FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Completitud e implementación de modalidades en MAS
Autores: Agustín Ambrossio – Leandro Mendoza
Director: Alejandro Fernández
Codirector: Clara Smith
Asesor profesional:
Carrera: Licenciatura en Informática

Resumen

En este trabajo de grado realizamos un estudio sobre lógicas modales y combinaciones de ellas, haciendo énfasis en operadores modales aplicables al desarrollo de sistemas multi-agentes. Algunos de estos operadores son normales y otros no-normales, como es habitual en esta clase de sistemas. Nos concentramos en la combinación de estos dos tipos de operadores, poniendo particular atención a la teoría de modelos y en su utilidad para demostrar completitud y/o decidibilidad de lógicas. Utilizamos dos tipos diferentes de técnicas de combinación: fibrado y unión. Para la técnica de fibrado probamos las propiedades de completitud y modelo finito para dar sustento a implementaciones computacionales para la lógica resultante y habilitar el diseño de algoritmos que computen satisfactibilidad de una fórmula dentro de un modelo; esto es: teniendo un modelo y una fórmula, podemos averiguar si la fórmula es verdadera en tal modelo. Sobre la base de estas propiedades definimos diferentes chequeadores de modelos. Logramos una implementación computacional en los lenguajes PROLOG y SPINdle partiendo de la definición de los chequeadores para el fibrado e investigamos la complejidad computacional de los algoritmos resultantes. Para la técnica de unión construimos un chequeador de modelos.

Palabras Claves

Lógica modal, Combinación de lógicas, Decidibilidad, Completitud, Chequeador de modelos, Sistemas-multi agente.

Trabajos Realizados

Prueba de Completitud. Fibrado. Prueba de existencia de modelo finito para el fibrado. Construcción de un chequeador de modelos (MC). Análisis de complejidad del MC. Construcción de un MC para una segunda combinación de las lógicas (joined logics). Implementación del fibrado en Prolog. Exploración de una implementación computacional de aspectos de la composición usando el lenguaje SPINdle.

Conclusiones

Las propiedades de completitud y modelo finito probadas en esta tesis son sustento de implementaciones computacionales para lógicas proposicionales modales, y habilitan para diseñar algoritmos que computen satisfactibilidad de una fórmula dentro de un modelo. Definimos diferentes chequeadores de modelos (sobre estructuras de Kripke y sobre estructuras de Scott-Montague). Dichos chequeadores tienen un enfoque de programación imperativa.

Trabajos Futuros

Explorar la línea de trabajo referida al estudio de la aplicabilidad de métodos orientados a pruebas. Estudiar métodos para reducir la complejidad de la lógica propuesta para el fibrado que, si bien es decidible, es EXPTIME completa. Extender la teoría lógica actual sobre la idea de razonamiento sobre el tiempo logrando un nivel más de fibrado. Extender las implementaciones computacionales logradas y explorar posibles áreas de aplicación.

Fecha de la presentación: Octubre de 2011

Completitud e Implementación de Modalidades en MAS

Agustín Ambrossio Leandro Mendoza

19 de octubre de 2011

Agradecemos a Antonino Rotolo (CIRSFID - Universidad de Bologna) la asistencia con el filtrado, el proveer bibliografía y la presentación del paper conjunto en AICOL 2011 en Frankfurt; a Cristiano Castelfranchi (ISTC, CNR Roma) por sugerir y discutir sobre modelos realísticos; a Benoit Gaudou (Universidad de Tolouse) por indicarnos un proyecto francés sobre modelos de confianza donde se implementan agentes, una teoría de confianza y reputación.

II

Índice general

Capítulo 1

Introducción

Actualmente los sistemas multi-agente(MAS) constituyen un aspecto importante en muchas áreas relacionadas a las ciencias de la computación (inteligencia artificial, robótica, etc) debido a que son considerados una tecnología clave para facilitar el diseño y la implementación de sistemas complejos. El estudio de MAS está motivado por dos enfoques principales. Uno de los enfoques comprende el análisis experimental y teórico de mecanismos de auto-organización y emergencia que surgen cuando varias entidades autónomas interactúan. El otro enfoque está relacionado con la creación de artefactos distribuidos capaces de realizar tareas complejas mediante cooperación e interacción.

Algunos MAS se conocen como sistemas "intencionales". En tales sistemas, con el fin de dar una representación de los estados mentales y cognitivos de los procesos involucrados, los agentes se representan manteniendo una postura intencional hacia su entorno. Los más conocidos e influyentes son los sistemas creencia-deseo-intención (BDI belief-desire-intention systems). Los agentes BDI son caracterizados a través de un "estado mental" interno que se describe en términos de creencias (beliefs), deseos (desires) e intenciones (intentions).

Uno de los enfoques para la construcción de MAS es el enfoque *formal*. En este enfoque se utilizan elementos de forma (entidades, reglas, axiomas, etc) para la descripción, inferencia y planificación de los agentes y su comportamiento en un ámbito de aplicación. Un enfoque actual formal sobre el diseño de MAS es el de lógicas *modales* o *multi-modales*. Los lenguajes de la lógica proposicional modal son lenguajes proposicionales a los que se les han añadido cierto tipo de operadores, usualmente llamados "modalida-

des" u "operadores modales". Las lógicas modales o multi-modales son, por lo general, una combinación de lógicas especificas (de propósitos especiales) que dan lugar a teorías complejas con varios operadores modales *normales* y "eventualmente" *no normales*. Las lógicas modales normales satisfacen ciertas condiciones de clausura. Una lógica modal es normal si contiene el axioma de distribución (usualmente llamado K) y es cerrada bajo la regla de Generalización [DKV02, BdRV01]. Algunos operadores modales normales muy utilizados en el área de *MAS* son, por ejemplo, los de creencias (Bel_x), objetivos ($Goal_x$), intenciones (Int_x), obligaciones (O). Otros operadores, como por ejemplo el de acción ($Does_x$) son no normales. Una lógica es no normal cuando no satisface el axioma K [Elg97, Che80].

En este trabajo de grado realizamos un estudio sobre lógicas modales o multi-modales y combinaciones de ellas, haciendo énfasis en operadores modales aplicables al diseño de MAS basados en modelos BDI. Proveemos detalles técnicos para la combinación de dos lógicas modales diferentes, una normal y otra no normal, en un mismo MAS. Una es la lógica que modela el comportamiento interno de los agentes (sus creencias, objetivos e intenciones) y la otra modela la interacción que los agentes tienen con su entorno (la habilidad de realizar acciones). Este trabajo está motivado por la dificultad inherente a la combinación de dos lógicas con semántica bien distinta y querer lograr resultados de completitud y/o decibilidad que permitan generar algoritmos e implementaciones computacionales correctas para MAS.

Como base de nuestro estudio usamos la teoría expuesta en [SR10] como teoría ejemplo para analizar y valorar los posibles modos de combinar lógicas con operadores normales y no normales. Esa teoría utiliza una aproximación *multi-modal multi-agente* y usa operadores normales y no normales para definir el concepto de confianza colectiva en MAS. En [SR10] está sugerida la prueba de completitud para los operadores involucrados, pero no desarrollada. En este trabajo de grado realizamos dicha prueba.

Combinar lógicas es una técnica en expansión en los últimos tiempos [AMdNdR99] inspirada por necesidades referidas a modularidad y también por el objetivo de unir lógicas muy específicas (llamadas de propósitos especiales o de poder expresivo restringido). A lo largo de los próximos capítulos ampliamos el contexto modal y combinamos lógicas de propósitos especiales para lograr lógicas aptas para trabajar con creencias internas, acciones, intenciones, objetivos, etc. Nos concentraremos en la combinación de una lógica modal normal y una lógica modal no normal y probamos completitud y decidibilidad de la lógica resultante.

Proponemos un análisis de la lógica [SR10] de modo de verla como una lógica combinada de una manera especial y de esta forma facilitar la construcción de las pruebas. Adoptar esta estrategia nos permite construir la prueba de completitud para la integración de todos los operadores de la lógica base. De esta manera, encontramos resultados de completitud para la teoría de [SR10] combinando pruebas de completitud para los diferentes operadores modales.

Nuestro estudio puede verse organizado en dos partes. La primera consiste en los resultados del análisis teórico del problema de completitud y decidibilidad de la lógica base y la segunda en los resultados de la exploración de posibles implementaciones en función de los resultados obtenidos en la primera etapa. El análisis teórico es de las pruebas de completitud pertinentes a la lógica usada en [SR10]. Para obtener las pruebas estudiamos, entre otros, las técnicas y conceptos del Cap. 4 de [BdRV01] y así poder demostrar la completitud de lógicas modales mediante la construcción de modelos, incluyendo:

1. la noción de completitud fuerte (*strong completeness*, def. 4.10 de [BdRV01]) de una lógica modal respecto de una clase de estructuras,

2. el *Teorema del Modelo Canónico* (toda lógica normal es fuertemente completa respecto de su modelo canónico), y

3. el resultado que permite extender el teorema previo a lógicas con más de un operador modal normal (def. 4.24 de [BdRV01]). Esto último es necesario porque la lógica para la cual pretendemos probar completitud tiene más de un operador.

Para probar entonces la completitud de la lógica testigo nos centramos primero en los resultados de [DKV02] que proveen una demostración de la completitud del operador modal $M\text{-}INT$ (intención mutua). Los autores sugieren que el método de prueba puede ser adaptado para probar la completitud de sistemas con esta clase de operadores. Siguiendo esta sugerencia, usamos el método allí provisto para lograr una demostración de completitud del operador $C\text{-}BEL$(creencia colectiva). En la construcción de modelos canónicos se utilizan conjuntos maximales consistentes, al igual que otras técnicas descriptas en [BdRV01]. Como la teoría testigo [SR10] utiliza como soporte los operadores modales iterativos $C\text{-}BEL$ y $M\text{-}INT$ de [DKV02], los cuales, respectivamente, capturan la semántica de *"creencia común"* e *"intención*

mutua", combinaremos las pruebas de completitud de éstas. Aquí finalizamos el tratamiento de los operadores normales.

Luego estudiamos la integración de las pruebas logradas para operadores normales con las pruebas de completitud de operadores no normales. Ello permite que articulemos todas las demostraciones de completitud necesarias para probar completitud de nuestra lógica base. Evaluamos las técnicas en [Elg97] y [GR04b], ambas para el operador no normal Does. La primera utiliza una aproximación mediante semántica de función de selección, y la segunda usa una semántica de *neighbourhood*. Sabemos que existen equivalencias entre ambas de acuerdo a lo tratado en [GR04b].

Para la prueba de completitud de la lógica testigo realizamos una combinación mediante el uso de dos tipos de técnicas: "fibrado" y "union" (join). El fibrado es una técnica que preserva muchas propiedades de las lógicas combinadas. El método consiste, de manera informal, en unir lógicas como si estuvieran una encima de la otra, en niveles. En nuestro caso, el nivel superior contiene a la restricción normal de la lógica base y el nivel inferior contiene al operador no normal para la acción. Combinamos ambas lógicas utilizando una función de *fibrado*. En el primer nivel nos manejamos con modelos de Kripke ya que la lógica es normal y en el segundo nivel usamos modelos de Scott-Montague [Hal73, GR04a]. Ambos niveles se conectan mediante la función de fibrado, que es una función de mapeo: a cada punto del modelo de Kripke le hace corresponder un modelo Scott-Montague. Al combinarlas, ya sabemos que son completas y que el fibrado en las condiciones en que se hace transfiere la completitud a la lógica resultante.

La otra técnica aplicada es la unión de lógicas. Una unión de lógicas es una unión bidimensional de lógicas donde los mundos posibles se organizan como pares ordenados; cada componente pertenece a una lógica: (parte normal, parte no normal). La unión es una combinación de lógicas más libre que un fibrado: las reglas de formación sintáctica de fórmulas permiten la escritura de un conjunto más amplio de expresiones.

Ambas combinaciones llevan a distintos niveles de expresividad del sistema.

En la segunda parte de nuestra tesis aplicamos los resultados teóricos obtenidos en la primera parte en algo más *"tangible"* para nosotros como informáticos. Estudiamos el formalismo definido en [Nute1987], es una teoría de lógica *rebatible* simple, eficiente y flexible, que permite tratar con diferen-

tes intuiciones acerca del razonamiento *no-monótono*. Después de estudiar esta teoría exploramos una implementación computacional de algunos de los operadores modales sobre la base de dicho formalismo. También enfrentamos al formalismo *Prolog-like* en [GR04a], de tipo declarativo. Construimos una implementación en lenguaje PROLOG utilizando como base la técnica de fibrado. Esta implementación incluye algunas de las definiciones dadas en [SR10] y permite comprobar el comportamiento computacional de los operadores modales involucrados.

A fin de determinar si nuestra propuesta se puede portar a otros lenguajes exploramos una implementación en un lenguaje lógico llamado SPINdle (un razonador para computar las consecuencias lógicas de una teoría "defeasible") *http://spin.nicta.org.au/spindle/* y documentamos las experiencias obtenidas. Así, facilitamos la comprensión de lo que conlleva la implementación de teorías modales en ambientes computacionales actuales.

El paso natural de lógica modal a un programa en versión imperativa está inspirado en los conceptos de "Standard Translation"(ST) y "Correspondence Theory"(CT) [BdRV01]. Dichos conceptos permiten establecer una traducción y consecuente correspondencia entre los lenguajes modales y los lenguajes de primer orden. ST es una reformulación en lógica de primer orden de la definición de satisfacción en lógica modal. Las fórmulas modales pueden ser traducidas a un fragmento de fórmulas con dos variables de la lógica de primer orden. Estas fórmulas contienen una única variable libre que representa el estado actual de un modelo en lógica modal. La variable libre permite imitar la noción de satisfacción local de lógica modal a través de la noción global de satisfacción en lógica de primer orden. Las modalidades son traducidas como cuantificadores que actúan sobre variables que representan estados relacionados al estado actual. Las letras proposicionales de las fórmulas modales son traducidas como predicados. En base a ST el concepto de CT establece que, cuando son interpretadas sobre modelos, las fórmulas modales son equivalentes a fórmulas de primer orden con una única variable libre. De esta manera, las modalidades del lenguaje modal básico con semántica de posibilidad y necesidad pueden ser traducidas a la lógica de primer orden utilizando el cuantificador existencial y universal, respectivamente.

Los resultados obtenidos en esta tesis proveen fundamento para mejoras en el área de *MAS* dentro de ramas como: cooperación, e-commerce, e-goverment, gestión electrónica administrativa, y también con otros temas relacionados, por ejemplo, con la seguridad informática donde existen grados de confianza, tenemos permisos y control de accesos, entre otros.

Organizamos los capítulos de la siguiente manera:

- Capítulo 2. Describimos conceptos esenciales de sistemas multi-agentes y lógica modal.
- Capítulo 3. Describimos conceptos útiles para las pruebas de completitud basadas en la construcción de modelos. Presentamos la teoría de confianza de [SR10] que utilizamos de aquí en adelante como teoría ejemplo/base y construimos la prueba de completitud para esta teoría. Damos la motivación para dividir la prueba según el tipo de operadores: normales o no normales. Luego, probamos completitud sobre el único operador de la teoría testigo para el cual no disponíamos la prueba de completitud (C-BEL). Por último, combinamos los operadores normales de la lógica obteniendo la lógica resultante **N** (que puede verse como una restricción de la lógica base) y probamos su completitud.
- Capítulo 4. Aplicamos la técnica de fibrado de lógicas para organizar y combinar las lógicas de propósitos especiales y lograr una estructura fibrada para la teoría de confianza.
- Capítulo 5. Utilizamos el método de unión (join) de lógicas para organizar y combinar las lógicas de propósitos especiales.
- Capítulo 6. Exploramos una implementación del fibrado en lenguaje PROLOG.
- Capítulo 7. Exploramos una implementación del fibrado en lenguaje SPINdle.
- Capítulo 8. Exponemos las conclusiones que obtuvimos luego de desarrollado todo el trabajo.
- Apéndice A. Código PROLOG de la implementación del fibrado.
- Apéndice B. Código SPINdle de la implementación del fibrado.

Capítulo 2

Sistemas multi-agentes y lógica modal

2.1. Sistemas multi-agentes

En sistemas multi-agentes (MAS) una cuestión de importancia es investigar cómo trabajan los grupos y cómo las tecnologías permiten implementar interacciones entre estos grupos. Desde una perspectiva "distribuida" de la inteligencia artificial, los sistemas multi-agente son sistemas computacionales en los cuales un conjunto débilmente acoplado de agentes autónomos interactúan con el fin de resolver algún problema determinado [RN03, DKV02]. Como dicho problema, por lo general, supera las capacidades individuales de cada agente, éstos explotan su habilidad para comunicarse, cooperar, coordinar y negociar unos con otros. Este tipo de interacción social depende de muchos factores y circunstancias. El área de resolución cooperativa de problemas (CPS, Cooperative Problem Solving) se ocupa de casos donde un conjunto de agentes autónomos deciden trabajar en conjunto, tanto para el cumplimiento de sus propios objetivos como así también en pos del avance del sistema como un "todo".

Considerando estas nociones colectivas, introduciremos el concepto de grupo de agentes. Un grupo o sistema multi-agente es un sistema de agentes, cuyas intenciones mutuas están limitadas de alguna manera [Fer01]. Por lo general, estas limitaciones surgen debido a que cada agente puede cumplir diferentes roles en el grupo, y dichos roles imponen requerimientos en cuanto a su comportamiento e interacción con otros agentes. Estas interacciones dan lugar a situaciones donde patrones colectivos complejos son involucrados.

Algunos MAS se conocen como sistemas intencionales (*intentional*). En tales sistemas, con el fin de dar una representación de los estados mentales y cognitivos de los procesos involucrados, los agentes se representan manteniendo una postura intencional hacia su entorno [DKV02]. Esto se basa en el paradigma de razonamiento práctico [DKV02]: el proceso de decidir, momento a momento, qué acción llevar a cabo para conseguir los objetivos deseados. Los más conocidos e influyentes son los sistemas creencia-deseo-intención (BDI belief-desire-intention systems) [DKV02, vdHV02]. Las arquitecturas BDI se basan en una teoría del razonamiento práctico humano expuesta en [Bra87] y son utilizados actualmente en el desarrollo de software para la programación de agentes inteligentes. Los agentes BDI son caracterizados a través de un "estado mental" interno que se describe en términos de creencias (beliefs), correspondientes a la información que el agente tiene sobre el entorno; deseos (desires), que representan las opciones disponibles para el agente, e intenciones (intentions) que representan la opción elegida. Las intenciones son vistas como una inspiración para la actividad dirigida por objetivos, reflejada en compromisos. Mientras que las creencias (beliefs) son vistas como actitudes "informales" de los agentes, los deseos (desires), objetivos (goals), intenciones y compromisos son vistos como actitudes "motivacionales". Uno de los aspectos mas importantes de la arquitectura BDI es la existencia de modelos lógicos que permiten trabajar (definir y razonar) acerca de agentes BDI. En este trabajo de grado utilizamos sistemas basados en modelos BDI.

2.2. Lógica modal

Los lenguajes de la lógica proposicional modal son lenguajes proposicionales a los que se les han añadido cierto tipo de operadores, usualmente llamados "modalidades" u "operadores modales". Gracias a su simplicidad sintáctica, estos lenguajes son buenas herramientas para describir y razonar acerca de estructuras relacionales. Una estructura relacional es un conjunto no vacío sobre el cual se han definido un cierto número de relaciones; tienen un amplio uso en matemáticas, ciencias de la computación, inteligencia artificial (IA) y lingüística, y también son utilizadas para interpretar lenguajes de primer orden.

La lógica modal es bien aceptada en filosofía y hoy en día en la comunidad de IA como herramienta para el análisis de conceptos. Una modalidad es una palabra o frase que puede aplicarse a una proposición $\mathscr{A}$ para crear una nueva

proposición que hace una afirmación acerca del modo de verdad de $\mathscr{A}$, acerca de las circunstancias bajo las cuales $\mathscr{A}$ es verdadera: cuándo, dónde o cómo $\mathscr{A}$ es verdadera. Ejemplos son: "en el futuro sucederá $\mathscr{A}$"(F$\mathscr{A}$), "está permitido $\mathscr{A}$"(P$\mathscr{A}$), "el agente sabe $\mathscr{A}$"(K$\mathscr{A}$), "es necesario $\mathscr{A}$"($\Box\mathscr{A}$), "alguna ejecución finita del programa π deja al programa en un estado con información $\mathscr{A}$"($<\pi,\mathscr{A}>$), "es demostrable $\mathscr{A}$", entre muchas otras. En la actualidad los simbolismos modales se usan en computación para formalizar esquemas de razonamiento. Para una buena y actual lectura introductoria de conceptos técnicos esenciales de la lógica modal, sugerimos abordar [BdRV01].

Usamos un lenguaje lógico para trabajar. El lenguaje modal básico utilizado se funda sobre un conjunto numerable de proposiciones, denotadas como $P = p, q, r, \ldots$. Expresiones complejas se forman sintácticamente a partir de aquellas de forma inductiva usual usando el operador $\bot$ (la constante *false*), el operador binario $\vee$ (disyunción), y el operador unario $\neg$ (negación). Como el comportamiento proposicional de esta lógica es clásico, asumimos que la constante *true* ($\top$), $\wedge$ (conjunción), $\rightarrow$ (condicional) se definen del modo esperado a partir de los símbolos ya provistos.

Dado el lenguaje, pasemos a las estructuras sobre las que tradicionalmente trabaja una lógica modal. Un *frame* es una dupla $\mathfrak{F}=(W,R)$, tal que W es un conjunto no vacío llamado universo (o dominio) de $\mathfrak{F}$ y R es una relación binaria sobre W. Los elementos de W se llaman puntos, situaciones, mundos o estados. Teniendo un lenguaje con el que escribir fórmulas, un modelo es un par $\mathfrak{M}=(\mathfrak{F},V)$ donde $\mathfrak{F}$ es un frame y V es una función de valuación que asigna a cada proposición p del lenguaje un subconjunto $V(p)$ de W. Así, $V(p)$ es el conjunto de situaciones en el modelo donde p es verdadera.

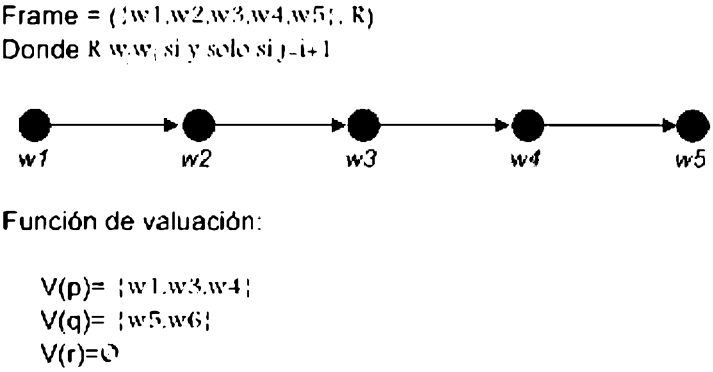


Figura 2.1: Ejemplo de modelo de Kripke

Seguidamente, el aspecto semántico. Sea $\mathfrak{F}=(W,R)$ un frame y sea $w \in W$ un mundo en un modelo $\mathfrak{M}=(\mathfrak{F},V)$. Sean $\mathscr{A},\mathscr{B}$ fórmulas, clásicamente tenemos entonces ($\mathscr{A} \models \mathscr{B}$ significa de $\mathscr{A}$ se deduce $\mathscr{B}$):

- Nunca sucede que $\mathfrak{M}, w \models \bot$,
- $\mathfrak{M}, w \models \mathscr{A}$ si y sólo si $w \in V(\mathscr{A})$,
- $\mathfrak{M}, w \models \neg\mathscr{A}$ si y sólo si no sucede $\mathfrak{M}, w \models \mathscr{A}$,
- $\mathfrak{M}, w \models \mathscr{A} \vee \mathscr{B}$ si y sólo si $\mathfrak{M}, w \models \mathscr{A}$, o $\mathfrak{M}, w \models \mathscr{B}$,
- $\mathfrak{M}, w \models \Box\mathscr{A}$ si y sólo si para todo v tal que wRv, $\mathfrak{M}, v \models \mathscr{A}$.

La precedente es la definición básica de una lógica modal normal [BdRV01].

A lo largo de los próximos capítulos ampliamos el contexto modal para trabajar con la teoría de confianza definida en [SR10].

Capítulo 3

Completitud de una lógica de confianza colectiva

En [SR10] la noción de confianza colectiva en un marco multi-modal es analizada y definida utilizando modalidades normales y no normales. La siguiente tabla muestra los operadores usados, con su respectivo significado.

Operador	Descripción
$BEL_i(\varphi)$	El agente i cree φ
$INT_i(\varphi)$	El agente i tiene la intención φ
$GOAL_i(\varphi)$	El agente i tiene el objetivo φ
$E\text{-}BEL_G(\varphi)$	Todo agente en el grupo G cree φ
$C\text{-}BEL_G(\varphi)$	El grupo G tiene la creencia colectiva φ
$E\text{-}INT_G(\varphi)$	Todo agente en el grupo G tiene la intención individual de hacer φ verdadera
$M\text{-}INT_G(\varphi)$	El grupo G tiene la intención mutua φ
$C\text{-}INT_G(\varphi)$	El grupo G tiene la intención colectiva de hacer φ verdadera
$O(\varphi)$	Es obligatorio φ
$DOES_i(\varphi)$	El agente i lleva a cabo φ

Las modalidades normales usadas son BEL, $GOAL$, INT, O y las modalidades colectivas $C\text{-}BEL$ [1] y $C\text{-}INT$[2], $E\text{-}BEL$[3], $E\text{-}INT$[4] y $M\text{-}INT$[5]. Estas

[1] $C\text{-}BEL_G(\varphi) \leftrightarrow E\text{-}BEL_G(\varphi \wedge C\text{-}BEL_G(\varphi))$

[2] $C\text{-}INT_G(\varphi) \leftrightarrow M\text{-}INT_G(\varphi) \wedge C\text{-}BEL_G(M\text{-}INT_G(\varphi))$

[3] $E\text{-}BEL_G(\varphi) \leftrightarrow \bigwedge_{i \in G} BEL(i, \varphi)$

[4] $E\text{-}INT_G(\varphi) \leftrightarrow \bigwedge_{i \in G} INT(i, \varphi)$

[5] $M\text{-}INT_G(\varphi) \leftrightarrow E\text{-}INT_G(\varphi \wedge M\text{-}INT_G(\varphi))$

modalidades están definidas y estudiadas en [DKV02]. BEL se utiliza para modelar creencias, $GOAL$ se utiliza para modelar objetivos, INT se utiliza para modelar intenciones y O se utiliza para describir obligaciones. Los operadores colectivos $C\text{-}BEL$ y $C\text{-}INT$ se utilizan para modelar creencias colectivas e intenciones colectivas, respectivamente. Los operadores $E\text{-}BEL$ y $E\text{-}INT$ se utilizan para modelar creencias e intenciones de un grupo de agentes, respectivamente. El operador $M\text{-}INT$ se utiliza para modelar intenciones mutuas. El operador $DOES$ es no normal y se utiliza para modelar acciones.

En [SR10] se definen distintos niveles de confianza para MAS y se describen las posibles conexiones entre diferentes formas de confianza grupal y el surgimiento de obligaciones en grupos de agentes.Por ejemplo, supongamos que un grupo de tres agentes x, y y z acuerdan llevar a cabo un objetivo común $\mathscr{A}$. Esto implica, además de algún tipo de coordinación, que cada uno de los agentes "cree" y tiene la intención conjunta de llevar a cabo el objetivo $\mathscr{A}$. Este simple acuerdo entre los agentes involucrados supone un marco de confianza colectiva relativamente elaborado. A continuación introducimos la noción de confianza definida en [SR10]:

$$\begin{aligned} Trust^x_y\mathscr{A} \leftrightarrow GOAL_x\mathscr{A} \wedge BEL_x Does_y\mathscr{A} \wedge \\ INT_x(Does_y\mathscr{A} \wedge \neg Does_x\mathscr{A}) \wedge BEL_x INT_y\mathscr{A} \wedge GOAL_x INT_y\mathscr{A} \end{aligned} \tag{3.1}$$

que significa: "el agente x confía en el agente y respecto a $\mathscr{A}$ siempre que x tenga como objetivo a $\mathscr{A}$, y cree que y realizara $\mathscr{A}$, y x no tiene la intención de realizar $\mathscr{A}$ él mismo sino que y lo realize por él, y x tiene el objetivo que y tenga la intención de hacerlo y crea en ello".

Los siguientes ejemplos muestran distintos niveles de confianza modelados:

- Ejemplo 1: *Confianza conjunta.* Supongamos que un agente y está en la parada del micro, y hay un grupo G de gente cerca de y pero que no está en la parada, esperando que y levante la mano y pare el micro.

$$Jtrust^G_y\mathscr{A} \leftrightarrow \bigwedge_{i \in G} Trust^i_y\mathscr{A} \tag{3.2}$$

- Ejemplo 2: *Credibilidad.* Es el cumpleaños de Juan. Sus compañeros de trabajo juntaron dinero y se lo dieron a y (otro compañero de trabajo

que irá al centro de la ciudad) confiando en que buscará y comprar un regalo para Juan. Todos confían en que y hará esa tarea.

$$Rel_y^G \mathscr{A} \leftrightarrow (Jtrust_y^G \mathscr{A}) \wedge M - INT_G(Jtrust_y^G \mathscr{A}) \tag{3.3}$$

- Ejemplo 3: *Confianza colectiva.* Grupos de vecinos de distintos barrios de una ciudad construyen grandes muñecos en las calles rellenos de petardos y fuegos artificiales que serán quemados el último día del año. Cada grupo construye su propio muñeco desde cero, a partir de la figura del año. Las autoridades gubernamentales locales regulan la actividad de los grupos y organizan un concurso otorgando un premio al mejor muñeco. Los muñecos construidos son cuidados y vigilados día y noche para evitar que sean "saboteados" ya que es muy común que otros grupos envíen a un "saboteador" para quemar muñecos de sus competidores antes de la fecha prevista. El fin de esta acción es eliminarlo de la competencia. De esta manera, un grupo de vecinos G encarga a uno de sus miembros s la tarea de quemar el muñeco del grupo H.

$$Ctrust_y^G \mathscr{A} \leftrightarrow (Rel_y^G) \wedge C - BEL_G(Rel_y^G \mathscr{A}) \tag{3.4}$$

La "confianza conjunta" (join trust), consiste en el hecho de que todos los individuos en un grupo confían en otro agente para llevar a cabo su objetivo, la "credibilidad" (reliance) requiere cierta coordinación intencional mutua con el grupo y "confianza colectiva" (collective trust) asume que el grupo es consciente de tal coordinación para alcanzar un objetivo.

Dos propiedades importantes de un procedimiento de deducción son su corrección y su completitud. Un procedimiento de deducción es correcto si cualquier consecuencia lógica que haya sido probada con él es válida. Un procedimiento de deducción que permita probar cualquier consecuencia válida es completo. Evidentemente, la corrección es una propiedad deseable en todo procedimiento de deducción, ya que en caso contrario el uso del procedimiento podría conducir a conclusiones inválidas. La completitud es también una propiedad deseable, aunque hemos estudiado que no es posible garantizarla para todas las lógicas.

Para realizar la prueba de completitud de la lógica base, dividimos la prueba en partes de acuerdo a la naturaleza de los operadores que la componen. En primer lugar, dedicaremos este capítulo a enfocarnos en la parte

normal de la teoría, es decir, aquella que involucra sólo los operadores normales. Dejaremos para el próximo capítulo el análisis de completitud para el operador no normal y para la combinación de operadores normales y no normales. La motivación de esta separación radica en el hecho de que al momento de enfrentarnos a esta teoría nuestra formación en computación teórica nos permitió tratar naturalmente los operadores normales primero y juntos. Luego intentamos unir al resultado de completitud para los operadores normales al estudio de completitud del operador no normal. Tenemos presente los conocimientos y herramientas para realizar pruebas de completitud para lógicas normales; los mecanismos para realizar ese tipo de pruebas sobre lógicas no normales podían requerir un abordaje diferenciado y así lo fue.

Al conjunto de fórmulas que contienen sólo modalidades normales lo llamamos lógica **N** (o restricción **N** de la teoría de confianza en [SR10]). La prueba de completitud para **N** implica, en principio, probar completitud para cada uno de los operadores normales y luego utilizar los resultados obtenidos para lograr una prueba de completitud de la lógica resultante de la combinación de dichos operadores. Ya conocemos las pruebas de completitud para todos los operadores normales [HM92], excepto para el operador *C-BEL* que no está probado. Construimos entonces una demostración para dicho operador sobre una clase particular de frames llamada $KD45$ [6][HM92].

En las próximas secciones introducimos definiciones básicas y desarrollamos una prueba de completitud para el operador *C-BEL* utilizando como base el método de prueba para *M-INT* [DKV02]. Posteriormente, desarrollamos una prueba de completitud para la lógica **N**, que integra todos los operadores modales normales tratados y que es una restricción de la lógica original a la parte normal.

[6] $KD45$ Es la clase de frames que se utiliza para razonar sobre estados epistémicos [vdHV02] de agentes en MAS. Utilizamos $KD45_n$ para distinguirla de la lógica monomodal $KD45$. $KD45_n$ es la lógica conocida tradicionalmente como "lógica doxástica". Se usa para caracterizar "beliefs" de agentes. La lógica tiene un subíndice n, ello porque tenemos un operador i para cada i menor o igual que n (o sea, se trata una lógica multi-modal). K: es la lógica modal normal mínima; D: los agentes no creen inconsistencias; 4: si creen algo, creen que lo creen (positive introspection); 5: si no creen algo, creen que no lo creen (negative introspection). Semántica: $KD45$ es la clase de los frames *sin cota a derecha* + *transitivos* + *euclideanos*.

3.1. Definiciones básicas

Sabemos que los teoremas de completitud son esencialmente teoremas de existencia de modelos. Como ya hemos mencionado, dada una lógica normal se prueba completitud con respecto a alguna clase de estructuras mostrando que cada subconjunto consistente de fórmulas en dicha lógica es satisfactible en algún modelo dado. Esta idea está en la base del *Teorema de Compacidad* [Ham78, BdRV01] que establece que un conjunto (posiblemente infinito) de fórmulas bien formadas tiene un modelo si todos sus subconjuntos finitos tienen un modelo.

Para las pruebas de las secciones siguientes usamos una clase especial de modelos basados en conjuntos maximales consistentes (MCSs) [BdRV01]. Cada estado del modelo (sección 2.2) se corresponde con un MCS. Estos modelos se denominan modelos canónicos y su uso está basado en dos observaciones principales:

- *Observación 1*: Cada estado w en un modelo $\mathfrak{M}$ está asociado a un conjunto de fórmulas: $\{ \varphi | \mathfrak{M}, w \models \varphi \}$. Estos conjuntos de fórmulas son en sí MCSs. Esto significa que si φ es verdadera en algún modelo dado, entonces φ pertenece a un MCSs.

- *Observación 2*: Si existe una relación entre dos estados w y w' en algún modelo $\mathfrak{M}$ significa que la información de los MCSs correspondientes está relacionada de algún modo. Esto nos permite suponer que los modelos permiten descubrir colecciones de MCSs relacionados.

El objetivo es probar un *Lema de Verdad* [BdRV01] que dice que si una fórmula φ pertenece a un MCS entonces φ es verdadera en algún modelo. Para realizar esta prueba, construimos un tipo especial de modelo, llamado modelo canónico de Kripke, cuyos puntos son todos los conjuntos maximales consistentes (MCSs) de la lógica en cuestión. La prueba del *Lema de Verdad* implica probar, entre otras cosas, un *Lema de Existencia* [BdRV01] que establece que existe un conjunto suficiente de MCSs relacionados que nos aseguran el éxito de la construcción del modelo.

De esta manera, antes de comenzar con la construcción de las pruebas de completitud, lo primero a resolver es cómo construir el modelo que necesitamos. Para ello, comenzamos por definir un modelo de Kripke para la teoría de confianza [SR10].

Definición 1 Modelo de Kripke para la restricción N de la teoría de confianza estudiada en [SR10]

Definimos un modelo de Kripke para **N** como una tupla:
$\mathfrak{M} = \{W, \{B_i : i \in \mathscr{A}\}, \{G_i : i \in \mathscr{A}\}, \{I_i : i \in \mathscr{A}\}, Val\}$, tal que:

- W es un conjunto de estados (o mundos) posibles.
- Para cada $i \in \mathscr{A}$ se cumple que B_i, G_i, $I_i \in W \times W$. Estas son las relaciones de accesibilidad para cada agente i con respecto a *Beliefs (creencias)*, *Goals (objetivos)* e *Intentions (intenciones)*, respectivamente.
- $Val : P \times W \to \{0, 1\}$ es la función de valuación que asigna valores de verdad a las proposiciones atómicas en estados del modelo. Es decir, dada una letra $p \in P$ y un estado $w \in W$, nos dice si p es verdadera o no en el estado w, indicando 1 o 0, respectivamente. Recordemos que P es el conjunto de proposiciones atómicas $p, q, r, ..., etc.$ La figura 2.1 muestra un ejemplo de un modelo de Kipke.

■

Ahora construimos la prueba de completitud de $KD45_n^{C-BEL_G}$.

3.2. Prueba de completitud para $KD45_n^{C-BEL_G}$

En esta sección probamos la completitud del operador de creencias colectivas *C-BEL*. Nos basamos en el capítulo 5 de [DKV02] (donde se desarrolla una demostración de completitud para el operador modal *M-INT*). Los autores sugieren allí que el método de prueba puede ser adaptado para probar la completitud de esta clase de operadores. Siguiendo esta sugerencia, usamos el método provisto por ellos para lograr una demostración de completitud del operador colectivo *C-BEL*.

El método de prueba en realidad se basa en un método comúnmente utilizado en lógica modal para realizar pruebas de completitud [7].

Debemos probar que, suponiendo que $KD45_n^{C-BEL_G} \nvdash \varphi$, hay un modelo $KD45_n$ $\mathfrak{M}$ y un $w \in \mathfrak{M}$, tal que $\mathfrak{M}, w \not\models \varphi$. Esto es, si φ no es teorema (aspecto sintáctico), entonces φ no es verdadera en $\mathfrak{M}$ (aspecto semántico).

[7]La prueba se realiza con respecto a modelos finitos. Estudiaremos mas adelante que los modelos con lo que nosotros trabajamos también cumplen esta propiedad

La prueba tiene cuatro pasos:

1. Construimos un conjunto finito de fórmulas Φ, llamado la clausura o cierre de φ. Este conjunto contiene a φ y a todas sus sub-fórmulas, además de otras fórmulas que serán utilizadas en pasos posteriores. Es importante notar que Φ es finito.

2. Aplicamos el *Lema de Lindenbaum* [BdRV01]: un conjunto consistente de sentencias que pertenecen a Φ siempre puede ser extendido a un conjunto de fórmulas *maximal consistente* en Φ.

3. Construimos un contra-modelo donde sus estados son conjuntos maximales consistentes (MCSs) y definimos un conjunto apropiado de relaciones de accesibilidad entre dichos estados.

4. Mostramos, utilizando inducción sobre las fórmulas en Φ, que el modelo construido en el paso 3 contiene un estado en el cual φ es falsa.

Definición 2 Clausura de una sentencia φ.

La clausura de una sentencia φ con respecto a $KD45_n^{C-BEL_G}$ es el conjunto mínimo de fórmulas $\Phi \in KD45_n^{C-BEL_G}$, tal que para cada $G \in \{1, 2, ..., n\}$ se cumple lo siguiente [8]:

1. $\varphi \in \Phi$.

2. Si $\psi \in \Phi$ y χ es una sub-fórmula de ψ, entonces $\chi \in \Phi$.

3. Si $\psi \in \Phi$ y ψ por si misma no es una negación, entonces $\neg\psi \in \Phi$.

4. Si C-BEL$_G(\psi) \in \Phi$, entonces E-BEL$_G(\psi) \wedge$ C-BEL$_G(\psi)) \in \Phi$.

5. Si E-BEL$_G(\psi) \in \phi$, entonces BEL$(i, \psi) \in \Phi$, para todo $i \in G$.

6. $\neg$BEL$(i, \bot) \in \Phi$, para todo $i \leq n$. (Ningún agente cree lo falso).

■

Definición 3 Conjunto maximal $KD45_n^{C-BEL_G}$ – *consistente*.

Un conjunto finito de fórmulas Γ ($\Gamma \in \Phi$) es maximal $KD45_n^{C-BEL_G}$ – *consistente* en Φ si y sólo si:

[8] La notación utilizada es la misma que en [DKV02]. Posteriormente se escribe con otra notación, por ejemplo, $BEL_i\ \varphi$

1. Γ es $KD45_n^{C-BEL_G}$ – *consistente*. Ej: $KD45_n^{C-BEL_G} \not\vdash \neg(\bigwedge_{\psi\in\Gamma} \psi)$. Esto significa que Γ no contiene una contradicción, es decir, que no puede deducirse simultáneamente ψ y $\neg\psi$.

2. No hay $\Gamma' \subset \Phi$ tal que $\Gamma \subset \Gamma'$ y Γ' siga siendo $KD45_n^{C-BEL_G}$ – *consistente*. Esto significa que, además de ser Γ consistente, cualquier otro conjunto Γ' obtenido a partir de Γ agregando cualquier otra nueva fórmula, será inconsistente.

■

Aplicación del Lema de Lindenbaum.

El *Lema de Lindenbaum* [BdRV01] establece que cualquier conjunto consistente de fórmulas puede ser extendido a uno maximal consistente. Por lo tanto, aplicamos dicho lema al conjunto de nuestro interés. Sea Φ la clausura (o cierre) de Φ con respecto a $KD45_n^{C-BEL_G}$. Si $\Gamma \subseteq \Phi$ es $KD45_n^{C-BEL_G}$ – *consistente*, entonces hay un conjunto $\Gamma' \supseteq \Gamma$ $KD45_n^{C-BEL_G}$ – *consistente* en Φ.

Construcción el modelo canónico.

Definición 4 Modelo canónico Consideremos $\mathfrak{M}_\varphi =< S_\phi, \pi, B_1, ..., B_n >$ un modelo de *Kripke* definido de la siguiente manera:

- Los estados de $\mathfrak{M}_\varphi$ son todos los conjuntos maximales $KD45_n^{C-BEL_G}$ – *consistentes*. Por el *Lema de Lindenbaum*, cualquier conjunto $KD45_n^{C-BEL_G}$ – *consistente* es un subconjunto de algún estado en $\mathfrak{M}_\varphi$, y por el *Lema Finito de Verdad* que probaremos luego, cualquier conjunto $KD45_n^{C-BEL_G}$ – *consistente* es verdadero en algún estado del modelo.

  Definimos un estado S_Γ para cada conjunto $KD45_n^{C-BEL_G}$ –*consistente* $\Gamma \subseteq \Phi$(Φ es la clausura de φ). Como Φ es finito, solo hay un número finito de estados. Formalmente, definimos

  $$CON_\Phi = \{\Gamma \; tal \; que \; \Gamma \; es \; maximal \; KD45_n^{C-BEL_G} - consistente, en \Gamma\} \; y$$

  $$S_\Gamma = \{s_\Gamma \mid \Gamma \in CON_\Phi\}$$

- Definimos relaciones B_i del siguiente modo:

$$B_i = \{(s_\Gamma, s_\Delta) \mid \psi \in \Delta \textit{ para todo } \psi \textit{ tal que } BEL_i \in \Gamma\}$$

La relación canónica de accesibilidad entre MCSs captura la idea que dos MCSs están "coherentemente" relacionados en función de las fórmulas que valen en cada uno. La figura 3.1 muestra un ejemplo de la relación definida.

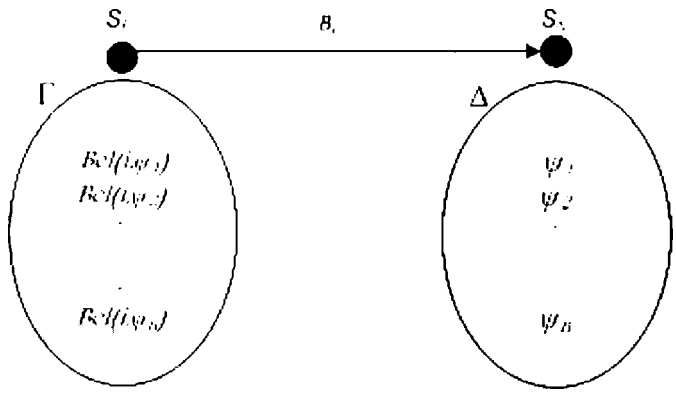


Figura 3.1: Ejemplo relación de accesibilidad B_i

- Para hacer una asignación π verdadera, queremos conformar los átomos proposicionales en los conjuntos maximales consistentes de cada estado. Por lo tanto, definimos $\pi(s_\Gamma)(p) = 1$ si y sólo si $p \in \Gamma$. La valuación canónica iguala la verdad de un símbolo proposicional en un estado s_Γ con su pertenencia a s_Γ. Observar que esto hace a todos los átomos proposicionales que no están en φ falsos en cada estado del modelo.

Con esta definición tenemos que $\mathfrak{M}_\varphi, s_\Gamma \models p$ *sii* $p \in \Gamma$ para los átomos proposicionales p.

■

Trabajamos con MCSs relacionados para construir el modelo canónico buscado. Ahora nuestro objetivo es probar el *Lema de Verdad*, que afirma que si una fórmula φ pertenece a un MCS, entonces es verdadera en algún modelo. Para llevar a cabo esta prueba es necesario probar algunas propiedades esenciales de de los conjuntos *maximales* $KD45_n^{C-BEL_G}$ *– consistentes* en Φ; el *Lema de Consecuencia* y el *Lema de Valuación.*

Probamos ahora que el *Lema de Consecuencia* se verifica en nuestro caso.

Si $\Gamma \in CON_\Phi$, $\psi_1, ..., \psi_n \in \Gamma$, $\chi \in \Phi$ y $KD45_n^{C-BEL_G} \vdash \psi_1 \rightarrow (\psi_2 \rightarrow (...(\psi_n \rightarrow \chi)...)$, entonces $\chi \in \Gamma$.

La prueba se realiza utilizando razonamiento proposicional estándar sobre el conjunto *maximal consistente.*

■

Probamos ahora del Lema de Valuación Finita.

Si Γ es $KD45_n^{C-BEL_G}$ – *consistente* en alguna clausura Φ, entonces para toda fórmula ψ, χ se tiene que:

1. Si $\neg\psi \in \Phi$, entonces $\neg\psi \in \Gamma$ *sii* $\psi \notin \Gamma$.
2. Si $(\psi \wedge \chi) \in \Phi$, entonces $(\psi \wedge \chi) \in \Gamma$ *sii* $\psi \in \Gamma$ y $\chi \in \Gamma$.
3. Si $BEL(i, \psi) \in \Phi$, entonces $BEL(i, \psi) \in \Gamma$ *sii* $\psi \in \Delta$ *para todo* Δ con $(s_\Gamma, s_\Delta) \in B_i$.
4. Si $E\text{-}BEL_G(\psi) \in \Phi$, entonces $E\text{-}BEL_G(\psi) \in \Gamma$ *sii* $\psi \in \Delta$ *para todo* Δ *y todo* $i \in G$ *tal que* $(s_\Gamma, s_\Delta) \in B_i$.
5. Si $C\text{-}BEL_G(\psi) \in \Phi$, entonces $C\text{-}BEL_G(\psi) \in \Gamma$ *sii* $\psi \in \Delta$ *para todo* Δ *que es* G_i*-alcanzable desde* s_Γ.

Prueba: Los puntos 1 y 2 se prueban utilizando técnicas estándar de lógica modal. Probaremos a continuación los puntos 3, 4 y 5 que son los que involucran al operador de nuestro interés (BEL).

3er Caso (BEL). Supongamos $BEL(i, \psi) \in \Phi$.

$\Longrightarrow$ Supongamos que $BEL(i, \psi) \in \Gamma$, y supongamos también que $(s_\Gamma, s_\Delta) \in B_i$. Luego, por definición de B_i, inmediatamente obtenemos que $\psi \in \Delta$.

$\Longleftarrow$ Supongamos que $BEL(i, \psi) \notin \Gamma$. Debemos mostrar que hay un Δ tal que $(s_\Gamma, s_\Delta) \in B_i$ y $\psi \notin \Delta$. Es suficiente para mostrar lo siguiente: el conjunto de fórmulas $\Delta' = \{\chi \mid BEL(i, \psi) \in \Gamma\} \cup \{\neg\psi\}$ es $KD45_n^{C-BEL_G}$*-consistente*. Si esta afirmación es verdadera, entonces por el *Lema de Lindenbaum* existe un conjunto maximal $KD45_n^{C-BEL_G}$*-consistente* $\Delta \supseteq \Delta'$ en Φ. Por las definiciones de Δ' y B_i tenemos que $(s_\Gamma, s_\Delta) \in B_i$, y por lo propuesto en el punto 1, tenemos que $\psi \notin \Delta$, como se desea. Ahora vamos a probar la afirmación planteada.

Con el fin de derivar en una contradicción, supongamos que Δ' no es $KD45_n^{C-BEL_G}$-*consistente*. Como Δ' es finito, podemos suponer que $\{\chi \mid BEL(i,\chi) \in \Gamma\} = \{\chi_1, ..., \chi_n\}$. Luego, por definición de inconsistencia tenemos que:

$$KD45_n^{C-BEL_G} \vdash \neg(\chi_1 \wedge ... \wedge \chi_n \wedge \neg\psi)$$

Aplicando razonamiento proposicional (equivalencias lógicas) obtenemos:

$$KD45_n^{C-BEL_G} \vdash \chi_1 \rightarrow (\chi_2 \rightarrow (...(\chi_n \rightarrow \psi)...))$$

Por aplicación de $(R2)$[9], más un numero de aplicaciones de $(A2)$[10] mas razonamiento proposicional, podemos derivar:

$$KD45_n^{C-BEL_G} \vdash BEL(i,\chi_1) \rightarrow (BEL(i,\chi_2) \rightarrow (...(BEL(i,\chi_n \rightarrow BEL(i,\psi))...))$$

Sin embargo, sabemos que $BEL(i,\chi_1), ..., BEL(i,\chi_n) \in \Gamma$ y $BEL(i,\psi) \in \Phi$, entonces, por el *Lema de Consecuencia*, $BEL(i,\psi) \in \Gamma$), que contradice nuestra suposición inicial.

4to Caso (E-Bel_G). Supongamos $E\text{-}BEL_G(\psi) \in \Phi$; luego, por construcción de Φ, también ocurre que, $BEL(i,\psi) \in \Phi$ para todo $i \in G$.

$\Longrightarrow$ Supongamos que $E\text{-}BEL_G(\psi) \in \Gamma$. Por aplicación del axioma $(C1)$ [11] y razonamiento proposicional sencillo tenemos que $KD45_n^{C-BEL_G} \vdash E\text{-}BEL_G(\psi) \rightarrow BEL(i,\psi)$ para todo $i \in G$. Debido a que $BEL(i,\psi) \in \Phi$ podemos usar el lema de consecuencia y derivar que $BEL(i,\psi) \in \Gamma$ para todo $i \in G$. Por esto, debido al paso $\Longrightarrow$ del caso 3, tenemos que $\psi \in \Delta$ para todo Δ y todo $i \in G$ tal que $(s_\Gamma, s_\Delta) \in B_i$, como se desea.

---

[9] $(R2)$ Regla de generalización para creencias (Beliefs). De φ inferimos BEL(i, φ) [DVDK07].

[10] $(A2)$ Axioma distribución de creencias (Beliefs). $BEL(i,\varphi) \wedge BEL(i,\varphi \rightarrow \psi) \rightarrow BEL(i,\psi)$ [DVDK07].

[11] Axioma creencia general. $E\text{-}BEL_G(\varphi) \leftrightarrow \bigwedge_{i \in G} BEL(i,\varphi)$ [DVDK07].

$\Longleftarrow$ La prueba tiene la estructura del paso $\Longrightarrow$, esta vez usando $(C1)$ y el paso $\Longleftarrow$ del caso 3 (BEL).

5to Caso: $C\text{-}Bel_G$

Sea $s_\Gamma \longrightarrow^k s_\Delta$ la representación de "s_Δ es G_1-*alcanzable* desde s_Δ en k pasos". Supongamos $C\text{-}BEL_G(\psi) \in \Phi$; luego por construcción de Φ también ocurre que $E\text{-}BEL_G(\psi \wedge C\text{-}BEL_G(\psi)) \in \Phi$. Lo mismo sucede para sus subfórmulas.

$\Longrightarrow$ Supongamos $C\text{-}BEL_G \in \Gamma$. Probaremos por inducción que para todo $k \geq 1$ y todo Δ, si $s_\Gamma \longrightarrow^k s_\Delta$, luego $\psi, C\text{-}BEL_G(\psi) \in \Delta$. Notar que esto es aún mas fuerte que lo que se necesita para el paso $\Longrightarrow$; pero trabajar con esta hipótesis inductiva facilita el trabajo en la demostración.

- **Para k=1.** Supongamos que $s_\Gamma \longrightarrow^1 s_\Delta$; esto significa que $\Gamma B_i \Delta$ para algún $i \in G$. Por axioma $C2$[12] tenemos que $KD45_n^{C\text{-}BEL_G} \vdash C\text{-}BEL_G(\psi) \rightarrow E\text{-}BEL_G(\psi \wedge C\text{-}BEL_G)$.
  De esta manera $C\text{-}BEL_G(\psi) \in \Gamma$ y $E\text{-}BEL_G(\psi \wedge C\text{-}BEL_G(\psi)) \in \Phi$, y aplicando el *Lema de Consecuencia* implica que $E\text{-}BEL_G(\psi \wedge C\text{-}BEL_G(\psi)) \in \Gamma$. Pero luego, por el punto 4, el paso $\Longrightarrow$ de 3, y el punto 2, concluimos que $\psi, C\text{-}BEL_G(\psi) \in \Delta$, como se desea.
- **Para k=k+1.** Supongamos que $s_\Gamma \longrightarrow^{k+1} s_\Delta$ para algún $n \geq 1$, luego hay un Δ' tal que $s_\Gamma \longrightarrow^n s_{\Delta'}$ y $s_{\Delta'} \longrightarrow s_\Delta$. Por hipótesis de inducción, tenemos ψ y que $C\text{-}BEL_G(\psi) \in \Delta'$. Ahora, igual que en el caso base $k = 1$, podemos probar que las fórmulas $\psi, C\text{-}BEL_G(\psi)$ son transferidas desde Δ' al sucesor directo Δ.

$\Longleftarrow$ Esta vez trabajamos directamente y no por contraposición. Supongamos $\psi \in \Delta$ para todo Δ para el cual s_Δ es G_B-*alcanzable* desde s_Γ. Tenemos que probar que $C\text{-}BEL_G(\psi) \in \Gamma$.

Antes de seguir hacemos una observación general. Como cada s_Δ se corresponde con un conjunto finito de fórmulas Δ, cada Δ puede ser representado como una conjunción finita de sus fórmulas, denotado

[12] Axioma creencia común. $C\text{-}BEL_G(\varphi) \leftrightarrow E\text{-}BEL_G(\varphi \wedge C\text{-}BEL_G(\varphi))$ [DVDK07].

como φ_Δ. Observar que es muy importante y fundamental que nos restrinjamos a la clausura finita de Φ.

Ahora definamos W como $\{\Lambda \in CON_\Phi \mid \psi \in \Delta$ *para todo* Δ *para el cual* s_Λ *es* G_B*-alcanzable desde* $s_\Gamma\}$. En particular, $\Gamma \in W$. Intuitivamente, W se convertirá en el conjunto de estados posibles en donde $C\text{-}BEL_G(\psi)$ es verdadera.

Sea $s_W = \bigvee_{\Lambda \in W} \varphi_\Lambda$. Esta fórmula es la disyunción de las descripciones de todos los estados en W. De la finitud de W, se desprende que φ_W es una fórmula del lenguaje. De manera similar, definimos $\varphi_{\overline{W}} = \bigvee_{\Theta \in \overline{W}} \varphi_\Theta$, donde $\overline{W} = \{\Theta \in CON_\Theta \mid \Theta \notin W\}$.

Nuestro objetivo es probar lo siguiente:

$$KD45_n^{C-BEL_G} \vdash \varphi_W \rightarrow E\text{-}BEL_G(\varphi_W)$$

En primer lugar, vamos a mostrar cómo la demostración de esta afirmación nos ayuda a probar la conclusión deseada: $C\text{-}BEL_G(\psi) \in \Gamma$. Como $\psi \in \Gamma$ para todo $\Lambda \in W$ y ψ ocurre en todas las conjunciones φ_Λ para todo $\Lambda \in W$, tenemos que $KD45_n^{C-BEL_G} \vdash \varphi_W \rightarrow \psi$. Partiendo de esto y de la afirmación propuesta anteriormente podemos distribuir $E\text{-}BEL_G$ sobre la implicación aplicando sucesivamente $R2$, $A2$, $C1$ y razonamiento proposicional para derivar $KD45_n^{C-BEL_G} \vdash \varphi_W \rightarrow E\text{-}BEL_G(\psi \wedge \varphi_W)$. La regla $C1$ inmediatamente nos da $KD45_n^{C-BEL_G} \vdash \varphi_W \rightarrow C\text{-}BEL_G(\psi)$. Ahora, como φ_Γ es una de las disyunciones de φ_W tenemos que $KD45_n^{C-BEL_G} \vdash \varphi_\Gamma \rightarrow C\text{-}BEL_G(\psi)$. Finalmente, usando el *Lema de Consecuencia* y razonamiento proposicional, concluimos $C\text{-}BEL_G(\psi) \in \Gamma$.

■

Por lo tanto, en este punto, solo nos queda probar la afirmación $KD45_n^{C-BEL_G} \vdash \varphi_W \rightarrow E\text{-}BEL_G(\varphi_W)$. Dicha prueba se realizará en los próximos cinco pasos:

1. Mostraremos que para todo $i \in G$ y para todo $\Lambda \in W$ y $\Theta \in \overline{W}$, $KD45_n^{C-BEL_G} \vdash \varphi_\Lambda \rightarrow BEL(i, \neg\phi_\Theta)$. Supongamos que $\Lambda \in W$ y $\Theta \in \overline{W}$. Por definición de W y $\overline{W}$, tenemos que $\psi \in \Delta$ para todo Δ para el cual s_Δ es G_B*-alcanzable* desde s_Λ, pero hay

un Δ' tal que $s_{\Delta'}$ es G_B-*alcanzable* desde s_Θ y $\psi \notin \Delta'$. Por consiguiente,$(s_\Lambda, s_\Theta) \notin B_i$, para cualquier $i \in G$. Elegimos un $i \in G$. Por definición de B_i, hay una fórmula χ_i tal que $BEL(i, \chi_i) \in \Lambda$ mientras que $\chi_i \notin \Theta$. Como Θ es *maximal* $KD45_n^{C-BEL_G}$-*consistente* en Φ, tenemos que $KD45_n^{C-BEL_G} \vdash \varphi_\Theta \rightarrow \neg\chi_i$, en consecuencia, por contraposición, $KD45_n^{C-BEL_G} \vdash \chi_i \rightarrow \neg\varphi_\Theta$. Usando (R2) y (A2), derivamos $KD45_n^{C-BEL_G} \vdash BEL(i, \chi_i) \rightarrow BEL(i, \neg\varphi_\Theta)$, y como $BEL(i, \chi_i) \in \Lambda$, tenemos que $KD45_n^{C-BEL_G} \vdash \varphi_\Lambda \rightarrow BEL(i, \neg\varphi_\Theta)$.

2. $KD45_n^{C-BEL_G} \vdash \varphi_\Lambda \rightarrow BEL(i, \bigwedge_{\Theta \in \overline{W}} \neg\varphi_\Theta)$. Esto se deriva de *1* por lógica proposicional y la regla de derivación de la lógica modal estándar donde las creencias (beliefs) se distribuyen sobre las conjunciones[13].

3. Ahora mostraremos que $KD45_n^{C-BEL_G} \vdash \bigvee_{\Delta \in CON_\Phi} \varphi_\Delta$.

   Prueba Supongamos lo contrario, que la fórmula $\neg \bigvee_{\Delta \in CON_\Phi} \varphi_\Delta$, (que es equivalente a $\bigwedge_{\Delta \in CON_\Phi} \neg\varphi_\Delta$), es $KD45_n^{C-BEL_G}$-*consistente*. Podemos encontrar para cada $\Delta \in CON_\Phi$ una conjunción ψ_Δ de φ_Δ tal que $\overline{\Delta} = \{\neg\psi_\Delta \mid \Delta \in CON_\Phi\}$ es $KD45_n^{C-BEL_G}$-*consistente*.

   Por lo tanto, por *Lema de Lindenbaum*, hay un conjunto de fórmulas $\Theta \supseteq \overline{\Delta}$ el cual es *maximal* $KD45_n^{C-BEL_G}$-*consistente* en Φ. Ahora, llegamos a la deseada contradicción mediante diagonalización: Θ contiene tanto a ψ_Θ (la cual fue definida como una conjunción de φ_Θ) y también a $\neg\psi_\Theta$ (debido a que $\Theta \supseteq \overline{\Delta}$).

4. $KD45_n^{C-BEL_G} \vdash \varphi_W \leftrightarrow \bigwedge_{\Theta \in \overline{W}} \neg\varphi_\Theta$. Esto se deriva de forma inmediata a partir del punto (3).

5. Aquí mostramos la proposición final, que establece:

$$KD45_n^{C-BEL_G} \vdash \varphi_W \rightarrow E\text{-}BEL_G(\varphi_W)$$

   Prueba. Por los puntos (2) y (4) tenemos que para todo $i \in G$ que $KD45_n^{C-BEL_G} \vdash \varphi_\Gamma \rightarrow BEL(i, \varphi_W)$, entonces por aplicación de *(M1)* y razonamiento proposicional, $KD45_n^{C-BEL_G} \vdash \varphi_\Gamma \rightarrow$

[13]Axioma K de distribución. $\Box(p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q)$ [BdRV01].

$E\text{-}BEL_G(\varphi_W)$; finalmente, como $\Gamma \in W$, nuestra proposición se cumple.

■

Probamos ahora el Lema de Verdad Finita.

Si $\Gamma \in CON_\Phi$, entonces para todo $\psi \in \Phi$ se cumple que $\mathcal{M}_\varphi, s_\Gamma \models \psi$ *sii* $\psi \in \Gamma$.

Prueba Se prueba inmediatamente utilizando *Lema de Valuación Finita* o por inducción sobre la estructura de ψ.

■

Finalmente, el resultado que estamos buscando:

Completitud de $KD45_n^{C-BEL_G}$

Si $KD45_n^{C-BEL_G} \nvdash \varphi$, hay un modelo finito $\mathfrak{M}$ y un $w \in \mathfrak{M}$, tal que $\mathfrak{M}, w \not\models \varphi$.

Prueba Supongamos que $KD45_n^{C-BEL_G} \nvdash \varphi$. Sea $\mathfrak{M}_\varphi$ un modelo canónico. Hay una fórmula χ lógicamente equivalente a $\neg\varphi$ que es un elemento de Φ; si φ no comienza con una negación, χ es la fórmula $\neg\varphi$ en sí misma. Ahora, aplicando el *Lema de Lindenbaum*, hay un conjunto maximal consistente $\Gamma \subseteq \Phi$ tal que $\chi \in \Gamma$. Por el Lema de Verdad Finita, esto implica que $\mathfrak{M}_\varphi, s_\Gamma \models \chi$, por lo tanto $\mathfrak{M}_\varphi, s_\Gamma \not\models \varphi$.

■

3.3. Transferencia de propiedades entre lógicas

Cuando hablamos de combinación de lógicas [AMdNdR99], nos referimos a integrar lógicas de propósitos especiales (con poder de expresión limitado) para lograr una lógica resultante de mayor poder de expresión. Este concepto de combinación de lógicas para propósitos de modelado está inspirado en ideas de modularidad con el fin de permitir la integración de diferentes tipos de información. Un aspecto importante a tener en cuenta al realizar una combinación es determinar si las propiedades individuales de cada lógica se

transfieren correctamente a la lógica resultante.

Una manera de ver a la lógica testigo es como una combinación de operadores normales, luego combinada con operadores no normales. Así, la parte normal que llamamos **N** es una restricción de la teoría testigo. Si bien sabemos que la propiedad de completitud para todos los operadores (normales y no normales) se cumple si los consideramos de a uno, el hecho de que cada uno de los operadores sea completo no implica que la combinación de todos lo sea [AMdNdR99, BdRV01]. Analizamos seguidamente a **N** como una lógica normal multimodal resultante de la unión de lógicas de propósitos especiales mas pequeñas. Los operadores normales en **N** son: BEL, $GOAL$, INT, $C\text{-}BEL$, $C\text{-}INT$, $E\text{-}BEL$, $E\text{-}INT$ y $M\text{-}INT$.

Es importante destacar que ver la lógica original como una combinación de lógicas es una estrategia para probar completitud.

A continuación construimos una prueba de completitud para **N**.

El método utilizado está otra vez inspirado en la prueba de completitud para operadores colectivos utilizada en [DKV02]. Adaptamos dicha demostración a **N** y, para concluir la prueba con el resultado esperado, aplicamos la definición 4.24 y el teorema 4.22 de [BdRV01]. La definición 4.24 describe cómo construir un modelo canónico para una lógica modal normal con mas de un operador. El teorema 4.22 dice que una lógica es completa con respecto a su modelo canónico.

Tenemos que probar que, suponiendo $\mathbf{N} \nvdash \varphi$, hay un modelo $\mathfrak{M}_N$ y un $w \in \mathfrak{M}_N$ tal que $\mathfrak{M}_N, w \not\models \varphi$. Esto significa que si probamos que φ no es teorema, entonces no es verdad en $\mathfrak{M}_N$.

La prueba tiene cuatro pasos:

1. Construir la clausura de una sentencia φ.
2. Aplicar el *Lema de Lindenbaum* sobre la clausura construida en el punto 1.
3. Construir el modelo canónico según la definición 4.24 [BdRV01].
4. Probar completitud de **N** aplicando el teorema 4.22 de [BdRV01].

Paso 1. Clausura de una sentencia φ.

Construimos un conjunto finito de fórmulas Φ, llamado la clausura de φ. Φ contiene φ y todas sus sub-fórmulas más un conjunto de otras fórmulas que serán necesarias en el paso 4 de la demostración para mostrar cómo una valuación apropiada puede hacer falsa a φ en un cierto estado w.

La clausura de una sentencia φ con respecto a la lógica **N** es el conjunto mínimo de fórmulas Φ de **N** tal que para cada $G \in \{1,2,...,n\}$ se cumple lo siguiente:

1. $\varphi \in \Phi$.
2. Si $\psi \in \Phi$ y χ es una sub-fórmula of ψ, entonces $\chi \in \Phi$;
3. Si $\psi \in \Phi$ y Φ por sí misma no es una negación, entonces $\neg\psi \in \Psi$;
4. Si $M\text{-}INT_G(\psi) \in \Phi$ entonces $E\text{-}INT_G(\psi \wedge M\text{-}INT_G(\psi)) \in \Phi$;
5. Si $E\text{-}INT_G(\psi) \in \Phi$ entonces $INT(i,\psi) \in \Phi$ para todo $i \in G$;
6. Si $C\text{-}INT_G(\psi) \in \Phi$ entonces $M\text{-}INT_G(\psi) \wedge C\text{-}BEL_G(M\text{-}INT_G(\psi)) \in \Phi$ para todo $i \in G$;
7. Si $E\text{-}BEL_G(\psi) \in \Phi$ entonces $BEL(i,\psi) \in \Phi$ para todo $i \in G$;
8. Si $C\text{-}BEL_G(\psi) \in \Phi$ entonces $E\text{-}BEL_G(\psi \wedge C\text{-}BEL_G(\psi)) \in \Phi$;
9. $\neg INT(i,\bot) \in \Phi$ para todo $i \leq m$;
10. $\neg BEL(i,\bot) \in \Phi$ para todo $i \leq m$;
11. $\neg GOAL(i,\bot) \in \Phi$ para todo $i \leq m$.

Observar que para cada fórmula φ, Φ es un conjunto finito de fórmulas.

Paso 2. Aplicación del Lema de Lindenbaum.

La construcción de un modelo canónico implica definir los estados que contendrá dicho modelo. Estos estados serán conjuntos maximales *N-consistentes*. Para obtener estos conjuntos, aplicamos el *Lema de Lindenbaum* sobre la clausura de una sentencia φ definida en el punto 1, de la siguiente manera:

Sea Φ la clausura (o cierre) de φ con respecto a la lógica normal **N**. Si $\Gamma \subseteq \Phi$ es $N - consistente$, entonces hay un conjunto $\Gamma' \supseteq \Gamma$ que es maximal

N-consistente en Φ.

Paso 3. Construcción del modelo canónico.

Construimos un modelo canónico utilizando la definición 4.24 [BdRV01]. Este modelo contendrá un estado donde $\neg\varphi$ sea verdadera.

Sea $\mathfrak{M}_\varphi =< S_\phi, \pi, I_1, ..., I_m, B_1, ... B_m, G_1, ..., G_m >$ un modelo de Kripke definido de la siguiente manera:

- Como dominio de estados, un estado s_Γ se define para cada conjunto $N-consistente$ $\Gamma \subseteq \Phi$. Notar que como Φ es finito, solo hay un número finito de estados. Formalmente, definimos $CON_\Phi = \{\Gamma | \Gamma$ es maximal $N - consistente$ en $\Phi\}$ y $S_\varphi = \{s_\Gamma | \Gamma \in CON_\Phi\}$.

- Las correspondientes relaciones canónicas se definen de la siguiente manera:

$$I_i = \{(s_\Gamma, s_\triangle) | \psi \in \triangle \text{ para todo } \psi \text{ tal que } INT_i(\psi) \in \Gamma\}$$

$$B_i = \{(s_\Gamma, s_\triangle) | \psi \in \triangle \text{ para todo } \psi \text{ tal que } BEL_i(\psi) \in \Gamma\}$$

$$G_i = \{(s_\Gamma, s_\triangle) | \psi \in \triangle \text{ para todo } \psi \text{ tal que } BEL_i(\psi) \in \Gamma\}$$

- Para hacer una asignación π verdadera, queremos conformar los átomos proposicionales en los conjuntos maximales consistentes de cada estado. Por lo tanto, definimos $\pi(s_\Gamma)(p) = 1$ si y sólo si $p \in \Gamma$. Notar que esta asignación hace que a los átomos proposicionales que no estén en φ falsos en cada estado del modelo.

Paso 4. Prueba de completitud de N.

Utilizamos lo desarrollado anteriormente y aplicamos el teorema 4.22 de [BdRV01] para concluir la demostración. Este teorema nos dice que una lógica modal normal es completa con respecto a su modelo canónico.

Si $\mathbf{N} \nvdash \varphi$ entonces hay un modelo $\mathfrak{M}$ y un estado w tal que $\mathfrak{M}, w \not\models \varphi$.

Prueba: Supongamos $\mathbf{N} \nvdash \varphi$. Tomemos $\mathfrak{M}_\varphi$ tal cual se definió en el paso (3). Notar que hay una fórmula χ lógicamente equivalente a $\neg\varphi$ que es un elemento de Φ; si ψ no comienza con una negación, χ es la fórmula $\neg\varphi$ en

sí misma. Ahora, aplicando el *Lema de Lindenbaum*, obtenemos un conjunto maximal consistente $\Gamma \subseteq \Phi$ tal que $\chi \in \Gamma$. Por el *Lema de Verdad* [BdRV01], sabemos que dada una fórmula p, $\mathfrak{M}_{\varphi}, s_{\Gamma} \models p$ sii $p \in \Gamma$, lo cual implica que $\mathfrak{M}_{\varphi}, s_{\Gamma} \models \chi$, por lo tanto $\mathfrak{M}_{\varphi}, s_{\Gamma} \not\models \varphi$.

Dejamos la demostración de la correcta aplicación de los lemas y propiedades necesarios para utilización del *Lema de Verdad*. El mecanismo de aplicación es similar al utilizado en la prueba de completitud construida en la sección anterior para el operador colectivo $C\text{-}BEL$.

■

Capítulo 4

Combinación de Lógicas

4.1. Combinación de Lógicas

En los capítulos anteriores hemos demostrado la completitud de algunas lógicas usadas como soporte formal de una teoría de confianza. Ahora nos concentramos en distintas formas de combinarlas y en las restricciones que cada tipo de combinación impone.

Entre los distintos tipos de combinaciones usadas habitualmente en el estudio de lógicas modales, trabajamos en esta tesis de grado con: el fibrado (temporalización), y la "unión' de lógicas (joined logics)[FMDR04]. Cada una de éstas conlleva intrínsecamente distintos niveles de expresabilidad, es decir, los conjuntos de *fbfs* que podemos escribir y manipular en cada una son diferentes.

4.2. Nociones Básicas

Antes de aplicar la técnica de fibrado a la lógica testigo, presentamos unas definiciones útiles.

Definición 5 ***Modelo de Scott-Montague.*** De acuerdo con Hansson y Gärdenfors [HG73], podemos generalizar la semántica tradicional de Kripke de la siguiente manera. En lugar de tener una colección de mundos conectados a un mundo w mediante una relación R, consideramos un conjunto de colecciones de mundos conectados a w. Estas colecciones son los *vecindarios* (*neighbourhoods*) de w. Formalmente, un frame de Scott-Montague es un par ordenado $\langle W, N \rangle$ donde W es un conjunto de mundos y N es una fun-

ción total que asigna para cada w en W un conjunto de subconjuntos de W (los vecindarios de w). Un modelo de Scott-Montague es una terna $\langle W, N, V\rangle$ donde $\langle W, N\rangle$ es un frame de Scott-Montague y V es una función de valuación definida como la de los frames de Kripke, excepto para $\Box\mathscr{A}$: es verdadero en w sii el conjunto de elementos de W donde $\mathscr{A}$ es verdadero en *uno* de los conjuntos en $N(w)$: es un vecindario de w.

Definición 6 ***Modelo Multi-Relacional.*** Recordemos la estructura del *MAS* en [SR10]. Es un frame multi-relacional de la forma:

$$\mathfrak{F} = \langle A, W, \{B_i\}_{i\in A}, \{G_i\}_{i\in A}, \{I_i\}_{i\in A}, \{D_i\}_{i\in A}\rangle$$

donde:

- A es un conjunto finito de agentes;
- W es un conjunto de situaciones, puntos, o mundos posibles;
- $\{B_i\}_{i\in A}$ es un conjunto de relaciones de accesibilidad respecto de BEL, las cuales son transitivas, euclidianas y seriales;
- $\{G_i\}_{i\in A}$ es un conjunto de relaciones de accesibilidad respecto de $GOAL$, (semántica K_n standard);
- $\{I_i\}_{i\in A}$ es un conjunto de relaciones de accesibilidad respecto de INT, las cuales son seriales; y
- $\{D_i\}_{i\in A}$ es una familia de conjuntos de relaciones de accesibilidad D_i con respecto a $DOES$, que son cerradas punto a punto con respecto a la intersección, reflexivas y seriales [GR04b].

Un modelo basado en un frame $\mathfrak{F}$ tiene la forma $\langle \mathfrak{F}, V\rangle$, donde V es la función de valuación correspondiente ([SR10], Definition 2)[1].

Ahora basándonos en la definición 4.24 y el teorema 4.22 de [BdRV01] (los cuales respectivamente establecen cómo construir un modelo canónico para una lógica modal normal, y que una lógica modal normal es fuertemente completa con respecto a su modelo canónico). Primero, la signatura **N** construida a partir de las modalidades normales tiene modelo canónico;

[1]En esta definición no incluimos la modalidad O. Ésta fue incorporada después para tratar con la connotación deóntica de la teoría base ([SR10], Section 4). Dejaremos de lado por el momento este operador para mantener manejable el conjunto de modalidades.

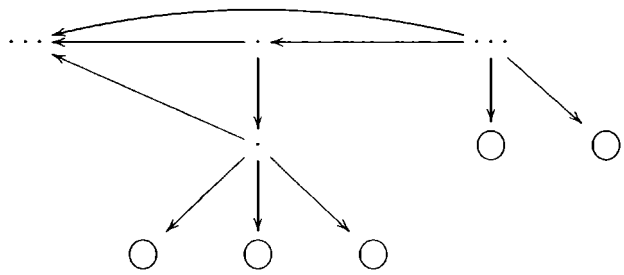
segundo, esta lógica es completa con respecto a su modelo canónico.

En [SR10], la modalidad $DOES$ es aplicada a constantes proposicionales que representan acciones o comportamientos, por ejemplo, $DOES_x Pagar$ (significa, "el agente x paga"). Ello restringe el conjunto de fórmulas a usar: por ejemplo la fbf $DOES_i(DOES_j Paga)$ (significa, "el agente i hace que el agente j pague") no puede escribirse en el sistema original.

Definición 7 ***Fibrado.*** Intuitivamente fibrar dos lógicas L y M (anotamos $L(M)$)consiste en organizarlas en dos niveles, el nivel superior para la lógica L, y en segundo la lógica M. Si aplicamos definición de fibrado dada por Finger y Gabbay [FG94], (también véase [FMDR04]) la lógica basada en $\mathfrak{F}$ puede ser vista como una combinación donde la lógica modal normal es puesta sobre una lógica no normal. La parte no normal también es multi-modal ya que, existe una modalidad $DOES_i$ para cada agente i. Notemos que es posible identificar dos "redes" de relaciones sobre el mismo conjunto W. La primera "red" (o multi-grafo) corresponde al "cableado" de los operadores normales, la segunda red corresponde a las relaciones de accesibilidad para las modalidades $Does_i$.

Avancemos sobre lo propuesto.

$\mathfrak{F}$ puede verse dividido en un frame multi-modal exterior, y frames de Scott-Montague interiores. Gráficamente:



Donde los puntos pequeños son elementos del modelo de Kripke y los círculos grandes son modelos de Scott-Montague.

Bajo esta organización, la intuición detrás de la valuación de las fórmulas en el sistema es la siguiente. Cuando evaluamos operadores normales (es decir, cuando parseamos una fbf), navegamos por el modelo de Kripke exterior. Cuando un $DOES_i$ aparece en algún punto w para ser evaluado, navegamos

por un modelo de Scott-Montague. Luego de evaluar la fbf en el modelo de Scott-Montague, substituiremos el resultado con algún objeto (fbf, variable proposicional, etc.) que pertenezca al dominio de los modelos de Kripke (nivel superior) para continuar con la evaluación.

A continuación organizamos a $\mathfrak{F}$ como un fibrado. La siguiente sección trata los aspectos técnicos del fibrado.

4.3. Fibrado: Sintaxis y Semántica

Llamamos **N** a la restricción de $\mathfrak{F}$ a su parte normal. Llamamos Does a la restricción de $\mathfrak{F}$ a su parte no normal. Podemos asumir que Does es una lógica proposicional ya que solo la estamos aplicando a fórmulas átomicas que pueden ser reemplazadas mediante constantes proposicionales sin perder expresividad [Elg97, GR04b]. De acuerdo con el método en [FMDR04], vamos a partir el conjunto de fórmulas de Does en dos subconjuntos: uno de fórmulas booleanas, BDoes, y otro de fórmulas monolíticas, MDoes. Una fórmula $\mathscr{A}$ pertenece a BDoes si su operador externo es booleano (ej. $DOES_x\ \mathscr{A} \wedge DOES_x\ \mathscr{B}$); cualquier otra pertenece a MDoes (ej. $DOES_x\ \mathscr{A}$). Queda claro que no hay intersección entre las modalidades de **N** y de Does. Llamaremos **N**(Does) al fibrado de Does mediante **N**.

N(Does): Sintaxis. Sea $\mathscr{L}_{\text{Does}}$ el lenguaje lógico asociado a los agentes (como lo vimos en el capítulo 3, sin modalidades normales y sin las correspondientes reglas sintácticas de formación). y $\mathscr{L}_{\mathbf{N}}$ el lenguaje asociado a **N** (como en el capítulo 2, sin la modalidad $DOES$ y sin su regla de formación sintáctica). El lenguaje $\mathscr{L}_{\mathbf{N}(\text{Does})}$ de **N**(Does) -sobre el conjunto de letras proposicionales P- es obtenido reemplazando la regla de formación de las fórmulas en $\mathscr{L}_{\mathbf{N}}$ que dice "cada letra proposicional en P es una fórmula" por la regla:

cada fórmula monolítica en $\mathscr{L}_{Does}$ es una fórmula

Como lo expone [FG94], este reemplazo puede ser visto como un proceso llamado "fuzzling" o layering: fórmulas en el sistema base (en este caso las de Does) pueden ser sustituidas por átomos en el sistema superior (en este caso las de **N**).

Para describir formalmente la semántica del fibrado, necesitamos reformular los modelos basados en $\mathfrak{F}$ en términos de los modelos restringidos.

Definición 8 ***Modelo Fibrado.*** Un modelo para **N**(Does) tiene la estructura:

$$\langle A, W, \{B_i\}_{i\in A}, \{G_i\}_{i\in A}, \{I_i\}_{i\in A}, \{d_i\}, V'\rangle \tag{4.1}$$

donde:

- A es un conjunto finito de agentes;

- W es un conjunto de puntos, situaciones, o mundos posibles;

- $\{B_i\}_{i\in A}$ es un conjunto de relaciones de accesibilidad con respecto a BEL, que son transitivas, euclideanas y seriales;

- $\{G_i\}_{i\in A}$ es un conjunto de relaciones de accesibilidad con respecto a $GOAL$, con semántica K estándar;

- $\{I_i\}_{i\in A}$ es un conjunto de relaciones de accesibilidad con respecto a INT, que son seriales;

- V' es la función de valuación V restringida a los operadores normales, definida como sigue:

    1. condiciones booleanas estándar;
    2. $V'(w, BEL_i\ \mathscr{A}) = 1$ sii $\forall v \in W$(si wB_iv entonces $V'(v, \mathscr{A}) = 1$);
    3. $V'(w, GOAL_i\ \mathscr{A}) = 1$ sii $\forall v \in W$(si wG_iv entonces $V'(v, \mathscr{A}) = 1$);
    4. $V'(w, INT_i\ \mathscr{A}) = 1$ sii $\forall v \in W$(si wI_iv entonces $V'(v, \mathscr{A}) = 1$); y

- cada d_i es una función total que mapea, para cada mundo w en W, para cada agente i, un modelo con estructura:

$$\langle W, D_i, \mathrm{v}\rangle \tag{4.2}$$

donde:

- W es el (mismo, el original) conjunto de mundos,

- D_i es una familia de relaciones de accesibilidad $\mathtt{D}_i$ con respecto al accionar del agente i, las cuales son cerradas punto a punto con respecto a la relación de inclusión, reflexivas y seriales [GR04b].

- v es V restringida a los operadores no normales (los $DOES_i$). Esto es, la función de valuación que dice cuándo $DOES_i\ \mathscr{A}$ es verdadera en w: si y sólo si el conjunto de mundos donde $\mathscr{A}$ es verdadera en alguno de los vecindarios de w. Formalmente v está definida como:

  1. condiciones booleanas standard;
  2. $\mathrm{v}(w, DOES_i\ \mathscr{A}) = 1$ *sii* $\exists \mathtt{D}_i \in D_i$ *tal que* $\forall u(w\mathtt{D}_i u$ *sii* $\mathrm{v}(u, \mathscr{A}) = 1)$.

Gráficamente nuestro fibrado tiene la siguiente forma:

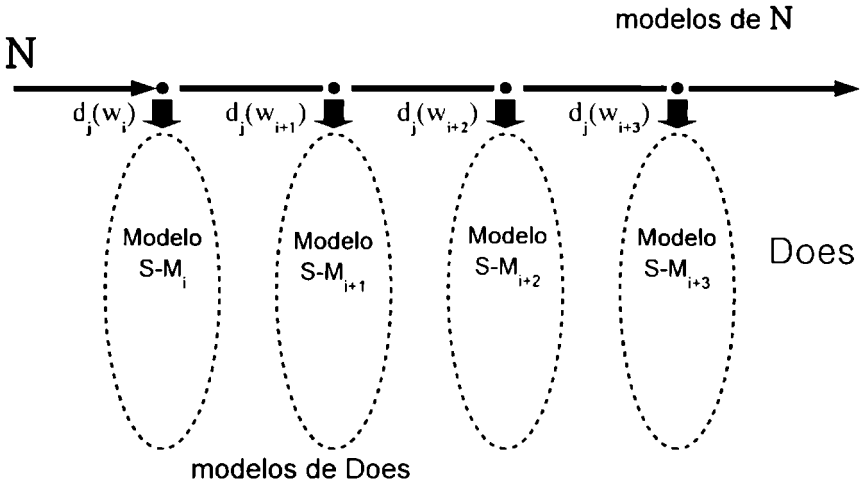


Figura 4.1: Estructura del fibrado **N**(Does)

Llamamos $\mathcal{K}^{\mathscr{L}_{\mathrm{Does}}}$ al conjunto de modelos para $\mathscr{L}_{\mathrm{Does}}$. Entonces d_i: $W \to \mathcal{K}^{\mathscr{L}_{\mathrm{Does}}}$.

N(Does): Semántica. Dado un modelo $\mathfrak{M}$ con estructura como en (4.1); un $w \in W$, y una función de valuación V' en $\mathfrak{M}$, y dadas funciones d_i (el fibrado propiamente dicho), la semántica para **N**(Does) es obtenida reemplazando la cláusula de **N** que dice

$$\mathfrak{M}, w \models p \textit{ sii } p \in V'(w), \textit{si } p \in P,$$

por la cláusula:

$$\mathfrak{M}, w \models \mathscr{A} \textit{ sii } d_i(w) \models \mathscr{A}, \textit{cuando } \mathscr{A} \in \mathrm{MDoes}.$$

Debemos notar que si $\mathscr{A}$ tiene la forma $DOES_i\mathscr{B}$, $\mathscr{A}$ es una fórmula monolítica.

Respecto de la evaluación de *fbfs*, comenzamos recorriendo el modelo exterior (de Kripke) hasta que nos encontramos con una modalidad *Does*, luego a través de la función de fibrado "traemos" un modelo de Scott-Montague de estructura (4.2).

Una vez que nos metemos en una componente *DOES* del sistema no podemos regresar al nivel superior [FMDR04]. Por ello, en el sistema no podremos verificar la validez de fórmulas como $DOES_i(GOAL_j\ \mathscr{A})$ (que puede ser vista como una forma de persuasión: "el agente i hace que el agente j tenga $\mathscr{A}$ como objetivo").

Con respecto a la noción de fibrado usada, hay que notar que combinamos las lógicas de una manera relativamente simple: no existen axiomas especiales ni interacciones complejas entre los operadores modales. Esto nos permite asegurar resultados de adecuación, completitud y decibilidad para la lógica resultante. Fijamos el número de agentes para manejar un número no infinito. Para los operadores normales, aplicamos los resultados del capítulo anterior; para la lógica de la acción, aplicamos [GR04b]. Procediendo de esta manera, aprovechamos las pruebas separadas de las propiedades de cada lógica [FMDR04] y construimos la hipótesis de que el fibrado $\mathbf{N}_{\text{Does}}$ es completo. En la sección siguiente nos ocupamos de los resultados de decidibilidad que verifican nuestra hipótesis.

4.4. Decidibilidad

Hemos dicho que si las lógicas que componen el fibrado son decidibles, la teoría resultante lo será [FMDR04, FG94]. Ahora probamos que esta hipótesis es cierta.

La lógica del Does tiene la propiedad de *modelo finito* (*fmp*)[GR04b, SP08]. Esta propiedad nos permite desentendernos de modelos arbitrarios infinitos, ya que siempre podemos encontrar uno finito equivalente [GR04b]. Además esta lógica posee la propiedad de *modelo de tamaño múltiple de un paso (OSPMP* es un propiedad de modelo pequeño para lógicas sin transiciones semáticas) [SP08]. Gracias a esta propiedad, podemos asegurar que la lógica también es decidible (y esta acotada a *P-SPACE*).

Ahora nos concentramos en probar la decibilidad de la lógica **N**. Podemos hacer esto trivialmente, como en el caso de Does, asegurando alguna propiedad como la de modelo finito u *OSPMP*. Usaremos el concepto de filtros (o filtraciones) para hallar la propiedad de modelo finito [BdRV01, Che80].

Para probar que **N** tiene la propiedad de modelo finito recordamos el concepto de *clausura* utilizado en la demostración de la completitud de la lógica **N** (esta vez sin las fórmulas extra que se mencionan en la demostración de completitud). En vez de clausurar una fórmula (φ) como en el caso anterior, aplicamos la clausura sobre un conjunto de fórmulas Σ. Asumimos que $G \subseteq A$, ambos son conjuntos finitos numerables.

Definición 9 ***Clausura.*** Un conjunto de fórmulas Σ es *cerrado (clausurado) con respecto a sus sub-fórmulas* si para fórmulas cualesquiera φ, φ' se cumple:

1. *si* $\varphi \vee \varphi' \in \Sigma$, *entonces* φ *y* $\varphi' \in \Sigma$
2. *si* $\neg\varphi \in \Sigma$, *entonces* $\varphi \in \Sigma$
3. *si* $C\text{-}BEL_G(\varphi) \in \Sigma$ *entonces* $E\text{-}BEL_G(\varphi \wedge C\text{-}BEL_G(\varphi)) \in \Sigma$;
4. *si* $E\text{-}BEL_G(\varphi) \in \Sigma$ *entonces* $BEL_i(\varphi) \in \Sigma$ *para todo* $i \in G$;
5. *si* $M\text{-}INT_G(\varphi) \in \Sigma$ *entonces* $E\text{-}INT_G(\varphi \wedge M\text{-}INT_G(\varphi)) \in \Sigma$;
6. *si* $E\text{-}INT_G(\varphi) \in \Sigma$ *entonces* $INT_i(\varphi) \in \Sigma$ *para todo* $i \in G$.

■

Recurrimos a la noción de *filtros* ("filtrations") para demostrar la existencia de modelos finitos equivalentes a cada modelo dado.

Definición 10 ***Filtros ("Filtrations").*** Sea $\mathfrak{M}$ un modelo de estructura (4.1) y Σ un conjunto de fórmulas cerrado respecto de sus sub-fórmulas. Sea $\rightsquigarrow_\Sigma$ la relación sobre los estados de $\mathfrak{M}$ definida por:

$$w \rightsquigarrow_\Sigma v \textit{ sii para toda } \varphi \in \Sigma : (\mathfrak{M}, w \models \varphi \textit{ sii } \mathfrak{M}, v \models \varphi)$$

Claramente $\rightsquigarrow_\Sigma$ es una relación de equivalencia. Intuitivamente si una fbf φ vale en v y en w, los mundos son "iguales". Denotamos la clase de equivalencia de un estado w de $\mathfrak{M}$ con respecto a $\rightsquigarrow_\Sigma$ de la manera usual

anotando $|w|_\Sigma$ o simplificando $|w|$ si no es necesaria la aclaración de sobre qué conjunto operamos. La función de mapeo de $w \mapsto |w|$ que envía un estado a su clase de equivalencia se llama *función de mapeo natural*[BdRV01, Che80].

Ahora definiremos el filtro para la lógica **N**:

Sea $W_\Sigma = \{|w|_\Sigma \mid w \in W\}$. Supongamos que $\mathfrak{M}^f_\Sigma$ es un modelo cualquiera $\langle W^f, \{B^f_i\}_{i\in A}, \{G^f_i\}_{i\in A}, \{I^f_i\}_{i\in A}, V^f\rangle$ tal que:

- $W^f = W_\Sigma$,
- *Para toda modalidad* $M_i(\varphi) \in \Sigma, \mathfrak{M}, w \models M_i(\varphi) \rightarrow \mathfrak{M}, v \models M_i(\varphi) \Leftrightarrow \mathfrak{M}^f_\Sigma, |w| \models M_i(\varphi) \rightarrow \mathfrak{M}^f_\Sigma, |v| \models M_i(\varphi)$,
- *Si* $B_i wv$ *entonces* $B^f_i |w||v|$,
- *Si* $G_i wv$ *entonces* $G^f_i |w||v|$,
- *Si* $I_i wv$ *entonces* $I^f_i |w||v|$,
- $V^f(p) = \{|w| \text{ entonces } \mathfrak{M}, w \models p\},\ \forall\ p \in \Sigma$.

Llamaremos a $\mathfrak{M}^f_\Sigma$ *filtración* de $\mathfrak{M}$ a través de Σ.

■

A través de la técnica de filtrado construimos un modelo finito para un modelo $\mathfrak{M}$ lo cual verifica la propiedad de modelo finito para la lógica **N**. Es fácil extraer de esta definición la propiedad de modelo finito [[BdRV01], Teorema 2.41] y debido a esto el problema de satisfactibilidad en nuestra teoría es decidible. Podemos ahora hacer una estimación del tiempo de ejecución para el problema de satisfactibilidad de una fórmula φ perteneciente a la lógica **N**, un modelo de a lo sumo tamaño $2^{|\varphi|}$ puede ser construido para φ [BdRV01, DVDK07]. Podemos decir entonces un chequeador de modelos para la lógica **N** tendrá intrínsecamente una complejidad temporal del orden *exponencial.*

Esto nos lleva a enfrentar el tema del diseño de chequeadores de modelos.

4.5. Chequeo de Modelos

Un model checker es un programa que resuelve el problema del chequeo de modelos. El problema global de chequeo de modelos para **N**(Does) consiste en chequear si, dada una *fbf* φ, y dado un modelo $\mathfrak{M}$ para **N**(Does);

existe un $w \in W$ tal que $\mathfrak{M}, w \models \varphi$.

Como base para la construcción de nuestro chequeador de modelos utilizaremos la construcción propuesta de manera general en [FMDR04]. Sea φ una fórmula y $\mathrm{MM}\mathscr{L}_{\mathrm{Does}}(\varphi)$ el conjunto de *sub-fórmulas monolíticas maximales de* φ pertenecientes a $\mathscr{L}_{\mathrm{Does}}$. Sea φ' la **N**-fórmula obtenida reemplazando cada sub-fórmula $\alpha \in \mathrm{MM}\mathscr{L}_{\mathrm{Does}}(\varphi)$ por una nueva letra proposicional p_α.

Estos son los pseudo-códigos de los model-checkers para **N**(Does):

Function $MC_{\mathbf{N}(\mathrm{Does})}((A, W, B_i, G_i, I_i, V', \{d_i\}), \varphi)$
input: un modelo fibrado $\mathfrak{M}$ de estructura (4.1) y una fórmula $\varphi \in \mathscr{L}_{\mathbf{N}(\mathrm{Does})}$
computar $\mathrm{MM}\mathscr{L}_{\mathrm{Does}}(\varphi)$
for every $\alpha \in \mathrm{MM}\mathscr{L}_{\mathrm{Does}}(\varphi)$
$i :=$ *identificar el agente involucrado en* α
for every $w \in W$
if $(MC_{\mathrm{Does}}(d_i(w), \alpha) = true)$ *then*
$V'(w) := V'(w) \bigcup \{p_\alpha\}$ /*fuzzling*/

construir φ' /* *sistemáticamente reemplazamos las variables generadas* */
return $MC_{\mathbf{N}}((A, W, B_i, G_i, I_i, V', \{d_i\}), \varphi')$;/**llama al chequeador normal con* φ'*/

Function $MC_{\mathrm{Does}}(d_i(w), \alpha)$
input: un modelo de Scott-Montague de estructura (4.2),
y una sub-fórmula monolítica maximal α.
while quedan neighbourhoods sin chequear en $d_i(w)$
$n_k =$ *set* $n_i \in d_i(w)$ /*n_k *itera sobre el conjunto de neighbourhoods**/
for every $w \in n_k$
if $\alpha \notin \mathrm{v}(w)$ *then return false*
return true

FunctionMC$_{\mathbf{N}}((A, W, B_i, G_i, I_i, V', d_i), \varphi')$
input: un modelo $\mathfrak{M}$ de estructura (4.1) y una fórmula φ'
for every $w \in W$
if check$((A, w, B_i, G_i, I_i, V'), \varphi')$
return w
returnfalse

Function $check((A, w, B_i, G_i, I_i, V'), \alpha)$
  case on the form of α
    $\alpha = p_{\alpha'}$:
      if $p_{\alpha'} \notin V'(w)$
        return false
    $\alpha = \neg\alpha'$:
      if $check((A, w, B_i, G_i, I_i, V'), \alpha')$
        return false
    $\alpha = \alpha_1 \wedge \alpha_2$:
      if not $check((A, w, B_i, G_i, I_i, V'), \alpha_1)$ *or*
      or not $check((A, w, B_i, G_i, I_i, V'), \alpha_2)$
        return false
    $\alpha = \alpha_1 \vee \alpha_2$:
      if not $check((A, w, B_i, G_i, I_i, V'), \alpha_1)$ *and*
      and not $check((A, w, B_i, G_i, I_i, V'), \alpha_2)$
        return false
    $\alpha = BEL_i(\alpha')$:
      for each v *such that* wB_iv
        if not $check((A, v, B_i, G_i, I_i, V'), \alpha')$
          return false
    $\alpha = GOAL_i(\alpha')$:
      for each v *such that* wG_iv
        if not $check((A, v, B_i, G_i, I_i, V'), \alpha')$
          return false
    $\alpha = INT_i(\alpha')$:
      for each v *such that* wI_iv
        if not $check((A, v, B_i, G_i, I_i, V'), \alpha')$
          return false
    others : *return false*
  return true

Estos procedimientos actúan de la siguiente manera. Dado un modelo fibrado y una fórmula φ, el chequeador $MC_{\mathbf{N}(\mathrm{Does})}$ primero computa el conjunto $\mathrm{MM}\mathscr{L}_{\mathrm{Does}}(\varphi)$ de sub-fórmulas monolíticas maximales de φ. Para cada una de éstas, el chequeador identifica cuál es el agente que está llevando a cabo la acción. Luego, establece los mundos donde esta acción es llevada a cabo satisfactoriamente. Luego, MC_{Does} es invocado con un modelo de Scott-Montague $d_i(w)$ como parámetro (recordemos que d_i tiene estructura η, [HG73]).

MC_{Does} es pseudo-código para la función de valuación v: testea si existe

un neighborhood n_i de w donde α es verdadera. De ser así, una nueva letra proposicional p_α es agregada a $V'(w)$ para registrar el éxito del actuar, esto es, si la acción es exitosa (es decir, $DOES_i$ es verdadera) agregamos una variable proposicional (con valor verdadero). Finalmente, antes de llamar al chequeador normal $MC_{\mathbf{N}}$, la nueva fórmula φ' es construida sin las modalidades $DOES$: éstas han sido reemplazadas en la etapa de fuzzling.

El chequeador $MC_{\mathbf{N}}$ es pseudo-código de la función de valuación de la lógica **N** evaluándose recursivamente sobre las sub-fórmulas $\alpha \in \varphi$ mediante la función *check*.

4.6. Complejidad de los Chequeadores

Nos concentramos en el análisis de complejidad computacional de los chequeadores propuestos en la sección anterior. Como ya fue dicho, la complejidad de la lógica **N** es de orden exponencial. Para este análisis nos inspiramos en el estudio de complejidad de los chequeadores de modelos en [FMDR04].

Una instancia del problema de chequeo de modelos tiene dos componentes: un modelo $\mathfrak{M}$ de estructura (4.1) y una fórmula φ. En nuestro análisis, consideramos tres parámetros: n la cardinalidad de W, m la suma de las cardinalidades de las relaciones B_i, G_i, I_i y k la longitud de φ (digamos el número de letras proposicionales en φ). Dado $w \in W$, las operaciones más utilizadas sobre la valuación $V'(w)$ son chequear si una fórmula α pertenece a $V'(w)$, y luego agregar α a $V'(w)$. Ambas operaciones pueden ser eficientemente implementadas en tiempo constante representando a V' como un arreglo de bits bidimensional de tamaño $n \times k$ [CES86].

Primero anotamos $C_{\mathbf{N}(\mathrm{Does})}(\cdot,\cdot,\cdot)$ (resp. $C_{\mathrm{Does}}(\cdot,\cdot)$, $C_{\mathbf{N}}(\cdot,\cdot,\cdot)$) la función de complejidad para el chequeador de modelos $MC_{\mathbf{N}(Does)}$ (resp. MC_{Does}, $MC_{\mathbf{N}}$), recordemos que son varios módulos chequeadores (Sección 4.5). Debemos notar que $C_{\mathrm{Does}}(\cdot,\cdot)$ tiene dos parámetros: el tamaño del modelo y la longitud de la fórmula.

Recordemos la notación que utilizaremos, $\mathcal{O}(n)$ significa cota superior asintótica, $\Theta(n)$ es la cota ajustada asintótica y $o(n)$ es una cota más "real".

Teorema 1 Complejidad de $MC_{N(Does)}$. Sea $\mathfrak{M}$ un modelo con estructura como en (4.1), un **N**(Does)-*modelo* finito y φ una **N**(Does)-*fórmula*. La

complejidad de $MC_{\mathbf{N}(\text{Does})}$ con entrada $\mathfrak{M}$ y φ es:

$$\mathcal{O}(n) \cdot [\mathcal{O}(k) \cdot C_{\text{Does}}(N, \mathcal{O}(1)) + \mathcal{O}(1) \cdot C_{\text{Does}}(N, \mathcal{O}(k))] + C_{\mathbf{N}}(n, m, \mathcal{O}(k)),$$

donde $n = |W|, m = \sum_{\text{R}\in(B_i,G_i,I_i)} |\text{R}|$, $k = |\varphi|$ *y* $N = max_{w\in W}|d_i(w)|$.

Prueba: El conjunto de fórmulas monolíticas $\text{MM}\mathcal{L}_{\text{Does}}(\varphi)$ y la construcción de la fórmula φ' puede ser computada de una pasada sobre φ (con un parseo simple), entonces su respectivo costo es de $\mathcal{O}(k)$.

Los ciclos pertenecientes a la sentencia **for** tienen el siguiente costo:

$$\sum_{\alpha\in \text{MM}\mathcal{L}_{\text{Does}}(\varphi)} \sum_{w\in W} C_{\text{Does}}(|g(w)|, |\alpha|) \leq n \cdot \sum_{\alpha\in \text{MM}\mathcal{L}_{\text{Does}}(\varphi)} C_{\text{Does}}(N, |\alpha|).$$

Para acotar la última sumatoria, hay que notar que el conjunto $\text{MM}\mathcal{L}_{\text{Does}}(\varphi)$ contiene solo sub-fórmulas de φ y su cardinalidad es $o(k)$. Dado que un conjunto de cardinalidad n puede ser particionado en $\Theta(n)$ conjuntos de cardinalidad $\Theta(1)$ o en $\Theta(1)$ conjuntos de cardinalidad $\Theta(n)$, la sumatoria anterior es

$$\mathcal{O}(n) \cdot [\mathcal{O}(k) \cdot C_{\text{Does}}(N, \mathcal{O}(1)) + \mathcal{O}(1) \cdot C_{\text{Does}}(N, \mathcal{O}(k))].$$

Finalmente, como la longitud de φ' es de orden $\mathcal{O}(k)$, la única llamada a $MC_{\mathbf{N}}$ tiene costo $C_{\mathbf{N}}(n, m, \mathcal{O}(k))$. Sumando las componentes, el costo total es

$$\mathcal{O}(n) \cdot [\mathcal{O}(k) \cdot C_{\text{Does}}(N, \mathcal{O}(1)) + \mathcal{O}(1) C_{\text{Does}}(N, \mathcal{O}(k))] + C_{\mathbf{N}}(n, m, \mathcal{O}(k)).$$

■

El costo del overhead generado por la comunicación requerida para computar el conjunto $\text{MM}\mathcal{L}_{\text{Does}}(\varphi)$ y la fórmula φ' es de $\mathcal{O}(k)$ y es inferior al costo del chequeo del modelo.

Ahora nos concentramos en el análisis de complejidad para los módulos chequeadores $C_{\mathbf{N}}$ y C_{Does}.

Lema 1 Complejidad de $MC_{\mathbf{N}}$ ($C_{\mathbf{N}}$). Dijimos que la complejidad de la lógica **N** es al menos EXPTIME. Un análisis un poco más detallado sobre el problema arroja dos resultados importantes: (i) La complejidad de la satisfactibilidad de la lógica **N** es EXPTIME-complete, esta prueba puede ser obtenida realizando una reducción de este problema al juego de *"tiling"*[2] instanciado para dos personas (ver el detalle en [DVDK07]). La prueba está inspirada en la provista en [BdRV01] (Capítulo 3, Sección 6.8) para probar satisfactibilidad de la teoría lógica **PDL**; (ii) Con una correcta restricción de cantidad de variables proposicionales y restringiendo la profundidad (cantidad de operadores anidados) de las fórmulas, estas instancias reducidas del problema de satisfactibilidad para la lógica **N** puede ser resuelto en un tiempo menor al exponencial.

Ahora analizaremos el tiempo computacional que insume nuestro chequeador de modelos para **N**:

Sea $\mathfrak{M}$ un modelo con estructura (4.1) y φ una fórmula ($\in \mathscr{L}_{\mathbf{N}}$). La complejidad de $MC_{\mathbf{N}}$ con entrada $\mathfrak{M}$ y φ es:

$$C_{\mathbf{N}}(n, m, k) \in \mathcal{O}(n \cdot m^k)$$

donde $n = |W|, m = \sum_{\mathrm{R} \in (B_i, G_i, I_i)} |\mathrm{R}|,\ k = |\varphi|$.

Prueba. El costo del **for** principal es de:

$$\sum_{w \in W} check(\mathfrak{M}, \varphi') \in \mathcal{O}(n \cdot \mathcal{O}(check(\cdot, \cdot))).$$

El análisis de la complejidad temporal de la función $check(\cdot, \cdot)$ es:

$$T(k) = \begin{cases} \mathcal{O}(1) & \text{si } \alpha \text{ es proposicional} \quad \in \mathcal{O}(1) \\ 2T(k-1) & \text{si } \alpha \neq R(\alpha') \quad \in \mathcal{O}(2^k) \\ mT(k-1) & \text{si } \alpha = R(\alpha') \quad \in \mathcal{O}(m^k) \end{cases}$$

Finalmente, el costo en el peor caso de la función $T(k) \in \mathcal{O}(m^k)$. Sumando todas los componentes de la prueba, el costo total del checkeador de modelos para la lógica **N** es del orden $\mathcal{O}(n \cdot m^k)$.

■

[2]El juego de tiling se basa en organizar cuadrados (tiles) de 1 × 1 los cuales tienen una orientación predefinida y sus bordes tiene colores asignados, hasta que un jugador complete su área.

Lema 2 Complejidad de MC_{Does} (C_{Does}). Los últimos resultados sobre la decibilidad la lógica de la acción (Does) acotan su complejidad computacional a la clase de problemas PSPACE [SP08] (este artículo trata de manera especial la lógica del Does y su complejidad, Sección 4). Además se conjetura que es PSPACE-hard y PSPACE-complete [SP08]. Como la complejidad del espacio requerido del problema esta acotado a PSPACE podemos afirmar que la complejidad temporal estará acotada a la clase NP.

Ahora analizaremos el tiempo computacional que insume nuestro chequeador de modelos para Does:

Sea η un modelo con estructura (4.2) y α una fórmula ($\in \text{MM}\mathscr{L}_{\text{Does}}(\varphi)$). La complejidad de MC_{Does} con entrada η y α es:

$$C_{\text{Does}}(N, k) \in \mathcal{O}(n \cdot N)$$

donde $n = |W|$, $k = |\alpha|$ *y* $N = max_{w \in W}|d_i(w)|$.

Prueba. El costo asociado al **while** y **for** es:

$$\sum_{n_i \in d_i(w)} \sum_{w \in n_i} \mathcal{O}(1)$$

Como α es una sub-fórmula maximal monolítica su cardinalidad $\in \mathcal{O}(1)$. La segunda sumatoria está acotada a la cantidad de mundos posibles en un *neighborhood* en particular, la cual es n. La primera está acotada a la cantidad de *neighborhoods* posibles para cada w, esto es N en el peor de los casos $N = Pow(W) = 2^n$.

Finalmente juntando estos dos resultados el costo del chequeador de modelos para la lógica de la acción es del orden $\mathcal{O}(n \cdot N)$

■

Capítulo 5

Lógicas Unidas

5.1. Lógicas Unidas

En los capítulos anteriores hemos demostrado la completitud de las lógicas **N** y Does. Las modalidades $DOES_i$ son siempre aplicadas a constantes proposicionales que representan acciones unitarias, por ejemplo, $DOES_x\ Paga$ significa, "el agente x paga". Los operadores normales interactúan con las modalidades $DOES$ de una manera restringida: las acciones de los agentes aparecen en la parte más interna de las *fbfs*, como en el ejemplo anterior. Esto significa que ninguna otra modalidad puede ocurrir dentro del alcance de un $DOES$; no es posible escribir fórmulas del tipo $DOES_i(DOES_j\ Paga)$ ("el agente i hace pagar al agente j"). Por ejemplo $DOES_i(GOAL_j\ \mathscr{A})$ es un tipo de fórmula en la que una modalidad normal aparece dentro del alcance del $DOES$. Esta no es una fórmula del sistema (tal como fue presentado hasta ahora).

Nuestra meta ahora es obtener un sistema donde podamos escribir y probar la validez de fórmulas arbitrarias compuestas por modalidades cognitivas y de la acción sin las restricciones mencionadas. Para hacer esto retornamos al análisis de $\mathfrak{F}$, considerándolo una vez más como dividido en dos estructuras: una que contiene las lógicas normales y otra las lógicas de la acción. Nuevamente tenemos dos "redes" de relaciones que se superponen sobre el mismo conjunto W (una de ellas con modelos de Kripke, las ontologías cognitivas; los objetivos (goals), creencias (beliefs), e intenciones (intentions); la segunda es un conjunto de estructuras de Scott-Montague que captura el comportamiento externo de los agentes).

Iremos directo a la combinación. Primero, duplicamos y sub-indicamos los

elementos de W para tener un conjunto de situaciones W_N y otro conjunto W_D. Seguidamente construimos una ontología $W_N \times W_D$ de pares (w_N, w_D) representando intuitivamente el par *configuración mental - conducta visible*. **N** es la parte normal de la lógica y Does es la parte no normal. Una "unión" de estas lógicas, anotamos **N** × Does, tiene la siguiente forma:

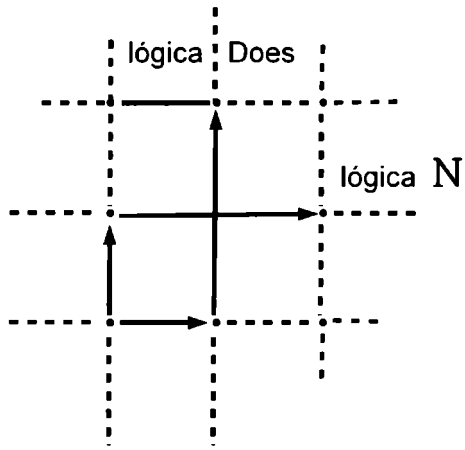


Figura 5.1: Estructura de la unión

A continuación damos la sintaxis y semántica de la unión propuesta.

Unión. Sintaxis. Sea $\mathscr{L}_{\mathbf{N}}$ el lenguaje de **N** (la lógica base restringida a los operadores normales), y $\mathscr{L}_{\text{Does}}$ el lenguaje de la lógica de acción. El lenguaje $\mathscr{L}_{\mathbf{N}\times\text{Does}}$ es obtenido de la unión de las reglas de composición de $\mathscr{L}_{\mathbf{N}}$ y $\mathscr{L}_{\text{Does}}$. Distinto de lo que pasaba con $\mathscr{L}_{\mathbf{N}(\text{Does})}$, las *fbfs* $DOES_i(GOAL_j\ \mathscr{A})$ y $GOAL_j(DOES_i\ \mathscr{A})$ son fórmulas de $\mathscr{L}_{\mathbf{N}\times\text{Does}}$.

Unión. Semántica. Asumimos dos estructuras: $\langle A,\ W_N,\ \{B_i\},\ \{G_i\},\ \{I_i\},\ V'\rangle$ y $\langle A, W_D, \{D_i\}, \mathrm{v}\rangle$. En ellas testeamos la validez de las modalidades normales y no normales ($DOES$), respectivamente. La primer estructura es un modelo de Kripke; y la otra es un modelo de Scott-Montague. Interpretamos las fórmulas pertenecientes a $\mathscr{L}_{\mathbf{N}\times\text{Does}}$ sobre un modelo con la siguiente estructura:

$$\langle A, W_N \times W_D, \{B_i\}_{i\in\Lambda}, \{G_i\}_{i\in\Lambda}, \{I_i\}_{i\in\Lambda}, \{D_i\}_{i\in\Lambda}, \mathrm{V}\rangle, \tag{5.1}$$

donde:

- A es el conjunto de agentes;
- $W_N \times W_D$ es un conjunto de pares de situaciones;
- $\{B_i\}_{i\in A}, \{G_i\}_{i\in A}, \{I_i\}_{i\in A}$ son las relaciones de accesibilidad para los operadores normales (con semántica como en la Sección 4.2);
- $\{D_i\}_{i\in A}$ son las relaciones de accesibilidad para el operador de acción; y
- $\mathtt{V} : W_N \times W_D \rightarrow Pow(P)$ es una función que asigna a cada par en $W_N \times W_D$ el conjunto de letras proposicionales de P que son verdaderas.

La definición de satisfacción de una fórmula de $\mathscr{L}_{\mathbf{N}\times\mathbf{Does}}$ en un modelo $\mathfrak{C}$ con estructura como en (5.1) en un estado (w_N,w_D) es:

$$\mathfrak{C},(w_N,w_D) \models BEL_i\ \mathscr{A} \textit{ sii } \forall v_N \in W_N(\textit{ if } w_N B v_N \textit{ entonces } \mathfrak{C},(v_N,w_D) \models \mathscr{A}).$$
$$\mathfrak{C},(w_N,w_D) \models GOAL_i\ \mathscr{A} \textit{ sii } \forall v_N \in W_N(\textit{ if } w_N G v_N \textit{ entonces } \mathfrak{C},(v_N,w_D) \models \mathscr{A}).$$
$$\mathfrak{C},(w_N,w_D) \models INT_i\ \mathscr{A} \textit{ sii } \forall v_N \in W_N(\textit{ if } w_N I v_N \textit{ entonces } \mathfrak{C},(v_N,w_D) \models \mathscr{A}).$$
$$\mathfrak{C},(w_N,w_D) \models DOES_i\ \mathscr{A} \textit{ sii existe un vecindario } n \textit{ de } w_D \textit{ tal que } \forall v \in n\ (\mathfrak{C},(w_N,v) \models \mathscr{A}).$$

Los operadores normales se mueven a través de la primer componente (w_N), y los no normales sobre la segunda componente del mundo en el que nos encontramos (w_D).

Ejemplo 1 Persuasión. La fórmula $DOES_i(GOAL_j\ \mathscr{A})$ puede ser vista como una forma de representar persuasión, su significado pretendido es "el agente i hace que el agente j tenga como objetivo $\mathscr{A}$". ¿Cómo probamos la validez de esta fórmula en un mundo (w_N,w_D)? Los movimientos dentro del multi-grafo están determinados por $\mathfrak{C},(w_N,w_D) \models DOES_i\ (GOAL_j\ \mathscr{A})$ sii $\exists$ vecindario n_i de w_D tal que $\forall v_k \in n_i\ (\mathfrak{C},(w_N,v_k) \models GOAL_j\ \mathscr{A})$, que nos permite evaluar $\forall v_k \in n_i($ iff $\forall u_N \in W_N($ if $w_N G_j u_N$ entonces $\mathfrak{C},(u_N,v_k) \models \mathscr{A}))$.

5.2. Chequeo de Modelos

El problema global de chequeo de modelos (como lo definimos en la Sección 4.5) para $\mathbf{N} \times$ Does consiste en chequear si, dada una *fbf* φ, y dado un modelo $\mathfrak{C}$ para $\mathbf{N} \times$ Does; existe un $w \in W$ tal que $\mathfrak{C}, w \models \varphi$.

Nuevamente como base para la construcción de nuestro chequeador de modelos utilizaremos la construcción propuesta de manera general en [FMDR04]. Este es el pseudo-códigos del model-checker para $\mathbf{N} \times$ Does:

Function $MC_{\mathbf{N}\times \mathrm{Does}}((A, W, B_i, G_i, I_i, \{D_i\}, V), \varphi)$
input: un modelo unido de estructura (5.1) y una fórmula $\varphi \in \mathscr{L}_{\mathbf{N}\times \mathrm{Does}}$
for every pair $(w_N, w_D) \in W_N \times W_D$
if $check((A, W_N \times W_D, B_i, G_i, I_i, \{D_i\}, \mathtt{V}), \varphi, (w_N, w_D))$ *then*
return (w_N, w_D)
return false

Function $check((A, W_N \times W_D, B_i, G_i, I_i, \{D_i\}, \mathtt{V}), \varphi, (w_N, w_D))$
case on the form of α
$\alpha = p_{\alpha'}$:
if $p_{\alpha'} \notin (w_N, w_D)$
return false
$\alpha = \neg\alpha'$:
if $check((A, W_N \times W_D, B_i, G_i, I_i, \{D_i\}, \mathtt{V}), \alpha', (w_N, w_D))$
return false
$\alpha = \alpha_1 \wedge \alpha_2$:
if not $check((A, W_N \times W_D, B_i, G_i, I_i, \{D_i\}, \mathtt{V}), \alpha_1, (w_N, w_D))$ *or*
or not $check((A, W_N \times W_D, B_i, G_i, I_i, \{D_i\}, \mathtt{V}), \alpha_2, (w_N, w_D))$
return false
$\alpha = \alpha_1 \vee \alpha_2$:
if not $check((A, W_N \times W_D, B_i, G_i, I_i, \{D_i\}, \mathtt{V}), \alpha_1, (w_N, w_D))$ *and*
and not $check((A, W_N \times W_D, B_i, G_i, I_i, \{D_i\}, \mathtt{V}), \alpha_2, (w_N, w_D))$
return false
$\alpha = BEL_i(\alpha')$:
for each v_N *such that* $w_N B_i v_N$
if not $check((A, W_N \times W_D, B_i, G_i, I_i, \{D_i\}, \mathtt{V}), \alpha', (v_N, w_D))$
return false
$\alpha = GOAL_i(\alpha')$:
for each v_N *such that* $w_N G_i v_N$
if not $check((A, W_N \times W_D, B_i, G_i, I_i, \{D_i\}, \mathtt{V}), \alpha', (v_N, w_D))$

```
                    return false
    α = INT_i(α') :
        for each v_N such that w_N I_i v_N
            if not check((A, W_N × W_D, B_i, G_i, I_i, {D_i}, V), α', (v_N, w_D))
                return false
    α = DOES_i(α') :
        while there are unchecked n_k neighbourhoods of w_D
            set i = 1..n_k
            for every v_D ∈ n_i
                if not check((A, W_N × W_D, B_i, G_i, I_i, {D_i}, V), α', (w_N, v_D))
                    return false
    others : return false
return true
```

Estos procedimientos actúan de la siguiente manera. Dado un modelo $\mathfrak{C}$ de estructura (5.1) y una fórmula φ, el chequeador $MC_{\mathbf{N}\times\mathbf{Does}}$ llama a la $check(\cdot,\cdot,\cdot)$ para cada par de mundos (w_N, w_D) (hasta que φ sea verdadera en alguno o alguna sub-fórmula de φ falle).

La función $check(\cdot,\cdot,\cdot)$ es el pseudo-código para la función de valuación para los operadores modales (tanto normales como no normales) y actúa recursivamente sobre la estructura de φ.

En el tratamiento de la decibilidad del fibrado solo bastó con probar la propiedad deseada en las lógicas que componen el fibrado. En el caso de la unión no es trivial y conlleva un análisis diferente. Lo dejamos fuera del alcance de esta tesis [FMDR04, FG94] y probablemente integre nuestros estudios futuros sobre este trabajo. Como ejemplo de la incertidumbre generada por la unión de dos lógicas tenemos el caso de **PLTL** (Propositional Linear Temporal Logic, lógica proposicional temporal y lineal) que es decidible. El fibrado de **PLTL** con ella misma **PLTL(PLTL)** también es decidible, pero su unión **PLTL** × **PLTL** ni siquiera es recursivamente numerable [Wol99] (no existe algoritmo que resuelva el problema de chequeo de modelos o satisfactibilidad para esta unión).

Para asegurar la decibilidad de **N** × Does podemos intentar probar que tiene la propiedad de modelo finito (*fmp*, para cada modelo siempre podemos encontrar un modelo finito equivalente). En caso de no encontrar la propiedad, bastará con hacer finitos a todos los conjuntos que componen el modelo (W, B_i, G_i, I_i, D_i y V).

El aumento de expresividad que resulta de aplicar el método de combinación previo viene acompañado por un mayor costo computacional y mayor profundidad de análisis. Como ejemplo de esto podemos citar algunas uniones realizadas entre lógicas como **S5** (**S5** está caracterizada por los modelos donde la relación de accesibilidad es: reflexiva, transitiva y simétrica). La complejidad computacional de $\mathbf{S5} \in$ NP-Complete y para la unión $\mathbf{S5} \times \mathbf{S5}$ esta complejidad escala a la clase de problemas NEXPTIME-Complete [FG94] (aún más, $\mathbf{S5} \times \mathbf{S5} \times \mathbf{S5}$ es indecidible). En estos casos no solo se incrementa la complejidad de la lógica sino que también el análisis teórico se vuelve más complejo teniendo que recurrir a técnicas matemáticos complejas [SP08] (coalgebras, categorías, etc).

Las técnicas requeridas para realizar estos análisis no son aplicadas a este pseudo-código en esta tesis debido a su extensión y dificultad extra.

Capítulo 6

Implementación

6.1. Programación lógica

La programación lógica [Llo84] es un tipo de paradigma de programación dentro del paradigma de programación declarativa. Comúnmente es utilizada en aplicaciones relacionadas a la inteligencia artificial, sistemas expertos, demostración automática de teoremas, procesamiento de lenguaje natural, entre otros. El estilo de la programación lógica es muy distinto del utilizado en lenguajes convencionales. En los lenguajes tradicionales, la programación consiste en indicar cómo resolver un problema mediante sentencias mientras que en la programación lógica se trabaja de forma descriptiva estableciendo relaciones entre objetos o entidades indicando qué hacer en vez de cómo.

La mayoría de los lenguajes de programación lógica se basan en la teoría lógica de primer orden [Ham78]. Un programa lógico está compuesto por un conjunto finito de cláusulas de programa[Llo84]. Estas cláusulas tienen una forma determinada y se denominan cláusulas de Horn [Llo84]. Las cláusulas de Horn son una subclase de cláusulas (disyunción de literales donde cada literal es un átomo o la negación de un átomo) que tienen como máximo un literal positivo. De esta manera, estas cláusulas se corresponden con fórmulas de la forma $A \leftarrow B_1, ..., B_n$ donde el consecuente A (que contiene un solo átomo) se denomina cabeza de cláusula y el antecedente $B_1, ..., B_n$ se denomina cuerpo de cláusula. Esta notación (notación clausal) asume que todas las variables están cuantificadas universalmente y que la comas en el antecedente $B_1, ..., B_n$ representan conjunciones. Las cláusulas de la forma $A \leftarrow$ (sin cuerpo) se denominan cláusulas incondicionales. Las cláusulas de la forma $\leftarrow B_1, ..., B_n$ (sin cabeza) se denominan cláusulas objetivo (o consultas).

Uno de los lenguajes más representativos de la programación lógica y muy utilizado en el área es PROLOG [CM03]. Una de las grandes ventajas de este lenguaje es su potencia y facilidad para describir soluciones de problemas con pocas reglas (forma declarativa). Las estructuras de listas dinámicas y los mecanismos de recursividad y "backtracking" son uno de los aspectos que le dan gran poderío y flexibilidad.

En este capítulo utilizamos las herramientas del lenguaje de programación PROLOG para implementar las bases de la teoría testigo y el fibrado desarrollado en capítulos anteriores.

6.2. Definiciones básicas sobre Prolog

El lenguaje PROLOG (o Prolog)[CM03], es un lenguaje de programación lógica interpretado que se enmarca en el paradigma de los lenguajes declarativos. La programación en PROLOG se basa en la descripción de hechos y relaciones entre objetos asociados a un problema en lugar de describir los pasos para resolver dicho problema.

PROLOG utiliza una base de conocimientos en forma de reglas de inferencia deductivas (cláusulas de Horn). Como lenguaje inspirado en el cálculo de predicados se basa en los siguientes conceptos:

- Un conjunto de axiomas o hechos sobre objetos y sus relaciones.
- Reglas de inferencias sobre objetos y sus relaciones, que se resuelven por Resolución y Unificación [Llo84].
- Objetivos a demostrar o consultas sobre objetos y sus relaciones (condiciones a unificar con las reglas).

Un programa PROLOG consiste en un conjunto de cláusulas donde cada cláusula es un *hecho* sobre la información dada o una regla que describe como inferir la solución.

6.3. Programación de lógica modal

La programación de lógica modal es un campo que extiende la programación lógica clásica con el fin de poder manejar "modalidades". En

[Ngu09, FndC86, Ngu04] se exploran algunos de los trabajos más relevantes en esta área. Actualmente dos enfoques principales para llevar a cabo la implementación de sistemas de programación de lógica modal son el enfoque directo y el enfoque traslacional. El primer enfoque trabaja directamente con modalidades mientras que el segundo traduce programas de lógica modal en programas lógicos clásicos.

Para enfrentar la implementación del fibrado motivamos nuestro desarrollo en los trabajos de [FndC86] y [Ngu04]. En [FndC86] se construye el sistema *Molog*; una extensión de la programación lógica convencional que permite trabajar con modalidades. *Molog* puede ser visto como un "framework"[1] instanciable con lógicas modales particulares. En [Ngu04] se desarrolla un sistema de programación de lógica modal llamado *MProlog*. *MProlog* es una extensión para PROLOG (desarrollada en PROLOG) que permite trabajar con modalidades.

Tomamos los conceptos de implementaciones de sistemas de lógica modal mencionados anteriormente y utilizamos PROLOG para implementar:

- los operadores modales de nuestra teoría testigo,
- los mecanismos para evaluar fórmulas sobre un modelo y,
- un chequeador de modelos para el fibrado.

El entorno utilizado para el desarrollo es *SWI-Prolog* [2]. *SWI-Prolog* es un producto de libre distribución y corre sobre una amplia variedad de sistemas operativos.

6.4. Implementación del fibrado

6.4.1. Representación de fórmulas

Definimos inductivamente el conjunto de fórmulas que vamos a implementar de la siguiente manera:

- Un hecho (proposición atómica) es una fórmula.
- Si F_1 y F_2 son fórmulas, también lo son $F_1 \wedge F_2$, $F_1 \vee F_2$, $F_1 \rightarrow F_2$, $F_1 \leftrightarrow F_2$ y $\neg F_1$.

[1] En el sentido de la programación orientada a objetos

[2] http://www.swi-prolog.org/

- Si F es una fórmula, A un conjunto finito de agentes, i un agente tal que $i \in A$ y G un subconjunto de agentes tal que $G \subseteq A$, entonces las siguientes modalidades también son fórmulas.
  - Modalidades epistémicas: $BEL(i, F)$, $E\text{-}BEL_G(F)$, $C\text{-}BEL_G(F)$.
  - Modalidades motivacionales: $GOAL(i, F)$, $INT(i, F)$, $E\text{-}INT_G(F)$, $M\text{-}INT_G(F)$, $M\text{-}INT_G(F)$.
  - Modalidades de obligación (la llamaremos OBL en vez de O): $OBL(i, F)$.
  - Modalidades *no-normales* de acción: $DOES(i, F)$.

Para implementar las fórmulas definidas anteriormente utilizamos la estructura de datos "lista"que provee PROLOG.

Para describir hechos (proposiciones que son verdaderas en algún mundo del modelo), utilizamos listas de un único elemento. Cada elemento se corresponde con una "constante" que representa la proposición.

- $[a]$ representa el hecho a.

Las fórmulas booleanas que contienen operadores booleanos unarios (negación) se representan utilizando listas de dos elementos. El primer elemento representa el operador unario y el segundo elemento representa la fórmula a la cual se aplica dicho operador.

- $[neg, F]$ representa la negación de la fórmula F.

Las fórmulas booleanas que contienen operadores booleanos binarios se representan utilizando listas de tres elementos. El primer elemento representa el primer operando, el segundo elemento representa un operador booleano binario (conjunción, disyunción, implicación o equivalencia) y el tercer elemento representa el segundo operando. Construimos las fórmulas "booleanas"básicas de la siguiente forma:

- $[-, F]$ representa la negación de la fórmula F.
- $[F_1, \&, F_2]$ representa la conjunción entre las fórmulas F_1 y F_2.
- $[F_1, v, F_2]$ representa la disyunción entre las fórmulas F_1 y F_2.
- $[F_1, =>, F_2]$ representa la implicación entre las fórmulas F_1 y F_2.

- $[F_1, <=>, F_2]$ representa la equivalencia lógica entre las fórmulas F_1 y F_2.

Las fórmulas modales normales se representan utilizando listas de tres elementos. El primer elemento representa el nombre de la modalidad, el segundo elemento representa al agente involucrado y el tercer elemento representa una fórmula (con la forma definida previamente). De esta manera, sea i un agente y F una fórmula, representamos las fórmulas modales individuales de la siguiente manera:

- $[bel, i, F]$, representa el operador BEL (El agente i cree F).
- $[int, i, F]$, representa el operador INT (El agente i tiene la intención F).
- $[goal, i, F]$, representa el operador $GOAL$ (El agente i tiene el objetivo F).
- $[obl, i, F]$, representa el operador O (El agente i tiene la obligación F).

De manera similar se representa el operador *no normal* Does:

- $[does, i, F]$, representa el operador $DOES$ (El agente i realiza F).

Los operadores modales normales colectivos se representan de la siguiente manera:

- $[e\text{-}bel, G, F]$, representa el operador $E\text{-}BEL_G$.
- $[e\text{-}int, G, F]$, representa el operador $E\text{-}INT_G$.
- $[c\text{-}bel, G, F]$, representa el operador $C\text{-}BEL_G$.
- $[c\text{-}int, G, F]$, representa el operador $C\text{-}INT_G$.
- $[m\text{-}int, G, F]$, representa el operador $M\text{-}INT_G$.

6.4.2. Representación de frames y modelos

Representamos los agentes, mundos y hechos utilizando constantes en PROLOG de la siguiente manera:

- Para describir agentes utilizamos constantes a1,a2,a3,...,an. Cada constante representa un único agente en el sistema. Representan el conjunto A de la definición 6 (modelo multi-relacional) del capítulo 4.

- Para describir mundos o situaciones del frame utilizamos constantes w1,w2,w3,....,wn. Cada constante representa un único mundo en un frame. Representan el conjunto W de la definición 6 (modelo multi-relacional) del capítulo 4.

- Para describir hechos utilizamos listas de un único elemento [a],[b],[c]...etc. Cada elemento se corresponde con una constante que representa el hecho.

Definimos un frame indicando los mundos y las relaciones que lo componen. Para esto, creamos reglas en PROLOG que nos permiten definir relaciones entre mundos previamente declarados. Como nuestra teoría base dispone de tres tipos de relaciones para los operadores normales (bel, int y goal) y un tipo de relación para el operador no normal $DOES$, definimos cuatro reglas distintas.

Sea:

- A una variable PROLOG que representa un agente.

- W una variable PROLOG que representa un mundo del frame.

- WA una variable PROLOG que representa una lista de mundos adyacentes.

describimos las relaciones de la siguiente manera:

- $rel\_bel(A, W, WA)$. Representa una relación de accesibilidad B_i.

- $rel\_int(A, W, WA)$. Representa una relación de accesibilidad I_i.

- $rel\_goal(A, W, WA)$. Representa una relación de accesibilidad G_i.

- $rel\_does(A, W, WA)$. Representa una relación de accesibilidad D_i.

Para representar la función de valuación del modelo, utilizamos la regla

$$hecho(W, F, R)$$

donde W representa un mundo del modelo, F representa un "hecho" tal cual se definió anteriormente y R representa un valor numérico que define si F es verdadera (1) o falsa (0) en algún mundo w.

6.4.3. Evaluación de fórmulas

Para evaluar las fórmulas implementamos diferentes reglas de PROLOG que describen el comportamiento de los operadores booleanos utilizados y las modalidades de la teoría testigo. Sea:

- F,F1,F2,F3,...,Fn variables PROLOG que representan fórmulas.
- W una variable PROLOG que representa un estado del frame.
- G una variable PROLOG que representa una lista de agentes.
- R una variable PROLOG numérica que representa el resultado de la ejecución de la regla (1 si es verdadera 0 si es falsa).
- N una variable Prolog numérica que representa la profundidad de ejecución (inicialmente es 0) [3].

La regla general para evaluar fórmulas se representa de la siguiente manera:

- $eval\_formula(W, F, R, N)$. Evalúa la fórmula F en W y retorna el resultado en N.

Los conectivos booleanos utilizados se implementan mediante las siguientes reglas:

- $eval\_formula(W, [neg, F], R, N)$. Evalúa la negación de una fórmula F.
- $eval\_formula(W, [F1, \&, F2], R, N)$. Evalúa la conjunción entre las fórmulas F1 y F2.
- $eval\_formula(W, [F1, v, F2], R, N)$. Evalúa la disyunción entre las fórmulas F1 y F2.
- $eval\_formula(W, [F1, =>, F2], R, N)$. Evalúa la implicación entre las fórmulas F1 y F2.

El comportamiento de los operadores modales normales individuales se implementa mediante las siguientes reglas:

- $eval\_formula(W, [bel, A, F], R, N)$. Describe el comportamiento del operador BEL. $[bel, A, F]$ es una fórmula modal.

[3] Durante la ejecución de las reglas que evalúan fórmulas es utilizada para armar la identación del texto que describe la ejecución en cada paso

- $eval\_formula(W, [int, A, F], R, N)$. Describe el comportamiento del operador INT. $[int, A, F]$ es una fórmula modal.
- $eval\_formula(W, [goal, A, F], R, N)$. Describe el comportamiento del operador $GOAL$. $[goal, A, F]$ es una fórmula modal.

El comportamiento del operador modal *no normal* se implementa con:

- $eval\_formula(W, [does, A, F], R, N)$. Describe el comportamiento del operador $DOES$. $[does, A, F]$ es una fórmula modal. Esta regla representa la función de fibrado que permite trabajar solo sobre modelos que contienen relaciones de accesibilidad del tipo rel_does.

El comportamiento de los operadores modales colectivos generales se implementa mediante las siguientes reglas:

- $eval\_formula(W, [e\text{-}int, G, F], R, N)$. Describe el comportamiento del operador $E\text{-}INT_G$. $[e\text{-}int, G, F]$ es una fórmula modal.
- $eval\_formula(W, [e\text{-}bel, G, F], R, N)$. Describe el comportamiento del operador $E\text{-}BEL_G$. $[e\text{-}bel, G, F]$ es una fórmula modal.

6.4.4. Representación de axiomas

Los axiomas y reglas utilizados son implementados sobre la representación de fórmulas propuesta anteriormente. En [DVDK07] se encuentran detallados los axiomas para la parte normal de la teoría y en [Elg97] los de la parte no normal. Para cada caso, construimos las reglas PROLOG que describen el comportamiento de tales axiomas. En el apéndice A del presente documento pueden encontrarse las implementaciones correspondientes.

6.5. Implementación del chequeador de modelos

Construidas todas las reglas que permiten evaluar fórmulas sobre modelos y axiomas y reglas del fibrado implementamos un chequeador de modelos utilizando como ejemplo el modelo de la figura 6.1.

Para poder utilizar el chequeador desarrollado primero debemos construir y cargar un modelo. Para cargar un modelo debemos indicar:

- los agentes involucrados,
- los mundos del frame,

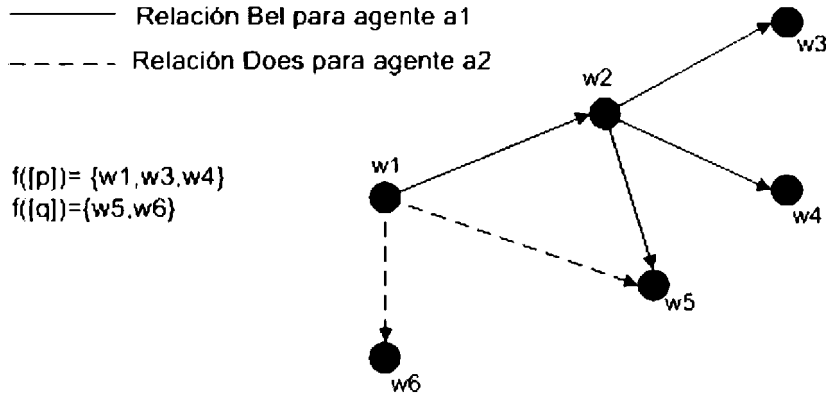


Figura 6.1: Ejemplo de modelo usando fibrado

- las relaciones entre los mundos definidos y,
- las reglas que determinan hechos verdaderos en cada mundo del frame (función de valuación del modelo).

El frame utilizado en el ejemplo contiene 6 mundos (w1,w2,w3,w4,w5,w6) y dos tipos de relaciones, *Bel* y *Does*. Utilizamos dos agentes que llamamos a1 y a2. Definimos la función de valuación f que determina hechos verdaderos en los mundos del frame, de la siguiente manera: f([a])={w1,w3,w4} y f([b])={w5,w6}.

A continuación describimos el código PROLOG para representar el modelo de la figura 6.1 con su correspondiente chequeador.

Definimos agentes.

- agentes([a1,a2]).

Definimos mundos del frame.

- mundos([w1,w2,w3,w4,w5,w6]).

Definimos relaciones del frame.

- rel_bel(a1,w1,[w2]).
- rel_bel(a1,w2,[w3,w4,w5]).
- rel_does(a2,w1,[w5,w6]).

Definimos la función de valuación (hechos verdaderos en cada w).

- hecho(w1,[p],1).
- hecho(w3,[p],1).
- hecho(w4,[p],1).
- hecho(w5,[q],1).
- hecho(w6,[q],1).

Definimos la regla que chequea una fórmula.

```
check_formula(F):-
   mundos(WS),
   model_checker(WS,F).
```

Definimos la regla que chequea fórmulas en el modelo construido. Esta regla, determina por cada w del modelo si la formula F es verdadera o no.

```
model_checker([W|Ws],F)
    eval_formula(W,F,R,1),
    ((R > 0 -> (write('LA FORMULA '),write(F),
    write(' ES VERDADERA EN '),write(W),nl);
    (write('LA FORMULA '),write(F),write(' ES FALSA EN '),write(W),nl))),
    model_checker(Ws,F),nl.
```

Ejemplos de invocación para determinar si una fórmula es verdadera en algún mundo w del modelo construido:

- check_formula([p]).
- check_formula([q]).
- check_formula([bel,a1,[p]]).
- check_formula([does,a2,[q]]).
- check_formula([does,a2,[q]]).

El detalle completo de la implementación desarrollada para cada uno de los aspectos mencionados anteriormente se encuentran en el apéndice A de este documento.

Es claro que las pruebas de completitud y decidibilidad logradas en capítulos anteriores permiten garantizar implementaciones computacionales correctas sin importar, desde el punto de vista teórico, la cantidad o tamaño de los datos con los que estemos trabajando. El ejemplo que construimos es útil para verificar el comportamiento en ejecución de algunas de las reglas PROLOG implementadas. El ejemplo facilita la lectura y seguimiento de dicha ejecución.

La creación de modelos reales es un área de investigación en la que no nos adentramos en este trabajo. Sugerimos la revisión de los trabajos realizados en los proyectos ART Testbed [GSMCM07] [4] y ForTrust [5] donde se definen escenarios concretos de aplicación en dominios específicos (tasación de obras de arte en el caso de ART Testbed y seguridad informática en el caso de ForTrust).

[4] http://megatron.iiia.csic.es/art-testbed/
[5] http://www.irit.fr/ForTrust/

Capítulo 7

Implementación en SPINdle

7.1. Programación Lógica Rebatible (DePL)

La programación lógica rebatible (Defeasible Logic Programming) es cada vez más utilizada como alternativa a otras opciones que nos propone el paradigma de la programación declarativa. Relativamente más "nuevo" que Prolog, éste tipo de programación se expandió a varios dominios como: reglas y regulaciones de comercio, modelización de agentes que negocian [GR04a], argumentación y Web semántica [BAV04]. DePL permite trabajar con información incompleta o contradictoria. Otra propiedad interesante de la lógica rebatible es la *no-monotonía*. Un lógica *monótona aumenta* o mantiene la cantidad de conclusiones generadas al aumentar el número de información dada (más hechos, reglas, etc. implica mas deducciones), las lógicas no monótonas pueden reducir el número de conclusiones a medida que se agrega información nueva.

En este capítulo hablamos de SPINdle [GHP09, Lam10]. Es un razonador lógico basado en lógica rebatible, programado en Java y puede ser incluido como motor de razonamiento en distintas aplicaciones.

7.2. Definiciones Básicas sobre DL

Una teoría rebatible D [Nut01] es una terna $(F, R, <)$ donde F y R son un conjunto finito de *hechos* y *reglas* respectivamente; y una relación $>$ de "superioridad" sobre R.

Los hechos representan (como es usual) afirmaciones verdaderas. Por ejemplo, "Laika es un perro" puede ser representado como $perro(Laika)$.

Las reglas describen relaciones entre los objetos del dominio. Podemos especificar la prioridad de una regla según éstas sean: reglas *estrictas*, reglas *rebatibles (defeasible)*, y *"rebatidores" (defeaters)*.

Reglas estrictas. Son reglas en el sentido clásico (iguales a las de Prolog). Por ejemplo, una regla estricta es: "los perros son mamíferos", formalmente:

$$perro(X) \rightarrow mamífero(X)$$

Reglas Rebatibles (Defeasible). Son reglas que pueden ser rebatidas si se provee evidencia contraria. Por ejemplo, la regla "los mamíferos no vuelan", formalmente:

$$mamífero(X) \Rightarrow \neg vuela(X)$$

En general, la idea es la siguiente, si sabemos que X es un mamífero podemos concluir que no puede volar *a menos que exista evidencia contraria de mayor prioridad* la cual sugiera que el mamífero pueda volar (por ejemplo, el mamífero que representa X es un murciélago).

Defeaters ("rebatidores"). Son un tipo especial de reglas que no pueden ser usadas para inferir conclusiones directamente. Solo previenen la inferencia de otras conclusiones. Son utilizadas para "rebatir" alguna regla rebatible produciendo evidencia contraria. Por ejemplo, la regla:

$$pesado(X) \rightsquigarrow \neg vuela(X)$$

la cual dice que si algo es pesado no es suficiente razón para concluir que no vuela. Solo sirve para ofrecer evidencia en contra de para no llegar a la conclusión de que algo pesado vuela. Dicho de otra forma, no queremos inferir que algo no vuela si es pesado, simplemente sirve para no inferir que esto vuela.

DL es una lógica no-monótona "escéptica", no permite inferir conclusiones contradictorias. Para ello DL trata de resolver los conflictos. Si se pueden concluir tanto $\mathscr{A}$ como $\neg\mathscr{A}$ no se tomarán como conclusiones ninguna de las dos. En el caso que la evidencia a favor de $\mathscr{A}$ tenga prioridad sobre la evidencia a favor de $\neg\mathscr{A}$ entonces concluimos $\mathscr{A}$.

Para implementar las reglas rebatibles y los rebatidores, DL utiliza la relación de superioridad para definir la prioridad sobre las reglas, de esta manera una regla puede invalidar la conclusión de otra. Por ejemplo, dado los siguientes hechos:

$$\rightarrow pájaro \quad \rightarrow alaRota$$

y las reglas rebatibles:

$$r: \quad pájaro \Rightarrow vuela$$
$$r': \quad alaRota \Rightarrow \neg vuela$$

que se contradicen, no se podrá concluir nada sobre si los pájaros con alas rotas pueden volar. Pero introduciendo la relación de superioridad $>$ con: $r' > r$, entonces podemos concluir que el pájaro no puede volar si su ala esta rota.

Ahora hablaremos informalmente sobre cómo obtener conclusiones con DL. Sea D una teoría de la lógica rebatible. Una *conclusión* de DL es un literal etiquetado de la forma:

$+\Delta\alpha$ donde α es definitivamente probable en D. (a partir de hechos y reglas estrictas).

$-\Delta\alpha$ existe prueba que α no es definitivamente probable en D.

$+\partial\alpha$ donde α es rebatiblemente probable en D.

$-\partial\alpha$ existe una prueba que α no es rebatiblemente probable en D.

Las derivaciones estrictas son obtenidas mediante la encadenación de reglas estrictas, mientras que una conclusión rebatible α puede ser derivada si existe una regla con una conclusión es α, cuyos pre-requisitos (antecedente) ya fue probado o dados (hechos), y cualquier otra regla que derive la conclusión $\neg\alpha$ ha fallado. Dicho de otra forma, una conclusión α es (rebatiblemente) probable cuando:

- α en un hecho; o
- existe una regla estricta o rebatible que aplica a α, y
  - todas las reglas para $\neg\alpha$ son descartadas (ej. no aplican) o
  - toda regla para $\neg\alpha$ es más débil que las reglas que aplican a α.

Una definición más completa y profunda de la teoría de prueba en lógicas rebatibles puede ser encontrada en [ABGM01].

7.2.1. Bloqueo de Ambigüedad y Propagación de Ambigüedad

Decimos que un literal es *ambiguo* si existe una cadena de razonamiento que da soporte tanto a la conclusión de α como a $\neg\alpha$, y la relación de superioridad no resuelve el este conflicto.

DL en vez de concluir contradicciones como $+\partial\ q$ y $+\partial\ \neg q$ esta ambigüedad resulta de las conclusiones $-\partial\ q$ y $-\partial\ \neg q$. El comportamiento llamado *bloqueo de ambigüedad* aparece si la ambigüedad ha *bloqueado* otras conclusiones ambiguas como $-\partial\ p$ y $-\partial\ \neg p$ y una conclusión no ambigua para p ha sido generada.

Otra opción ante el bloqueo de ambigüedad es la *propagación de ambigüedad*, pero existe poco acuerdo entre aquellos autores que se dedican a estudiar estos temas. Hasta ahora la posición mas fuerte sobre el tema dice que el bloqueo de ambigüedad resulta en un patron de razonamiento anti-natural. La propagación de ambigüedad resulta en un menor número de conclusiones, lo cual puede ser útil en el caso donde las conclusiones incorrectas tienen un alto costo. Ambas se encuentran contempladas en SPINdle por lo cual podemos abstraernos del tema.

7.3. Lógica Modal Rebatible

Aquí además de hablar sobre la lógica modal rebatible introduciremos la sintaxis del lenguaje SPINdle.

Usualmente las lógicas modales son una extención de la lógica proposicional con algún operador intencional. En un sentido muy general, una lógica modal esta compuesta por dos elementos:

- la estructura lógica de base; y
- el comportamiento lógico de los operadores modales.

Hasta este momento, SPINdle implementa la semántica propuesta por (Gobernatori & Rotolo 2008)[GR08] sobre el razonamiento de lógica modal rebatible.

Átomo. Un *átomo* es un símbolo de predicado (el nombre del átomo) que puede estar seguido de una lista de parámetros. Por ejemplo: $animal(X), a, ab, animal$.

La versión actual de SPINdle es puramente proposicional. Todos los parámetros son ignorados.

Literal Básico. Un *literal básico* es un átomo a o su negación $-a$ (la negación $\neg$ se denota con el signo $-$ inmediatamente antes del nombre del átomo).

Literal Modal. Un *literal modalizado* tiene la forma

$$[\Box]a$$

donde $\Box$ es un operador modal representando un esta mental (o acción, etc) del literal. Por ejemplo: $[BEL]\ \mathscr{A}$, $[INT]\ \mathscr{A}$.

Hechos, Reglas y Rebatidores. La siguiente es la forma básica de una regla:

$$Rx[\Box] : a_1, a_2, ..., a_i, -b_1, -b_2, ..., -b_j \rhd c_1, c_2, ..., c_k$$

donde Rx es una regla etiquetada, $\Box$ es el operador modal de la regla, $a_1, a_2, ..., a_i, -b_1, -b_2, ..., -b_j$ y $c_1, c_2, ..., c_k$ son literales (modalizados) del cuerpo y la cabeza respectivamente, y $\rhd$ es el símbolo del tipo de regla (Tabla 7.1).

Tipo de Regla	Símbolo
Hecho	>>
Regla Estricta	– >
Regla Rebatible	=>
Rebatidores	~>

Cuadro 7.1: Tipo de Reglas - Símbolo

Relaciones de Superioridad. Una relación de superioridad esta representada por el símbolo ">" entre dos etiquetas de reglas.

7.4. Implementación en SPINdle

Al ser SPINdle un lenguaje en desarrollo orientado hacia la compatibilidad tanto con la lógica rebatible como con RuleML [Gov05] (lenguaje de modelado de reglas, tipo *XML*) actualmente se encuentra limitado. SPINdle se adapta mejor a la generación de ejemplos prácticos de interacción entre

operadores y reglas. que a la implementación de teorías completas y con interacciones complejas. Esto es debido a las limitaciones (falta de implementación) que este lenguaje tiene y son sumamente necesarias para la implementación de una teoría completa, a continuación desarrollaremos esta afirmación.

7.4.1. Exploración del lenguaje y restricciones

SPINdle aún no soporta las muchas funcionalidades que son indispensables para una implementación correcta del chequeador de modelos. Una de estas restricciones (la imposibilidad de parametrizar los átomos) se encuentra documentada en [Lam10] (SPINdle User Guide). Las demás restricciones fueron surgiendo a medida exploramos el lenguaje y avanzamos en la implementación del model checker. Detallaremos a continuación las más importantes:

- Estructuras de datos complejas: hasta ahora[1] la única estructura de datos soportada es la variable proposicional. No provee ningún tipo de lista o conjunto que agrupe literales, variables, etc.
- Operadores modales: solo podemos aplicar una modalidad a un literal o regla. Es imposible aplicar operadores modales a mas de un literal o variable. No podemos escribir *fbfs* de la forma: $BEL(\varphi \wedge \psi)$.
- Operadores modales "planos": un operador modal y solo uno pueden ser aplicados a un y solo un literal o variable. Esto dificulta la escritura de algunos axiomas y fórmulas. No podemos escribir *fbfs* de la forma: $BEL(INT(\varphi))$.
- Débil parametrización: no podemos representar predicados ya que no se hacen los reemplazos de variables pertinentes (ver definición de átomo en la Sección anterior).
- El operador $\sim>$: este operador no se encuentra completamente implementado, por lo cual podemos escribir este tipo de reglas pero no podemos generar conclusiones a partir de éstas.

A pesar de estas y otras falencias del lenguaje que aquí no se listan vamos a implementar un ejemplo de la literatura específica utilizando la noción de *confianza conjunta* introducida en el capítulo 2. Cabe destacar que la implementación de todo lo referido a las modalidades y relaciones entre agentes

[1] La ultima versión actual estable (2.1.0), Marzo 2011

deberá ser realizado por extensión, teniendo que definir un operador modal para cada par o pares de agentes.

Ejemplo 2 (Ejemplo de la parada de autobús, revisado).
Supongamos que el agente y se encuentra en la parada de autobús, y además hay mas gente esperando no en la parada pero cerca de y, esperando que y levante la mano para parar el autobús. Llamemos al grupo de personas G. Esta situación puede ser modelada como $JTrust^{G}_{y}(PararBus)$.

Observaciones. Los agentes pueden ser completamente independientes unos de otros, por lo cual solo podrán ser vistos como un grupo desde el exterior. Sin embargo, podemos asumir que en esta forma de delegación débil los "confiados" se sienten cómodos dependiendo de y. (3.2) no refleja ningún tipo de intención colectiva o acuerdo previo de ningún tipo (no incluye ningún operador colectivo). $JTrust$ captura un estado mental conjunto que solo es apreciable sobre la base que los integrantes de G "predicen" que y parará el autobús. Vale la pena enfatizar que uno puede confiar (incondicionalmente). En este ejemplo no existe ningún compromiso entre y y G, sin embargo G confía. Ahora, sea x un miembro de G. Si el autobús se acerca e y no parece interesado (ej. espera por otra línea de autobuses), y temiendo que x pierda su autobús, x probablemente levantará la mano para detenerlo. Ergo, en este caso $Jtrust$ se mantiene verdadera hasta que un miembro de G decide no delegar la tarea de detener el autobús a y (el estudio de este escenario escapa a el alcance de este trabajo).

7.4.2. Implementación del Ejemplo del Autobús

Como ya lo habíamos anticipado, debido a las restricciones que nos impone SPINdle debemos implementar el ejemplo por extensión. Es decir, tenemos que generar un operador modal para cada agente i (`[BEL_X1]`, `[BEL_X2]`, ...), también si el agente x cree en el agente y tiene la intención de $\mathscr{A}$, esto deberá ser representado con una nueva modalidad que lo indique `[BEL\_X\_INT\_Y] A` y así con todo operador modal y toda interacción entre agentes, modalidades, etc.

Presentamos a continuación un prototipo (el cual extenderemos para cada agente o relación entre ellos) para algunas modalidades involucradas, así como una breve descripción (el código completo puede encontrarse en el apéndice B.

Conflictos. Definimos un conflicto a cualquier interacción entre operadores que lleve a una conclusión errónea (se deduce $\perp$).

```
#PROTOTIPE# OP != -OP
```

Este prototipo de regla sirve para prevenir que un operador colacione con la negación del mismo.

Axiomas relacionados al operador BEL. Se listan aquí los prototipos para la axiomatización de la lógica y explicaremos brevemente a cuál refiere. Estos axiomas son los que definen su semántica y por ello es necesario implementarlos.

Los prototipos de axioma se corresponden con los de [DKV02] y explicaremos brevemente su significado:

- *Belief Distribution (K).* Es el axioma K aplicado para el operador BEL

  ```
  #PROTOTYPE# A2 [BEL]: [BEL] Phi(X) -> Psi(X)
  ```

- *Positive Introspection.* Si agente cree en algo, también cree que lo cree.

  ```
  #PROTOTYPE# A4: [BEL_X] Phi(X) -> [BEL_X_BEL_X] Phi(X)
  ```

- *Negative Introspection.* Si agente no cree en algo, también cree que lo no cree.

  ```
  #PROTOTYPE# A5: [-BEL_X] Phi(X) -> [BEL_X_-BEL_X] Phi(X)
  ```

- *Consistency.* Los agentes no pueden creer en "nada".

  ```
  #PROTOTYPE# A6: [-BEL_X] BOTTOM
  ```

- *Belief Generalization.* Los agentes pueden creer en algo que se sabe verdadero.

  ```
  #PROTOTYPE# R2: Phi(X) -> [BEL_X] Phi(X)
  ```

- *General Belief.* Todos los agentes de un grupo creen en algo.

  ```
  #PROTOTYPE# G={X1, X2, ..., XN} C1: [BEL_X1] Phi(X), [BEL_X2] Phi(X),
                    ..., [BEL_XN] Phi(X) -> [E-BEL_G] Phi(X)
  ```

- *Common Belief.* Todos los agentes de un grupo creen en lo mismo y lo saben (creen).

```
#PROTOTYPE# G={X1, X2, ..., XN} C2: [E-BEL_G] Phi(X),
                    [E-BEL_G_C-BEL_G] Phi(X) -> [C-BEL_G] Phi(X)
```

Axiomas relacionados al operador $GOAL$ ***e*** INT**.** Se detallan los demás prototipos para los operadores $GOAL$ e INT se definen los axiomas que le dan su semántica:

```
#--- GOAL AXIOMS ---
#PROTOTYPE# A2D [GOAL]: [GOAL] Phi(X) -> Psi(X)
#PROTOTYPE# R2D: Phi(X) -> [GOAL_X] Phi(X)
#PROTOTYPE# A7DB:[GOAL_X] Phi(X) -> [BEL_X_GOAL_X] Phi(X)
#PROTOTYPE# A8DB:[-GOAL_X] Phi(X) -> [BEL_X_-GOAL_X] Phi(X)

#--- INT AXIOMS ---
#PROTOTYPE# A2I [INT]: [INT] Phi(X) -> Psi(X)
#PROTOTYPE# R2I: Phi(X) -> [INT_X] Phi(X)
#PROTOTYPE# A6I: [-INT_X] BOTTOM
#PROTOTYPE# A7IB:[INT_X] Phi(X) -> [BEL_X_INT_X] Phi(X)
#PROTOTYPE# A8IB:[-INT_X] Phi(X) -> [BEL_X_-INT_X] Phi(X)
#PROTOTYPE# A9IB:[INT_X] Phi(X) -> [GOAL_X] Phi(X)
#PROTOTYPE# G={X1, X2, ..., XN} M1: [INT_X1] Phi(X), [INT_X2] Phi(X),
                    ..., [INT_XN] Phi(X) -> [E-INT_G] Phi(X)
#PROTOTYPE# G={X1, X2, ..., XN} M2: [E-INT_G] Phi(X),
                    [E-INT_G_M-INT_G] Phi(X) -> [M-INT_G] Phi(X)
#PROTOTYPE# G={X1, X2, ..., XN} M3: [M-INT_G] Phi(X),
                    [C-BEL_G_M-INT_G] Phi(X) -> [C-INT_G] Phi(X)
```

Axiomas relacionados al operador $DOES$**.** Estos son los axiomas referentes a la modalidad $DOES$:

- *Axioma T.* Expresa el éxito de realizar una acción.

```
#PROTOTYPE#  E1: [DOES_X] Phi(X) -> Phi(X)
```

- *Axioma NO.* Expresa el concepto de que no se puede realizar $\top$ ya que es independiente a los agentes.

```
#PROTOTYPE# E2: [-DOES_X] _TOP
```

- *Aglomeración.* Si realizamos dos acciones separadas, podemos realizarlas "juntas".

```
#PROTOTYPE#  E3: [DOES_X] Phi(X), [DOES_X] Psi(X) ->
                                        [DOES_X] Phi_and_Psi(X)
```

Axiomas relacionados al operador $TRUST$ ***y*** $JTRUST$. Los siguientes son la representación de (3.1) y (3.2) respectivamente.

```
#--- TRUST AXIOMS ---
#PROTOTYPE# [GOAL_X]Phi(X), [BEL_X_DOES_Y]Phi(X), [INT_X_DOES_X]Phi(X),
                            [-INT_X_DOES_Y]Phi(X), [BEL_X_INT_Y]Phi(X),
                             [GOAL_X_INT_Y]Phi(X), -> [TRUST_X_Y]Phi(X)

#--- JTRUST AXIOMS ---
#PROTOTIPE# G={X1, X2, ..., XN} [TRUST_X1_Y] Phi(X), [TRUST_X2_Y] Phi(X),
                           ..., [TRUST_XN_Y] Phi(X) -> [JTRUST_G_Y] Phi(X)
```

Extendiendo estos prototipos de axioma para cada agente, cada interacción entre modalidades y cada agente podemos implementar nuestra teoría. Esto genera una gran cantidad de reglas y resulta poco práctico para una implementación "real". Por ejemplo, suponiendo un grupo de 4 agentes, probar la validez de una fórmula debe utilizar una axiomatización donde todos los axiomas deben ser implementados y todas las interacciones entre modalidades y agentes deba ser implementada (el peor caso) tenemos al menos 24^4 reglas (el axioma `E2` no cuenta) más otra gran cantidad de hechos y reglas no relacionadas con los axiomas.

Para este ejemplo implementamos solamente aquellos axiomas que son necesarios, no solo para reducir la cantidad de axiomas, sino también para decrementar la cantidad de deducciones y mantener el ejemplo manejable. Solo implementamos instancias particulares de los axiomas `R2`, `R2D`, `R2I` y `E1` (mencionadas más arriba).

El ejemplo está instanciado para 3 agentes, nuestro grupo G consta de los agentes `A1`, `A2`, y `A3` que se encuentran en la parada de autobús, los agentes `A2` y `A3` confían en que el agente `A1` detendrá el autobús. Esto lo modelizamos de la siguiente forma:

```
#--- AGENT A3
```

```
>>[GOAL_A3] PararBus(X)
>>[-DOES_A3] PararBus(X)
#--- AGENT A2 ---
>>[GOAL_A2] PararBus(X)
>>[-DOES_A2] PararBus(X)
#--- AGENT A1 ---
>>[INT_A1] PararBus(X)
>>[DOES_A1] PararBus(X)
```

El agente A3 tiene el objetivo de parar el autobús ([GOAL_A3] PararBus(X)) pero no tiene contemplado hacerlo por sí mismo ([-DOES_A3] PararBus(X)), del mismo modo para el agente A2. Mientras que el agente A1 (o las suposiciones de los agentes sobre lo que hará) tenga intención de detener el autobús ([INT_A1] PararBus(X)) y cuando este llegue lo detenga ([DOES_A1] PararBus(X)) los agentes en el grupo confiarán en él.

La ejecución bajo estas suposiciones arroja los siguientes resultados (mostramos solo los relevantes):

```
Conclusions
===========
+D PararBus(X)

+D [-DOES_A2]PararBus(X)
+D [-DOES_A3]PararBus(X)

+D [DOES_A1]PararBus(X)

+D [JTRUST_G_A1]PararBus(X)
+D [TRUST_A2_A1]PararBus(X)
+D [TRUST_A3_A1]PararBus(X)
```

Es decir, el autobús es detenido por el agente A1, los agentes A2 y A3 confían en A1 y los integrantes del grupo confían conjuntamente en A1.

Si cuando el autobús llega el agente A1 no lo detiene, ni tampoco tenía la imtención de hacerlo, entonces el grupo dejará de confiar en él. Esto lo podemos modelizar de la siguiente forma:

```
#--- AGENT A1 ---
>>[-INT_A1] PararBus(X)
>>[-DOES_A1] PararBus(X)
```

Las deducciones generadas son las siguientes:

```
Conclusions
===========
+D [-DOES_A1]ParарBus(X)
+D [-DOES_A2]ParарBus(X)
+D [-DOES_A3]ParарBus(X)

-D ParарBus(X)

-D [JTRUST_G_A1]ParарBus(X)
-D [TRUST_A2_A1]ParарBus(X)
-D [TRUST_A3_A1]ParарBus(X)
```

El autobús no es detenido por `A1` y los agentes `A2` y `A3` además de perder el autobús dejan de confiar en `A1`.

La implementación y deducciones completas se proveen en el apéndice B.

Capítulo 8

Conclusiones

En este trabajo hemos ofrecido detalles técnicos para la combinación de lógicas normales y no-normales que permiten modelar la noción de confianza colectiva y sirven para probar completitud y decidibilidad de la lógica resultante. Las combinaciones logradas llevan a diferentes niveles de expresividad del sistema mediante el uso de técnicas de fibrado (temporalización y modalización).

Las propiedades de completitud y modelo finito son sustento de implementaciones computacionales para lógicas proposicionales modales y habilitan para diseñar algoritmos que computen satisfactibilidad de una fórmula dentro de un modelo; esto es: teniendo un modelo y una fórmula, podemos averiguar si la fórmula es verdadera en tal modelo. Recordemos que muchos problemas interesantes y complicados (clique, viajante, colorabilidad, subsunción de fórmulas de la lógica de primer orden, por citar sólo algunos) pueden ser reducidos al problema de satisfactibilidad (SAT), con lo cual, al tener una implementación para SAT tenemos manera de resolver muchos de estos problemas.

Ver a la lógica base (la multi-modal multi-agente para representar confianza en MAS) como una lógica combinada sugirió la estrategia de combinación de chequeadores de modelos, todo sobre la base de las propiedades de completitud y modelo finito probadas previamente. Al aplicar la técnica de fibrado pudimos identificar claramente diferentes partes de la teoría con sus diferentes semánticas y diferentes estructuras. Consecuentemente definimos diferentes chequeadores de modelos en correspondencia con las estructuras de Kripke y de Scott-Montague). Definimos dichos chequeadores con un enfoque de programación imperativa y a un nivel muy elevado de abstracción.

Luego del trabajo teórico, exploramos una posible implementación computacional en los lenguajes PROLOG y SPINdle partiendo de la definición de los chequeadores. Cláusulas, reglas y hechos sumado a la versatilidad en el manejo de listas facilitaron el desarrollo de la implementación en PROLOG con respecto a SPINdle. En PROLOG implementamos las reglas y axiomas de la lógica del fibrado y construimos un chequeador de modelos. En SPINdle, la ausencia de estructuras de datos complejas, la falta de mecanismos de sustitución de variables y la débil parametrización para manejar operadores modales dificultó la escritura natural de axiomas y fórmulas. Sin embargo, logramos la implementación de un ejemplo sencillo instanciando de forma extensiva los axiomas involucrados que componen el fibrado.

Calculamos la complejidad computacional de los algoritmos resultantes para el fibrado. También realizamos un chequeador para la técnica de unión ("join") cuyo análisis profundo de complejidad excede este trabajo.

A partir de lo realizado consideramos tres temas de investigación para futuros trabajos:

- Se abre una linea de trabajo referida al estudio de la aplicabilidad de métodos orientados a pruebas (Ej. fibrado de *tableaux*).

- *Complejidad* La lógica propuesta para el fibrado, aunque decidible, es EXPTIME completa. En [DVDK07] se proponen algunos métodos para reducir esta complejidad, como limitar la profundidad de las fórmulas modales o restringir el número de átomos proposicionales. Es interesante comprobar si estas técnicas pueden ser útiles en el marco de trabajo actual.

- *Otras combinaciones* En términos teóricos, la idea de razonamiento sobre el tiempo debería extender la teoría lógica actual. Con esto obtendríamos un nivel más de fibrado. Por ejemplo: una temporalización básica es equivalente a colocar un sistema temporal "sobre" un sistema modalizado de manera similar a lo desarrollado en nuestro trabajo. Consideremos el modelo $(T, <, g, t_0)$. El frame externo $(T, <)$ corresponde con la evolución temporal del sistema; t_0 en T es el punto inicial en el tiempo. El sistema evoluciona a través del tiempo en el sentido de que nuevos grupos y creencias, intenciones, relaciones de confianza y obligaciones genéricas/individuales son generadas (mientras otras se vuelven obsoletas). g es la función total que permite obtener un modelo $\mathfrak{M}$ para cada punto en el tiempo.

Apéndice A

Código PROLOG

```
%************************************************************************
% Autores :     Agustin Ambrossio - Leandro Mendoza.
% Descripción : Implementación del fibrado.
%************************************************************************

%% FÓRMULAS.
% Las fórmulas se representan mediante listas. De esta manera, una
% fórmula es una lista, cuya estructura depende si se trata
% de una una fórmula booleana, una fórmula modal, un hecho o un
% combinación de éstas.

% Describimos las fórmulas bien formadas con las que trabajamos:

%% F : letra proposicional.
%% F : F & F. Conectivo and (Expresión booleana).
%% F : F v F. Conectivo or (Expresión booleana).
%% F : F => F. Implicacion (Expresión booleana).
%% F : F <=>. Equivalencia (Expresión booleana).
%% F : -(F). Negación.
%% F : bel(i,F). Donde i es un agente.
%% F : int(i,F). Donde i es un agente.
%% F : goal(i,F). Donde i es un agente.
%% F : c-bel(g,F). Donde g es un grupo de agentes.
%% F : c-int(g,F). Donde g es un grupo de agentes.
%% F : e-bel(g,F). Donde g es un gruopo de agentes.
%% F : e-int(g,F). Donde g es un grupo de agentes.
%% F : m-int(g,F). Donde g es un grupo de agtentes.


%% FÓRMULAS BOOLEANAS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Las expresiones booleanas se representan utilizando listas
```

```
% Alfabeto utilizado.
%% Conectivas lógicas: neg(negación), &(conjuncion), v(disyuncion),
=>(implicación), <=>(equivalencia o sii).
%% Símbolos auxiliares: (,).

% Sean F y G fórmulas, también lo son: -F, (F & G), (F v G), (F => G),
(F <=> G)

%% Operadores unarios se representan de la siguiente
% forma: [operador_unario, fórmula].
%% Ejemplo: Sea F una fórmula, el operador not  aplicado a dicha
% fórmula se define: [neg,F].
%% Operadores binarios, se representan de la siguiente forma:
% [fórmula,operador_binario, fórmula].
%% Ejemplo: Sean F y G fórmulas, la fórmula que representa la
% conjunción entre dichas fórmulas se define [F,&,G].
%% Ejemplo: Sean F y G fórmulas, la fórmula que representa la
% disyunción entre dichas fórmulas se define [F,v,G].
%% Ejemplo: Sean F y G fórmulas, la fórmula que representa la
% implicación de F hacia G se define [F,=>,G].
%% Ejemplo: Sean F y G fórmulas, la fórmula que representa la
% implicación de F hacia G se define [F,=>,G].
%% Ejemplo: Sean F y G fórmulas, la fórmula que representa la
% equivalencia lógica entre F y G se define [F,<=>,G].

% Eliminación de paréntesis:
%%  Eliminación de paréntesis externos.
%%  Precedencia: neg, &, v, =>, <=>.
%%  Asociatividad: & y v asocian por la derecha.

%  Declaración de operadores:
:- op(610, fy, neg). % negacion.
:- op(620, xfy, &). % conjuncion.
:- op(630, xfy, v). % disyuncion.
:- op(640, xfy, =>). % condicional.
:- op(650, xfy, <=>). % equivalencia.

%  Valores de verdad:
%%   true: 1 y false: 0

%  Def. de valor de verdad:
%%   valor_de_verdad(V) si V es un valor de verdad.

valor_de_verdad(1).
valor_de_verdad(0).

% Funciones de verdad para los operadores definidos
%% funcion_de_verdad(+Op, +V1, +V2, -V) si Op(V1,V2)=V.
%% funcion_de_verdad(+Op, +V1, -V) si Op(V1)) = V.
```

```
funcion_de_verdad(v, 0, 0, 0) :- !.
funcion_de_verdad(v, _, _, 1).
funcion_de_verdad(&, 1, 1, 1) :- !.
funcion_de_verdad(&, _, _, 0).
funcion_de_verdad(=>, 1, 0, 0) :- !.
funcion_de_verdad(=>, _, _, 1).
funcion_de_verdad(<=>, X, X, 1) :- !.
funcion_de_verdad(<=>, _, _, 0).
funcion_de_verdad(neg, 1, 0).
funcion_de_verdad(neg, 0, 1).


%% FÓRMULAS MODALES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Representamos las fórmulas modales utilizando listas.
%% [Constante_Modal, Agente (o grupo de agentes), Fórmula]
%% Ejemplo. Sea i un agente y F una fórmula:
%%  bel(A,F) se representa como [bel,A,F]
%%      int(A,F) se representa como [int,A,F]
%%  goal(A,F) se representa como [goal,A,F]
%%      e-bel(G,F) se representa como [e-bel,G,F]


%%%%% REPRESENTACIÓN DE FRAMES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Definimos un frame indicando sus mundos y relaciones.
%% Existen tres tipo de relaciones normales relacionadas con los
% operadores bel, int y goal.
%% rel_bel(A,W,V). Define una relación "bel" dirigida, para el agente
% A entre el mundo W y el mundo V.
%% rel_int(A,W,V). Define una relación "int" dirigida, para el agente
% A entre el mundo W y el mundo V.
%% rel_goal(A,W,V). Define una relación "goal" dirigida, para el
% agente A entre el mundo W y el mundo V.

%% Existe un tipo  de relación no-normal relacionada con el operador
% does
%% rel_does(A,W,V). Define una relación "does" dirigida, para el
% agente A entre el mundo W y el mundo V.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% REPRESENTACIÓN DE HECHOS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Los hechos se corresponden con proposiciones atómicas que son
% verdaderas en un mundo del frame.
%% Los hechos se representan mediante listas de un único elemento
```

```
% (constantes) utilizando letras proposicionales p,q,r...
%% Para representar los hechos utilizamos reglas de la siguiente manera
% hecho(W,F,R).
%% Esta regla define la función de valuación del modelo.
%% W es una constante que representa un estado del frame o modelo,
% F representa una fórmula atómica y R un valor entero que define si
% F vale (1) o no vale (0) en el estado W.

%% Ejemplo hecho(w2,[a],1). En el mundo o estado w2, vale el hecho [a].
%% Ejemplo hecho(w2,[a],1). En el mundo o estado w3, no vale el hecho [b].


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%% EVALUACIÓN DE FÓRMULAS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Para evaluar fórmulas en un modelo, definimos diferentes reglas que
% describen el comportamiento de los en cada caso.

%% Regla auxiliar de corte
%% N determina la tabulación (y profundidad de la fórmula).
corte(1,N):-
%NO is N+2,
%tab(NO),
%write('La formula evaluada es verdadera'),nl,
true.
corte(0,N):-
NO is N+2,
tab(NO),
write('La formula evaluada es falsa'),nl,
false.


%% EVALUACIÓN DE FÓRMULAS BOOLEANAS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

eval_formula(W,[neg,F],R,N):-
NO is N+2,
tab(NO+1),write('Evaluando negacion de: '),
write(F),tab(1),nl,
eval_formula(W,F,R1,N),
tab(NO+2),write('Llamando a la funcion de
verdad de la negacion con: '),tab(1),write(F),nl,
funcion_de_verdad(neg, R1, R),
tab(NO),write('Fin de ejecución regla
eval_formula(W,[neg,F],R,N)'),nl,!.


eval_formula(W,[A,&,B],R,N):-
```

```
N0 is N+2,
tab(N0),write('Ejecutando regla
eval_formula(W,[A,&,B],R,N)'),nl,
tab(N0+2),write('Evaluando and entre'),tab(1),
write(A),tab(1),write('y'),tab(2),write(B),nl,
eval_formula(W,A,R1,N0),
eval_formula(W,B,R2,N0),%!,
funcion_de_verdad(&,R1,R2,R),
tab(N0+2),write('Resultado de: '),write('['),
write(A),write(' & '),write(B),write(']'),write(' en '),write(W),write(' es '),
write(R),nl,
tab(N0),write('Fin de ejecución regla
eval_formula(W,[A,&,B],R,N)'),nl.


eval_formula(W,[A,v,B],R,N):-
N0 is N + 2,
tab(N0),write('Ejecutando regla
eval_formula(W,[A,v,B],R,N)'),nl,
tab(N0+2),write('Evaluando or entre'),tab(1),
write(A),tab(1),write('y'),tab(2),write(B),nl,
eval_formula(W,A,R1,N0),
eval_formula(W,B,R2,N0),%!,
funcion_de_verdad(v,R1,R2,R),
tab(N0+2),write('Resultado de: '),write('['),
write(A),write(' v '),write(B),write(']'),write(' en '),write(W),write(' es .'),
write(R),nl,
tab(N0),write('Fin de ejecución regla
eval_formula(W,[A,v,B],R,N)'),nl.


eval_formula(W,[A,=>,B],R,N):-
N0 is N + 2,
tab(N0),write('Ejecutando regla
eval_formula(W,[A,=>,B],R,N)'),nl,
tab(N0+2),write('Evaluando => entre'),tab(1),
write(A),tab(1),write('y'),tab(2),write(B),nl,
eval_formula(W,A,R1,N0),
eval_formula(W,B,R2,N0),%!,
tab(N0+2),write('Evaluando funcion => entre'),
tab(1),write(R1),tab(1),write('y'),tab(2),
write(R2),nl,
funcion_de_verdad(=>,R1,R2,R),nl,
tab(N0+2),write('Resultado de: '),write('['),
write(A),write(' => '),write(B),write(']'),write(' en '),write(W),
                              write(' es '),write(R),nl,
tab(N0),write('Fin de ejecución regla
eval_formula(W,[A,=>,B],R,N)'),nl.
```

```
% EVALUACIÓN DE FÓRMULAS MODALES INDIVIDUALES NORMALES %%%%%%%%%%%%%%%%%%%%

eval_formula(W,[bel,A,F],R,N):-
N0 is N + 2,
tab(N0),write('Ejecutando regla
eval_formula(W,[bel,A,F],R,N)'),nl,
tab(N0+2),write('Evaluando: '),write('[bel,'),write(A),
write(','),write(F),write(']'),write(' en '),write(W),nl,
rel_bel(A,W,Ad),
tab(N0+2),write('Mundos adyacentes a '),write(W),write(' '),
write(Ad),nl,
eval_in_adjacent(Ad,F,R,N0),
corte(R,N),
tab(N0+2),write('El resultado de: '),write('[bel,'),
                    write(A),write(','),write(F),write(']'),write(' en '),
                    write(W),write(' es '),write(R),nl,
tab(N0),write('Fin ejecucion regla:
eval_formula(W,[bel,A,F],R,N). '),nl.


eval_formula(W,[int,A,F],R,N):-
N0 is N + 2,
tab(N0),write('Ejecutando regla
eval_formula(W,[int,A,F],R,N)'),nl,
tab(N0+2),write('Evaluando: '),write('[int,'),write(A),
write(','),write(F),write(']'),write(' en '),write(W),nl,
rel_int(A,W,Ad),
tab(N0+2),write('Mundos adyacentes a '),write(W),write(' '),
write(Ad),nl,
eval_in_adjacent(Ad,F,R,N0),
corte(R,N),
tab(N0+2),write('El resultado de: '),write('[int,'),write(A),
write(','),write(F),write(']'),write(' en '),write(W),
write(' es '),write(R),nl,
tab(N0),write('Fin ejecucion regla:
eval_formula(W,[int,A,F],R,N). '),nl.


eval_formula(W,[goal,A,F],R,N):-
N0 is N + 2,
tab(N0),write('Ejecutando regla
eval_formula(W,[goal,A,F],R,N)'),nl,
tab(N0+2),write('Evaluando: '),write('[goal,'),write(A),
write(','),write(F),write(']'),write(' en '),write(W),nl,
rel_goal(A,W,Ad),
tab(N0+2),write('Mundos adyacentes a '),write(W),write(' '),
write(Ad),nl,
eval_in_adjacent(Ad,F,R,N0),
```

```
corte(R,N),
tab(NO+2),write('El resultado de: '),write('[goal,'),write(A),
write(','),write(F),write(']'),write(' en '),write(W),
write(' es '),write(R),nl,
tab(NO),write('Fin ejecucion regla:
eval_formula(W,[goal,A,F],R,N). '),nl.

eval_in_adjacent([],F,R,N):-
NO is N+2,
tab(NO),write('Ejecutando regla eval_in_adjacent([],F,R,N)'),nl,
R is 1,
tab(NO),write('Fin ejecucion regla eval_in_adjacent([],F,R,N)'),nl.

eval_in_adjacent([X|Xs],F,R,N) :-
                    NO is N+2,
tab(NO),write('Ejecutando regla eval_in_adjacent([X|Xs],F,N)'),nl,
tab(NO+2),write('Evaluando: '),write(F),write(' en '),write(X),nl,
    eval_formula(X,F,R,NO),
corte(R,N),
tab(NO+2),write('Evaluando: '),write(F),write(' en '),write(Xs),nl,
eval_in_adjacent(Xs,F,R,NO),
tab(NO),write('Fin ejecucion regla: eval_in_adjacent([X|Xs],F,N)'),nl.


%% EVALUACIÓN DE FÓRMULAS MODALES INDIVIDUALES NO-NORMALES %%%%%%%%%%%%%%%%%

eval_formula(W,[does,A,F],R,N):-
NO is N + 2,
tab(NO),write('Ejecutando regla
eval_formula(W,[does,A,F],R,N)'),nl,
tab(NO+2),write('Evaluando: '),write('[does,'),write(A),
write(','),write(F),write(']'),write(' en '),write(W),nl,
rel_does(A,W,Ad),
tab(NO+2),write('Mundos adyacentes a '),write(W),
write(' '),write(Ad),nl,
eval_does_in_adjacent(Ad,F,R,NO),
%corte(R,N),
tab(NO+2),write('El resultado de: '),write('[does,'),
write(A),write(','),write(F),write(']'),write(' en '),
write(W),write(' es '),write(R),nl,
tab(NO),write('Fin ejecucion regla:
eval_formula(W,[does,A,F],R,N). '),nl.

eval_does_in_adjacent([X|Xs],F,R,N) :-
                    NO is N+2,
tab(NO),write('Ejecutando
regla eval_does_in_adjacent([X|Xs],F,N)'),nl,
tab(NO+2),write('Evaluando: '),write(F),write(' en '),
write(X),nl,
```

```
    ((eval_formula(X,F,R,NO),! -> (tab(NO+2),
    write('La formula'),write(F),write(' es verdadera en '),
    write(X),!,nl);
(tab(NO+2),
write('La formula '),
write(F),write(' es falsa en '),
write(X),nl),
nl,
tab(NO+2),write('Evaluando: '),write(F),write(' en '),
write(Xs),nl,
eval_does_in_adjacent(Xs,F,R,NO))),
tab(NO),write('Fin ejecucion regla:
eval_does_in_adjacent([X|Xs],F,N)'),nl.

eval_does_in_adjacent([],F,R,N):-
NO is N+2,
tab(NO),write('Ejecutando regla
eval_does_in_adjacent([],F,R,N)'),nl,
R is 0,
tab(NO),write('Fin ejecucion regla
eval_does_in_adjacent([],F,R,N)'),nl.


%% EVALUACIÓN DE FÓRMULAS MODALES COLECTIVAS GENERALES %%%%%%%%%%%%%%%%%%%%%%%%

%%%% GENERAL BELIEF (C1)
% G es una lista que representa un grupo de agentes
eval_formula(W,[e-bel,G,F],R,N):-
NO is N + 2,
tab(NO),write('Ejecutando regla
eval_formula(W,[e-bel,G,F],R,N)'),nl,
tab(NO+2),write('Evaluando: '),write('[e-bel'),write(','),
write(G),write(','),write(F),write(']'),write(' en '),
write(W),nl,
tab(NO+2),write('Agentes en el grupo G'),write(' '),
write(G),nl,
eval_bel_in_group(G,W,F,R,NO), %devolver R.
corte(R,N),
tab(NO+2),write('El resultado de: '),write('[e-bel,'),
write(','),write(G),write(','),write(F),write(']'),
write(' en '),write(W),write(' es '),write(R),nl,
tab(NO),write('Fin ejecucion regla:
eval_formula(W,[e-bel,A,F],R,N). '),nl.

% []: lista (grupo) de agentes.
% W: mundo o punto del modelo.
% F: fórmula.
% R: resultado.
% N: identación (utilizado para seguir la ejecución a través de logs).
```

```
eval_bel_in_group([],W,F,R,N):-
NO is N+2,
tab(NO),write('Ejecutando regla
eval_bel_in_group([],W,F,R,N)'),nl,
R is 1,
tab(NO),write('Fin ejecucion regla
eval_bel_in_group([],W,F,R,N)'),nl.

eval_bel_in_group([X|Xs],W,F,R,N) :-
                    NO is N+2,
tab(NO),write('Ejecutando regla
eval_bel_in_group([X|Xs],W,F,N)'),nl,
tab(NO+2),write('Evaluando: '),write(F),write(' en '),
write(W),nl,
    eval_formula(W,[bel,X,F],R,NO),
corte(R,N),
tab(NO+2),write('Evaluando: '),write(F),write(' en '),
write(Xs),nl,
eval_bel_in_group(Xs,W,F,R,NO),
tab(NO),write('Fin ejecucion regla:
eval_bel_in_group([X|Xs],F,N)'),nl.


%%%% GENERAL INTENTION (M1)
% G es una lista que representa un grupo de agentes
eval_formula(W,[e-int,G,F],R,N):-
NO is N + 2,
tab(NO),write('Ejecutando regla
eval_formula(W,[e-int,G,F],R,N)'),nl,
tab(NO+2),write('Evaluando: '),write('[e-int'),write(','),
write(G),write(','),write(F),write(']'),write(' en '),
write(W),nl,
tab(NO+2),write('Agentes en el grupo G'),write(' '),
write(G),nl,
eval_int_in_group(G,W,F,R,NO),.
corte(R,N),
tab(NO+2),write('El resultado de: '),
write('[e-int,'),write(','),write(G),write(','),write(F),
write(']'),write(' en '),write(W),write(' es '),write(R),nl,
tab(NO),write('Fin ejecucion regla:
eval_formula(W,[e-int,G,F],R,N). '),nl.

% []: lista (grupo) de agentes.
% W: mundo del modelo.
% F: fórmula.
% R: resultado.
% N: identación.
eval_int_in_group([],W,F,R,N):-
NO is N+2,
```

```
tab(NO),write('Ejecutando regla
eval_int_in_group([],W,F,R,N)'),nl,
R is 1,
tab(NO),write('Fin ejecucion regla
eval_int_in_group([],W,F,R,N)'),nl.

eval_int_in_group([X|Xs],W,F,R,N) :-
                    NO is N+2,
tab(NO),write('Ejecutando regla
eval_int_in_group([X|Xs],W,F,N)'),nl,
tab(NO+2),write('Evaluando: '),
write(F),write(' en '),write(W),nl,
    eval_formula(W,[int,X,F],R,NO),
corte(R,N),
tab(NO+2),write('Evaluando: '),write(F),
write(' en '),write(Xs),nl,
eval_int_in_group(Xs,W,F,R,NO),
tab(NO),write('Fin ejecucion
regla: eval_int_in_group([X|Xs],F,N)'),nl.


eval_formula(W,[F],R,N):-
NO is N + 2,
tab(NO),write('Ejecutando regla
eval_formula(W,[F],R,N)'),nl,
tab(NO+2),write('Evaluando hecho '),
write([F]),write(' en '),write(W),tab(1),nl,
hecho(W,[F],R),
tab(NO+2), write('Resultado de evaluar hecho '),write(F),
write(' en '),write(W),write(' '),write(R),nl,
tab(NO),write('Fin ejecucion regla hecho
eval_formula(W,[F],R,N)'),nl.


% AXIOMAS Y REGLAS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% AXIOMAS Y REGLAS GENERALES

%%%% MODUS PONENS
eval_formula(W,F2,R,N):-
                            NO is N + 2,
tab(NO),write('Evaluando regla: modus ponens'),nl,
tab(NO+2),
                            write('Evaluando regla: modus ponens. Formula F1'),nl,
eval_formula(W,F1,R,N),
tab(NO+2),
                            write('Evaluando regla: modus ponens. Formula F1=>F2'),nl,
eval_formula(W,[F1,=>,F2]],R,N),
tab(NO),write('Fin evaluacion regla modus ponens'),nl.
```

```
%%%% AXIOMA K PARA OPERADORES MODALES

eval_formula(W,[[bel,A,F1],=>,[bel,A,F2]],R,N):-
NO is N + 2,
tab(NO),write('Ejecutando axioma K
eval_formula(W,[[bel,A,F1],=>,[bel,A,F2]],R,N)'),nl,
tab(NO),write('Ejecutando
eval_formula(W,[bel,A,[F1,=>,F2]],R,N).'),nl,
eval_formula(W,[bel,A,[F1,=>,F2]],R,N),
tab(NO),write('Fin ejecucion
eval_formula(W,[bel,A,[F1,=>,F2]],R,N)'),nl,
tab(NO),write('Fin de ejecución axioma
K eval_formula(W,[[bel,A,F1],=>,[bel,A,F2]],R,N). Resultado: '),write(R),nl.

eval_formula(W,[[int,A,F1],=>,[int,A,F2]],R,N):-
NO is N + 2,
tab(NO),write('Ejecutando axioma K
eval_formula(W,[[int,A,F1],=>,[int,A,F2]],R,N)'),nl,
tab(NO),write('Ejecutando
eval_formula(W,[int,A,[F1,=>,F2]],R,N).'),nl,
eval_formula(W,[int,A,[F1,=>,F2]],R,N),
tab(NO),write('Fin ejecucion
eval_formula(W,[int,A,[F1,=>,F2]],R,N)'),nl,
tab(NO),write('Fin de ejecución
axioma K eval_formula(W,[[int,A,F1],=>,[int,A,F2]],R,N). Resultado: '),
                              write(R),nl.

eval_formula(W,[[goal,A,F1],=>,[goal,A,F2]],R,N):-
NO is N + 2,
tab(NO),write('Ejecutando axioma K
eval_formula(W,[[goal,A,F1],=>,[goal,A,F2]],R,N)'),nl,
tab(NO),write('Ejecutando
eval_formula(W,[goal,A,[F1,=>,F2]],R,N).'),nl,
eval_formula(W,[int,A,[F1,=>,F2]],R,N),
tab(NO),write('Fin ejecucion
eval_formula(W,[goal,A,[F1,=>,F2]],R,N)'),nl,
tab(NO),write('Fin de ejecución
axioma K eval_formula(W,[[goal,A,F1],=>,[goal,A,F2]],R,N). Resultado: '),
                              write(R),nl.


%%% AXIOMAS Y REGLAS PARA BELIEFS INDIVIDUALES

%%%% BELIEF DISTRIBUTION (A2)
eval_formula(W,[bel,A,F2],R,N):-
                           NO is N + 2,
tab(NO),write('Evaluando axioma:
belief distribution'),nl,
```

```
tab(NO+2),write('Evaluando axioma: belief
distribution. Formula F1'),nl,
eval_formula(W,[bel,A,_],R,N),
tab(NO+2),write('Evaluando axioma:
belief distribution. Formula F1=>F2'),nl,
eval_formula(W,[bel,A,[_,=>,F2]],R,N),
tab(NO),write('Fin evaluacion axioma:
belief generalization'),nl.

%%%% POSITIVE INTROSPECTION (A4)
eval_formula(W,[bel,A,[bel,A,F]],R,N):-
                            NO is N + 2,
                            tab(NO),write('Evaluando
axioma: positive introspection '),nl,
eval_formula(W,[bel,A,F],R,N),nl,
tab(NO+2), write('Resultado de
positive introspection '),write(F),write(' en '),write(W),write(' '),write(R),nl,
tab(NO),write('Fin evaluacion
axioma: positive introspection'),nl.

%%%% NEGATIVE INTROSPECTION (A5)
eval_formula(W,[bel,A,[neg,[bel,A,F]]],R,N):-
                            NO is N + 2,
                            tab(NO),write('Evaluando axioma:
negative introspection '),nl,
eval_formula(W,[neg,[bel,A,F]],R,N),nl,
tab(NO+2), write('Resultado de negative introspection '),
write(F),write(' en '),write(W),write(' '),write(R),nl,
tab(NO),write('Fin evaluacion axioma
negative introspection'),nl.

%%%% CONSISTENCY (A6)
eval_formula(W,[bel,A,false],R,N):-
                            NO is N + 2,
                            tab(NO),write('Evaluando belief false '),nl,
R is 0,
tab(NO),write('Fin evaluacion belief false'),nl,!.


eval_formula(W,[neg,[bel,A,false]],R,N):-
                            NO is N + 2,
                            tab(NO),write('Evaluando axioma:
belief consistency '),nl,
R is 0,
tab(NO),write('Fin evaluacion axioma:
belief consistency'),nl,!.


%%%% BELIEF GENERALIZATION (R2)
```

```
eval_formula(W,[bel,A,F],R,N):-
                            N0 is N + 2,
                            tab(N0),write('Ejecutando regla: belief
generalization '),nl,
eval_formula(W,F,R,N),nl,
tab(N0+2), write('Resultado de belief
generalization '),write(F),write(' en '),write(W),write(' '),write(R),nl,
tab(N0),write('Fin ejecucion regla:
belief generalization'),nl.


%%%% GOAL DISTRIBUTION (A2d)
eval_formula(W,[goal,A,F2],R,N):-
                           N0 is N + 2,
tab(N0),write('Evaluando axioma:
 goal distribution'),nl,
tab(N0+2),write('Evaluando axioma:
 goal distribution. Formula F1'),nl,
eval_formula(W,[goal,A,_],R,N),
tab(N0+2),write('Evaluando axioma:
 goal distribution. Formula F1=>F2'),nl,
eval_formula(W,[goal,A,[_,=>,F2]],R,N),
tab(N0),write('Fin evaluacion axioma:
 goal generalization'),nl.

%%%% INTENTION DISTRIBUTION (A2i)
eval_formula(W,[int,A,F2],R,N):-
                           N0 is N + 2,
tab(N0),write('Evaluando axioma:
 intention distribution'),nl,
tab(N0+2),write('Evaluando axioma:
 intention distribution. Formula F1'),nl,
eval_formula(W,[int,A,_],R,N),
tab(N0+2),write('Evaluando axioma:
 intention distribution. Formula F1=>F2'),nl,
eval_formula(W,[int,A,[_,=>,F2]],R,N),
tab(N0),write('Fin evaluacion axioma:
 intention distribution'),nl.


%%%% GOAL GENERALIZATION (R2d)
eval_formula(W,[goal,A,F],R,N):-
                            N0 is N + 2,
                            tab(N0),write('Ejecutando regla: goal
generalization '),nl,
eval_formula(W,F,R,N),nl,
tab(N0+2), write('Resultado de goal
generalization '),write(F),write(' en '),write(W),write(' '),write(R),nl,
tab(N0),write('Fin ejecucion regla: goal
```

```
generalization'),nl.

%%%% INTENTION GENERALIZATION (R2I)
eval_formula(W,[int,A,F],R,N):-
                            NO is N + 2,
                            tab(NO),write('Ejecutando regla: intention
generalization '),nl,
eval_formula(W,F,R,N),nl,
tab(NO+2), write('Resultado de intention
generalization '),write(F),write(' en '),write(W),write(' '),write(R),nl,
tab(NO),write('Fin ejecucion regla:
intention generalization'),nl.


%%%% INTENTION CONSISTENCY (A6i)
eval_formula(W,[int,A,false],R,N):-
                            NO is N + 2,
                            tab(NO),write('Evaluando intention false '),nl,
R is 0,
tab(NO),write('Fin evaluacion intention false'),nl,!.

eval_formula(W,[neg,[int,A,false]],R,N):-
                            NO is N + 2,
                            tab(NO),write('Evaluando axioma: intention
consistency '),nl,
R is 0,
tab(NO),write('Fin evaluacion axioma:
intention consistency'),nl,!.


%%% INTERDEPENDENCIAS ENTRE INTENCIONES Y OTRAS ACTITUDES

%%%% POSITIVE INTROSPECTION PARA GOALS (A7db)

eval_formula(W,[bel,A,[goal,A,F]],R,N):-
NO is N + 2,
tab(NO),write('Evaluando axioma:
positive introspection for goals '),nl,
eval_formula(W,[goal,A,F],R,N),nl,
tab(NO+2), write('Resultado de
positive introspection for goals'),write(F),write(' en '),write(W),
                            write(' '),write(R),nl,
tab(NO),write('Fin evaluacion axioma:
positive introspection for goals'),nl,!.


%%%% POSITIVE INTROSPECTION PARA INTENTIONS (A7ib)

eval_formula(W,[bel,A,[int,A,F]],R,N):-
```

```
NO is N + 2,
tab(NO),write('Evaluando axioma: positive
introspection for intentions '),nl,
eval_formula(W,[int,A,F],R,N),nl,
tab(NO+2), write('Resultado de positive
introspection for intentions'),write(F),write(' en '),write(W),write(' '),write(R),nl,
tab(NO),write('Fin evaluacion axioma:
positive introspection for intentions'),nl,!.

%%%% NEGATIVE INTROSPECTION PARA GOALS (A8db)

eval_formula(W,[bel,A,[neg,[goal,A,F]]],R,N):-
NO is N + 2,
tab(NO),write('Evaluando axioma: negative
introspection for goals '),nl,
eval_formula(W,[neg,[goal,A,F]],R,N),nl,
tab(NO+2), write('Resultado de negative
introspection for goals'),write(F),write(' en '),write(W),write(' '),write(R),nl,
tab(NO),write('Fin evaluacion axioma:
negative introspection for goals'),nl,!.


%%%% NEGATIVE INTROSPECTION PARA INTENTIONS (A8ib)

eval_formula(W,[bel,A,[neg,[int,A,F]]],R,N):-
NO is N + 2,
tab(NO),write('Evaluando axioma: negative
introspection for intentions '),nl,
eval_formula(W,[neg,[int,A,F]],R,N),nl,
tab(NO+2), write('Resultado de negative
introspection for intentions'),write(F),write(' en '),write(W),write(' '),write(R),nl,
tab(NO),write('Fin evaluacion axioma:
negative introspection for intentions'),nl,!.

%%%% INTENTIONS IMPLICA GOALS (A9id)
eval_formula(W,[goal,A,F],R,N):-
NO is N + 2,
tab(NO),write('Evaluando axioma: intentions
implies goal '),nl,
eval_formula(W,[int,A,F],R,N),nl,
tab(NO+2), write('Resultado de intentions
implies goal'),write(F),write(' en '),write(W),write(' '),write(R),nl,
tab(NO),write('Fin evaluacion axioma: intentions
implies goal'),nl.


%%% AXIOMAS Y REGLAS PARA OERADORES COLECTIVOS

%%%% COMMON BELIEF (C2)
```

```
eval_formula(W,[c-bel,G,F],R,N):-
NO is N + 2,
tab(NO),write('Evaluando axioma: common belief '),nl,
eval_formula(W,[e-bel,G,[F,&,[c-bel,G,F]]],R,N),nl,
tab(NO),write('Fin evaluacion axioma: common belief'),nl.

eval_formula(W,[e-bel,G,[F,&,[c-bel,G,F]]],R,N):-
NO is N + 2,
tab(NO),write('Evaluando axioma: common belief <='),nl,
eval_formula(W,[c-bel,G,F],R,N),nl,
tab(NO),write('Fin evaluacion axioma: common belief <= '),nl.

%%%% INDUCTION RULE (RC1)
eval_formula(W,[F1,=>,[c-bel,G,F2]],R,N):-
NO is N + 2,
tab(NO),write('Evaluando regla induction rule'),nl,
eval_formula(W,[F1,=>,[e-bel,G,[F2,&,F1]]],R,N),nl,
tab(NO),write('Fin evaluacion evaluando regla
induction rule'),nl.


%%%% MUTUAL INTENTION (M2)
eval_formula(W,[m-int,G,F],R,N):-
NO is N + 2,
tab(NO),write('Evaluando axioma: mutual
intention => '),nl,
eval_formula(W,[e-int,G,[F,&,[m-int,G,F]]],R,N),nl,
tab(NO),write('Fin evaluacion axioma: mutual
intention '),nl.

eval_formula(W,[e-int,G,[F,&,[m-int,G,F]]],R,N):-
NO is N + 2,
tab(NO),write('Evaluando axioma: mutual
intention <='),nl,
eval_formula(W,[m-int,G,F],R,N),nl,
tab(NO),write('Fin evaluacion axioma: mutual
intention '),nl.

%%%% COLLECTIVE INTENTION (M3)
eval_formula(W,[c-int,G,F],R,N):-
NO is N + 2,
tab(NO),write('Evaluando axioma: collective
intention '),nl,
eval_formula(W,[[m-int,G,F],&,[c-bel,G,[m-int,G,F]]],R,N),nl,
tab(NO),write('Fin evaluacion axioma: collective
intention'),nl.

%%%% INDUCTION RULE (RM1)
eval_formula(W,[F1,=>,[m-int,G,F2]],R,N):-
```

```
N0 is N + 2,
tab(N0),write('Evaluando regla induction rule'),nl,
eval_formula(W,[F1,=>,[e-int,G,[F2,&,F1]]],R,N),nl,
tab(N0),write('Fin evaluacion evaluando regla
induction rule'),nl.


%% AXIOMAS PARA EL OPERADOR NO NORMAL DOES
eval_formula(W,[F],R,N):-
N0 is N + 2,
tab(N0),write('Evaluando axioma: Does F => F '),nl,
eval_formula(W,[does,A,F],R,N),nl, %EVALUA SOLO HECHOS
tab(N0),write('Fin evaluacion aaxioma: Does A => A'),nl.

eval_formula(W,[does,A,[F1,&,F2]],R,N):-
N0 is N + 2,
tab(N0),write('Evaluando axioma: (Does F1 & Does F2)=>
Does (F1 & F2) '),nl,
eval_formula(W,[[does,A,F1],&,[does,A,F2]],R,N),
tab(N0),write('Fin evaluacion aaxioma: Does A => A'),nl.

eval_formula(W,[does,A,false],R,N):-
                            N0 is N + 2,
                            tab(N0),write('Evaluando does false '),nl,
R is 0,
tab(N0),write('Fin evaluacion does false'),nl,!.

eval_formula(W,[neg,[does,A,true]],R,N):-
                            N0 is N + 2,
                            tab(N0),write('Evaluando axioma: not(does T) '),nl,
R is 0,
tab(N0),write('Fin evaluacion axioma: not(does T)'),nl,!.



%%% EJEMPLO DE MODELO: FRAME Y FUNCIÓN DE VALUACIÓN %%%%%%%%%%%%%%%%%%%%%%%%

%Definicion de agentes
agentes([a1,a2]).

%Definición de mundos frame.
mundos([w1,w2,w3,w4,w5,w6]).

%Definición de relaciones del frame.
rel_bel(a1,w1,[w2]).
rel_bel(a1,w2,[w3,w4,w5]).
rel_bel(_,[]).

rel_does(a2,w2,[w5,w6]).
```

```
rel_does(_,_,[]).


%Definición de hechos
hecho(w1,[p],1):- !.
hecho(w3,[p],1):- !.
hecho(w4,[p],1):- !.

hecho(w5,[q],1):- !.
hecho(w6,[q],0):- !.
hecho(_,_,0).

% Ejemplo de invocación a la regla eval_formula
% eval_formula(w2,[[a]&[b]],R.N).

%Model checker. Chequea una fórmula F en un modelo.
model_checker([W|Ws],F):-
write('MC: Model checker. Evaluando fórmula: '),
write(F),write(' en '),write(W),nl,
eval_formula(W,F,R,1),
nl,
((R > 0 -> (write('MC: LA FORMULA '),write(F),
write(' ES VERDADERA EN '),write(W),nl);
(write('MC: LA FORMULA '),write(F),
write(' ES FALSA EN '),write(W),nl))),nl,
    model_checker(Ws,F),nl.


check_formula(F):-
mundos(WS),
write('Mundos o estados del modelo: '),write(WS),nl,
write('Evaluando: '),write(F),write(' en los mundos: '),write(WS),nl,
model_checker(WS,F).


% Ejemplo de invocación a la regla check_formula
% check_formula([bel,a1,[p]]).
```

Apéndice B

Código SPINdle

A continuación se detalla el código fuente SPINdle para el ejemplo de la parada de autobús. Para reducir el tamaño del código y de las conclusiones generadas sólo se han implementado los prototipos de axioma necesarios para el ejemplo.

```
# Defeasible theory generated by SPINdle (ver. 2.1.0)

########################################
# symbols used
# ------------
# [set] Literal variable/boolean function
# [>>] Fact
# [->] Strict rule
# [=>] Defeasible rule
# [~>] Defeater
# [>] Superiority relation
# [<] Inferiority relation
# [==] Mode conversion
# [!=] Mode conflict
#
# [-] negation
#
# [==] mode conversion
# [!=] mode conflict
########################################

#-- MODE CONVERSIONS --

#--- MODE CONFLICTS ---
#PROTOTIPE# OP != -OP

DOES_A1 != -DOES_A1
DOES_A2 != -DOES_A2
```

```
DOES_A3 != -DOES_A3
GOAL_A2 != -GOAL_A2
GOAL_A3 != -GOAL_A3
BEL_A2_DOES_A1 != BEL_A2_-DOES_A1
BEL_A3_DOES_A1 != BEL_A3_-DOES_A1
INT_A2_DOES_A1 != INT_A2_-DOES_A1
INT_A3_DOES_A1 != INT_A3_-DOES_A1
INT_A2_DOES_A2 != INT_A2_-DOES_A2
INT_A3_DOES_A3 != INT_A3_-DOES_A3
BEL_A2_INT_A1 != BEL_A2_-INT_A1
BEL_A3_INT_A1 != BEL_A3_-INT_A1
GOAL_A2_INT_A1 != GOAL_A2_-INT_A1
GOAL_A3_INT_A1 != GOAL_A3_-INT_A1

#--- BEL AXIOMS ---
#PROTOTYPE# A2 [BEL]: [BEL] Phi(X) -> Psi(X)
#PROTOTYPE# A4: [BEL_X] Phi(X) -> [BEL_X_BEL_X] Phi(X)
#PROTOTYPE# A5: [-BEL_X] Phi(X) -> [BEL_X_-BEL_X] Phi(X)
#PROTOTYPE# A6: [-BEL_X] BOTTOM
#PROTOTYPE# R2: Phi(X) -> [BEL_X] Phi(X)
#PROTOTYPE# G={X1, X2, ..., XN} C1: [BEL_X1] Phi(X), [BEL_X2] Phi(X),
#                                ..., [BEL_XN] Phi(X) -> [E-BEL_G] Phi(X)
#PROTOTYPE# G={X1, X2, ..., XN} C2: [E-BEL_G] Phi(X),
#                           [E-BEL_G_C-BEL_G] Phi(X) -> [C-BEL_G] Phi(X)

db7 : [DOES_A1] ParарBus ->  [BEL_A2_DOES_A1] ParарBus  #R2
db13 : [DOES_A1] ParарBus ->  [BEL_A3_DOES_A1]ParарBus  #R2

ib7 : [INT_A1] ParарBus -> [BEL_A2_INT_A1] ParарBus  #R2
ib13 : [INT_A1] ParарBus -> [BEL_A3_INT_A1] ParарBus  #R2

#--- GOAL AXIOMS ---
#PROTOTYPE# A2D[GOAL]: [GOAL] Phi(X) -> Psi(X)
#PROTOTYPE# R2D: Phi(X) -> [GOAL_X] Phi(X)
#PROTOTYPE# A7DB:[GOAL_X] Phi(X) -> [BEL_X_GOAL_X] Phi(X)
#PROTOTYPE# A8DB:[-GOAL_X] Phi(X) -> [BEL_X_-GOAL_X] Phi(X)

ig7 : [INT_A1] ParарBus -> [GOAL_A2_INT_A1] ParарBus  #R2D
ig13 : [INT_A1] ParарBus -> [GOAL_A3_INT_A1] ParарBus  #R2D

#--- INT AXIOMS ---
#PROTOTYPE# A2I [INT]: [INT] Phi(X) -> Psi(X)
#PROTOTYPE# R2I: Phi(X) -> [INT_X] Phi(X)
#PROTOTYPE# A6I: [-INT_X] BOTTOM
#PROTOTYPE# A7IB:[INT_X] Phi(X) -> [BEL_X_INT_X] Phi(X)
#PROTOTYPE# A8IB:[-INT_X] Phi(X) -> [BEL_X_-INT_X] Phi(X)
#PROTOTYPE# A9IB:[INT_X] Phi(X) -> [GOAL_X] Phi(X)
#PROTOTYPE# G={X1, X2, ..., XN} M1: [INT_X1] Phi(X), [INT_X2] Phi(X),
#                                ..., [INT_XN] Phi(X) -> [E-INT_G] Phi(X)
```

```
#PROTOTYPE# G={X1, X2, ..., XN} M2: [E-INT_G] Phi(X),
#                           [E-INT_G_M-INT_G] Phi(X) -> [M-INT_G] Phi(X)
#PROTOTYPE# G={X1, X2, ..., XN} M3: [M-INT_G] Phi(X),
#                           [C-BEL_G_M-INT_G] Phi(X) -> [C-INT_G] Phi(X)


di7 : [DOES_A1] PararBus -> [INT_A2_DOES_A1] PararBus  #R2I
di8: [DOES_A2] PararBus -> [INT_A2_DOES_A2] PararBus  #R2I
di11 : [-DOES_A2] PararBus -> [INT_A2_-DOES_A2] PararBus #R2I
di13 : [DOES_A1] PararBus -> [INT_A3_DOES_A1] PararBus  #R2I
di15: [DOES_A3] PararBus -> [INT_A3_DOES_A3] PararBus  #R2I
di18 : [-DOES_A3] PararBus -> [INT_A3_-DOES_A3] PararBus #R2I


#--- DOES AXIOMS ---
#PROTOTYPE#  E1: [DOES_X] Phi(X) -> Phi(X)
#PROTOTYPE#  E3: [DOES_X] Phi(X), [DOES_X] Psi(X) -> [DOES_X] Phi_and_Psi(X)
#PROTOTYPE# E2: [-DOES_X] _TOP

[DOES_A1] PararBus -> PararBus
[DOES_A2] PararBus -> PararBus
[DOES_A3] PararBus -> PararBus

#--- TRUST AXIOMS ---
#PROTOTYPE# [GOAL_X]Phi(X), [BEL_X_DOES_Y]Phi(X), [INT_X_DOES_X]Phi(X),
#                                  [-INT_X_DOES_Y]Phi(X),[BEL_X_INT_Y]Phi(X),
#                                      [GOAL_X_INT_Y]Phi(X), -> [TRUST_X_Y]Phi(X)

[GOAL_A2] PararBus, [BEL_A2_DOES_A1] PararBus, [INT_A2_DOES_A1] PararBus,
#                       [INT_A2_-DOES_A2] PararBus, [BEL_A2_INT_A1] PararBus,
#                               [GOAL_A2_INT_A1] PararBus -> [TRUST_A2_A1] PararBus

[GOAL_A3] PararBus, [BEL_A3_DOES_A1] PararBus, [INT_A3_DOES_A1] PararBus,
#                       [INT_A3_-DOES_A3] PararBus, [BEL_A3_INT_A1] PararBus,
#                               [GOAL_A3_INT_A1] PararBus -> [TRUST_A3_A1] PararBus

#--- JTRUST AXIOMS ---
#PROTOTIPE# G={X1, X2, ..., XN} [TRUST_X1_Y] Phi(X), [TRUST_X2_Y] Phi(X),
#                                  ..., [TRUST_XN_Y] Phi(X) -> [JTRUST_G_Y] Phi(X)

[TRUST_A2_A1] PararBus, [TRUST_A3_A1] PararBus -> [JTRUST_G_A1] PararBus

#-----------------------------------------------------------------
# AGENTS = { A1, A2, A3 }
# GROUP = { A1, A2, A3 }

#--- AGENT A3
>>[GOAL_A3] PararBus
>>[-DOES_A3] PararBus
```

```
#--- AGENT A2 ---
>>[GOAL_A2] PararBus
>>[-DOES_A2] PararBus
#--- AGENT A1 ---
>>[INT_A1] PararBus
>>[DOES_A1] PararBus
```

Las conclusiones deducidas del código se detallan a continuación junto con una breve explicación:

```
Conclusions
===========
+D PararBus(X)
+D [-DOES_A2]PararBus(X)
+D [-DOES_A3]PararBus(X)
+D [BEL_A2_DOES_A1]PararBus(X)
+D [BEL_A2_INT_A1]PararBus(X)
+D [BEL_A3_DOES_A1]PararBus(X)
+D [BEL_A3_INT_A1]PararBus(X)
+D [DOES_A1]PararBus(X)
+D [GOAL_A2]PararBus(X)
+D [GOAL_A2_INT_A1]PararBus(X)
+D [GOAL_A3]PararBus(X)
+D [GOAL_A3_INT_A1]PararBus(X)
+D [INT_A1]PararBus(X)
+D [INT_A2_-DOES_A2]PararBus(X)
+D [INT_A2_DOES_A1]PararBus(X)
+D [INT_A3_-DOES_A3]PararBus(X)
+D [INT_A3_DOES_A1]PararBus(X)
+D [JTRUST_G_A1]PararBus(X) ****************************
+D [TRUST_A2_A1]PararBus(X)
+D [TRUST_A3_A1]PararBus(X)
-D [-DOES_A1]PararBus(X)
-D [BEL_A2_-DOES_A1]PararBus(X)
-D [BEL_A3_-DOES_A1]PararBus(X)
-D [DOES_A2]PararBus(X)
-D [DOES_A3]PararBus(X)
-D [INT_A2_DOES_A2]PararBus(X)
-D [INT_A3_DOES_A3]PararBus(X)
+d PararBus(X)
+d [-DOES_A2]PararBus(X)
+d [-DOES_A3]PararBus(X)
+d [BEL_A2_DOES_A1]PararBus(X)
+d [BEL_A2_INT_A1]PararBus(X)
+d [BEL_A3_DOES_A1]PararBus(X)
+d [BEL_A3_INT_A1]PararBus(X)
+d [DOES_A1]PararBus(X)
+d [GOAL_A2]PararBus(X)
```

```
+d [GOAL_A2_INT_A1]PararBus(X)
+d [GOAL_A3]PararBus(X)
+d [GOAL_A3_INT_A1]PararBus(X)
+d [INT_A1]PararBus(X)
+d [INT_A2_-DOES_A2]PararBus(X)
+d [INT_A2_DOES_A1]PararBus(X)
+d [INT_A3_-DOES_A3]PararBus(X)
+d [INT_A3_DOES_A1]PararBus(X)
+d [JTRUST_G_A1]PararBus(X)
+d [TRUST_A2_A1]PararBus(X)
+d [TRUST_A3_A1]PararBus(X)
-d [-DOES_A1]PararBus(X)
-d [BEL_A2_-DOES_A1]PararBus(X)
-d [BEL_A3_-DOES_A1]PararBus(X)
-d [DOES_A2]PararBus(X)
-d [DOES_A3]PararBus(X)
-d [INT_A2_DOES_A2]PararBus(X)
-d [INT_A3_DOES_A3]PararBus(X)
```

Observarmos en la linea resaltada, la conclusión `+D [JTRUST_G_A1]PararBus(X)`, esto indica que el grupo de agentes confía conjuntamente en el agente `A1` para que él detenga el autobús. Esto es porque el agente `A1` tiene la intención de detener el autobús (`[INT_A1] PararBus`) y en efecto este lo detiene (`[DOES_A1] PararBus`).

Si en otro caso el agente `A1` mientras que está en la parada de autobús no tiene intención de detenerlo (`[-INT_A1] PararBus`) y tampoco lo detiene (`[-DOES_A1] PararBus`) entonces los agentes pertenecientes al grupo no confiará mas en él. Reemplazando en el código anterior estas líneas:

```
#--- AGENT A1 ---
>>[-INT_A1] ParарBus
>>[-DOES_A1] PararBus
```

volvemos a ejecutar el razonador para obtener nuevas conclusiones:

```
Conclusions
===========
+D [-DOES_A1]PararBus(X)
+D [-DOES_A2]PararBus(X)
+D [-DOES_A3]PararBus(X)
+D [-INT_A1]PararBus(X)
+D [BEL_A2_-DOES_A1]PararBus(X)
+D [BEL_A3_-DOES_A1]PararBus(X)
+D [GOAL_A2]PararBus(X)
+D [GOAL_A3]PararBus(X)
+D [INT_A2_-DOES_A2]PararBus(X)
```

```
+D [INT_A3_-DOES_A3]PararBus(X)
-D PararBus(X)
-D [BEL_A2_DOES_A1]PararBus(X)
-D [BEL_A2_INT_A1]PararBus(X)
-D [BEL_A3_DOES_A1]PararBus(X)
-D [BEL_A3_INT_A1]PararBus(X)
-D [DOES_A1]PararBus(X)
-D [DOES_A2]PararBus(X)
-D [DOES_A3]PararBus(X)
-D [GOAL_A2_INT_A1]PararBus(X)
-D [GOAL_A3_INT_A1]PararBus(X)
-D [INT_A1]PararBus(X)
-D [INT_A2_DOES_A1]PararBus(X)
-D [INT_A2_DOES_A2]PararBus(X)
-D [INT_A3_DOES_A1]PararBus(X)
-D [INT_A3_DOES_A3]PararBus(X)
-D [JTRUST_G_A1]PararBus(X) ********************************
-D [TRUST_A2_A1]PararBus(X)
-D [TRUST_A3_A1]PararBus(X)
+d [-DOES_A1]PararBus(X)
+d [-DOES_A2]PararBus(X)
+d [-DOES_A3]PararBus(X)
+d [-INT_A1]PararBus(X)
+d [BEL_A2_-DOES_A1]PararBus(X)
+d [BEL_A3_-DOES_A1]PararBus(X)
+d [GOAL_A2]PararBus(X)
+d [GOAL_A3]PararBus(X)
+d [INT_A2_-DOES_A2]PararBus(X)
+d [INT_A3_-DOES_A3]PararBus(X)
-d PararBus(X)
-d [BEL_A2_DOES_A1]PararBus(X)
-d [BEL_A2_INT_A1]PararBus(X)
-d [BEL_A3_DOES_A1]PararBus(X)
-d [BEL_A3_INT_A1]PararBus(X)
-d [DOES_A1]PararBus(X)
-d [DOES_A2]PararBus(X)
-d [DOES_A3]PararBus(X)
-d [GOAL_A2_INT_A1]PararBus(X)
-d [GOAL_A3_INT_A1]PararBus(X)
-d [INT_A1]PararBus(X)
-d [INT_A2_DOES_A1]PararBus(X)
-d [INT_A2_DOES_A2]PararBus(X)
-d [INT_A3_DOES_A1]PararBus(X)
-d [INT_A3_DOES_A3]PararBus(X)
-d [JTRUST_G_A1]PararBus(X)
-d [TRUST_A2_A1]PararBus(X)
-d [TRUST_A3_A1]PararBus(X)
```

Aquí podemos ver en la linea resaltada como ahora la deducción deja de ser probable cuando el agente `A1` actúa de manera diferente.

Bibliografía

[ABGM01] Grigoris Antoniou, David Billington, Guido Governatori, and Michael Maher. Representation results for defeasible logic. *ACM Transactions on Computational Logic*, 2:255–287, 2001.

[AMdNdR99] C. Areces, C. Monz, H. de Nivelle, and M. de Rijke. The guarded fragment: Ins and outs. In J. Gerbrandy, M. Marx, M. de Rijke, and Y. Venema, editors, *JFAK. Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday*. Vossiuspers, AUP, Amsterdam, 1999. Non refereed.

[BAV04] Nick Bassiliades, Grigoris Antoniou, and Ioannis Vlahavas. A defeasible logic reasoner for the semantic web. In *In Proc. of the Workshop on Rules and Rule Markup Languages for the Semantic Web*, pages 49–64, 2004.

[BdRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge. 2001.

[Bra87] Michael Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8:244–263, 1986.

[Che80] Brian F. Chellas. *Modal Logic: An Introduction*. Cambridge University Press, 1980.

[CM03] William F. Clocksin and Christopher S. Mellish. *Programming in Prolog*. Springer, Berlin, 5. edition, 2003.

[DKV02] Barbara Dunin-Keplicz and Rineke Verbrugge. Collective intentions. *Fundam. Inform.*, pages 271–295, 2002.

[DVDK07] Marcin Dziubinski, Rineke Verbrugge, and Barbara Dunin-Keplicz. Complexity issues in multiagent logics. *Fundam. Inform.*, 75(1-4):239–262, 2007.

[Elg97] Dag Elgesem. The modal logic of agency. *Nordic Journal of Philosophical Logic*, 2:1–46, 1997.

[Fer01] Jacques Ferber. Multi-agent system: An introduction to distributed artificial intelligence. *J. Artificial Societies and Social Simulation*, 4(2), 2001.

[FG94] Marcelo Finger and Dov Gabbay. Combining temporal logic systems. *Notre Dame Journal of Formal Logic*, 37, 1994.

[FMDR04] Massimo Franceschet, Angelo Montanari, and Maarten De Rijke. Model checking for combined logics with an application to mobile systems. *Automated Software Engg.*, 11:289–321, June 2004.

[FndC86] L Fariñas del Cerro. Molog: A system that extends prolog with modal logic. *New Gen. Comput.*, 4:35–50, January 1986.

[GHP09] Guido Governatori, John Hall, and Adrian Paschke, editors. *Proceedings of the 2009 International Symposium on Rule Interchange and Applications (RuleML-2009)*, volume 5858 of *Lecture Notes in Computer Science*. Springer, 2009.

[Gov05] Guido Governatori. Representing business contracts in *uleml*. *Int. J. Cooperative Inf. Syst.*, 14(2-3):181–216, 2005.

[GR04a] Guido Governatori and Antonino Rotolo. Defeasible logic: Agency, intention and obligation. In *Deontic Logic in Computer Science, number 3065 in LNAI*, pages 114–128. Springer, 2004.

[GR04b] Guido Governatori and Antonino Rotolo. On the axiomatization of elgesem's logic of agency. In *AiML 2004-Advances in Modal Logic*, pages 130–144. Department of Computer Science, University of Manchester, 2004.

[GR08] Guido Governatori and Antonino Rotolo. Bio logical agents: Norms, beliefs, intentions in defeasible logic. *Autonomous Agents and Multi-Agent Systems*, 17:36–69, 2008. 10.1007/s10458-008-9030-4.

[GSMCM07] Mario Gomez, J Sabater-Mir, Javier Carbo, and Guillaume Muller. *Improving the art testbed, thoughts and reflections*, pages 1–15. Citeseer, 2007.

[Hal73] Sören Halldén. *Modality, Morality and Other Problems of Sense and Nonsense.* Lund,Gleerup, 1973.

[Ham78] A. G. Hamilton. *Logic for Mathematicians.* Cambridge University Press, 1978.

[HG73] Bengt Hansson and Peter Gärdenfors. A guide to intensional semantics. Modality, Morality and Other Problems of Sense and Nonsense. Essays Dedicated to Sören Halldén. 1973.

[HM92] J. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:311–379, 1992.

[Lam10] Ho-Pun Lam. Spindle - user guide. Technical report, NICTA QRL, Australia, 2010.

[Llo84] J. W. Lloyd. *Foundations of logic programming.* Springer-Verlag New York, Inc., New York, NY, USA, 1984.

[Ngu04] Linh Anh Nguyen. The modal logic programming system MProlog. In José Júlio Alferes and João Alexandre Leite, editors, *Proceedings of JELIA 2004, LNCS 3229*, pages 266–278. Springer, 2004.

[Ngu09] Linh Anh Nguyen. Modal logic programming revisited. *Journal of Applied Non-Classical Logics*, 19(2):167–181, 2009.

[Nut01] Donald Nute. Defeasible logic. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, pages 353–395. Oxford University Press, 2001.

[RN03] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: a modern approach.* Prentice Hall, 2nd international edition edition, 2003.

[SP08] Lutz Schröder and Dirk Pattinson. The craft of model making: Pspace bounds for non-iterative modal logics. *CoRR*, abs/0802.0116, 2008.

[SR10] Clara Smith and Antonino Rotolo. Collective trust and normative agents. *Logic Journal of IGPL*, 18(1):195–213, 2010.

[vdHV02] Wiebe van der Hoek and Rineke Verbrugge. Epistemic logic: A survey, 2002.

[Wol99] Frank Wolter. The product of converse pdl and polymodal k. *JOURNAL OF LOGIC AND COMPUTATION*, 10:2000, 1999.