BUAP

Benemérita Universidad
Autónoma de Puebla

Facultad de Ciencias Físico Matemáticas

Interrelación de algunas lógicas intermedias y Answer Set

Tesis

Presentada para obtener el título de
Maestría en Ciencias Matemáticas

Presenta:
L. C. José Alfonso del Carmen Garcés Báez

Asesor:
Dr. José Enrique Arrazola Ramírez

Marzo de 2003

Contents

Chapter 1

Introducción

1.1 Negación clásica y negación por falla

La *negación clásica* $-F$ no es parte de la programación lógica estandar ni de las bases de datos deductivas. Algunas razones por las que la negación clásica no es adecuada para los formalismos no monotónicos, son:

- La negación clásica $-F$ satisface la "ley del tercero excluido", $F \vee -F$ para toda fórmula F. Este es un teorema natural en el dominio de la lógica formal pero no es adecuado para el razonamiento con sentido común donde nos encontramos con situaciones en las que algunas proposiciones son verdaderas o falsas y algunas otras que no se pueden determinar si son verdaderas o falsas. Por ejemplo, supongamos que para reclutar personal para una empresa, la persona encargada de hacerlo aplica examenes y determina que algunos candidatos son claramente calificados para el trabajo y son aceptados, otros que claramente no son calificados para el trabajo y son rechazados. Lo cual podemos describir con las siguientes dos clausulas:

$aceptado(x) \leftarrow calificado(x)$ rechazado(x)$\leftarrow -calificado(x)$

pero hay otros candidatos cuyas calificaciones no son claras y que deben ser entrevistados y así determinar su condición para el trabajo. Esta situación nos obliga a tratar con *información incompleta* o con propiedades que involucran *cambio gradual* en las que ni una propiedad ni su opuesta se cumplen.

- Supongamos que Juan es un candidato para ser contratado en un empleo y necesita ser entrevistado, es decir que tanto $not\,(calificado(Juan))$ como $not\,(-calificado(Juan))$ se cumplen, en otras palabras, la *negación*

por falla o *not*, indica por un lado que no hay "evidencia suficiente" de que Juan esté calificado para el trabajo y por el otro que no hay "evidencia suficiente" de que Juan no esté calificado para el trabajo. Esta situación no puede ser tratada con la negación clásica $-F$ porque la ley del tercero excluido no permite minimizar F o $-F$ ya que uno de ellos debe ser siempre verdad. *El caracter de la ley del tercero excluido presupone que se tiene información para determinar cuando una proposición dada o su opuesta es cierta.*

En la programación lógica, en bases de datos deductivas y en general en las teorías no-monotonicas, se usan varias formas de *negación por falla*, *not* A, cuyo principal distintivo es el hecho de que *not* A se infiere por la ausencia de "evidencia suficiente" para apoyar la fórmula atómica A. El significado de "evidencia suficiente" depende de la semántica específica utilizada. Por ejemplo, en la *suposición del mundo cerrrado* (CWA) de Reiter *not* A se infiere si A no es demostrable, o, equivalentemente si hay un modelo mínimo en el que A es falso. En la *suposición del mundo cerrado generalizada* (GCWA) de Minker o en *circunscripción* de McCarthy *not* A se infiere si y sólo si A es falso en todos los modelos mínimos. En las semánticas completas de predicados para programas lógicos esta forma de negación es llamada *negación por falla,* porque *not* A es inferido si el intento por probar A falla. Esto no es algo justificable por la negación clásica, al menos no en programas normales. Por ejemplo, supongamos el programa P: $A \leftarrow not\ B$ entonces la prueba de B falla, así *not* B sucede y A sucede, pero A no es una consecuencia lógica del programa P.

La clausula: $absuelto(x) \leftarrow tiene\_cargo(x), not\ culpable(x)$ indica que uno puede ser absuelto de algún cargo a menos que se pruebe la culpabilidad, en otras palabras, a menos que la "evidencia suficiente" de su culpabilidad sea demostrada.

Las semánticas propuestas para programas lógicos y bases de datos deductivas tales como las semánticas estacionarias o de punto fijo, semánticas bien formadas y semánticas estables proponen cada vez más significados sofisticados para la negación por falla, enmarcada en formalismos no-monotónicos más generales como Lógica Default, Lógica Autoepistémica y Lógica Autoepistémica de Creencias.

Aunque la negación por falla es de mucha utilidad en varios dominios y aplicaciones de programación lógica, ésta no es suficiente para el tratamiento

de la no-monotonicidad. A continuación se presentan dos razones importantes por las que no sólo esta forma de la negación es necesaria en formalismos no-monotónicos:

a). La negación *not* A de una fórmula atómica A es siempre supuesta "por falla", sin embargo, se requiere algo más para poder llegar a conclusiones negativas. Por ejemplo, si decimos *culpable(x)*←*not inocente(x)* para expresar el hecho de que culpable es opuesto de inocente, estaríamos implicando que la gente es considerada culpable porque no se puede probar su inocencia, es decir, que todos somos culpables por omisión. Esta aseveración es incorrecta ya que no podemos suponer que la información disponible para probar la inocencia de una persona sea completa.

b). Pensemos que la negación *not* F es siempre supuesta "por falla" para la fórmula atómica F, lo que es lo mismo, no es verdad para el átomo negado $F = -A$, así la negación por falla no trata las literales A y $-A$ simétricamente. Por ejemplo, supongamos que aplicamos GCWA a la siguiente teoría::

tiene_cargo(Juan).
sentenciado(x)← *tiene_cargo*$(x)\wedge$*culpable(x).*

de donde obtenemos: *not sentenciado(Juan).* Ahora, si reemplazamos culpable por $-$*inocente*, tenemos:

tiene_cargo(Juan).
sentenciado(x) ←*tiene_cargo(x)*$\wedge -$ *inocente*(x).

La nueva teoría tiene modelos mínimos para los que *sentenciado*$(Juan)$ es actualmente verdad.

Como se puede ver, el reemplazo de *culpable* por $-$*inocente* tiene un impacto significante en la semántica resultante de la teoría . Como una consecuencia, *not* A, no es invariante bajo equivalencia simple o con el renombramiento de predicados.

1.2 Negación fuerte

Además de la negación por falla y de la negación clásica, es necesaria una negación que permita el tratamiento simétrico de información positiva y negativa.

Otro tipo de negación es la conocida como *negación explícita* o *fuerte* donde $\tilde{}A$ se interpreta como "$\tilde{}A$ *es opuesto de* A" por la suposición del axioma $A \wedge \tilde{}A \rightarrow \bot$, o equivalentemente $\tilde{}A \rightarrow not\ A$, lo cual indica que A y su opuesta $\tilde{}A$ no pueden ser ambos verdad.

En particular, la negación fuerte agregada a programas lógicos generales con semánticas stable coinciden con la llamada "negación clásica", originalmente introducida por Gelfond-Lifschitz [13].

1.3 Expresiones anidadas

En muchos artículos sobre semánticas de la programación lógica, el cuerpo de una regla es considerado como una lista de expresiones sintacticamente simples, tales como átomos o literales. La sintaxis de Prolog, sin embargo, permite expresiones anidadas. Junto con la conjunción $(p \wedge q\ )$ y la negación como falla $(not\ p)$ se pueden incluir disyunciones $(p \vee q)$ anidadas arbitrariamente [6].

En las expresiones anidadas, el operador de la negación clásica $-$, es diferente de los operadores "$\wedge$", "$\vee$" y *not*. La negación clásica es permitida solamente frente a un átomo.

Consideremos las siguientes definiciones de [6]:

♦ *Atomo* como es definido en el cálculo proposicional.

♦ *Literal*, es un átomo posiblemente precedido por el signo de la negación clásica $-$.

♦ *Fórmulas elementales*, son literales y también los conectivos 0-arios $\bot\ (falso)$ y $\top\ (verdadero)$.

♦ *Fórmulas*, son construidas a partir de fórmulas elementales usando el conectivo unario *not* y los conectivos binarios $\wedge$ (conjunción) y $\vee$ (disyunción).

♦ *Regla*, es una expresión de la forma $F \leftarrow G$ donde F y G son fórmulas llamadas cabeza y cuerpo de la regla respectivamente. La regla de la forma $F \leftarrow \top$ se escribe como $F \leftarrow$ y se indentifica con la fórmula F. Las reglas de la forma $\bot \leftarrow G$ son llamadas restricciones y se escriben como $\leftarrow G$.

♦ *Programa* es un conjunto de reglas.

♦ La *ocurrencia* de una fórmula dentro de otra fórmula o regla es *singular* si el símbolo antes de esta ocurrencia es $-$. En otro caso la ocurrencia es *regular*. Por ejemplo, en $not\ -p$, la ocurrencia de p es singular y en $not\ p$ es regular.

♦ Las fórmulas, reglas y programas que no contienen el operador de negación como falla *not*, son llamados *básicos*.

♦ Un conjunto consistente de literales X, *satisface* una fórmula básica F (simbólicamente, $X \models F$) recursivamente, como sigue:

- Para F elemental, $X \models F$ si $F \in X$ o $F = \top$.
- $X \models (F, G)$ si $X \models F$ y $X \models G$.
- $X \models (F; G)$ si $X \models F$ o $X \models G$.

♦ Sea Π un programa básico. Un conjunto consistente de literales X es *cerrado* bajo Π si, para cada regla $F \leftarrow G$ en Π, $X \models F$ cuando $X \models G$.

♦ Decimos que X es un *answer set* para el programa básico Π si X es la cantidad mínima del conjunto consistente de literales cerrado bajo Π. Por ejemplo, considerémos el programa: $q \leftarrow p \vee -p$. La cerradura bajo este programa es caracterizada por la siguiente condición: si $p \in X$ o $-p \in X$ entonces $q \in X$. Es claro que el answer set para dicho programa es vacío. Si agregamos la regla p (que es, $p \leftarrow \top$) a este programa entonces $\{p, q\}$ serán los answer set.

♦ La *reducción* de una fórmula, regla o programa Π, relativo a un conjunto consistente X de literales es definido recursivamente, como sigue:

- Para F elemental, $F^X = F$.
- $(F \wedge G)^{\,X} = \left(F^X \wedge G^X\right)$.
- $(F \vee G)^X = \left(F^X \vee G^X\right)$.
- $(not\ F)^X = \{\bot$, si $X \models F^X.\top$, en otro caso.$\}$
- $(F \leftarrow G)^X = F^X \leftarrow G^X$.
- $\Pi^X = \left\{(F \leftarrow G)^X : F \leftarrow G \in \Pi\right\}$.

♦ Un conjunto consistente X de literales es un *answer set* para un programa Π si éste es un answer set para la reducción Π^X.

1.4 Caracterización del presente trabajo

La negación dentro de la programación lógica ha jugado un papel sumamente importante, en el presente documento se presentan algunos resultados recientes en este campo basados en el trabajo [1]. Así mismo, en la parte final se propone y presenta una forma de obtener información denominada *El que calla otorga* basada en la interpretación del silencio.

Answer Set Programming (ASP), también conocida como Stable Logic Programming o A-Prolog, es la realización de mucho trabajo teórico en aplicaciones de Programación Lógica (LP) para Razonamiento Nomotónico e Inteligencia Artificial en los últimos 15 años. La principal restricción sintáctica necesaria en este paradigma es la eliminación de símbolos funcionales del lenguaje. Esto es porque usando dominios infinitos de modelos stable, éstos no son necesariamente recursivamente numerables. Los dos sistemas más conocidos que computan Answer Sets son *dlv*[1] y *smodels*[2].

Con la primera parte de este trabajo se intenta dar un punto de vista alternativo de la teoría de Answer Set Programming a través de diferentes herramientas y relaciones con lógica intuicionista y otras lógicas intermedias. Una caracterización de ASP por lógica intuicionista es:

Una fórmula está vinculada a un programa en la semánticas de answer set si y solamente si pertenece a cada extensión consistente y completa desde el punto de vista intuicionista, del programa que se forma al agregar sólo literales negadas.

Esto es, una generalización de un resultado dado recientemente por D. Pearce donde considera solamente programas disyuntivos. En esta aplicación se considera la clase de programas aumentados, es decir, programas que permiten fórmulas o expresiones con anidaciones de $\neg$, $\wedge$ y $\vee$ en la cabeza y en el cuerpo de las reglas [4]. Erdem y Lifschitz dan evidencia para la aplicación de programas aumentados en la representación y solución de problemas reales.

En este trabajo se dan los fundamentos para definir la noción de inferencia nomonotónica de una teoría proposicional usando los conectivos estandar $\{\neg, \wedge, \vee, \rightarrow\}$ en términos de una lógica monotónica llamada lógica intuicionista. Se propone la siguiente interpretación: Dada una teoría T, dicho *conocimiento* se entiende como las fórmulas F tales que F es derivado en T usando lógica intuicionista. Esto tiene sentido porque de acuerdo a Brouwer, A es identificado con "Yo conozco A". También se identifica un conjunto de *creencias* para la teoría T. Un agente cuya base de conocimiento es la teoría T cree F si y sólo si F pertenece a cada extensión intuicionisticamente completa y consistente de T, obtenida al agregar sólo literales negadas. Por ejemplo en la instancia $\neg a \rightarrow b$ el agente *conoce* $\neg a \rightarrow b, \neg b \rightarrow \neg\neg a$.

[1] *http://www.dbai.tuwien.ac.at/proj/dlv/*
[2] *http://saturn.hut.fi/pub/smodels/*

Sin embargo el agente *no conoce* b. Por lo regular, se cree más de lo que se conoce. Pero un agente cauteloso debe tener creencias consistentes a su conocimiento. Este agente supondrá literales negadas para inferir más información. Así, en el ejemplo, el agente creerá $\neg a$ y puede concluir b. Esto tiene sentido ya que un agente cauteloso creerá $\neg a$ o $\neg\neg a$ antes que creer a, recordemos que a no es equivalente a $\neg\neg a$ en lógica intuicionista. Este resultado concuerda con la posición de Kowalski, a saber:

La lógica y la programación lógica necesitan tomar su lugar: la lógica dentro de la componente de conocimiento en el ciclo observación-pensamiento-acción de un agente simple, y la programación lógica dentro de la componente creencia del pensamiento [5].

Un hecho importante que se quiere resaltar es cuando dos programas son "equivalentes" con respecto a las semánticas answer set. Consideramos una definición para "equivalencia", decimos que dos programas P_1 y P_2 son equivalentes sii poseen los mismos answer set. Ademas, diremos que P_1 y P_2 son fuertemente equivalentes si y sólo si para cada programa $P, P_1 \cup P$ y $P_2 \cup P$ tienen los mismos answer sets. Si dos programas son fuertemente equi-valentes sabemos que puede ser reemplazado uno por el otro en algún programa grande sin cambiar la semántica declarativa. Este concepto es importante para la *ingeniería del software,* en particular en reutilización de código y migración de sistemas. Es conocido que la lógica HT (here-and-there) o G_3 caracteriza la clase de programas aumentados fuertemente equivalentes bajo esta definición.

Debido a que hay mucho más investigación y software disponible en lógica clásica y lógica intuicionista se presentan algunos resultados de equivalencia fuerte con respecto a esas lógicas. Los resultados particulares son los siguientes: Dados dos programas aumentados P_1 y P_2, se muestra que hay un algoritmo computable en tiempo lineal que construye una fórmula F (de tamaño lineal con respecto a $P_1 \cup P_2$) tal que P_1 es fuertemente equivalente a P_2 si y sólo si F es un teorema en lógica intuicionista. Dado un programa P aumentado y un átomo negado $\neg a$, entonces P es fuertemente equivalente a $P \cup \{\neg a\}$ si y solo si $\neg a$ puede ser probado de P usando logica clásica. Dado un programa disyuntivo P y un átomo a, se muestra que P es fuertemente equivalente a $P \cup \{a\}$ si y sólo si a puede ser probado con lógica intuicionista del fragmento positivo de P. También presentamos una reducción lineal de programas libres a programas generales.

Los modelos mínimos son de interés general por varias razones teóricas y prácticas, dedicamos aquí una sección para estudiarlos. Se presenta una caracterización de modelos stable y mínimos en términos de lógica intuicionista. También se presenta que dos programas son fuertemente equivalentes con respecto a su modelo stable y mínimo si y solo si son equivalentes en la lógica trivaluada G_3.

Enfocamos nuestra atención en la teoría proposicional finita: las semánticas pueden ser extendidas a teorías con variables y sin símbolos funcionales para estar seguros que un programa ground será finito. Esto es, un procedimiento estandar en ASP.

Chapter 2

Lógica proposicional

Toda teoría tiene sus propios ingredientes, a continuación se describen los que componen la lógica proposicional.

El lenguaje de la lógica proposicional tiene un alfabeto que consiste de
Símbolos proposicionales: $p_0, p_1, ...$ Conectivos: $\vee, \wedge, \leftarrow, \bot$
Símbolos auxiliares: $(,)$.

Donde $\vee, \wedge, \leftarrow$ son conectivos binarios y $\bot$ es un conectivo 0-ario, los símbolos proposicionales son llamados también átomos o proposiciones atómicas. Las fórmulas y teorías son definidas como se usan en lógica. La fórmula $\neg F$ es introducida como una abreviación de $\bot \leftarrow F$, y $F \equiv G$ como una abreviación de $(G \leftarrow F) \wedge (F \leftarrow G)$. La fórmula $F \rightarrow G$ es otra manera de escribir $G \leftarrow F$.

Un alfabeto $\mathcal{L}$ es un conjunto finito de símbolos proposicionales. Si F es una fórmula entonces el alfabeto de F, denotada por $\mathcal{L}_F$, es el conjunto de símbolos proposicionales que ocurren en F. Una literal es un átomo a, literal positiva, o la negación de un átomo $\neg a$, una literal negativa. Una literal negada es el signo de la negación $\neg$ seguido por alguna literal, es decir, $\neg a$ o $\neg\neg a$. El símbolo $\odot$ denota un conectivo perteneciente a $\{\wedge, \vee\}$. F es una fórmula de la forma $l_1 \odot l_2 \odot ... \odot l_n$, donde cada l_i es una literal, se denota por $Lit(F)$ el conjunto de literales $\{l_1, l_2, ..., l_n\}$. Dado un conjunto de fórmulas $\mathcal{F}$, definimos $\neg\mathcal{F} = \{\neg F \mid F \in \mathcal{F}\}$. También, para un conjunto finito de fórmulas $\mathcal{F} = \{F_1, ..., F_n\}$, definimos $\wedge\mathcal{F} = F_1 \wedge ... \wedge F_n$ y $\vee\mathcal{F} = F_1 \vee ... \vee F_n$. Si $\mathcal{F} = \emptyset$ entonces definimos $\wedge\mathcal{F} = \top$ y $\vee\mathcal{F} = \bot$, donde $\top$ es la abreviación de $\bot \leftarrow \bot$.

Una $\wedge \vee \neg f\acute{o}rmula$ es una fórmula construida sólo con los conectivos lógicos $\{\wedge, \vee, \neg\}$ usados de forma arbitraria debidamente anidados. Si A_1,A_2 y A_3 son conjuntos de átomos, no todos vacíos, entonces la fórmula

$\wedge (A_1 \cup \neg A_2 \cup \neg\neg A_3)$ es una *conjunción simple* mientras la fórmula

$\vee (A_1 \cup \neg A_2 \cup \neg\neg A_3)$ es una *disyunción simple*.

2.1 Programas lógicos

Una cláusula es una fórmula de la forma $H \leftarrow B$ donde H y B son fórmulas arbitrarias en principio, llamadas *cabeza* y *cuerpo* de la clausula respectivamente. Hay varios tipos de clausulas. Si $H = \bot$, la clausula es llamada una *restricción*, podemos omitir el símbolo $\bot$ y escribir dicha clausula como $\leftarrow B$. Análogamente, si $B = \top$ entonces la clausula es denominada un *hecho* y puede escribirse como H. Una clausula *aumentada* es una clausula donde H y B son alguna $\wedge \vee \neg f\acute{o}rmula$. Nótese que hay una clase amplia de fórmulas que incluyen dos tipos de negación. Nosotros por ahora sólo usamos un tipo de negación que corresponde a la negación por falla o por default, se usa *not* en lugar de nuestro símbolo $\neg$.

Una *clausula libre* es una clausula de la forma

$\vee (H_p \cup \neg H_n) \leftarrow \wedge (B_p \cup \neg B_n)$

donde H_p, H_n, B_p y B_n son conjuntos de átomos posiblemente vacíos. Si la clausula no contiene negación en la cabeza ($H_n = \emptyset$) es llamada *clausula general* y si, además, no es una *constraint o restricción* ($H_p \neq \emptyset$)es llamada *clausula disyuntiva*. Nótese que una clausula disyuntiva es un caso particular de una clausula general, una clausula general es una clausula libre y cada clausula libre es una clausula aumentada. Los siguientes son ejemplos de una clausula general, libre y aumentada, respectivamente:

$a \vee b \vee c \leftarrow d \wedge \neg e.$

$\neg a \vee b \vee \neg f \leftarrow a \wedge \neg h \wedge d.$

$((a \wedge \neg c) \vee d) \leftarrow ((\neg d \vee h) \wedge b)\,.$

Un programa lógico es entonces un conjunto finito de clausulas. Si todas las clausulas en un programa son de un cierto tipo, decimos que el programa es también de ese tipo. Por ejemplo, un conjunto de clausulas aumentadas es un *programa aumentado*, un conjunto de clusulas libres es un *programa*

libre y asi sucesivamente. Sea P un programa y M un conjunto de átomos tales que $M \subseteq \mathcal{L}_p$ entonces definimos $\widetilde{M}=\mathcal{L}_p \backslash M$.

2.2 Conjuntos respuesta

Ahora definimos los antecedentes básicos para conjuntos respuestas, mejor conocidos como answer sets o equivalentemente modelos estable. El presente material tiene algunas pequeñas variantes en relación a la negación clásica.

Decimos que una fórmula contiene el operador $\neg$ de negación como falla si contiene la subfórmula $\bot \leftarrow F$ y F no es $\bot$. Esta definición se extiende a programas de manera similar. Las fórmulas y programas que no contienen el operador de negación como falla son llamados básicos. Las fórmulas elementales son átomos asi como los conectivos $\bot$ y $\top$ [6].

Definición 1. [6] Un conjunto X de literales satisface una fórmula básica F, denotado por $X \vDash F$, recursivamente como sigue:
Para F elemental, $X \vDash F$ si $F \in X$ o $F = \top$.
$X \vDash F \wedge G$ si $X \vDash F$ y $X \vDash G$.
$X \vDash F \vee G$ si $X \vDash F$ o $X \vDash G$.

Definición 2. [6] Sea P un programa básico. Un conjunto de átomos X es cerrado bajo P si para cada clausula $H \leftarrow B \in P$, $X \vDash H$ cuando $X \vDash B$.

Definición 3. Sea X un conjunto de átomos y P un programa básico. X es llamado un conjunto answer para P si X es mínimo entre los conjuntos de átomos cerrados bajo P.

Definición 4. [6] La *reducción* de una fórmula aumentada o programa relativo a un conjunto de átomos X, es definido recursivamente como sigue:

Para F elemental, $F^X = F$.
$(F \wedge G)^X = F^X \wedge G^X$.
$(F \vee G)^X = F^X \vee G^X$.
$(\neg F)^X = \bot$ si $X \vDash F^X$ y $(\neg F)^X = \top$ en otro caso.
$(H \leftarrow B)^X = H^X \leftarrow B^X$.
$P^X = \left\{ (H \leftarrow B)^X \,|H \leftarrow B \in P \right\}$.

Definición 5 [6] (Conjunto Answer para un programa). Sea P un programa aumentado y X un conjunto de átomos. X es llamado un conjunto answer para P si es un conjunto answer para la reducción P^X.

Ejemplo 1. Considere el siguiente programa

$$P: \quad a \leftarrow \neg\neg a.$$
$$\neg b \leftarrow c \vee b.$$

Si elegimos $X = \{a\}$ entonces la reducción es

$$P^X: \quad a \leftarrow \top.$$
$$\top \leftarrow c \vee b.$$

Es fácil verificar que $\{a\}$ es cerrado bajo esta *reducción* y el conjunto vacío $\emptyset$ no lo es, este es el conjunto mínimo con esta propiedad. Se sigue que $\{a\}$ es un answer set de P. Sin embargo, nótese que el conjunto vacío $\emptyset$ es también un conjunto de P, esto produce una *reducción* diferente y cerrada.

2.3 Nociones básicas de lógicas intermedias

Introducimos algunas lógicas intermedias debido a que son de utilidad en el estudio de algunas propiedades y relaciones en ASP.

Lógicas Axiomáticas. Una lógica importante que ha sido de gran interés en los últimos años es la lógica intuicionista. Esta lógica está basada en el concepto de *prueba* antes que el concepto de *verdad* de la lógica clásica para explicar el significado y uso de las constantes lógicas o los cuantificadores.

La lógica intuicionista puede ser definida en términos de los siguientes esquemas de axioma:

1. $A \rightarrow (B \rightarrow A)$
2. $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
3. $A \wedge B \rightarrow A$
4. $A \wedge B \rightarrow B$
5. $A \rightarrow (B \rightarrow (A \wedge B))$
6. $A \rightarrow (A \vee B)$
7. $B \rightarrow (A \vee B)$
8. $(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow (A \vee B \rightarrow C))$

9. $(A \to B) \to ((A \to \neg B) \to \neg A)$
10. $\neg A \to (A \to B)$

Modus Ponens es la única regla de inferencia, es decir, si tenemos A y $A \to B$ entonces podemos deducir B. Nótese que si agregamos $(\neg A \to A) \to A$ a nuestros esquemas obtenemos los axiomas de la lógica clásica. Gödel resaltó que hay un número infinito de lógicas definidas entre la lógica intuicionista y la lógica clásica algunas veces llamadas lógicas intermedias, super-intuicionista o simplemente *si-lógica*, lo cual se puede ver de la siguiente manera:

$$I \subset ... \subset G_{i+1} \subset G_i \subset ... \subset G_3 \subset G_2 = C$$

Suponemos que la lógica intuicionista está incluida en las lógicas intermedias. Para una lógica axiomatica X, definida como anteriormente se indicó, decimos que una fórmula A es demostrable en X, denotado como $\vdash_X A$, si usa el conjunto definido de átomos y reglas de inferencia correspondientes tanto como es posible para obtener la fórmula A. Así mismo, si Γ es un conjunto de fórmulas entonces $\Gamma \vdash_X A$ tiene el significado usual.

Definiciones Generales. Hay conceptos que pueden definirse en alguna lógica sin depender su naturaleza. Decimos que una teoría T es *consistente* con respecto a la lógica X sii no hay una fórmula A tal que $T \vdash_X A$ y $T \vdash_X \neg A$. Decimos que T es una teoría *completa* sii para todo $A \in \mathcal{L}_T$ tenemos que $T \vdash_X A$ o $T \vdash_X \neg A$. Dos programas P_1 y P_2 son *equivalentes* bajo la lógica X, denotada como $P_1 \equiv_X P_2$, sii $P_1 \vdash_X A$ para cada $A \in P_2$ y $P_2 \vdash_X A$ para cada $A \in P_1$.

Para un conjunto de átomos M y un programa P escribiremos $P \vdash_X M$ para abreviar $P \vdash_X a$ para todo $a \in M$ y $P \nvdash_X M$ para indicar el hecho de que P es *consistente*, *completo* y $P \vdash_X M$. Si uno de los símbolos $\vdash_X$ o $\nvdash_X$carece de el subíndice X, suponemos que nos referimos a la lógica intuicionista I.

Tambien usaremos los resultados básicos ya conocidos siguientes:

Lemma 1. [11] *Sea T una teoría, es decir, un conjunto de fórmulas y sean F y G un par de fórmulas equivalentes bajo alguna $si-lógica$ X.*

Alguna teoría obtenida de T por el remplazo de algunas ocurrencias de F por G es equivalente a T bajo X.

Lemma 2. [8] *Sean T_1,T_2 dos teorías y A una fórmula tal que $\mathcal{L}_{T_1 \cup \{A\}} \cap \mathcal{L}_{T_2} = \emptyset$, T_2 es un conjunto de literales negativas y $T_1 \cup T_2 \vdash_I A$. Entonces $T_1 \vdash_I A$.*

Prueba:

Supongamos que $T_1 \cup T_2 \vdash_I A$, donde T_1, T_2 y A satisfacen la hipótesis dada en el lema. Sea θ la sustitución que reemplaza cada ocurrencia de x en $\mathcal{L}_{T_2}$ por $\bot$. Entonces una inducción directa muestra que $T_1\theta \cup T_2\theta \vdash_I A\theta$. Pero $A = A\theta$, $T_1 = T_1\theta$, y $T_2\theta \equiv_I \{\bot \leftarrow \bot\}$. Así, $T_1 \cup \{\bot \leftarrow \bot\} \vdash_I A$ y $T_1 \vdash_I A$ como se quería.■

Definición 6. [9] *El conjunto $\mathcal{P}$ de fórmulas positivas es el conjunto más pequeño que contiene a todas las fórmulas sin el conectivo de negación $\neg$. El conjunto $\mathcal{N}$ de fórmulas doblemente negadas es el conjunto X más pequeño con las siguientes propiedades:*

1. *Si a es un átomo entonces $(\neg\neg a) \in X$.*
2. *Si $A \in X$ entonces $(\neg\neg A) \in X$.*
3. *Si $A, B \in X$ entonces $(A \wedge B) \in X$.*
4. *Si $A \in X$ y B es alguna fórmula entonces $(A \vee B)$,$(B \vee A)$,$(A \leftarrow B) \in X$.*

Para algún conjunto de fórmulas Γ, el *subconjunto positivo* de Γ, denotado como $Pos\,(\Gamma)$, es el conjunto $\Gamma \cap \mathcal{P}$.

Proposición 1. [9] *Sea Γ un subconjunto de $\mathcal{P} \cup \mathcal{N}$ y sea $A \in \mathcal{P}$ una fórmula positiva. Si $\Gamma \vdash_I A$ entonces $Pos\,(\Gamma) \vdash_I A$.*

Prueba:

La prueba es por inducción sobre la longitud de la deducción para $\Gamma \vdash_I A$ usando el cálculo secuente.

Si $\Gamma \vdash_I A$ es un axioma, la base de la inducción, entonces $\Gamma = \{A\}$ y la proposición es trivial (son triviales).

Si la última regla en la deducción de $\Gamma \vdash_I A$ es una regla estructural izquierda $\mathcal{L}$X, $\mathcal{L}$W, $\mathcal{L}$C o una de las reglas lógicas $\mathcal{L}_i\wedge$, $\mathcal{R}\wedge$, $\mathcal{L}\vee$, $\mathcal{R}_i\vee$, $\mathcal{L}\rightarrow$,

$\mathcal{R}\rightarrow$, la hipótesis inductiva puede ser aplicada a los secuentes previos en la deducción. Aunque tomemos subconjuntos positivos en las premisas de estas reglas se preserva "demostrabilidad" y es posible probar $Pos(\Gamma) \vdash_I A$.

Por ejemplo, si la última regla es

$$\frac{\Gamma_1, B \vdash_I A \qquad \Gamma_2, C \vdash_I A}{\Gamma_1, \Gamma_2, B \vee C \vdash_I A} \quad \mathcal{L}\vee$$

donde $\Gamma = \Gamma_1,\ \Gamma_2,\ B \vee C$.

Si $B \vee C \in P$ entonces también B y A están en P, asi, por la hipótesis inductiva tenemos pruebas para $Pos\,(\Gamma_1, B) \vdash_I A$ y $Pos\,(\Gamma_2, C) \vdash_I A$. Usando nuevamente la regla $\mathcal{L}\vee$ ahora probamos

$$\frac{Pos(\Gamma_1), B \vdash_I A \qquad Pos(\Gamma_2), C \vdash_I A}{Pos(\Gamma_1, \Gamma_2), B \vee C \vdash_I A}$$

el cual finalmente nos lleva a que $Pos\,(\Gamma) \vdash_I A$.

Si $B \vee C \in N$ entonces B o C están en N. Supóngase que $B \in N$, el otro caso es análogo. Aplicando la hipótesis inductiva sobre $\Gamma_1, B \vdash_I A$, tenemos una prueba para $Pos\,(\Gamma_1) \vdash_I A$. Ahora, usando la regla debilitada $\mathcal{L}W$, es posible probar $Pos\,(\Gamma_1)\,, Pos\,(\Gamma_2) \vdash_I A$ o, equivalentemente, $Pos\,(\Gamma) \vdash_I A$.

La última regla no puede ser una regla de negación aunque $\mathcal{L}\neg$ necesitará A para ser vacía, y $\mathcal{R}\neg$ implica $A \notin P$ contradiciendo la hipótesis de la proposición. La última regla tampoco es $\mathcal{R}W$ ya que tendríamos $\Gamma \vdash_I$ o, equivalentemente, $\vdash_I \neg\,(\wedge\Gamma)$. Pero $\vdash_C \neg\,(\wedge\Gamma)$ no es demostrable aunque hay una interpretación, la cual asigna *verdad* a cada átomo, que evalua la fórmula como $falsa$.■

Otra lógica axiomática importante es la lógica de Jankov (Jn), la cual es obtenida al agregar al conjunto de axiomas intuicionistas el nuevo esquema de axioma $\neg A \vee \neg\neg A$. Este axioma que caracteriza la lógica de Jankov es también conocido como la ley del tercero excluido débil.

Proposición 2. [9] *Sean* a_1,..., a_n *todos los átomos que ocurren en una fórmula* A. *Entonces* $\vdash_{Jn} A$ *sii* $(\neg a_1 \vee \neg\neg a_1)$,...,$(\neg a_n \vee \neg\neg a_n) \vdash_I A$.

Prueba:

Para esta prueba es suficiente con mostrar que alguna instancia del esquema de axioma $\neg A \vee \neg\neg A$ puede ser probado del conjunto de hipótesis $\{\neg a_1 \vee \neg\neg a_1 ... \neg a_m \vee \neg\neg a_m\}$, donde $a_1, ..., a_m$ son todos los átomos que ocurren en A.

La prueba es por inducción sobre el tamaño de A. Para el caso base suponemos que $A \in \{a_1, ..., a_m, \bot\}$. Si $A \in \{a_1, ..., a_m, \bot\}$ el enunciado es verdad por hipótesis. Si $A = \bot$ entonces el resultado es verdad porque $\neg \bot \vee \neg\neg \bot$ es un teorema. Por inducción suponemos que:

$(\neg a_1 \vee \neg\neg a_1) ... (\neg a_m \vee \neg\neg a_m) \vdash_I \neg F_1 \vee \neg\neg F_1$ y

$(\neg a_1 \vee \neg\neg a_1) ... (\neg a_m \vee \neg\neg a_m) \vdash_I \neg F_2 \vee \neg\neg F_2$

Ahora, no es dificil de probar que:

$(\neg F_1 \vee \neg\neg F_1) ... (\neg F_2 \vee \neg\neg F_2) \vdash_I \neg (F_1 \odot F_2) \vee \neg\neg (F_1 \odot F_2)$

de donde:

$(\neg a_1 \vee \neg\neg a_1) ... (\neg a_m \vee \neg\neg a_m) \vdash_I \neg (F_1 \odot F_2) \vee \neg\neg (F_1 \odot F_2)$,

donde $\odot$ representa algún conectivo binario, recordemos que $\neg A$ abrevia la fórmula $\bot \leftarrow A$ y no tenemos considerado este caso en la prueba.■

Lemma 3. [9] *Sea A una fórmula positiva y Γ un subconjunto de $\mathcal{P} \cup \mathcal{N}$. Entonces $\Gamma \vdash_{Jn} A$ sii $Pos(\Gamma) \vdash_I A$.*

Prueba:

Observe que alguna instancia del esquema de los axiomas de Jankov's $\neg A \vee \neg\neg A$ está contenida en el conjunto N. La *proposición 1* permite borrar algunas instancias de este axioma usado en la prueba $\Gamma \vdash_{Jn} F$ lo cual resulta como una prueba para $Pos(\Gamma) \vdash_I F$.■

Lógicas Multivaluadas. Las lógicas también pueden estar definidas en términos de sus valores de verdad y funciones de evaluación. Gödel definió las lógicas multivaluadas G_i con valores en $\{0, 1, ..., i-1\}$ donde $\tau = i - 1$ es el valor designado con la siguiente función de evaluación:

$f(B \leftarrow A) = \tau$ si $f(A) \leq f(B)$ y $f(B \leftarrow A) = f(B)$ en otro caso.
$f(A \vee B) = max(f(A), f(B))$.
$f(A \wedge B) = min(f(A), f(B))$.
$f(\neg A) = 0$ si $f(A) > 0$ y $f(\neg A) = \tau$ si $f(A) = 0$.
$f(\top) = \tau$ y $f(\bot) = 0$.

Una interpretación en tales lógicas multivaluadas es una función que asigna a cada átomo en $\mathcal{L}$ un valor de $\{0, 1, ..., \tau\}$. La interpretación de una fórmula arbitraria se obtiene propagando la evaluación de cada conectivo como fue definido anteriormente. Una interpretación se dice *definida*

si asigna sólo valores 0 o τ a un átomo e *indefinida* si le asigna algún valor intermedio.

Para una interpretación I y una fórmula F decimos que I es un *modelo* de F, o que I *modela* F, si $I(F) = \tau$. Extendemos esta definición hacia una teoría o conjunto de fórmulas. Una tautología es una fórmula que evalua a τ para cada posible interpretación.

De las lógicas mencionadas utilizaremos la lógica G_3. Nótese que G_2 es la lógica proposicional clásica.

Chapter 3

Nociones de equivalencia

Dados dos programas definiremos varios tipos de relaciones de equivalencia básica. Primero recordemos que dos programas P_1 y P_2 son equivalentes bajo una lógica X cuando existe una prueba para $\vdash_X \wedge P_1 \equiv \wedge P_2$, lo cual se puede escribir como $P_{1 \equiv_X} P_2$.

Otra forma de equivalencia básica puede ser definida en términos de semánticas stable. Dados dos programas, decimos que son equivalentes si comparten los mismos modelos stable, lo cual podemos denotar como $P_1 \equiv_{stable} P_2$.

3.1 Equivalencia fuerte

Para cada tipo de relaciones de equivalencia básica descrita anteriormente, podemos definir algunas otras nociones de equivalencia. Una de tales nociones es la equivalencia fuerte.

Definición 7. [7] *Dos programas P_1 y P_2 son $fuertemente\ equivalentes$ si $P_1 \cup P$ es equivalente a $P_2 \cup P$ para cada programa P.*

Esta definición general puede ser usada por alguna relación de equivalencia básica pero es particularmente útil en el contexto de las semánticas del modelo stable. Por ejemplo, si dos programas son fuertemente equivalentes bajo stable, sabemos que uno de ellos puede ser reemplazado por el otro en un programa más grande sin cambiar la semántica declarativa.

En general, es claro que la equivalencia fuerte implica equivalencia, pero lo contrario no siempre es verdad.

Ejemplo 2. Consideremos los programas $P_1 = \{a \leftarrow \neg b\}$ y $P_2 = \{a\}$ equivalentes en stable porque $\{a\}$ es el único modelo stable para ambos programas. Sin embargo, $P_1 \cup \{b \leftarrow a\}$ no tiene modelos stables, mientras $P_2 \cup \{b \leftarrow a\}$ tiene a $\{a, b\}$ como modelo stable.

Como un resultado del estudio de la equivalencia entre programas, en los siguientes teoremas se presenta una relación importante entre semántica stable y lógicas intermedias.

Teorema 1. [7] *Sean P_1 y P_2 dos programas aumentados, entonces P_1 y P_2 son fuertemente equivalentes bajo stable sii P_1 y P_2 son equivalentes en lógica G_3.*

Teorema 2. [8] *Sean P_1 y P_2 dos programas aumentados, entonces P_1 y P_2 son fuertemente equivalentes bajo stable sii P_1 y P_2 son equivalentes en lógica de Jankov.*

La intención de tales definiciones de equivalencia es la simplificación de programas. Si podemos mostrar que un programa en algún sentido es equivalente a uno más simple entonces utilizamos la versión más simple.

Lemma 4. *Sean P_1 y P_2 dos programas aumentados y sean $\{a_1, ..., a_n\}$ los átomos en $\mathcal{L}_{P_1 \cup P_2}$, entonces los programas P_1 y P_2 son fuertemente equivalentes sii $(\neg a_1 \vee \neg\neg a_1)$,...,$(\neg a_n \vee \neg\neg a_n) \vdash_I \wedge P_{1\equiv} \wedge P_2$.*

Prueba:

Directa por la aplicación del *teorema 2* y *proposición 2.*■

Lemma 5. *Sea P un programa disyuntivo y a un átomo. P es fuertemente equivalente a $P \cup \{a\}$ sii $Pos(P) \vdash_I a$.*

Prueba:

Para una clausula disyuntiva C de la forma $\vee H_p \leftarrow \wedge (B_p \cup \neg B_n)$, escribiremos $tr(C)$ para denotar la clase aumentada $\vee (H_p \cup \neg\neg B_n) \leftarrow \wedge B_p$. Para un programa disyuntivo P, el programa $tr(P)$ es obtenido sólo por la aplicación de la transformación en cada clausula. Algunas propiedades simples y fáciles para verificar esta trasformación son: $P \equiv_{J_n} tr(P), tr(P) \subset P \cup N$ y $Pos(tr(P)) = Pos(P)$.

Ahora P es fuertemente equivalente a $P \cup \{a\}$ sii por el teorema 2, $P \equiv_{J_n} P \cup \{a\}$ sii $P \vdash_{J_n} a$ sii, puestp que $P \equiv_{J_n} tr(P), tr(P) \vdash_{J_n} a$ sii, por el *lema 3*, $Pos(tr(P)) \vdash_I a$ sii $Pos(P) \vdash_I a$.■

Lemma 6. [9] *Dados un programa aumentado P y un átomo negado $\neg a$, entonces P es fuertemente equivalente a $P \cup \{\neg a\}$ sii $P \vdash_C \neg a$.*

Prueba:

Es conocido que $P \vdash_C \neg a$ si y solamente si $P \vdash_{Jn} \neg a$, y a su vez $P \vdash_{Jn} \neg a$ sii $P \cup \{\neg a\} \equiv_{J_n} P$ y por el *teorema 2*, *sii* esos dos programas son fuertemente equivalentes.■.

3.2 Extensiones conservativas o fieles

Si queremos usar un programa P_2 en lugar de otro programa P_1 será adecuado si ambos programas tienen los mismos modelos estables. Sin embargo, no siempre necesitaremos ésta condición para hacerlo, será suficiente con la construcción de una simple función para mapear modelos stable entre los dos programas.

Una *extensión conservativa o fiel* es una forma de equivalencia débil.

Dados dos programas P_1 y P_2 diremos que P_2 es una extensión conservativa de P_1 bajo stable si $P_1 \subset P_2$ y M_1 es un modelo stable de P_1 sii M_2 es un modelo stable de P_2 tal que $M_1 = M_2 \cap \mathcal{L}_{P_1}$.

También decimos que P_2 es una transformación conservativa de un programa P_1, si P_2 puede ser obtenida de P_1 por una secuencia finita de equivalencias o extensiones conservativas. Nótese que para este caso si ya hemos determinado modelos stable de P_2, es posible obtener varios modelos facilmente a partir de P_1 seleccionando el conjunto intersección de cada modelo con $\mathcal{L}_{P_1}$.

Inoue y Sakama han mostrado que cada programa libre puede ser transformado a través de una transformación conservativa en un programa general. Sin embargo, esta transformación es demasiado extensa e incrementa cúbicamente el tamaño del programa. Más adelante, se muestra que esta misma meta se puede alcanzar con una transformación computable en tiempo

lineal lo cual representa un incremento en el tamaño del programa también lineal.

Lema 7.*Sea P un programa libre. Para un conjunto dado $S \subseteq \mathcal{L}_P$ y $\varphi : S \to \Sigma$ una función biyectiva, donde Σ es un conjunto de átomos tal que $\Sigma \cap \mathcal{L}_P = \emptyset$. Sea $\triangle_S = \cup_{a \in S} \{\varphi(a) \leftarrow \neg a, \bot \leftarrow a \wedge \varphi(a)\}$. Entonces $P \cup \triangle_S$ es una extensión conservativa de P.*

Prueba:

Primero probaremos que si M es un conjunto respuesta de P entonces $M^* = M \cup \varphi(S \backslash M)$ es un conjunto respuesta de $P \cup \triangle_S$. De acuerdo a la definición de conjunto respuesta dado en la sección 2 debemos mostrar que M^* es cerrada bajo la *reducción* $(P \cup \triangle_S)^{M^*}$ y estar entre los mínimos de los conjuntos con esta propiedad.

Por construcción M^*, es cerrada bajo $(\triangle_S)^{M^+}$ y también M es cerrado bajo P^M porque es un conjunto respuesta de P, también tenemos que M^* es cerrado bajo $(P \cup \triangle_S)^{M^*}$. Nótese que solamente la condición de cerradura y el operador *reducción* pueden ser distribuidas entre las clausulas . También los átomos extra en M^* no en $\mathcal{L}_P$, es decir no en M, son insignificantes durante el cálculo de la *reducción* del programa P.

Ahora checaremos que M^* es minimal. Supóngase que hay otro conjunto de átomos N^* cerrado bajo $(P \cup \triangle_S)^{M^*}$ y $N^* \subset M^*$. Escribimos N^* como la unión disjunta $N^* = N \cup N'$ donde $N = N^* \cap \mathcal{L}_P$ y $N' = N^* \backslash L_P$. Note que $M^* = M \cup \varphi(S \backslash M)$ se escribe también en la misma forma. Observe que N^* es cerrado bajo P^{M^*} y así N es cerrado bajo P^M. Como $N^* \subset M^*$ tenemos que $N \subseteq M$ y , en el otro caso, como M está entre los conjuntos mínimos de átomos cerrados bajo P^M tenemos $M \subseteq N$. Así $N = M$.

Entonces si $N^* \neq M^*$ debe haber un átomo $x \in \varphi(S \backslash M)$ tal que x no está en N',ni en N^*. Sea $a \in S$ es un átomo tal que $\varphi(a) = x$. También sabemos de $x \in \varphi(S \backslash M)$ que a no está en M y no está en N^* ni en M^*.

Ahora, ni a ni x están en N^* así que este conjunto no satisface la clausula $(x \leftarrow \neg a)^{M^*} = x \leftarrow \top$ contenida en $(\Delta_S)^{M^*}$. Así, N^* no es cerrado bajo $(P \cup \Delta_S)^{M^*}$, generando contradicción.

Para el inverso tenemos que probar que si M^* es un conjunto respuesta de $P \cup \triangle_S$ entonces $M = M^* \cap \mathcal{L}_P$ es un conjunto respuesta de P. Descomponiendo como antes , $M^* = M \cup M'$. Es inmediato que M^*es cerrado bajo P^{M^*} y así M es cerrado bajo P^M.

Antes de finalizar observemos que dado un átomo $a \in S$ y $x = \varphi(a)$, tenemos que $a \in M$ si y sólo si $x \notin M'$. Es fácil de verificar esto teniendo ambos átomos (o no) en M^* haciendo imposible para M^* que sea cerrado bajo $(\Delta_S)^{M^*}$.

Finalmente, podemos probar que M es mínimo. Supóngase que N es cerrado bajo P^M y $N \subset M$. Construimos $N^* = N \cup M'$, así $N^* \subset M^*$. Por lo dicho anteriormente es fácil verificar que N^* es cerrado bajo $(\Delta_S)^{M^*}$ contradiciendo el hecho de que M^*es el conjunto mínimo con tal propiedad.■

Ejemplo 3. Considere $S = \{a\}, \Sigma = \{x\}$ y el programa $P : a \vee \neg a$.

Así, $P \cup \triangle$ es el programa:

$a \vee \neg a.$
$x \leftarrow \neg a.$
$\leftarrow x \wedge a.$

Observe que P tiene dos modelos stables, $M_1 = \emptyset$ y $M_2 = \{a\}$. Note que $\{x\}$ y $\{a\}$ son sólo los dos modelos stable de $P \cup \triangle$.

Proposición 3. [9] *Sea P un programa libre y sea S el conjunto que contiene todos los átomos a tales que $\neg a$ aparece en la cabeza de alguna clausula en P. Sean φ y $\triangle_S$ definidos como en el lema 7. Sea P' el programa general obtenido de P por el reemplazo de cada ocurrencia de $\neg a$ con $\varphi(a)$ para todo $a \in S$. Entonces $P' \cup \triangle_S$ es un programa general que es una transformación conservativa de P. En particular M es un modelo stable de P sii M_S es un modelo stable de $P' \cup \triangle_S$, donde $M_S = M \cup \varphi(S \backslash M)$.*

Prueba:

Primero, por el *lema 7*, sabemos que $P \cup \Delta_S$ es una extensión conservativa de P. Note que $\Delta_S \vdash_{G_3} \varphi(a) \equiv \neg a$ para cada $a \in S$, entonces por el lema 1, $P \cup \Delta_S \equiv_{G_3} P' \cup \Delta_S$ donde P' es obtenido de P por el reemplazo de cada ocurrencia de $\neg a$ con $\varphi(a)$ para todo $a \in S$. Esta equivalencia en lógica G_3 es, por el teorema 1, la misma equivalencia fuerte con respecto a

las semánticas estables. Así, $P' \cup \Delta_S$ es un programa general que es una transformación conservativa de P.■

Ejemplo 4. Considerando el último programa nuevamente, tenemos:

$$P : a \vee \neg a.$$

Así, $P' \cup \triangle_S$ es el programa:

$$a \vee x.$$
$$x \leftarrow \neg a.$$
$$\leftarrow x \wedge a.$$

Recordemos que $\mathrm{M}_1 = \emptyset$ y $\mathrm{M}_2 = \{a\}$ son todos los modelos stable de P. Así mismo, $\{x\}$ y $\{a\}$ son los dos únicos modelos stable de $P' \cup \triangle_S$.

3.3 Reglas de transformación para programas libres

En esta sección se definirán algunas reducciones que pueden ser aplicadas en cierto orden para simplificar programas libres. También serán utilizadas como una herramienta teórica en la prueba del teorema 5 y serán estudiadas algunas propiedades de esas mismas reducciones.

Definición 8 (Primera Reducción). *Sea P un programa general. Supongamos que $P{=}P' \cup U_{i=1}^{n} \{\bot \leftarrow a_i\}$ tal que P' carece de clausulas de la forma $\bot \leftarrow b$. Denotamos y definimos la primera reducción de P como $redu1\,(P)$ obtenido de P por la aplicación ordenada de los siguientes pasos para cada a_i:*

(a) *Borrar todas las clausulas $(H \leftarrow B) \in P'$ donde $a_i \in Lit\,(B)$.*

(b) *Borrar $\neg a_i$ de B en todas las clausulas $(H \leftarrow B) \in P'$ tal que $\neg a_i \in Lit\,(B)$.*

(c) *Borrar a_i de H en todas las clusulas $(H \leftarrow B) \in P'$ tal que $a_i \in Lit\,(H)$.*

Ejemplo 5. Este ejemplo ilustra como se obtiene la primera reducción de un programa:

$$\begin{array}{lcl} b \leftarrow \neg c \wedge a. & & \\ \bot \leftarrow \neg b \wedge c. & & \bot \leftarrow \neg c \wedge a. \\ \bot \leftarrow b \wedge a \wedge d. & \Longrightarrow & \bot \leftarrow c. \end{array}$$

$f \leftarrow \neg e \wedge \neg b.$ $\quad\quad$ $f \leftarrow \neg e.$
$d \leftarrow b \wedge e \wedge f.$ $\quad\quad$ $\bot \leftarrow b.$
$\bot \leftarrow b.$

Nótese que cuando borramos b de la cabeza de la clausula $b \leftarrow \neg c \wedge a$ dicha clausula queda como $\bot \leftarrow \neg c \wedge a$.

Definición 9 (Segunda Reducción). *Sea P un programa general. Supóngase que $P = P' \cup U_{i=1}^{n} \{\bot \leftarrow \neg a_i\}$ tal que P' carece de clausulas de la forma $\bot \leftarrow \neg b$. Denotamos y definimos la segunda reducción de P como $redu2\,(P)$ obtenido de P por la aplicación ordenada de los siguientes pasos para cada restricción $\bot \leftarrow \neg a_i \in P$.*

(a) *Borrar todas las clausulas $(H \leftarrow B) \in P'$ donde $\neg a_i \in Lit\,(B)$.*

(b) *Borrar a_i de B en todas las clausulas $(\bot \leftarrow B) \in P'$ tal que $a_i \in Lit\,(B)$.*

La idea de estas reducciones fue motivada por [3].

Algunas propiedades de las formas reducidas son que ambas trasformaciones preservan equivalencia con respecto a la lógica intuicionista; en particular serán equivalentes en lógica de Jankov y por el teorema 1, fuertemente equivalentes bajo stable. Otra propiedad de la primera reducción consiste en que mediante una aplicación conveniente permitirá simplificar la estructura de programas.

Lema 8. *Sea P un programa general consistente que es la unión disjunta de P_1 y N_1 donde P_1 es un programa general y N_1 es un conjunto de restricciones de la forma $\bot \leftarrow a_i$. Entonces se cumplen las siguientes propiedades:*

1. *$P \equiv_I redu1\,(P)$.*
2. *Existe P_2 tal que $redu1(P)$ es la unión disjunta de P_2 y N_1 donde P_2 es un programa general y $\mathcal{L}_{N_1} \cap \mathcal{L}_{P_2} = \emptyset$.*

Prueba:

Por la *propiedad 1*, puede verificarse fácilmente que cada paso de la transformación reemplaza fragmentos del programa los cuales son equivalentes en lógica intuicionista, asi que dicha transformación será también equivalente al programa original. La prueba de la *propiedad 2* se sigue inmediatamente por construcción. Si $\vdash \leftarrow a \in N_1$ entonces en cada clausula en P_1 donde a ocurre entonces o borramos la clausula o borramos la ocurrencia dada de a, esto construye P_2.■

Lema 9. *Sea P un programa libre, entonces $P \equiv_I redu2(P)$.*

Prueba:

Observe que cada paso de la transformación es borrada solamente de la hipótesis redundante del programa. Las clausulas simplificadas con sus correspondientes hechos $\bot \leftarrow \neg a_i$, son intuicionisticamente equivalentes a las clausulas originales.■

Chapter 4

Lógicas intermedias

Como hemos visto en las secciones previas existen algunas relaciones entre semántica stable y lógicas intermedias. En la presente sección se prueban algunas proposiciones en lógicas de *Jankov* y G_3. Estos resultados serán de utilidad posteriormente para probar otras propiedades de semántica stable y para construir propuestas de nuevas semánticas .

Proposición 4. *Un programa aumentado es equivalente bajo la lógica de Jn a un programa libre.*

Prueba:

Sea P un programa aumentado. Usando las propiedades distributivas de la conjunción, disyunción y negación, todas verdad en lógica de *Jankov*, es posible remplazar cada clausula en P con una clausula de la forma $H \leftarrow B$, donde H es una conjunción de disyunciones simples y B es una disyunción de conjunciones simples.

Las siguientes reglas permiten eliminar conjunciones en al cabeza de clausulas, disyunciones en cuerpos y literales con dos (o más) negaciones.

$$\frac{A \leftarrow C}{A \wedge B \leftarrow C \equiv_{Jn}} \qquad \frac{A \leftarrow B}{A \leftarrow B \vee C \equiv_{Jn}}$$

$$\frac{B \leftarrow C}{A \vee \neg\neg B \leftarrow C \equiv_{Jn} A \leftarrow \neg B \wedge C; A \leftarrow \neg\neg B \wedge C \equiv_{Jn} A \vee \neg B \leftarrow C} \qquad \frac{A \leftarrow C}{}$$

Finalmente, obtenemos un programa con todas las clusulas en formato libre.■

Se probarán algunos resultados para lógica G_3. En G_3 una interpretación es una función I la cual asigna a cada átomo un valor de verdad del conjunto $\{0, 1, 2\}$ y se evaluan los valores de verdad de cada fórmula proposicional como se detalló en la subsección 4.2.

Definición 10. *Para una interpretación I dada en la lógica G_3. I' es definida como $I'(a) = I(\neg\neg a)$ para cada átomo a en el contexto del lenguaje.*

Observe que I' es equivalente al reemplazar en I todos los valores intermedios , es decir excepto 0 y τ por el valor designado τ.

Proposición 5. *Sea A una fórmula arbitraria e I una interpretación para ella en G_3. Entonces $I'(A) = I(\neg\neg A)$.*

Prueba:

Se prueba por inducción sobre n, el número de conectivos en A. Si $n = 0$ entonces A es un átomo y por definición $I'(A) = I(\neg\neg A)$, o si $A = \perp$ entonces $I'(\perp) = I(\neg\neg \perp) = 0$. Para $n \geq 1$, si $A = B \odot C$, donde $\odot$ es un conectivo de $\{\vee, \wedge, \leftarrow\}$, por inducción, tenemos $I'(B \odot C) = I(\neg\neg(B \odot C)) = I(\neg\neg A)$.■

Corolario 1. *Sea I una interpretación. Si I modela A entonces I' también modela A.*

Prueba:

Tenemos $I(A) = I(\neg\neg A) = \intercal$ y, por la *proposición 5*, $I(\neg\neg A) = I'(A)$.■

Proposición 6. *Sean A y B dos fórmulas. Si $A \not\cong_{G_3} B$ entonces hay una interpretación I la cual modela A y no modela a B, o modela B y no modela A.*

Prueba:

Como $A \neq_{G_3} B$ entonces existe una interpretación I tal que $I(A) \neq I(B)$. Supongamos, sin pérdida de generalidad, que $I(A) > I(B)$.

Si I modela A entonces el problema esta resuelto. En otro caso la única posibilidad es que $I(A) = 1$ y $I(B) = 0$. Pero I' como fue definida anteriormente será $I'(A) = 2$ y $I'(B) = 0$.■

Definición 11. *Dada una interpretación I en G_3 el programa $T(I)$ es definido como el conjunto mínimo X el cual satisface:*

1. *Si* $I(a) = I(b) = 1$ *y* $a \neq b$ *entonces* $(b \leftarrow a) \in X$.
2. *Si* $I(a) = 1$ *entonces* $(a \leftarrow \neg a) \in X$.
3. *Si* $I(a) = 2$ *entonces* $(a) \in X$.
4. *Si* $I(a) = 0$ *entonces* $(\bot \leftarrow a) \in X$.

Proposición 7. *Sean* P_1 *y* P_2 *dos programas arbitrarios. Si hay una interpretación* I *tal que modela* P_1 *y no modela* P_2 *entonces existe un programa* P *tal que:*

(i). $P_1 \cup P$ *es una extensión consistente y completa de* P_1 *mientras* $P_2 \cup P$ *es inconsistente.*

(ii) $P_1 \cup P$ *es incompleta y no puede ser completada, preservando consistencia, agregando solamente átomos negados mientras* $P_2 \cup P$ *es consistente y completa.*

Prueba :

Tenemos dos casos principales:

1. Hay una interpretación definite I que modela P_1 y no modela P_2. Sea $P = T(I)$ como en la *definición 11.* Entonces, es inmediato por construcción que $P_1 \cup P$ es una extensión completa y consistente de P_1, mientras $P_2 \cup P$ es inconsistente.

2. Si cada interpretación que modela P_1 y no modela P_2 es indefinida, entonces I es una de tales interpretaciones y $P = T(I)$. Como I modela P_1 tenemos, por el corolario 1, que I' modela P_1.Nótese que $I \neq I'$ debido a que I contiene algunos asignamientos unarios e I' no.

Como I modela P, tenemos que I modela $P_1 \cup P$. Entonces por el *corolario 1*, I' también modela $P_1 \cup P$. Pero $I \neq I'$ asi que hay dos interpretaciones diferentes que modelan $P_1 \cup P$ y por lo cual no es completa, además, no puede ser completado agregando átomos negados $\neg a$ porque el programa sería incosistente (si $I(a) = 1$ o $I(a) = 2$) o será incompleto (si $I(a) = 0$).

Ahora I' modela P_2 también, si I' no modelara P_1 y tampoco P_2, I' se define contradiciendo la hipótesis de este caso. Otra vez el corolario 1 proporciona I' que modela P y además, I' modela $P_2 \cup P$.

Mostraremos que sólo I' es la interpretación que modela $P_2 \cup P$. Supongamos que K es otra ineterpretación que modela $P_2 \cup P$, y que además K modela P_2.

Caso 1. Si $I(a) = 2$, entonces $(a) \in P \subset P_2 \cup P$. Pero $K(a) \neq 2$ implicará $K(P_2 \cup P) \neq 2$ contradiciendo el hecho de que K modela $P_2 \cup P$. Así, si $I(a) = 2$ entonces $K(a) = 2$.

Caso 2. Si $I(a) = 0$, entonces $(\neg a) \in P \subset P_2 \cup P$. Pero $K(a) \neq 0$ implicará $K(\neg a) \neq 2$ y $K(P_2 \cup P) \neq 2$ contradiciendo el hecho de que K modela $P_2 \cup P$. Asi, si $I(a) = 0$ entonces $K(a) = 0$.

Caso 3a. Si $I(a) = 1$ y $K(a) = 0$, entonces $(\neg a \rightarrow a) \in P \subset P_2 \cup P$. Pero K evaluará $K(\neg a \rightarrow a) = 0$ y $K(P_2 \cup P) = 0$ generando contradicción otra vez.

Caso 3b. $I(a) = 1$ y $K(a) = 1$, entonces, si existe, tomar otro átomo b tal que $I(b) = 1$. Ahora $\{a \rightarrow b, b \rightarrow a\} \subset P \subset P_2 \cup P$ y como K modela $P_2 \cup P$, $K(a \equiv b) = 2$, por tanto $K(a) = K(b) = 1$. Así, en este caso $I(a) = 1$ implica $K(b) = 1$ para todos los átomos b, por principio $I = K$. Pero de la hipótesis, I no modela P_2 y ni a K. Contradicción.

Caso 3c. De los dos casos previos $I(a) = 1$ implica $K(a) = 2$ y a través de los casos 1 y 2 se deriva a $K = I'$. Así como fue afirmado, solo I' es el modelo para $P_2 \cup P$.■

Ejemplo 6. Consideremos el programa $P_1 = \{a \leftarrow a\}, P_2 = \{a \vee \neg a\}$. Sea I una interpretación tal que $I(a) = 1$. Observe que I modela P_1 y no modela P_2. Si tomamos $P = \{a \leftarrow \neg a\}$ el programa $P_1 \cup P$ será incompleto y no podrá ser completado agregando átomos negados, mientras que $P_2 \cup P$ es consistente, completo y prueba a.

Chapter 5

Aplicaciones de ASP

En esta sección se da uno de los principales resultados que es es el *teorema 5*, el cual da una caracterización de modelos stable de programas aumentados en términos de lógica intuicionista. Basándose en este resultado, se propone una definición de modelo stable para la teoría general proposicional .

5.1 Simplificación de un programa

Lifschitz, Tang y Turner dieron una generalización de semántica stable para programas aumentados. También mostraron que es posible transformar un programa aumentado en uno libre sin alterar los modelos stable resultantes.

Teorema 3. *Un programa aumentado es fuertemente equivalente bajo stable a un programa libre.*

Los autores no usan el término "fuertemente equivalente", sino "equivalente". Sin embargo su noción de equivalencia como fue definida en la sección anterior corresponde realmente a la equivalencia fuerte. Usando la maquinaria de la lógica, este teorema es una simple consecuencia de la proposición 4 y del teorema 2 de este reporte.

Ejemplo 7. Considere el siguiente programa aumentado:

$$P: \quad \begin{array}{l} a \leftarrow \neg\neg a. \\ \neg b \leftarrow c \vee b. \end{array}$$

Es posible construir, aplicando las reglas descritas en la prueba de la proposicion 4, un programa libre el cual, por el teorema previo, es fuertemente equivalente a P. Tal programa sería:

$a \vee \neg a.$
$\neg b \leftarrow c.$
$\neg b \leftarrow b.$

Usando los resultados de las secciones previas podemos completar una cadena de equivalencias en cierto orden para mostrar que los programas aumentados no son más expresivos que los disyuntivos. Asi, estamos listos para computar modelos stables de simples programas disyuntivos, facilitaremos el cálculo de los modelos stables a partir de programas más complicados incluyendo los del tipo aumentados.

Teorema 4. *Para cada programa aumentado P hay un programa disyuntivo P' que es una transformación conservativa (bajo stable) de P.*

Prueba:

Sea P un programa aumentado. Por el *teorema 3* podemos construir un programa libre P_1 equivalente a P. Entonces por aplicación de la *proposición 3* a P_1, obtenemos una transformación conservativa P_2 la cual es un programa general. Finalmente remplazaremos en P_2 cada restricción $\bot \leftarrow B$ con $x \leftarrow B \wedge \neg x$, donde no es un átomo en $\mathcal{L}_{P_2}$, para obtener un programa disyuntivo P_3, tal que P_3 es una transformación conservativa de P_2. Finalmente P_3 es un programa disyuntivo el cual es una transformación conservativa de P.■

5.2 Caracterización de conjuntos respuesta

Aquí se presenta uno de los principales resultados de [1] que es el teorema 5 donde se da una caracterización de answer set de programas aumentados en términos de lógica intuicionista. Basados en dicho resultado los autores proponen una definición de answer set para teorías proposicionales generales.

Pearce mostró que una fórmula está vinculada a un programa en la semántica del modelo stable si y solamente, si ella pertenece a cada extensión intuicionisticamente completa y consistente del programa que se forma al agregar sólo átomos negados.

Lema 10. *Sea P un programa disyuntivo. M es un modelo stable de P si y sólo si $P \cup \neg\widetilde{M} \Vdash_I M$.*

Pearce consideró la clase de programas disyuntivos, sin embargo, este resultado fácilmente puede ser generalizado a programas generales, es decir, programas disyuntivos con restricciones.

Lema 11. *Sea P un programa general. M es un modelo stable de P si y sólo si $P \cup \neg\widetilde{M} \Vdash_I M$.*

Prueba:

Sea x un átomo tal que $x \notin \mathcal{L}_P$. Sea $C = \{\bot \leftarrow B_1, ..., \bot \leftarrow B_n\}$ el conjunto de restricciones en P. Sea $P_1 = P \backslash C$. Nótese que P es un programa disyuntivo desde que borramos todas las restricciones del programa.

Entonces M es un modelo stable de $P = P_1 \cup C$ sii, por un resultado bien conocido, M es un modelo stable de $P_1 \cup \{x \leftarrow B_1 \wedge \neg x, ..., x \leftarrow B_n \wedge \neg x\}$ sii, por el *lema 10* y considerando ahora a x en el nuevo lenguaje extendido.

$P_1 \cup \{x \leftarrow B_1 \wedge \neg x, ..., x \leftarrow B_n \wedge \neg x\} \cup \neg\widetilde{M} \cup \{\neg x\} \Vdash_I M$
sii $\{x \leftarrow B_i \wedge \neg x, \neg x\} \equiv_I \{\bot \leftarrow B_i, \neg x\}$ para cada $1 \preceq i \leq n$,
$P_1 \cup \{\bot \leftarrow B_{1,...}, \bot \leftarrow B_n\} \cup \neg\widetilde{M} \cup \{\neg x\} \Vdash_I M$
sii, por definición de P, $P \cup \neg\widetilde{M} \cup \{\neg x\} \Vdash_I M$ sii, por el *lema 2* y sabiendo que $x \notin L_P$, $P \cup \neg\widetilde{M} \Vdash_I M$ como se desea.■

Sin embargo, el resultado de Pearce no considera programas libres. Por ejemplo, si elegimos $P := a \vee \neg a$, P tiene sólo dos modelos stables, a saber $\{a\}$ y $\emptyset$, pero sólo una extensión de P completa y consistente se puede obtener al agregar átomos negados como $\{\neg a\}$, el cual corresponde a $\emptyset$. En nuestra aplicación podemos dar el efecto deseado si nos permitimos agregar literales negadas en general, además de átomos negados. Podemos agregar $\neg a$ para obtener el modelo $\{a\}$, since $\neg\neg a, a \vee \neg a \vdash_I a$. En la próxima sección veremos que la aplicación de Pearce caracteriza actualmente modelos stable y mínimos para programas aumentados.

Lema 12. *Sea P un programa general y M un conjunto de átomos. Tenemos que $P \cup \neg\widetilde{M} \Vdash_I M$ sii $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$.*

Prueba:

Primero supongamos que $P \cup \neg\widetilde{M} \Vdash_I M$, como $A \rightarrow \neg\neg A$ es un teorema intuicionista $P \cup \neg\widetilde{M} \Vdash_I \neg\neg M$. Además $P \cup \neg\widetilde{M} \cup \neg\neg M$ es consistente y como en lógica intuicionista es monótona tenemos que $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$.

Ahora supongamos que $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$, es inmediato que $P \cup \neg\widetilde{M}$ es consistente. Ahora mostramos que $P \cup \neg\widetilde{M} \vdash_I M$. Podemos aplicar el *lema 8* al programa $(P \cup \neg\neg M) \cup \neg\widetilde{M}$ y obtener un programa P' tal que $\mathcal{L}_{P'} \cap L_{\neg\widetilde{M}} = 0$ y $P \cup \neg\widetilde{M} \cup \neg\neg M \equiv_I P' \cup \neg\widetilde{M}$.

Como $P \cup \neg\widetilde{M}U\neg\neg M \Vdash_I M$, entonces $P' \cup \neg\widetilde{M} \vdash_I M$. Claramente $\mathcal{L}_{P' \cup M} \cap \mathcal{L}_{\neg\widetilde{M}} = 0$,asi, podemos aplicar el *lema 2* para obtener $P' \vdash_I M$.

Sea P":= $redu2(P')$. Nótese que P"$\backslash\neg\neg M$ es un programa positivo, la aplicación de $redu1$ y entonces $redu2$ ha borrado todas sus negaciones dentro de sus clausulas. Por la proposición 1, $(P''\backslash\neg\neg M) \vdash_I M$ **(1)**.

$P \cup \neg\widetilde{M} \cup \neg\neg M \equiv_I P' \cup \neg\widetilde{M}$ entonces $P \cup \neg\widetilde{M} \vdash_I (P''\backslash\neg\neg M)$ **(2)**.

Por **1, 2** y transitividad de $\vdash$ obtenemos $P \cup \neg\widetilde{M} \vdash_I M$ como se desea.■

Para el siguiente lema suponemos que S, $\triangle_S$ y M_S son definidos como en la *proposición 3*.

Lema 13. *Sea P un programa libre, M es un conjunto de átomos. Entonces tenemos que $P \cup \triangle_S \cup \neg\widetilde{M_S} \cup \neg\neg M_S \Vdash_I M_S$ sii $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$.*

Prueba:

Supóngamos que $P \cup \Delta_\varphi \cup \neg\widetilde{M_\varphi} \cup \neg\neg M_\varphi \Vdash_I M_\varphi$. Como $M_\varphi = M \cup \varphi(S\backslash M)$ podemos partir los conjuntos $\neg\widetilde{M_\varphi}$ y $\neg\neg M_\varphi$ como dos subconjuntos disjuntos de $\mathcal{L}_P$ y Σ. Esto es, $\neg\widetilde{M_\varphi} = \neg\widetilde{M} \cup \neg[\varphi(S \cap M)]$, y $\neg\neg M_\varphi = \neg\neg M \cup \neg\neg[\varphi(S\backslash M)]$.

Podemos escribir Δ_φ como la unión de dos conjuntos disjuntos $\Delta_M \cup \Delta_{\widetilde{M}}$, donde $\Delta_M = \cup_{a \in M}\{\varphi(a) \leftarrow \neg a., \bot \leftarrow a \wedge \varphi(a).\}$. Nótese que también $\neg\neg M \cup \neg[\varphi(S \cap M)] \vdash_I \Delta_M$. Por tanto, tenemos que $P \cup \Delta_{\widetilde{M}} \cup \neg\widetilde{M} \cup \neg\neg M \cup \neg[\varphi(S \cap M)] \cup \neg\neg[\varphi(S\backslash M)] \Vdash_I M$.

En la prueba, para cada $a \in M$, como fue mostrado anteriormenete, podemos mapear a cada símbolo $x \in \varphi(S \cap M)$ a $\bot$ y cada símbolo $y \in \varphi(S\backslash M)$ a $\top$. Esto conducirá a una prueba para $P \cup \neg\widetilde{M} \cup \neg\neg M \vdash_I M$, como premisa en $\Delta_{\widetilde{M}}$, $\neg[\varphi(S \cap M)]$ y $\neg\neg[\varphi(S\backslash M)]$ son mapeados a teoremas intuicionistas o elementos en $\neg\widetilde{M}$.

Para probar la otra implicación suponemos $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$, de la definición de Δ_φ tenemos que $P \cup \Delta_\varphi \cup \neg\widetilde{M} \cup \neg\neg M$ es consistente y además $P \cup \Delta_\varphi \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$. También nótese que $\Delta_\varphi \cup \neg\widetilde{M} \vdash_I \varphi(S\backslash M)$ y

$\Delta_\varphi \cup \neg\neg M \vdash_I \neg\varphi\,(S \cap M)$,de donde se sigue que $P \cup \Delta_\varphi \cup \neg\widetilde{M} \cup \neg\neg M \cup \neg\,[\varphi\,(S \cap M)] \cup \neg\neg\,[\varphi\,(S\backslash M)] \Vdash_I M \cup \varphi\,(S\backslash M)$ como se buscaba.■

Teorema 5. *Sea P un programa aumentado y M un conjunto de átomos. M es un modelo stable de P si y sólo si $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$.*

Prueba:

Para el programa aumentado P, por el *teorema 3*, hay un programa libre P_1 tal que P y P_1 son fuertemente equivalentes y por tanto tienen los mismos modelos stable.

Asi, M es un modelo stable de P sii M es un modelo stable de P_1 sii, por la *proposición 3*, M_φ es un modelo stable de $P'_1 \cup \Delta_\varphi$ la extensión general de P_1 sii, por el *lema 11*, debido a que $P'_1 \cup \Delta_\varphi$ es un programa general, $P'_1 \cup \Delta_\varphi \cup \neg\widetilde{M}_\varphi \Vdash_I M_\varphi$ sii, por el *lemma 12*, $P'_1 \cup \Delta_\varphi \cup \neg\widetilde{M}_\varphi \cup \neg\neg M_\varphi \Vdash_I M_\varphi$ sii, como $\triangle_\varphi \vdash_I P_1 \equiv P'_1$, $\quad P_1 \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$, sii (*esto último debido a que por el teorema 2, $P_1 \equiv_{Jn} P$, además, al agregar $\neg M \cup \neg\neg M$ a las teorías se puede probar en lógica intuicionista la ley fuerte del tercero excluido y el lema 4. Asi, $P_1 \cup \neg M \cup \neg\neg M \equiv_I P \cup \neg M \cup \neg\neg M$) $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$.*■

El siguiente ejemplo explica el *teorema 5* en una instancia particular.

Ejemplo 8. Consideremos nuevamente el programa aumentado

P: $\quad a \leftarrow \neg\neg a.$
$\quad \neg b \leftarrow c \vee b.$

Sabemos del ejemplo 1 que el conjunto $M = \{a\}$ es un conjunto respuesta para este programa. A partir del *teorema 3* y del *ejemplo 7* construimos el programa libre equivalente:

P_1: $\quad a \vee \neg\, a.$
$\quad \neg b \leftarrow c.$
$\quad \neg b \leftarrow b.$

Para este programa remplazamos los átomos de $S = \{a, b\}$ los cuales aparecen negados en la cabeza de alguna clausula por nuevos átomos, como en la *proposición 3*, para construir un programa aun equivalente. Este programa $P'_1 \cup \triangle_S$ será

P'_1: $a \vee x.$
$\quad y \leftarrow c.$

$\triangle_S$: $\quad x \leftarrow \neg a.$
$\quad \leftarrow a \wedge x.$

$y \leftarrow b.$ $\qquad\qquad$ $y \leftarrow \neg b.$

$\leftarrow b \wedge y.$

Donde $M_S = \{a, y\}$ es el correspondiente answer set para este programa. Ahora podemos aplicar el resultado de Pearce para programas generales, es decir el *lema 11*, y obtener una prueba para la aseveración intuicionistica $P_1' \cup \triangle_S \cup \{\neg x, \neg b, \neg c\} \Vdash_I \{a, y\}$. A partir de este punto obtendremos una prueba para $P \cup \{\neg b, \neg c\} \cup \{\neg\neg a\} \vdash_I a$ como exige el teorema.

Como se describio en en la prueba del *lema 13* podemos incluir sin mucho problema el conjunto $\neg\neg M_\varphi = \{\neg\neg a, \neg\neg y\}$ como premisa extra y remplazamos P_1' con la P_1 original obtenemos la prueba

$P_1 \cup \{x \leftarrow \neg a., \leftarrow a \wedge x., y \leftarrow \neg b., \leftarrow b \wedge y.\} \cup \{\neg b, \neg c, \neg x\} \cup \{\neg\neg a, \neg\neg y\} \vdash_I a.$

Los átomos en $\{x, y\}$ agregados a la trasformación del programa son mapeados a los símbolos $\bot$ y $\top$ respectivamente para eliminarlos de la prueba y obtener

$P_1 \cup \{\bot \leftarrow \neg a., \leftarrow a \wedge \bot ., \top \leftarrow \neg b., \leftarrow b \wedge \top.\} \cup \{\neg b, \neg c, \neg \bot\} \cup \{\neg\neg a, \neg\neg\top\} \vdash_I a.$

de donde, después de algunas simplificaciones, tenemos

$P_1 \cup \{\neg\neg a, \top, \top, \neg b\} \cup \{\neg b, \neg c, \top\} \cup \{\neg\neg a, \top\} \vdash_I a.$

Después de aplicar la eliminación de premisas duplicadas y remplazar P_1 con el P original, finalmente obtenemos $P \cup \{\neg b, \neg c\} \cup \{\neg\neg a\} \vdash_I a$.

Un corolario inmediato del *teorema 5* es la caracterización de equivalencia, bajo la semántica stable en términos de lógica intuicionista.

Corololario 2. *Sean P_1 y P_2 dos programas aumentados definidos bajo el alfabeto $\mathcal{L}$. Entonces $P_1 \equiv P_2$ sii para cada $M \subseteq \mathcal{L}$, $P_1 \cup \neg\widetilde{M} \cup \neg\neg M \equiv_I P_2 \cup \neg\widetilde{M} \cup \neg\neg M$.*

Prueba:

Tenemos que $P_1 \equiv_{stable} P_2$

sii (M es un modelo stable de P_1 sii M es un modelo stable de P_2)

sii ($P_1 \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$ sii $P_2 \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$)

sii, porque $P_1 \cup \neg\widetilde{M} \cup \neg\neg M$ y $P_2 \cup \neg\widetilde{M} \cup \neg\neg M$ son teorías completas de literal, $P_1 \cup \neg\widetilde{M} \cup \neg\neg M \equiv_I P_2 \cup \neg\widetilde{M} \cup \neg\neg M$.■

5.3 Implicaciones encajadas

Ahora, sugerimos una área de aplicación importante para el *teorema 5*: implicaciones encajadas. Una implicación encajada puede tener una clausula por si misma "encajada" en su antecedente. Esto ha sido propuesto por varios investigadores como una extensión natural de la programación lógica. Interpretada intuicionisticamente, dicha regla, constituye un subconjunto propio de la lógica con propiedades semánticas interesantes. N-Prolog es quizás el primer lenguaje de programación lógica que extendió Prolog para permitir implicaciones encajadas.

Gracias a la caracterización de modelos stable dado en el *teorema 5*, podemos definir las semánticas stable de algún programa proposicional P como sigue:

Definición 12. *Sea P alguna teoría proposicional. M es un modelo stable de P si y sólo si $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$.*

En este punto podemos explorar las semánticas stable sobre la base de diferentes ejemplos. consideremos la regla siguiente:

$$nearGrad(S, D) \leftarrow stud\,(S)\,, dept\,(D)\,, course\,(C)\,, (grad\,(S, D) \leftarrow tk\,(S, C)).$$

Esta regla significa: *El estudiante S está próximo a ser graduado del departamento D si hay algún curso C, tal que si S toma C, entonces S será graduado de D.*

Una aplicación importante en particular para el área sugerida es la de *funciones agregadas*. A continuación se describe el problema de agregación y se propone una solución basada en aplicaciones encajadas.

Set−grouping y *agregación* son operaciones poderosas de interés práctico en lenguajes para querys en base de datos. Una función de agregación es una función que mapea un conjunto de algunos valores, por ejemplo, el *máximo* y el *mínimo* en el conjunto, la *cardinalidad* del conjunto, la *suma* de todos sus miembros, etc. Las operaciones de agregación son típicamente de naturaleza no-monotónica, los programas recursivos hacen uso de operaciones de agragación que no pueden ser expresadas naturalmente en lógica de primer orden de forma monotónica.

Gelfond apunta:

Necesitamos encontrar caminos elegantes para acoplar tareas simples, por ejemplo, el conteo de número de elementos que satisfacen alguna propiedad P en una base de datos. Por el momento no tenemos semánticas claras ni implementaciones eficientes de set of de otros agregados usados para este tipo de problemas

Hay algunas investigaciones en este sentido y se tiene evidencia experimental para ejercer algunas propuestas que ayuden a obtener respuestas viables.

La idea de expresar programas de *agregación* vía *programas lógicos con alguna forma de negación como falla* ha sido considerada varias veces. La razón para esto es que la negación como falla es una operación no-monotónica que ha sido investigada extensamente en el campo de la programación lógica durante los últimos 15 años.

La idea principal es considerar una especificación que involucre agregación como un macro para programas aumentados, tomando en cuenta que un programa con agregación usualmente contiene *negaciones*. De tal forma, una traslación ayuda a entender en un nivel declarativo la conección entre varias construcciones relativamente cercanas. Esto es observado en algunos artículos sin embargo, esta estrategia tiene algunos problemas. En este trabajo también proponemos un vía basada en traslación, para lo cual exploraremos el uso de inmersiones en lugar de la negación. En el siguiente ejemplo se muestra cómo las inmersiones pueden ser usadas para expresar la noción del máximo de un conjunto de valores, una forma básica de agregación.

Ejemplo 9. Considérese el siguiente programa que consiste de algunos datos D y una regla R con la intención de calcular el máximo de algunos objetos con una propiedad p dada.

$$D: \quad p(1). \quad p(2). \quad q(3). \qquad R: \quad max(X) \leftarrow p(X) \wedge \forall Y(p(Y) \rightarrow X \geq Y).$$

La notación $\forall Y$ es una abreviación para indicar el reemplazo de la expresión que le sigue por la conjunción finita de instanciaciones de tales expresiones con todos los símbolos constantes de nuestro programa. La relación $X \geq Y$ es interpretada como un predicado built-in. Después del grounding obtendremos el siguiente programa proposicional:

P: p1. $max1 \leftarrow p1 \wedge (p1 \rightarrow 1 \geq 1) \wedge (p2 \rightarrow 1 \geq 2) \wedge (p3 \rightarrow 1 \geq 3)$.
p2. $max2 \leftarrow p2 \wedge (p1 \rightarrow 2 \geq 1) \wedge (p2 \rightarrow 2 \geq 2) \wedge (p3 \rightarrow 2 \geq 3)$.
q3. $max3 \leftarrow p3 \wedge (p1 \rightarrow 3 \geq 1) \wedge (p2 \rightarrow 3 \geq 2) \wedge (p3 \rightarrow 3 \geq 3)$.

Un modelo para este programa será $M = \{p1, p2, q3, max2\}$. $P \cup \neg\widetilde{M} \cup \neg\neg M$ es completa: $p1, p2, q3, \neg p3, \neg max1$ y $\neg max3$ son conocidos y $p1, p2$, $\neg p3$ son suficientes para probar intuicionisticamente $max2$. La consistencia sigue inmediatamente.

También es claro al menos intuitivamente que no hay otro modelo. Como no hay forma de probar que $p3$ y $p2$ son siempre verdad las definiciones para $max1$ y $max3$ no pueden satisfacerse.

Ejemplo 10. En este ejemplo se usa min agregado en varias formas embarazosas para definir la propiedad p sobre el dominio $\{0, 1, 2\}$, en términos del mismo p.

$p(X) \leftarrow min(Y|Y = 1 \vee p(Y), X)$.

Usando la implicación en el cuerpo podemos reescribir esta clausula como el programa:

P: $r(Y) \leftarrow Y = 1 \vee p(Y)$.
$minr(X) \leftarrow r(X) \wedge \forall Y\,(r(Y) \rightarrow X \leq Y)$.
$p(X) \leftarrow minr(X)$.

El único modelo stable de P que puede ser verificado como el anterior corresponde a $\{r(1), minr(1), p(1)\}$.

Pensamos que las implicaciones encajadas pueden hacer la agregación de modelo. El ejemplo sugiere que esta es una línea de investigación interesante.

Chapter 6

Las semánticas estable y mínima

En esta sección consideraremos interpretaciones bivaluadas, modelos mínimos y stable como son utilizados en Programación Lógica. Los modelos mínimos son de interés general por las siguientes razones: primero, cada modelo stable de un programa general es un modelo mínimo. Segundo, están relacionados completamente con circunscripción y la teoría de la lógica default. Tercero, son de interés teórico y práctico de una clase grande de problemas de optimización . Finalmente, la computación de modelos mínimos pueden ser el primer paso a través de la computación de modelos stable de programas generales.

Lema 14. *Para un programa aumentado* P, $P \cup \neg\widetilde{M} \Vdash_C M$ *sii* M *es un modelo mínimo de* P.

Prueba:

Supongamos que $P \cup \neg\widetilde{M} \Vdash_C M$. Es inmediato que M es un modelo de $P \cup \neg\widetilde{M}$. Ahora supongamos que M no es un modelo mínimo de P, entonces existe J, un modelo de P tal que $J \subset M$, así que hay una $a \in (M \backslash J)$. De aquí $\neg a \in \left(\neg J \backslash \neg\widetilde{M}\right)$, de donde $P \cup \neg\widetilde{J} \vdash_I \neg a$, pero $P \cup \neg\widetilde{M} \vdash_I a$ y $P \cup \neg\widetilde{M} \subseteq P \cup \neg\widetilde{J}$, entonces J no es un modelo. Por el contrario: si M es un modelo mínimo de P entonces P es consistente, por tanto $P \cup \neg\widetilde{M}$ también es consistente. Es fácil checar que M, como es mínimo, es el único modelo de $P \cup \neg\widetilde{M}$. Así que $P \cup \neg\widetilde{M} \vdash_C M$.■

El siguiente teorema también es un resultado importante del trabajo que se revisa. Esto proporciona una caracterización de modelos mínimos y stable en términos de lógica intuicionista.

Teorema 6. *Sea P un programa aumentado. M es un modelo minimo y stable de P sii $P \cup \neg\widetilde{M} \Vdash_I M$.*

Prueba:

Supongamos que M es un modelo stable y mínimo de P, así $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$, porque M es un modelo stable de P. Como M es un modelo mínimo, por el lema 14, sabemos que $P \cup \neg\widetilde{M} \vdash_C M$, entonces $P \cup \neg\widetilde{M} \vdash_I \neg\neg M$. Por la última afirmación y como $P \cup \neg M \cup \neg\neg M \Vdash_I M$,tenemos que $P \cup \neg\widetilde{M} \vdash_I M$ y $P \cup \neg\widetilde{M}$ es consistente, es decir, $P \cup \neg\widetilde{M} \Vdash_I M$. Para el inverso, supongamos $P \cup \neg\widetilde{M} \Vdash_I M$ así $P \cup \neg\widetilde{M} \vdash_I \neg\neg M$, entonces $P \cup \neg\widetilde{M} \cup \neg\neg M$ es consistente en lógica intuicionista y M es un modelo stable de P. En el otro caso, si $P \cup \neg\widetilde{M} \Vdash_I M$ entonces $P \cup \neg\widetilde{M} \vdash_C M$ y sabemos que $P \cup \neg\widetilde{M}$ es consistente en lógica intuicionista, lo cual implica consistencia en lógica clásica. Por el *lema 14*, tenemos que M es un modelo mínimo de P.■

Proposición 8. *Si M es un modelo mínimo y stable de P, entonces M es un modelo stable y mínimo de P.*

Prueba:

Supóngase que M es un modelo mínimo y stable de P. Pero P no es un modelo stable y mínimo de P, entonces, hay un subconjunto propio $M_1 \subset M$, tal que M_1 es un modelo stable mínimo de P. Pero esto no es correcto porque M es un modelo mínimo de P.■

Ejemplo 11. El inverso de la *proposicón 8* no es verdad. Sea P el programa:

$$\begin{aligned} P: \quad & a \vee \neg a. \\ & b \leftarrow a. \\ & b \leftarrow \neg b. \end{aligned}$$

El único modelo stable de P es $\{a, b\}$ y su único modelo mínimo es $\{b\}$.

El siguiente corolario nos dice de que manera la clase de modelos stable no es más expresiva que la clase de modelos stables mínimos.

Corolario 3. *Para cada programa aumentado P, existe un programa disyuntivo compatible P' tal que los modelos stables y mínimos de P' restringidos a $\mathcal{L}_P$, son cada uno modelos stables de P.*

Prueba:

Se sigue por aplicación del *teorema 4* y el hecho conocido que para programas disyuntivos los modelos stable son modelos mínimos.■

El siguiente teorema es similar al correspondiente para las semánticas stable, es decir, el *teorema 1.* La prueba que se muestra es más complicada que la utilizada para probar el teorema en otros trabajos como pudimos observar en el *ejemplo 12.* Además, usamos la lógica 3-valuada descrita en las semánticas de Kripke para HT.

Teorema 7. *Sean F_1 y F_2 dos fórmulas proposicionales. Entonces F_1 y F_2 son fuertemente equivalentes con respecto a las semánticas del modelo stable y mínimo si y solamente, sí F_1 y F_2 son equivalentes en G_3.*

Prueba:

Sea F_1 y F_2 dos fórmulas proposicionales. Si F_1 y F_2 son equivalentes en G_3 entonces para algún F, $F_1 \cup F$ y $F_2 \cup F$ son equivalentes en G_3. Probaremos, suponiendo que M es un modelo mínimo y stable de F_1 que es un modelo mínimo y stable de F_2. Como $F_1 \equiv_{G_3} F_2$ es inmediato que M es un modelo stable de F_2, por el *teorema 1.* Ahora, como M es un modelo mínimo, $F_1 \cup \neg M \Vdash_C M$, pero en particular, $F_1 \equiv_C F_2$ y además $F_2 \cup \neg M \Vdash_C M$. el mismo argumento proporciona todos los modelos mínimos y stable de F_2 son también modelos mínimos y stable de F_1. Así, $F_1 \cup F$ y $F_2 \cup F$ son equivalentes en la semántica mínima y stable, y entonces F_1 y F_2 son fuertemente equivalentes.

Por el contrario, supongamos que F_1 y F_2 son equivalentes en G_3, entonces por la *proposición 6*, hay una interpretación la cual modela F_1 pero no modela F_2, la cual por la *proposición 7*, implica que ellas no son fuertemente equivalentes con respecto a las semánticas mínimas y stable.■

Ejemplo 12. Consideremos los programas $P_1 = \{a \vee \neg a\}$ y $P_2 = \{a \leftarrow a\}$. Dichos programas no son fuertemente equivalentes con respecto a las semánticas stable simplemente porque no son equivalentes. Para la semántica de modelos stable y mínimos la situación es más complicada. Ambos programas son equivalentes porque solamente M=$\emptyset$ es el modelo mínimo y stable de

cada programa. Sin embargo, ellos no son fuertemente equivalentes porque si agregamos la clausula $a \leftarrow \neg a$ para cada programa, los dos programas son muy diferentes bajo stable.

Chapter 7

El que calla otorga

En el proceso de comunicación de los seres humanos es común encontrar en diversas manifestaciones el caso en que el *silencio* o las *acciones* de una persona, implican la toma de desiciones en otra, es decir, en algunos casos decidimos y trabajamos con información incompleta o con lo que pensamos pueden ser las intenciones de las otras personas.

El *hablante* tiene que interpretar el silencio que el *oyente* envía con cierta intención. Para facilitar la descripción entre este tipo de interlocución donde el destinatario se convierte en emisor y el emisor en destinatario a través de la interpretación del silencio, distinguiremos a quien *interpreta el silencio* en la comunicación como el *Agente Lector (AL)* **,** esto es porque la semiótica prefiere este término para el receptor incluso si se trata de una fotografía o pintura [14]. Al que envia una señal mediante el silencio lo identificaremos como el *Agente Maya (AM)***.**

Considerando que el silencio puede ser trasmitido por parte del AM voluntariamente (+) o involuntariamente (-) y recibido por parte del AL también voluntariamente (+) o involuntariamente (-) [15] y tomando en cuenta que este último puede atribuir al AM una intención (IAM) referente al envio de la señal, tenemos diferentes posibilidades, las cuales se muestran en la siguiente matriz :

	AM	AL	IAM
1	+	+	+
2	+	+	−
3	+	−	(+)
4	+	−	(−)
5	−	+	+
6	−	+	−
7	−	−	(+)
8	−	−	(−)

Situaciones significativas en relación a la matriz de posibilidades:

1. Un compañero (AL) de equipo de trabajo le pregunta a otro (AM) ¿Terminaste tu parte del proyecto?. El AM se queda callado y envía *intencionalmente* el signo *no*. El AL *reconoce* que el AM *contestó no* con toda *intención*. Veamos otro ejemplo.

A un estudiante que no ha sido regular durante un curso ni en la entrega de sus trabajos y que no lleva a la entrevista con el profesor lo que se comprometió a entregar, el profesor (AL) le pregunta: ¿ Quieres reprobar la materia ?. El estudiante se queda callado y molesto *envía intencionalmente* el signo *si* retirándose del lugar. El AL *reconoce* que el AM *contestó si* clara e intencionalmente.

2. Un jefe de oficina (AL) le pregunta a su subordinado (AM) ¿ Hiciste lo que te encargué ?, el subordinado envía intencionalmente el signo *no*. El jefe recibe la señal e *interpreta* que el subordinado ha mandado involuntariamente la señal, es decir, el jefe piensa que el subordinado *no sabe* si el avance que logró podrá ser considerado como un *sí* o como un *no* y que por eso se quedó callado.

3. En un bazar una mujer (AL) le pregunta a su pareja (AM) si está de acuerdo en comprar un portarretratos, en ese preciso momento su pareja se distrae con otro puesto donde descubre algo que buscaba hace pocos dias al tiempo que envía voluntariamente el signo *no*. La mujer *no capta* voluntariamente el signo, y por lo tanto no puede saber si su pareja *dijo que si o dijo que no*. Nada excluye que *más tarde* la mujer se dé cuenta de que ha recibido un mensaje intencional y reconozca un *no*.

4. Este caso puede ser el mismo que el anterior solo que la mujer (AL) *cree* que su pareja (AM) *no la escuchó*, es decir, que le envió una señal involuntariamente. Más tarde asume involuntariamente que el envio de la

señal fue sin intención.

5. Un programador (AM) trabajando en la computadora está a punto de ser interumpido por su jefe de área (AL) quien llega hasta él y le dice "porfavor atiende esta orden de programación", a lo cual el programador no contesta porque está concentrado en la parte más delicada de la solución a un problema, enviando involuntariamente la señal no. El jefe de área acepta e interpreta equivocadamente la señal como un *no* y pensando tomar en cuenta la actitud del programador en la próxima entrega de incentivos lleva la orden a otro programador.

6. Este puede ser el mismo caso que el anterior, sólo que el jefe de área acepta la señal como un *no momentáneo* o involuntario, pues se percata de la ocupación del programador.

7. Este caso es parecido al número 3, con la variante de que aqui, a la pareja que se distrae repentinamente, no le da tiempo de contestar la pregunta y envía sin querer la señal *no*.

8. Caso análogo al anterior sólo que más tarde la mujer se da cuenta que su pareja se distrajo y la señal que envió fue *sin intención*.

De de las situaciones significativas antes expuestas, la primera es el ejemplo más claro de trasmisión de una negación o afirmación a través del silencio ya que la *intención* del AM es recibida correctamente por el AL como si se tratara de un código, los otros casos son ambigüos y la diferencia entre una y otra la da el contexto.

Un proceso de comunicación en el que no exista código, y por consiguiente en el que no exista significación, queda reducido a un proceso de estímulo-respuesta. Los estímulos no se adecuan a una de las definiciones más elementales del signo, la que dice que "se pone en lugar de otra cosa". El estímulo no se pone en lugar de otra cosa, sino que "provoca directamente" esta otra cosa. Una luz deslumbrante que me obliga a cerrar los ojos es una cosa distinta de una orden verbal que me imponga cerrar los ojos. En el primer caso cierro los ojos sin reflexionar; en el segundo, antes que nada he de entender la orden y decodificar el mensaje (proceso sígnico) para luego decidir si obedezco.

El estudio de los signos y su funcionamiento se llama semiótica o semiología. La semiótica tiene tres áreas principlaes [15]:

1. *El signo mismo.* Es el estudio de diferentes tipos de signos, de su manera de llevar significados y de relacionarse con quienes los usan. Porque los signos son creaciones humanas, y solo pueden ser comprendidos en función

del uso que la gente haga de ellos.

2. *Los códigos o sistemas de organización de los signos.* Aquí se estudia cómo se ha desarrollado una variedad de códigos para satisfacer las necesidades de una sociedad o de una cultura, o para explotar los canales de comunicación disponibles para su transmisión.

3. *La cultura* dentro de la cual operan estos códigos y signos. Ésta, a su vez, depende para su propia existencia y forma, del uso de estos códigos y signos.

Las tres maneras de considerar el signo son: semántica, sintáctica y pragmática [15]:

Semántica, el signo se considera en relación con lo que significa.

Sintáctica, el signo se considera como susceptible de ser insertado en secuencias de otros signos, según unas reglas combinatorias.

Pragmática, el signo se considera en relación con sus propios orígenes, los efectos sobre sus destinatarios o por ejemplo, la utilización que hacen de ellos. Esta tercera dimensión es la más oscura.

Con la intención de acotar la posibilidad de generar deducciones a partir del *silencio* ya que se trata de un signo, se presentan a continuación dos ejemplos del uso de este concepto.

7.1 El problema del criminal

Aanalicemos el siguiente acertijo:

Una persona "d" fue asesinada y hay tres sospechosos de dicho asesinato. La persona "a" dice que ella no lo hizo, también dice que la persona "b" era amigo de la victima pero que la persona "c" es quien realmente odiaba a la victima. La persona "b" dice que estaba fuera de la ciudad el dia del asesinato y que además el no conoció a dicho tipo. La persona "c" dice que es inocente y que vió a las personas "a" y "b" con la victima justo antes del asesinato. ¿Quién es el asesino?.

Debemos suponer que todas las personas involucradas dicen la verdad, excepto posiblemente, el asesino. La historia y testimonio de estas cuatro personas es registrado en el siguiente programa para smodels. Smodels es un programa que encuentra, en caso de existir, el conjunto respuesta o modelo estable de un programa lógico.

```
% ACERTIJO ORIGINAL:
% LA PERSONA "d" FUE ASESINADA Y LAS PERSONAS
% "a", "b" Y "c" SON SOSPECHOSAS.
persona(a). persona(b). persona(c). persona(d).
% La persona a dice:
dice(a,i(a)).
dice(a,o(c,d)).
dice(a,am(b,d)).
% La persona b dice:
dice(b,f(b)).
dice(b,nc(b,d)).
% La persona c dice:
dice(c,i(c)).
dice(c,j(a,d)).
dice(c,j(b,d)).
% En la siguiente regla, s(F) se lee "se sostiene F",
% al estilo de un investigador.
% Todos dicen la verdad excepto posiblemente el asesino.
s(F):-dice(P,F), not as(P).
% LAS REGLAS SIGUIENTES, CONTIENEN CONOCIMIENTO CON
% SENTIDO COMUN ACERCA DEL SIGNIFICADO DE LAS
% RELACIONES USADAS POR LOS SOSPECHOSOS
% Relaciones simetricas y transitivas
s(j(A,B)):-persona(A),persona(B),s(j(B,A)).
s(j(A,B)):-persona(A),persona(B),persona(C),s(j(A,C)),s(j(C,B)).
s(am(A,B)):-persona(A),persona(B),s(am(B,A)).
% SE CONSIDERAN LAS RESTRICCIONES SIGUIENTES:
% Los asesinos no son inocentes.
:-persona(P),s(i(P)),as(P).
% Una persona no puede ser vista junto con
% gente que esta fuera de la ciudad.
:-persona(P),persona(P1),s(f(P)),s(j(P,P1)).
% Los amigos se conocen.
:-persona(A),persona(B),s(am(A,B)),s(nc(A,B)).
% Una persona que esta fuera de la ciudad no puede ser la asesina.
:-persona(P),s(f(P)),as(P).
% LAS SIGUIENTES DOS DECLARACIONES
```

% SON DIRECTIVAS PARA SMODELS
% Exactamente uno de los sospechosos es el asesino.
1{ as(a), as(b), as(c) }1.
% Se ocultan todos los atomos excepto as(F).
hide.show as(F).

El modelo obtenido para este programa es: *as* (*b*).

Esto significa que de acuerdo al testimonio y las reglas de conocimiento con sentido común anteriores, el asesino es la persona "b".

¿Qué obtendríamos si alguno de los sospechosos hubiese callado?

Si el agente que investiga el caso adopta la regla *"El que calla, deja vacía su declaración y sólo se usa el testimonio de las otras personas"*, el testimonio original del que calla es sustituido por el vacío.

Los resultados que se obtienen son:

1) Si calla "a", entonces se eliminan las clausulas del tipo *dice(a,F)*, se corre lparse y smodels, dando como resultado el modelo *as(b)*.

2) Si calla "b", entonces se eliminan las clausulas del tipo *dice(b,F)*, se corre lparse y smodels, dando como resultado el modelo *as(c)*.

3) Si calla "c", entonces se eliminan las clausulas del tipo *dice(c,F)*, se corre lparse y smodels, dando como resultado el modelo *as(a)*.

Como podemos darnos cuenta, se habría llegado al mismo resultado del acertijo original si la persona "a" hubiese permanecido callada. Desde el punto de vista del resultado obtenido, intuitivamente podemos decir que el programa que se obtiene al borrar las clusulas del tipo *dice(a,F)* del programa original, es una simplificación del primero.

Ahora, analizaremos otra posibilidad.

Si el agente que investiga el caso adopta la regla: *"El que calla, cede y le da la razón a los demás"*, el testimonio original del que calla es sustituido por el testimonio de los otros adoptado por él mismo. Con esta modalidad tenemos las sustituciones siguientes:

Si "a" calla, entonces se eliminan las clausulas del tipo *dice(a,F)* del programa original y se ingresan las siguientes:

dice(a,f(b)).
dice(a,nc(b,d)).
dice(a,i(c)).
dice(a,j(a,d)).
dice(a,j(b,d)).

Ejecutando lparse y smodels, obtenemos como resultado que *¡ No se encuentran modelos !*

Si "b" calla, entonces se eliminan las clausulas del tipo *dice(b,F)* del programa original y se ingresan las siguientes:

dice(b,i(a)).
dice(b,o(c,d)).
dice(b,am((b,d)).
dice(b,i(c)).
dice(b,j(a,b)).
dice(b,j(b,d)).

Ejecutando lparse y smodels, obtenemos como resultado el modelo *as(b).*

Si "c" calla, entonces se eliminan las clausulas del tipo *dice(c,F)* del programa original y se ingresan las siguientes:

dice(c,i(a)).
dice(c,o(c,d)).
dice(c,am(b,d)).
dice(c,f(b)).
dice(c,nc(b,d)).

Ejecutando lparse y smodels, obtenemos como resultado que *¡ No se encuentran modelos !*

Con esta modalidad sólo se obtiene respuesta en el caso en que la persona "b" es la que calla y el resultado es el mismo que el del acertijo original, es decir que por virtud de la regla utilizada, la persona "b" resulta culpable aun cuando no presente su testimonio. Intuitivamente, el programa modificado y el original son equivalentes desde el punto de vista del resultado obtenido.

7.2 El problema de los cinco discos

La princesa Dahizé, fue pedida en matrimonio por tres príncipes ricos e inteligentes, su padre, el rey Cassim estaba indeciso, quería ganar un yerno pero no dos rencorosos enemigos y ya que los tres eran inteligentísimos, pues habían resuelto todos los problemas complicados, fueron sometidos a la prueba de los cinco discos por un conocedor de magia y de ocultismo, el cual les dijo: AQUI HAY CINCO DISCOS DE MADERA MUY FINA, DEL

MISMO TAMAÑO Y DE IDENTICO PESO, DOS DE ELLOS SON NEGROS Y TRES BLANCOS. LES VENDARON LOS OJOS A LOS TRES PRINCIPES, EL VIEJO MAGO TOMO AL AZAR TRES DE LOS CINCO DISCOS Y COLGO UNO A LA ESPALDA DE CADA UNO DE LOS PRETENDIENTES. EL SABIO LES DIJO: EL QUE DESCUBRA EL COLOR DEL DISCO QUE LE CAYO EN SUERTE, SERA DECLARADO VENCEDOR Y SE CASARA CON LA BELLA PRINCESA. EL PRIMER INTERROGADO PODRA VER LOS DISCOS DE LOS OTROS DOS COMPETIDORES, EL SEGUNDO PODRA VER EL DISCO DEL ULTIMO Y EL ULTIMO TENDRA QUE FORMULAR SU RESPUESTA SIN VER NADA. EL PRIMERO Y EL SEGUNDO DIERON SU RESPUESTA EN SECRETO Y NO ACERTARON, EN CADA CASO EL REY ANUNCIABA EL FRACASO EN VOZ ALTA PARA QUE SE ENTERARAN LOS DEMAS. EL TERCERO CON LOS OJOS AUN VENDADOS SE ACERCO AL TRONO Y DIJO EN VOZ ALTA CUAL ERA EL COLOR EXACTO DE SU DISCO.¿Cuál fue la res-puesta del último príncipe?.

Estableciendo los hechos, algunas reglas y restricciones, tenemos el siguiente programa en smodels:

```
% Hay tres príncipes, dos discos negros y tres blancos
principe(a).
principe(b).
principe(c).
color(blanco).
color(negro).
% Reglas
ve(a,tiene(b,D)):-color(D).
ve(a,tiene(c,D)):-color(D).
ve(b,tiene(c,D)):-color(D).
% Restricciones
:-principe(A),principe(B),principe(C),tiene(A,negro),
  tiene(B,negro),tiene(C,negro).
:-principe(a),principe(b),color(D),ve(b,tiene(a,D)).
:-principe(a),principe(c),color(D),ve(c,tiene(a,D)).
:-principe(b),principe(c),color(D),ve(c,tiene(b,D)).
% Exactamente se trata de un color
1{tiene(c,negro),tiene(c,blanco)}1.
```

hide.show tiene(c,F).

El modelo encontrado para este programa es: *tiene(c,blanco).*

Analicemos el problema: Se parte del hecho de que los principes son muy inteligentes y sólo hablan con certeza. Si el primer príncipe calla, lo que aporta con su silencio al segundo príncipe es que *no vio dos discos negros.* Si el segundo príncipe calla, la nueva información que aporta al tercer príncipe es *que vio un disco blanco* porque si habría visto uno negro él tendría uno blanco y habría acertado.

Decir que el primer principe calla equivale a cambiar las últimas directivas por:

1{tiene(b,negro),tiene(b,blanco)}1. show tiene(b,F).

de donde obtenemos el modelo: *tiene(b,blanco).*

Decir que el primero y el segundo príncipe callan, equivale a dejar las directivas como están en el programa completo de donde se obtiene el modelo: *tiene(c,blanco).*

Chapter 8

Conclusiones

Este trabajo es una continuación del trabajo iniciado por D. Pearce en [10].

Hay una pregunta que no hemos contestado todavia, en el *corolario 3* dijimos de alguna manera que la clase de modelos stable no es más expresiva que la clase de modelos stable mínimos. ¿ Es la clase de modelos stable y mínimos más expresiva que la clase de modelos stable ?. Si la respuesta fuera no, lo cual corresponde a nuestra conjetura, entonces ambas semánticas podrán ser equivalentes desde el punto de vista del poder de problemas representados. Entonces tendríamos la posibilidad de hacer un intercambio entre esos dos paradigmas. Si la respuesta fuera si entonces los modelos que son mínimos y stables podrán ser más poderosos que los modelos sólo stable. Esto podrá abrir una nueva linea de investigación sobre la clase de problemas que pueden ser expresados por modelos stable y minimos, no por modelos sólo stable.

En relación al concepto de *El que calla otorga*, es necesario extender el programa smodels o el programa lparse con el fin de incluir metareglas que permitan su utilización de diversas maneras, durante la búsqueda de modelos. También es necesario explorar más la idea de *El que calla otorga* para analizar su posible formalización y su utilización en bases de datos deductivas, en las áreas de interacción hombre-máquina, en planeación y en la comunicación de agentes, por mencionar algunas.

Chapter 9

Anexos

9.1 Prueba del teorema 5.

Antes de escribir la prueba del teorema 5 que se encuentra en [16], escribiremos algunas reglas para realizar las transformaciones de una clase de programas lógicos a otra.

I). Obtención de un programa libre (P_L) a partir de un programa aumentado (P_A).

$P_L = AugFree(P_A)$.

Esta transformación consiste en la eliminación de conjunciones en la cabeza de la cláusula, disyunciones en el cuerpo y átomos con dos o más negaciones.

$A \wedge B \leftarrow C \equiv_{G_3} \{A \leftarrow C, B \leftarrow C\}$

$A \leftarrow B \vee C \equiv_{G_3} \{A \leftarrow B, A \leftarrow C\}$

$A \leftarrow \neg\neg B \leftarrow C \equiv_{G_3} A \leftarrow \neg B \wedge C$

$A \leftarrow \neg\neg B \wedge C \equiv_{G_3} A \vee \neg B \leftarrow C$

II). Obtención de un programa disyuntivo (P_D) a partir de un programa general (P_G).

$P_D = GenDisy(P_G)$.

$P_G = P_D \cup C = P_D \cup \{p \leftarrow B \wedge \neg p | (\bot \leftarrow B) \in C\}$. Donde C es un conjunto de restricciones de P_G y p es un átomo nuevo reservado en el alfabeto $\mathcal{L}_R$.

III). Obtención de un programa general (P_G) a partir de un programa libre (P_L).

$P_G = FreeGen(P_L) = P' \cup \Delta_S$

Dado un programa libre P_L, sea S el conjunto que contiene a todos los átomos a tal que $\neg a$ aparece en la cabeza de alguna cláusula en P, y sea φ un afunción biyectiva $\varphi : S \rightarrow \mathcal{L}_R$, tal que asigna un nuevo átomo reservado a cada elemneto en S. Sea P' el programa obtenido de P reemplazando cada ocurrencia de $\neg a$ con $\varphi(a)$ para cada átomo $a \in S$, y sea $\Delta_S = \cup_{a \in S} \{\varphi(a) \leftarrow \neg a,\ \ \bot \leftarrow a \wedge \varphi(a)\}$.

Ahora enunciaremos y probaremos el teorema.

Teorema 5. Sea P_A un programa aumentado y M un conjunto de átomos. M es un answer set de $P_A \leftrightarrow P_A \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$.

Prueba:

El conjunto M es un anwer set de un P_A

$\updownarrow$ **Tr.** $P_A \equiv^f_{stable} P_L$

M es un answer set de $P_L = AugFree(P_A)$

$\updownarrow$ **Prop**^{n}. Sea P_L un programa libre, $P_G = FreeGen(P_L)$ es una extensión conservativa fuerte de P_L.

M_S es un answer set de $P_G = FreeGen(P_L)$

$\updownarrow$ **Lema.** Sea P_G un programa general. $P_D = GenDisj(P_G)$ es una extensión conservativa fuerte de P_G.

M_S es un answer set de $P_D = GenDisj(P_G)$

$\updownarrow$ **Pearce** Sea P_D un programa disyuntivo y M_S un conjunto de átomos

$P_D \cup \neg\widetilde{M_S} \Vdash_I M_S$

$\updownarrow$ **Lema.** Sea P_G un programa general y M_S un conjunto de átomos. $P_D = GenDisj(P_G)$

$P_G \cup \neg\widetilde{M_S} \Vdash_I M_S$

$\updownarrow$ **Lema.** Sea P_G un programa general y M_S un conjunto de átomos.

$P_G \cup \neg\widetilde{M_S} \cup \neg\neg M_S \Vdash_I M_S$

↕ **Lema.** Sea P_L un programa libre y M_S un conjunto de átomos. $P_G = FreeGen(P_L)$

$P_L \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$.

↕ **Lema.** Sea P_A y M un conjunto de átomos $P_L = AugFree(P_A)$

$P_A \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$.

9.2 Expresibilidad en reglas disyuntivas.

La clase de todos los programas lógicos disyuntivos positivos $\mathcal{D}$, incluye a las que hemos identificado antriormente como disyuntivos, además de otros, de la siguiente manera [17]:

$$\begin{array}{ll} \mathcal{D}^{+} & (H_n = B_n = \oslash) \\ \mathcal{D}^{h+} & (H_n = \oslash) \\ \mathcal{D}^{b+} & (B_n = \oslash) \end{array}$$

Dichos conjuntos cumplen las siguientes propiedades: $\mathcal{D}^{+} \subset \mathcal{D}^{h+} \subset \mathcal{D}$ y $\mathcal{D}^{+} \subset \mathcal{D}^{b+} \subset \mathcal{D}$

La semántica del modelo stable de los programas lógicos disyuntivos es obtenida vía la reducción de Gelfond-Lifschitz de un programa lógico disyuntivo P, para el cual hay un candidato M. El programa reducido $P^M = \{A \leftarrow C \mid A \vee \tilde{} B \leftarrow C \wedge \tilde{} D \in P, B \subseteq M, \text{and } D \cap M = \oslash\}$

Un modelo M de un programa lógico disyuntivo P es stable si M es un modelo mínimo de P^M, es decir, $M = Mm(P^M)$.

Una clase C de programas lógicos tiene las propiedades básicas siguientes:

1. la clase C es cerrada bajo uniones, es decir, dados dos programas P y P' de C, entonces también $P \cup P' \in C$.

2. La clase C tiene un operador semántico Sem_C asociado con ella. El operador Sem_C asigna un conjunto de interpretaciones $I \subseteq Hb\,(P)$ para cada programa P de C.

La semántica de un programa lógico disyuntivo P es para distinguir los modelos de P que son mínimos de la siguiente manera:

- Una interpretación I de P es simplemente un subconjunto de $Hb(P)$ y una regla $A \leftarrow C$ de P es satisfecha en una interpretación $I \subseteq Hb(P)$ si $C \subseteq I$ implica $A \cap I \neq \emptyset$.

- Un modelo M de P es un conjunto de átomos $M \subseteq Hb\,(P)$ si todas las reglas de P se satisfacen en M. Un modelo M de P es un modelo mínimo de P si no hay un modelo M' de P tal que $M' \subset M$.

Una función Tr se llama de traslación porque transforma los progrmas P de una clase C en programas $Tr\,(P)$ de otra clase C'.

La longitud de un programa P en símbolos, se denota por $\|P\|$.

Definición. Dadas dos clases de programas lógicos C y C' cerradas bajo la unión y los operadores semánticos correspondientes Sem_C y $Sem_{C'}$, una función de traslación $Tr : C \rightarrow C'$ es

- Polinomial (P) si para todo programa lógico $P \in C$, el tiempo requerido para calcular la traslación $Tr\,(P) \in C'$ es polinomial en $\|P\|$.

- Fiel (F) si

i) Para todo programa lógico $P \in C$, la base $Hb(P) \subseteq Hb\,(Tr\,(P))$ y

ii) Los modelos/interpretaciones en $Sem_C\,(P)$ y $Sem_{C'}\,(Tr\,(P))$ están en una correspondencia uno a uno y coincide totalmente con $Hb\,(P)$

-Modular (M) si

i) Para todo programa lógico $P_1 \in C$ y $P_2 \in C$, la traslación $Tr\,(P_1 \cup P_2) = Tr(P_1) \cup Tr(P_2)$ y

ii) $C' \subset C$ implica que la traslación $Tr\,(P') = P'$ para todo programa lógico $P' \in C'$.

Una función de traslación $Tr : C \rightarrow C'$ es PFM si satisface los tres criterios. Si tal función de traslación existe, escribimos $C \rightarrow_{PFM} C'$ y consideramos que C' es tan expresiva como C. Si tal función no existe se usa la notación $C \nrightarrow_{PFM} C'$.

Una clase C es menos expresiva que C' si $C \rightarrow_{PFM} C'$ y $C' \nrightarrow_{PFM} C$, lo cual se denota como $C \Longrightarrow_{PFM} C'$.

Las clases C y C' son igualmente expresivas si $C \rightarrow_{PFM} C'$ y $C' \rightarrow_{PFM} C$, lo cual se denota como $C \leftrightarrow_{PFM} C'$.

Las clases C y C' son mutuamente incomparables si $C \nrightarrow_{PFM} C'$ y $C' \nrightarrow_{PFM} C$, lo cual es denotado por $C \nleftrightarrow_{PFM} C'$.

Recordemos que $\mathcal{D}^+ \subset \mathcal{D}^{h+} \subset \mathcal{D}$, de donde obtenemos que:

$$\mathcal{D}^+ \rightarrow_{PFM} \mathcal{D}^{h+} \quad \text{y} \quad \mathcal{D}^{h+} \rightarrow_{PFM} \mathcal{D}$$

estableciendo que $\mathcal{D}^+ \Rightarrow_{PFM} \mathcal{D}^{h+}$, ahora, como $\mathcal{D} \rightarrow_{PFM} \mathcal{D}^{h+}$ implicamos que:

$$\mathcal{D} \leftrightarrow_{PFM} \mathcal{D}^{h+}$$

La prueba de los siguientes teoremas se encuentra en [17]:

$\mathcal{D}^{h+} \nrightarrow_{FM} \mathcal{D}^+$,
$\mathcal{D} \rightarrow_{PFM} \mathcal{D}^{h+}$ ... (1)

Como $\mathcal{D}^+ \subset \mathcal{D}^{b+} \subset \mathcal{D}$ tenemos que

$\mathcal{D}^+ \rightarrow_{PFM} \mathcal{D}^{b+}$ y
$\mathcal{D}^{b+} \rightarrow_{PFM} \mathcal{D}$ ... (2)

De (1) y (2), tenemos que $\mathcal{D}^{b+} \rightarrow_{PFM} \mathcal{D}^{h+}$, lo cual indica que los programas lógicos disyuntivos de cuerpo positivo son estrictamente menos expresivos que los programas disyuntivos de cabeza positiva.

Otros teoremas, son:
$\mathcal{D} \nrightarrow_{FM} \mathcal{D}^{b+}$ y
$\mathcal{D}^{b+} \nrightarrow_{PFM} \mathcal{D}^+$

9.3 Resumen general.

■ DEFINICIONES BÁSICAS.

Lógica Proposicional:

♦ Un alfabeto $\mathcal{L}$ es un conjunto finito de símbolos proposicionales. Si F es una fórmula entonces el alfabeto de F, denotada por $\mathcal{L}_F$, es el conjunto de símbolos proposicionales que ocurren en F.

♦ Una $\wedge \vee \neg$*fórmula* es una fórmula cons-truida sólo con los conectivos lógicos $\{\wedge, \vee, \neg\}$ usados de forma arbitraria debidamente anidados. Si A_1,A_2 y A_3 son conjuntos de átomos, no todos vacíos, entonces:

$\wedge(A_1 \cup \neg A_2 \cup \neg\neg A_3)$ es una conjunción simple y $\vee(A_1 \cup \neg A_2 \cup \neg\neg A_3)$ es una disyunción simple.

♦ Una cláusula es una fórmula de la forma: $H \leftarrow B$ donde H y B son fórmulas arbitrarias en principio, llamadas cabeza y cuerpo de la cláusula respectivamente.

♦ Sea P un programa y M un conjunto de átomos tales que $M \subseteq \mathcal{L}_p$ entonces definimos $\widetilde{M}=\mathcal{L}_p \backslash M$.

■ LOGICAS

♦ Lógica Intuicionista:

La lógica intuicionista puede ser definida en términos de los siguientes esquemas de axioma (Lloyd):

1.$A \rightarrow (B \rightarrow A)$
2.$(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
3.$A \wedge B \rightarrow A$
4.$A \wedge B \rightarrow B$
5.$A \rightarrow (B \rightarrow (A \wedge B))$
6.$A \rightarrow (A \vee B)$
7.$B \rightarrow (A \vee B)$
8.$(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow (A \vee B \rightarrow C))$
9.$(A \rightarrow B) \rightarrow ((A \rightarrow \neg B) \rightarrow \neg A)$
10.$\neg A \rightarrow (A \rightarrow B)$

Modus Ponens es la única regla de inferencia, es decir, si tenemos A y $A \rightarrow B$ entonces podemos deducir B.

Nótese que si agregamos $(\neg A \rightarrow A) \rightarrow A$ a estos esquemas, se obtiene la lógica clásica.

♦ Lógicas Intermedias:

Las lógicas intermedias, super-intuicionista o simplemente si-lógica, se puede ver de la sigui-ente manera:

$I \subseteq J_n \subseteq ... \subseteq G_{i+1} \subseteq G_i \subseteq ... \subseteq G_3 \subseteq G_2 = C$

Decimos que una fórmula A es demostrable en X, denotado como $\vdash_X A$, si usa el conjunto definido de átomos y reglas de inferencia correspondientes tanto como sea posible para obtener la fórmula A.

♦ Lógica de Jankov:

Otra lógica axiomática importante es la ló-gica de Jankov (Jn), la cual es obtenida al agregar al conjunto de axiomas intuicionistas el nuevo esquema de axioma: $\neg A \vee \neg\neg A$.

Este axioma que caracteriza la lógica de Jankov es también conocido como la ley del tercero excluido débil.

■ MODELO STABLE

♦ Un conjunto X de átomos satisface una fórmula básica F, denotado por $X \vDash F$, recursivamente como sigue:

Para F elemental, $X \vDash F$ si $F \in X$ o $F = \top$.

$X \vDash F \wedge G$ si $X \vDash F$ y $X \vDash G$.

$X \vDash F \vee G$ si $X \vDash F$ o $X \vDash G$.

♦ Sea P un programa básico. Un conjunto de átomos X es cerrado bajo P si para cada clausula $H \leftarrow B \in P$, $X \vDash H$ cuando $X \vDash B$.

♦ Sea X un conjunto de átomos y P un programa básico. X es llamado un answer set para P si X es mínimo entre los conjuntos de átomos cerrados bajo P.

♦ La *reducción* de una fórmula aumentada o programa relativo a un conjunto de átomos X, es definido recursivamente como sigue:

Para F elemental, $F^X = F$.

$(F \wedge G)^X = F^X \wedge G^X$.

$(F \vee G)^X = F^X \vee G^X$.

$(\neg F)^X = \bot$ si $X \vDash F^X$ y $(\neg F)^X = \top$ en otro caso.

$(H \leftarrow B)^X = H^X \leftarrow B^X$.

$P^X = \left\{ (H \leftarrow B)^X \,|H \leftarrow B \in P \right\}$.

♦ P un programa aumentado y X un conjunto de átomos. X es llamado un answer set para P si es un answer set para la reducción P^X.

Ejemplo 1. Considere el siguiente programa

P : $\quad a \leftarrow \neg\neg a.$

$\quad\quad \neg b \leftarrow c \vee b.$

Si elegimos $X = \{a\}$ entonces la reducción es

P^X : $\quad a \leftarrow \top.$

$\quad\quad \top \leftarrow c \vee b.$

■ CONCLUSIONES

♦Este trabajo es una continuación del trabajo iniciado por D. Pearce, que representa un eslabón importante entre lógica intuicionista y answer set o modelos stable.

⧫Answer set programming (ASP) es una herramienta poderosa de programación lógica que abarca cada día, más campos de aplicación.

⧫En relación al concepto de El que calla otorga, es necesario extender algún programa como dlv, smodels, el programa lparse o programar un nuevo generador de modelos, con el fin de incluir meta-reglas que permitan la utilización de este concepto de diversas maneras, durante la búsqueda de modelos. También es necesario explorar más la idea de El que calla otorga para analizar su posible formalización y su utilización en bases de datos deductivas, en las áreas de interacción hombre-máquina, en planeación y en la comunicación de agentes, por mencionar algunas.

9.4 Resumen de figuras.

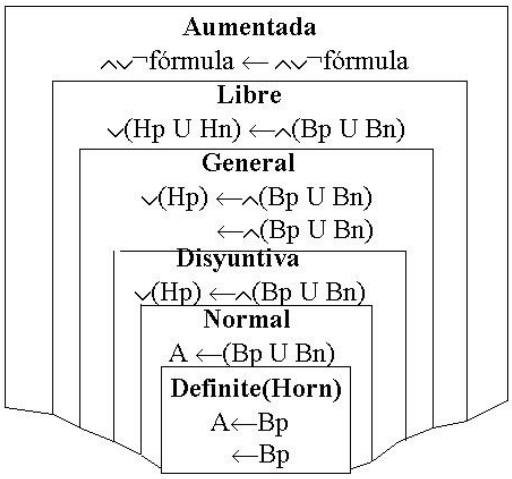

Fig. 1. Tipos de cláusulas

Figure 9.1:

Transformación	Justificación
Pl=AugFree(Pa)	$A \wedge B \leftarrow C \equiv \{A \leftarrow C, B \leftarrow C\}$ $A \leftarrow B \vee C \equiv \{A \leftarrow B, A \leftarrow C\}$ $A \vee \neg\neg B \leftarrow C \equiv A \leftarrow \neg B \wedge C$ $A \leftarrow \neg\neg B \wedge C \equiv A \vee \neg B \leftarrow C$
Pg=FreeGen(Pl)	$Pg = P' \cup \Delta s$ Donde: $P' = Pl\{\neg a / \varphi(a)\}, \forall a \in S$ $S = \{a \mid \neg a \in$ Cabeza de cláusula de $Pl\}$ $\varphi: S \rightarrow Lr$ Lr es un Conjunto de átomos reservados $\Delta s = \cup \{\varphi(a) \leftarrow \neg a, \bot \leftarrow a \wedge \varphi(a)\}, \forall a \in S$
Pd=GenDisy(Pg)	$Pd = Pg\{(\bot \leftarrow B) / (p \leftarrow B \wedge \neg p)\}$ Donde: $(\bot \leftarrow B) \in$ Restricciones de Pg $p \in Lr$

Fig. 2. Transformaciones entre clases de programas lógicos

Figure 9.2:

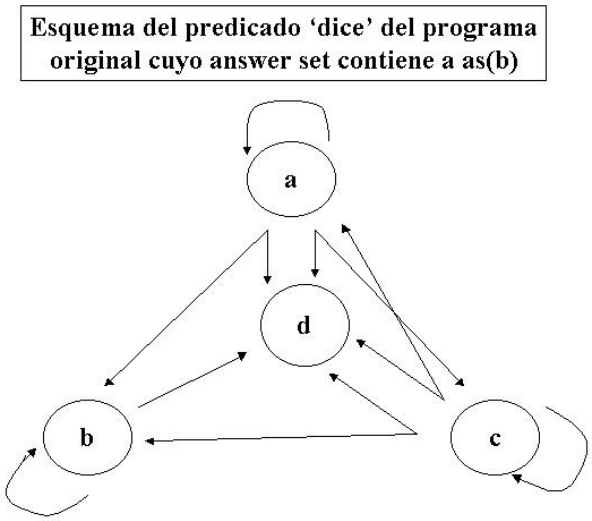
Esquema del predicado 'dice' del programa original cuyo answer set contiene a as(b)
a
d
b
c

Figure 9.3:

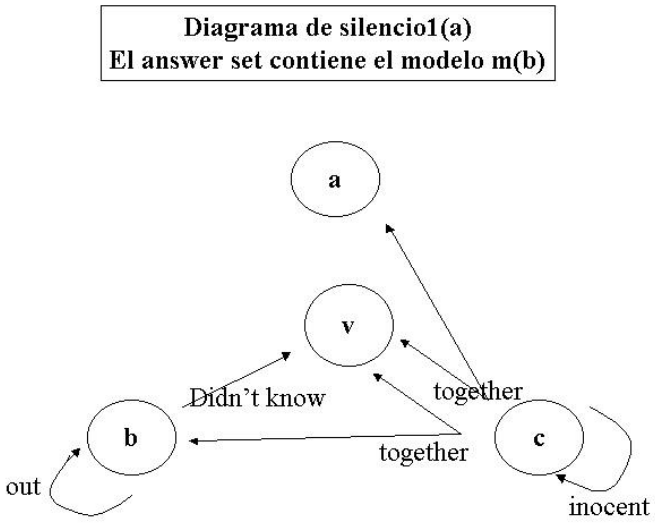
Diagrama de silencio1(a)
El answer set contiene el modelo m(b)
a
v
Didn't know
together
b
c
together
out
inocent

Figure 9.4:

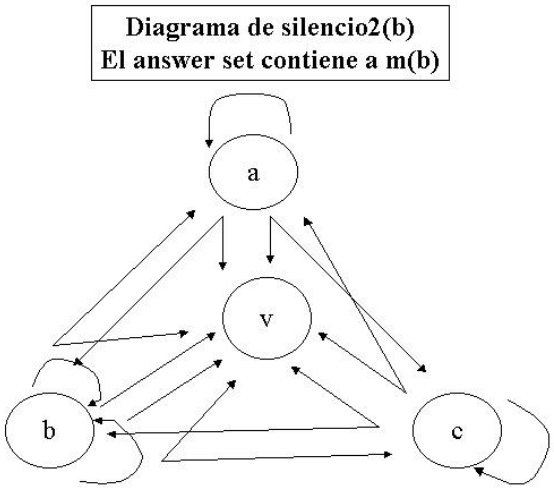

Figure 9.5:

Bibliography

[1] Mauricio Osorio Galindo, Juan Antonio Navarro y José Arrazola Ramírez. *Aplications of Intermediate logic in Answer Set Programming.* Sometido a evaluación a TLP. 2002.

[2] Juan Antonio Navarro. *Answer Set Programming Througth G_3 Logic.* ESSLLI-2002. Trento, Italy, 2002.

[3] J. Dix, M. Osorio and C. Zepeda. *A general theory of confluent rewriting systems for logic programming and its applications.* Annals of Pure and Applied logic, 108(1-3):153-188, 2001.

[4] E. Erdem and V. Lifschitz. *Fages theorem for programs with nested expressions.* in Poceedings of the 17th ICLP, pages 242-254.2001.

[5] R. Kowalski. *Is logic really dead or just sleeping.* In Proceedings of the 17th ICLP, pages 2-3. 2001.

[6] V. Lifschitz, L. R. Tang, and H. Turner. *Nested expressions in logic programs.* Annals of Mathematics and Artifiial intelligence, 25:369-389, 1999.

[7] V. Lifschitz, David Pearce, and Agustin Valverde. *Strongly equivalent logic programs.* ACM Transactions on Computational Logic, 2:526-541, 2001.

[8] Mauricio Osorio, Juan Antonio Navarro, and José Arrazola. *Equivalence in answer set programming (extended version).* in Proceeding of LOPSTR 01, LNCS 2372, Springer Verlag, pages 57-75, 2001.

[9] Mauricio Osorio Galindo, José Arrazola Ramírez, J. Antonio Navarro . *A logical approach to A-prolog.* 9th Workshop on Logic, Language, Information and Computation. pp 265-276. Rio de Janeiro Brazil. Julio 30 a Agosto 2.2002.

[10] David Pearce. *Stable inference as intuitionistic validity.* Logic Programming, 38:79-91, 1999.

[11] Dirk van Dalen. *Logic and Structure.* Springer, Berlin, second edition, 1980.

[12] M. Zakharyaschev, F. Wolter, and A. Chagrov. *Advanced modal logic.* In D. M. Gabbay, editor, to appear as a chapter in the 2nd edition of the Handbook of Philosophical Logic. Kluwer Academic Publishers, 2001.

[13] M. Gelfond and V. Lifschitz. *Logic programs with classical negation.* In Proceedings of the Seventh Int. Logic Programming Conference, Jerusalem, Israel, pages 579-597, Cambridge, Mass., 1990.ALP, MIT Press.

[14] John Fiske. *Introducción al estudio de la comunicación.* Editorial Norma 1984.

[15] Umberto Eco. *Signo.* Editorial Labor.1976.

[16] Mauricio Osorio, Juan Antonio Navarro y José Arrazola. *Applications of Intuitionistic Logic in Anwer Set Programming.* Sometido a evaluación. 2003.

[17] Tomi Janhunen. *On the Effect of Default Negation on the Expresiveness of Disjuntive Rules.*

[18] Y. Dimopoulos, J. Koehler, and B. Nebel. *Encoding planning problems in nonmonotonic logic programs.* In Proceedings of the 4th European Conference on Planning, volume 1348 of Lecture notes in Artificial Intelligence (LNCS), pages 169-181, 1997.

[19] Chitta Baral and Michael Gelfond. *Reasoning about effects of concurrent actions.* Journal of Logic Programming, 31(1-3):85-118, 1997.

[20] Marcello Balduccini and Michael Gelfond. *Diagnostic reasoning with a-prolog.* Journal of Theory and Practice of Logic Programming (TPLP), 2002. (to appear).

[21] Michel Gelfond and Vladimir Lifschitz. *Action languages.* Electronic Transactions on AI, 3(16):193-210, 1998.

[22] Chitta Baral, Michael Gelfond, and Alessandro Provetti. *Representing actions: Laws, observations and hypotheses.* Journal of Logic Programming, 31(1-3):201-243, 1997.

[23] Vladimir Lifschitz. *Two components of an action language.* Annals of Mathematics and Artificial Intelligence, 21:305-320, 1997.

[24] Patrick J. Hayes and John McCarthy. *Some philosophical problems from the standpoint of artificial intelligence.* In B. Meltzer and D. Michie, editors, Machine Intelligence 4, pages 463-502. Edinburgh University Press, 1969.

[25] Chitta Baral and Michael Gelfond. *Reasoning agents in dynamic domains.* In Workshop on Logic-Based Artificial Intelligence. Kluwer Academic Publishers, Jun 2000.

[26] Raymond Reiter. *Knowledge in Action*-Logical Foundations for Specifying and Implementing Dynamical Systems. Sep 2001.

[27] Robert A. Kowalski. *Using meta-logic to reconcile reactive with rational agents.* Meta-logic and Logic Programming, pages 227-242, Jan 1995.

[28] Michael Wooldridge. *Reasoning about Rational Agents.* MIT Press, 2000.

[29] Marcello Balduccini, Michael Gelfond, and Monica Nogueira. *A-prolog as a tool for declarative programming.* In Proceeding of the 12th International Conference on Software Engineering and Knowledge Engineering (SEKE´2000), pages 63-72, 2000.

[30] Tommi Syrjanen. *Implementation of logical grounding for logic programs with stable model semantics.* Technical Report 18, Digital Systems Laboratory, Helsinki University of Technology, 1998.

[31] David Kortenkamp, editor. 3rd NASA International workshop on Planning and Scheduling for Space, Oct 2002.

[32] Luigia Aiello and Fabio Massacci. *Verifying security protocols as planning in logic programming.* ACM Transactions on Computational Logic, 2001.

[33] Chitta Baral. *Knowledge Representation, Reasoning, and Declarative Problem Solving.* Cambridge University Press, Jan 2003.

[34] Thomas Eiter, Wolfgang Faber, Christoph Koch, Nicola Leone, and Gerald Pfeifer. *DLV – A System for Declarative Problem Solving.* In Chitta Baral and Miroslaw Truszczynski, editors, Proceedings of the 8th International Workshop on Non-Monotonic Reasoning (NMR'2000), Breckenridge, Colorado, USA, April 2000.

[35] Marcello Balduccini, Michael Gelfond, Monica Nogueira, and Richar Watson. *Planning with the usa-advisor.* In David Kortenkamp, editor, 3rd NASA International wokshop on Planning and Scheduling for Space, Oct 2002.

[36] Steve Chien and Nicola MUscettola, editors. 2nd NASA International wokshop on Planning and Scheduling for Sapce, Mar 2000.

[37] A. K. Jonsson, J. Frank, and D. E. Smith. *Bridging the gap between planning and scheduling.* Knowledge Engineering Review, 15(1):61-94, 2000.

[38] Wolfgang Faber, Nicola Leone, Cristinel Mateis, and Gerard Pfeifer. *Using database optimization techniques for nonmonotonic reasoning.* in Proceeding of the 7th International Workshop on Deductive databases and Logic programming (DDLP'99), pages 135-139, Sep 1999.

[39] Deborah East and Miroslaw Truszczynski. *dcs: An implemenattion of datalog with constraints.* In Proceedings of the 8th International workshop on Non-Monotonic Reasoning (NMR'2000). Apr 2000.

[40] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. *Adding weak constraints to disjuntive datalog.* In proceedings of the 1997 Joint Conference on Declarative Programming APPIA-GULP-PRODE'97, 1997.

[41] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. *Strong and weak constraints in disjuntive datalog.* In Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97), volume 1265 of Lecture notes in Artificial Intelligence (LNCS), pages 2-17, 1997.

[42] Alessandro Provetti and Son Cao Tran, editors. *Answer Set Programming*: Towards Efficient and Scalable Knowledge Representation and Reasoning, AAAI 2001 Spring Symposium Series, Mar 2001.

[43] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Francesco Calimeri, Tina Dell'Armi, Thomas Eiter, Georg Gottlob, Giovambattista Ianni, Giuseppe Ielpa, Christoph Koch, Simona Perri, and Axel Polleres. *The DLV System.* In Giovambattista Ianni and Sergio Flesca, editors, Proceedings of the 8th European Conference on Artificial Intelligence (JELIA), number 2424 in Lecture Notes in Computer Science, September 2002. To appear.

[44] Jurgen Dix and Frieder Stolzenburg. *A framework to incorporate nonmonotonic reasoning into constraint logic programming.* Journal of Logic Programming, 37(1-3):47-76, 1998. Special Issue on Constraint Logic Programming.

[45] Michael Gelfond. *Representing knowledge in a-prolog.* In Antonis C. Kakas and Fariba Sadri, editors, Computational logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II, volume 2408, pages 413-451. Springer Verlag, berlin. 2002.

[46] I. Niemelä, P. Simons, and T. Syrjänen. *Smodels: a system for answer set programming.* In Proceedings of the 8th International Workshop on Non-Monotonic Reasoning (cs.AI/0003073)}, Breckenridge, Colorado, USA, April 2000. (CoRR: arXiv:cs.AI/0003033)

[47] I. Niemelä and P. Simons. *Smodels - an implementation of the stable model and well-founded* semantics for normal logic programs. In Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning, volume 1265 of Lecture Notes in Artificial Intelligence, pages 420-429, Dagstuhl, Germany, July 1997

[48] J. Alfonso del C. Garcés Báez. *Introducción a la programación lógica*. Informe técnico No. 56, serie amarilla. CINVESTAV-IPN. 1987.

[49] J. Alfonso del C. Garcés Báez. *Migración de software de aplicaciones*. Tesis de licenciatura. FCFM-UAP. 1992.