, las reglas rebatibles no necesariamente requieren que aparezca como creencia el literal que se quiere mantener. Esto se debe a que una regla en la base de metas de mantenimiento expresa las razones para querer mantener cierto literal. De esta manera, si el agente quiere mantener L y cree actualmente en L, tratará de protegerlo. En el caso en que el agente no crea actualmente en L, deberá alcanzarlo nuevamente a través de las metas de logro. Más adelante, cuando se estudie la dinámica del agente, se profundizarán estos conceptos, así como también el comportamiento de estas metas en el modelo de razonamiento del agente.

Una característica de las metas de mantenimiento en ArgAPL que lo diferencian de las aproximaciones existentes en 3APL, es que el agente será proactivo con respecto a la manutención de las metas. Es decir, el agente impedirá ejecutar acciones o elegir planes que perjudiquen sus metas de mantenimiento. Esto se visualizará cuando se presente la dinámica e interacción de los componentes mentales del agente.

Base de metas

Una vez introducidas las bases para los distintos tipos de metas en ArgAPL, se presentará la base de metas general. Esta base simplemente será la unión de la base de metas de logro y la base de metas de mantenimiento.

Definición 6.5 (Base de Metas) *Sean Δ^{AG} y Δ^{MG} la base de metas de logro y la base de metas de mantenimiento de un agente* ArgAPL. *La base de metas Δ^{G} para ese agente será $\Delta^{G} = \Delta^{AG} \cup \Delta^{MG}$.*

Cuando sea necesario, una base de metas Δ^{G} será notada a través del par $(\Delta^{AG}, \Delta^{MG})$, donde Δ^{AG} y Δ^{MG} son las bases de metas de logro y mantenimiento a partir de las cuales se formó Δ^{G}.

Ejemplo 6.6 *Continuando con el ejemplo del agente en el dominio presentado en la Sección 6.1.1, su base de metas será $\Delta_1^{G} = \Delta_1^{AG} \cup \Delta_1^{MG}$, donde Δ_1^{AG} es la base de metas de logro presentada en el Ejemplo 6.4 y Δ_1^{MG} es la base de metas de mantenimiento del Ejemplo 6.5.*

Note que, a diferencia de 3APL y sus variantes, en ArgAPL la base de metas contendrá dos tipos de metas. Más aun, en 3APL las metas suelen ser simples átomos, y por lo tanto incondicionales, mientras que en ArgAPL las metas son condicionales. Por otra parte, observe que ninguna de las reglas de la base de metas, ya sea de la base de metas de logro o de la de mantenimiento, tiene como cabeza literales tipados de la forma L^G. Esto se debe a que el tipo G será utilizado para caracterizar a todos los tipos de metas, ya sean de mantenimiento o de logro. Más adelante, cuando se presente el modelo de razonamiento del agente, se introducirán las relaciones entre todos estos tipos.

A partir de las reglas rebatibles tipadas utilizadas en la base de creencias, la base de metas de logro y la base de metas de mantenimiento, es posible observar cómo los tipos permiten definir claramente las restricciones sobre cómo se relacionarán la información e inferencias de los componentes mentales del agente. Además, los tipos en estas reglas también pueden ser utilizados para efectuar un chequeo de tipos y controlar si una especificación ArgAPL es sintácticamente correcta. Más adelante se verá que los tipos también constituyen un rol importante en la semántica de estos componente mentales.

6.2.3. Especificación de literales en conflicto

Si bien un agente ArgAPL dispone de la negación fuerte "$\sim$" para representar conflictos involucrando creencias y metas, es posible que el desarrollador quiera expresar conflictos entre ciertos literales más allá del inherente a los literales complementarios. Esto puede visualizarse en la base de metas de logro del Ejemplo 6.4, en la que existe una regla que provee razones para que el agente tenga como meta estar en una isla u otra, a través de los literales "$en(I, yo)$" (donde I se instanciará con las diferentes islas a las que el agente quisiese ir). Claramente, tales literales deberían estar en conflicto ya que el agente no puede estar en dos islas a la vez.

Para modelar este tipo de conflictos, en ArgAPL se proveerá un conjunto de literales en conflicto. Básicamente, este conjunto contendrá todos los pares de literales que, además de los literales complementarios con respecto a "$\sim$", representarán una situación conflictiva para el agente. La única restricción que deberá cumplir este conjunto es que un literal no puede estar en conflicto consigo mismo.

Definición 6.6 (Conjunto de literales en conflicto) *Un conjunto de literales en conflicto para un agente* ArgAPL *es un conjunto* $\oslash$ *de pares de literales tal que* $(L, L) \notin \oslash$*, y si* $(L_1, L_2) \in \oslash$ *entonces* $(L_2, L_1) \in \oslash$*.*

Ejemplo 6.7 *Considere el agente que se desenvuelve en el dominio presentado en la Sección 6.1.1. Este agente tendrá el conjunto de literales en conflicto* $\oslash_1$*, donde:*

- $(en(I, tesoro(X)), tiene(Ag, tesoro(X))) \in \oslash_1$
- $(en(I_1, tesoro(X)), en(I_2, tesoro(X))) \in \oslash_1$ *si* $I_1 \neq I_2$
- $(tiene(Ag1, tesoro(X)), tiene(Ag2, tesoro(X))) \in \oslash_1$ *si* $Ag1 \neq Ag2$
- $(en(I_1, Ag), en(I_2, Ag)) \in \oslash_1$ *si* $I_1 \neq I_2$
- $(energia(Ag, E_1), energia(Ag, E_2)) \in \oslash_1$ *si* $E_1 \neq E_2$

Por lo tanto, en $\oslash_1$ *estará el par de literales* $(en(i1, yo), en(i2, yo))$*, entre otros.*

Una cuestión importante a notar con respecto al conjunto de literales en conflicto, es que utiliza literales simples y no literales tipados. Esto se debe a que este conjunto captura los conflictos aplicables a todos los tipos de información. Por lo tanto, los pares de este conjunto estarán en conflicto a nivel de creencias, percepciones, metas de logro y metas de mantenimiento. Esto se analizará en detalle más adelante, cuando se determinen los desacuerdos al momento de estudiar el razonamiento de los agentes ArgAPL.

6.2.4. Acciones

Las acciones básicas especifican las capacidades del agente, es decir, corresponden a acciones que el agente puede ejecutar para alcanzar sus metas. Las acciones básicas son los elementos atómicos que constituyen un plan, como se verá más adelante en esta sección.

Los agentes ArgAPL contarán con dos tipos de acciones: acciones mentales, para modificar el conocimiento almacenado de los componentes mentales, y acciones externas, para interactuar con el ambiente. Existen más tipos de acciones prácticas al momento de implementar un agente, como es el caso de las acciones de test o las de comunicación. En ArgAPL estos tipos de acciones no serán consideradas, ya que pueden modelarse

utilizando las acciones externas y las mentales. Por lo tanto, si se incluyeran tales acciones, introducirían una complejidad adicional innecesaria al modelo de ejecución de los agentes, la cual escapa a los aportes de esta tesis.

Para presentar tanto las acciones mentales como las acciones externas se caracterizarán tres elementos sintácticos que las identifican: las precondiciones, los efectos mentales y los efectos perceptivos. Las precondiciones serán elementos en los que el agente tiene que creer para poder ejecutar la acción. Los efectos mentales representarán modificaciones sobre los componentes mentales del agente (creencias, metas de logro o metas de mantenimiento). Los efectos perceptivos representarán elementos que aparecerán en la futura percepción del agente.

Definición 6.7 (Conjunto de precondiciones) *Un conjunto β de literales tipado es un conjunto de precondiciones* ArgAPL *si y solo si $\beta = \{L_0^B, \ldots, L_k^B\}$, con $k \geq 0$.*

Definición 6.8 (Conjunto de efectos mentales) *Un conjunto de efectos mentales* ArgAPL *es un conjunto $E_M = \{-Y_1, \ldots, -Y_m, +X_1, \ldots, +X_n\}$ con $n \geq 0$, $m \geq 0$. Cada X_i e Y_j de E_M representa un elemento a agregar y remover de los componentes mentales de un agente respectivamente, con X_i e Y_j reglas rebatibles tipadas de una base de creencias o una base de metas.*

Definición 6.9 (Conjunto de efectos perceptivos) *Un conjunto E_P de literales tipados es un efecto perceptivo* ArgAPL *si y solo si $E_P = \{L_0^P, \ldots, L_k^P\}$ con $k \geq 0$.*

A continuación se mostrará cómo se utilizarán estos conjuntos para definir las acciones con las que cuenta el agente. Más adelante se mostrará cómo estos conceptos son utilizados para definir los efectos de un plan en las reglas de selección de planes.

Acciones Mentales

Las acciones mentales en ArgAPL son similares a las acciones de 3APL introducidas en el Capítulo 2. Representan un componente de cambio introspectivo, ya que solo cambian el conocimiento interno del agente. Sin embargo, a diferencia de las acciones de 3APL, las acciones mentales en ArgAPL podrán cambiar tanto la base de creencias como la

base de metas, permitiendo así el modelado de acciones de actualización de metas como en [dBHvdHM07, Das08].

Por lo tanto, una acción mental en ArgAPL actualizará la base de creencias o la base de metas del agente cuando sea ejecutada. Este tipo de acción puede ser utilizada para almacenar información recibida de otros agentes, como también para remover metas no deseadas, cambiar el comportamiento del agente, o almacenar datos temporales resultantes de algún cómputo. Estas acciones, al igual que en 3APL, son especificadas en términos de precondiciones y efectos mentales. Un agente podrá ejecutar estas acciones si cree en las precondiciones y, como se verá más adelante, se utilizará la función FAcon() para determinar cómo se aplicarán los efectos mentales una vez ejecutada la acción.

Definición 6.10 (Acción Mental ArgAPL) *Una acción mental (o ma) es una tupla* $ma = (\beta, Name, E_M)$*, donde* β *es un conjunto de precondiciones,* $Name$ *es un átomo utilizado para referenciar a ma desde un plan, y* E_M *es un conjunto de efectos metales. El conjunto todas las acciones mentales disponibles para el agente será denotado como* S_{ma}*.*

Para una mejor visualización de estas acciones, una acción mental $Ma{=}(\beta, Name, E_M)$ se notará como:

$$Name = \begin{cases} \{\beta \\ \{E_M \end{cases}$$

Ejemplo 6.8 *Considere el agente que se desenvuelve en el dominio presentado en la Sección 6.1.1. Este agente contará con dos acciones mentales (denotadas por el conjunto* S_{ma1}*) para cambiar de modo agresivo a modo defensivo y viceversa.*

$$S_{ma1} = \left\{ \begin{array}{l} agresivo = \begin{cases} \{modo(defensivo)^B\} \\ \begin{cases} -(modo(defensivo)^B \prec), \\ -(presaFacil(A)^B \prec rival(A)^B, tiene(A, tesoro(T))^B, cansado(A)^B), \\ -(peligrosa(I)^B \prec en(I, A)^B, rival(A)^B), \\ +(modo(agresivo)^B \prec), \\ +(presaFacil(A)^B \prec rival(A)^B, tiene(A, tesoro(T))^B) \end{cases} \end{cases} \\ \\ defensivo = \begin{cases} \{modo(agresivo)^B\} \\ \begin{cases} -(modo(agresivo)^B \prec), \\ -(presaFacil(A)^B \prec rival(A)^B, tiene(A, tesoro(T))^B), \\ +(modo(defensivo)^B \prec), \\ +(presaFacil(A)^B \prec rival(A)^B, tiene(A, tesoro(T))^B, cansado(A)^B), \\ +(peligrosa(I)^B \prec en(I, A)^B, rival(A)^B) \end{cases} \end{cases} \end{array} \right\}$$

La primer acción indica que, para cambiar a modo agresivo, el agente necesita creer que está en modo defensivo; luego el agente removerá el modo defensivo, así como también las reglas de la base de creencias que denotan que una presa fácil es un agente rival cansado que posee un tesoro, y que una isla es peligrosa si tiene un agente rival, para agregar que ahora está en modo agresivo y que una presa fácil será cualquier agente rival que posea un tesoro. Observe que la segunda acción es utilizada para cambiar de modo agresivo a modo defensivo, agregando lo que remueve la primer acción mental y quitando lo que esta acción agrega.

Acciones Externas

Las acciones externas en ArgAPL son acciones que el agente utiliza para interactuar con el ambiente en el que se desenvuelve. A diferencia de las acciones mentales, en lugar de producir una actualización sobre las bases de conocimiento del agente, las acciones externas actualizan el mundo. Por lo tanto, estas acciones afectarán a las percepciones futuras del agente. Esto resulta una diferencia fundamental con las acciones externas de 3APL, ya que en 3APL no se modela el concepto de percepción y, por lo tanto, estas acciones sólo son utilizadas para obtener información del entorno.

En ArgAPL las acciones externas, de manera similar a las acciones mentales, serán especificadas en términos de pre y post condiciones. En particular, las post condiciones corresponderán a efectos perceptivos. Las precondiciones y los efectos perceptivos de las acciones externas serán determinados por el entorno, a diferencia de lo ocurrido con las acciones mentales en las que estos elementos son arbitrarios. Los efectos perceptivos de estas acciones no siempre se sabrán certeros, ya que pueden depender de factores aleatorios o dinámicos del ambiente. Al igual que para las acciones mentales, un agente ArgAPL podrá ejecutar estas acciones si cree en las precondiciones. Sin embargo, al ejecutarlas, en lugar de cambiar los componentes mentales del agente, afectarán su futura percepción. Más adelante, cuando se presente la semántica operacional del agente, se estudiará cómo se determinan los efectos de estas acciones.

Definición 6.11 (Acción Externa ArgAPL) *Una acción externa (o ea) es una tupla* $Ea = (\beta, Name, E_P)$*, donde* β *es un conjunto de precondiciones,* $Name$ *es un átomo utilizado para referenciar a* Ea *desde un plan, y* E_P *es un conjunto de efectos perceptivos. El conjunto de todas las acciones externas disponibles para el agente será denotado* S_{ea}*.*

Note que, a diferencia de las acciones mentales que agregan y remueven información de las bases de conocimiento del agente, las acciones externas sólo expresan los efectos que producirá la acción en la percepción. Es decir, especifican los literales que serán agregados a la percepción en el próximo ciclo perceptivo si la acción se ejecuta con éxito.

Para una mejor visualización de estas acciones, una acción externa $Ea = (\beta, Name, E_P)$ se notará como:

$$Name = \begin{cases} \{\beta \\ \{E_P \end{cases}$$

Ejemplo 6.9 *Considere el agente que se desenvuelve en el dominio presentado en la Sección 6.1.1. Como se mencionó en esa sección, el agente podrá viajar de una isla a otra, levantar un tesoro, atacar a otro agente o recargar energía.*

$$S_{ea1} = \left\{ \begin{array}{l} viajar(I_o, I_d) = \begin{cases} \{en(I_o, yo)^B, dist(I_o, I_d, D)^B, energia(yo, X)^B, X \geq D\}, \\ \{en(I_d, yo)^P, energia(yo, X - D)^P\} \end{cases} \\ levantar(tesoro(T)) = \begin{cases} \{en(I, yo)^B, en(I, tesoro(T))^B\}, \\ \{tiene(yo, tesoro(T))^P\} \end{cases} \\ atacar(Ag) = \begin{cases} \{en(I, yo)^B, en(I, Ag)^B, energia(yo, X)^B, X \geq 1\}, \\ \{energia(yo, X - 1)^P\} \end{cases} \\ descansar = \begin{cases} \{en(I, yo)^B, en(I, refugio)^B\}, \\ \{energia(yo, 100)^P\} \end{cases} \end{array} \right\}$$

La primer acción externa denota que para viajar de la isla I_o a la isla I_d el agente tiene que creer que está en la isla I_o, que la distancia entre I_o e I_d es D, y que tiene una energía suficiente para recorrer la distancia D; entonces el realizar ese viaje llevará al agente a percibir que ahora está en la isla I_d y que su energía se disminuyó en D unidades. La tercer acción externa expresa que, para atacar a un agente "Ag", el agente "yo" tiene que estar en la misma isla que "Ag" y poseer energía suficiente para realizar el ataque (i.e., debe tener al menos una unidad de energía); el resultado del ataque que percibirá el agente "yo" será únicamente la disminución de su energía en una unidad. Note que no será posible saber cómo quedó el nivel de energía del agente "Ag" luego del ataque, ya que esto es determinado por un factor de azar que depende del entorno.

Observe que la primer acción externa del Ejemplo 6.9 expresa que se agregará el literal "$en(Id, yo)$" a la próxima percepción, sin remover la ubicación anterior del agente. Esto se debe a que los efectos de las acciones externas se expresan en términos de actualización de percepciones. Es decir, en caso de conflictos entre la nueva percepción y una anterior, la nueva percepción prevalecerá. Como se mencionó anteriormente, la actualización de percepciones será presentada formalmente cuando se introduzca la dinámica de los agentes ArgAPL.

Planes

Como se vio en el Capítulo 2, los planes brindan a un agente los medios para alcanzar sus metas. En ArgAPL los planes para un agente constituirán secuencias de acciones.

Definición 6.12 (Plan) *Sean S_{ma} y S_{ea} los conjuntos de acciones mentales y acciones externas de un agente* ArgAPL *respectivamente. Un plan π para ese agente* ArgAPL *es una secuencia $\pi = [a_1, \ldots, a_n]$, donde $n \geq 0$ y todo $a_i \in (S_{ma} \cup S_{ea})$.*

Al igual que en 3APL, en ArgAPL no se especifica cómo se obtienen estos planes. En general, se asumirá que los planes son especificados por el desarrollador. Sin embargo, note que dada la representación de las acciones que pueden componer un plan, también sería posible utilizar un planificador automático. No obstante, el origen de los planes de los agentes ArgAPL está fuera de los alcances de esta tesis.

Note que, a diferencia de las acciones, no se define un conjunto con todos los planes con los que contará un agente ArgAPL. Esto se debe a que los planes se especificarán directamente con las reglas de selección de planes, como se verá a continuación.

6.2.5. Reglas de Selección de Planes

Para poder decidir cómo actuar, los agentes ArgAPL utilizarán reglas de selección de planes al igual que en 3APL. Estas reglas establecerán un mapeo entre metas y planes. Básicamente, una regla de selección de planes determinará el plan a ejecutar con el objetivo de alcanzar una meta que el agente quiere perseguir actualmente. Para determinar si la regla es aplicable, deberán cumplirse un conjunto de precondiciones establecidas por la regla. Adicionalmente, en ArgAPL estas reglas tendrán un componente extra: los efectos

que producirá la ejecución del plan. Estos efectos serán caracterizados por un par que identifica efectos perceptivos y efectos mentales.

Definición 6.13 (Regla de Selección de Planes) *Una regla de selección de planes de un agente* ArgAPL *tiene la forma* $\kappa \leftarrow \beta \mid \pi : (E_P, E_M)$*, donde* κ *es una meta de logro,* β *es un conjunto de precondiciones,* π *es un plan,* E_P *es un conjunto de efectos perceptivos, y* E_M *un conjunto de efectos mentales. El conjunto de todas las reglas de selección de plan se denotará como* $\mathcal{R}$*.*

Ejemplo 6.10 *Considere el agente que se desenvuelve en el dominio presentado en la Sección 6.1.1. Teniendo en cuenta las acciones mentales del Ejemplo 6.8, las acciones externas del Ejemplo 6.9, y sean* $E_{Magresivo}$ *y* $E_{Mdefensivo}$ *los efectos mentales de las acciones mentales "agresivo" y "defensivo" respectivamente, considere el siguiente conjunto* $\mathcal{R}_1$ *de reglas de selección de planes:*

$$\mathcal{R}_1 = \left\{ \begin{array}{l} en(I_d, yo)^{AG} \leftarrow \{en(I_o, yo)^B, dist(I_o, I_d, D)^B, energia(yo, E)^B\} \mid \\ \qquad [viajar(I_o, I_d)] : (\{en(I_d, yo)^P, energia(yo, E-D)^P\}, \emptyset) \\ tiene(yo, tesoro(X))^{AG} \leftarrow \{en(I, tesoro(X))^B, en(I, yo)^B\} \mid [levantar(tesoro(X))] : \\ \qquad (\{tiene(yo, tesoro(X))^P\}, \emptyset) \\ {\sim}cansado(yo)^{AG} \leftarrow \{en(I, yo)^B, en(I, refugio)^B\} \mid [descansar] : (\{energia(yo, 100)^P\}, \emptyset) \\ {\sim}cansado(yo)^{AG} \leftarrow \{en(I_y, yo)^B, en(I_r, refugio)^B, I_y \neq I_r\} \mid \\ \qquad [viajar(I_y, I_r), descansar] : (\{energia(yo, 100)^P, en(I_r, yo)^P\}, \emptyset) \\ atacar(Ag)^{AG} \leftarrow \{en(I, yo)^B, en(I, Ag)^B, energia(yo, E)^B\} \mid [atacar(Ag)] : (\{energia(yo, E-1)^P\}, \emptyset) \\ atacar(Ag)^{AG} \leftarrow \{en(I_y, yo)^B, en(I_a, Ag)^B, I_y \neq I_a, dist(I_y, I_a, D)^B, energia(yo, E)^B\} \mid \\ \qquad [viajar(I_y, I_a), atacar(Ag)] : (\{en(I_a, yo)^P, energia(yo, E-D-1)^P\}, \emptyset) \\ modo(defensivo)^{AG} \leftarrow \{modo(agresivo)^B\} \mid [defensivo] : (\emptyset, E_{Mdefensivo}) \\ modo(agresivo)^{AG} \leftarrow \{modo(defensivo)^B\} \mid [agresivo] : (\emptyset, E_{Magresivo}) \end{array} \right\}$$

La primer regla expresa que si el agente quiere estar en la isla I_d *y cree que está en la isla* I_o*, ejecutará un plan que consiste en viajar de* I_o *a* I_d*. La cuarta regla expresa que si el agente quiere tener su energía al máximo y cree que no está en la isla donde se encuentra el refugio, debe ejecutar un plan para primero viajar a la isla donde está el refugio y luego descansar. La sexta regla denota que si el agente "yo" quiere atacar a otro agente "Ag" y Ag no está en la misma isla que él, debe ejecutar un plan en donde primero viaje a la isla en la que está "Ag" y luego lo ataque. Las últimas dos reglas son utilizadas para generar el cambio de modo en el agente. Por lo tanto, tienen como plan efectuar la acción mental correspondiente, y como efecto únicamente el efecto mental producido por la acción.*

Observación 6.1 *Como se puede observar en la Definición 6.13 y el Ejemplo 6.10, las reglas de selección de planes contienen un componente para indicar los efectos que producirá la ejecución de ese plan. En esta tesis se asumirá que estos efectos, al igual que para*

las acciones mentales y externas, son provistos ya sea por el desarrollador o por un sistema externo. Aun así, en la literatura para metas de mantenimiento se puede notar que existen varías aproximaciones para determinar de manera automática los efectos finales de un plan [HvR07, DHT06]. Más adelante, cuando se muestre la semántica de ejecución del agente, se estudiará cómo utilizar estos efectos para evitar que el agente ponga en riesgo alguna de sus metas de mantenimiento.

6.2.6. Agente ArgAPL

Como en la mayoría de los lenguajes de programación de agentes declarativos, programar un agente ArgAPL implicará especificar las bases de conocimiento que formarán sus componente mentales, los literales que están en conflicto, las acciones con las que contará, y el conjunto de reglas de selección de planes que podrá utilizar.

Definición 6.14 (Especificación de un agente ArgAPL) *La especificación de un agente* ArgAPL *Ag será un tupla* $Ag = (\Delta^B, \Delta^G, \oslash, S_{ma}, S_{ea}, \mathcal{R})$*, donde* Δ^B *es una base de creencias,* Δ^G *es una base de metas,* $\oslash$ *es un conjunto de literales en conflicto,* S_{ma} *es el conjunto de todas las acciones mentales utilizables por Ag,* S_{ea} *es el conjunto de todas las acciones externas utilizables por Ag, y* $\mathcal{R}$ *es el conjunto de todas las reglas de selección de planes disponibles para Ag.*

Ejemplo 6.11 *Considerando el dominio presentado en la Sección 6.1.1, se puede especificar el agente* ArgAPL $A_1 = (\Delta_1^B, \Delta_1^G, \oslash_1, S_{ma1}, S_{ea1}, \mathcal{R}_1)$*, donde* Δ_1^B *es la base de creencias del Ejemplo 6.2,* Δ_1^G *es la base de metas presentada en el Ejemplo 6.6,* $\oslash_1$ *es el conjunto de literales en conflicto presentado en el Ejemplo 6.7,* S_{ma1} *es el conjunto de acciones mentales introducido en el Ejemplo 6.8,* S_{ea1} *es el conjunto de acciones externas presentadas en el Ejemplo 6.9, y* $\mathcal{R}_1$ *es el conjunto de reglas de selección de planes especificado en el Ejemplo 6.10.*

Observe que un agente ArgAPL no contiene el conjunto de percepciones en su especificación. Esto se debe a que, como se mencionó al introducir las percepciones, estas corresponden al modelo dinámico y no son explícitamente especificadas por el desarrollador. En la siguiente sección, cuando se presenten los estados por los que puede pasar un agente en su modelo dinámico, se mostrará cómo se utiliza el conjunto de percepciones.

6.3. Semántica de ArgAPL

En la sección anterior se definieron los componentes que son utilizados para especificar un agente ArgAPL. En esta sección se especificará qué significa ejecutar un agente. En este contexto, se explicará qué implica el uso de cada uno de sus componentes y cómo se relacionan entre sí. Por lo tanto, se presentará la semántica formal de los agentes ArgAPL.

Dado que la semántica de los agentes está totalmente ligada a su dinámica, se introducirá el concepto de configuración ArgAPL, o estado en el cual un agente puede estar durante su ejecución. A partir de un estado, se mostrará cómo el agente razonará para determinar sus creencias y metas en ese estado. Luego, teniendo en cuenta las metas y creencias en un determinado estado, se mostrará cómo al ejecutar acciones o elegir planes el agente pasará de estados a través de un sistema de transiciones.

Configuraciones ArgAPL

Las configuraciones en ArgAPL, al igual que en 3APL, serán utilizadas para representar el estado de los componentes que representan a un agente en cada momento de su ejecución. Es decir, una configuración para un momento en particular de la ejecución del agente denotará una imagen de sus componentes mentales en ese momento. De esta manera, las configuraciones de un agente estarán caracterizadas únicamente por aquellos componentes que puedan variar durante su ejecución. Por lo tanto, las reglas de razonamiento, las acciones mentales, las acciones externas y el conjunto de literales en conflicto (presentes en la especificación inicial del agente) no formarán parte de una configuración. Entonces, una configuración estará caracterizada por un conjunto de percepciones actuales, una base de creencias, una base de metas y una base de planes. Esta última base indicará el plan que el agente está ejecutando y la meta que lo motivó.

Definición 6.15 (Configuración ArgAPL) *Una configuración de un agente* ArgAPL *Ag será un tupla* $C = \langle \Theta^P, \Delta^B, \Delta^G, \Pi \rangle$*, donde* Θ^P *es un conjunto de percepciones,* Δ^B *es una base de creencias,* Δ^G *es una base de metas, y* Π *es una base de planes constituida por un par* (π, κ) *con* π *un plan y* κ *una meta de logro, o bien por* ϵ*.*

Observe que, al igual que como se mostró para 3APL en el Capítulo 2, en ArgAPL se utilizará ϵ para denotar que la base de planes es vacía. Note también que el agente

podrá ejecutar un solo plan por vez, dado que su base de planes no permite el almacenamiento de múltiples planes.

A diferencia de 3APL, en ArgAPL se incluye el conjunto de percepciones en la configuración de un agente[1]. Como se verá mas adelante, manejar las percepciones como un componente de la configuración permitirá modelar su dinámica de manera concreta y modular. Adicionalmente, esto resultará beneficioso para detectar si un plan viola las metas de mantenimiento.

Ejemplo 6.12 *Considere el agente que se desenvuelve en el dominio introducido en la Sección 6.1.1. Suponga que el agente que se encuentra en la situación ilustrada por la Figura 6.1 y que actualmente no está ejecutando ningún plan. Una configuración posible en este estado sería* $(\Theta_1^P, \Delta_1^B, \Delta_1^G, \epsilon)$*, donde* Δ_1^B *es la base de creencias del Ejemplo 6.2,* Δ_1^G *es la base de metas del Ejemplo 6.6, y* Θ_1^P *es el conjunto de percepciones presentado en el Ejemplo 6.3 (el cual caracteriza la situación ilustrada por el rectángulo con borde punteado en la Figura 6.1).*

Inicialmente, cuando el agente comienza su ejecución, su estado estará determinado por la configuración inicial. Esta configuración contendrá el conjunto de percepciones iniciales, las bases de creencias y metas especificadas inicialmente para el agente, y la base de planes vacía.

Definición 6.16 (Configuración Inicial ArgAPL) *Sea* $Ag = (\Delta_0^B, \Delta_0^G, \oslash, S_{ma}, S_{ea}, \mathcal{R})$ *la especificación de un agente* ArgAPL *y* Θ_0^P *un conjunto de percepciones. Si* Θ_0^P *es el primer conjunto de percepciones que recibe* Ag*, entonces la configuración inicial de* Ag *será* $(\Theta_0^P, \Delta_0^B, \Delta_0^G, \epsilon)$*.*

6.3.1. Razonamiento en ArgAPL

El modelo de razonamiento describe la semántica de las creencias, metas de logro, y metas de mantenimiento de los agentes ArgAPL. Es decir, este modelo determina cómo y en qué momento el agente cree, busca o quiere mantener cierto literal, a partir de una

[1]Los autores de 3APL se abstraen del manejo dinámico de las percepciones y lo dejan librado al manejo del desarrollador utilizando acciones mentales y la base de creencias.

configuración del agente. Por lo tanto, el modelo de razonamiento será el que establezca qué cree, qué persigue y qué busca mantener el agente un determinado momento.

Como se mencionó en la introducción de este capítulo, el razonamiento en ArgAPL se efectuará haciendo uso del mecanismo argumentativo de T-DeLP. De esta manera, aquellos literales garantizados constituirán las metas y creencias actuales del agente en una configuración. A continuación se verá cómo construir el programa T-DeLP para obtener esas garantías.

Como se vio en el Capítulo 5 un programa T-DeLP consta de tres componentes: un conjunto de hechos tipados, un conjunto de reglas rebatibles tipadas y una signatura que establece las relaciones entre los distintos tipos de argumentos. Dada una configuración, los hechos del programa estarán determinados por el conjunto de percepciones de la configuración, y las reglas rebatibles tipadas será la unión de las bases de creencias y metas de la configuración. De esta manera, el agente tendrá toda la información del estado actual para efectuar su razonamiento.

La signatura del programa describirá todas las relaciones entre percepciones, creencias, metas de logro y metas de mantenimiento. Como se vio en el Capítulo 5, una signatura $(\mathbb{T}, \rhd, \otimes, >_T, \mathcal{TP})$ está caracterizada por: un conjunto de identificadores de tipo $\mathbb{T}$, una relación de herencia $\rhd$ entre estos identificadores, una relación de desacuerdo $\otimes$ entre literales tipados con esos identificadores de tipo, una relación de preferencia $>_T$ entre los identificadores de tipo, y un conjunto de tipos propagables $\mathcal{TP}$. A continuación se especificará cómo determinar cada uno de estos componentes en un agente ArgAPL.

Los identificadores de tipo para un agente ArgAPL serán fijos. Básicamente, este conjunto contendrá todos los identificadores de tipo introducidos en la Sección 6.2, es decir, el conjunto $\{P, B, G, AG, MG\}$. Como se mencionó anteriormente, estos identificadores de tipo son utilizados para distinguir la información de los diferentes componentes mentales del agente. Más allá de su utilidad a nivel de representación, estos tipos serán los que determinen los tipos de los argumentos con los que razonará el agente ArgAPL.

La relación de herencia en ArgAPL contempla por una parte que, como fue mencionado anteriormente, todas las percepciones son creencias (*i.e.*, $P \rhd B$) y, por otra parte, que tanto las metas de logro como las metas de mantenimiento son metas (*i.e.*, $AG \rhd G$ y $MG \rhd G$). Esto permitirá aprovechar el concepto de conformidad al momento de construir de los argumentos del agente. Por ejemplo, considere el conjunto de percepciones Θ_1^P presentado en el Ejemplo 6.3 y la base de creencias Δ_1^B del Ejemplo 6.2. Note que si

$P \triangleright B$, entonces teniendo en cuenta que Δ_1^B y Θ_1^P contienen, entre otros, a los siguientes elementos:

$$\begin{array}{cc} \Delta_1^B & \Theta_1^P \\ promisoria(I)^B \prec en(I, tesoro(X))^B & en(i1, ag2)^P, \\ peligrosa(I)^B \prec en(I, Ag)^B, rival(Ag)^B & en(i2, tesoro(t1))^P, \\ rival(Ag)^B \prec Ag \neq yo & en(i1, tesoro(t4))^P \end{array}$$

se podrían construir argumentos para $promisoria(i1)^B$, $promisioria(i2)^B$, $peligrosa(i1)^B$. Esto se debe a que tanto "$en(i2, tesoro(t1))^P$" como "$en(i1, tesoro(t4))^P$" conforman con "$en(I, tesoro(X))^B$", y "$en(i1, ag2)^P$" conforma con "$en(I, Ag)^B$". De esta manera, la herencia entre estos tipos resultará fundamental para la construcción de los diferentes tipos de argumentos en ArgAPL.

Definición 6.17 (Identificadores de tipo y Herencia ArgAPL) *El conjunto de identificadores de tipo* ArgAPL *es* $\mathbb{T} = \{P, B, G, AG, MG\}$*, donde* P *representa percepciones,* B *creencias,* G *metas,* AG *metas de logro y* MG *metas de mantenimiento. La relación de herencia* ArgAPL *es* $\triangleright = \{(P, B),\ (AG, G), (MG, G)\}$.

La relación de desacuerdo $\otimes$, requiere un análisis más profundo. En primer lugar, como toda relación de desacuerdo T-DeLP, cada par de literales complementarios del mismo tipo estarán en desacuerdo. Por lo tanto, por ejemplo, $promisoria(i1)^B$ y $\sim promisoria(i1)^B$ están en desacuerdo. Esto es:

$$L^X \otimes \overline{L}^X \text{ con } X \in \{P, B, G, AG, MG\}$$

En segundo lugar, como se vio al momento de especificar el agente ArgAPL, el conjunto de literales en conflicto $\oslash$ (ver Definición 6.6) determina ciertos literales que, a pesar de no ser complementarios, deberían estar en desacuerdo. Por lo tanto, estos literales también deberían estar en desacuerdo para todos los tipos de información manejados por el agente. De esta manera:

$$\text{si } (L_1, L_2) \in \oslash, \text{ entonces } L_1^X \otimes L_2^X \text{ con } X \in \{P, B, G, AG, MG\}$$

En tercer lugar, dado de que las percepciones son creencias, todo par de literales en desacuerdo a nivel de creencias también estará en desacuerdo cuando se involucren creencias y percepciones. Por ejemplo, el literal $\sim en(i1, tesoro(t1))^B$ debería en desacuerdo con

el literal $en(i1, tesoro(t1))^P$. Esto es:

$$\text{si } L_1^B \otimes L_2^B, \text{ entonces } L_1^B \otimes L_2^P \text{ y } L_1^P \otimes L_2^B$$

Similar es el caso de las metas, ya que tanto metas de logro como metas de mantenimiento se refieren a metas. Por lo tanto, los literales en desacuerdo a nivel de metas también estarán en desacuerdo entre metas, metas de mantenimiento y metas de logro. Por lo tanto:

$$\text{si } L_1^G \otimes L_2^G, \text{ entonces } L_1^G \otimes L_2^{AG},\ L_1^G \otimes L_2^{MG},\ L_1^{AG} \otimes L_2^G,\ L_1^{AG} \otimes L_2^{MG},\ L_1^{MG} \otimes L_2^G, \text{ y } L_1^{MG} \otimes L_2^{AG}$$

Por último, el desacuerdo más especial que aparece en ArgAPL es el desacuerdo entre creencias (percepciones) y metas de logro. Un agente solo perseguirá aquellas metas de logro que cree no a haber alcanzado. Es decir, un literal creído o percibido estará en desacuerdo con el literal correspondiente a una meta de logro. Entonces:

$$L^B \otimes L^{AG} \text{ y } L^P \otimes L^{AG}$$

Como se verá más adelante, todos estos desacuerdos producirán ataques entre argumentos tanto de un mismo tipo como de tipos diferentes. Note, en particular, que el último caso hará que se produzcan ataques entre argumentos de creencias y metas de logro que concluyan el mismo literal (cada uno con su respectivo tipo). Además, observe que no habrá desacuerdos entre metas de mantenimiento y creencias.

Todos los desacuerdos establecidos caracterizan a la relación de desacuerdo de un agente ArgAPL, como se define a continuación.

Definición 6.18 (Desacuerdo ArgAPL) *Sea* $Ag = (\Delta^B, \Delta^G, \oslash, S_{ma}, S_{ea}, \mathcal{R})$ *la especificación de un agente* ArgAPL *y* $\mathbb{T}$ *el conjunto de identificadores de tipo* ArgAPL. *La relación* $\otimes$ *de desacuerdo de Ag es una relación de desacuerdo para* $\mathbb{T}$ *tal que:*

- *si* $(L_1, L_2) \in \oslash$, *entonces* $(L_1^X, L_2^X) \in \otimes$ *con* $X \in \{P, B, G, AG, MG\}$;
- *si* $(L_1^B, L_2^B) \in \otimes$ *entonces* $(L_1^B, L_2^P), (L_1^P, L_2^B) \in \otimes$;
- *si* $(L_1^G, L_2^G) \in \otimes$ *entonces* $(L_1^G, L_2^{AG}) \in \otimes$ $(L_1^G, L_2^{MG}) \in \otimes$, $(L_1^{AG}, L_2^G) \in \otimes$, $(L_1^{AG}, L_2^{MG}) \in \otimes$, $(L_1^{MG}, L_2^G) \in \otimes$, $(L_1^{MG}, L_2^{AG}) \in \otimes$; *y*

- *cualquier sea* L, $(L^B, L^{AG}), (L^P, L^{AG}) \in \otimes$.

Ejemplo 6.13 *Considere la especificación del agente* A_1 *del Ejemplo 6.11, el cual tiene el conjunto de desacuerdo* $\bigcirc_1$ *presentado en el Ejemplo 6.7, donde se mostró que* $\bigcirc_1$ *contiene, entre otros, los pares* $(en(I_1, Ag), en(I_2, Ag))$ *si* $I_1 \neq I_2$. *Por lo tanto, observe que la relación de desacuerdo* $\otimes_{A_1}$ *de* A_1 *tendrá, entre otros, los siguientes pares:* $(en(I_1, Ag)^B, en(I_2, Ag)^B)$, $(en(I_1, Ag)^P, en(I_2, Ag)^P)$, $(en(I_1, Ag)^G, en(I_2, Ag)^G)$, $(en(I_1, Ag)^{AG}, en(I_2, Ag)^{AG})$, $(en(I_1, Ag)^{MG}, en(I_2, Ag)^{MG})$, *determinados por el primer ítem de la Definición 6.18;* $(en(I_1, Ag)^B, en(I_2, Ag)^P)$, $(en(I_1, Ag)^P, en(I_2, Ag)^B)$ *determinados por el segundo ítem de la Definición 6.18; y* $(en(I_1, Ag)^G, en(I_2, Ag)^{AG})$, $(en(I_1, Ag)^G, en(I_2, Ag)^{MG})$, $(en(I_1, Ag)^{AG}, en(I_2, Ag)^G)$, $(en(I_1, Ag)^{AG}, en(I_2, Ag)^{MG})$, $(en(I_1, Ag)^{MG}, en(I_2, Ag)^G)$, $(en(I_1, Ag)^{MG}, en(I_2, Ag)^{AG})$, *determinados por el tercer ítem de la Definición 6.18. También observe que, por ejemplo,* $(en(I, Ag)^B, en(I, Ag)^{AG}) \in \otimes_{A_1}$ *de acuerdo al último ítem de la Definición 6.18.*

Note que, como $\otimes_{A_1}$ *es una relación de desacuerdo T-DeLP, también estarán en desacuerdo literales complementarios del mismo tipo. Por lo tanto, entonces en* $\otimes_{A_1}$ *estará por ejemplo el par* $(\sim en(I, Ag)^{AG}, en(I, Ag)^{AG})$, *y por el tercer ítem de la Definición 6.18 también estarán los pares* $(\sim en(I, Ag)^{AG}, en(I, Ag)^{MG})$ *y* $(\sim en(I, Ag)^{MG}, en(I, Ag)^{AG})$.

Las preferencias entre los identificadores de tipo en ArgAPL denotarán qué tipos de argumentos son preferidos cuando haya un ataque entre ellos. Como se vio en el Capítulo 5, las preferencias entre tipos son utilizadas para determinar qué ataques entre dos argumentos de diferente tipo resultan en derrotas. Siguiendo esta noción, la relación de preferencia capturará las siguientes intuiciones. En primera medida, las percepciones siempre serán preferidas por sobre las creencias. Esto se debe a que, siguiendo los estándares de lenguajes de programación de agentes, lo que el agente percibe de su entorno es indiscutible. Por otra parte, las metas de mantenimiento serán preferidas a las metas de logro. Esto se debe a que las metas de mantenimiento suelen modelar situaciones importantes que el agente quiere mantener. Por último, las creencias y las percepciones serán preferidas por sobre las metas de logro. Esto se debe a que si hay un ataque entre estos tipos de argumento, implica que el agente cree (percibe) que lo que pretende alcanzar la meta de logro ya ha sido alcanzado y, por lo tanto, el argumento de creencia (percepción) debería derrotar al argumento de meta de logro.

Definición 6.19 (Preferencia de Tipos ArgAPL) *Sea* $\mathbb{T}$ *el conjunto de identificadores de tipo* ArgAPL*. La relación de preferencia de tipos* ArgAPL *es* $>> = \{(P, B), (MG, AG), (P, AG), (B, AG)\}$.

Por último, en ArgAPL no habrá tipos propagables ya que, como se puede observar en la Sección 6.2, todas las reglas rebatibles utilizadas en los agentes están caracterizadas explícitamente por los tipos que derivarán. Es decir, no hay reglas generales que puedan ser aprovechadas para propagar un tipo más específico.

De esta manera, con los elementos arriba descriptos se podrá construir la signatura que identifica a un agente ArgAPL, la cual se define a continuación.

Definición 6.20 (Signatura ArgAPL) *Sea Ag la especificación de un agente* ArgAPL. Γ *será una signatura* ArgAPL *para Ag si y solo si* Γ *es una signatura T-DeLP tal que* $\Gamma = (\mathbb{T}, \rhd, \otimes, >>, \emptyset)$*, donde* $\mathbb{T}$ *es el conjunto de identificadores de tipo* ArgAPL*,* $\rhd$ *es la relación de herencia* ArgAPL*,* $\otimes$ *es la relación de desacuerdo de Ag, y* $>>$ *es la relación de preferencia de tipos* ArgAPL.

Observación 6.2 *Es importante notar que la signatura de un agente* ArgAPL *es estática y depende puramente de la especificación del agente. Por lo tanto, durante toda su ejecución, el agente utilizará la misma signatura para razonar acerca de sus metas o sus creencias.*

Adicionalmente, observe que el único elemento realmente significativo para determinar la signatura de un agente es el conjunto de literales en conflicto que será utilizado para determinar la relación de desacuerdo de la signatura. En contrapartida, los demás elementos de la signatura serán iguales para todos los agentes. Entonces, en aquellos agentes con un conjunto de literales en conflicto vacío, la signatura será siempre la misma. Por lo tanto, los desacuerdos en esa signatura estarán dados por literales complementarios.

Diferente es el caso de la parte de conocimiento del programa T-DeLP que utilizará el agente ArgAPL. Como se mencionó anteriormente, esta parte estará constituida por el conjunto de percepciones, la base de creencias y la base de metas del agente. Estos elementos pueden cambiar de una configuración a otra del agente. Por ejemplo, considerando el dominio introducido en la Sección 6.1.1, el conjunto de percepciones del agente puede cambiar turno a turno, a medida que el u otros agentes se mueven y/o levantan tesoros.

Más aún, el agente podría modificar su base de metas utilizando una acción mental. Claramente, las creencias y metas que el agente infiere en un turno podrían llegar a diferir de las inferidas en otro turno, ya que las bases utilizadas para obtener las inferencias en cada turno son distintas. Es por esto que el programa T-DeLP utilizado por el agente para efectuar el razonamiento en un determinado estado se basará en los elementos de esa configuración.

Definición 6.21 (Programa T-DeLP para una configuración ArgAPL) *Sea Ag la especificación de un agente* ArgAPL*,* Γ *la signatura* ArgAPL *para Ag y* $C = (\Theta^P, \Delta^B, \Delta^G, \Pi)$ *una configuración de Ag. El programa T-DeLP para Ag en C, será* $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$.

Ejemplo 6.14 *Considere el agente* $A_1 = (\Delta_1^B, \Delta_1^G, \oslash_1, S_{ma1}, S_{ea1}, \mathcal{R}_1)$ *del Ejemplo 6.11 y la configuración* $C_1 = (\Theta_1^P, \Delta_1^B, \Delta_1^G, \epsilon)$ *presentada en el Ejemplo 6.12. La signatura para* A_1 *será* $\Gamma_1 = (\mathbb{T}, \rhd, \otimes_1, >>, \emptyset)$ *donde* $\otimes_1$ *es la relación de desacuerdo del Ejemplo 6.13. El programa T-DeLP para* A_1 *en* C_1 *será* $\mathcal{P}(A_1, C_1) = (\Gamma_1, \Theta_1^P, \Delta_1^B \cup \Delta_1^G)$.

Note que, si bien en este caso las bases de creencias y metas utilizadas en el programa coinciden con las de la especificación del agente, el programa T-DeLP asociado utilizará las bases establecidas por la configuración. La coincidencia en este caso se debe a que la configuración actual C_1 *es en particular la configuración inicial, por lo que utiliza las bases correspondientes a la especificación del agente. Además observe que, como se mencionó anteriormente, la signatura se encuentra determinada por la especificación del agente y no por la configuración actual.*

Ahora que se determinó el programa T-DeLP utilizado para razonar en una configuración, será posible definir la semántica de las percepciones, creencias, metas de logro y metas de mantenimiento en ArgAPL. Para ello se empleará la noción de garantía presentada en el Capítulo 5. Como se vio en ese capítulo, para determinar la garantía de un literal se lleva a cabo un proceso de dialéctica donde se consideran los argumentos para ese literal. En particular, se utiliza un criterio de comparación para determinar la preferencia entre dos argumentos de un mismo tipo. Como se vio en el Capítulo 5 este criterio es modular y puede utilizarse el de mayor conveniencia en cada caso. En ArgAPL se seguirá la misma política, con lo cual este criterio será modular y adaptable. Aún así, en los ejemplos que se mostrarán en lo que resta del capítulo se asumirá que se utiliza el criterio de especificidad generalizada [GS04].

6.3.2. Semántica de percepciones y creencias

Los literales que el agente percibe en una determinada configuración serán llamados percepciones actuales. Estas percepciones serán aquellos literales que el agente garantice en esa configuración, describiendo así la semántica de percepciones ArgAPL.

Definición 6.22 (Percepción actual) *Sea Ag la especificación de un agente* ArgAPL*, C una configuración de Ag y $\mathcal{P}(Ag, C)$ el programa T-DeLP de Ag en la configuración C. El literal tipado L^P será una percepción actual para Ag en C, notado $C \vdash_S L^P$, si y solo si $\mathcal{P}(Ag, C)$ garantiza L^P. Un conjunto $PS = \{L_1^P \dots L_n^P\}$ será un conjunto de percepciones actuales en C, notado $C \vdash_S PS$, si y solo si para todo $L_i^P \in PS$ se tiene que $C \vdash_S L_i^P$.*

Note que, en una configuración determinada, el agente garantizará todos los literales del conjunto de percepciones ya que son representados a través de hechos. Por lo tanto, todos los literales en el conjunto de percepciones de esa configuración serán percepciones actuales.

Proposición 6.1 *Sea Ag la especificación de un agente* ArgAPL *y $C = (\Theta^P, \Delta^B, \Delta^G, \Pi)$ una configuración de Ag. Si L^P pertenece a Θ^P, entonces $C \vdash_S L^P$.*

Prueba : Sea $\mathcal{P}^\Gamma = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ el programa T-DeLP asociado a C, con Γ la signatura resultante de la especificación del agente Ag. Como L^P pertenece al conjunto de hechos Θ^P de $\mathcal{P}^\Gamma$, entonces por Proposición 5.13 se tiene que $\mathcal{P}^\Gamma$ garantiza L^P. Luego por Definición 6.22 $C \vdash_S L^P$. □

Ejemplo 6.15 *Considere la configuración C_1 presentada en el Ejemplo 6.12 y su programa T-DeLP asociado presentado en el Ejemplo 6.14. Dado que en C_1 el conjunto de percepciones es $\Theta_1^P = \{en(i3, yo)^P, energia(yo, 95)^P, en(i1, ag2)^P, energia(ag2, 100)^P, en(i1, tesoro(t4))^P, en(i2, tesoro(t1))^P, pos(i1, 1, 1)^P, pos(i2, 2, 4)^P, pos(i3, 3, 2)^P\}$, entonces $C_1 \vdash_S en(i3, yo)^P$, $C_1 \vdash_S energia(ag2, 100)^P$, $C_1 \vdash_S en(i2, tesoro(t1))^P$, y $C_1 \vdash_S pos(i3, 3, 2)^P$, entre otros.*

A partir de una configuración es posible determinar qué es lo que el agente cree en ese estado, conocido como creencias actuales. Básicamente, las creencias actuales serán los literales garantizados del programa T-DeLP asociado a la configuración actual.

Definición 6.23 (Creencia actual) *Sea Ag la especificación de un agente* ArgAPL*, C una configuración de Ag y $\mathcal{P}(A, C)$ el programa T-DeLP de Ag en C. El literal tipado L^B será una creencia actual para Ag en C, notado $C \vdash_S L^B$, si y solo si $\mathcal{P}(A, C)$ garantiza L^B o garantiza L^P. Un conjunto $BS = \{L_1^B \ldots L_n^B\}$ será un conjunto de creencias actuales en C, notado $C \vdash_S BS$, si y solo si para todo $L_i^B \in BS$ vale que $C \vdash_S L_i^B$.*

Ejemplo 6.16 *Considere un agente con el conjunto de literales en conflicto vacío, que se encuentra en la configuración $CL_2 = (\Theta_2^P, \Delta_2^B, \Delta_2^G, \epsilon)$, donde:*

$$\Theta_2^P = \left\{ \begin{array}{l} d^P \\ e^P \end{array} \right\} \quad \Delta_2^B = \left\{ \begin{array}{l} c^B \prec e^B \\ a^B \prec b^B \\ \sim a^B \prec b^B, e^B \\ b^B \prec \\ \sim d^B \prec b^B \end{array} \right\} \quad \Delta_2^G = \left\{ \begin{array}{l} a^{AG} \prec \\ c^{AG} \prec \\ \sim b^{AG} \prec b^B \\ b^{MG} \prec \\ \sim d^{MG} \prec \end{array} \right\}$$

A partir de esta configuración observe que es posible construir los siguientes argumentos representativos para percepciones y creencias:

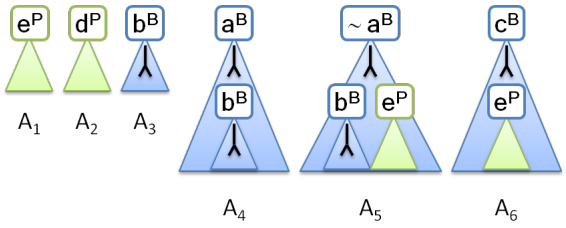


Figura 6.2: Argumentos para percepciones y creencias en CL_2.

Note que el argumento A_5 se construye utilizado un argumento para percepción, dado que el c^P conforma con c^B. Si bien hay argumentos representativos para estos literales, no todos serán creencias actuales en CL_2. Observe que a^B no estará garantizado, dado que el argumento A_4 que lo sustenta es derrotado por A_5, quien lo ataca ($a^B \otimes \sim a^B$) y es preferido ya que son del mismo tipo y A_5 es más específico. Note que los demás argumentos A_1, A_2, A_3, A_5 y A_6 están garantizados. Por lo tanto $CL_2 \vdash_S \{e^B, d^B, b^B, \sim a^B, c^B\}$.

Ejemplo 6.17 *Considere la configuración $C_1 = (\Theta_1^P, \Delta_1^B, \Delta_1^G, \epsilon)$ presentada en el Ejemplo 6.12 y su programa T-DeLP asociado presentado en el Ejemplo 6.14. En C_1 las*

creencias actuales del agente serán: $promisoria(i2)^B$, $\sim promisoria(i1)^B$, $peligrosa(i1)^B$, $\sim cansado(yo)^B$, $\sim cansado(ag2)^B$, $en(i1, refugio)^B$, $rival(ag2)^B$, $modo(defensivo)^B$, $dist(i1, i2, 4)^B$, $dist(i1, i3, 3)^B$, $dist(i3, i2, 3)^B$, $masLejos(i2, i3, ag2)^B$ *y todos los literales que son percepción actual (mostrados en el Ejemplo 6.15). Note que* $promisoria(i1)^B$ *no es una creencia actual porque el argumento representativo que la sustenta es derrotado por el argumento representativo para* $\sim promisoria(i1)^B$.

Como se puede observar en los ejemplos 6.17 y 6.16, todas las percepciones en una configuración de un agente también serán creencias actuales para el agente en esa configuración. Esto se debe a que las percepciones en efecto son creencias.

Proposición 6.2 *Sea Ag la especificación de un agente* ArgAPL *y* $C = (\Theta^P, \Delta^B, \Delta^G, \Pi)$ *una configuración de Ag. Si* $L^P \in \Theta^P$ *entonces* $C \vdash_S L^B$.

Prueba : *Como* $L^P \in \Theta^P$, *por Proposición 6.1 vale que* $C \vdash_S L^P$. *Luego por Definición 6.23 se tiene que* $C \vdash_S L^B$. □

El resultado más importante con respecto a las creencias actuales de un agente ArgAPL es que no garantiza literales de creencias en desacuerdo. Es decir, el agente no creerá en dos literales complementarios o dos literales que se encuentren en el conjunto de literales en conflicto de su especificación. Este resultado es de suma importancia, ya que muestra que aún representado información potencialmente conflictiva en su base de creencias, el conjunto de creencias actuales de una configuración será consistente.

Teorema 6.1 *Sea Ag la especificación de un agente* ArgAPL *con* $\oslash$ *su conjunto de literales en conflicto, y C una configuración de Ag. Si* $C \vdash_S L_1^B$, *entonces* $C \nvdash_S L_2^B$ *tal que* $L_2 = \overline{L_1}$ *o* $(L_1, L_2) \in \oslash$.

Prueba : *Suponga que* $C \vdash_S \overline{L_1}^B$. *Entonces por Definición 6.23 de creencia actual* $\mathcal{P}(Ag, C)$ *garantiza* $\overline{L_1}^B$. *Luego, por hipótesis y Definición 6.23* $\mathcal{P}(Ag, C)$ *también garantiza* L_1^B, *pero por Definición 5.6 de relación de desacuerdo* $\overline{L_1}^B \otimes L_1^B$, *lo cual contradice el Teorema 5.3. Ahora suponga que* $C \vdash_S L_2^B$ *con* L_2 *tal que* $(L_1, L_2) \in \oslash$. *Entonces por Definición 6.23 se tiene que* $\mathcal{P}(Ag, C)$ *garantiza* L_2^B. *Dado que* $\mathcal{P}(Ag, C)$ *también garantiza* L_1^B *y por Definición 6.18 de desacuerdo en* ArgAPL $L_1^B \otimes L_2^B$, *entonces se contradice el Teorema 5.3.* □

La siguiente proposición muestra que las metas del agente no influyen en el cálculo de sus creencias. Por lo tanto, no habrá ninguna meta que invalide una creencia. Este resultado es importante porque, como se mostró en la Definición 6.18, una creencia estará en desacuerdo con una meta de logro para un mismo literal (el agente no debe tratar de alcanzar estado del mundo en el que ya cree que se encuentra).

Proposición 6.3 *Sea* $C = (\Theta^P, \Delta^B, \emptyset, \Pi)$ *una configuración de un agente* ArgAPL *Ag. Si* $C \vdash_S L^B$*, entonces* $C' = (\Theta^P, \Delta^B, \Delta^G, \Pi) \vdash_S L^B$ *cualquiera sea* $\Delta^G \neq \emptyset$*.*

Prueba : Sea $\mathcal{P}(Ag, C)$ *el programa T-DeLP para la configuración C y* $\mathcal{P}(Ag, C')$ *el programa para la configuración* C'*. Por definiciones 6.1, 6.2, 6.3, 6.4, y 5.18 de base de creencias, conjunto de percepciones, base de metas de logro, base de metas de mantenimiento y de argumento representativo respectivamente, en la construcción de los argumentos para creencias y percepciones sólo influyen los elementos de* Δ^B *y* Θ^P*. Por lo tanto, todo argumento para creencias o percepciones que esté en* $\mathcal{P}(A, C)$ *también estará en* $\mathcal{P}(A, C')$*, en particular algún argumento* $\mathcal{A}$ *que garantiza a* L^B *en* $\mathcal{P}(A, C)$*. Por definiciones 6.19 y 6.18 de preferencia y desacuerdo en Ag, un argumento de creencias sólo podrá ser derrotado por otros argumentos de creencias o por argumentos de percepciones. Entonces todos los argumentos representativos en el árbol de dialéctica* $\mathcal{T}^*_{\mathcal{A}}$ *que garantiza a* $\mathcal{A}$ *en* $\mathcal{P}(Ag, C)$ *se construyen solamente con elementos de* Δ^B *y* Θ^P*. Por lo tanto, en* $\mathcal{P}(Ag, C')$ *también estará el árbol* $\mathcal{T}^*_{\mathcal{A}}$*. Luego* $\mathcal{P}(Ag, C')$ *garantiza a* $\mathcal{A}$*, y por lo tanto* $C' \vdash_S L^B$*.* □

Una propiedad adicional que involucra las creencias actuales y las percepciones actuales es que si un agente tiene una percepción actual, entonces no tendrá ninguna creencia actual que la contradiga. Este resultado se formaliza en la siguiente proposición.

Proposición 6.4 *Sea Ag la especificación de un agente* ArgAPL *con* $\oslash$ *su conjunto de literales en conflicto, y C una configuración de Ag. Si* $C \vdash_S L_1^P$*, entonces* $C \nvdash_S L_2^B$ *tal que* $L_2 = \overline{L_1}$ *o* $(L_1, L_2) \in \oslash$*.*

Prueba : Suponga que $C \vdash_S L_2^B$ *tal que* $L_2 = \overline{L_1}$*. Entonces, por Definición 6.23 de creencia actual* $\mathcal{P}(Ag, C)$ *garantiza* L_2^B*. Luego, por Teorema 5.3 y Definición 6.18 de desacuerdo en* ArgAPL*,* $\mathcal{P}(Ag, C)$ *no garantiza* L_1^P *dado que* $L_2^B \otimes L_1^P$ *porque* $L_2 = \overline{L_1}$*. De esta manera,* $C \nvdash_S L_1^B$*, lo cual contradice la hipótesis. Suponga ahora que* $C \vdash_S L_2^B$ *y que* $(L_1, L_2) \in \oslash$*. Por Definición 6.23 de creencia actual* $\mathcal{P}(Ag, C)$ *garantiza* L_2^B*. Por Teorema 5.3 y Definición 6.18 de desacuerdo en* ArgAPL*,* $\mathcal{P}(Ag, C)$ *no garantiza* L_1^P *dado que* $L_2^B \otimes L_1^P$ *porque* $(L_1, L_2) \in \oslash$*. Luego* $C \nvdash_S L_1^P$*, lo cual contradice la hipótesis.* □

6.3.3. Semántica de metas de logro y metas de mantenimiento

La semántica de metas describe qué metas el agente querrá alcanzar (mantener o lograr) en una configuración. Al igual que con las creencias, la semántica para las metas de logro y mantenimiento se determinará a partir de los argumentos garantizados en el programa T-DeLP asociado a una configuración.

En primer lugar se presentará la semántica de las metas de mantenimiento, la cual dará lugar a las metas de mantenimiento actuales en una configuración del agente. La semántica para este tipo de metas es presentada primero dado que, como se verá más adelante, repercutirá en la semántica de las metas de logro.

Definición 6.24 (Meta de mantenimiento actual) *Sea Ag la especificación de un agente* ArgAPL*, C una configuración de Ag y $\mathcal{P}(Ag, C)$ el programa T-DeLP de Ag en C. El literal tipado L^{MG} será una meta de mantenimiento actual para Ag en C, notado $C \vdash_S L^{MG}$ si y solo si $\mathcal{P}(Ag, C)$ garantiza L^{MG}. Un conjunto $MGS = \{L_1^{MG} \dots L_n^{MG}\}$ será un conjunto de metas de mantenimiento actuales en C, notado $C \vdash_S MGS$ si y solo si para todo $L_i^{MG} \in MGS$ vale $C \vdash_S L_i^{MG}$.*

Ejemplo 6.18 *Considere la configuración del Ejemplo 6.16 $CL_2 = (\Theta_2^P, \Delta_2^B, \Delta_2^G, \epsilon)$. A partir de CL_2 el agente tendrá los siguientes argumentos para sus metas, tanto de logro como de mantenimiento:*

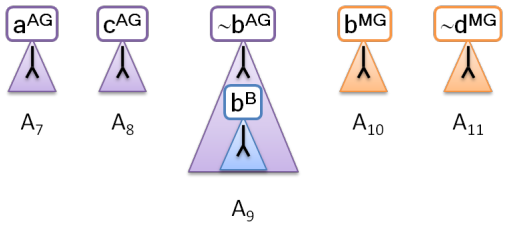


Figura 6.3: Argumentos para metas de logro y de mantenimiento en CL_2.

Observe que hay dos argumentos A_{10} y A_{11} para metas de mantenimiento. En este caso ambos están garantizados. En particular, note que si bien A_9 y A_{10} se atacan mutuamente (dado que $\sim b^{AG} \otimes b^{MG}$), A_9 no derrotará a A_{10} porque $MG>>AG$. Por lo tanto, las metas de mantenimiento actuales en CL_2 serán b^{MG} y $\sim d^{MG}$ ($CL_2 \vdash_S \{b^{MG}, \sim d^{MG}\}$).

Al igual que como se presentó para las creencias actuales del agente, se puede mostrar que el conjunto de metas de mantenimiento actuales para un agente ArgAPL es consistente.

Teorema 6.2 *Sea Ag la especificación de un agente* ArgAPL *con* $\oslash$ *su conjunto de literales en conflicto, y C una configuración de Ag. Si* $C \vdash_S L_1^{MG}$*, entonces* $C \nvdash_S L_2^{MG}$ *tal que* $L_2 = \overline{L_1}$ *o* $(L_1, L_2) \in \oslash$*.*

Prueba : Suponga que $C \vdash_S L_2^{MG}$ *tal que* $L_2 = \overline{L_1}$*. Entonces, por Definición 6.24 de meta de mantenimiento actual el programa* $\mathcal{P}(Ag, C)$ *asociado a C garantiza* L_2^{MG}*. Luego, por Teorema 5.3 y Definición 6.18 de desacuerdo en* ArgAPL *se tiene que* $\mathcal{P}(Ag, C)$ *no garantiza* L_1^{MG} *dado que* $L_2^{MG} \otimes L_1^{MG}$ *porque* $L_2 = \overline{L_1}$*. Entonces,* $C \nvdash_S L_1^{MG}$*, lo cual contradice la hipótesis. Suponga ahora que* $C \vdash_S L_2^{MG}$ *y que* $(L_1, L_2) \in \oslash$*. Por Definición 6.24 se tiene que* $\mathcal{P}(Ag, C)$ *garantiza* L_2^{MG}*. Luego, por Teorema 5.3 y Definición 6.18 de desacuerdo en* ArgAPL*,* $\mathcal{P}(Ag, C)$ *no garantiza* L_1^{MG} *dado que* $L_2^{MG} \otimes L_1^{MG}$ *ya que* $(L_1, L_2) \in \oslash$*. Por lo tanto,* $C \nvdash_S L_1^{MG}$*, lo cual contradice la hipótesis.* $\square$

Otra propiedad que cumplen las metas de mantenimiento actuales, similar a la de las creencias actuales, es que no dependen de las metas de logro. Este resultado es importante porque, como se mostró en la Sección 6.3.1, las metas de logro y las metas de mantenimiento pueden estar en desacuerdo, pero una meta de logro no debería invalidar a una meta de mantenimiento.

Proposición 6.5 *Sea Ag la especificación de un agente* ArgAPL *y* $C = (\Theta^P, \Delta^B, \Delta^{MG}, \Pi)$ *una configuración de Ag, donde* Δ^{MG} *es una base de metas de mantenimiento. Si* $C \vdash_S L^{MG}$*, entonces* $C' = (\Theta^P, \Delta^B, \Delta^{MG} \cup \Delta^{AG}, \Pi) \vdash_S L^{MG}$ *cualquiera sea* Δ^{AG}*.*

Prueba : Sea $\mathcal{P}(Ag, C)$ *el programa T-DeLP para la configuración C y* $\mathcal{P}(Ag, C')$ *el programa para la configuración* C'*. Por definiciones 6.1, 6.2, 6.3, 6.4 y 5.18 de base de creencias, conjunto de percepciones, base de metas de logro, base de metas de mantenimiento y de argumento representativo respectivamente, en la construcción de los argumentos para metas de mantenimiento, creencias y percepciones sólo influyen los elementos de* Δ^{MG}*,* Δ^B *y* Θ^P*. Por lo tanto, todo argumento para metas de mantenimiento, creencias o percepciones que esté en* $\mathcal{P}(Ag, C)$ *también estará en* $\mathcal{P}(Ag, C')$*, en particular algún argumento* $\mathcal{A}$ *que garantiza a* L^{MG} *en* $\mathcal{P}(Ag, C)$*. Luego, por definiciones 6.19, 6.18 de preferencia y desacuerdo en Ag, un argumento de meta de mantenimiento sólo puede ser derrotado por*

argumentos de creencias o de percepciones que lo ataquen internamente, o por argumentos de metas de mantenimiento que lo ataquen directamente. De esta manera, todos los argumentos representativos en el árbol de dialéctica $\mathcal{T}^*_{\mathcal{A}}$ que garantiza a $\mathcal{A}$ en $\mathcal{P}(Ag, C)$ se construyen únicamente con elementos de Δ^{MG}, Δ^B y Θ^P. Por lo tanto, en $\mathcal{P}(Ag, C')$ también esta el árbol $\mathcal{T}^*_{\mathcal{A}}$. Luego $\mathcal{P}(Ag, C')$ garantiza a $\mathcal{A}$, con lo cual $C' \vdash_S L^B$. □

Las metas de mantenimiento actuales de una configuración representarán las creencias que el agente quiere mantener en esa configuración. Sin embargo, esto no implica que el agente actualmente tenga esas creencias, es decir, que los literales que son metas de mantenimiento actuales en la configuración también sean creencias actuales en la configuración. Por lo tanto, las metas de mantenimiento actuales podrán estar en dos situaciones: mantenidas o no mantenidas.

Definición 6.25 (Meta de mantenimiento actualmente mantenida) *Sea Ag la especificación de un agente* ArgAPL *y C una configuración de Ag tal que $C \vdash_S L^{MG}$. La meta de mantenimiento actual L^{MG} estará actualmente mantenida en C si y solo si $C \vdash_S L^B$. En caso contrario, si $C \nvdash_S L^B$, L^{MG} no estará actualmente mantenida en C. El conjunto de todas las metas de mantenimiento actualmente mantenidas en C será denotado* Mantenidas(C)*, y el conjunto de las metas de mantenimiento no mantenidas será denotado* NoMantenidas(C)*.*

La importancia de esta distinción radica en que si el agente tiene una meta de mantenimiento actual no mantenida, debería tratar de alcanzar un estado del mundo donde se alcance la situación especificada por la meta. Observe entonces que, en ese caso, la meta de mantenimiento pasaría a ser una meta de logro para el agente. Esto se verá a continuación cuando se presenten las metas de logro actuales. Otro aspecto importante de esta distinción es que las metas que el agente actualmente está manteniendo deben ser protegidas y, por lo tanto, serán utilizadas para descartar reglas de selección de planes como para cancelar acciones que pongan en juego la creencia actual que el agente trata de mantener. Más adelante, cuando se presente la semántica operacional de estos componentes, se analizará este comportamiento en detalle.

Ejemplo 6.19 *Considere la configuración CL_2 presentada en el Ejemplo 6.16. Como se vio en el Ejemplo 6.18, CL_2 tiene como metas de mantenimiento actuales a $\sim d^{MG}$ y*

b^{MG}. *Por otra parte, en el Ejemplo 6.16 se mostró que en esa configuración el agente tiene como creencias actuales* e^B, d^B, b^B, $\sim a^B$ *y* c^B. *Por lo tanto,* $\mathsf{Mantenidas}(CL_2) = \{b^{MG}\}$ *y* $\mathsf{NoMantenidas}(CL_2) = \{\sim d^{MG}\}$.

Ejemplo 6.20 *Considere la configuración* $C_1 = (\Theta_1^P, \Delta_1^B, \Delta_1^G, \epsilon)$ *presentada en el Ejemplo 6.12 y su programa T-DeLP asociado presentado en el Ejemplo 6.14. En* C_1 *la única meta de mantenimiento actual del agente será* $\sim cansado(yo)^{MG}$. *En particular, será una meta de mantenimiento actualmente mantenida ya que, como se vio en el Ejemplo 6.17,* $\sim cansado(yo)^B$ *es una creencia actual del agente en* C_1.

A continuación se presentará la semántica para las metas de logro. Concretamente, las metas de logro actuales para una configuración serán aquellas metas de logro que se puedan garantizar del programa T-DeLP asociado a esa configuración pero, adicionalmente, también serán metas de logro aquellas metas de mantenimiento que el agente busca pero que no mantiene actualmente. Este último comportamiento sigue el espíritu reactivo de las metas de mantenimiento, como se presenta en [vRDW08], y en [PBL05].

Definición 6.26 (Meta de logro actual) *Sea Ag la especificación de un agente* ArgAPL, *C una configuración de Ag y* $\mathcal{P}(C, Ag)$ *el programa T-DeLP de A en C. El literal tipado* L^{AG} *será una meta de logro actual para Ag en C, notado* $C \vdash_S L^{AG}$, *si y solo si* $\mathcal{P}(C, Ag)$ *garantiza* L^{AG}, *o* $\mathcal{P}(C, Ag)$ *garantiza* L^{MG} *pero no garantiza* L^B. *Un conjunto* $AGS = \{L_1^{AG} \ldots L_n^{AG}\}$ *será un conjunto de metas de logro actuales en C, notado* $C \vdash_S AGS$, *si y solo si para todo* $L_i^{AG} \in AGS$ *vale* $C \vdash_S L_i^{AG}$.

Ejemplo 6.21 *Considere la configuración* CL_2 *presentada en el Ejemplo 6.16, los argumentos para las metas de logro de y mantenimiento de* CL_2 *ilustrados en la Figura 6.3, y los argumentos para creencias y percepciones de* CL_2 *presentados en la Figura 6.2. Observe que* c^{AG} *no será una meta de logro en* CL_2, *ya que el argumento* A_8 *que lo sustenta está derrotado por el argumento* A_6 *para* c^B. *Esto ocurre porque* A_6 *ataca a* A_8 ($c^B \otimes c^{AG}$) *y* A_6 *es preferido a* A_8 *ya que* $B >> AG$ *(Recuerde que un agente no buscará lograr aquellas metas que ya cree logradas). Note también que* $\sim b^{AG}$, *a pesar de estar sustentado por el argumento* A_9, *no será una meta de logro actual en* CL_2. *Esto se debe a que el argumento* A_{10} *lo ataca* ($b^{MG} \otimes \sim b^{AG}$) *y lo derrota dado que* $MG >> AG$. *Por lo tanto, las metas de logro actuales en* CL_2 *serán* a^{AG}, *dado que su argumento* A_7 *está no derrotado, y* $\sim d^{AG}$, *porque como se vio en el Ejemplo 6.19 es una meta de mantenimiento no mantenida.*

Ejemplo 6.22 *Considere la configuración* $C_1 = (\Theta_1^P, \Delta_1^B, \Delta_1^G, \epsilon)$ *presentada en el Ejemplo 6.12 y su programa T-DeLP asociado presentado en el Ejemplo 6.14. En* C_1 *las metas de logro actuales del agente serán:* $en(yo, i2)^{AG}$*, y* $modo(agresivo)^{AG}$*. Esto se debe a que, como se vio en el Ejemplo 6.17,* $promisoriai2^B$ *y* $energia(yo, 95)^B$ *son creencias actuales del agente en* C_1*.*

Ejemplo 6.23 *Considere que el agente del Ejemplo 6.11 se encuentra en una configuración* C_{1a}*, la cual difiere de* C_1 *en que el agente se encuentra en la isla* $i2$ *(donde había un tesoro) en lugar de estar en la isla* $i3$*, es decir:* $C_{1a} = (\Theta_1^P a, \Delta_1^B, \Delta_1^G, \epsilon)$*, donde* $\Theta_1^P a = \Theta_1^P \setminus \{en(i3, yo)^P\} \cup \{en(i2, yo)^P\}$*. En* C_{1a} *el agente tendrá como metas de logro actuales* $tiene(tesoro(t1), yo)^{AG}$ *y* $modo(agresivo)^{AG}$*. En este caso, las metas de mantenimiento actuales del agente serán* $en(i2, yo)^{MG}$ *y* $\sim cansado(yo)^{MG}$*, las cuales están actualmente mantenidas. Observe que si bien el agente podría llegar a tener como meta de logro* $en(i2, yo)^{AG}$*, esto no sucederá ya que en la configuración* C_{1a} $en(i2, yo)^P$ *es una percepción actual del agente (i.e., se trata de una meta que el agente ya alcanzó).*

Al igual que para las creencias y metas de mantenimiento, un resultado importante con respecto a la semántica de las metas de logro es que el conjunto de las metas de logro actuales de una configuración será consistente.

Teorema 6.3 *Sea Ag la especificación de un agente* ArgAPL *con* $\oslash$ *su conjunto de literales en conflicto, y C una configuración de Ag. Si* $C \vdash_S L_1^{AG}$*, entonces* $C \nvdash_S L_2^{AG}$ *tal que* $L_2 = \overline{L_1}$ *o* $(L_1, L_2) \in \oslash$*.*

Prueba : Suponga que $C \vdash_S L_2^{AG}$ *tal que* $L_2 = \overline{L_1}$*. Entonces, por Definición 6.26 de meta de logro actual el programa asociado* $\mathcal{P}(Ag, C)$ *garantiza* L_2^{AG}*. Luego, por Teorema 5.3 y Definición 6.18 de desacuerdo en* ArgAPL*,* $\mathcal{P}(Ag, C)$ *no garantiza* L_1^{AG} *dado que* $L_2^{AG} \otimes L_1^{AG}$ *porque* $L_2 = \overline{L_1}$*. Entonces* $C \nvdash_S L_1^{AG}$*, lo cual contradice la hipótesis. Suponga ahora que* $C \vdash_S L_2^{AG}$ *y que* $(L_1, L_2) \in \oslash$*. Por Definición 6.26,* $\mathcal{P}(Ag, C)$ *garantiza* L_2^{AG}*. Luego, por Teorema 5.3 y Definición 6.18 de desacuerdo en* ArgAPL*,* $\mathcal{P}(Ag, C)$ *no garantiza* L_1^{AG} *ya que* $L_2^{AG} \otimes L_1^{AG}$ *al tener* $(L_1, L_2) \in \oslash$*. Por lo tanto,* $C \nvdash_S L_1^{AG}$*, lo cual contradice la hipótesis.* □

Otra propiedad muy importante que cumplen las metas de logro actuales es que el agente no perseguirá ninguna meta de logro que ya crea alcanzada. Recuerde que las

metas de logro representan situaciones del mundo que agente quiere que se cumplan. Por lo tanto, si el agente cree actualmente en L, la meta de logro para L no debería considerarse dado que L ya ha sido alcanzado. Es decir, un agente sólo tendrá metas de logro para literales que no sean creencias actuales.

Teorema 6.4 *Sea Ag la especificación de un agente* ArgAPL *con $\oslash$ su conjunto de literales en conflicto, y C una configuración de Ag. Si $C \vdash_S L^B$ entonces $C \nvdash_S L^{AG}$.*

Prueba : Suponga que $C \vdash_S L^{AG}$. Entonces, por Definición 6.26 de meta de logro actual el programa $\mathcal{P}(Ag, C)$ asociado a C garantiza L^{AG}. Luego, por Teorema 5.3 y Definición 6.18 de desacuerdo en ArgAPL*, $\mathcal{P}(Ag, C)$ no garantiza L^B ya que $L^{AG} \otimes L^B$. Por lo tanto, $C \nvdash_S L^B$, lo cual contradice la hipótesis.* □

Estos últimos dos resultados son centrales para la semántica de ArgAPL ya que muestran que, a pesar de utilizar metas de logro condicionales y potencialmente conflictivas, el agente se comporta de manera esperada. Como se vio en el Capítulo 2, cuando un agente 3APL cree en L simplemente remueve todas las metas para L. En contraste, en ArgAPL no es necesario remover las reglas que generan las metas (esto sería indeseable), ya que es el mecanismo argumentativo es el encargado de determinar si una meta está suficientemente justificada. Por ejemplo, como se vio en el Ejemplo 6.21, si bien el agente tiene un argumento para c^{AG}, este es derrotado por un argumento para c^B que se encuentra no derrotado. Por lo tanto, c^B será una creencia actual para el agente mientras que c^{AG} no será una meta actual.

Un propiedad que surge a partir del Teorema 6.4, es que el agente no buscará alcanzar metas que también sean metas de mantenimiento actualmente mantenidas. Esta propiedad se caracteriza a través del siguiente corolario.

Corolario 6.1 *Sea Ag la especificación de un agente* ArgAPL *con $\oslash$ su conjunto de literales en conflicto, y C una configuración de Ag. Si $L^{MG} \in \mathsf{Mantenidas}(C)$, entonces $C \nvdash_S L^{AG}$.*

Prueba : Como L^{MG} está en $\mathsf{Mantenidas}(C)$, entonces por Definición 6.25 de metas de mantenimiento actualmente mantenidas se tiene que $C \vdash_S L^B$. Luego por Teorema 6.4, vale que $C \nvdash_S L^{AG}$. □

Por último, se mostrará que si el agente tiene una meta de mantenimiento actual, entonces no tendrá ninguna meta de logro actual que esté en conflicto con ella.

Proposición 6.6 *Sea Ag la especificación de un agente* ArgAPL *con* $\oslash$ *su conjunto de literales en conflicto, y C una configuración de Ag. Si* $C \vdash_S L_1^{MG}$*, entonces* $C \nvdash_S L_2^{AG}$ *con* $L_2 = \overline{L_1}$ *o* $(L_1, L_2) \in \oslash$*.*

Prueba : Suponga que $C \vdash_S L_2^{AG}$ *tal que* $L_2 = \overline{L_1}$*. Entonces, por Definición 6.26 de meta de logro actual el programa* $\mathcal{P}(Ag, C)$ *asociado a C garantiza* L_2^{AG}*. Luego, por Teorema 5.3 y Definición 6.18 de desacuerdo en* ArgAPL*,* $\mathcal{P}(Ag, C)$ *no garantiza* L_1^{AG} *dado que* $L_2^{AG} \otimes L_1^{MG}$ *por tener* $L_2 = \overline{L_1}$*. De esta manera,* $C \nvdash_S L_1^{MG}$*, lo cual contradice la hipótesis. Suponga ahora que* $C \vdash_S L_2^{AG}$ *y que* $(L_1, L_2) \in \oslash$*. Por Definición 6.26,* $\mathcal{P}(Ag, C)$ *garantiza* L_2^{AG}*. Luego, por Teorema 5.3 y Definición 6.18 de desacuerdo en* ArgAPL*,* $\mathcal{P}(Ag, C)$ *no garantiza* L_1^{MG} *ya que* $L_2^{AG} \otimes L_1^{MG}$ *porque* $(L_1, L_2) \in \oslash$*. Por lo tanto,* $C \nvdash_S L_1^{MG}$*, lo cual contradice la hipótesis.* □

Funciones de Actualización y Proyección

Para definir la semántica operacional del agente será necesario considerar dos funciones de actualización con las que contará el agente. Estas son la función de actualización de percepciones y la función de actualización de conocimiento. Si bien estas funciones no son fijas (como la función $\mathcal{F}$ en 3APL), a continuación se presentará una versión concreta de cada una de ellas para ilustrar el comportamiento de los agentes ArgAPL.

La función de actualización de percepciones, notada FAper(), recibe como entrada un conjunto de efectos perceptivos (representado la nueva percepción) y la percepción anterior, y retorna como resultado la percepción del agente revisada. En el dominio presentado en la Sección 6.1.1, esta función básicamente agregará consistentemente todos los elementos de la percepción nueva a la percepción anterior. Entonces, por una parte, será necesario remover de la percepción anterior todos los literales que estén en desacuerdo con los literales de la nueva percepción. Por otra parte, también será necesario remover aquellos literales de la percepción anterior que estén desactualizados. Es decir, se removerán aquellos literales para los cuales se haya visualizado algún cambio.

Ejemplo 6.24 *Considere que el agente que se desenvuelve en el dominio presentado en la Sección 6.1.1 tiene como conjunto de percepciones* $\Theta_1^P = \{en(i3, yo)^P,$

$energia(yo, 95)^P, en(i1, ag2)^P, energia(ag2, 100)^P, en(i1, tesoro(t4))^P, en(i2, tesoro(t1))^P, pos(i1, 1, 1)^P, pos(i2, 4, 2)^P, pos(i3, 3, 2)^P\}$. *Suponga ahora que el agente recibe una nueva percepción* $NP = \{en(i1, yo)^P, energia(yo, 92)^P, pos(i1, 1, 1)^P, pos(i3, 3, 2)^P\}$, *que caracteriza la situación ilustrada en la figura 6.4 (recuerde que el agente sólo percibe la información de las casillas dentro del rectángulo con borde punteado).*

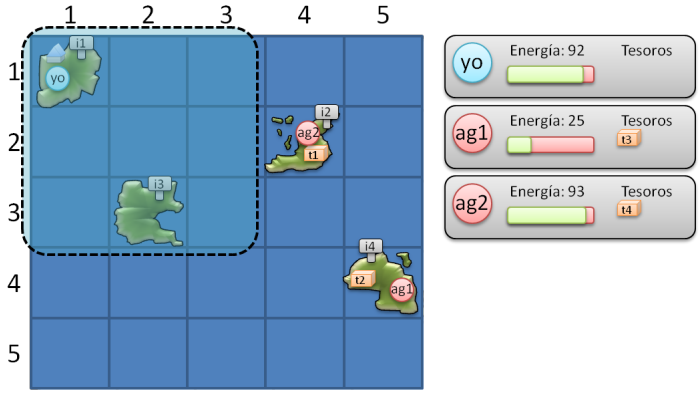


Figura 6.4: Nueva Percepción.

El resultado de aplicar $\mathsf{FAper}(NP, \Theta_1^P) = \Theta_n^P = \{en(i1, yo)^P, energia(yo, 92)^P, en(i2, tesoro(t1))^P, energia(ag2, 100)^P, pos(i1, 1, 1)^P, pos(i2, 4, 2)^P, pos(i3, 2, 3)^P\}$. *Note que* $en(i3, yo)^P$ *y* $energia(yo, 95)^P$ *fueron removidos por estar en desacuerdo con* $en(i1, yo)^P$ *y* $energia(yo, 92)^P$ *de la percepción nueva. Por otra parte,* $en(i1, ag2)^P$, $energia(ag2, 100)^P$ *y* $en(i1, tesoro(t4))^P$ *fueron removidas porque el agente ve que no están más allí (aunque no sabe donde están ahora). En contraste, note que, por ejemplo,* $en(i2, tesoro(t1))^P$ *no será removido. Esto se debe a que la nueva percepción no permite visualizar cambios con respecto a la información de la isla* $i2$.

Note que el comportamiento de FAper() en este dominio no diferencia percepciones viejas de nuevas. Este modelo es similar al utilizado para la actualización de observaciones en O-DeLP [CCS05]. Claramente, se podrían desarrollar versiones más complejas de la función de actualización, por ejemplo, distinguiendo el turno donde se percibió cierta información. Sin embargo, el estudio de funciones de actualización de percepciones excede los alcances de este capítulo.

La función de actualización de conocimiento, notada FAcon(), recibe como entrada un conjunto de efectos metales, la base de creencias y la base de metas del agente; y

retornará las bases de creencias y de metas revisadas por el conjunto de efectos mentales. Para los ejemplos de este capítulo se asumirá una función de actualización de conocimiento que simplemente añade los elementos a agregar (+) y remueve los elementos a eliminar (−) según lo establecido por el conjunto de efectos mentales.

Ejemplo 6.25 *Considere la base de creencias Δ_1^B del Ejemplo 6.2, la base de metas Δ_1^G del Ejemplo 6.6, y la acción mental "agresivo" presentada en el Ejemplo 6.8. Recuerde que los efectos de esta acción mental eran los siguientes (cada efecto fue numerado para ser referenciado en el siguiente párrafo):*

$$E_{M_{agresivo}} = \begin{cases} (1) - (modo(defensivo)^B \prec), \\ (2) - (presaFacil(A)^B \prec rival(A)^B, tiene(A, tesoro(T))^B, cansado(A)^B), \\ (3) - (peligrosa(I)^B \prec en(I, A)^B, rival(A)^B), \\ (4) + (modo(agresivo)^B \prec), \\ (5) + (presaFacil(A)^B \prec rival(A)^B, tiene(A, tesoro(T))^B) \end{cases}$$

Se tendrá entonces que $\mathsf{FAcon}(E_{M_{agresivo}}, \Delta_1^B, \Delta_1^G) = (\Delta_1^B a, \Delta_1^G)$*, donde* $\Delta_1^B a = \Delta_1^B \setminus \{(1), (2), (3)\} \cup \{(4), (5)\}$*, y* $(1), (2), (3), (4), (5)$ *son los numeros de las reglas en* $E_{M_{agresivo}}$*. Note que* Δ_1^G *no se modifica ya que las reglas que se agregan y remueven en la acción mental "agresivo" corresponden únicamente a la base de creencias.*

Si bien esta función de actualización de conocimiento es suficiente para los ejemplos de este capítulo, en el Capítulo 7 se mostrará una función de actualización más sofisticada, la cual se basa en un operador de revisión de creencias. Como se verá en ese capítulo, la función cumplirá ciertas propiedades relacionadas con la semántica de creencias y metas del agente ArgAPL.

Para determinar si una acción o un plan amenaza alguna de las creencias que el agente se encuentra manteniendo actualmente será necesario analizar qué creencias actuales tendría el agente luego de ejecutar esa acción o ese plan. Para tal fin a continuación se presentará la función de proyección que, dada una configuración y un conjunto de efectos sobre la percepción y los componentes metales del agente (estos últimos presentados en las definiciones 6.9 y 6.8 respectivamente), retornará la configuración resultante de aplicar los cambios correspondientes.

Definición 6.27 (Función de Proyección) *Sea* $C = (\Theta^P, \Delta^B, \Delta^G, \Pi)$ *una configuración de un agente* ArgAPL*,* E_P *un conjunto efectos perceptivos,* E_M *un conjunto de efectos mentales,* $\mathsf{FAper}()$ *una función de actualización de percepciones y* $\mathsf{FAcon}()$ *una función de actualización de conocimiento. Si* $\mathsf{FAcon}(E_M, \Delta^B, \Delta^G) = (\Delta^B_u, \Delta^G_u)$ *y* $\mathsf{FAper}(E_P, \Theta^P) = \Theta^P_u$*, la función de proyección de* E_P *y* E_M *en* C*, notada* $\mathsf{FProy}(C, E_P, E_M)$*, retornará la configuración* $C_u = (\Theta^P_u, \Delta^B_u, \Delta^G_u, \Pi)$*.*

6.3.4. Dinámica de ejecución en ArgAPL

En esta sección se mostrará la dinámica de ejecución de un agente ArgAPL. Es decir, se indicará cómo se eligen los planes, se ejecutan acciones y se actualiza el entorno. En ArgAPL, al igual que en los lenguajes de programación de agentes que cuentan con su semántica operacional formalizada (*e.g.*, 3APL), se utilizará un sistema de transiciones para especificar esta semántica.

Como se vio en el Capítulo 2, un sistema de transiciones [Plo04] consiste de un conjunto de reglas de transición que explican cómo el sistema descripto pasa de un estado a otro. Por lo tanto, una transición en un agente será la transformación de una configuración en otra configuración, que además corresponderá a un paso computacional individual.

Una regla de transición en ArgAPL, estará compuesta por tres elementos: una condición bajo la cual la regla de transición es aplicable, una configuración de origen y una configuración de destino. Por ejemplo, una regla de transición T con condición C, configuración origen O y configuración destino D, se notará:

$$T = \frac{\text{condición C}}{\text{configuración origen O}\rightarrow\text{configuración destino D}}$$

Como se verá a continuación las diferentes reglas de transición ArgAPL se valdrán de la semántica de percepciones, creencias, metas de logro y metas de mantenimiento actuales, así también como de las funciones de actualización presentadas en la sección anterior. La referencia a estos elementos se efectuará en la parte condicional del las reglas de transición, por ejemplo, requiriendo que el agente tenga como creencias actuales ciertos elementos para poder realizar la transformación.

Selección de Planes

Utilizando las creencias y metas actuales el agente podrá determinar qué reglas de selección de planes son aplicables en una configuración. Básicamente, una de estas reglas será aplicable si y solo si su meta asociada es una meta actual, si sus precondiciones son creencias actuales y si no amenaza ninguna de las metas de mantenimiento actualmente mantenidas. Si se aplica una regla de selección de planes, su plan pasará a estar en la base de planes listo para ser ejecutado. Sin embargo, este tipo de reglas solo se aplicará cuando no haya un plan adoptado.

Definición 6.28 (Selección de planes) *Sea* $C = (\Theta^P, \Delta^B, \Delta^G, \epsilon)$ *la configuración de un agente* ArgAPL *Ag con un conjunto de reglas de selección de planes* $\mathcal{R}$*, donde* $(\kappa \leftarrow \beta \mid \pi : (P, M)) \in \mathcal{R}$*. La transición para esta regla de selección de plan en la configuración* C *se define como:*

$$\frac{(C \vdash_S \kappa) \wedge (C \vdash_S \beta) \wedge (\forall L^{MG} \in \mathsf{Mantenidas}(C) : \mathsf{FProy}(C, P, M) \vdash_S L^B)}{(\Theta^P, \Delta^B, \Delta^G, \epsilon) \rightarrow (\Theta^P, \Delta^B, \Delta^G, (\pi, \kappa))}$$

Ejemplo 6.26 *Considere la configuración* $C_1 = (\Theta_1^P, \Delta_1^B, \Delta_1^G, \epsilon)$ *presentada en el Ejemplo 6.12 para el agente del Ejemplo 6.11, el cual cuenta con el conjunto de reglas de selección de planes presentado en el Ejemplo 6.10. En* C_1 *será posible aplicar la regla selección de planes*

$$en(i2, yo)^{AG} \leftarrow en(i3, yo)^B, dist(i3, i2, 3)^B, energia(yo, 95)^B \mid [viajar(i3, i2)] : (\{en(i2, yo)^P, energia(yo, 92)^P\}, \emptyset)$$

ya que, como se mostró en el Ejemplo 6.22, $en(i2, yo)^{AG}$ *en una meta de logro actual en* C_1*; como se mostró en el Ejemplo 6.17,* $en(i3, yo)^B$*,* $dist(i2, i3, 3)^B$ *y* $energia(yo, 95)^B$ *son creencias actuales en* C_1*; y como se motró en el Ejemplo 6.20,* $\sim cansado(yo)^{MG}$ *es la única de mantenimiento actualmente mantenida, la cual no es violada por los efectos del plan. Es decir, al aplicar la función de proyección con los efectos del plan, la configuración resultante tendrá* $\sim cansado(yo)^B$ *como creencia actual. Por lo tanto, si el agente aplica esta regla en la configuración en* C_1*, entonces resultará en la configuración* $C_{2a} = (\Theta_1^P, \Delta_1^B, \Delta_1^G, ([viajar(i3, i2)], en(i2, yo)^{AG}))$*.*

Note que la siguiente regla de selección de planes también es aplicable en C_1

$$modo(agresivo)^{AG} \leftarrow modo(defensivo)^{B} \mid [agresivo] : (\emptyset, E_{Magresivo})$$

donde $E_{Magresivo}$ es el efecto de la acción mental "agresivo"presentada en el Ejemplo 6.8. De manera similar a la regla anterior, esta regla es aplicable porque $modo(agresivo)^{AG}$ es una meta de logro actual, $modo(defensivo)^{B}$ es una creencia actual, y el plan no amenaza ninguna meta de mantenimiento actualmente mantenida. Entonces, si el agente aplica esta regla en C_1 resultará en la configuración $C_{2b} = (\Theta_1^P, \Delta_1^B, \Delta_1^G, ([agresivo], modo(agresivo)^{AG}))$.

Está semántica de selección de planes establece el modelo de compromiso del agente. Cuando el plan de una regla es adoptado, el agente se compromete a tratar de alcanzar la meta de esa regla. Sin embargo, es importante notar que la ejecución del plan no asegura que la meta será alcanzada. Esto depende de cómo el desarrollador especifique el plan, por lo que la meta sólo se considerará alcanzada, como se vio en la Sección 6.3.1, cuando sea una creencia para el agente.

Claramente, como se vio en el Ejemplo 6.26, el agente ArgAPL podría tener más de una regla de selección de planes aplicable. En ese caso, al igual que en 3APL, deberá elegir entre una de ellas utilizando alguna estrategia. Por ejemplo, podría preferir el plan más corto o tratar de alcanzar las metas más valiosas. Aun así, al igual que en 3APL, en ArgAPL este criterio se maneja de manera abstracta y modular.

Como puede verse en la Definición 6.28, como parte de la condición para aplicar una regla de selección de planes, al proyectar los efectos del plan el agente debería seguir creyendo en lo que actualmente mantiene. Este es uno de los elementos que, además del uso de argumentación, diferencian a ArgAPL de otras aproximaciones de programación de agentes. Puntualmente, este chequeo hace que los agentes tengan proactividad con respecto a la protección de sus metas de mantenimiento, ya que no seleccionarán planes que no aseguren que sus metas de mantenimiento seguirán valiendo. Aun así, es importante destacar que la proactividad no es absoluta ya que el agente sólo considerará los efectos especificados en la regla de selección de planes, en lugar de considerar todos los estados producidos por la ejecución de cada una de las acciones del plan de la regla. Claramente, contemplar todos los estados generados por los planes al momento de determinar cuál ejecutar es un proceso muy costoso [vRDW08]. Por lo tanto, si bien ArgAPL no provee una solución completa a este problema, brinda un método eficiente que aprovecha el mecanismo argumentativo para proteger sus metas de mantenimiento. Adicionalmente,

recuerde que el proceso argumentativo también protege las condiciones a mantener al descartar aquellas metas de logro que van en contra de las metas de mantenimiento actuales, como se vio en la propiedad 6.6.

Como se explicó anteriormente, una vez que una regla de selección de planes es aplicada, su plan es puesto en ejecución. Este plan permanecerá en la base de planes hasta que finalice su ejecución, hasta que la meta por la cual fue adoptado ya no sea una meta actual, o hasta que por alguna otra razón falle una de sus acciones.

La ejecución de un plan corresponde a una ejecución secuencial de sus acciones asociadas. A continuación, se presentará la semántica de ejecución para las acciones presentadas en este capítulo.

Acciones Mentales y Externas

Las acciones mentales introducen cambios permanentes en los componentes mentales del agente. Estos efectos, como se mencionó anteriormente, serán descriptos por la función de actualización de conocimiento $\mathsf{FAcon}()$. Si durante la ejecución de un plan se cumple la precondición de una acción mental y los efectos proyectados de esa acción no violan las metas de mantenimiento actualmente mantenidas, entonces se aplica la función de actualización de conocimiento. En caso contrario, si las precondiciones no se cumplen, si la meta asociada al plan ya no está justificada, o si se viola alguna meta actualmente mantenida, se dirá que el plan está bloqueado y será removido de la ejecución.

Definición 6.29 (Ejecución de Acción Mental) *Sea* $C = (\Theta^P, \Delta^B, \Delta^G, ([nma, a_0, \ldots, a_m], \kappa))$ *la configuración de un agente* ArgAPL *con un conjunto de acciones mentales* S_{ma}*, donde* $[nma, a_0, \ldots, a_m]$ *es plan adoptado para una meta* κ*, y* nma *es el nombre de la acción mental* $ma = (\beta, nma, E_M) \in S_{ma}$*. La ejecución de* ma *en* C *se define como:*

$$\frac{\begin{array}{c}(C\vdash_S \beta) \wedge (C\vdash_S \kappa) \wedge (\mathsf{FAcon}(E_M, \Delta^B, \Delta^G) = (\Delta^{B\prime}, \Delta^{G\prime}))\wedge \\ (\forall L^{MG} \in \mathsf{Mantenidas}(C) : \mathsf{FProy}(C, \emptyset, E_M)\vdash_S L^B)\end{array}}{(\Theta^P, \Delta^B, \Delta^G, ([nma, a_0, \ldots, a_m], \kappa)) \rightarrow (\Theta^P, \Delta^{B\prime}, \Delta^{G\prime}, ([a_0, \ldots, a_m], \kappa))}$$

$$\frac{(C\nvdash_S \beta) \vee (C\nvdash_S \kappa) \vee (\exists L^{MG} \in \mathsf{Mantenidas}(C) : \mathsf{FProy}(C, \emptyset, E_M)\nvdash_S L^B)}{(\Theta^P, \Delta^B, \Delta^G, ([nma, a_0, \ldots, a_m], \kappa)) \rightarrow (\Theta^P, \Delta^B, \Delta^G, \epsilon)}$$

Ejemplo 6.27 *Considere la configuración* $C_{2b} = (\Theta_1^P, \Delta_1^B, \Delta_1^G, ([agresivo], modo(agresivo)^{AG}))$ *producto de aplicar la regla de selección de plan para* $modo(agresivo)^{AG}$ *en la configuración* C_1*, como se mostró en el Ejemplo 6.26. Note que el primer elemento del plan en ejecución es la acción mental "agresivo". Como se vio en el Ejemplo 6.8 esta acción fue especificada como:*

$$agresivo = \begin{cases} \{modo(defensivo)^B\} \\ EM_{agresivo} = \begin{cases} (1) - (modo(defensivo)^B \prec), \\ (2) - (presaFacil(A)^B \prec rival(A)^B, tiene(A, tesoro(T))^B, cansado(A)^B), \\ (3) - (peligrosa(I)^B \prec en(I, A)^B, rival(A)^B), \\ (4) + (modo(agresivo)^B \prec), \\ (5) + (presaFacil(A)^B \prec rival(A)^B, tiene(A, tesoro(T))^B) \end{cases} \end{cases}$$

En primer lugar observe que las metas de logro, metas de mantenimiento y creencias actuales serán las mismas en C_{2b} *que en* C_1*, ya que las bases de creencias y metas no cambiaron de una configuración a la otra. Por lo tanto, note que esta acción será aplicable en* C_{2b}*, dado que la precondición* $modo(defensivo)^B$ *es una creencia actual, la meta de logro en la base de planes* $modo(agresivo)^{AG}$ *es una meta de logro actual, y no se viola la meta de mantenimiento actualmente mantenida* $\sim cansado(yo)^{MG}$ *al aplicar la acción (ya que las reglas que se añaden y remueven como efecto de esta acción no afectarán la garantía de* $\sim cansado(yo)^B$*). El efecto de aplicar esta acción mental en* C_{2b} *estará dado por el resultado de la función* FAcon() *con los efectos mentales* $E_{Magresivo}$ *,* Δ_1^B *y* Δ_1^G*. En el Ejemplo 6.25 se mostró que* $FAcon(E_{Magresivo}, \Delta_1^B, \Delta_1^G) = (\Delta_1^B a, \Delta_1^G)$*, donde* $\Delta_1^B a$ *es* Δ_1^B *sin las reglas* $(1), (2), (3)$ *de* $E_{Magresivo}$ *y con el añadido de las reglas* $(4), (5)$ *de* $E_{Magresivo}$*. Entonces si se ejecuta esta acción mental en* C_{2b} *se obtendrá como configuración resultante* $C_{3b} = (\Theta_1^P, \Delta_1^B a, \Delta_1^G, ([\,], modo(agresivo)^{AG}))$*.*

Las acciones externas, como se presentó en la Sección 6.2, están compuestas de un nombre, un conjunto de precondiciones y un conjunto de efectos perceptivos. El efecto producido por estas acciones será afectar la futura percepción del agente. Al igual que lo ocurrido con las acciones mentales, para poder aplicar estas acciones será necesario que se cumplan las precondiciones, que la meta para la que se está ejecutando la acción siga justificada y que el efecto producido por la ejecución de la acción no viole las metas de mantenimiento actualmente mantenidas. En caso que no se cumpla alguna de estas condiciones, al igual que para las acciones mentales, se removerá el plan de ejecución.

Definición 6.30 (Ejecución de Acción Externa) *Sea* $C = (\Theta^P, \Delta^B, \Delta^G, ([nea, a_0, \ldots, a_m], \kappa))$ *la configuración de un agente* ArgAPL *con un conjunto de acciones externas* S_{ea}*, donde* $[nea, a_0, \ldots, a_m]$ *es plan adoptado para una meta* κ*, y nea es el nombre de la acción externa* $ea = (\beta, nea, E_P) \in S_{ea}$*. La ejecución de ea en C se define como:*

$$\frac{(C \vdash_S \beta) \wedge (C \vdash_S \kappa) \wedge (\forall L^{MG} \in \mathsf{Mantenidas}(C) : \mathsf{FProy}(C, E_P, \emptyset) \vdash_S L^B)}{(\Theta^P, \Delta^B, \Delta^G, ([nea, a_0, \ldots, a_m], \kappa)) \rightarrow (\Theta^P, \Delta^B, \Delta^G, ([a_0, \ldots, a_m], \kappa))}$$

$$\frac{(C \nvdash_S \beta) \vee (C \nvdash_S \kappa) \vee (\exists L^{MG} \in \mathsf{Mantenidas}(C) : \mathsf{FProy}(C, \emptyset, E_M) \nvdash_S L^B)}{(\Theta^P, \Delta^B, \Delta^G, ([nea, a_0, \ldots, a_m], \kappa)) \rightarrow (\Theta^P, \Delta^B, \Delta^G, \epsilon)}$$

Note que, a diferencia de las acciones mentales, las acciones externas no producen su efecto ni bien son ejecutadas. Esto se debe a que modifican la futura percepción del agente y, por lo tanto, los cambios que produzcan se verán reflejados cuando el agente realice la actualización de su percepción. Más adelante se mostrará cómo se realiza esta actualización de la percepción del agente.

Adicionalmente, observe que las acciones externas, al igual que en 3APL, no establecen explícitamente cómo se comunican con el entorno. Esto se debe a que depende de la implementación de la conexión entre el agente y el entorno, la cual va mas allá de los objetivos de este capítulo.

Ejemplo 6.28 *Considere la configuración* $C_{2a} = (\Theta_1^P, \Delta_1^B, \Delta_1^G, ([viajar(i3, i2)], en(i2, yo)^{AG}))$ *producto de aplicar la regla de selección de plan para* $en(i2, yo)^{AG}$ *en la configuración* C_1*, como fue mostrado en el Ejemplo 6.26. Note que el primer elemento del plan en ejecución es la acción externa* $viajar(i3, i2)$*. Como se vio en el Ejemplo 6.9, esta acción fue especificada como:*

$$viajar(I_o, I_d) = \begin{cases} \{en(I_o, yo)^B, dist(I_o, I_d, D)^B, energia(yo, X)^B, X \geq D\}, \\ \{en(I_d, yo)^P, energia(yo, X - D)^P\} \end{cases}$$

En primer lugar observe que las metas de logro, metas de mantenimiento y creencias actuales serán las mismas en C_{2a} *que en* C_1*, ya que las bases de creencias y metas no cambiaron de una configuración a la otra. Por lo tanto, note que esta acción será aplicable en* C_{2ab} *dado que las precondiciones* $en(i3, yo)^B$*,* $dist(i3, i2, 3)^B$ *y* $energia(yo, 95)^B$ *son creencias actuales, la meta de logro en la base de planes* $en(i2, yo)^{AG}$ *es una meta de logro actual, y no se viola ninguna meta de mantenimiento mantenida al aplicar la acción.*

Entonces si se ejecuta esta acción mental en C_{2a} se obtendrá como configuración resultante $C_{3a} = (\Theta_1^P, \Delta_1^B, \Delta_1^G, ([], en(i2, yo)^{AG}))$. El efecto de la acción externa se recibirá a través de la nueva percepción.

Como se puede observar a partir de la semántica para la ejecución de las acciones mentales y las acciones externas, el agente ArgAPL también cuenta con un mecanismo de protección reactiva para sus metas de mantenimiento. Como se vio, el agente cancelará una acción (en conjunto con el plan al que pertenece) si esta viola alguna de sus metas de mantenimiento actualmente mantenidas.

Como se presentó en las definiciones 6.29 y 6.30, no cumplir alguna de las condiciones de una acción (mental o externa) implica descartar el plan. La motivación para esta semántica está relacionada con lo que implica el incumplimiento de cada una de estas condiciones. Si las precondiciones de la acción a ejecutar no se cumplen, entonces la acción quedará bloqueada y el agente no podrá actuar hasta que no se cumplan. Por lo tanto, es preferible que el agente busque un nuevo plan en lugar de quedarse esperando. Si la meta deja de estar justificada (ya sea porque las razones que la motivaron ya no valen o porque ya ha sido alcanzada), claramente no tiene sentido continuar con la ejecución del plan que se está ejecutando. Si la acción a ejecutar viola una meta de mantenimiento, entonces el agente no debería ejecutarla, con lo cual debería removerse el plan en ejecución. Cabe destacar que, si el agente contara con reglas de revisión de planes como en 3APL [DvRM05], sería posible aprovecharlas para cambiar la acción que se intenta ejecutar cuando no se cumplen las precondiciones o cuando se amenazan las metas de mantenimiento.

Cuando un agente ejecuta todas las acciones de un plan, el plan ha finalizado y por lo tanto debe ser removido de la base de planes. A continuación se presentará la transición que modela esta situación.

Definición 6.31 (Finalización de Plan) *Sea $C = (\Theta^P, \Delta^B, \Delta^G, ([\,], \kappa))$ la configuración de un agente. La transición para finalizar el plan se define como:*

$$\frac{\top}{(\Theta^P, \Delta^B, \Delta^G, ([\,], \kappa)) \rightarrow (\Theta^P, \Delta^B, \Delta^G, \epsilon)}$$

Actualización de Percepciones

Cuando el agente recibe una nueva percepción debe actualizar la percepción anterior con la nueva. Como se presentó anteriormente, para actualizar las percepciones el agente

utilizará la función FAper(). La nueva percepción se representará mediante un conjunto de efectos perceptivos. Para presentar esta transición se utilizará la función Meta que recibe una base de planes Π y retorna la meta de logro asociada a Π, o $\emptyset$ en caso de que la base de planes sea ϵ. De esta manera, si la meta asociada al plan en ejecución sigue siendo una meta actual se continuará con la ejecución del plan. En caso contrario, el plan será removido.

Definición 6.32 (Actualización de Percepciones) *Sea* $C = (\Theta^P, \Delta^B, \Delta^G, \Pi)$ *una configuración de un agente* ArgAPL*, y* E_P *un conjunto de efectos perceptivos representando la nueva percepción. La semántica para actualización de percepciones en* C *se define como:*

$$\frac{\mathit{FAper}(E_P, \Theta^P) = \Theta^P_n \wedge C \vdash_S \mathit{Meta}(\Pi)}{(\Theta^P, \Delta^B, \Delta^G, \Pi) \rightarrow (\Theta^P_n, \Delta^B, \Delta^G, \Pi)}$$

$$\frac{\mathit{FAper}(E_P, \Theta^P) = \Theta^P_n \wedge C \nvdash_S \mathit{Meta}(\Pi)}{(\Theta^P, \Delta^B, \Delta^G, \Pi) \rightarrow (\Theta^P_n, \Delta^B, \Delta^G, \epsilon)}$$

Ahora que se han introducido todas las transiciones para un agente ArgAPL, se presentará una proposición que muestra que cuando se alcanza la meta adoptada en medio de la ejecución de un plan, este es removido de la ejecución.

Proposición 6.7 *Sea* $C = (\Theta^P, \Delta^B, \Delta^G, (\pi, L^{AG}))$ *una configuración de un agente* ArgAPL*. Si* $C \vdash_S L^B$*, entonces cualquier posible transición llevará al agente a la configuración* $C' = (\Theta^{P'}, \Delta^B, \Delta^G, \epsilon)$*.*

Prueba : Por hipótesis y el Teorema 6.4 se tiene que $C \nvdash_S L^{AG}$*. Si la transición es de actualización de percepción, como* $C \nvdash_S L_i^{AG}$*, por Definición 6.32 se tiene que la configuración resultante es* $C' = (\Theta^{P'}, \Delta^B, \Delta^G, \epsilon)$*, donde* $\Theta^{P'}$ *el resultado de aplicar la función* FAper()*. Sino, como* C *tiene un plan para ejecutar, existen dos posibilidades. En el primer caso* $\pi = [\ ]$ *y la única transición posible es la de finalización de plan (Definición 6.31), la cual conduce a la configuración* $C' = (\Theta^{P'}, \Delta^B, \Delta^G, \epsilon)$*, con* $\Theta^{P'} = \Theta^P$*. El otro caso es que* $\pi = [a_0, \ldots, a_n]$*, con lo cual la única posible transición es que se ejecute la primer acción del plan (la transición dependerá de qué tipo de acción sea* a_0*). Luego, como* $C \nvdash_S L_i^{AG}$ *puede observarse, por definiciones 6.30 y 6.29, que ya sea una acción mental o una acción externa, la transición siempre llevará a la configuración* C'*, con* $\Theta^{P'} = \Theta^P$*.* $\square$

6.3.5. Ciclo Deliberativo

Las reglas de transición presentadas en la sección anterior por sí solas no definen cómo será la ejecución completa de un agente ArgAPL. Estas reglas solo describen cómo el agente pasará de un estado a otro bajo ciertas condiciones. El modelo de ejecución completo de un agente ArgAPL estará dado por el ciclo deliberativo que se construye a partir de las reglas de transición.

El ciclo deliberativo en ArgAPL es básicamente un conjunto de pasos que siguen cierto orden y se repiten indefinidamente. Cada paso deliberativo es especificado por un conjunto de reglas transición que indican qué cambios de estado realizará el agente en cada momento. Por lo tanto, este ciclo puede verse como un programa deliberativo que determina qué operaciones de deliberación deben aplicarse y cuándo. Por ejemplo, podría programarse para determinar cómo son elegidas las metas de selección de planes, los momentos en que se realiza la actualización de conocimiento, o cuántas acciones mentales podrá ejecutar el agente antes de esperar por una nueva percepción.

En ArgAPL, al igual que en 3APL, este ciclo deliberativo no es fijo. Por lo tanto, utilizando las reglas de transición de la sección anterior se podrá definir el ciclo deliberativo adecuado. En la Figura 6.5 se ilustra un ciclo deliberativo simple que puede ser utilizado por el intérprete de ArgAPL.

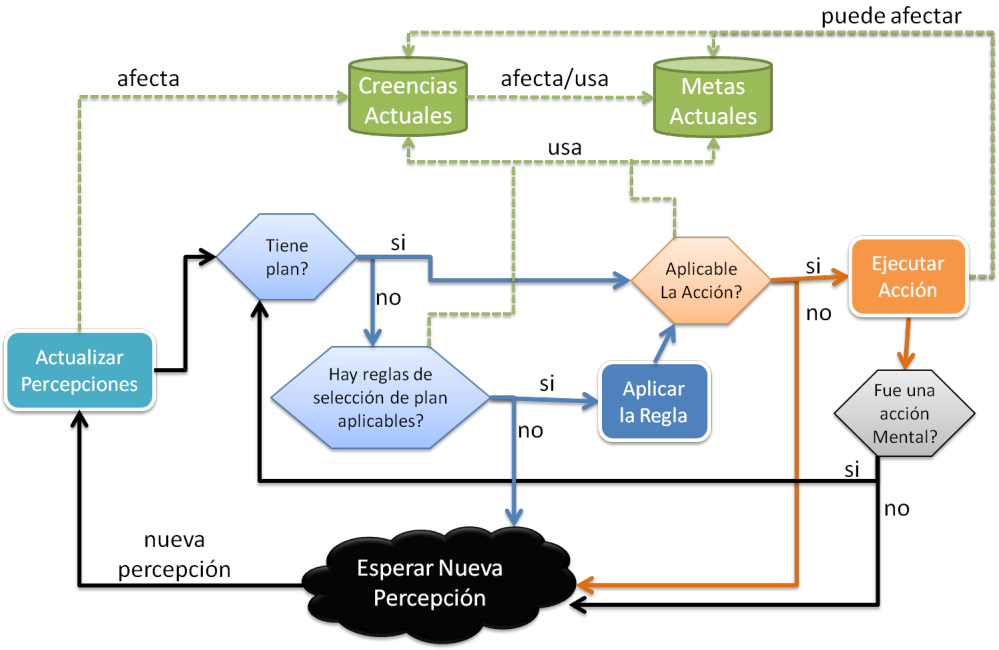


Figura 6.5: Un ciclo deliberativo para los agentes ArgAPL.

Cada iteración de este ciclo comienza con la actualización de las percepciones. Luego se verifica si el agente tiene un plan en ejecución, y en caso de ser el plan vacío se lo remueve. Posteriormente, si no hay plan actualmente en ejecución, se busca una regla de selección de plan (en su orden de ocurrencia en el programa) realizando las consultas adecuadas por las creencias actuales y por las metas de logro y de mantenimiento actuales. La primer regla aplicable pondrá su plan en ejecución. En caso de no haber reglas de selección de plan aplicables se esperará la llegada de una nueva percepción. Una vez que hay un plan adoptado, el agente intentará ejecutar la primer acción de ese plan. Para esto se consultan las creencias y metas actuales, y en caso de que la acción no sea aplicable se remueve el plan y se espera por una nueva percepción. Si la acción se puede ejecutar y se trata de una acción externa se espera a una nueva percepción. Por otra parte, si la acción ejecutada es una acción mental, se aplican los efectos mentales (afectando a las creencias y metas actuales) y se vuelve a la etapa en la que se controla si hay un plan por ejecutar.

El orden de ejecución arriba presentado no es bajo ninguna circunstancia universal. Es decir, este ciclo deliberativo podría no ser adecuado para todas las situaciones posibles. Finalmente, dado que ArgAPL puede verse como una variante de 3APL, las operacio-

nes para programar el ciclo deliberativo presentadas en [DdBDM03] podrían adaptarse fácilmente para ArgAPL.

6.4. Trabajo Relacionado

Modelos BDI basados en Argumentación

La idea de utilizar argumentación rebatible en agentes inteligentes no es nueva. Previos a esta tesis existen resultados que vinculan estas dos áreas [RA06, RGS07, ADL08]. Sin embargo, ninguna de estas aproximaciones propone lenguajes de programación de agentes sino que sólo presentan arquitecturas de agente. Su principal objetivo es determinar qué es lo mejor para hacer en una situación determinada. Por lo tanto, se focalizan en el impacto de argumentación en la deliberación, abstrayéndose del modelo dinámico. Por ejemplo, esos trabajos no especifican cómo las acciones modificarán los componentes mentales del agente y, en consecuencia, cómo afectarán a los argumentos utilizados para razonar. En ArgAPL se estudiaron estas cuestiones presentando reglas de semántica formal para todas las acciones e interacciones relacionadas con los componentes del agente, en conjunto con resultados que respaldan al comportamiento esperado.

Por ejemplo, ninguno de los trabajos [RA06, RGS07, ADL08] analiza la situación para un argumento cuando una meta es alcanzada. En [RGS07] los lenguajes para representar metas y creencias son completamente distintos, imposibilitado la capacidad de representar metas declarativas. En [RA06, ADL08], al igual que en ArgAPL, es posible utilizar un lenguaje de representación común identificando la fuente de la información (creencias, metas, etc). Sin embargo, en estos modelos un agente podrá tener un argumento para una creencia L y un argumento para una meta L, ambos garantizados. Por lo tanto, en este escenario el agente podría buscar y ejecutar un plan para una meta que ya ha sido alcanzada. Como se vio en la Sección 6.3.3, en ArgAPL esta situación no ocurre ya que si el agente tiene un argumento garantizado para una creencia L, todo argumento de meta de logro para L estará derrotado, como muestra el Teorema 6.4.

Al no considerar la dinámica del agente, [RA06, RGS07, ADL08] no especifican cómo el agente ejecutará sus acciones y sus planes. En particular, esto implica que tampoco analizan cuál es su nivel de compromiso con las metas que buscará el agente ni cómo el mecanismo argumentativo estará involucrado en este proceso. Como se mostró en este

capítulo, en ArgAPL se presentan reglas formales donde se especifica cómo el agente ejecutará sus acciones, y se muestra qué tipo de compromiso tendrá el agente con las metas que está persiguiendo.

Finalmente, otra diferencia relacionada con el modelo deliberativo del agente es que [RA06, RGS07, ADL08] no consideran las metas de mantenimiento. En contraste, ArgAPL permite su representación y, como se mostró anteriormente, estarán caracterizadas por argumentos y tendrán un rol fundamental al momento de determinar qué metas de logro y qué planes terminará seleccionando el agente.

3APL con Programación en Lógica Dinámica

En [NL06] se presenta una modificación de 3APL utilizando *Dynamic Logic Programming* (DLP). En ese trabajo, al igual que en esta tesis, se propone un lenguaje más sofisticado y poderoso para representar metas y creencias. Sin embargo, si bien DLP y T-DeLP permiten representar y razonar con información conflictiva, son formalismos completamente diferentes. DLP está basado en la semántica de *Answer-Set Programaming* y su objetivo es la actualización de conocimiento. En contraste, T-DeLP es un mecanismo argumentativo general, por lo que su objetivo es transformar la información potencialmente conflictiva en argumentos, y luego utilizar un proceso argumentativo para decidir cuáles de ellos prevalecen.

Una de las principales diferencias entre ArgAPL y [NL06] radica en que el criterio de decisión para información contradictoria en DLP está fijo (se prefiere información más nueva). En contraste, ArgAPL se basa en T-DeLP, por lo que utiliza un criterio de comparación modular para decidir entre los argumentos en conflicto de un mismo tipo. Por lo tanto, el programador del agente puede utilizar un criterio adecuado a sus necesidades y, en particular, podría emplear uno que, al igual que DLP, prefiera argumentos con información más nueva. Por otra parte, si bien cuando se comparan argumentos del distinto tipo en ArgAPL la preferencia es fija, en [NL06] toda preferencia es fija.

Por otra parte, el hecho de preferir la información más nueva no siempre es adecuado. En [NL06], cuando se alcanza una meta de logro L, el agente agrega a la base de metas de logro $\sim L$. Luego, dado que DLP siempre prefiere la información más reciente, L no será mas perseguible, al margen de si el agente cree o no en L. En ArgAPL esta situación se contempla a través de los conflictos entre argumentos. Así, en ArgAPL el agente no

perseguirá una meta logro L (que no esté derrotada por otra meta) mientras crea en L; en caso contrario podrá adoptar como meta a L sin inconvenientes.

En [NL06] se presenta un modelo para representar metas de mantenimiento. Sin embargo, en ese trabajo no se establece ninguna forma, reactiva o proactiva, para proteger las metas de mantenimiento. Es decir, sólo se plantea la posibilidad de representarlas pero no se captura la semántica de la interacción de estas metas con el resto de los componentes, especialmente con metas de logro, acciones y planes. En contrapartida, en ArgAPL se mostró cómo representar este tipo de metas y cómo los mecanismos de protección interactúan con los otros componentes del agente.

Metas en 3APL con Lógica Default

En [vRDM09] se presenta una aproximación basada en una versión reducida de 3APL donde se añaden reglas especiales para poder generar nuevas metas de logro, las cuales pueden ser potencialmente contradictorias. Por lo tanto, al igual que en ArgAPL, se podrán representar metas condicionales y en conflicto. Luego, para decidir qué metas de logro prevalecen finalmente se utiliza *lógica default* [Rei80].

El uso de lógica default conduce a [vRDM09] a incluir todas los posibles conflictos para una meta en la regla que la genera. Esto implica que, por ejemplo, si dos reglas para generar metas tienen como cabeza literales complementarios (con lo cual pueden concluir metas en conflicto), será necesario incluir el complemento respectivo en el cuerpo de cada regla, ya que de lo contrario será posible derivar ambos literales. En ArgAPL esto no es necesario ya que los conflictos se manejan de manera modular a partir de la relación de desacuerdo, la cual se construye con el conjunto de literales en conflicto, los literales complementarios de un mismo tipo y las relaciones entre los tipos.

Una de las principales diferencias entre el trabajo de [vRDM09] y ArgAPL está en los efectos de alcanzar una meta de logro. En [vRDM09], si bien se discute la situación, se plantean dos alternativas cuando se alcanza una meta de logro L (*i.e.*, cuando el agente cree en L). La primer alternativa consiste en remover todas las reglas de generación de meta para L, con lo cual el agente no podrá intentar alcanzarla nuevamente si en algún momento dejó de creer en L. La segunda alternativa consiste en no realizar ningún cambio, lo que permitiría al agente intentar perseguir una meta que ya ha sido alcanzada. En ArgAPL, como se vio a lo largo de este capítulo, y especialmente como muestra el

Teorema 6.4, el sistema argumentativo es el encargado de modelar esta situación. De esta manera, si el agente garantiza una creencia L no podrá tener como meta de logro actual a L, ya que el argumento de creencia para L derrotará al argumento de meta de logro para L. Así, si el agente ya alcanzó L, podrá seguir intentando alcanzarlo en caso de no creer en L, y no lo intentará alcanzar en caso de creer en L.

También es interesante destacar que en [vRDM09] se presentan dos modelos razonamiento aprovechando lógica default: uno crédulo y otro escéptico. En el modelo crédulo, en particular, se aceptan las metas pertenecientes a cualquier extensión. En ArgAPL solo se provee un modelo escéptico a través del proceso de prueba dialéctico presentado para T-DeLP. Claramente, ArgAPL se podrá adaptar al modelo extensional de las semánticas de aceptabilidad de los MATMs presentadas para T-DeLP, y considerar diferentes escenarios para cada una de estas semánticas. Por otra parte, si bien en [vRDM09] se presenta el modelo cauto, no se explica cómo el agente generará los planes a partir de las diferentes extensiones.

Otra diferencia entre ArgAPL y la aproximación presentada en [vRDM09] está en la forma en que se vincula la base de creencias con la base de metas para decidir qué metas tendrá el agente. En [vRDM09] se asume una base de creencias consistente, y para determinar si una meta es actual se consulta a la base de creencias por cada literal que sea una precondición de creencia. En ArgAPL la base de creencias puede tener conflictos, y el mecanismo argumentativo será utilizado para determinar qué creencias serán finalmente aceptadas. Además, a diferencia de [vRDM09], para determinar si una meta de logro es actual se necesita una única consulta al programa T-DeLP asociado, donde en el proceso de respuesta se buscarán los argumentos para las creencias necesarias para sustentar a la meta.

Finalmente, cabe destacar que [vRDM09] no considera la existencia de metas de mantenimiento, mientras que ArgAPL sí lo hace. Introducir metas de mantenimiento en [vRDM09] requeriría considerar una nueva base y sería necesario revisar cómo se generan las reglas default, ya que todos los posibles conflictos deben estar especificados en las reglas. Como se vio en este capítulo, en ArgAPL el uso de distintos tipos de argumento en conjunto con sus relaciones hacen natural la interacción entre los distintos tipos de metas.

Aproximaciones que utilizan Metas de Mantenimiento

Las metas de mantenimiento han sido abordadas por algunos lenguajes de programación de agentes. En particular en, Jadex [PBL05] y en una extensión de Jason [HBW06] es posible representar metas de mantenimiento de una manera completamente reactiva. Tanto en [PBL05] como en [HBW06], el único mecanismo que tiene un agente para proteger sus metas de mantenimiento es actuar para repararlas si han sido violadas. Esta es la forma más eficiente pero menos comprometida para proteger las metas de mantenimiento. En ArgAPL esto se modela a través de la generación de metas de logro cuando el agente tiene una meta de mantenimiento actual que no es mantenida. Además, se mostró que un agente ArgAPL protege sus metas de mantenimiento proactivamente antes de elegir un plan o antes de ejecutar una acción que podría ponerlas en riesgo.

En [HvR07] se propone una extensión para GOAL en la que se consideran las metas de mantenimiento. En este trabajo, al igual que en ArgAPL, se provee un mecanismo proactivo para proteger a este tipo de metas. El mecanismo presentado en [HvR07] realiza una proyección de cada uno de los estados por los que pasará el agente si ejecuta un plan y evalúa para cada uno de estos estados si se viola alguna de las metas de mantenimiento. Esto permite realizar una protección completa de las metas de mantenimiento. Sin embargo, este proceso es muy costoso. Por cada plan posible el agente deberá analizar todos los estados posibles por los que puede pasar. En ArgAPL, si bien no se brinda una protección completa, se presenta una aproximación eficiente para la protección proactiva, analizando las condiciones en un solo estado al momento de decidir si ejecutar el plan.

Otra distinción entre ArgAPL y el trabajo de [HvR07] está relacionada con el lenguaje de representación. En [HvR07] las metas (tanto de mantenimiento como de logro) son incondicionales y consistentes. Por lo tanto, el mecanismo de proyección y decisión trabaja sobre un dominio mucho más simple que el de ArgAPL. Si se extendiera [HvR07] para considerar metas condicionales y conflictivas, el mecanismo para determinar si un plan viola una meta de mantenimiento sería mucho más costoso.

Compromiso de mente abierta

En 3APL, GOAL y 2APL los agentes tienen un compromiso *ciego* o de *mente simple* con respecto a la meta que están tratando de alcanzar. Un compromiso ciego implica que el agente sólo descartará un plan cuando la meta se crea alcanzada, mientras que el

compromiso de mente simple indica que el agente sólo descartará la meta cuando ya se haya alcanzado o cuando sea imposible alcanzarla.

En contrapartida, a través de la semántica de ejecución de las acciones mentales y las acciones externas, se mostró que ArgAPL exhibe un compromiso de *mente-abierta* [RG91, HBW06] con respecto a la meta que el agente está tratando de alcanzar. Este tipo de compromiso establece que el agente debe descartar una meta y su plan si la meta ya ha sido alcanzada, si la meta es inalcanzable en el estado actual del mundo, o si las condiciones bajo las cuales la meta fue adoptada ya no válidas. Note que el modelo de ArgAPL contempla todas estas situaciones utilizando el mecanismo argumentativo.

Claramente, el compromiso de mente-abierta es más sofisticado que los compromisos de mente simple o ciego. En particular, si se quisiera implementar el compromiso de mente abierta en lenguajes como 3APL sería costoso o complejo, ya que habría que analizar de manera retrospectiva cuáles fueron las condiciones que llevaron a adoptar la meta, analizando reglas de razonamiento y planes. En cambio, en ArgAPL se podría modelar el compromiso de mente simple o el compromiso ciego simplemente agregando condiciones en las reglas semánticas anteriormente mencionadas.

6.5. Conclusiones

En este capítulo se presentó ArgAPL, un lenguaje de programación de agentes basado en 3APL, donde los componentes son representados a través de reglas rebatibles tipadas y hechos tipados, y aprovechan el mecanismo argumentativo de T-DeLP para efectuar el razonamiento. Para presentar ArgAPL, a lo largo del capítulo se mostró cómo especificar sus agentes y las reglas de semántica formal que describen cómo se ejecutarán los agentes. Además, se mostraron diversas propiedades que muestran el funcionamiento esperado de sus componentes.

Durante el desarrollo de este capítulo se vio que T-DeLP provee un lenguaje de representación declarativo adecuado para las necesidades de ArgAPL. Esto se debe a que, además de permitir especificaciones declarativas, T-DeLP permite representar información conflictiva y utilizar el mecanismo argumentativo para determinar que información prevalecerá. En particular, el concepto de tipo de argumento resultó adecuado para representar y distinguir la información y propiedades de los diferentes componentes mentales

de un agente ArgAPL. Además, las relaciones entre estos tipos permitieron modelar de manera simple y modular los conflictos entre la información de los diferentes componentes mentales. En los lenguajes de programación de agentes estos conflictos suelen ser dejados de lado o resueltos de manera *ad-hoc* dada la complejidad que traen aparejada. Un ejemplo de esta situación, como se vio en la sección de trabajo relacionado, es cómo y cuándo el agente descarta metas de logro que ya cree alcanzadas. En ArgAPL esto se modeló relacionando los argumentos de creencias y los de metas de logro.

El concepto de tipo de argumento también resultó útil para introducir a las metas de mantenimiento como un tipo de meta adicional en ArgAPL. Como se pudo observar, las metas de mantenimiento no deben ser tratadas trivialmente. La mayoría de las aproximaciones que tratan estas metas lo hacen sólo de manera reactiva o a través de procesos muy costosos. En ArgAPL se diseñó una estrategia que se vale de las relaciones entre argumentos de distinto tipo y de las nociones semánticas del lenguaje para brindar tanto una solución proactiva eficiente y como dos soluciones reactivas. La primera permite que el agente evite ejecutar un plan que puede amenazar sus metas de mantenimiento, mientras que las soluciones reactivas son utilizadas para cancelar una acción que violará un meta de mantenimiento, o para generar una meta de logro cuando el agente tiene una meta de mantenimiento que actualmente no mantiene.

Por otra parte, la interacción desarrollada en ArgAPL para los distintos tipos de argumento y el uso de metas condicionales permitieron modelar de manera sencilla la política de compromiso a las metas del agente. Como se vio en la sección de semántica de ArgAPL, antes de ejecutar una acción de un plan el agente controla que el plan pueda continuar (controlando las precondiciones del plan y que la acción no viole las metas de mantenimiento mantenidas), así como también controla que la meta asociada a ese plan siga justificada (*i.e.*, que sea una meta de logro actual para el agente). Esto último implica que, para que el agente continúe con la ejecución del plan, tiene que existir un argumento no derrotado para la meta, el cual representará la justificación para esa meta. Note que si existe un argumento de creencias no derrotado para el literal que busca alcanzar la meta de logro, el argumento de meta de logro quedaría derrotado (como muestra el Teorema 6.4) y por lo tanto el agente debería cancelar el plan. De esta manera, el agente tendrá un compromiso de *mundo-abierto* con respecto a sus metas. Usualmente este tipo de compromisos es costoso o imposible de lograr en lenguajes como 3APL. En ArgAPL, se mostró que es posible alcanzarlo efectuando una única consulta T-DeLP.

En ArgAPL se utilizó como estructura para representar los planes una secuencia de acciones mentales y externas. El utilizar una estructura de plan sencilla permitió mostrar con mayor claridad y de manera más directa cómo el razonamiento argumentativo afecta a la dinámica del agente. No obstante, es posible extender ArgAPL de manera directa para considerar subplanes, estructuras repetitivas y condicionales en los planes, con solo agregar las transiciones para estos constructores. De manera similar, también es posible incorporar las reglas de revisión de planes presentadas para 3APL.

Por último, el modelo de tipos de argumento permitió hacer una distinción entre percepciones y creencias. De esta manera fue posible identificar a las percepciones como un componente mental separado, el cual se encuentra directamente vinculado con la base de creencias, ya que toda percepción será una creencia para el agente ArgAPL. Como se vio a lo largo del capítulo, los agentes ArgAPL tendrán una preferencia de las percepciones por sobre las creencias, lo cual se modeló sencillamente a través de la relación de preferencia entre tipos.

Capítulo 7

Extensiones para ArgAPL: remoción de información y poda heurística

En este capítulo se detallan propuestas que pueden aplicarse al lenguaje ArgAPL presentado en el Capítulo 6 con el objetivo de avanzar sobre dos aspectos: la formalización de un concepto de remoción de información basado en operadores de la teoría del cambio; y un método que acelere el cómputo de la garantía que determina cuales son las creencias y metas actuales de un agente ArgAPL. Estas dos extensiones son independientes entre sí, sin embargo, se incluyen en forma conjunta en este capítulo dado que ambas están relacionadas con el proceso de prueba dialéctico con el cual se obtiene una garantía en T-DeLP (que fuera explicado en la Sección 5.4.5 del Capítulo 5).

Remoción de información basada en contracción

En el capítulo anterior se introdujo ArgAPL, un lenguaje de programación de agentes basado en 3APL, el cual utiliza el sistema argumentativo T-DeLP presentado en el Capítulo 5. En ArgAPL una base de creencias Δ^B está formada por un conjunto de reglas rebatibles tipadas. Por lo tanto, una creencia actual para un agente en una configuración determinada será un elemento que está garantizado a partir del programa T-DeLP que considera a Δ^B. Al igual que 3APL, ArgAPL cuenta con acciones mentales que permiten remover información de Δ^B. Sin embargo, es posible que a pesar de remover una regla rebatible, el literal correspondiente a la cabeza de esta regla

siga siendo una creencia actual para el agente. Por ejemplo, considere la base de creencias $\Delta^B = \{(a^B -\prec c^B), (c^B -\prec), (a^B -\prec)\}$, y suponga que una acción mental remueve "$a^B -\prec$ ". A partir de la configuración resultante de aplicar esta acción mental el agente seguiría teniendo a "a^B" como una creencia actual, ya que el literal igual estará garantizado a partir de ${\Delta^B}_1 = \{(a^B -\prec c^B), (c^B -\prec)\}$. De manera análoga, esta situación también ocurre al efectuar la remoción de metas de logro y metas de mantenimiento.

En las versiones concretas de 3APL [DvRM05, Das08] la remoción de información de las bases de conocimiento de un agente sigue el comportamiento arriba mencionado para ArgAPL. Sin embargo, como se ha sugerido en la literatura [HdBvdHM99], es esperable que al remover una pieza de información de una base de conocimiento esta no sea inferible a partir de la configuración resultante. Por lo tanto, inspirado en los operadores de contracción de la teoría de cambio, es posible definir un mecanismo de remoción más sofisticado que logre el comportamiento deseado.

En la primera parte de este capítulo (Sección 7.1) se presentará un conjunto de acciones mentales que permiten capturar el comportamiento mencionado en el párrafo anterior al remover información de las bases de conocimiento de un agente ArgAPL. Para esto, se adaptarán los operadores de contracción para DeLP presentados en [GGK+11]. Para cada uno de estos operadores se definirá una acción mental asociada que permite aplicar el operador adaptado a las bases de conocimiento de un agente ArgAPL.

De esta manera, las nuevas acciones mentales incrementarán el poder expresivo de ArgAPL, acercándolo más a los modelos teóricos presentados para los lenguajes de programación de agentes [HdBvdHM99]. Contar con diversas acciones mentales basadas en contracción también le aporta un grado de flexibilidad a los agentes ArgAPL, permitiéndoles utilizar acciones mentales más rápidas, o acciones mentales más complejas pero que preservan una mayor cantidad de información.

Aceleración en el proceso de cálculo de creencias y metas actuales en ArgAPL

Como se mostró en el Capítulo 6 ArgAPL provee un lenguaje de representación más expresivo que otros APLs, ya que utiliza el mecanismo argumentativo provisto por T-DeLP para modelar el razonamiento y los conflictos entre los componentes mentales de los agentes. Aun así, otro aspecto que resulta importante en el área de los lenguajes de programación de agentes es la eficiencia. Si bien el mecanismo de prueba dialéctico pro-

visto por T-DeLP es atractivo computacionalmente, en la segunda parte de este capítulo (Sección 7.2) se mostrará una técnica para mejorar su eficiencia.

En particular, se presentará una técnica especializada de poda con el objetivo de reducir el tamaño de los árboles de dialéctica construidos al computar las creencias y metas actuales de los agentes ArgAPL. Esta técnica de poda se basará en una heurística, la cual determinará qué argumentos son más propensos a estar no derrotados y será calculada de manera *offline* con respecto a la ejecución del agente. Se mostrará que la construcción de los árboles de dialéctica guiados por esta heurística lleva a considerar un menor número de argumentos al calcular las creencias y metas actuales, lo cual acelerará el razonamiento de los agentes ArgAPL.

7.1. Remoción de información basada en operadores de contracción

En el Capítulo 6 se mostró que los agentes ArgAPL pueden utilizar las *acciones mentales* para modificar sus bases de creencias o sus bases de metas. En particular, estas acciones permiten remover información de tales bases. Esta remoción, como se vio en el Capítulo 6, es puramente sintáctica. Por ejemplo, al aplicar una acción mental con efecto $\{-(x^B -\!\prec), -(y^B -\!\prec z^B)\}$, el agente simplemente removerá las reglas $(x^B -\!\prec)$ y $(y^B -\!\prec z^B)$ de su base de creencias.

Como se vio en el Capítulo 2 este comportamiento con respecto a la remoción de información para las acciones mentales sigue el estilo de las versiones concretas de 3APL [DvRM05, Das08]. Sin embargo, en las versiones fundacionales de 3APL y en algunas de sus variantes abstractas, si bien no se explicita cómo se realizan estas modificaciones, se espera que al remover una pieza de información de una base de conocimiento esta no sea inferible en la configuración resultante. Por ejemplo, si la acción mental remueve x^B, entonces será esperable que x^B no sea una creencia actual en la configuración resultante de aplicar la acción. En esta sección se presentará un conjunto de acciones mentales que capturen esta intuición en ArgAPL.

La situación enunciada en párrafo anterior para 3APL puede verse como un problema de *contracción* en el contexto de la teoría del cambio [AGM85]. En estas teorías, el problema de la contracción concierne el estudio de la forma en que se altera una base de

conocimiento Δ de manera tal que no derive más una fórmula determinada α. En particular, se espera que el resultado de la contracción $\Delta - \alpha$ adhiera ciertas propiedades de racionalidad como *éxito* (α no debería inferirse de Δ luego de la contracción) y mínimo cambio (perder la menor cantidad de fórmulas posibles).

Como se vio en el capítulo anterior, en ArgAPL las creencias, metas de logro y metas de mantenimiento actuales son las inferencias del programa T-DeLP resultante de la unión de las bases de conocimiento del agente en una configuración. En el capítulo 5 se mostró que las inferencias de un programa T-DeLP determinan aquellos literales tipados que están garantizados luego del proceso de dialéctica en el que se consideran argumentos y contra-argumentos. De esta manera, una forma de ver a las acciones mentales que se propondrán en esta sección es como contracciones sobre el programa T-DeLP resultante de la configuración en la cual se aplicarán.

En [GGK+11] el autor de esta tesis, en conjunto con Diego R. García, Patrick Krümpelmann y Matthias Thimm, formó parte del grupo que desarrolló mecanismos para realizar contracciones por un literal determinado en DeLP. En ese trabajo se presentó un conjunto de operadores de contracción, donde cada uno de ellos representa una de las distintas nociones de inferencia en DeLP. Por lo tanto, en esta sección se presentará una adaptación de las versiones constructivas de esas contracciones en el contexto de ArgAPL, y se propondrán acciones mentales para los agentes que se basarán en ellas. Estas acciones mentales serán identificadas como *acciones mentales de contracción* y buscarán asegurar que luego de ser aplicadas la configuración resultante no infiera el literal que se intentó remover.

A continuación se presentarán algunos conceptos preliminares. Luego se irán introduciendo las nociones que motivan las distintas contracciones en conjunto con las acciones mentales de contracción que las utilizan. Para cada acción se mostrará su semántica operacional y, a partir de esta, que la remoción se realiza de manera exitosa.

7.1.1. Conceptos Preliminares

Las acciones mentales que de desarrollarán a lo largo de las siguientes secciones son llamadas acciones mentales de contracción. Estas acciones estarán caracterizadas por el literal tipado que deben remover y deberán verificar que al ser aplicadas el literal tipado a remover no será inferible por el agente en la configuración resultante.

Como se vio en el Capítulo 6 al presentar las acciones mentales en ArgAPL, se mostró que éstas afectan a las bases de conocimiento del agente. Por lo tanto, las acciones mentales presentadas en este capítulo sólo podrán asegurar que ciertos elementos inferibles a partir de las bases de creencias, de metas de logro o de metas de mantenimiento, ya no sean inferibles luego de aplicar la acción mental (*i.e.*, sólo podrán remover literales de tipo B, AG y MG). En consecuencia, note que estas acciones no afectarán a los elementos de las percepciones.

Para que las acciones mentales de contracción cumplan el requisito de que luego de aplicarlas no se infiera el literal que se quiere remover, será necesario que se eliminen ciertas reglas del programa T-DeLP correspondiente. Como se mencionó anteriormente, la elección de las reglas que se eliminarán estará determinada por operadores de contracción. Como se verá más adelante, los operadores se guiarán por diferentes nociones relacionadas con la derivación, estado de garantía e influencia que exhibe el literal a contraer en el programa T-DeLP resultante.

Aun así todas las contracciones sólo eliminarán reglas que permiten inferir el literal que se quiere remover. Esto se debe a que, si se eliminasen otras reglas, la acción mental podría poner en juego inferencias para otros literales tipados, haciendo que esto sea indeseado para el modelo de razonamiento del agente. Por lo tanto, a continuación se presentará el conjunto de posibles remociones, el cual básicamente contendrá todas las reglas rebatibles tipadas cuya cabeza sea el literal a remover del programa T-DeLP resultante de la configuración. Como se verá más adelante, este conjunto será la base que utilizarán las contracciones para determinar qué reglas remover.

Definición 7.1 (Conjunto de posibles remociones) *Sea* $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ *el programa T-DeLP resultante de una configuración* C *para un agente* ArgAPL *Ag, y* L^X *un literal tipado tal que* $X \in \{B, AG, MG\}$. *El conjunto de posibles remociones* S *para* L^X *en* $\mathcal{P}(Ag, C)$ *es tal que* $(L^X \prec Cuerpo) \in S$ *si y solo si* $(L^X \prec Cuerpo) \in \Delta^B \cup \Delta^G$.

Ejemplo 7.1 *Considere el programa T-DeLP* $\mathcal{P}(Ag_7, C_7) = (\Gamma, \Theta^P{}_7, \Delta^B{}_7 \cup \Delta^G{}_7)$ *resultante de una configuración* C_7 *para un agente* ArgAPL Ag_7, *donde:*

$$\Delta^B{}_7 \cup \Delta^G{}_7 = \left\{ \begin{array}{ll} (a^{AG} -\prec b^B) & (b^B -\prec c^B) \\ (a^{AG} -\prec q^B, c^B) & (\sim b^B -\prec c^B, d^B) \\ (a^{AG} -\prec p^B) & (p^B -\prec c^B) \\ (\sim a^{AG} -\prec q^B) & (\sim p^B -\prec d^B) \\ (a^{AG} -\prec r^B) & \end{array} \right\} \quad \Theta^P{}_7 = \left\{ \begin{array}{l} c^P \\ d^P \\ q^P \end{array} \right\}$$

y Γ *es la signatura por defecto para* ArgAPL *(la signatura tal que los desacuerdos sólo ocurren por literales complementarios, ver Capítulo 6). Note que a partir de este programa, además de los argumentos para la percepción (argumentos basados en los hechos), se pueden construir los siguientes argumentos representativos:*

$$\begin{array}{ll} \mathcal{A}_1 = \langle \{(a^{AG} -\prec b^B), (b^B -\prec c^B), c^P\}, a^{AG} \rangle & \mathcal{B}_1 = \langle \{(b^B -\prec c^B), c^P\}, b^B \rangle \\ \mathcal{A}_2 = \langle \{(a^{AG} -\prec q^B, c^B), q^P, c^P\}, a^{AG} \rangle & \mathcal{B}_2 = \langle \{(\sim b^B -\prec c^B, d^B), c^P, d^P\}, \sim b^B \rangle \\ \mathcal{A}_3 = \langle \{(a^{AG} -\prec p^B), (p^B -\prec c^B), c^P\}, a^{AG} \rangle & \mathcal{B}_3 = \langle \{(p^B -\prec c^B), c^P\}, p^B \rangle \\ \mathcal{C}_1 = \langle \{(\sim a^{AG} -\prec q^B), q^P\}, \sim a^{AG} \rangle & \mathcal{B}_4 = \langle \{(\sim p^B -\prec d^B), d^P\}, \sim p^B \rangle \end{array}$$

Si se utiliza especificidad generalizada como criterio de comparación de argumentos, se podrán construir los árboles de dialéctica ilustrados en la Figura 7.1, donde los triángulos representan los argumentos arriba mencionados, las flechas punteadas representan derrotas por bloqueo y las flechas enteras corresponden a derrotas propias.

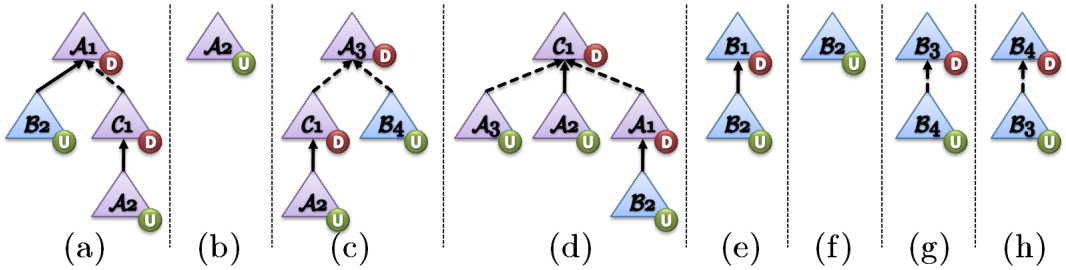


Figura 7.1: Árboles de dialéctica para el programa $\mathcal{P}(Ag_7, C_7)$.

Considere ahora el literal tipado para la meta de logro a^{AG}*. El conjunto de posibles remociones para* a^{AG} *en* $\mathcal{P}(Ag_7, C_7)$ *contendrá todas las reglas rebatibles de* $\Delta^B{}_7 \cup \Delta^G{}_7$ *cuya cabeza es* a^{AG}*, es decir,* $S = \{(a^{AG} -\prec b^B), (a^{AG} -\prec q^B, c^B), (a^{AG} -\prec p^B), (a^{AG} -\prec r^B)\}$.

A continuación se presentarán las distintas acciones mentales de contracción basadas en los operadores propuestos en [GGK$^+$11]. Para cada operador se indicará cómo es el programa resultante luego de aplicar la contracción. Esta contracción se basará en las definiciones constructivas para los operadores (ver [GGK$^+$11]) y en la noción de conjunto de posibles remociones. La caracterización formal de estos operadores y los postulados de racionalidad en que se basan escapa de los alcances de esta tesis. Para detalles de esta caracterización remitirse a [GGK$^+$11].

7.1.2. Remoción de Derivación

En esta sección se presentará la primer acción mental de contracción, la cual utiliza el operador de contracción basado en la noción de derivación. Este operador de contracción asegurará que luego de efectuar la contracción en el programa T-DeLP resultante no habrá ninguna derivación para el literal tipado contraído y, por lo tanto, no habrá argumentos para ese literal tipado. A continuación se define cuál es el resultado de aplicar este operador para un literal tipado sobre el programa resultante de una configuración.

Básicamente, esta contracción para un literal tipado L^X removerá del programa de la configuración aquellas reglas con cabeza L^X tales que son parte de una derivación para L^X. Por lo tanto, el conjunto de reglas a remover por la contracción será un subconjunto de las posibles remociones para L^X.

Definición 7.2 (Contracción basada en Derivación T-DeLP) *Sea* $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ *un programa T-DeLP resultante de una configuración* C *para un agente* ArgAPL *Ag, y* L^X *un literal tipado tal que* $X \in \{B, AG, MG\}$*. Una contracción basada en derivación de* L^X *en* $\mathcal{P}(Ag, C)$*, notada* $\mathcal{P}(Ag, C) \overset{d}{-\!\!-} L^X$*, es tal que* $\mathcal{P}(Ag, C) \overset{d}{-\!\!-} L^X = (\Gamma, \Theta^P, (\Delta^B \cup \Delta^G) \setminus S_d)$*, donde:*

1. $S_d \subseteq S$*, con* S *el conjunto de posibles remociones de* L^X *en* $\mathcal{P}(Ag, C)$*;*
2. *en* $(\Gamma, \Theta^P \setminus \{L^P\}, (\Delta^B \cup \Delta^G) \setminus S_d)$ *no hay una derivación rebatible tipada para* L^X*;*
3. S_d *es minimal: no existe* $S'_d \subset S_d$ *tal que satisface las condiciones 1 y 2.*

Ejemplo 7.2 *Considere el programa T-DeLP* $\mathcal{P}(Ag_7, C_7)$ *resultante de la configuración* C_7 *presentado en el Ejemplo 7.1. Al realizar* $\mathcal{P}(Ag_7, C_7) \overset{d}{-\!\!-} a^{AG}$ *se removerán de*

$\mathcal{P}(Ag_7, C_7)$ *todas las reglas rebatibles con cabeza* a^{AG} *que llevan a derivar* a^{AG}*, esto es,* $S_{d_7} = \{(a^{AG} -\prec b^B), (a^{AG} -\prec q^B, c^B), (a^{AG} -\prec p^B)\}$*. Note que en* S_{d_7} *no se incluye* $(a^{AG} -\prec r^B)$ *la cual, como se vio en el Ejemplo 7.1, es parte del conjunto de posibles remociones para* a^{AG} *en* $\mathcal{P}(Ag_7, C_7)$*. Esto se debe a que no es utilizada en ninguna derivación para* a^{AG} *en* $\mathcal{P}(Ag_7, C_7)$*. A continuación, en la Figura 7.2, se presentan los árboles de dialéctica obtenidos a partir del programa resultante de efectuar* $\mathcal{P}(Ag_7, C_7) \overset{d}{\sim} a^{AG}$ *(considerando sólo los argumentos rellenos).*

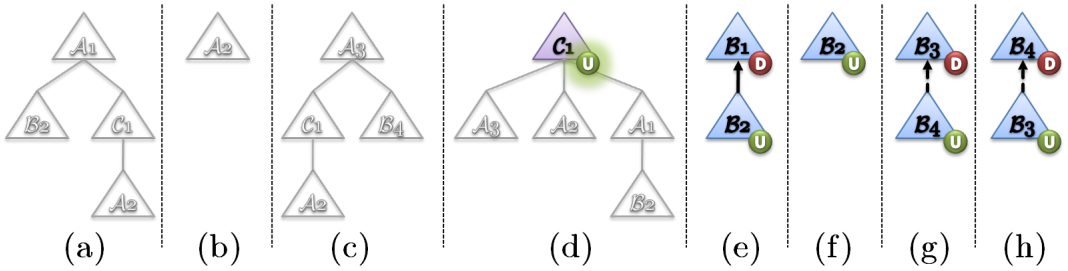


Figura 7.2: Árboles de dialéctica para el programa $\mathcal{P}(Ag_7, C_7) \overset{d}{\sim} a^{AG}$.

Habiendo presentado el resultado de aplicar el operador es posible definir una acción mental de contracción que lo utiliza. Este tipo de acciones serán llamadas *acciones mentales de contracción-d*, cuya sintaxis se define a continuación.

Definición 7.3 (Acción mental de contracción-d) *Una acción mental de contracción-d (o dcma) es una tupla* $Dcma = (\beta, Name, L^X)$*, donde* β *es un conjunto de precondiciones (ver Definición 6.7 del Capítulo 6),* $Name$ *es un átomo para referirse a Dcma desde un plan, y* L^X *es un literal tipado tal que* $X \in \{B, AG, MG\}$*.*

La semántica de ejecución de estas acciones se presentará siguiendo el sistema de transiciones para ArgAPL introducido en el Capítulo 6. Básicamente, estas acciones al igual que las acciones mentales presentadas en el capítulo anterior, serán aplicables siempre y cuando se cumpla su precondición en la configuración actual y su aplicación no viole ninguna meta de mantenimiento actualmente mantenida. El efecto de aplicarla estará dado por la contracción para derivación presentado en la Definición 7.2. De esta manera, el programa resultante de la contracción determinará cómo quedarán la base de creencias y la base de metas en la nueva configuración.

Definición 7.4 (Ejecución de acción mental de contracción-d) *Sea* $C = (\Theta^P, \Delta^B, \Delta^G, ([ndcma, a_0, \ldots, a_m], \kappa))$ *la configuración de un agente* ArgAPL *Ag, donde ndcma es el nombre de la acción mental de contracción-d* $dcma = (\beta, ndcma, L^X)$ *con* $X \in \{B, AG, MG\}$. *Si* $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ *es el programa T-DeLP asociado a Ag en C, la ejecución de dcma en C se define como:*

$$\frac{(C \vdash_S \beta) \wedge (C \vdash_S \kappa) \wedge L^P \notin \Theta^P \wedge \mathcal{P}(Ag, C) \xrightarrow{d} L^X = (\Gamma, \Theta^P, \Delta^{B\prime} \cup \Delta^{G\prime}) \wedge (\forall L^{MG} \in \textit{Mantenidas}(C) : (\Theta^P, \Delta^{B\prime}, \Delta^{G\prime}, ([a_0, \ldots, a_m], \kappa)) \vdash_S L^B)}{(\Theta^P, \Delta^B, \Delta^G, ([ndcma, a_0, \ldots, a_m], \kappa)) \rightarrow (\Theta^P, \Delta^{B\prime}, \Delta^{G\prime}, ([a_0, \ldots, a_m], \kappa))}$$

A continuación se mostrará que las acciones mentales de contracción-d cumplen con el requisito establecido para las acciones mentales de contracción. Es decir, se mostrará que luego de aplicar una acción mental de contracción-d para un literal tipado, ese literal ya no será inferido en la configuración resultante.

Proposición 7.1 *Sea* $C = (\Theta^P, \Delta^B, \Delta^G, ([ndcma, a_0, \ldots, a_m], \kappa))$ *la configuración de un agente* ArgAPL *Ag, donde ndcma es el nombre de la acción mental de contracción-d* $dcma = (\beta, ndcma, L^X)$ *con* $X \in \{B, AG, MG\}$, *y* $L^P \notin \Theta^P$. *Si* C' *es la configuración resultante de ejecutar dcma en C, entonces* $C' \nvdash_S L^X$.

Prueba : *Sea* $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ *el programa T-DeLP asociado a Ag en C, y sea* $(\Gamma, \Theta^P, \Delta^{B\prime} \cup \Delta^{G\prime})$ *el programa resultante de efectuar* $\mathcal{P}(Ag, C) \xrightarrow{d} L^X$. *Por Definición 7.7 de ejecución de acción mental de contracción-d,* $(\Gamma, \Theta^P, \Delta^{B\prime} \cup \Delta^{G\prime})$ *será el programa asociado a Ag en la configuración* C'. *Luego, por Definición 7.2,* $(\Gamma, \Theta^P, \Delta^{B\prime} \cup \Delta^{G\prime})$ *no deriva* L^X *y, por lo tanto, tampoco garantiza* L^X. *En consecuencia,* $C' \nvdash_S L^X$. □

Si bien estas acciones mentales verifican que en la configuración resultante ya no se inferirá el literal a remover, son bastante drásticas. Como se pudo observar en el Ejemplo 7.2, estas acciones removerán todas las derivaciones para el literal tipado, rompiendo todos los argumentos para ese literal (incluso aquellos que no aseguran su garantía). En la siguiente sección se presentarán acciones mentales que cumplen este objetivo pero que sólo rompen los argumentos que garantizan el literal a remover.

7.1.3. Remoción de Garantía

En esta sección se presentará una acción mental de contracción que utiliza un operador de contracción basado en la noción de garantía de T-DeLP. Este operador asegurará que

luego de efectuar la contracción en un programa T-DeLP, el literal tipado contraído no estará garantizado. A continuación se define cuál es el resultado de aplicar este operador para un literal tipado sobre el programa asociado a una configuración.

Definición 7.5 (Contracción basada en garantía T-DeLP) *Sea* $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ *un programa T-DeLP asociado a una configuración* C *para un agente* ArgAPL Ag*, y* L^X *un literal tipado tal que* $X \in \{B, AG, MG\}$*. Una contracción basada en garantía de* L^X *en* $\mathcal{P}(Ag, C)$*, notada* $\mathcal{P}(Ag, C) \overset{w}{\multimap} L^X$*, es tal que* $\mathcal{P}(Ag, C) \overset{w}{\multimap} L^X = (\Gamma, \Theta^P, (\Delta^B \cup \Delta^G) \setminus S_w)$*, donde:*

1. $S_w \subseteq S$*, con* S *el conjunto de posibles remociones de* L^X *en* $\mathcal{P}(Ag, C)$*;*
2. $(\Gamma, \Theta^P \setminus \{L^P\}, (\Delta^B \cup \Delta^G) \setminus S_w)$ *no garantiza* L^X*; y*
3. S_w *es minimal: no existe* $S'_w \subset S_w$ *tal que satisface las condiciones 1 y 2.*

Ejemplo 7.3 *Considere el programa T-DeLP* $\mathcal{P}(Ag_7, C_7)$ *resultante de la configuración* C_7 *presentado en el Ejemplo 7.1. Al realizar* $\mathcal{P}(Ag_7, C_7) \overset{w}{\multimap} a^{AG}$ *se removerán de* $\mathcal{P}(Ag_7, C_7)$ *todas las reglas rebatibles con cabeza* a^{AG} *que llevan a garantizar* a^{AG}*, es decir, aquellas reglas con cabeza* a^{AG} *tal que forman parte de los argumentos garantizados que concluyen* a^{AG}*, en este caso* $S_{w_7} = \{(a^{AG} -\!\prec q^B, c^B)\}$*. A continuación, en la Figura 7.3, se presentan los árboles de dialéctica obtenidos a partir del programa resultante de efectuar* $\mathcal{P}(Ag_7, C_7) \overset{w}{\multimap} a^{AG}$*.*

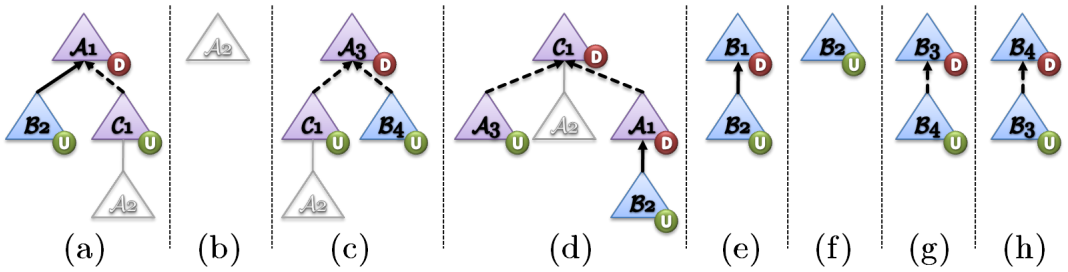


Figura 7.3: Árboles de dialéctica para el programa $\mathcal{P}(Ag_7, C_7) \overset{w}{\multimap} a^{AG}$.

Note que, como se puede observar al contrastar las figuras 7.2 y 7.3, la contracción basada en garantía preserva muchos más argumentos que la contracción basada en derivación. Por otra parte, ambas cumplen con el requisito necesario para ser utilizadas como mecanismo de actualización de las acciones mentales de contracción.

Una vez presentado el resultado del operador es posible definir una acción mental de contracción basada en garantía. Estas acciones serán llamadas *acciones mentales de contracción-w* (w por la traducción de garantía del inglés *warrant*). A continuación se definirá su sintaxis y semántica operacional de manera similar a lo realizado para las acciones mentales de contracción-d.

Definición 7.6 (Acción mental de contracción-w) *Una acción mental de contracción-w (o wcma) es una tupla* $Wcma = (\beta, Name, L^X)$*, donde* β *es un conjunto de precondiciones,* $Name$ *es un átomo para referenciar a* $Wcma$ *desde un plan, y* L^X *es un literal tipado tal que* $X \in \{B, AG, MG\}$*.*

Definición 7.7 (Ejecución de acción mental de contracción-w) *Sea* $C = (\Theta^P, \Delta^B, \Delta^G, ([nwcma, a_0, \ldots, a_m], \kappa))$ *la configuración de un agente* ArgAPL *Ag, donde* $nwcma$ *es el nombre de la acción mental de contracción-w* $wcma = (\beta, nwcma, L^X)$ *con* $X \in \{B, AG, MG\}$*. Si* $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ *es el programa T-DeLP asociado a Ag en C, la ejecución de wcma en C se define como:*

$$\frac{(C \vdash_S \beta) \wedge (C \vdash_S \kappa) \wedge L^P \notin \Theta^P \wedge \mathcal{P}(Ag, C) \xrightarrow{w} L^X = (\Gamma, \Theta^P, \Delta^{B\prime} \cup \Delta^{G\prime}) \wedge (\forall L^{MG} \in \mathsf{Mantenidas}(C) : (\Theta^P, \Delta^{B\prime}, \Delta^{G\prime}, ([a_0, \ldots, a_m], \kappa)) \vdash_S L^B)}{(\Theta^P, \Delta^B, \Delta^G, ([nwcma, a_0, \ldots, a_m], \kappa)) \rightarrow (\Theta^P, \Delta^{B\prime}, \Delta^{G\prime}, ([a_0, \ldots, a_m], \kappa))}$$

A continuación se mostrará que la acción mental de contracción-w cumple con los requisitos de las acciones mentales de contracción.

Proposición 7.2 *Sea* $C = (\Theta^P, \Delta^B, \Delta^G, ([nwcma, a_0, \ldots, a_m], \kappa))$ *la configuración de un agente* ArgAPL *Ag, donde nwcma es el nombre de la acción mental de contracción-w* $wcma = (\beta, nwcma, L^X)$ *con* $X \in \{B, AG, MG\}$*, y* $L^P \notin \Theta^P$*. Si* C' *es la configuración resultante de ejecutar wcma en C, entonces* $C' \nvdash_S L^X$*.*

Prueba : Sea $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ *el programa T-DeLP asociado a Ag en C. Sea* $(\Gamma, \Theta^P, \Delta^{B\prime} \cup \Delta^{G\prime})$ *el programa resultante de efectuar* $\mathcal{P}(Ag, C) \xrightarrow{w} L^X$*. Por Definición 7.6 de ejecución de acción mental de contracción-w* $(\Gamma, \Theta^P, \Delta^{B\prime} \cup \Delta^{G\prime})$ *será el programa asociado a Ag en la configuración* C'*. Luego, por Definición 7.5,* $(\Gamma, \Theta^P, \Delta^{B\prime} \cup \Delta^{G\prime})$ *no garantiza* L^X *y, por lo tanto,* $C' \nvdash_S L^X$*.* □

A partir de la Proposición 7.2 se puede observar que estas acciones mentales cumplen con el requisito de que la configuración resultante ya no infiera el literal a remover. Sin embargo, note que en la Figura 7.3 el argumento $\mathcal{A}_3$ para a^{AG} no fue removido ya que no estaba garantizado. Aun así, puede verse que, si bien $\mathcal{A}_3$ aparece marcado como **D** en su propio árbol de dialéctica (el árbol (c) en la figura), $\mathcal{A}_3$ aparece marcado como **U** en otro árbol (el árbol (d) en la figura). Más aún, $\mathcal{A}_3$ en el árbol que aparece como **U** está determinado que $\mathcal{C}_1$ quede marcado como **D** y, por lo tanto, su conclusión $\sim a^{AG}$ no sea garantizada en el programa. Es decir, si bien el argumento $\mathcal{A}_3$ no garantiza a a^{AG}, está *influenciando* el estado de garantía de otros literales. En la siguiente sección se caracterizará un conjunto de acciones mentales que consideran esta noción de influencia.

7.1.4. Remoción Basada en Influencia

Como se mencionó en el último párrafo de la sección anterior, pueden existir argumentos que, si bien estén derrotados en su propio árbol de dialéctica, aparezcan como no derrotados en otros árboles. De esta manera, estos argumentos para el literal tipado a remover exhiben *influencia* sobre el estado de inferencia de otros literales. Por lo tanto, como se vio en la sección anterior, utilizando una acción mental que sólo se basa en el estado de garantía del literal a remover, es posible que queden argumentos que influyan en el razonamiento del agente. Claramente, determinar si esto es deseable o no depende del dominio de aplicación en que se trate de utilizar el agente. Si resultara indeseable, entonces es posible utilizar las acciones mentales de contracción-d, que eliminan absolutamente todos los argumentos para el literal a contraer. No obstante, estas acciones mentales son muy drásticas en el sentido del mínimo cambio.

Por lo tanto, en esta sección se buscará presentar un conjunto de acciones mentales de contracción que cierren la brecha entre la noción débil basada en garantía y la noción fuerte basada en derivación. Para esto, al igual que como se realizó en [GGK+11], se continuará con el estudio de las diferentes formas de influencia para así poder obtener acciones mentales de contracción más racionales.

Influencia Argumental

La noción de influencia se basará en el análisis del efecto que produce la remoción de los argumentos para el literal que se quiere contraer en los árboles de dialéctica. Para

realizar este análisis, a continuación se introducirá el concepto de árbol recortado, el cual básicamente es un árbol de dialéctica donde se remueven todos los argumentos (y los subárboles enraizados en estos argumentos) para un literal tipado determinado.

Definición 7.8 (Árbol de Dialéctica Recortado) *Sea $\mathcal{T}^*_{\mathcal{A}}$ un árbol de dialéctica marcado del programa T-DeLP asociado a un agente* ArgAPL *en una configuración, y L^X un literal tipado tal que $X \in \{B, AG, MG\}$. El árbol de dialéctica $\mathcal{T}^*_{\mathcal{A}}$ recortado para L^X, notado como $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$, es una versión árbol $\mathcal{T}^*_{\mathcal{A}}$ sin los subárboles $\mathcal{T}^*$' de $\mathcal{T}^*_{\mathcal{A}}$, donde $\mathcal{T}^*$' tiene como raíz un argumento representativo $\mathcal{B}$ tal que $\mathcal{B}$ tiene un subargumento representativo $\mathcal{C}$ cuya conclusión es L^X.*

Una propiedad importante que permitirá caracterizar la primer noción de influencia es que sólo al remoción de argumentos marcados como **U** es la que puede hacer cambiar la marca de la raíz de un árbol de dialéctica. Este resultado se encuentra formalizado por la siguiente proposición.

Proposición 7.3 *Sea L^X un literal tipado tal que $X \in \{B, AG, MG\}$, y $\mathcal{T}^*_{\mathcal{A}}$ un árbol de dialéctica marcado del programa T-DeLP asociado a un agente* ArgAPL *en una configuración. Si $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$ es no vacío y la marca de la raíz de $\mathcal{T}^*_{\mathcal{A}}$ difiere de la marca de la raíz de $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$, entonces existe un argumento representativo $\mathcal{B}$ con un subargumento representativo $\mathcal{C}$ para L^X que está marcado como **U** en $\mathcal{T}^*_{\mathcal{A}}$.*

*Prueba : Suponga que $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$ es no vacío, que la marca de la raíz de $\mathcal{T}^*_{\mathcal{A}}$ difiere de la marca de la raíz de $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$, y que no existe un argumento $\mathcal{B}$ que tenga un subargumento $\mathcal{C}$ para L^X que esté marcado como **U** en $\mathcal{T}^*_{\mathcal{A}}$. Entonces ocurre una de las siguientes situaciones:*

1. *En $\mathcal{T}^*_{\mathcal{A}}$ no hay ningún argumento $\mathcal{B}$ con subargumento $\mathcal{C}$ para L^X. Entonces por Definición 7.8 $\mathcal{T}^*_{\mathcal{A}} = \mathcal{T}^*_{\mathcal{A}} \setminus L^X$, lo cual contradice la hipótesis.*

2. *En $\mathcal{T}^*_{\mathcal{A}}$ existe al menos un argumento $\mathcal{B}$ con subargumento $\mathcal{C}$ para L^X, con lo cual se da uno de los siguientes casos:*

   a) *$\mathcal{A} = \mathcal{B}$, es decir, $\mathcal{B}$ es la raíz de $\mathcal{T}^*_{\mathcal{A}}$. Por lo tanto, por Definición 7.8 $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$ es vacío, lo cual contradice la hipótesis.*

b) *$\mathcal{B}$ es un nodo interno de $\mathcal{T}^*_{\mathcal{A}}$. Por hipótesis se tiene que $\mathcal{B}$ aparece como* **D** *en $\mathcal{T}^*_{\mathcal{A}}$. Entonces, por Definición 5.31 de marcado del árbol de dialéctica, si se remueve todo el subárbol enraizado en $\mathcal{B}$ no cambiará el marcado del padre de $\mathcal{B}$. Por lo tanto, por Definición 7.8, la marca de todo argumento en $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$ será la misma que en $\mathcal{T}^*_{\mathcal{A}}$. En particular, la marca de la raíz de $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$ será la misma que la de $\mathcal{T}^*_{\mathcal{A}}$, lo cual contradice la hipótesis.* □

La Proposición 7.3 establece que un argumento para L^X puede exhibir influencia únicamente si está marcado como no derrotado en un árbol de dialéctica. Esto conduce a la primer y más general noción de influencia, llamada *influencia argumental*.

Definición 7.9 (Influencia Argumental) *Un literal tipado L^X tal que $X \in \{B, AG, MG\}$ exhibe influencia argumental sobre el programa T-DeLP $\mathcal{P}(Ag, C)$ asociado a un agente* ArgAPL *Ag en una configuración C, notado $L^X \hookrightarrow_{\mathcal{A}} \mathcal{P}(Ag, C)$, si y solo si existe un argumento $\mathcal{B}$ con un subargumento $\mathcal{C}$ para L^X tal que $\mathcal{B}$ aparece marcado como* **U** *en algún árbol de dialéctica de $\mathcal{P}(Ag, C)$.*

A partir de esta noción de influencia se presentará una acción mental que utiliza un operador de contracción que contempla estas intuiciones. Básicamente, este operador asegurará que luego de contraer un literal tipado no habrá influencia argumental de ese literal en el programa resultante. A continuación se define cuál es el resultado de aplicar este operador para un literal tipado sobre el programa asociado a un agente en una configuración.

Definición 7.10 (Contracción basada en influencia argumental) *Sea $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ un programa T-DeLP asociado a un agente* ArgAPL *Ag en una configuración C, y L^X un literal tipado tal que $X \in \{B, AG, MG\}$. Una contracción basada en influencia argumental de L^X en $\mathcal{P}(Ag, C)$, notada $\mathcal{P}(Ag, C) \xrightarrow{IA} L^X$, es tal que $\mathcal{P}(Ag, C) \xrightarrow{IA} L^X = (\Gamma, \Theta^P, (\Delta^B \cup \Delta^G) \setminus S_{ia})$, donde:*

1. *$S_{ia} \subseteq S$, con S el conjunto de posibles remociones de L^X en $\mathcal{P}(Ag, C)$;*
2. *$L^X \not\hookrightarrow_{\mathcal{A}} (\Gamma, \Theta^P \setminus \{L^P\}, (\Delta^B \cup \Delta^G) \setminus S_{ia})$; y*
3. *S_{ia} es minimal: no existe $S'_{ia} \subset S_{ia}$ tal que satisface las condiciones 1 y 2.*

Ejemplo 7.4 *Considere el programa T-DeLP $\mathcal{P}(Ag_7, C_7)$ resultante de la configuración C_7 presentado en el Ejemplo 7.1. Al realizar $\mathcal{P}(Ag_7, C_7) \overset{IA}{\sim} a^{AG}$ se removerán de $\mathcal{P}(Ag_7, C_7)$ todas las reglas rebatibles con cabeza a^{AG} que formen parte de argumento que aparezcan marcados como* **U** *en algún árbol de dialéctica de $\mathcal{P}(Ag_7, C_7)$. En la Figura 7.1 presentada en el Ejemplo 7.1 se puede observar que los argumentos que verifican esta condición son $\mathcal{A}_2$ y $\mathcal{A}_3$. Por lo tanto, el conjunto de reglas a remover de C será $S_{ia_7} = \{(a^{AG} \prec q^B, c^B), (a^{AG} \prec p^B)\}$. A continuación, en la Figura 7.4 se presentan los árboles de dialéctica obtenidos a partir del programa resultante de efectuar $\mathcal{P}(Ag_7, C_7) \overset{IA}{\sim} a^{AG}$.*

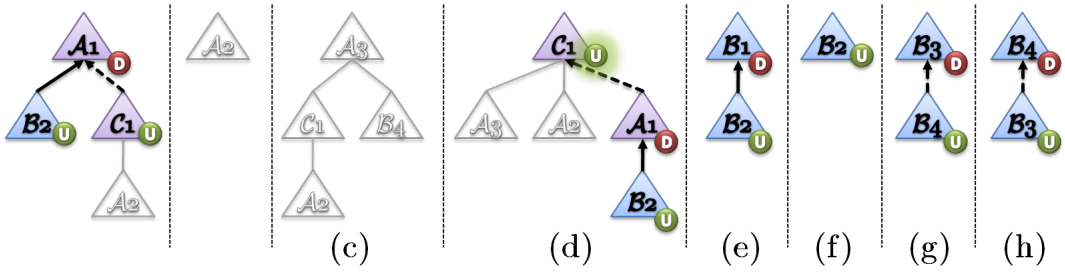


Figura 7.4: Árboles de dialéctica para el programa $\mathcal{P}(Ag_7, C_7) \overset{\mathsf{IA}}{\sim} a^{AG}$.

Note que de esta manera el argumento $\mathcal{C}_1$ queda garantizado, ya que los argumentos para a^{AG} que lo influenciaban fueron removidos. Por lo tanto, observe que el estado de garantía de $\mathcal{C}_1$ es el mismo que para la contracción basada en derivación, y diferente del obtenido con la contracción basada en garantía.

A continuación se presentará la acción mental de contracción que utiliza esta contracción para actualizar la base de conocimiento de un agente. Estas acciones mentales serán llamadas *acciones mentales de contracción-IA* y, al igual que para las acciones mentales anteriormente presentadas en este capítulo, se mostrará su sintaxis y su semántica en ArgAPL.

Definición 7.11 (Acción mental de contracción-IA) *Una acción mental de contracción-IA (o iacma) es una tupla $IAcma = (\beta, Name, L^X)$, donde β es un conjunto de precondiciones, $Name$ es un átomo para referenciar a $IAcma$ desde un plan, y L^X es un literal tipado tal que $X \in \{B, AG, MG\}$.*

Definición 7.12 (Ejecución de acción mental de contracción-IA) *Sea* $C = (\Theta^P, \Delta^B, \Delta^G, ([niacma, a_0, \ldots, a_m], \kappa))$ *la configuración de un agente* ArgAPL *Ag, donde niacma es el nombre de la acción mental de contracción-IA* $iacma = (\beta, niacma, L^X)$ *con* $X \in \{B, AG, MG\}$. *Si* $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ *es el programa T-DeLP asociado a Ag en C, la ejecución de iacma en C se define como:*

$$\frac{(C \vdash_S \beta) \wedge (C \vdash_S \kappa) \wedge L^P \notin \Theta^P \wedge \mathcal{P}(Ag, C) \xrightarrow{IA} L^X = (\Gamma, \Theta^P, \Delta^{B\prime} \cup \Delta^{G\prime}) \wedge (\forall L^{MG} \in \mathsf{Mantenidas}(C) : (\Theta^P, \Delta^{B\prime}, \Delta^{G\prime}, ([a_0, \ldots, a_m], \kappa)) \vdash_S L^B)}{(\Theta^P, \Delta^B, \Delta^G, ([niacma, a_0, \ldots, a_m], \kappa)) \rightarrow (\Theta^P, \Delta^{B\prime}, \Delta^{G\prime}, ([a_0, \ldots, a_m], \kappa))}$$

Para mostrar que esta acción mental es en efecto una acción mental de contracción, primero será necesario mostrar que si un literal tipado no tiene influencia argumental en un programa T-DeLP entonces tampoco estará garantizado en ese programa.

Proposición 7.4 *Sea* $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ *el programa T-DeLP asociado a un agente* ArgAPL *Ag en una configuración C, y* L^X *un literal tipado tal que* $X \in \{B, AG, MG\}$. *Si* $L^X \not\rightarrow_A \mathcal{P}(Ag, C)$, *entonces* $\mathcal{P}(Ag, C)$ *no garantiza a* L^X.

Prueba : Si $L^X \not\rightarrow_A \mathcal{P}(Ag, C)$ *entonces, por Definición 7.9 de influencia argumental, no existe ningún argumento* $\mathcal{B}$ *con un subargumento* $\mathcal{C}$ *para* L^X *que aparezca marcado como* **U** *en algún árbol de dialéctica de* $\mathcal{P}(Ag, C)$. *Entonces, en particular, todo* $\mathcal{C}$ *para* L^X *tampoco aparecerá marcado como* **U**. *Luego, como no existe ningún argumento para* L^X *marcado* **U**, L^X *no está garantizado en* $\mathcal{P}(Ag, C)$. □

De esta manera, utilizando la proposición anterior, a continuación se mostrará que la acción mental de contracción-IA es en efecto una acción mental de contracción.

Proposición 7.5 *Sea* $C = (\Theta^P, \Delta^B, \Delta^G, ([niacma, a_0, \ldots, a_m], \kappa))$ *la configuración de un agente* ArgAPL *Ag, donde niacma es el nombre de la acción mental de contracción-IA* $iacma = (\beta, niacma, L^X)$ *con* $X \in \{B, AG, MG\}$, *y* $L^P \notin \Theta^P$. *Si* C' *es la configuración resultante de ejecutar iacma en C, entonces* $C' \nvdash_S L^X$.

Prueba : Sea $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ *el programa T-DeLP asociado a Ag en C. Sea* $(\Gamma, \Theta^P, \Delta^{B\prime} \cup \Delta^{G\prime})$ *el programa resultante de efectuar* $\mathcal{P}(Ag, C) \xrightarrow{IA} L^X$. *Por Definición 7.12 de ejecución de acción mental de contracción-IA* $(\Gamma, \Theta^P, \Delta^{B\prime} \cup \Delta^{G\prime})$ *será el programa asociado a Ag en la configuración* C'. *Luego, por Definición 7.10, vale que* $L^X \not\rightarrow_A (\Gamma, \Theta^P, \Delta^{B\prime} \cup \Delta^{G\prime})$. *Entonces, por Proposición 7.4* $(\Gamma, \Theta^P, \Delta^{B\prime} \cup \Delta^{G\prime})$ *no garantiza* L^X, *con lo cual* $C' \nvdash_S L^X$. □

Influencia Arbórea

En la sección anterior se mostró que identificando los argumentos no derrotados para el literal L^X a remover es posible eliminar la influencia del literal en la configuración resultante. Sin embargo, puede ocurrir que los argumentos marcados como **U** que contienen subargumentos para el literal L^X no necesariamente exhiban una influencia razonable. Por ejemplo, sean $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{Z}$ argumentos representativos, con $\mathcal{Z}$ tal que tiene un subargumento para L^X, y $\mathcal{A}, \mathcal{B}, \mathcal{C}$ no tienen subargumentos para L^X. Considere los árboles de dialéctica para estos argumentos ilustrados en la Figura 7.5.

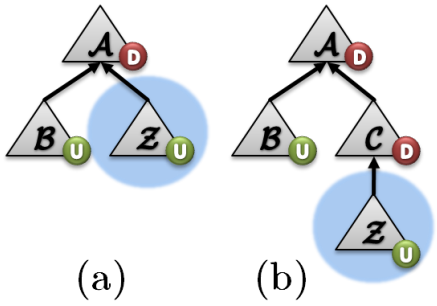


Figura 7.5: Árboles de dialéctica en los que $\mathcal{Z}$ no necesariamente influye en el estado de la raíz.

Note que en ambos árboles $\mathcal{Z}$ aparece como no derrotado. Por lo tanto, bajo la noción de influencia argumental, L^X estaría influenciando al programa T-DeLP. Sin embargo, si se remueve $\mathcal{Z}$ de ambos árboles, el estado de la raíz no cambiará. En particular, en el árbol (a) $\mathcal{B}$ seguirá derrotando a $\mathcal{A}$, con lo cual $\mathcal{A}$ seguiría marcado como **D** aún cuando $\mathcal{Z}$ fue removido. La situación en el árbol (b) es similar, ya que a pesar de la remoción de $\mathcal{Z}$, $\mathcal{C}$ seguiría marcado como **U**.

Por lo tanto, intuitivamente, la remoción de $\mathcal{Z}$ (o más específicamente la remoción de las reglas de $\mathcal{Z}$ que tienen como cabeza L^X) es innecesaria en situaciones como esta. Esto conduce a la siguiente noción de influencia, la cual sólo tomará en cuenta argumentos cuya remoción cambiará el marcado de la raíz. Esta forma de influencia será denotada *influencia arbórea*.

Definición 7.13 (Influencia arbórea) *Un literal tipado L^X tal que $X \in \{B, AG, MG\}$ exhibe influencia arbórea sobre un programa T-DeLP $\mathcal{P}(Ag, C)$ asociado a un agente*

ArgAPL *Ag en una configuración C, notado* $L^X \hookrightarrow_T \mathcal{P}(Ag, C)$*, si y solo si existe un árbol de dialéctica* $\mathcal{T}^*_{\mathcal{A}}$ *en* $\mathcal{P}(Ag, C)$ *tal que:*

- *la marca de la raíz de* $\mathcal{T}^*_{\mathcal{A}}$ *difiere de la marca de la raíz de* $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$*; o*
- *la marca de la raíz de* $\mathcal{T}^*_{\mathcal{A}}$ *es* **U** *y* $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$ *es vacío.*

Al igual que lo realizado para la influencia argumental, la influencia arbórea permitirá definir un operador de contracción para luego definir una acción mental basada en ese operador. A continuación se presentará el operador de contracción basado en influencia arbórea y luego las *acciones mentales de contracción-IT*.

Definición 7.14 (Contracción basada en influencia arbórea) *Sea* $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ *un programa T-DeLP asociado a un agente* ArgAPL *Ag en una configuración C, y* L^X *un literal tipado tal que* $X \in \{B, AG, MG\}$*. Una contracción basada en influencia arbórea de* L^X *en* $\mathcal{P}(Ag, C)$*, notada* $\mathcal{P}(Ag, C) \xrightarrow{IT} L^X$*, es tal que* $\mathcal{P}(Ag, C) \xrightarrow{IT} L^X = (\Gamma, \Theta^P, (\Delta^B \cup \Delta^G) \setminus S_{it})$*, donde:*

1. $S_{it} \subseteq S$*, con S el conjunto de posibles remociones de* L^X *en* $\mathcal{P}(Ag, C)$*;*
2. $L^X \not\hookrightarrow_T (\Gamma, \Theta^P \setminus \{L^P\}, (\Delta^B \cup \Delta^G) \setminus S_{it})$*; y*
3. S_{it} *es minimal: no existe* $S'_{it} \subset S_{it}$ *tal que satisface las condiciones 1 y 2.*

Definición 7.15 (Acción mental de contracción-IT) *Una acción mental de contracción-IT (o itcma) es una tupla* $ITcma = (\beta, Name, L^X)$*, donde* β *es un conjunto de precondiciones, Name es un átomo para referenciar a ITcma desde un plan, y* L^X *es un literal tipado tal que* $X \in \{B, AG, MG\}$*.*

Definición 7.16 (Ejecución de acción mental de contracción-IT) *Sea* $C = (\Theta^P, \Delta^B, \Delta^G, ([nitcma, a_0, \ldots, a_m], \kappa))$ *la configuración de un agente* ArgAPL *Ag, donde nitcma es el nombre de la acción mental de contracción-IT* $itcma = (\beta, nitcma, L^X)$ *con* $X \in \{B, AG, MG\}$*. Si* $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ *es el programa T-DeLP asociado a Ag en C, la ejecución de itcma en C se define como:*

$$\frac{(C \vdash_S \beta) \wedge (C \vdash_S \kappa) \wedge L^P \notin \Theta^P \wedge \mathcal{P}(Ag, C) \xrightarrow{IT} L^X = (\Gamma, \Theta^P, \Delta^{B'} \cup \Delta^{G'}) \wedge \quad (\forall L^{MG} \in \mathsf{Mantenidas}(C) : (\Theta^P, \Delta^{B'}, \Delta^{G'}, ([a_0, \ldots, a_m], \kappa)) \vdash_S L^B)}{(\Theta^P, \Delta^B, \Delta^G, ([nitcma, a_0, \ldots, a_m], \kappa)) \rightarrow (\Theta^P, \Delta^{B'}, \Delta^{G'}, ([a_0, \ldots, a_m], \kappa))}$$

Para mostrar que esta acción mental es en efecto una acción mental de contracción, será necesario mostrar antes que si un literal tipado no tiene influencia arbórea en un programa T-DeLP, entonces tampoco estará garantizado en ese programa.

Proposición 7.6 *Sea* $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ *el programa T-DeLP asociado a un agente* ArgAPL *Ag en una configuración C, y* L^X *un literal tipado tal que* $X \in \{B, AG, MG\}$. *Si* $L^X \not\leftrightarrow_{\mathsf{T}} \mathcal{P}(Ag, C)$, *entonces* $\mathcal{P}(Ag, C)$ *no garantiza a* L^X.

Prueba : *Para que* L^X *esté garantizado en* $\mathcal{P}(Ag, C)$ *tiene que existir un árbol de dialéctica que tenga como raíz un argumento para* L^X *que esté marcado como* **U**. *Por otra parte, por Definición 7.8, si* $\mathcal{T}^*_{\mathcal{B}}$ *es un árbol para un argumento de* L^X *entonces el árbol* $\mathcal{T}^*_{\mathcal{B}} \setminus L^X$ *es vacío. Luego, como por hipótesis* $L^X \not\leftrightarrow_{\mathsf{T}} \mathcal{P}(Ag, C)$, *entonces por Definición 7.13 de influencia arbórea no existe ningún árbol de dialéctica* $\mathcal{T}^*_{\mathcal{A}}$ *en* $\mathcal{P}(Ag, C)$ *tal que su raíz está marcada como* **U** *y* $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$ *sea vacío. Por lo tanto, en* $\mathcal{P}(Ag, C)$ *no existe ningún árbol de dialéctica enraizado en un argumento para* L^X *que esté marcado como* **U**, *con lo cual* L^X *no está garantizado en* $\mathcal{P}(Ag, C)$. □

Por lo tanto, utilizando la proposición anterior, a continuación se mostrará que la acción mental de contracción-IT cumple los requisitos de las acciones mentales basadas en contracción.

Proposición 7.7 *Sea* $C = (\Theta^P, \Delta^B, \Delta^G, ([nitcma, a_0, \ldots, a_m], \kappa))$ *la configuración de un agente* ArgAPL *Ag, donde nitcma es el nombre de la acción mental de contracción-IT* $itcma = (\beta, nitcma, L^X)$ *con* $X \in \{B, AG, MG\}$, *y* $L^P \notin \Theta^P$. *Si* C' *es la configuración resultante de ejecutar itcma en C, entonces* $C' \nvdash_S L^X$.

Prueba : *Sea* $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ *el programa T-DeLP asociado a Ag en C. Sea* $(\Gamma, \Theta^P, \Delta^{B'} \cup \Delta^{G'})$ *el programa resultante de efectuar* $\mathcal{P}(Ag, C) \xrightarrow{IT} L^X$. *Por Definición 7.16 de ejecución de acción mental de contracción-IT* $(\Gamma, \Theta^P, \Delta^{B'} \cup \Delta^{G'})$ *será el programa asociado a Ag en la configuración* C'. *Luego, Por Definición 7.14 vale que* $L^X \not\leftrightarrow_{\mathsf{T}} (\Gamma, \Theta^P, \Delta^{B'} \cup \Delta^{G'})$. *Entonces, por Proposición 7.6* $(\Gamma, \Theta^P, \Delta^{B'} \cup \Delta^{G'})$ *no garantiza* L^X, *con lo cual* $C' \nvdash_S L^X$. □

Influencia de Garantía

En la sección anterior se presentaron acciones mentales de contracción basadas en una noción más refinada del concepto de influencia. Para determinar si un literal tipado exhibe

influencia arbórea se consideró cada árbol de dialéctica de forma aislada. Sin embargo, recuerde que para que un literal tipado esté garantizado alcanza con que exista un solo árbol enraizado en un argumento para ese literal que esté marcado como **U**. Por ejemplo, considere los argumentos representativos $\mathcal{A}, \mathcal{B}, \mathcal{Z}$, donde $\mathcal{Z}$ tiene un subargumento para L^X mientras que $\mathcal{A}$ y $\mathcal{B}$ soportan a un literal tipado M^Y y no tienen un subargumento para L^X. Considere los árboles de dialéctica que involucran a estos argumentos ilustrados en la Figura 7.6.

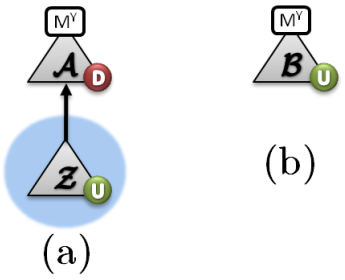


Figura 7.6: Árboles de dialéctica donde $\mathcal{Z}$ no necesariamente influencia el estado de garantía del literal tipado M^Y.

Como puede observarse en la Figura 7.6, M^Y está garantizado por el argumento $\mathcal{B}$, el cual aparece como no derrotado en su propio árbol (b). No obstante, $\mathcal{Z}$ derrota a $\mathcal{A}$ en el árbol (a) haciendo que L^X tenga influencia arbórea en el programa T-DeLP, ya que si es removido $\mathcal{A}$ cambiaría su marcado. Sin embargo, independientemente de la influencia de L^X sobre el programa, M^Y seguiría garantizado por el argumento $\mathcal{B}$. Por lo tanto, la remoción de $\mathcal{Z}$ (o las reglas con cabeza L^X en $\mathcal{Z}$) es claramente innecesaria en esta situación. Estas consideraciones conducen a la última noción de influencia denotada *influencia de garantía*.

Definición 7.17 (Influencia de garantía) *Un literal tipado L^X tal que $X \in \{B, AG, MG\}$ exhibe influencia de garantía sobre un programa T-DeLP $\mathcal{P}(Ag, C)$ asociado a un agente* ArgAPL *Ag en una configuración C, notado $L^X \hookrightarrow_W \mathcal{P}(Ag, C)$, si y solo si existe un literal tipado M^Y con $Y \in \{B, AG, MG\}$ tal que:*

- *M^Y está garantizado en $\mathcal{P}(Ag, C)$ y para todo árbol de dialéctica $\mathcal{T}^*_{\mathcal{A}}$ con $\mathcal{A}$ un argumento para M^Y vale que la raíz de $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$ está como marcada* **D** *o $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$ es vacío; o bien*

- *M^Y no está garantizado en $\mathcal{P}(Ag, C)$ y existe un árbol de dialéctica $\mathcal{T}^*_{\mathcal{A}}$ con $\mathcal{A}$ un argumento para M^Y tal que la raíz de $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$ está marcada como* **U**.

Al igual que lo realizado para las nociones de influencia anteriormente presentadas, a continuación se definirá el operador contracción basado en influencia de garantía y posteriormente la sintaxis y semántica de las acciones mentales basadas en este operador. El operador será llamado contracción basada en influencia de garantía y las acciones mentales se llamarán *acciones mentales de contracción-IW*.

Definición 7.18 (Contracción basada en influencia de garantía) *Sea $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ un programa T-DeLP asociado a un agente* ArgAPL *Ag en una configuración C, y L^X un literal tipado tal que $X \in \{B, AG, MG\}$. Una contracción basada en influencia de garantía de L^X en $\mathcal{P}(Ag, C)$, notada $\mathcal{P}(Ag, C) \xrightarrow{IW} L^X$, es tal que $\mathcal{P}(Ag, C) \xrightarrow{IW} L^X = (\Gamma, \Theta^P, (\Delta^B \cup \Delta^G) \setminus S_{iw})$, donde:*

1. *$S_{iw} \subseteq S$, con S el conjunto de posibles remociones de L^X en $\mathcal{P}(Ag, C)$;*
2. *$L^X \not\sim\!\!\!\rightarrow_W (\Gamma, \Theta^P \setminus \{L^P\}, (\Delta^B \cup \Delta^G) \setminus S_{iw})$; y*
3. *S_{iw} es minimal: no existe $S'_{iw} \subset S_{iw}$ tal que satisface las condiciones 1 y 2.*

Definición 7.19 (Acción mental de contracción-IW) *Una acción mental de contracción-IW (o iwcma) es una tupla $IWcma = (\beta, Name, L^X)$, donde β es un conjunto de precondiciones, $Name$ es un átomo para referenciar a $IWcma$ desde un plan, y L^X es un literal tipado tal que $X \in \{B, AG, MG\}$.*

Definición 7.20 (Ejecución de acción mental de contracción-IT) *Sea $C = (\Theta^P, \Delta^B, \Delta^G, ([niwcma, a_0, \ldots, a_m], \kappa))$ la configuración de un agente* ArgAPL *Ag, donde niwcma es el nombre de la acción mental de contracción-IW $iwcma = (\beta, niwcma, L^X)$ con $X \in \{B, AG, MG\}$. Si $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ es el programa T-DeLP asociado a Ag en C, la ejecución de iwcma en C se define como:*

$$\frac{(C \vdash_S \beta) \wedge (C \vdash_S \kappa) \wedge L^P \notin \Theta^P \wedge \mathcal{P}(Ag, C) \xrightarrow{IW} L^X = (\Gamma, \Theta^P, \Delta^{B\prime} \cup \Delta^{G\prime}) \wedge (\forall L^{MG} \in \textsf{Mantenidas}(C) : (\Theta^P, \Delta^{B\prime}, \Delta^{G\prime}, ([a_0, \ldots, a_m], \kappa)) \vdash_S L^B)}{(\Theta^P, \Delta^B, \Delta^G, ([niwcma, a_0, \ldots, a_m], \kappa)) \rightarrow (\Theta^P, \Delta^{B\prime}, \Delta^{G\prime}, ([a_0, \ldots, a_m], \kappa))}$$

Para mostrar que esta acción mental es en efecto una acción mental de contracción, será necesario mostrar que si un literal tipado no tiene influencia garantía en un programa T-DeLP, entonces tampoco estará garantizado en ese programa.

Proposición 7.8 *Sea* $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ *el programa T-DeLP asociado a un agente* ArgAPL *Ag en una configuración* C*, y* L^X *un literal tipado tal que* $X \in \{B, AG, MG\}$*. Si* $L^X \not\hookrightarrow_W \mathcal{P}(Ag, C)$*, entonces* $\mathcal{P}(Ag, C)$ *no garantiza a* L^X*.*

Prueba : Suponga que L^X *está garantizado en* $\mathcal{P}(Ag, C)$*. Por lo tanto, existe un árbol de dialéctica* $\mathcal{T}^*_{\mathcal{A}}$ *donde* $\mathcal{A}$ *es un argumento para* L^X *y está marcado como* **U***. Luego, por Definición 7.8 se tiene que* $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$ *es vacío. Entonces, por Definición 7.17* L^X*,* $\hookrightarrow_W$ $\mathcal{P}(Ag, C)$*, lo cual contradice la hipótesis.* □

Proposición 7.9 *Sea* $C = (\Theta^P, \Delta^B, \Delta^G, ([niwcma, a_0, \ldots, a_m], \kappa))$ *la configuración de un agente* ArgAPL *Ag, donde niwcma es el nombre de la acción mental de contracción-IW* $iwcma = (\beta, niwcma, L^X)$ *con* $X \in \{B, AG, MG\}$*, y* $L^P \notin \Theta^P$*. Si* C' *es la configuración resultante de ejecutar iwcma en* C*, entonces* $C' \nvdash_S L^X$*.*

Prueba : Sea $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ *el programa T-DeLP asociado a Ag en* C*. Sea* $(\Gamma, \Theta^P, \Delta^{B'} \cup \Delta^{G'})$ *el programa resultante de efectuar* $\mathcal{P}(Ag, C) \xrightarrow{IW} L^X$*. Por Definición 7.20 de ejecución de acción mental de contracción-IW* $(\Gamma, \Theta^P, \Delta^{B'} \cup \Delta^{G'})$ *será el programa asociado a Ag en la configuración* C'*. Luego, por Definición 7.18 vale que* $L^X \not\hookrightarrow_W (\Gamma, \Theta^P, \Delta^{B'} \cup \Delta^{G'})$*. Entonces, por Proposición 7.8 se tiene que* $(\Gamma, \Theta^P, \Delta^{B'} \cup \Delta^{G'})$ *no garantiza* L^X*, con lo cual* $C' \nvdash_S L^X$*.* □

Esta última acción mental de contracción introdujo la versión más refinada de influencia y, por lo tanto, es la acción mental que más reglas rebatibles de la base de conocimiento preserva. Esta es la acción mental más cercana, en términos de cantidad de reglas removidas, a las acciones mentales de contracción-w. Por otra parte, las acciones mentales de contracción basadas en influencia argumental son las más cercanas a las acciones mentales de contracción basadas en derivación. Por lo tanto, es posible pensar en una jerarquía de acciones que va desde las mas incisivas (dado que remueven más reglas) a las menos incisivas (porque remueven menos reglas). Esta jerarquía es ilustrada a continuación en la Figura 7.7.

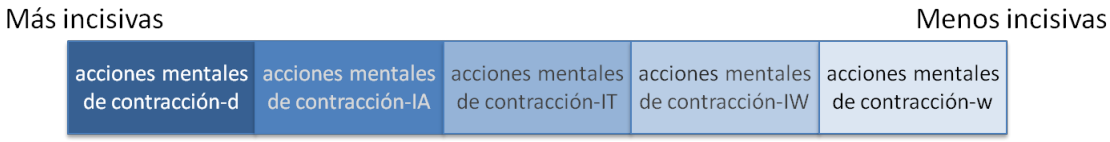


Figura 7.7: Relación jerárquica entre las acciones mentales de contracción.

Esta jerarquía se construye a partir de las nociones de derivación, influencia y garantía sobre las cuales se construyen los operadores en que se basan las acciones mentales de contracción. En la siguiente proposición se muestran la relaciones existentes entre las diferentes nociones, que dan lugar a esta jerarquía.

Proposición 7.10 *Sea* $\mathcal{P}(Ag, C) = (\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$ *un programa T-DeLP asociado a un agente* ArgAPL *Ag en una configuración C, y* L^X *un literal tipado tal que* $X \in \{B, AG, MG\}$. *Entonces:*

1. *Si* $\mathcal{P}(Ag, C)$ *garantiza* L^X, *entonces* $L^X \hookrightarrow_W \mathcal{P}(Ag, C)$.
2. *Si* $L^X \hookrightarrow_W \mathcal{P}(Ag, C)$, *entonces* $L^X \hookrightarrow_T \mathcal{P}(Ag, C)$.
3. *Si* $L^X \hookrightarrow_T \mathcal{P}(Ag, C)$, *entonces* $L^X \hookrightarrow_A \mathcal{P}(Ag, C)$.
4. *Si* $L^X \hookrightarrow_A \mathcal{P}(Ag, C)$, *entonces* $\mathcal{P}(Ag, C) \nvdash\sim L^X$.

Prueba :

1. *Como* L^X *está garantizado en* $\mathcal{P}(Ag, C)$ *existe un árbol de dialéctica* $\mathcal{T}^*_{\mathcal{A}}$ *donde* $\mathcal{A}$ *es un argumento para* L^X *y está marcado como* **U**. *Luego, por Definición 7.8* $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$ *es vacío. Por lo tanto, por Definición 7.17, se tiene que* $L^X \hookrightarrow_W \mathcal{P}(Ag, C)$. □
2. *Dado que* $L^X \hookrightarrow_W \mathcal{P}(Ag, C)$, *por Definición 7.17 existe un árbol de dialéctica* $\mathcal{T}^*_{\mathcal{A}}$ *tal que:*

   a) *Si la raíz de* $\mathcal{T}^*_{\mathcal{A}}$ *es* **D**, *entonces la raíz de* $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$ *es* **U**.

   b) *Si la raíz de* $\mathcal{T}^*_{\mathcal{A}}$ *es* **U**, *entonces la raíz de* $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$ *es* **D** *o* $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$ *es vacío.*

   Por lo tanto, por Definición 7.13, $L^X \hookrightarrow_T \mathcal{P}(Ag, C)$. □

3. *$L^X \hookrightarrow_T \mathcal{P}(Ag, C)$, por Definición 7.13 se da uno de los siguientes casos:*

   a) *Existe un árbol de dialéctica $\mathcal{T}^*_{\mathcal{A}}$ con raíz* **U** *y $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$ es vacío. Luego, por definición 7.8, como $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$ es vacío $\mathcal{A}$ tiene un subargumento para L^X. Por lo tanto, como $\mathcal{A}$ está marcado* **U** *en $\mathcal{T}^*_{\mathcal{A}}$, se tiene que, por Definición 7.9, $L^X \hookrightarrow_A \mathcal{P}(Ag, C)$.*

   b) *Existe un árbol de dialéctica $\mathcal{T}^*_{\mathcal{A}}$ tal que la marca de su raíz difiere de la maraca de la raíz de $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$, con lo cual $\mathcal{T}^*_{\mathcal{A}} \neq \mathcal{T}^*_{\mathcal{A}} \setminus L^X$. Luego, por Definición 7.8, existe algún argumento $\mathcal{B}$ en $\mathcal{T}^*_{\mathcal{A}}$ que tiene como subargumento un argumento para L^X que no aparece en $\mathcal{T}^*_{\mathcal{A}} \setminus L^X$. Por Proposición 7.3 sólo los argumentos marcados como* **U** *pueden cambiar la marca de la raíz de un árbol. Por lo tanto, $\mathcal{B}$ está* **U** *en $\mathcal{T}^*_{\mathcal{A}}$ y, por Definición 7.9, se tiene que $L^X \hookrightarrow_A \mathcal{P}(Ag, C)$.* □

4. *Considerando que $L^X \hookrightarrow_A \mathcal{P}(Ag, C)$, por definición 7.9 existe un argumento para L^X. Luego, por Definición 5.15 de argumento se tiene que $\mathcal{P}(Ag, C) \mid\sim L^X$.* □

Por otra parte, una relación entre las acciones mentales de contracción es que todas (a excepción de las acciones basadas en la noción de garantía) no exhiben influencia. Como se mostró anteriormente, la diferencia entre ellas está en cuan incisivas son. De esta manera, las acciones mentales de contracción basadas en garantía, si bien son las menos incisivas, son las únicas que pueden dejar reglas que activen argumentos que finalmente terminan influenciando al resultado del razonamiento del agente.

7.2. Poda heurística en el proceso de razonamiento de los agentes ArgAPL

En esta sección se presentará un método basado en una técnica de poda con el objetivo de reducir el tamaño de los árboles de dialéctica construidos al calcular las creencias y metas actuales del agente ArgAPL. Por lo tanto, un menor número de argumentos será utilizado para determinar el estado de la raíz de estos árboles, acelerando así el cálculo de estos elementos del agente ArgAPL. Esta técnica constituye una adaptación del método presentado en [RGGS11], el cual fue realizado en conjunto con Nicolás D. Rotstein.

Las podas para los árboles de dialéctica se producen aprovechando que estos árboles se construyen siguiendo la política de primero en profundidad, así como también la forma en que está definido el marcado. Básicamente, las podas establecen que si se encontró un derrotador marcado como **U** no tiene sentido seguir explorando derrotadores ya que el nodo padre quedará marcado como **D**. El método que se presentará en esta sección buscará tratar de explorar los nodos **U** lo mas rápido posible para así podar la mayor cantidad de nodos.

El enfoque utilizado para la técnica de poda propuesta se basa en la noción de *fuerza argumental*, la cual indica la probabilidad de que un argumento sea derrotado. Se propondrá una fórmula concreta que captura esta intuición. Esta fuerza será calculada de manera *offline* para todos los argumentos. Luego, la fuerza argumental será utilizada como un valor heurístico para ordenar los derrotadores de un nodo interno en la construcción del árbol de dialéctica para una creencia o meta actual. Se mostrará que este orden fomenta la oportunidades de poda y que, por lo tanto, subárboles completos pueden ser obviados en la construcción del árbol sin afectar el estado de garantía de la raíz. Para esto se introducirá una estrategia formal para construir estos *bonsai de dialéctica*[1] basada en la propuesta de fuerza argumental. De esta manera, los bonsai de dialéctica serán utilizados en las configuraciones de los agentes ArgAPL para determinar sus creencias y metas actuales.

En particular, dado que se requiere que la fuerza sea calculada de manera *offline*, se mostrará cómo calcular esta fuerza para cada argumento en base a la especificación de un agente ArgAPL. Para esto se identificarán todos los argumentos que el agente podría llegar a tener en sus configuraciones. Aun así, dada la naturaleza de la técnica presentada en esta sección, solo podrán aplicarse de la manera más eficiente a un subconjunto significativo de agentes ArgAPL. Este subconjunto estará caracterizado por aquellos agentes que no utilizan acciones mentales, es decir, que no modifican dinámicamente sus bases de creencias y/o metas. Si bien esto supone una desventaja, es posible aplicar la técnica a agentes que utilizan acciones mentales, en cuyo caso tendrá una menor efectividad.

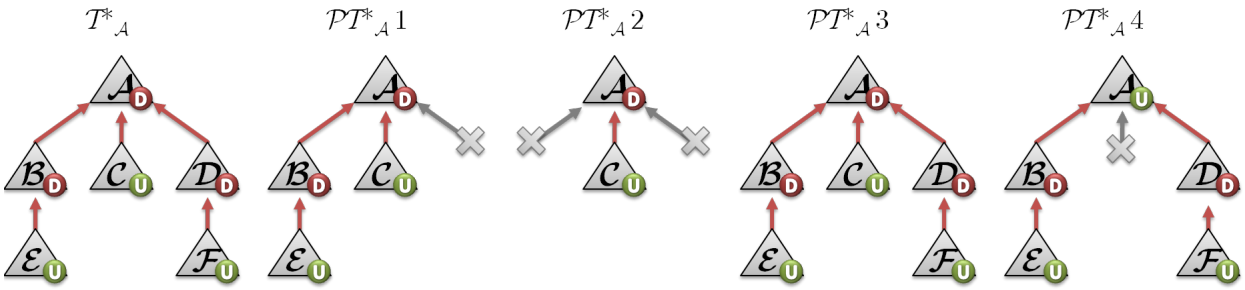
[1]La técnica de poda busca mantener los árboles lo más chicos posibles pero aun así mantener las propiedades del árbol completo, como un bonsai.

7.2.1. Podas de Árboles de Dialéctica

En esta sección se presentarán los conceptos relacionados a las podas en los árboles de dialéctica. La construcción del bonsai de dialéctica depende de que los árboles de dialéctica puedan ser podados. Intuitivamente, una poda es una versión de un árbol de dialéctica donde se "cortan" ciertos argumentos. Como se vio en el Capítulo 5, un árbol de dialéctica en T-DeLP es un árbol en que los nodos son argumentos y los arcos denotan derrotas. Entonces, una poda para un árbol de dialéctica será un árbol que contiene un subconjunto de los argumentos y las derrotas del árbol original. Esta intuición se encuentra formalizada en la siguiente definición.

Definición 7.21 (Poda) *Sea $\mathcal{T}^*_{\mathcal{A}}$ un árbol de dialéctica de un programa T-DeLP $\mathcal{P}(C, Ag)$ asociado a un agente* ArgAPL *Ag en una configuración C, Args el conjunto de argumentos de $\mathcal{P}(C, Ag)$ utilizados en $\mathcal{T}^*_{\mathcal{A}}$, y Derrs el conjunto de derrotas utilizadas en $\mathcal{T}^*_{\mathcal{A}}$. Una poda $\mathcal{PT}^*_{\mathcal{A}}$ para $\mathcal{T}^*_{\mathcal{A}}$ es un árbol con raíz $\mathcal{A}$ tal que contiene un conjunto de argumentos $\mathsf{S}_A \subseteq \mathsf{Args}$ y un conjunto de derrotas $\mathsf{S}_D \subseteq \mathsf{Derrs}$.*

Ejemplo 7.5 *Considere el árbol de dialéctica marcado $\mathcal{T}^*_{\mathcal{A}}$ ilustrado debajo. A su derecha se incluyen cuatro posibles podas $\mathcal{PT}^*_{\mathcal{A}}i$ $(1 \leq i \leq 4)$ para $\mathcal{T}^*_{\mathcal{A}}$.*



La definición anterior brinda una noción general del concepto de poda. Sin embargo, no cualquier poda de un árbol clasificará como un bonsai de dialéctica. Dado un árbol de dialéctica $\mathcal{T}$, un bonsai para $\mathcal{T}$ será una versión podada de él tal que el marcado del bonsai mantenga, al menos, el marcado de la raíz de $\mathcal{T}$. Por ejemplo, la poda $\mathcal{PT}^*_{\mathcal{A}}4$ del Ejemplo 7.5 no verifica el criterio asumido. Esto se debe a que se poda $\mathcal{C}$ sin razón aparente, y se llega a un árbol cuya marca para la raíz difiere de la marca en el árbol original. Por lo tanto, este tipo de poda como calificaría como un bonsai.

La intuición del párrafo anterior busca capturar que durante la construcción de los bonsai de dialéctica debe ser posible determinar cuándo se ha obtenido suficiente información para computar la misma marca para la raíz que en el árbol original. Por ejemplo, un escenario simple es aquel donde hay un derrotador **U** para la raíz; cuando este argumento es encontrado, el procedimiento de construcción debería finalizar (no buscar más derrotadores) marcando la raíz como **D**. Una poda que sigue este razonamiento es conocida como 1U, y constituirá la base para los bonsai de dialéctica. Esta poda ha sido utilizada en sistemas argumentativos existentes como DeLP [GS04]. La intuición de la poda 1U es que cuando un derrotador para un argumento $\mathcal{A}$ es marcado como **U**, $\mathcal{A}$ puede ser directamente marcado como **D** sin seguir explorando el resto de sus derrotadores.

Definición 7.22 (Poda 1U) *Sea $\mathcal{PT}^*_{\mathcal{A}}$ una poda para un árbol de dialéctica $\mathcal{T}^*_{\mathcal{A}}$ del programa T-DeLP $\mathcal{P}(C, Ag)$ asociado a un agente* ArgAPL *Ag en la configuración C, y sea $\mathcal{B}$ un nodo interno de $\mathcal{PT}^*_{\mathcal{A}}$ tal que su conjunto de derrotadores en $\mathcal{T}^*_{\mathcal{A}}$ es $\mathsf{DerrT}_{\mathcal{B}}$ y su conjunto de derrotadores en $\mathcal{PT}^*_{\mathcal{A}}$ es $\mathsf{DerrP}_{\mathcal{B}} \subseteq \mathsf{DerrT}_{\mathcal{B}}$. La poda $\mathcal{PT}^*_{\mathcal{A}}$ será una poda 1U para $\mathcal{T}^*_{\mathcal{A}}$ si en caso de existir algún $\mathcal{C}$ en $\mathsf{DerrT}_{\mathcal{B}}$ marcado como* **U** *en $\mathcal{T}^*_{\mathcal{A}}$, entonces existirá un único argumento $\mathcal{F} \in \mathsf{DerrP}_{\mathcal{B}}$ marcado como* **U**.

En otras palabras, la poda 1U es tal que para todo argumento interno su conjunto de derrotadores tiene como máximo un derrotador marcado como **U**. Note que la definición no especifica nada con respecto a los argumentos **D**, ya que el objetivo de la poda es finalizar la exploración de derrotadores al encontrar un derrotador marcado como **U**.

Como se puede observar en el Ejemplo 7.5, sólo serán podas 1U para $\mathcal{T}^*_{\mathcal{A}}$ las podas $\mathcal{PT}^*_{\mathcal{A}}1$, $\mathcal{PT}^*_{\mathcal{A}}2$ y $\mathcal{PT}^*_{\mathcal{A}}3$, mientras que $\mathcal{PT}^*_{\mathcal{A}}4$ no lo será dado que el conjunto de derrotadores para $\mathcal{A}$ en $\mathcal{T}^*_{\mathcal{A}}$ contiene al argumento $\mathcal{C}$ marcado como **U**, pero en el conjunto de derrotadores para $\mathcal{A}$ en $\mathcal{PT}^*_{\mathcal{A}}4$ no hay ningún argumento marcado como **U**. Note que, por lo tanto, pueden existir varias podas 1U para un árbol de dialéctica. Dado que la construcción de los árboles de dialéctica sigue una política de primero en profundidad, la determinación de cuál de estas podas sería la resultante durante la ejecución del agente dependerá puramente del orden en que se elija explorar los derrotadores de un argumento. En el Ejemplo 7.5, se puede ver que en $\mathcal{PT}^*_{\mathcal{A}}1$ primero se selecciona $\mathcal{B}$, luego $\mathcal{C}$ (el cual está no derrotado), y por lo tanto $\mathcal{D}$ se poda. En $\mathcal{PT}^*_{\mathcal{A}}2$ el primer argumento seleccionado es $\mathcal{C}$ (que está marcado como **U**), por lo que se podan los subárboles de $\mathcal{B}$ y $\mathcal{D}$. En $\mathcal{PT}^*_{\mathcal{A}}3$ se selecciona $\mathcal{B}$, luego $\mathcal{D}$ y por último $\mathcal{C}$, con lo cual no se producen podas.

Si bien la técnica de poda 1U no resulta muy restrictiva e incluso con un recorrido a ciegas podría ser utilizada frecuentemente dentro de un mismo árbol, encontrar todas las oportunidades de poda sin guía alguna resulta una cuestión de azar, ya que los derrotadores **U** deben ser encontrados en primera instancia, antes que sus hermanos (ver Ejemplo 7.5). En la literatura, la selección de derrotadores durante el proceso de construcción de árboles generalmente no sigue un criterio determinado; esto ocurre no sólo en los enfoques prácticos, sino también en la teoría. En las implementaciones basadas en reglas como DeLP, las reglas se encuentran ubicadas arbitrariamente y los contra-argumentos se construyen recorriéndolas de arriba hacia abajo. Por otra parte, en los enfoques teóricos simplemente se cuenta con un conjunto de argumentos a partir del cual se eligen los contra-argumentos arbitrariamente. En resumen, aún no se ha propuesto una metodología que aumente la capacidad de poda de un sistema argumentativo. Esta información debería ser provista por un mecanismo externo; en nuestro caso, el valor de fuerza de los argumentos. En la siguiente sección se introducirá formalmente esta noción de fuerza argumental.

7.2.2. Fuerza Argumental

La noción de fuerza explorada en esta sección es similar a la propuesta en [MT08, BH01a], y busca capturar la siguiente intuición:

Un argumento es tan fuerte como débiles son sus derrotadores.

De esta manera, la cantidad de derrotadores no es el único parámetro que afecta la fuerza de un argumento, sino que este enfoque también tiene en cuenta la estructura del árbol de dialéctica. Por ejemplo, dado un argumento $\mathcal{A}$ de soporte, su respectivo subárbol conteniendo varios argumentos de soporte debería darle a $\mathcal{A}$ un valor alto de fuerza. Por lo tanto, ese valor de fuerza concentra la fuerza de todo el subárbol. Es decir, el valor de fuerza de cada argumento en el árbol actúa como un representante del subárbol debajo de ella. En consecuencia, la idea detrás de esta medida de fuerza es establecer cuan probablemente un argumento estará finalmente derrotado. Por lo tanto, como se verá más adelante, los valores de fuerza de los argumentos serán utilizados como heurística para guiar la construcción de los árboles de dialéctica, buscando maximizar las posibilidades de poda.

A continuación se propone una fórmula para calcular la fuerza argumental, la cual se basa en los conceptos presentados en [BH01a]. La fuerza de un argumento $\mathcal{A}$ se obtiene a partir de el árbol de dialéctica $\mathcal{T}^*_{\mathcal{A}}$. Para calcular la fuerza de $\mathcal{A}$ se calcula la fuerza de los derrotadores de $\mathcal{A}$ en $\mathcal{T}^*_{\mathcal{A}}$, lo que lleva a calcular la fuerza de los derrotadores de los derrotadores, y así sucesivamente hasta considerar todo el árbol de dialéctica.

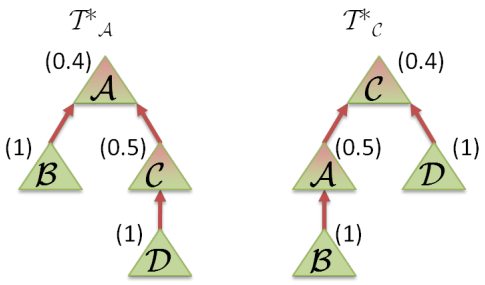
Definición 7.23 (Fuerza Argumental) *La fuerza argumental de un argumento $\mathcal{B}$ en una línea argumentativa aceptable exhaustiva λ de un árbol de dialéctica $\mathcal{T}^*_{\mathcal{A}}$ se calcula como:*

$$\mu(\mathcal{B}, \lambda, \mathcal{T}^*_{\mathcal{A}}) = \frac{1}{1 + \sum_i (\mu(\mathcal{C}_i, \lambda_i, \mathcal{T}^*_{\mathcal{A}}))}$$

donde $\mathcal{C}_i$ es un hijo de $\mathcal{B}$ en la línea λ_i dentro del árbol $\mathcal{T}^*_{\mathcal{A}}$.

Dado que un argumento puede aparecer en diferentes líneas dentro de un mismo árbol, debe individualizarse a través de tres parámetros para calcular su fuerza. La fuerza para un argumento $\mathcal{A}$ se nota como $\mu(\mathcal{A}, \cdot, \mathcal{T}^*_{\mathcal{A}})$, se abrevia como $\mu(\mathcal{A})$, y denotará lo que se denomina como *fuerza real* del argumento. Esto se debe a que un argumento no raíz en un árbol de dialéctica, si bien tiene una medida de fuerza de acuerdo a la Definición 7.23, esta fuerza será *local* a ese árbol de dialéctica. Es decir, no será la fuerza real que tendría si se calculase su fuerza en el árbol enraizado en él, sino una medida parcial utilizada para calcular la fuerza del argumento raíz del árbol en que aparece. Más aún, un argumento que aparece en dos líneas de un árbol de dialéctica podría llegar a tener dos medidas de fuerza local diferentes. Por lo tanto, como se verá en la próxima sección, la medida de fuerza argumental para un argumento será su fuerza real.

Ejemplo 7.6 *Considere los árboles de dialéctica que se ilustran debajo:*



A continuación se procederá a mostrar cómo se calcula la fuerza real $\mu(\mathcal{A})$ de $\mathcal{A}$. Como los argumentos $\mathcal{B}$ y $\mathcal{D}$ no tienen derrotadores, la suma de la fuerza de sus derrotadores es 0 y, por lo tanto, $\mu(\mathcal{B}, \lambda_1, \mathcal{T}^*_{\mathcal{A}}) = \mu(\mathcal{D}, \lambda_2, \mathcal{T}^*_{\mathcal{A}}) = 1 \backslash (1 + 0) = 1$. El argumento $\mathcal{C}$ sólo está derrotado por $\mathcal{D}$, con lo cual $\mu(\mathcal{C}, \lambda_2, \mathcal{T}^*_{\mathcal{A}}) = 1 \backslash (1 + 1) = 0{,}5$. Luego, dado que $\mathcal{A}$ está derrotado por $\mathcal{B}$ y $\mathcal{C}$, su fuerza será $\mu(\mathcal{A}, \cdot, \mathcal{T}^*_{\mathcal{A}}) = 1/(1 + 1 + 0{,}5) = 0{,}4$. De esta manera, la fuerza real de $\mathcal{A}$ será $\mu(\mathcal{A}) = 0{,}4$.

Efectuando un análisis similar se puede observar, a partir del árbol de dialéctica $\mathcal{T}^*_{\mathcal{C}}$, que la fuerza real de $\mathcal{C}$ es $\mu(\mathcal{C}) = 0{,}4$. En este caso se puede ver claramente cómo la fuerza real de un argumento difiere de la fuerza local de ese argumento en el árbol de dialéctica para otro argumento ($\mu(\mathcal{C}, \lambda_2, \mathcal{T}^*_{\mathcal{A}}) = 0{,}5$).

La medida de fuerza real para los argumentos de un agente ArgAPL será utilizada para guiar al proceso de construcción de los árboles de dialéctica al momento de determinar sus creencias y metas actuales en una configuración. Claramente, los argumentos que el agente podrá construir variarán de una configuración a otra ya que, por ejemplo, podría cambiar su conjunto de percepciones. Por lo tanto, es muy importante realizar la siguiente observación.

Observación 7.1 *Cabe destacar que la fuerza real para cada argumento debe calcularse teniendo en cuenta todos los posibles argumentos con los que razonará el agente. Además, para que la técnica sea efectiva, la fuerza debe calcularse de forma previa a la ejecución del agente. De esta manera, cuando el agente construya los árboles de dialéctica para los argumentos que se pueden obtener a partir de su configuración actual, podrá utilizar la medida de fuerza previamente calculada para así guiar las podas. En la Sección 7.2.4 se mostrará cómo, a partir de la especificación de un agente* ArgAPL*, es posible construir todos los posibles argumentos con los que razonará en su ejecución para así calcular su fuerza.*

7.2.3. Podas Heurísticas

En esta sección se formalizará la noción de bonsai de dialéctica y, seguidamente, se presentarán los *fast-prune bonsai*. Estos últimos serán bonsai en los que se utiliza una poda guiada por heurística. El objetivo del uso de heurísticas es obtener podas más similares a una poda óptima. Es decir, si los valores heurísticos indican con un buen grado de

certeza qué argumentos del árbol terminarán no derrotadas, la poda heurística tenderá a encontrar los argumentos **U** primero, cortando el resto de los derrotadores pendientes de análisis.

A continuación se definirán los bonsai de dialéctica como un caso particular de poda, cuyo único requisito es que el estado de derrota de sus argumentos sea igual al que tenían en árbol original.

Definición 7.24 (Bonsai de dialéctica) *Sea $\mathcal{T}^*_{\mathcal{A}}$ un árbol de dialéctica del programa T-DeLP $\mathcal{P}(Ag, C)$ asociado a un agente* ArgAPL *Ag en la configuración C, un bonsai de dialéctica $\mathcal{BT}^*_{\mathcal{A}}$ para $\mathcal{T}^*_{\mathcal{A}}$ es una poda de $\mathcal{T}^*_{\mathcal{A}}$ tal que todo argumento de $\mathcal{BT}^*_{\mathcal{A}}$ tiene el mismo marcado que en $\mathcal{T}^*_{\mathcal{A}}$.*

Proposición 7.11 *Sea $\mathcal{T}^*_{\mathcal{A}}$ un árbol de dialéctica del programa T-DeLP $\mathcal{P}(Ag, C)$ asociado a un agente* ArgAPL *Ag en la configuración C. Una poda 1U $\mathcal{PT}^*_{\mathcal{A}}$ para $\mathcal{T}^*_{\mathcal{A}}$ es un bonsai de dialéctica.*

Prueba : Sea $\mathcal{PT}^*_{\mathcal{A}}$ una poda 1U para $\mathcal{T}^*_{\mathcal{A}}$. Todo argumento $\mathcal{C}$ con un conjunto de derrotadores D_C en $\mathcal{T}^*_{\mathcal{A}}$ tal que al menos un argumento de D_C está marcado como **U**, *estará marcado como* **D** *en $\mathcal{T}^*_{\mathcal{A}}$. Luego, por Definición 7.22 de poda 1U, si $\mathcal{C}$ está en $\mathcal{PT}^*_{\mathcal{A}}$ entonces tendrá a lo sumo un derrotador marcado como* **U**, *con lo cual $\mathcal{C}$ también estará marcado como* **D** *en $\mathcal{PT}^*_{\mathcal{A}}$. Por otra parte, si el conjunto de derrotadores de $\mathcal{C}$ en $\mathcal{T}^*_{\mathcal{A}}$ es vacío o todos sus argumentos están marcados como* **D**, *entonces $\mathcal{C}$ estará marcado como* **U** *en $\mathcal{T}^*_{\mathcal{A}}$. Por lo tanto, por Definición 7.22 y dado que se mostró que un argumento marcado como* **D** *en $\mathcal{T}^*_{\mathcal{A}}$ no puede cambiar su marcado a* **U** *en $\mathcal{PT}^*_{\mathcal{A}}$, el argumento $\mathcal{C}$ también aparece como* **U** *en $\mathcal{PT}^*_{\mathcal{A}}$.* □

Por Definición 7.24 un bonsai de dialéctica no es una poda arbitraria del árbol de dialéctica asociado, sino una que mantiene la misma marca para todo argumento común a ambos; en particular, la marca de la raíz. La propiedad más importante que un bonsai de dialéctica debería satisfacer es garantizar exactamente los mismas argumentos que el árbol no podado.

Proposición 7.12 *Sea $\mathcal{T}^*_{\mathcal{A}}$ un árbol de dialéctica del programa T-DeLP $\mathcal{P}(Ag, C)$ asociado a un agente* ArgAPL *Ag en la configuración C, y $\mathcal{BT}^*_{\mathcal{A}}$ un bonsai de dialéctica para*

$\mathcal{T}^*_{\mathcal{A}}$. El argumento $\mathcal{A}$ está garantizado a partir de $\mathcal{P}(Ag, C)$ si y solo si $\mathcal{A}$ está marcado como **U** *en $\mathcal{BT}^*_{\mathcal{A}}$.*

Prueba : Si $\mathcal{P}(Ag, C)$ garantiza $\mathcal{A}$, entonces por Definición 5.32 de argumento garantizado existe un árbol de dialéctica $\mathcal{T}^*_{\mathcal{A}}$ construido a partir de $\mathcal{P}(Ag, C)$ tal que $\mathcal{A}$ está marcado como **U** *en $\mathcal{T}^*_{\mathcal{A}}$. Luego, por Definición 7.24 de bonsai de dialéctica, se tiene que la marca de $\mathcal{A}$ en $\mathcal{T}^*_{\mathcal{A}}$ también será* **U**. *Por otra parte, si la marca de $\mathcal{A}$ en $\mathcal{BT}^*_{\mathcal{A}}$ es* **U**, *entonces por Definición 7.24 de bonsai de dialéctica vale que la marca de $\mathcal{A}$ en $\mathcal{T}^*_{\mathcal{A}}$ también es* **U**, *con lo cual $\mathcal{P}(Ag, C)$ garantiza $\mathcal{A}$.* □

Como se mencionó anteriormente, cuando se construye un árbol de dialéctica y se intenta podarlo utilizando una poda 1U, el orden en que se exploran los derrotadores de un nodo interno es muy relevante. En la técnica que se presentará en esta sección cada argumento tendrá un valor asociado de fuerza, el es calculado previamente a la construcción de los árboles de dialéctica por parte del agente para determinar sus metas y creencias actuales. Luego, en lugar de explorar aleatoriamente los derrotadores, se explorarán en orden del más fuerte al mas débil. Para determinar este orden se utilizará la medida fuerza, la cual captura cuan proclive es un argumento a ser derrotado. Es decir, un argumento fuerte (con un valor cercano a 1) en general estará marcado como **U**. Por lo tanto, se explorarán los argumento más fuertes primero, buscando encontrar un argumento marcado como **U** lo más rápido posible, de manera tal que sus hermanos sean podados. Este tipo de poda se denomina *fast-prune bonsai* (en español, bonsai de poda veloz).

Intuitivamente, si un argumento $\mathcal{A}$ en un fast-prune bonsai tiene algún derrotador $\mathcal{C}$ marcado como **U**, implica que $\mathcal{C}$ es el argumento más fuerte que pudo derrotar a $\mathcal{A}$. Por lo tanto, todos los otros derrotadores para $\mathcal{A}$ que estaban en el árbol original y no aparecen en el fast-prune bonsai serán mas débiles que $\mathcal{C}$. Esta intuición se encuentra formalizada en la siguiente definición.

Definición 7.25 (Fast-prune bonsai) *Sea $\mathcal{BT}^*_{\mathcal{A}}$ un bonsai de dialéctica para un árbol de dialéctica $\mathcal{T}^*_{\mathcal{A}}$ del programa T-DeLP $\mathcal{P}(Ag, C)$ asociado a un agente* ArgAPL *Ag en la configuración C, y $\mu()$ la fuerza real previamente calculada para todo argumento de $\mathcal{P}(Ag, C)$. $\mathcal{BT}^*_{\mathcal{A}}$ será un fast-prune bonsai para $\mathcal{T}^*_{\mathcal{A}}$ si y solo si todo argumento $\mathcal{F}$ en $\mathcal{BT}^*_{\mathcal{A}}$ con un conjunto de derrotadores DT_F en $\mathcal{T}^*_{\mathcal{A}}$ y un conjunto de derrotadores $DB_F \subseteq DT_F$ en $\mathcal{BT}^*_{\mathcal{A}}$ es tal que, si existe $\mathcal{G}_k \in DB_F$ marcado como* **U** *en $\mathcal{BT}^*_{\mathcal{A}}$, entonces:*

- *no existe otro* $\mathcal{G}_j \in DB_F$ *tal que está marcado como* **U** *en* $\mathcal{BT}^*_{\mathcal{A}}$*, y*
- *para todo* $\mathcal{G}_i \in DT_F$ *tal que no está en* BD_F *vale que* $\mu(\mathcal{G}_k) \geq \mu(\mathcal{G}_i)$*.*

Otra forma de interpretar esta definición es considerar a los derrotadores $\mathcal{G}_j$ más fuertes para $\mathcal{F}$ como aquellos argumentos que serán explorados hasta que se encuentre un derrotador no derrotado $\mathcal{G}_k$. Luego, todos los derrotadores $\mathcal{G}_i$ que quedan fuera del fast-prune bonsai serán más débiles que los derrotadores que forman parte del bonsai. Como se puede observar en la Definición 7.25, no se considera la situación en que todos los derrotadores de $\mathcal{F}$ son marcados como **D**. Esto se debe a que cualquier subconjunto de derrotadores para $\mathcal{F}$ en el fast-prune bonsai preservará la marca para $\mathcal{F}$. Sin embargo, proceduralmente el algoritmo para construir el fast-prune bonsai explorará todos los argumentos yendo desde los más fuertes hacia los más débiles, para finalmente detectar que ninguno de ellos está marcado como **U**. Por lo tanto, en la práctica, cuando todos los derrotadores de un argumento están marcados como **D**, estos también aparecerán en el bonsai.

Una propiedad que cumplen los fast-prune bonsai es que son podas 1U. A continuación se formaliza este resultado mediante la siguiente proposición.

Proposición 7.13 *Sea* $\mathcal{T}^*_{\mathcal{A}}$ *un árbol de dialéctica del programa T-DeLP* $\mathcal{P}(Ag, C)$ *asociado a un agente* ArgAPL *Ag en la configuración* C*. Un fast-prune bonsai* $\mathcal{BT}^*_{\mathcal{A}}$ *para* $\mathcal{T}^*_{\mathcal{A}}$ *es una poda 1U para* $\mathcal{T}^*_{\mathcal{A}}$*.*

Prueba : *El fast-prune bonsai requiere que para cualquier conjunto de derrotadores de un argumento* $\mathcal{C}$ *tal que al menos uno de ellos está marcado como* **U** *en el árbol original: (1) en el bonsai aparece un único derrotador* $\mathcal{G}_k$ *marcado como* **U** *de ese conjunto; y (2)* $\mathcal{G}_k$ *es más débil que el resto de los derrotadores de* $\mathcal{C}$ *que aparecen en el bonsai. Dado que la primer condición coincide con el requerimiento para que una poda sea 1U, se tiene que* $\mathcal{BT}^*_{\mathcal{A}}$ *es una poda 1U para* $\mathcal{T}^*_{\mathcal{A}}$*.* □

Ejemplo 7.7 *Considere el árbol de dialéctica* $\mathcal{T}^*_{\mathcal{A}}$ *para el argumento* $\mathcal{A}$ *con los valores de fuerza previamente calculados, el cual surge de una configuración de un agente* ArgAPL *y es ilustrado en la Figura 7.8(a). En la Figura 7.8(b) puede observarse el respectivo fast-prune bonsai, el cual se obtiene al elegir primero los derrotadores más fuertes:* $\mathcal{A}_2$ *antes que* $\mathcal{A}_1$*, luego* $\mathcal{A}_4$ *antes que* $\mathcal{A}_5$*. En este punto se elige primero* $\mathcal{A}_7$*, el cual es una hoja,*

por lo que se poda a $\mathcal{A}_6$*. Luego, como* $\mathcal{A}_4$ *está derrotado, se explora* $\mathcal{A}_5$ *y se elige a* $\mathcal{A}_8$ *antes que a* $\mathcal{A}_9$*. Como* $\mathcal{A}_8$ *está no derrotado por ser hoja, se poda a* $\mathcal{A}_9$ *y se marca a* $\mathcal{A}_5$ *como derrotado. Esto conduce a marcar a* $\mathcal{A}_2$ *como no derrotado, con lo cual se poda a* $\mathcal{A}_1$*. Finalmente se marca a la raíz* $\mathcal{A}$ *como derrotada.*

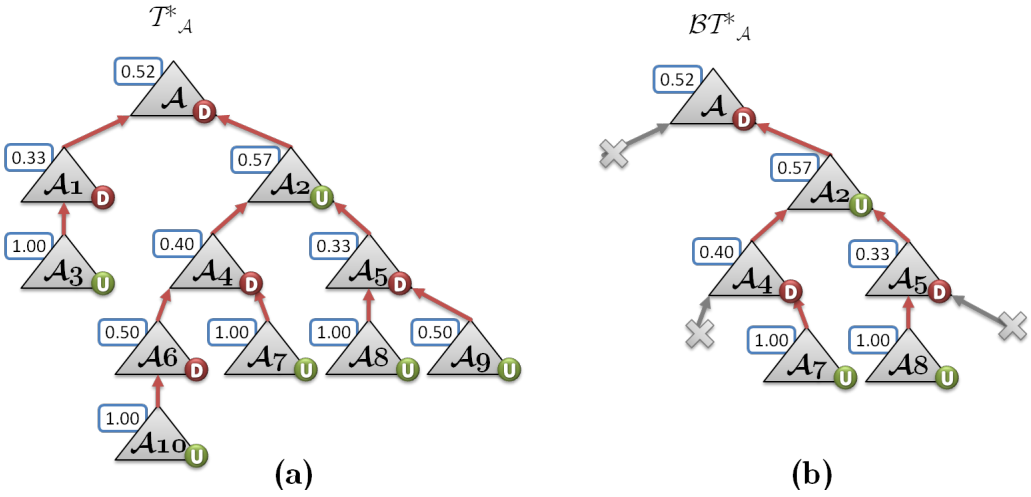


Figura 7.8: Árbol de dialéctica para $\mathcal{A}$ en una configuración y su fast-prune bonsai asociado.

Observe que los valores de fuerza asociados a algunos argumentos en el árbol de dialéctica $\mathcal{T}^*_{\mathcal{A}}$ no coinciden con los valores que se obtendrían al aplicar la función $\mu()$ sobre $\mathcal{T}^*_{\mathcal{A}}$. Por ejemplo, esto ocurre en el caso de $\mathcal{A}_1$ y $\mathcal{A}_1$ Esta diferencia se debe a que, como se explicó anteriormente, los valores de fuerza utilizados en $\mathcal{T}^*_{\mathcal{A}}$ son calculados previamente y corresponden a la fuerza real de los argumentos del árbol. En la siguiente sección se mostrará cómo un agente ArgAPL puede calcular la fuerza para cada uno de sus argumentos antes de comenzar su ejecución.

7.2.4. Argumentos Posibles y Árboles Potenciales en ArgAPL

En esta sección se presentará un método para calcular la fuerza argumental de todos los argumentos de un agente ArgAPL directamente a partir de su especificación. Es decir, la fuerza para cada posible argumento que el agente pueda construir durante su ejecución se calculará antes de comenzar a ejecutar. Para esto se identificarán todos los argumentos que el agente podría llegar a construir durante su ejecución. Utilizando estos argumentos

posibles se podrán construir los árboles de dialéctica, los cuales serán conocidos como árboles potenciales. Por lo tanto, se aplicará la fórmula para calcular la fuerza de cada argumento posible. La fuerza asignada a cada argumento posible será luego utilizada durante la ejecución para realizar las podas heurísticas. Esta técnica está parcialmente inspirada en la aproximación para la construcción de argumentos potenciales presentada en [CCS05].

Un argumento posible es un argumento que el agente puede construir durante su ejecución. Es decir, corresponde a algún argumento que el agente generará en una configuración. Como se puede observar en [RGGS11], para calcular adecuadamente la fuerza de un argumento es necesario considerar todos los argumentos que el agente puede construir. Por lo tanto, será necesario identificar todos los argumentos posibles de un agente ArgAPL.

Claramente, el agente en una configuración podrá construir algunos argumentos y otros no; luego, cuando durante su ejecución pase a otra configuración, dejará de construir algunos y podrá construir otros argumentos que antes no podía. Como se enunció al comienzo de la Sección 7.2, los agentes ArgAPL tratados aquí no cuentan con acciones mentales. Por lo tanto, las bases de creencias y metas se mantendrán iguales durante todas las configuración del agente. De esta manera, la variación que existe entre los argumentos construibles de una configuración a otra depende puramente del conjunto de percepciones.

Por lo tanto, los argumentos posibles se construirán a partir de un programa T-DeLP que se obtiene en forma directa de la especificación del agente ArgAPL en combinación con algún conjunto arbitrario de percepciones para el agente. Este programa T-DeLP será llamado programa T-DeLP potencial ya que no corresponde a una configuración particular del agente, sino al estado previo a su ejecución.

Definición 7.26 (Programa T-DeLP potencial para un agente ArgAPL) *Sea* $Ag = (\Delta^B, \Delta^G, \oslash, S_{ma}, S_{ea}, \mathcal{R})$ *la especificación de un agente* ArgAPL. *El programa T-DeLP potencial para Ag será un programa T-DeLP* $\mathcal{P}(Ag) = (\Gamma, \emptyset, \Delta^B \cup \Delta^G)$, *donde* Γ *es la signatura para el agente* ArgAPL *que se determina a partir de el conjunto de literales en conflicto* $\oslash$.

Note que, si bien el programa T-DeLP potencial para un agente no corresponde directamente al programa que se generará en una configuración, la única diferencia radica

en que el programa potencial tiene un conjunto de percepciones vacío. Es decir, el programa T-DeLP en una configuración y el programa T-DeLP potencial solo diferirán en el conjunto de hechos.

Siguiendo las intuiciones arriba mencionadas, a continuación se definen los argumentos posibles. Básicamente, un argumento posible es un argumento representativo que se puede construir a partir del programa potencial con algún conjunto de percepciones que el agente pueda llegar a tener.

Definición 7.27 (Argumento posible ArgAPL) *Sea Ag un agente* ArgAPL *y $\mathcal{P}(Ag) = (\Gamma, \emptyset, \Delta^B \cup \Delta^G)$ su programa T-DeLP potencial. Si existe un conjunto de percepciones Θ^P tal que $\langle \mathcal{A}, L^X \rangle$ con $X \in \{P, B, AG, MG\}$ es un argumento representativo de $(\Gamma, \Theta^P, \Delta^B \cup \Delta^G)$, entonces $\langle \mathcal{A}, L^X \rangle$ es un argumento posible para Ag.*

Note que el hecho de exigir que exista algún conjunto de percepciones permite que se pueda construir cualquier argumento posible, inclusive argumentos que tal vez nunca lleguen a coexistir en una configuración.

Es importante remarcar que los conjuntos de percepciones para un agente dependen puramente del dominio en que se desenvuelva. Básicamente, el dominio define qué literales pueden llegar a aparecer en conjunto como una percepción. Por ejemplo, si se considera el dominio presentado en la Sección 6.1.1 del Capítulo 6 para el agente que recorre las islas en busca de tesoros, un conjunto de posibles percepciones podría contener literales tipados como $en(i1, yo)^P$, $en(i2, ag2)^P$, $tiene(ag2, tesoro(t1))^P$ o $en(i1, tesoro(t2))^P$, mientras que literales como $promisoria(i1)^P$ o $peligrosa(i2)^P$ no serán parte de un conjunto de percepciones. No obstante, note que el conjunto de todos los literales que pueden aparecer en la percepción del agente podría llegar a ser inconsistente. Por ejemplo, considerando el dominio arriba mencionado, los literales $en(i1, yo)^P$ y $en(i2, yo)^P$ están en desacuerdo según los ejemplos del capítulo anterior. Por lo tanto, en lugar de utilizar todos los posibles literales, la Definición 7.27 considera conjuntos de percepciones posibles para construir los argumentos posibles, los cuales, como se mostró en el Capítulo 6, son conjuntos consistentes.

Ejemplo 7.8 *Considere el programa T-DeLP potencial $\mathcal{P}(A_8) = (\Gamma_8, \emptyset, \Delta^B{}_8 \cup \Delta^G{}_8)$ para un agente* ArgAPL *A_8, donde:*

$$\Delta^B{}_8 \cup \Delta^G{}_8 = \left\{ \begin{array}{lll} (a^{AG} -\!\prec h^B) & (c^B -\!\prec d^B) & (e^B -\!\prec f^B) \\ (\sim a^{MG} -\!\prec c^B) & (\sim c^B -\!\prec d^B, e^B) & (\sim e^B -\!\prec \sim j^B) \\ (f^B -\!\prec j^B) & (h^B -\!\prec) & \\ (\sim f^B -\!\prec j^B, g^B) & (\sim h^B -\!\prec f^B) & \end{array} \right\}$$

y Γ *es la signatura por defecto para* ArgAPL *(la signatura en la que los desacuerdos sólo ocurren por literales complementarios, ver Capítulo 6). En los conjuntos de percepción de este escenario sólo pueden aparecer los literales* d^P, g^P, j^P, *y* $\sim j^P$. *Por lo tanto, posibles conjuntos de percepciones para* A_8 *son* $\Theta^P{}_{81} = \{d^P, g^P, j^P\}$, $\Theta^P{}_{82} = \{\sim j^P, g^P\}$, *y* $\Theta^P{}_{83} = \{d^P\}$*; mientras que cualquier conjunto que contenga* j^P *y* $\sim j^P$ *no lo será. Por lo tanto, los argumentos posibles para* A_8 *serán:*

$\mathcal{A}_1 = \langle\{(a^{AG} -\!\prec h^B), (h^B -\!\prec)\}, a^{AG}\rangle$
$\mathcal{A}_2 = \langle\{(\sim a^{MG} -\!\prec c^B), (c^B -\!\prec d^B), d^P\}, \sim a^{MG}\rangle$
$\mathcal{A}_3 = \langle\{(c^B -\!\prec d^B), d^P\}, c^B\rangle$
$\mathcal{A}_4 = \langle\{(\sim c^B -\!\prec d^B, e^B), (e^B -\!\prec f^B), (f^B -\!\prec j^B), d^P, j^P\}, \sim c^B\rangle$
$\mathcal{A}_5 = \langle\{(e^B -\!\prec f^B), (f^B -\!\prec j^B), j^P\}, e^B\rangle$
$\mathcal{A}_6 = \langle\{(\sim e^B -\!\prec \sim j^B), \sim j^P\}, \sim e^B\rangle$
$\mathcal{A}_7 = \langle\{(f^B -\!\prec j^B), j^P\}, f^B\rangle$
$\mathcal{A}_8 = \langle\{(\sim f^B -\!\prec j^B, g^B), j^P, g^P\}, \sim f^B\rangle$
$\mathcal{A}_9 = \langle\{(h^B -\!\prec)\}, h^B\rangle$
$\mathcal{A}_{10} = \langle\{(\sim h^B -\!\prec f^B), (f^B -\!\prec j^B), j^P\}, \sim h^B\rangle$
$\mathcal{A}_{11} = \langle\{d^P\}, d^P\rangle$
$\mathcal{A}_{12} = \langle\{g^P\}, g^P\rangle$
$\mathcal{A}_{13} = \langle\{j^P\}, j^P\rangle$
$\mathcal{A}_{14} = \langle\{\sim j^P\}, \sim j^P\rangle$

Note que, como establece la Definición 7.27, los argumentos posibles se construyen considerando cualquier posible conjunto de percepciones. Por lo tanto, es posible construir a la vez los argumentos posibles $\mathcal{A}_5$ *y* $\mathcal{A}_6$ *que se basan en* j^P *y* $\sim j^P$. *Sin embargo, durante su ejecución el agente no podrá construir estos argumentos a la vez ya que se originan de percepciones conflictivas, y el conjunto de percepciones del agente siempre será libre de conflictos. No obstante, para aplicar la técnica del fast-prune bonsai es importante que la fuerza real se calcule considerando todos los argumento posibles, ya que esta describe cuan probable es que un argumento esté no derrotado.*

Es importante destacar que para calcular la fuerza de los argumentos posibles no tiene sentido considerar los argumentos posibles estrictos. Como se vio en el Capítulo 5, los argumentos estrictos no pueden ser atacados y tampoco derrotarán a ningún otro argumento, ya que no es posible construir argumentos que estén en conflicto con un argumento estricto. Por lo tanto, los argumento estrictos siempre tendrán fuerza máxima y no influenciarán en la fuerza de ningún otro argumento no estricto. En consecuencia, de aquí en adelante sólo se considerarán los *argumentos posibles no estrictos.*

Para calcular la fuerza de los argumentos potenciales será necesario construir los árboles de dialéctica enraizados en ellos. Para tal fin, en primer lugar será necesario determinar

cómo son los ataques entre los argumentos posibles. Los ataques entre argumentos posibles son similares a los ataques entre argumentos representativos en T-DeLP: un argumento posible $\mathcal{A}$ atacará a otro argumento posible $\mathcal{B}$ si $\mathcal{B}$ tiene un subargumento $\mathcal{C}$ cuya conclusión está en desacuerdo con la conclusión de $\mathcal{A}$. La única diferencia entre este ataque y el ataque entre argumentos representativos de T-DeLP es que el subargumento $\mathcal{C}$ debe ser no estricto. Esto se debe a que, al considerar cualquier percepción, se pueden construir argumentos posibles que estén en desacuerdo con algún argumento estricto. Por lo tanto, si no se controla esto se modelaría un ataque que no existirá nunca, ya que si en ejecución existiese el argumento estricto entonces no existirá el argumento no estricto en desacuerdo con él.

Ejemplo 7.9 *Considere el programa T-DeLP potencial* $\mathcal{P}(A_9) = (\Gamma_9, \emptyset, {\Delta^B}_9 \cup {\Delta^G}_9)$ *para un agente* ArgAPL A_9*, donde los posibles conjuntos de percepciones pueden contener* x^P *o* $\sim x^P$*, y:*

$$ {\Delta^B}_9 \cup {\Delta^G}_9 = \left\{ \ (\sim x^B \prec) \quad (w^B \prec x^B) \ \right\} $$

A partir de este programa se pueden construir los argumentos posibles no estrictos:

$$ \mathcal{B}_1 = \langle \{(\sim x^B \prec)\}, \sim x^B \rangle \quad y \quad \mathcal{B}_2 = \langle \{(w^B \prec x^B), x^P\}, w^B \rangle $$

Claramente, si se utilizara la noción de ataque de T-DeLP en este escenario, $\mathcal{B}_1$ *atacaría* $\mathcal{B}_2$ *en* x^P*. Sin embargo, esto no es adecuado ya que si en ejecución se puede construir* $\mathcal{B}_2$ *es por* x^P*, con lo cual* $\mathcal{B}_1$ *no podría ser construido. Por lo tanto, cuando se consideran los argumentos posibles en una situación como esta, el argumento* $\mathcal{B}_1$ *no debería atacar al argumento* $\mathcal{B}_2$*.*

Definición 7.28 (Ataque entre argumentos posibles) *Sean* $\langle \mathcal{A}, L^T \rangle$ *y* $\langle \mathcal{B}, M^U \rangle$ *argumentos posibles no estrictos de un programa T-DeLP potencial* $\mathcal{P}(Ag) = (\Gamma, \emptyset, \Delta^B \cup \Delta^G)$ *para el agente* ArgAPL *Ag. El argumento posible* $\langle \mathcal{A}, L^T \rangle$ *ataca al argumento posible* $\langle \mathcal{B}, M^U \rangle$ *si y solo si existe un subargumento representativo no estricto* $\langle \mathcal{C}, N^V \rangle$ *de* $\langle \mathcal{B}, M^U \rangle$ *tal que* $L^T \otimes N^V$*.*

A diferencia de los ataques, las nociones de preferencia, derrota y línea argumentativa aceptable exhaustiva para los argumentos posibles son exactamente las mismas que para los argumentos representativos presentados en el Capítulo 5, considerando cómo son los

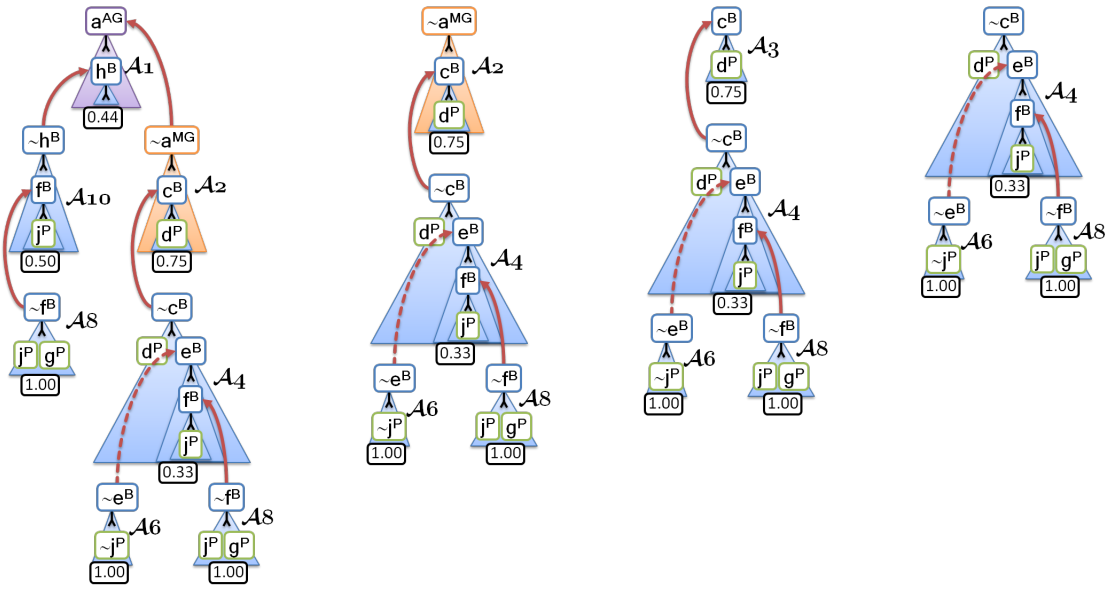
programas de los agentes ArgAPL (*i.e.*, los tipos que caracterizan y sus relaciones). Por este motivo, no se presentarán definiciones para estas nociones.

Similar a las nociones presentadas en el párrafo anterior es el caso de los árboles de dialéctica. No obstante, se presentará una definición para los árboles de dialéctica basados en argumentos posibles, los cuales serán identificados como árboles de dialéctica potenciales.

Definición 7.29 (Árbol de dialéctica potencial) *Sea $\mathcal{A}$ un argumento posible no estricto de un programa T-DeLP potencial $\mathcal{P}(Ag)$ para el agente* ArgAPL *Ag. El árbol de dialéctica potencial $\mathcal{T}^*_{\mathcal{A}}$ para $\mathcal{A}$ es un árbol de dialéctica para $\mathcal{A}$ en el que sólo se consideran argumentos posibles no estrictos de $\mathcal{P}(Ag)$.*

Como se mencionó anteriormente, los árboles potenciales serán utilizados para calcular la fuerza de los argumentos posibles de un agente ArgAPL. A continuación se mostrará cómo se calcula la fuerza para los argumentos posibles del Ejemplo 7.8.

Ejemplo 7.10 *Considere los argumentos posibles no estrictos presentados en el Ejemplo 7.8 para el agente A_8. La Figura 7.9 ilustra los árboles de dialéctica potenciales para cada uno de ellos, donde las flechas enteras denotan derrotas propias y las flechas punteadas derrotas por bloqueo. Además, en la Figura 7.9 se indica la fuerza de cada uno de estos argumentos a través del árbol.*



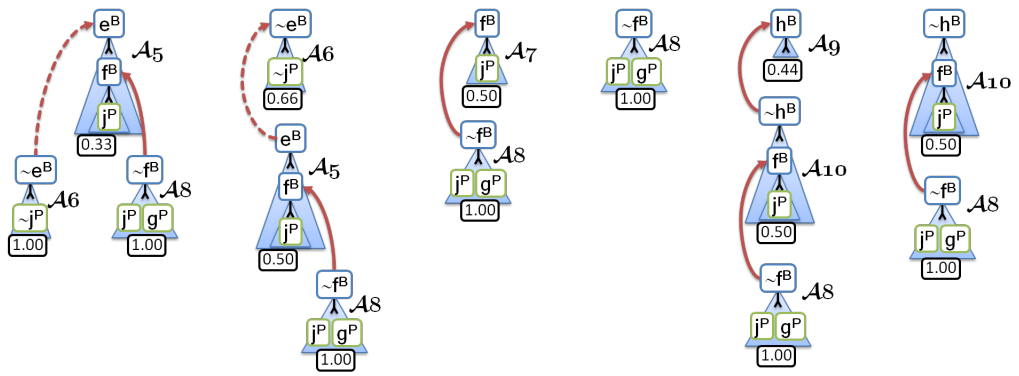


Figura 7.9: Árboles potenciales y cálculo de fuerza para los argumentos posibles del agente ArgAPL A_8.

Por lo tanto, como puede observarse en la Figura 7.9, la fuerza de cada argumento será:

$$\mu(\mathcal{A}_1) = 0.44 \quad \mu(\mathcal{A}_2) = 0.75 \quad \mu(\mathcal{A}_3) = 0.75 \quad \mu(\mathcal{A}_4) = 0.33 \quad \mu(\mathcal{A}_5) = 0.33$$
$$\mu(\mathcal{A}_6) = 0.66 \quad \mu(\mathcal{A}_7) = 0.50 \quad \mu(\mathcal{A}_8) = 1.00 \quad \mu(\mathcal{A}_9) = 0.44 \quad \mu(\mathcal{A}_10) = 0.5$$

Estos árboles pueden construirse previamente a la ejecución del agente y consideran todos los argumentos que pueden llegar a aparecer en la configuración de un agente. Por lo tanto, la medida de fuerza calculada sobre estos árboles cumple todos los requisitos para que el fast-prune bonsai funcione de manera efectiva.

Ejemplo 7.11 *Considere que el agente A_8 del Ejemplo 7.8 se encuentra en la configuración C_8 tal que el conjunto de percepciones es $\Theta^P{}_8 = \{d, j, g\}$, y suponga que desea averiguar si a^{AG} es una meta de logro actual. En la Figura 7.10 se muestra cómo será el bonsai que construirá A_8 para a^{AG} en la configuración C_8 valiéndose de la medida de fuerza calculada para cada argumento en el Ejemplo 7.10.*

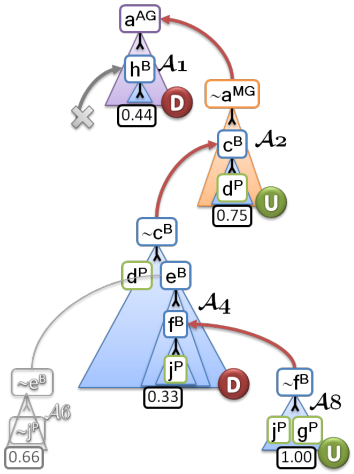


Figura 7.10: fast-prune bonsai para a^{AG} en la configuración C_8 del agente ArgAPL A_8.

Al considerar los derrotadores de $\mathcal{A}_1$ observe que se elige a $\mathcal{A}_2$ antes que a $\mathcal{A}_{10}$, dado que tiene mayor fuerza. Luego se elige a $\mathcal{A}_8$, el cual está marcado como **U**. *En particular note que, si bien $\mathcal{A}_8$ es el derrotador de $\mathcal{A}_4$ con mayor fuerza, es el único que se encuentra activo con lo cual no se producirán podas en este punto. Por lo tanto, $\mathcal{A}_4$ quedará marcado como* **D** *y $\mathcal{A}_2$ como* **U**. *Finalmente, como $\mathcal{A}_2$ termina marcado* **U**, *se poda $\mathcal{A}_{10}$ y se marca la raíz como* **D**. *En consecuencia, el agente A_8 no tendrá a^{AG} como meta de logro actual en la configuración C_8.*

7.3. Conclusiones

En este capítulo se presentaron dos extensiones para ArgAPL relacionadas a T-DeLP. En la primera se introdujo un conjunto de acciones mentales capaces de contraer un literal de la base de conocimiento de un agente y asegurar que el literal no será inferido por el agente luego de la ejecución de la acción. La segunda extensión corresponde a una técnica de poda heurística para reducir el tamaño de los árboles de dialéctica utilizados para determinar las creencias y metas actuales del agente.

Se mostró cómo las acciones mentales basadas en contracción aseguran que el literal a remover no sea más inferido utilizando los operadores de contracción adaptados de [GGK+11]. En primer lugar se analizaron las acciones mentales para remover la derivación del literal, y para remover los argumentos que garantizan al literal. Estas acciones

resultaron ser muy extremas, ya que las primeras remueven demasiado y las segundas remueven muy poco dando posibilidad a que algún argumento para el literal a remover afecte la garantía de otro literal. En consecuencia, se estudió la noción de influencia, y se presentaron tres acciones mentales basadas en contracciones que siguen esta noción. Cada una de estas tres acciones representa una versión más sofisticada de la anterior y busca remover la influencia del literal quitando la menor cantidad de reglas. Se mostró que las cinco acciones mentales pueden ordenarse con respecto a la cantidad de reglas que remueven del programa, y que las basadas en influencia y la que elimina la derivación aseguran que no habrá influencia del literal removido en el programa.

De esta manera, las acciones mentales de contracción incrementan el poder expresivo de ArgAPL, acercándolo más a los modelos teóricos presentados para los lenguajes de programación de agentes [HdBvdHM99]. Contar con varias acciones mentales de contracción también le brinda un grado de flexibilidad al agente, pudiendo utilizar acciones mentales más rápidas pero más incisivas como la que remueve la derivación, o acciones mentales más complejas pero que preservan la mayor cantidad de reglas posibles.

El método de los bonsai de dialéctica presentado en este capítulo mostró cómo reducir la cantidad de argumentos explorados al determinar creencias y metas actuales del agente, basándose en una técnica de poda guiada por una heurística de exploración de argumentos. La heurística propuesta se basa en la noción de fuerza argumental, la cual representa cuan proclive es un argumento a estar derrotado. Se propuso una fórmula para calcular la fuerza de cada argumento a partir de su árbol de dialéctica. Luego se mostró cómo, teniendo la fuerza previamente calculada, un agente ArgAPL en una configuración puede utilizar el método de fast-prune bonsai para construir árboles explorando primero los argumentos más fuertes, apuntando a maximizar la cantidad de podas. Para estos fast-prune bonsai se mostró cómo es su morfología y qué propiedades cumplen. Dado que la fuerza de cada argumento debe ser conocida de antemano para poder ser utilizada en los fast-prune bonsai, se presentó una forma de calcularla a partir de la especificación de un agente ArgAPL, es decir, previamente a su ejecución. Para esto se identificaron los argumentos posibles y los árboles potenciales sobre los cuales es posible calcular la fuerza de cada argumento.

Por lo tanto, los bonsai de dialéctica proveen una mejora a la eficiencia del cálculo de las inferencias en ArgAPL. En [RGGS11] se realizó un conjunto pruebas empíricas del método, el cual mostró una clara ventaja por sobre las podas sin heurística. Si bien estos

bonsai dependen del cálculo de la fuerza de cada argumento, lo cual puede llegar a ser costoso, en ArgAPL esto corresponde al escenario previo a la ejecución del agente y, por lo tanto, no tiene un impacto negativo en la performance de ejecución del agente. Más aun, el hecho de identificar los argumentos posibles de manera previa a la ejecución del agente permite aprovechar técnicas de pre-compilación de argumentos y derrotas como las presentadas en [CCS05], lo cual aceleraría considerablemente el cálculo de las inferencias de un agente en una configuración.

Capítulo 8

Conclusiones y Trabajo a Futuro

En esta tesis se desarrolló un lenguaje de programación de agentes y dos formalismos de argumentación centrados en la noción de tipo de argumento. Estos formalismos argumentativos extienden a aproximaciones existentes en la literatura de manera tal que permiten representar argumentos tipados y definir relaciones de conflicto, herencia y preferencia entre los tipos de argumento. En particular, el primero de estos formalismos extiende a los marcos argumentativos abstractos, mientras que el segundo constituye una extensión de DeLP que se vale del primero para su definición. El lenguaje de programación de agentes propuesto en esta tesis, llamado ArgAPL, permite especificaciones declarativas y provee una semántica formal. Adicionalmente, a diferencia de otros lenguajes de la literatura, se mostró que ArgAPL permite representar información conflictiva al especificar los componentes mentales de sus agentes, además de modelar diferentes tipos de metas y expresar los conflictos entre estos componentes mentales de manera sofisticada. Para tal fin se mostró que ArgAPL emplea los formalismos argumentativos desarrollados en esta tesis, en particular, utilizando los tipos de argumento para identificar los diferentes componentes mentales de un agente.

En el Capítulo 4 se presentó una formalización para el concepto de tipo de argumento en el contexto de los marcos argumentativos abstractos. Se introdujo un modelo formal en el que los tipos de argumento son representados utilizando marcos argumentativos abstractos con preferencias y se relacionan entre sí a través de ataques, preferencias y herencia entre tipos. Los tipos individuales en conjunto con las relaciones entre tipos componen un *marco argumentativo de tipos múltiples* o MATM. Para establecer qué argumentos son aceptables en un MATM se estudió cómo determinar la relación de derrota en este contex-

to. Para esto fue necesario analizar las dependencias entre los distintos tipos de argumento determinadas por la relación de herencia. En particular, para definir la preferencia entre dos argumentos se analizó la *preferencia interna*, determinada localmente entre argumentos del mismo tipo, y la *preferencia externa*, determinada por la preferencia entre tipos. Combinando estas dos nociones se presentó la noción de *preferencia global* y, consecuentemente, la relación de derrota. Utilizando la relación de derrota y los argumentos de un MATM se mostró cómo construir un marco argumentativo abstracto asociado al MATM. A partir de este marco asociado se mostró cómo aplicar cualquier semántica de aceptabilidad definida para los marcos argumentativos abstractos a los efectos de determinar los argumentos aceptables del MATM. Esto último brindó a los MATMs la posibilidad de heredar todas las propiedades existentes y futuras para los marcos abstractos, los cuales son ampliamente utilizados en la comunidad argumentativa.

En el Capítulo 5 se presentó el sistema argumentativo T-DeLP que extiende a DeLP y, a diferencia de otras propuestas en la literatura, contempla de manera general la noción de tipo de argumento. T-DeLP es un formalismo argumentativo que: provee un lenguaje de representación, utiliza un método de construcción de argumentos tipados, caracteriza los conflictos entre argumentos tipados, identifica derrotas entre esos argumentos y provee métodos para determinar qué argumentos son aceptados. Para esto fue necesario desarrollar un formalismo capaz de construir argumentos tipados a partir de un conjunto de reglas y hechos, identificar cuáles de estos argumentos son representativos y, por último, combinar este formalismo con los resultados obtenidos en el Capítulo 4 para determinar la aceptabilidad de sus argumentos.

Al presentar T-DeLP se mostró que el conocimiento está representado por reglas y hechos tipados, los cuales están constituidos por literales tipados. Los tipos de estos literales fueron utilizados para determinar el tipo de los argumentos. Además, como se mostró en el Capítulo 5, T-DeLP permite representar desacuerdos, preferencias, herencia y propagaciones de estos tipos. Para construir los argumentos en T-DeLP se presentó un método de derivación, más sofisticado que el de DeLP, donde se consideran las relaciones entre los tipos. Para esto se adaptó el concepto de *conformidad*, una noción clásica en la literatura de sistemas de tipos. De este modo, como se mostró en la Proposición 5.3, la derivación de T-DeLP es capaz no sólo de derivar un literal bajo un tipo, sino también bajo todos sus tipos ancestros y los tipos que puede propagar. Luego se mostró que con estas derivaciones se construyen *argumentos tipados*. Dadas las características del mecanismo de derivación,

se mostró que es posible generar varias versiones de un mismo argumento, todas ellas con tipos diferentes relacionados a través herencia (Proposición 5.6). Se mostró que es posible identificar los *argumentos representativos* del sistema, es decir la versión de cada argumento que corresponde al tipo más especializado, los cuales engloban las características de todas las versiones. A partir de la Proposición 5.7 y el Teorema 5.1 se demostró que estos argumentos siempre existen y que son únicos.

Para poder determinar qué argumentos representativos son aceptados en T-DeLP fue necesario especificar las derrotas entre ellos, para lo cual se utilizó el formalismo propuesto para los MATMs. Luego, se presentaron dos alternativas para finalmente determinar la aceptabilidad de estos argumentos: una empleando las semánticas de aceptabilidad presentadas para los MATMs, y otra a través de un procedimiento de prueba dialéctico como el utilizado en DeLP. En particular, a través de la Proposición 5.12, el Lema 5.1 y el Teorema 5.3, se mostró que T-DeLP no garantiza argumentos en conflicto ni literales en desacuerdo.

En el Capítulo 6 se presentó el lenguaje de programación de agentes declarativo ArgAPL. Este lenguaje está basado en 3APL y utiliza a T-DeLP para representar los componentes mentales de los agentes, así como también para razonar argumentativamente sobre las inferencias de cada componente mental en cada estado durante la ejecución. Para presentar ArgAPL se mostró cómo especificar sus agentes y las reglas de semántica formal que describen cómo ejecutarán tales agentes. Además, se mostraron propiedades que ilustran el funcionamiento esperado de los componentes del lenguaje.

En ese capítulo se mostró que las percepciones de un agente ArgAPL son representadas mediante *hechos tipados*, y que las bases de creencias, metas de logro y metas de mantenimiento se representan mediante *reglas rebatibles tipadas*. Dada la naturaleza de T-DeLP, esto permite a ArgAPL realizar *especificaciones declarativas*, y le brinda la posibilidad de *representar información conflictiva* y *metas condicionales* (por ejemplo, metas que dependen de creencias). Un agente ArgAPL construye argumentos para razonar con la información de cada componente mental y, en caso de haber conflictos, decidir cuál prevalece. El concepto de *tipo de argumento* permitió distinguir fácilmente los argumentos de cada componente mental a partir de su especificación, además de caracterizar en cada tipo las propiedades del componente mental asociado. Las relaciones de tipo provistas por T-DeLP permitieron formalizar adecuadamente los conflictos y preferencias existentes entre componentes mentales como, por ejemplo, cuando hay una meta

de logro que ya se cree alcanzada o cuando una meta de logro busca alcanzar un estado que está en contra de una meta de mantenimiento actual. En particular, se mostró que un agente ArgAPL reacciona adecuadamente ante estos conflictos (Teorema 6.4 y Proposición 6.6) y que, a pesar de permitir información conflictiva en sus componentes mentales, sus inferencias en un estado de su ejecución son consistentes (teoremas 6.1, 6.2, y 6.3).

Por otra parte, el mecanismo argumentativo de T-DeLP resultó útil para modelar adecuadamente las metas de mantenimiento en ArgAPL. En el Capítulo 6 se vio que la mayoría de las aproximaciones que tratan estas metas lo hacen sólo de manera reactiva o a través de procesos muy costosos. En ArgAPL se diseñó una estrategia que utiliza las relaciones entre los argumentos tipados y la semántica operacional del lenguaje para brindar una solución proactiva eficiente y dos soluciones reactivas. La primera de ellas permite que el agente evite ejecutar un plan que puede amenazar sus metas de mantenimiento, mientras que las soluciones reactivas son utilizadas para cancelar una acción que violará una meta de mantenimiento o para generar una meta de logro cuando el agente posea una meta de mantenimiento actual que no está siendo mantenida.

Adicionalmente se mostró que en ArgAPL, al permitir metas condicionales y modelar adecuadamente los conflictos entre los distintos tipos de argumento, fue posible modelar de manera directa una política de compromiso de *mundo-abierto* con respecto a las metas del agente. De esta manera, al ejecutar una acción de plan, un agente ArgAPL controlará que la meta asociada siga justificada, es decir, que exista un argumento garantizado para esa meta en ese estado del agente. Como se vio en el Capítulo 6, si no existe tal argumento o está derrotado ya sea por un argumento de creencia (que denota que ya se alcanzó la meta) o por un argumento de meta de mantenimiento (que denota que alcanzar la meta actual pone en riesgo la condición a mantener), el agente deberá cancelar el plan dado que la meta para la cual lo está ejecutando ya no está justificada.

Finalmente, en el Capítulo 7 se presentaron dos extensiones para ArgAPL con respecto a T-DeLP. En la primera de ellas se desarrolló un conjunto de acciones mentales para remover información de las bases de conocimiento de un agente ArgAPL, las cuales se definen a partir de operadores de contracción especialmente definidos para T-DeLP. Estas acciones aseguran que al ser aplicadas la información removida no será inferida en la configuración resultante (proposiciones 7.1, 7.2, 7.5, 7.7 y 7.9). En la segunda extensión se presentó una técnica especializada de poda cuyo objetivo es reducir el tamaño de los árboles de dialéctica construidos para determinar las creencias y metas actuales

de un agente ArgAPL. Esta técnica poda se basa en una heurística, la cual identifica qué argumentos son más propensos a estar no derrotados y es calculada de manera *offline* con respecto a la ejecución del agente. Se mostró que la construcción de los árboles de dialéctica guiados por esta heurística lleva a considerar un menor número de argumentos al computar las creencias y metas actuales, lo cual acelera el razonamiento de los agentes ArgAPL.

8.1. Trabajo a futuro

Como trabajo a futuro se buscará que los agentes ArgAPL puedan intercambiar argumentos con otros agentes de forma tal que puedan entablar un debate. Para esto será necesario determinar cómo los argumentos recibidos afectan al estado mental del agente. En este sentido, el hecho de que los agentes ArgAPL razonen argumentativamente es una ventaja, ya que la problemática de la recepción de un argumento se reduce a analizar cómo el argumento recibido afectará a los argumentos construidos a partir del estado mental del agente. Más aun, el hecho de utilizar T-DeLP permitiría identificar a los argumentos recibidos con tipos especiales o tipos especializados (que heredan) de los tipos del agente, y brindarles la preferencia adecuada con respecto a los tipos de argumento ya existentes. No obstante, es necesario estudiar con mayor profundidad qué tipos serían adecuados para modelar esta situación y cómo serían estas relaciones.

Otro trabajo a futuro que resulta interesante y sigue la línea de los avances en 3APL [Das08], es proveer a los agentes ArgAPL la capacidad de ejecutar múltiples planes para múltiples metas. Para esto la aproximación argumentativa de ArgAPL supone una ventaja, ya que sería necesario que las metas que el agente actualmente está tratando de alcanzar sean consistentes. Por lo tanto, un agente podría adoptar una nueva meta con su respectivo plan siempre y cuando no esté en conflicto con las metas asociadas a los planes actualmente en ejecución. Aun así, será necesario estudiar la semántica operacional asociada a la ejecución de múltiples planes de manera intercalada, identificando qué acciones se podrían realizar en paralelo y qué secuencias de acciones deberían tratarse de manera atómica.

Por otra parte, con respecto a la eficiencia en ArgAPL, resulta interesante considerar la inclusión de técnicas adecuadas de *memoization* al momento de calcular las metas y

creencias actuales de un agente. Es decir, se buscará que el agente en un estado particular pueda memorizar la garantía establecida para un literal y, en caso de necesitarla nuevamente en ese estado, no recalcular los árboles de dialéctica asociados a ese literal. Adicionalmente, esto implica determinar cómo la memorización correspondiente a ese literal tipado influirá en el cálculo de la garantía de otros literales.

Como trabajo a futuro con respecto a T-DeLP se buscará tratar a los tipos como literales en lugar de utilizar identificadores. Esto permitirá modelar situaciones en las que existen diferentes instancias de un mismo tipo. Por ejemplo, podría utilizarse para modelar cada una de las percepciones recibidas por un agente y que, en caso de existir conflictos, el agente prefiera siempre las nuevas. Además, se buscará permitir que se asocien diferentes tipos a un literal, permitiendo así expresar literales tipados de múltiples tipos.

Bibliografía

[ABCM05] Atkinson, K., Bench-Capon, T. J. M., and McBurney, P. Multi-agent argumentation for edemocracy. In *EUMAS* (2005), M. P. Gleizes, G. A. Kaminka, A. Nowé, S. Ossowski, K. Tuyls, and K. Verbeeck, Eds., Koninklijke Vlaamse Academie van Belie voor Wetenschappen en Kunsten, pp. 35–46.

[AC02] Amgoud, L., and Cayrol, C. Inferring from inconsistency in preference-based argumentation frameworks. *Journal Autonomous Reasoning 29*, 2 (2002), 125–169.

[ACLSL08] Amgoud, L., Cayrol, C., Lagasquie-Schiex, M.-C., and Livet, P. On bipolarity in argumentation frameworks. *Int. J. Intell. Syst. 23*, 10 (2008), 1062–1093.

[ADL08] Amgoud, L., Devred, C., and Lagasquie, M. A constrained argumentation system for practical reasoning. In *Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS'08)* (2008), pp. 429–436.

[AGH00] Arnold, K., Gosling, J., and Holmes, D. *The Java Programming Language, Third Edition*. Addison-Wesley, 2000.

[AGM85] Alchourrón, C. E., Gärdenfors, P., and Makinson, D. On the logic of theory change: Partial meet contraction and revision functions. *J. Symb. Log. 50*, 2 (1985), 510–530.

[AK07] Amgoud, L., and Kaci, S. An argumentation framework for merging conflicting knowledge bases. *Int. J. Approx. Reasoning 45*, 2 (2007), 321–340.

[AP05] Amgoud, L., and Prade, H. Handling threats, rewards, and explanatory arguments in a unified setting. *International Journal of Intelligent Systems 20*, 12 (2005), 1195–1218.

[APP00] Amgoud, L., Parsons, S., and Perrussel, L. An argumentation framework based on contextual preferences. In *In International Conference on Formal and Applied and Practical Reasoning (FAPR'2000)* (2000), pp. 59–67.

[BBD+06] Bordini, R., Braubach, L., Dastani, M., Seghrouchni, A. E. F., Gomez-Sanz, J., Leite, J., O'Hare, G., Pokahr, A., and Ricci, A. A survey of programming languages and platforms for multi-agent systems. In *Informatica 30* (2006), pp. 33–44.

[BC03] Bench-Capon, T. J. M. Persuasion in practical argument using value-based argumentation frameworks. *Journal of Logic and Computation 13*, 3 (2003), 429–448.

[BCD07] Bench-Capon, T. J. M., and Dunne, P. E. Argumentation in artificial intelligence. *Artif. Intell. 171*, 10-15 (2007), 619–641.

[BDD+10] Behrens, T. M., Dastani, M., Dix, J., Köster, M., and Novák, P. The multi-agent programming contest from 2005-2010 - from gold collecting to herding cows. *Ann. Math. Artif. Intell. 59*, 3-4 (2010), 277–311.

[BDDFS05] Bordini, R. H., Dastani, M., Dix, J., and Fallah-Seghrouchni, A. E., Eds. *Multi-Agent Programming: Languages, Platforms and Applications*, vol. 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Springer, 2005.

[BDH08] Besnard, P., Doutre, S., and Hunter, A., Eds. *Computational Models of Argument: Proceedings of COMMA 2008, Toulouse, France, May 28-30, 2008* (2008), vol. 172 of *Frontiers in Artificial Intelligence and Applications*, IOS Press.

[BFKIT10] Beierle, C., Freund, B., Kern-Isberner, G., and Thimm, M. Using defeasible logic programming for argumentation-based decision

support in private law. In *Third International Conference on Computational Models of Argument (COMMA'10)* (Desenzano del Garda, Italy, September 2010).

[BG09] Baroni, P., and Giacomin, M. Semantics of abstract argumentation systems plans. In *Argumentation in Artificial Intelligence*, I. Rahwan and G. Simari, Eds. Springer, 2009, pp. 25–44.

[BGG05] Baroni, P., Giacomin, M., and Guida, G. Scc-recursiveness: a general schema for argumentation semantics. *Artif. Intell. 168*, 1-2 (2005), 162–210.

[BH01a] Besnard, P., and Hunter, A. A logic-based theory of deductive arguments. *Artif. Intell. 128*, 1-2 (2001), 203–235.

[BH01b] Besnard, P., and Hunter, A. A logic-based theory of deductive arguments. *Artificial Intelligence 128*, 1-2 (2001), 203–235.

[BH09] Black, E., and Hunter, A. An inquiry dialogue system. *Autonomous Agents and Multi-Agent Systems (JAAMAS 19*, 2 (October 2009), 173–209.

[Bra87] Bratman, M. E. *Intention, Plans and Practical Reason.* Harvard University Press, 1987.

[BWH07] Bordini, R. H., Wooldridge, M., and Hübner, J. F. *Programming Multi-Agent Systems in AgentSpeak using Jason.* John Wiley & Sons, 2007.

[CA07] Caminada, M., and Amgoud, L. On the evaluation of argumentation formalisms. *Artif. Intell. 171*, 5-6 (2007), 286–310.

[CCS05] Capobianco, M., Chesñevar, C. I., and Simari, G. R. Argumentation and the dynamics of warranted beliefs in changing environments. *Autonomous Agents and Multi-Agent Systems 11*, 2 (2005), 127–151.

[CML00] Chesñevar, C. I., Maguitman, A. G., and Loui, R. P. Logical models of argument. *ACM Comput. Surv. 32*, 4 (2000), 337–383.

[CMS06] Chesñevar, C. I., Maguitman, A. G., and Simari, G. R. Argument-based critics and recommenders: A qualitative perspective on user support systems. *Data Knowl. Eng. 59*, 2 (2006), 293–319.

[CRL00] Carbogim, D. V., Robertson, D., and Lee, J. Argument-based applications to knowledge engineering. *Knowl. Eng. Rev. 15* (June 2000), 119–149.

[CS07] Chesñevar, C., and Simari, G. R. Modelling inference in argumentation through labelled deduction: Formalization and logical properties. *Logica Universalis 1* (2007), 93–124.

[Das08] Dastani, M. 2apl: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems 16*, 3 (2008), 214–248.

[DBC03] Dunne, P. E., and Bench-Capon, T. J. M. Two party immediate response disputes: Properties and efficiency. *Artif. Intell. 149*, 2 (2003), 221–250.

[dBHvdHM07] de Boer, F. S., Hindriks, K. V., van der Hoek, W., and Meyer, J.-J. C. A verification framework for agent programming with declarative goals. *J. Applied Logic 5*, 2 (2007), 277–302.

[DBL03] *The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings* (2003), ACM.

[DdBDM03] Dastani, M., de Boer, F. S., Dignum, F., and Meyer, J.-J. C. Programming agent deliberation: an approach illustrated using the 3apl language. In *AAMAS* [DBL03], pp. 97–104.

[Den87] Dennett, D. *The Intentional Stance*. The MIT Press, Cambridge MA, 1987.

[DHT06] Duff, S., Harland, J., and Thangarajah, J. On proactivity and maintenance goals. In *AAMAS* (2006), H. Nakashima, M. P. Wellman, G. Weiss, and P. Stone, Eds., ACM, pp. 1033–1040.

[DKT06] Dung, P. M., Kowalski, R. A., and Toni, F. Dialectic proof procedures for assumption-based, admissible argumentation. *Artif. Intell. 170*, 2 (2006), 114–159.

[DMT07] Dung, P. M., Mancarella, P., and Toni, F. Computing ideal sceptical argumentation. *Artif. Intell. 171*, 10-15 (2007), 642–674.

[Dun95] Dung, P. M. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. *Artificial Intelligence 77*, 2 (1995), 321–358.

[DvRDM03] Dastani, M., van Riemsdijk, B., Dignum, F., and Meyer, J.-J. C. A programming language for cognitive agents goal directed 3apl. In *PROMAS* (2003), M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, Eds., vol. 3067 of *Lecture Notes in Computer Science*, Springer, pp. 111–130.

[DvRM05] Dastani, M., van Riemsdijk, M. B., and Meyer, J.-J. C. Programming multi-agent systems in 3apl. In Bordini et al. [BDDFS05], pp. 39–67.

[DZ05] Dix, J., and Zhang, Y. Impact: A multi-agent framework with declarative semantics. In Bordini et al. [BDDFS05], pp. 69–94.

[FEGS07] Ferretti, E., Errecalde, M., García, A. J., and Simari, G. R. An application of defeasible logic programming to decision making in a robotic environment. In *LPNMR* (2007), C. Baral, G. Brewka, and J. S. Schlipf, Eds., vol. 4483 of *Lecture Notes in Computer Science*, Springer, pp. 297–302.

[FEGS08] Ferretti, E., Errecalde, M., García, A. J., and Simari, G. R. Decision rules and arguments in defeasible decision making. In Besnard et al. [BDH08], pp. 171–182.

[Gab96] Gabbay, D. *Labelling Deductive Systems (vol.1)*. Oxford University Press (Volume 33 of Oxford Logic Guides), 1996.

[GCS08] Gómez, S. A., Chesñevar, C. I., and Simari, G. R. Defeasible reasoning in web-based forms through argumentation. *International*

Journal of Information Technology and Decision Making 7, 1 (2008), 71–101.

[GGK+11] García, D. R., Gottifredi, S., Krümpelmann, P., Thimm, M., Kern-Isberner, G., Falappa, M. A., and García, A. J. On influence and contractions in defeasible logic programming. In *LPNMR* (2011), J. P. Delgrande and W. Faber, Eds., vol. 6645 of *Lecture Notes in Computer Science*, Springer, pp. 199–204.

[GGS07] Gottifredi, S., García, A. J., and Simari, G. R. Agent programming using defeasible argumentation for knowledge representation and reasoning. In *13vo. Congreso Argentino de Ciencias de las Computación (CACIC 2007)* (2007), pp. 1464–1475.

[GGS08] Gottifredi, S., García, A. J., and Simari, G. R. Defeasible knowledge and argumentative reasoning for 3apl agent programming. In *Twelfth International Workshop on Non-Monotonic Reasoning (NMR 2008)* (2008), pp. 170–178.

[GGS09] Gottifredi, S., García, A. J., and Simari, G. R. Argumentation systems and agent programming languages. In *AAAI 2010 Fall Symposium: The Uses of Computational Argument* (2009).

[GGS10] Gottifredi, S., García, A. J., and Simari, G. R. Query-based argumentation in agent programming. In *IBERAMIA* (2010), Á. F. K. Morales and G. R. Simari, Eds., vol. 6433 of *Lecture Notes in Computer Science*, Springer, pp. 284–295.

[GGS11] Gottifredi, S., García, A. J., and Simari, G. R. Argument types and typed argumentation frameworks. In *(A Aparecer) Theory and Applications of Formal Argumentation* (2011).

[GL87] Georgeff, M. P., and Lansky, A. L. Reactive reasoning and planning. In *National Conference of Artificial Intelligence (AAAI'87)* (1987), pp. 677–682.

[GS04] Garcia, A., and Simari, G. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming (TPLP) 4* (2004), 95–138.

[GTC+10] Gottifredi, S., Tucat, M., Corbata, D., García, A. J., and Simari, G. R. A bdi architecture for high level robot deliberation. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial 14*, 46 (2010), 74–83.

[GTGS09] Gottifredi, S., Tucat, M., García, A. J., and Simari, G. R. An argumentative intentional model for high level reasoning of mobile robots. In *10mo Simposio Argentino de Inteligencia Artificial (ASAI 2009)* (Agosto 2009), pp. 203–214.

[HBW06] Hübner, J. F., Bordini, R. H., and Wooldridge, M. Programming declarative goals using plan patterns. In *DALT* (2006), M. Baldoni and U. Endriss, Eds., vol. 4327 of *Lecture Notes in Computer Science*, Springer, pp. 123–140.

[HdBvdHM99] Hindriks, K. V., de Boer, F. S., van der Hoek, W., and Meyer, J.-J. C. Agent programming in 3apl. *Autonomous Agents and Multi-Agent Systems 2*, 4 (1999), 357–401.

[HdBvdHM00] Hindriks, K. V., de Boer, F. S., van der Hoek, W., and Meyer, J.-J. C. Agent programming with declarative goals. In *ATAL* (2000), C. Castelfranchi and Y. Lespérance, Eds., vol. 1986 of *Lecture Notes in Computer Science*, Springer, pp. 228–243.

[HTT90] Horty, J. F., Thomason, R. H., and Touretzky, D. S. A skeptical theory of inheritance in nonmonotonic semantic networks. *Artif. Intell. 42*, 2-3 (1990), 311–348.

[HvR07] Hindriks, K. V., and van Riemsdijk, M. B. Satisfying maintenance goals. In *DALT* (2007), M. Baldoni, T. C. Son, M. B. van Riemsdijk, and M. Winikoff, Eds., vol. 4897 of *Lecture Notes in Computer Science*, Springer, pp. 86–103.

[IGR92] Ingrand, F. F., Georgeff, M. P., and Rao, A. S. An architecture for real-time reasoning and system control. *IEEE Expert 7*, 6 (1992), 34–44.

[Kra87] Krause, E. F. *Taxicab Geometry*. Dover, 1987.

[KvdT08] Kaci, S., and van der Torre, L. Preference-based argumentation: Arguments supporting multiple values. *International Journal of Approximate Reasoning 48*, 3 (2008), 730–751.

[KW91] Kifer, M., and Wu, J. A first-order theory of types and polymorphism in logic programming. In *LICS* (1991), IEEE Computer Society, pp. 310–321.

[Mey02] Meyer, J.-J. C. Tools and education towards formal methods practice. In *FAABS* (2002), M. G. Hinchey, J. L. Rash, W. Truszkowski, C. Rouff, and D. F. Gordon-Spears, Eds., vol. 2699 of *Lecture Notes in Computer Science*, Springer, pp. 274–279.

[MGS08] Martínez, D. C., García, A. J., and Simari, G. R. Strong and weak forms of abstract argument defense. In Besnard et al. [BDH08], pp. 216–227.

[MT08] Matt, P.-A., and Toni, F. A game-theoretic measure of argument strength for abstract argumentation. In *JELIA* (2008), S. Hölldobler, C. Lutz, and H. Wansing, Eds., vol. 5293 of *Lecture Notes in Computer Science*, Springer, pp. 285–297.

[NL06] Nigam, V., and Leite, J. Adding knowledge updates to 3apl. In *PROMAS* (2006), R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, Eds., vol. 4411 of *Lecture Notes in Computer Science*, Springer, pp. 165–181.

[PBL05] Pokahr, A., Braubach, L., and Lamersdorf, W. Jadex: A bdi reasoning engine. In Bordini et al. [BDDFS05], pp. 149–174.

[Plo04] Plotkin, G. D. A structural approach to operational semantics. *J. Log. Algebr. Program. 60-61* (2004), 17–139.

[Pol94] Pollock, J. L. Justification and defeat. *Artif. Intell. 67*, 2 (1994), 377–407.

[Poo85] Poole, D. On the comparison of theories: Preferring the most specific explanation. In *IJCAI* (1985), pp. 144–147.

[Pra10] Prakken, H. An abstract framework for argumentation with structured arguments. *Argument and Computation 1*, 1 (2010), 93–124.

[PS97] Prakken, H., and Sartor, G. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics 7*, 1 (1997), 25–75.

[PS02] Prakken, H., and Sartor, G. The role of logic in computational models of legal argument: A critical survey. In *Computational Logic: Logic Programming and Beyond* (2002), A. C. Kakas and F. Sadri, Eds., vol. 2408 of *Lecture Notes in Computer Science*, Springer, pp. 342–381.

[PSJ98] Parsons, S., Sierra, C., and Jennings, N. R. Agents that reason and negotiate by arguing. *J. Log. Comput. 8*, 3 (1998), 261–292.

[PV02] Prakken, H., and Vreeswijk, G. Logics for defeasible argumentation. In *Handbook of Philosophical Logic, second edition, vol. 4*, D. Gabbay and F. Guenthner, Eds. Dordrecht etc., 2002, pp. 219–318.

[RA06] Rahwan, I., and Amgoud, L. An argumentation-based approach for practical reasoning. In *ArgMAS* (2006), N. Maudet, S. Parsons, and I. Rahwan, Eds., vol. 4766 of *Lecture Notes in Computer Science*, Springer, pp. 74–90.

[Rao96] Rao, A. S. Agentspeak(l): Bdi agents speak out in a logical computable language. In *MAAMAW* (1996), W. V. de Velde and J. W. Perram, Eds., vol. 1038 of *Lecture Notes in Computer Science*, Springer, pp. 42–55.

[Rei80] Reiter, R. A logic for default reasoning. *Journal of Artificial Intelligence 13*, 1-2 (1980), 81–132.

[RG91] Rao, A. S., and Georgeff, M. P. Modeling rational agents within a bdi-architecture. In *Second Conference on Principles of Knowledge Representation and Reasoning (KR'91)* (1991), pp. 473–484.

[RG98] Rao, A. S., and Georgeff, M. P. Decision procedures for bdi logics. *Journal Logic and Computation 8*, 3 (1998), 293–342.

[RGGS11] Rotstein, N. D., Gottifredi, S., García, A. J., and Simari, G. R. A heuristics-based pruning technique for argumentation trees. In *SUM* (2011), S. Benferhat and J. Grant, Eds., vol. 6929 of *Lecture Notes in Computer Science*, Springer, pp. 177–190.

[RGS07] Rotstein, N. D., Garcia, A. J., and Simari, G. R. Reasoning from desires to intentions: A dialectical framework. In *AAAI Conference on Artificial Intelligence* (2007), pp. 136–141.

[RJ90] R., C. P., and J., L. H. Intention is choise with commitment. *Artificial Intelligence 42* (1990), 213–261.

[RS09] Rahwan, I., and Simari, G. R. *Argumentation in Artificial Intelligence*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[Seb99] Sebesta, R. W. *Concepts of programming languages (4. ed.)*. Addison-Wesley-Longman, 1999.

[SGCS03] Stolzenburg, F., García, A. J., Chesñevar, C. I., and Simari, G. R. Computing generalized specificity. *Journal of Applied Non-Classical Logics 13*, 1 (2003), 87–.

[Sho93] Shoham, Y. Agent-oriented programming. *Artificial Intelligence 60*, 1 (1993), 51–92.

[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.

[SMCS08] Sagui, F. M., Maguitman, A. G., Chesñevar, C. I., and Simari, G. R. Modeling news trust: A defeasible logic programming approach. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial 12*, 40 (2008), 63–72.

[Ten91] Tennent, R. D. *Semantics of programming languages*. Prentice Hall International Series in Computer Science. Prentice Hall, 1991.

[TK93] Thirunarayan, K., and Kifer, M. A theory of nonmonotonic inheritance based on annotated logic. *Artif. Intell. 60*, 1 (1993), 23–50.

[Tou03] Toulmin, S. E. *The Uses of Argument.* Cambridge University Press, 2003.

[vdHWvLBC90] van der Hoek W., van Linder B., and Ch., M. J. J. An integrated modal approach to rational agents. *Fundations of Rational Agency, Applied Logic Series 14* (1990), 133–168.

[VP00] Vreeswijk, G., and Prakken, H. Credulous and sceptical argument games for preferred semantics. In *JELIA* (2000), M. Ojeda-Aciego, I. P. de Guzmán, G. Brewka, and L. M. Pereira, Eds., vol. 1919 of *Lecture Notes in Computer Science*, Springer, pp. 239–253.

[vRDM05] van Riemsdijk, B., Dastani, M., and Meyer, J.-J. C. Semantics of declarative goals in agent programming. In *AAMAS* (2005), F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, Eds., ACM, pp. 133–140.

[vRDM09] van Riemsdijk, M. B., Dastani, M., and Meyer, J.-J. C. Goals in conflict: semantic foundations of goals in agent programming. *Autonomous Agents and Multi-Agent Systems 18*, 3 (2009), 471–500.

[vRDW08] van Riemsdijk, M. B., Dastani, M., and Winikoff, M. Goals in agent systems: a unifying framework. In *Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS'08)* (2008), pp. 713–720.

[vRMdB06] van Riemsdijk, M. B., Meyer, J.-J. C., and de Boer, F. S. Semantics of plan revision in intelligent agents. *Theoretical Computer Science 351*, 2 (2006), 240–257.

[vRvdHM03] van Riemsdijk, B., van der Hoek, W., and Meyer, J.-J. C. Agent programming in dribble: from beliefs to goals using plans. In *AAMAS* [DBL03], pp. 393–400.

[Win05] Winikoff, M. Jack intelligent agents: An industrial strength platform. In Bordini et al. [BDDFS05], pp. 175–193.

[WJ95] Wooldridge, M., and Jennings, N. Intelligent agents: Theory and practice. *Knowledge Engineering Review 2*, 10 (1995), 115–152.

[WPHT02] Winikoff, M., Padgham, L., Harland, J., and Thangarajah, J. Declarative & procedural goals in intelligent agent systems. In *KR* (2002), D. Fensel, F. Giunchiglia, D. L. McGuinness, and M.-A. Williams, Eds., Morgan Kaufmann, pp. 470–481.