Universidad Autónoma Metropolitana–Iztapalapa
División de Ciencias Básicas e Ingeniería

Lógica y Semántica Computacionales

Tesis que presenta
Habersheel Acevedo Atenco
Para obtener el grado de
Maestro en Ciencias Matemáticas

Asesor: **Dr. José Jorge Max Fernández de Castro Tapia**

Jurado Calificador:

Presidente: **Dr. Favio Ezequiel Miranda Perea**

Secretario: **Dr. Julio Ernesto Solís Daun**

Vocal: **Dr. José Jorge Max Fernández de Castro Tapia**

México, D.F. Enero 2014

Casa abierta al tiempo

UNIVERSIDAD AUTÓNOMA METROPOLITANA

Fecha : 07/01/2014
Página : 1/1

CONSTANCIA DE PRESENTACION DE EXAMEN DE GRADO

La Universidad Autónoma Metropolitana extiende la presente CONSTANCIA DE PRESENTACION DE EXAMEN DE GRADO de MAESTRO EN CIENCIAS (MATEMÁTICAS) del alumno HABERSHEEL ACEVEDO ATENCO, matrícula 2111800403, quien cumplió con los 132 créditos correspondientes a las unidades de enseñanza aprendizaje del plan de estudio. Con fecha ocho de enero del 2014 presentó la DEFENSA de su EXAMEN DE GRADO cuya denominación es:

LOGICA Y SEMANTICA COMPUTACIONALES

Cabe mencionar que la aprobación tiene un valor de 60 créditos y el programa consta de 192 créditos.

El jurado del examen ha tenido a bien otorgarle la calificación de:

aprobar

JURADO

Presidente

DR. FAVIO EZEQUIEL MIRANDA PEREA

Secretario

DR. JULIO ERNESTO SOLIS DAUN

Vocal

DR. JOSE JORGE MAX FERNANDEZ DE CASTRO TAPIA

UNIDAD IZTAPALAPA
Coordinación de Sistemas Escolares
Av. San Rafael Atlixco 186, Col. Vicentina, México, DF, CP 09340 Apdo. Postal 555-320-9000, Tels. 5804-4880 y 5804-4883 Fax: 5804-4876

Resumen

El capítulo 1 inicia revisando la lógica de primer orden, desde qué es una expresión bien construida del lenguaje lógico de primer orden sobre un vocabulario, las condiciones de satisfacibilidad respecto de un modelo de primer orden, y la forma de asociar oraciones de un lenguaje natural a un lenguaje lógico de primer orden, todo esto ejemplificado para obtener mejor claridad en la exposición. El capítulo continúa con una forma de implementar computacionalmente los mecanismos revisados previamente, y finalmente se exponen deficiencias en el uso de este tipo de aproximaciones. Aquí se introducen las ideas que se usan en el resto del trabajo para escoger un vocabulario adecuado al traducir un fragmento del lenguaje natural al lenguaje lógico.
El capítulo 2 se dedica a revisar los lenguajes lógicos de la lógica modal y la lógica temporal. Se presentan estos lenguajes en sus dos vertientes más conocidas: la proposicional y la de primer orden. Al igual que su predecesor, se introducen las definiciones de expresiones bien formadas, satisfacibilidad respecto de un modelo de Kripke y se dan ejemplos de cómo usar estos lenguajes para modelar el lenguaje natural. Se da una manera de implementar computacionalmente estos lenguajes, y por último se discute lo adecuado de una representación del lenguaje natural en estos lenguajes lógicos.
El capítulo 3 presenta el lenguaje de la teoría de tipos estándar. Se discurre sobre la manera de representar oraciones del lenguaje natural en la teoría de tipos, y se presentan los métodos más usados en la traducción de un lenguaje natural a un lenguaje lógico, mediante los árboles de análisis y las gramáticas de frases estructuradas junto con el lambda cálculo. También se provee de implementaciones computacionales de algoritmos que realicen los procedimientos discutidos.
El capítulo 4 introduce la sintaxis y semántica de la gramática de Montague, muestra el poder expresivo de este sistema, y describe las motivaciones originales de Montague para desarrollar su programa. Para esto, se junta el material expuesto en los capítulos previos. También brinda las aproximaciones de Blackburn y Bos basadas en la semántica de hoyos como alternativa al camino original de Montague.

Introducción

La semántica computacional (CS por sus siglas en inglés) es un área de muy reciente creación, que se dedica principalmente a estudiar todo lo relacionado con: maneras de calcular los significados de expresiones de un lenguaje natural dado, generar las condiciones de verdad de dichas expresiones, formas de representar la verdad y el significado de las expresiones, estudiar las relaciones de inferencia lógica de las expresiones, describir el comportamiento de las presuposiciones, y otras más; todo lo anterior, a partir de diferentes teorías semánticas y algoritmos preferentemente implementables computacionalmente. Notable es, entre otros de sus objetivos, el darle un tratamiento formal a los lenguajes naturales, y recientemente, a la pragmática de los mismos.

Aunque sus orígenes pueden rastrearse hasta la época de Frege, quien ya postulaba algunas maneras de cómo podría hacerse esto, y cuya labor es la base de la CS actual, no es sino hasta principios de este siglo que el interés en esta área creció enormemente, impulsado fuertemente por la necesidad de procesar automáticamente grandes cantidades de texto y las nuevas tecnologías en la lingüística computacional (la mayoría basadas en un desarrollo probabilista). Luego, la obra de Patrick Blackburn y Johan Bos [BB05] intitulada: Representation and Inference for Natural Language: A First Course in Computational Semantics, abre la puerta a una nueva etapa en el desarrollo de esta área, introduciendo algoritmos computacionales para revisar las teorías que se venían manejando hasta entonces simplemente como propuestas en papel, o algunas más empíricas; todas ellas teorías semánticas directas. Aunque cabe mencionar que Gazdar y Mellish en [GM89] ya habían dado algunos algoritmos para el manejo de la semántica dentro del ámbito del Procesamiento del Lenguaje Natural (NLP), y otros han desarrollado en ese mismo ámbito una suerte de semántica estadística para el NLP.

Durante años se pensó que un tratamiento formal (basado en la Teoría de Modelos matemática) para los lenguajes naturales estaba más allá de lo posible, pero a finales de los años 60 y principios de los 70 del siglo XX, Richard Montague probó lo contrario, iniciando un programa que pretendía proveer

dicha formalización y dando los primeros pasos hacia esa meta. Desafortunadamente sus avances permanecieron varios años casi en el olvido, siendo hasta los años 90 que se retoman. A principios del siglo XXI, con el auge de las computadoras, ha surgido una nueva tendencia y se está desarrollando nueva teoría y diferentes aproximaciones en esta importante y creciente área del conocimiento.

Como mencionamos antes, la base de la CS (por lo menos la línea que predomina actualmente[1]) está dada por los desarrollos de Frege, en la parte de teoría referencial del significado, Russell en la teoría de tipos subyacente al manejo de la sintaxis, Church en el cálculo lambda y Kripke en la parte de semántica de mundos posibles. Todos estos elementos fueron combinados por Montague en su obra cúspide Proper Treatment of Quantifiers (PTQ) en 1970 dando lugar a lo que ahora se conoce como Gramática de Montague (MG) y a su popular programa.

En este trabajo pretendemos dar una aproximación a la CS dando las aproximaciones originales de Montague y las más recientes por Blackburn-Bos. Para este fin, iniciamos dando un rápido repaso de los alcances de la lógica clásica para el modelado del lenguaje natural, dando especial énfasis en la satisfacibilidad de una fórmula respecto de un modelo dado. Veremos cómo esta lógica clásica es incapaz de dar un tratamiento a expresiones que contienen ciertos cuantificadores llamados generalizados. Revisaremos después con cierto detalle la lógica intensional que logra resolver algunas de las ambigüedades que la lógica clásica no nos permitía. Introduciremos el cálculo lambda que nos sirve como herramienta en el proceso de traducción del lenguaje objeto al lenguaje lógico. Y finalmente uniremos todos estos conceptos en la llamada MG donde discutiremos con cierto detalle los límites de esta teoría viendo la necesidad de tratar las presuposiciones y abriremos algunas puertas a la pragmática. Evidentemente proveeremos de algoritmos para estas tareas en el lenguaje de programación PROLOG.

Es pertinente mencionar la elección de este lenguaje de programación para los propósitos de la CS. Los algoritmos que manejan la CS pueden realizarse en cualquier lenguaje de programación (C, C^{++}, Java, Phyton, y demás) pero existe una ventaja en hacerlo en un lenguaje de programación declarativo (como PROLOG o Haskell), y es que los programas son muy directos en el sentido de que no tenemos que preocuparnos por detalles técnicos.

También conviene mencionar que el lenguaje natural que emplearemos como ejemplo para probar las teorías semánticas será el Inglés. La razón para hacerlo así es que es un lenguaje con una gramática sencilla y que

[1]David Lewis entre otros propuso en los años 70 una forma no referencial de tratar el lenguaje natural, pero esta línea no ha sido tan explorada como la de Montague.

prácticamente carece de las llamadas flexiones morfológicas, que complican el tratamiento sintáctico. Dado que la teoría referencial del significado que se usa en la CS hace uso extenso de la gramática (sintaxis) que genera el lenguaje natural en cuestión, y puesto que no queremos ahondar mucho en sintaxis (algo que nos desviaría y consumiría mucho tiempo irremediablemente), hemos optado por esta elección. Además de que la mayoría de los fenómenos que trataremos pueden ejemplificarse fácilmente en el Inglés.

A este comentario conviene ahondar un poco más. Si bien en esta obra seguiremos esta línea metodológica, a saber, aquella que se basa en una sintaxis de por medio, y que entraría en lo que Jurafsky y Martin catalogarían como *sintax-driven semantics* (algo que podríamos traducir como semántica dirigida por la sintaxis), no es la única opción explorada en la actualidad. Una gran parte de la CS se ha desarrollado, como ya dijimos, por los lingüistas computacionales, al tratar el NLP. La gran diferencia entre este tipo de aproximaciones y las provenientes a partir del programa de Montague es que en las primeras la semántica y las condiciones de verdad de las oraciones surgen por la "experiencia", mientras que en las aproximaciones basadas en Montague sí se tienen condiciones de verdad explícitas. Algo que ocurre entonces es que las condiciones de verdad en un caso vienen por el uso que se hace de las expresiones, la frecuencia con que ciertas palabras aparecen en las oraciones, y cosas similares, mientras que en el otro caso se establecen primero dichas condiciones de verdad, tratando de generalizarlas a la mayor cantidad de casos posible. Pero los métodos desarrollados en ambos tipos de aproximaciones son frecuentemente compatibles unos con otros. Esto ha ocasionado últimamente que se desarrollen sistemas en donde se apliquen ambas metodologías, ya sea paralelamente o bien dirigir el proceso con una aproximación y refinarlo con otra.

La manera en que se representa el significado también requiere especial atención. Los primeros intentos en CS se vieron limitados por este problema. Y recientemente se ha visto que existen ventajas y desventajas en representar una oración por medio de redes semánticas, fórmulas de algún lenguaje lógico, marcadores semánticos, etc. Evidentemente el título de la presente edición sugiere que estaremos usando fórmulas de lenguajes lógicos para este fin, lo que nos permitirá usar herramientas de dicho lenguaje, en nuestros distintos objetivos. Pero es muy curioso notar que algunas de las maneras en que se tratan los problemas de ambigüedades de alcance o inferencia, en la manera en que lo haremos, pueden ser compatibles con otra forma de representación, por ejemplo, redes semánticas. Representación que carece de parecido con una expresión bien formada de un lenguaje lógico. Las nuevas maneras de representación, son en sí un avance impresionante, y han permitido dar un tratamiento formal a fenómenos complicados del lenguaje. Hablar sobre ellos

merecería un amplio tratado.

Discutido lo anterior, podemos esbozar el objetivo que pretendemos: Representar (un fragmento de) el lenguaje natural desde una perspectiva formal, con el fin de imponer condiciones de verdad explícitas que nos permitan recuperar consecuencias lógicas y satisfacibilidad en el lenguaje natural, analizando lo adecuado del lenguaje lógico usado para este fin. Terminamos esta breve sección mencionando una de las características de la gente que trabaja en CS, que coincide con la cita siguiente de David Lewis:

> Semantics with no treatment of truth conditions is not semantics.
> David Lewis

Índice general

Capítulo 1

Lógica Clásica

En este capítulo revisaremos de manera fugaz qué tan adecuado es utilizar la lógica clásica para modelar el lenguaje natural. Introduciremos informalmente el lenguaje de la lógica de primer orden, revisaremos algunos ejemplos, diremos la manera en que representaremos la lógica de primer orden en Prolog y concluiremos mostrando los alcances que una aproximación de este tipo tiene y la necesidad de introducir una nueva lógica para tratar algunos de los problemas que se presentan.

1.1. Lógica de Primer Orden

La Lógica de Primer Orden (FOL por sus siglas en inglés) es una extraordinaria herramienta que permite describir las cosas de manera clara y sin ambigüedades. Principalmente por esta razón se ha convertido en el lenguaje usado por los matemáticos para establecer sus diferentes teorías. Resultaría muy conveniente entonces si pudiéramos darle al lenguaje natural un tratamiento con el lenguaje de la FOL. Además, por muchos años se ha investigado la FOL desde diversas perspectivas, por ejemplo, la *Teoría de la Prueba* y el *Razonamiento Automático*, que si pudieran usarse para el modelado del lenguaje natural nos permitiría formalizar las deducciones que uno realiza cotidianamente en el lenguaje natural y nos proveería de métodos algorítmicos, implementables en algún lenguaje de programación de forma directa.

1.1.1. Sintaxis y Semántica de la FOL

Como la FOL es muy conocida, el tratamiento que daremos será algo informal. Iniciaremos dando una Gramática de Frases Estructurada (PSG) que generará un lenguaje de la FOL sobre cierto alfabeto, que coincidirá con

un fragmento del Inglés. Esta manera es una de las más sencillas para modelar el lenguaje natural con la FOL, pues de forma muy intuitiva, las fórmulas así obtenidas, tienen un parecido con las oraciones en el Inglés que pretenden modelar. Además, usaremos una metodología similar en los capítulos 3 y 4, por lo que es conveniente introducirla desde ahora.

Sea A un alfabeto (es decir, un conjunto de símbolos de constantes, relaciones y el símbolo de igualdad $=$.[1]), con símbolos de constante denotados $cte = \{a, b, c, d, \ldots\}$, símbolos de relación denotados P_1, P_2, P_3,..., Q_1, Q_2, Q_3,..., R_1, R_2, R_3,..., donde el subídice indica la aridad de la relación. Y consideremos un conjunto numerable de variables, denotémoslas mediante $var = \{x_0, x_1, x_2, \ldots\}$. Representaremos al conjunto de fórmulas bien formadas mediante **FOR**, al conjunto de conectivos lógicos **CONN** $= \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ (los leeremos *it is not the case that...*, *and*, *or*, *if...then...* e *if and only if*, respectivamente). Los cuantificadores los leeremos $\forall$ como *every*, y $\exists$ como *some*. Entonces la siguiente PSG genera un FOL sobre el alfabeto A, que denotaremos por $L(A)$, y que corresponde a un fragmento del lenguaje Inglés. Los símbolos no terminales son alguna categoría (en negrita), los símbolos Rel_i, t, cte y var, mientras que los terminales son los conectivos lógicos, los símbolos de constante, los símbolos de variable y los símbolos de relación.

$$
\begin{aligned}
(1.1)\quad \mathbf{FOR} &\rightarrow Rel_n(t, t, \ldots, t)\\
Rel_1 &\rightarrow P_1, Q_1, R_1, \ldots\\
Rel_2 &\rightarrow P_2, Q_2, R_2 \ldots\\
Rel_3 &\rightarrow P_3, Q_3, R_3 \ldots\\
t &\rightarrow cte, var\\
cte &\rightarrow a, b, c, d, \ldots\\
var &\rightarrow x_0, x_1, x_2, \ldots\\
\mathbf{FOR} &\rightarrow \mathbf{FOR\ CONN\ FOR}\\
\mathbf{CONN} &\rightarrow \vee, \wedge, \rightarrow, \leftrightarrow\\
\mathbf{FOR} &\rightarrow \mathbf{NEG\ FOR}\\
\mathbf{NEG} &\rightarrow \neg\\
\mathbf{FOR} &\rightarrow \forall x_n \mathbf{FOR}, \exists x_n \mathbf{FOR}\\
\mathbf{FOR} &\rightarrow t = t
\end{aligned}
$$

De aquí en adelante pensaremos que A siempre contiene a los conectivos, cuantificadores y el símbolo de igualdad, así que no los mencionaremos explícitamete. En cuanto a los símbolos de función, no los consideraremos en

[1] Por cuestiones de simplicidad no consideraremos a los símbolos de función.

la presente, pues aunque resultan de gran utilidad para muchas aplicaciones, no es necesario considerarlos para los fenómenos que pretendemos tratar.
Veamos un ejemplo sencillo. Sea $A = \{alis,\ bill,\ cat,\ dina,\ woman,\ man,\ old,\ beverage,\ dancer,\ rock\text{-}star,\ dance,\ walk,\ run,\ love,\ like,\ offer,\ give\}$ un alfabeto, con símbolos de constante $cte = \{alis,\ bill,\ cat,\ dina\}$, símbolos de relación $Rel = \{man_1,\ woman_1,\ old_1,\ beverage_1,\ dancer_1,\ rock\text{-}star_1,\ dance_1,\ walk_1,\ run_1,\ love_2,\ like_2,\ drink_2,\ offer_3,\ give_3\}$, donde el subíndice nos dice la aridad de la relación. Con este alfabeto buscamos modelar algunas de las siguientes oraciones del Inglés:

Alis loves every rock-star.
It is not the case that Cat likes Bill.
Dina walks or Bill runs.
Cat offers Alis a beverage.
If there is some dancer then it is Cat.

Las cuales corresponden, intuitivamente, a las fórmulas generadas por la PSG (1.1), sobre A, respectivamente:

$\forall x(rock\text{-}star_1(x) \rightarrow love_2(alis, x))$
$\neg like_2(cat, bill)$
$walk_1(dina) \vee run_1(bill)$
$beverage_1(x) \wedge offer_3(cat, alis, x)$
$(\exists x dancer_1(x) \rightarrow x = cat)$

Veamos cómo se obtienen estas fórmulas a partir del Alfabeto A. Por ejemplo, la fórmula $\forall x(rock\text{-}star(x) \rightarrow love_2(alis, x))$, queremos ver si la expresión es generada por la PSG (1.1). Por la regla para el cuantificador universal, podemos ver que una fórmula es un cuantificador $\forall$, seguido de una variable x_n (para algún subíndice n), seguido por una fórmula. Esto es precisamente lo que tenemos en la fórmula $\forall x(rock\text{-}star_1(x) \rightarrow love_2(alis, x))$, pues aquí aparece $\forall$, seguido de la variable x, seguido de algo que esperamos que sea fórmula, es decir, la expresión $rock\text{-}star_1(x) \rightarrow love_2(alis, x)$. Ahora, para esta expresión podemos usar la regla referente a los conectivos: una fórmula puede ser del tipo: fórmula seguida por un connectivo, seguida por otra fórmula. Resta ver que lo que aparece en los extremos del conectivo sea verdaderamente una fórmula amparada por alguna de las reglas de (1.1). Usando esta vez la regla para los símbolos de relación, podemos ver que una fórmula puede ser un símbolo Rel_n seguido por (, seguido por una sucesión de símbolos del tipo $t_1, t_2, \ldots, t_n$, seguido por). En ambas expresiones a los lados del conectivo $\rightarrow$, se tiene este tipo de forma. Luego, la regla referente

a los símbolos de relación nos dice que Rel_n puede ser alguno de los símbolos de relación del alfabeto A, cuya aridad sea n. De manera que, mirando que sólo hay un símbolo del tipo t_1 en la expresión $\textit{rock-star}_1(x)$ (es decir, x), tenemos que ver que el símbolo $\textit{rock-star}_1$ sea miembro de nuestro alfabeto con esa misma aridad. Esto resulta ser así, por lo que no se presenta problema alguno, hasta el momento, con esta expresión. En cuanto a la otra expresión, $\textit{love}_2(\textit{alis}, x)$, también corresponde a un símbolo de relación de aridad 2 de nuestro alfabeto A con dos argumentos de la forma t, así que las mismas reglas que usamos para $\textit{rock-star}_1(x)$ nos garantizan que vamos por buen camino. Finalmente, la regla para los términos nos permite justificar la aparición de x como argumento de $\textit{rock-star}_1$ y la aparición de *alis* y x como argumentos de $\textit{love}_2$. De manera que las fórmulas más pequeñas están bien formadas y como las fórmulas más grandes están formadas por las primeras, y hemos usado las reglas en la construcción de cada una, podemos estar seguros que la fórmula $\forall x(\textit{rock-star}(x) \to \textit{love}_2(\textit{alis}, x))$ es una fórmula bien formada, o sea, aceptada (generada) por la PSG (1.1). Las demás fórmulas se generan de una forma análoga.

Sin embargo, hay un pequeño inconveniente, que debemos mencionar. Y es al momento de buscar el alfabeto adecuado para representar a las oraciones del lenguaje natural. Recurriremos entonces aquí, a la intuición del lector, tanto para asociar a una oración del Inglés el alfabeto adecuado, como la fórmula de $L(A)$ que le corresponde. Cierto que la gramática provee de las reglas para hacerlo, pero una misma oración del lenguaje natural podría en principio tener varias alternativas de representación. Esto lo describiremos más adelante con ejemplos precisos. Además, en el capítulo 3 formalizaremos el método para hacer esto con un lenguaje de mayor capacidad expresiva. Lo que sí notamos es que, para que las fórmulas estén intuitivamente ligadas a oraciones del lenguaje natural, debe haber una asociación entre los conectivos lógicos, los cuantificadores y los símbolos del alfabeto A con frases preconstruidas del Inglés. Podemos pensar en asociar las categorías sintácticas del Inglés de la siguiente manera:

(i) La categoría de verbos, que denotaremos **V**, la asociamos a algunos de los símbolos de predicado de A.

(ii) La categoría de nombres propios, que denotaremos **PN**, la asociamos con los símbolos de constante de A.

(iii) La categoría de determinadores (en castellano más conocida por artículos), que denotaremos **DET**, la asociamos con los cuantificadores $\forall$ y $\exists$.

(iv) La categoría de conjunciones, que denotaremos por **CNJ**, la asociamos

con los operadores lógicos $\vee$ y $\wedge$.

(v) La frase "*it is not the case that...*" la asociaremos con el símbolo $\neg$.

(vi) La frase "*if...then...*" la asociaremos con el conectivo $\rightarrow$.

(vii) La frase "*...if and only if...*" la asociamos al conectivo $\leftrightarrow$.

(viii) La categoría de pronombres, que denotaremos **PRO**, la asociamos con las variables $x_0, x_1, x_2, \ldots$.

(ix) La categoría de sustantivos comunes, que denotamos **Noun**, la asociamos con algunos símbolos de predicado de A.

Ya que los símbolos de predicado de A tienen diferente aridad n, debemos decir qué verbos tendran aridad uno, cuáles dos, cuáles tres, etc. La manera de hacerlo es como sigue:

- ♣ Los verbos intransitivos y los sustantivos comunes serán asociados a símbolos de predicado de aridad $n = 1$.
- ♣ Los verbos transitivos serán asociados a símbolos de predicado de aridad $n = 2$.
- ♣ Los verbos ditransitivos serán asociados a símbolos de predicado de aridad $n = 3$.

Supondremos que sólo tenemos símbolos de relación de aridad uno, dos o a lo más tres. Esta misma asociación la debemos hacer con los artículos.

- ♢ El determinador "some..." se asocia a $\exists$.
- ♢ El determinador "every..." se asocia a $\forall$.

y con las conjunciones.

- ♡ La conjunción "or" se asocia con $\vee$.
- ♡ La conjunción "and" se asocia con $\wedge$.

Con estas convenciones debería ser clara la manera en que llegamos a dar las fórmulas de $L(A)$ para las oraciones de arriba. Veamos cómo se obtiene la primera de las oraciones ejemplificadas arriba. A saber, *Alis loves every rock star*. Como la oración hace referencia al nombre propio *Alis*, por la asociación mencionada, debemos de usar en su representación en FOL al símbolo de constante *alis*, luego vemos que la oración usa al verbo transitivo *love* así que usamos al 2-símbolo de relación $love_2$, el determinador *every* se asocia con un cuantificador $\forall$, y por último la *rock star* que es el objeto de la acción de *Alis loves...*, como este sustantivo representa a muchos individuos (todos

aquellos que sean estrellas del rock), podemos usar un pronombre, por las reglas dictadas antes eso corresponde a una variable, digamos x, que precisamente es la que yace bajo el alcance del cuantificador $\forall$, y que debe estar en la relación $\textit{rock-star}_1$. Llegamos así a la fórmula de la FOL que representa a la oración: $\forall x(\textit{rock-star}_1(x) \to \textit{love}_2(alis, x))$. En este ejemplo hemos dejado explícitamente los subíndices de los símbolos de relación, algo que no haremos en los ejemplos subsecuentes. Esto se hace para mayor legibilidad en la lectura. Trataremos de omitirlos siempre que eso no nos lleve a alguna confusión. El resto de las traducciones a la FOL de las oraciones dadas se obtienen similarmente.

Continuaremos ahora con la terminología que usaremos para la semántica. Sean A un alfabeto, $L(A)$ el lenguaje de la FOL sobre dicho alfabeto, $\Sigma \in L$ un conjunto de fórmulas y M un modelo de $L(A)$ (es decir, una pareja (D, F), donde D es un conjunto llamado dominio de interpretación y F es una función de interpretación tal que a cada símbolo σ de A, $F(\sigma) \in D^n$ con un n adecuado). Llamaremos a M una *situación* y a cada $s \in \Sigma$ una *descripción*. Si s es tal que $M \not\models s$ decimos que la descripción es *inconsistente* con el contexto M o que el contexto M no permite a la descripción s. Si por el contrario, s es tal que $M \models s$ decimos que la descripción es *consistente* con M.

La razón para considerar a las fórmulas de $L(A)$ como descripciones es que la semántica de la FOL nos permite decir si una fórmula ϕ es verdadera en un modelo M, muy parecido a como funcionan las descripciones en el lenguaje natural. Los modelos por su parte, son parecidos a situaciones en los lenguajes naturales, pues si bien una fórmula puede no ser verdadera en un modelo M, puede serlo en otro modelo distinto M'. Todo esto, soportado por la semántica de la FOL. Además, la semántica usual de la FOL es muy similar a la intuición que sus equivalentes en el lenguaje natural tienen. Por ejemplo, se piensa normalmente que una oración como: *Alis walks and Bill runs*, es verdadera en una situación, si en esa situación tanto Alis está efectivamente caminando, como Bill está corriendo. Pero esto es precisamente lo que la semántica de la fórmula $\phi \equiv \textit{walk}(alis) \wedge \textit{run}(bill)$ nos dice, pues $M \models \phi$ si y sólo si $M \models \textit{walk}(alis)$ y $M \models \textit{run}(bill)$, lo cual sucede si y sólo si en el modelo M se tiene que $F(alis) \in F(walk)$ y $F(bill) \in F(run)$, o sea, si Alis camina y además Bill corre.

Ahora veamos para que nos sirve esto. Por ejemplo, supongamos una situación M, donde Cat está enferma y planea hacer un viaje por lo que solicita su pasaporte en la dependencia correspondiente, entonces es natural que la persona encargada le pregunte a Cat algo como: *Do you currently have any illness?*. Ésto lo podría conocer calculando la oración $s = \textit{Cat is ill}$, en esta situación M.

Supongamos que tenemos ciertos enunciados Σ y que tenemos una situación real o hipotética M. Si nos interesa cambiar la situación M a otra situación M' en base a la veracidad de los enunciados de Σ, debemos ser capaces de calcular dicha veracidad respecto de M. La semántica de la FOL nos permite determinar el valor de verdad de una oración respecto de un modelo de una manera directa. En el ejemplo anterior, si $M \not\models s$ (lo que significaría que Cat no está enferma), la encargada podría pasar a otra situación M' para continuar con el trámite, quizá después preguntando algo como *Have you ever done anything illegal?*

Este tipo de ideas son muy utilizadas en tareas como el Razonamiento Automático en el área de Inteligencia Artificial (AI). Codificando la información en un modelo, la máquina trata de tomar decisiones, basadas en la veracidad de "enunciados importantes", que le permiten analizar su situación, en relación con un objetivo dado. La ventaja que allí se tiene es que esos enunciados son propuestos por el programador quien se encarga de darle información tratable a la computadora. En nuestro caso, dar un tratamiento adecuado a la semántica del lenguaje natural nos lleva a tener que analizar todo tipo de enunciados, algunos de los cuales son conflictivos incluso para los más experimentados hablantes de su lengua nativa. Inclusive en el sencillo lenguaje que recién se introdujo, podría presentarse el caso. Consideremos por ejemplo:

(1.2) *Every woman loves some rock-star.*

A esta oración le podríamos asociar dos fórmulas de $L(A)$, a saber,

(1.3) $\forall x(woman(x) \rightarrow \exists y(rock\text{-}star(y) \wedge love(x, y)))$

o bien

(1.4) $\exists y(rock\text{-}star(y) \wedge (\forall x woman(x) \rightarrow love(x, y)))$

cuyos significados, si bien están relacionados de alguna manera, no son totalmente iguales. Lo mismo ocurre con la oración:

(1.5) *Every old man and woman likes some beverage.*

a la que se le puede asociar la fórmula

(1.6) $\forall x \forall y(man(x) \wedge old(x) \wedge woman(y) \rightarrow \exists z(beverage(z) \wedge likes(x,z) \wedge likes(y,x)))$

o bien la fórmula

(1.7) $\forall x \forall y(man(x) \wedge old(x) \wedge woman(y) \wedge old(y) \rightarrow \exists z(beverage(z) \wedge likes(x,z) \wedge likes(y,x)))$

donde, de nueva cuenta, los significados no son precisamente los mismos. Este fenómeno ocasionó que se pensara durante muchos años que un tratamiento con la FOL para el lenguaje natural era imposible, pues al parecer existen muchas ambigüedades en este último. Dado que la FOL no posee ambigüedades, era razonable llegar a esta conclusión. Todavía en el área de NLP existen la creencia de que tratar de usar deducciones de la FOL para modelar las deducciones del lenguaje natural es una aproximación con escaso éxito. No obstante, la principal razón para llegar a tales conclusiones yace en la manera de concebir los modelos (situaciones). Dada la naturaleza de los modelos de la FOL, es posible incluir una cantidad considerable de detalle en ellos, lo que le permite mucha flexibilidad en lo que se refiere al modelado de diversas situaciones. Regresaremos a esta discusión cuando veamos los alcances de la FOL.

Revisemos un ejemplo de la semántica. Sean A el alfabeto que hemos estado trabajando, y consideremos el modelo $M = (D, F)$, donde $D = \{a, b, c, d, e, g\}$ y F está definida como sigue:

$F(alis) = a$

$F(bill) = b$

$F(cat) = c$

$F(dina) = d$

$F(woman) = \{a, c, d\}$

$F(man) = \{b\}$

$F(old) = \{d, e\}$

$F(beverage) = \{g\}$

$F(dancer) = \{c\}$

$F(rock\text{-}star) = \{b, e\}$

$F(dance) = \emptyset$

$F(walk) = \{a, d\}$

$F(run) = \{b\}$

$F(love) = \{(b, a), (c, b)\}$

$F(like) = \{(a, b), (a, e), (c, e), (d, e)\}$

$F(drink) = \emptyset$

$F(offer) = \{(a, b, g)\}$

$F(give) = \emptyset$

Con este modelo, las fórmulas

$\forall x \exists y((woman(x) \wedge rock\text{-}star(y)) \rightarrow like(x,y) \vee \exists z(man(z) \wedge offer(alis,z,beverage)))$,
$\forall \alpha(woman(\alpha) \rightarrow \neg dance(\alpha))$,
$\forall x(man(x) \wedge dance(x) \rightarrow drink(x, beverage))$ y
$\exists x(old(x) \wedge walk(x))$

son consistentes (satisfacibles en M). Mientras que las fórmulas

$\exists \alpha(woman(\alpha) \wedge dance(\alpha))$,
$\exists x(beverage(x) \wedge give(alis, bill, x))$,
$\exists z(dancer(z) \wedge woman(z) \rightarrow z = d)$ y
$\forall x \exists y(love(y, x))$

no lo son.

1.2. Lógica de Primer Orden en PROLOG

Llegó la hora de revisar la manera en que implementaremos la FOL en Prolog. Empezamos traduciendo los símbolos del lenguaje para luego construir las representaciones en Prolog de las fórmulas y los modelos. Despúes veremos como traducir la semántica de la FOL en Prolog. Y finalmente daremos un programa que permita evaluar una fórmula dada en un modelo dado.

1.2.1. Traduciendo a Prolog los Términos, Fórmulas y Modelos

Ahora daremos la manera de implementar las fórmulas, variables y modelos para un lenguaje $L(A)$ en Prolog. Regularmente nos referiremos a nuestro ejemplo generado por (1.1) y el alfabeto A allí pesentado, con el fin de ver lo que ocurre de manera más explícita. Lo primero es representar el alfabeto A. Esto lo logramos así: usaremos un átomo de Prolog por cada símolo de constante en A. En nuestro ejemplo, los átomos correspondientes a los símbolos de constante podrían ser:

```
alis,bill,cat,dina.
```

Los símbolos de relación (a excepción del símbolo de igualdad) los representaremos por términos complejos de Prolog, cuya aridad corresponda con la aridad del símbolo de relación que representa, y cuyo functor sea lo más parecido posible al símbolo de relación. Por ejemplo, los símbolos de relación

de nuestro ejemplo quedarían:

```
woman(X), man(Y), old(Z), beverage(U), dancer(W), rock_star(X),
dance(X), walk(X), run(X), love(X,Y), like(X,Y), drink(V,W),
offer(X,Y,Z), give(X,Y,Z).
```

En realidad, lo importante es la manera en que éstos se representen respecto de un modelo, pues ultimadamente, serán los elementos que influirán en el valor de verdad de una fórmula. En cuanto al símbolo de igualdad, usaremos el término complejo de Prolog:

```
eq(X,Y).
```

Utilizaremos a las variables de Prolog como las análogas de las variables de la FOL. Y los predicados de Prolog:

`all/2`, `some/2`, `and/2`, `or/2`, `imp/2`, `iff/2`, `not/1`

para capturar las fórmulas asociadas a los símbolos $\forall, \exists, \wedge, \vee, \rightarrow, \leftrightarrow, \neg$. De modo que representaremos a las fórmulas $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \rightarrow \psi$, $\phi \leftrightarrow \psi$, $\forall x\phi$ y $\exists x\phi$ por medio de:

```
not(Phi), and(Phi,Psi), or(Phi,Psi), imp(Phi,Psi), iff(Phi,Psi),
all(X,Phi), some(X,Phi),
```

respectivamente. Algo que no hemos discutido aun es el papel que desempeñan las asignaciones en la evaluación de las fórmulas respecto de un modelo. Sin embargo, es necesario tener una forma de representarlas en Prolog. Así que recordemos la manera en que se define la satisfacción de una fórmula ϕ respecto de un modelo M y una asignación g, para las fórmulas donde son necesarias las asignaciones.

1.2.1 **Definición**.
Sean $M = (D, F)$ un modelo, g una asignación y $\phi \in$ **FOR** una fórmula.

(i) Si $\phi \equiv \exists x\psi$ entonces $M, g \models \psi$ si y sólo si hay una *x-variante* g' de g tal que $M, g' \models \psi$.

(ii) Si $\phi \equiv \forall x\psi$ entonces $M, g \models \psi$ si y sólo si toda *x-variante* g' de g es tal que $M, g' \models \psi$.

Donde g' es una x-variante de g si es una asignación tal que para toda variable $y \neq x$ se tiene que $g(x) = g(y)$.

La definición nos da la idea de como resolver nuestro dilema. Lo que haremos para representar a las asignaciones en Prolog será utilizar listas que enumeren los valores que toma cada variable bajo la asignación. Esto representa un problema si tomamos en cuenta que estamos trabajando bajo el supuesto de que disponemos de un número infinito numerable de variables. Sin embargo, sólo tenderemos en cuenta las variables libres al momento de evaluar una fórmula respecto de un modelo, esto lo haremos así debido a que

las proposiciones siguientes nos garantizan obtener el mismo resultado que hacerlo directamente de la definición.

1.2.2 **Proposición**.
Sean ϕ un enunciado de $L(A)$, g, g' asignaciones y M un modelo. Entonces $M, g \models \phi$ si y sólo si $M, g' \models \phi$.

1.2.3 **Proposición**.
Sean ϕ una fórmula de $L(A)$, g, g' asignaciones que difieren solamente en variables que no ocurren en ϕ y M un modelo. Entonces $M, g \models \phi$ si y sólo si $M, g' \models \phi$.

Lo que nos dice el primer resultado es que si la fórmula no tiene variables libres (es un *enunciado*) entonces no importa la asignación que usemos para evaluar. Mientras que el segundo nos indica que las únicas variables que importan al momento de hacer la evaluación son aquellas que ocurren libres en la fórmula a evaluar. De manera que usando esto a nuestro favor, la lista que usaremos para representar en Prolog a las asignaciones, será una lista vacía en el caso de los enunciados, y una que contenga un término de Prolog de la forma:

```
g(X,Value)
```

donde, `Value` es el valor que toma la variable libre x, haremos esto por cada variable libre x que aparezca en la fórmula a evaluar. Finalmente, los modelos $M = (D, F)$ los representaremos mediante el predicado de Prolog:

```
model(D,F)
```

donde, D lo representaremos por una lista que contenga a los átomos que representan la interpretación de los símbolos de constante. Por ejemplo, podríamos representarlos por:

```
[a,b,c,d].
```

o bien podemos usar átomos indexados como:

```
[a1, a2, a3, a4].
```

La función de interpretación F será una lista también, la cual contiene la asociación que cada símbolo del alfabeto tiene con su interpretación, indicada por el término complejo:

```
f(Arity,Atom,[F(Symbol)])
```

en donde el primer argumento es la aridad del símbolo (consideraremos que las constantes tienen aridad 0), el segundo argumento es el átomo que representa al símbolo, y el tercer argumento es una lista que contiene a los elementos que caen bajo la interpretación del símbolo. Esta lista, que corresponde al tercer argumento, no es otra cosa que la representación en Prolog

del conjunto D^n, así que a veces será el conjunto (lista) vacío(a), a veces un solo elemento, en ocasiones un conjunto de elementos de D, otras veces parejas ordenadas de elementos de D, y así sucesivamente. Por ejemplo, la representación del modelo que dimos anteriormente sería:

```
model([a,b,c,d,e,g],
      [f(0,alis,a),
      f(0,bill,b),
      f(0,cat,c),
      f(0,dina,d),
      f(1,woman,[a,c,d]),
      f(1,man,[b]),
      f(1,old,[d,e]),
      f(1,beverage,[g]),
      f(1,dancer,[c]),
      f(1,rock_star,[b,e]),
      f(1,dance,[]),
      f(1,walk,[a,d]),
      f(1,run,[b]),
      f(2,love,[(b,a),(c,b)]),
      f(2,like,[(a,b),(a,e),(c,e),(d,e)]),
      f(2,drink,[]),
      f(2,offer,[(a,b,g)]),
      f(2,give,[])
      ]
   ).
```

Y las fórmulas

$\forall x \exists y((woman(x) \wedge rock\text{-}star(y)) \rightarrow like(x,y) \vee \exists z(man(z) \wedge offer(alis,z,beverage)))$,
$\forall \alpha(woman(\alpha) \rightarrow \neg dance(\alpha))$,
$\forall x(man(x) \wedge dance(x) \rightarrow drink(x, beverage))$ y
$\exists x(old(x) \wedge walk(x))$,

se representan respectivamente como

```
all(X,some(Y,
   or(imp(and(woman(X),rock_star(Y)),like(X,Y)),
   some(Z,and(man(Z),offer(a,Z,g))))))),
all(Alpha,imp(woman(Alpha),not(dance(Alpha)))),
all(X,imp(and(man(X),dance(X)),drink(X,g))),
some(X,and(old(X),man(X))).
```

1.2.2. Semántica de la Lógica de Primer Orden en Prolog

Ahora procedemos a ver la manera de representar la semántica de la FOL en el lenguaje de programación Prolog. Para esto, utilizaremos el predicado `sat(Formula,Model,Assignment,Polarity)`, que será el encargado de representar la satisfacción en Prolog, dadas la representación en Prolog de una fórmula (lo que corresponde al primer argumento del predicado `sat/4`), la representación en Prolog de un modelo (el segundo argumento de `sat/4`), la representación en Prolog de una asignación (el tercer argumento de `sat/4`), y la polaridad. Este último argumento es la bandera que nos indica si estamos tratando de evaluar la fórmula como verdadera (positiva) o falsa (negativa) en el modelo. Revisemos como funciona esto, primero veamos los casos sencillos: los conectivos lógicos. Las cláusulas para la negación son

```
sat(not(Formula),Model,G,pos):-
      sat(Formula,Model,G,neg).
sat(not(Formula),Model,G,neg):-
      sat(Formula,Model,G,pos).
```

Como se puede ver, el predicado refleja las condiciones de satisfacción para la negación usuales. Lo único que estamos pidiendo para que una fórmula negada `not(Formula)` se satisfaga en un modelo `Model`, es pedir que en el mismo modelo no se satisfaga la formula sin la negación. Esto es lo que declara la primera cláusula que trata con la polaridad positiva. Ahora para la polaridad negativa es justamente lo opuesto, es decir, la fórmula `not(Formula)` no se satisface en el modelo `Model` si la fórmula sin la negación se satisface en el modelo. Y esto es precisamente lo que transmite la segunda cláusula.

Ahora, las cláusulas para la conjunción serán

```
sat(and(Formula1,Formula2),Model,G,pos):-
      sat(Formula1,Model,G,pos),
      sat(Formula2,Model,G,pos).
sat(and(Formula1,Formula2),Model,G,neg):-
      sat(Formula1,Model,G,neg);
      sat(Formula2,Model,G,neg).
```

De nuevo miramos que lo unico que las cláusulas hacen es imitar las condiciones de satisfacción de la conjunción usuales. Esto es, que ambos conyuntos sean satisfacibles en el modelo para la polaridad positiva (primera cláusula, nótese el uso de operador `,` que es el operador interno de Prolog para la conjunción). Y que al menos uno de ellos no sea satisfacible para la polaridad negativa (segunda cláusula, nótese el uso del operador interno de Prolog para la disyunción `;`). Con estos dos conectivos representados, podemos

continuar en dos formas distintas: ya sea continuar imponiendo directamente las condiciones de satisfacibilidad usuales de los conectivos como hicimos para la negación y la conjunción, o bien, empleándolos como conjunto de conectivos básico, y usando las relaciones que hay entre ellos (por ejemplo, $p \vee q \equiv \neg(p \wedge q)$), representar al resto de los conectivos usando sólo las representaciones en Prolog que ya tenemos para la negación y la conjunción. Nosotros usaremos el primer camino. Así, las cláusulas para representar a la disyunción, implicación y la doble implicación quedan

```
sat(or(Formula1,Formula2),Model,G,pos):-
      sat(Formula1,Model,G,pos);
      sat(Formula2,Model,G,pos).
sat(or(Formula1,Formula2),Model,G,neg):-
      sat(Formula1,Model,G,neg),
      sat(Formula2,Model,G,neg).

sat(imp(Formula1,Formula2),Model,G,pos):-
      (
      sat(Formula1,Model,G,pos),
      sat(Formula2,Model,G,pos)
      );
      sat(Formula1,Model,G,neg).
sat(imp(Formula1,Formula2),Model,G,neg):-
      sat(Formula1,Model,G,pos),
      sat(Formula2,Model,G,neg).

sat(iff(Formula1,Formula2),Model,G,pos):-
      (
      sat(Formula1,Model,G,pos),
      sat(Formula2,Model,G,pos)
      );
      (
      sat(Formula1,Model,G,neg),
      sat(Formula2,Model,G,neg)
      ).
sat(iff(Formula1,Formula2),Model,G,neg):-
      (
      sat(Formula1,Model,G,pos),
      sat(Formula2,Model,G,neg)
      );
      (
      sat(Formula1,Model,G,neg),
```

```
    sat(Formula2,Model,G,pos)
    ).
```

El siguiente paso es dar las cláusulas para lidiar con fórmulas cuantificadas, para el cuantificador existencial tenemos entonces

```
sat(some(X,Formula),model(D,F),G,pos):-
    member(V,D),
    sat(Formula,model(D,F),[g(X,V)|G],pos).
```

Lo que las cláusulas nos dicen es que una fórmula existencialmente cuantificada es satisfacible en un modelo con dominio D si hay un valor V (esto es lo que se logra al usar el predicado `member(V,D)`) en D, tal que al modificar la asignación G en el valor X, sustituyéndolo por V (o sea probando diferentes x-variantes de G), la fórmula matriz es satisfacible en el modelo. Básicamente la idea es ir probando las posibles x-variantes de G intentando con cada elemento del dominio D para ver si la fórmula matriz es satisfacible con alguno de esos valores. Ahora para la polaridad negativa tenemos

```
sat(some(X,Formula),model(D,F),G,neg):-
    setof(V,
       (
          member(V,D),
          sat(Formula,model(D,F),[g(X,V)|G],neg)
       ),
       Dom),
    setof(V,member(V,D),Dom).
```

Las cláusulas expresan lo que la definición de satisfacibilidad nos dice: una fórmula existencialmente cuantificada no es satisfacible si, bajo cada x-variante, la formula matriz no es satisfacible. En este caso vamos probando cada x-variante de G y colectando los valores que no la hacen satisfacible (esto lo logramos con `setof/3`), si al final el conjunto de esos valores (Dom) es igual a D, entonces ningún valor satisface a la fórmula matriz y por lo tanto, la fórmula existencialmente cuantificada no es satisfacible. Para la fórmula universalmente cuantificada volvemos a tener dos opciones: ya sea imponer condiciones directamente, imitando la definición de satisfacibilidad, como hicimos con la fórmula existencialmente cuantificada, o bien, usar la relación que hay entre los cuantificadores existencial y universal: $\forall x\phi \equiv \neg\exists x\neg\phi$, y usar la representación de Prolog que acabamos de dar para la fórmula existencialmente cuantificada, para construir la representación en Prolog de la fórmula universalmente cuantificada. Usaremos la primera forma mencionada

```
sat(all(X,Formula),model(D,F),G,pos):-
    setof(V,
```

```
          (
          member(V,D),
          sat(Formula,model(D,F),[g(X,V)|G],pos)
          ),
          Dom),
      setof(V,member(V,D),Dom).
sat(all(X,Formula),model(D,F),G,neg):-
      member(V,D),
      sat(Formula,model(D,F),[g(X,V)|G],neg).
```

Sólo nos resta representar la satisfacibilidad de las fórmulas atómicas. En esta parte se presenta una dificultad: diferenciar las variables de las constantes. Debido a que la interpretación de las constantes se lleva acabo mediante la función de interpretación, y la de las variables mediante la función de asignación, debemos ser capaces de saber qué clase de término es el que aparece como argumento del símbolo de relación. Afortunadamente, disponemos del predicado `compose/3`, que nos permite descomponer un término complejo en su functor (símbolo) y argumentos (términos). Así que éste hará parte del trabajo. Una vez que tenemos el tipo de símbolo y el(los) argumento(s), podemos proceder a buscar su interpretación, de acuerdo al tipo de término (constante o variable) que encontremos; esto lo realizarán, primero el predicado `termSymbol/4`, el cual interpreta un término con el mecanismo adecuado para éste, es decir, a una constante empleando la función de interpretación `f(Arity,Symbol,Value)`, y a una variable usando la función de asignación `g(X,Value)`. Y después deberemos de ver, que el valor que el término obtiene de esta manera, caiga dentro de los valores del modelo. Vayamos por partes, daremos primero las cláusulas para las relaciones unarias

```
sat(Formula,model(D,F),G,pos):-
      compose(Formula,Symbol,[Argument]),
      termSymbol(Argument,model(D,F),G,Value),
      member(f(1,Symbol,Values),F),
      member(Value,Values).

sat(Formula,model(D,F),G,neg):-
      compose(Formula,Symbol,[Argument]),
      termSymbol(Argument,model(D,F),G,Value),
      member(f(1,Symbol,Values),F),
      \+member(Value,Values).
```

Las cláusulas presentan la discusión anterior. Como se puede ver la mayor carga la realiza el predicado que se encarga de interpretar los argumentos adecuadamente (ya sean constantes o variables). Ahora, para las relaciones

binarias, como las cláusulas que siguen muestran, tenemos que usar dos veces el predicado `termsymbol/4`

```
sat(Formula, model(D,F),G,pos):-
      compose(Formula,Symbol,[Argument1,Argument2]),
      termSymbol(Argument1,model(D,F),G,Value1),
      termSymbol(Argument2,model(D,F),G,Value2),
      member(f(2,Symbol,Values),F),
      member((Value1,Value2),Values).
sat(Formula,model(D,F),G,neg):-
      compose(Argument,model(D,F),[Argument]),
      termSymbol(Argument1,model(D,F),G,Value1),
      termSymbol(Argument2,model(D,F),G,Value2),
      member(f(2,Symbol,Values),F),
      \+member((Value1,Value2),Values).
```

En el caso de los predicados de orden tres, será necesario usar ese mismo predicado tres veces, una vez por cada argumento

```
sat(Formula,model(D,F),G,pos):-
      compose(Formula,Symbol,[Argument1,Argument2,Argument3]),
      termSymbol(Argument1,model(D,F),G,Value1),
      termSymbol(Argument2,model(D,F),G,Value2),
      termSymbol(Argument3,model(D,F),G,Value3),
      member(f(3,Symbol,Values),F),
      member((Value1,Value2,Value3),Values).
sat(Formula,model(D,F),G,neg):-
      compose(Formula,Symbol,[Argument1,Argument2,Argument3]),
      termSymbol(Argument1,model(D,F),G,Value1),
      termSymbol(Argument2,model(D,F),G,Value2),
      termSymbol(Argument3,model(D,F),G,Value3),
      member(f(3,Symbol,Values),F),
      \+member((Value1,Value2,Value3),Values).
```

Con este último ejemplo, podemos darnos una idea de cómo manejar al símbolo de relación particular $=$. Basta ligar los términos a símbolos y preguntarnos si unifican, usando el predicado interno de Prolog

```
sat(eq(X,Y),Model,G,pos):-
      termSymbol(X,Model,G,Value1),
      termSymbol(Y,Model,G,Value2),
      Value1=Value2.
sat(eq(X,Y),Model,G,neg):-
      termSymbol(X,Model,G,Value1),
      termSymbol(Y,Model,G,Value2),
      \+Value1=Value2.
```

1.2.3. Evaluando Fórmulas en Modelos con Prolog

En esta parte implementaremos un predicado que nos permita evaluar, una representación de Prolog de una fórmula, en una representación de Prolog de un modelo. Esto es muy sencillo de lograr en realidad. Ya que en la sección anterior desarrollamos una manera de representar la satisfacibilidad de una fórmula dada en un Modelo y respecto de una asignación dados. Pero ya que la forma de trabajar de Prolog es verificar la satisfacibilidad de sus cláusulas, lo único que debemos hacer es usar el predicado `sat/4` en algún modelo ya dado, con una fórmula que pretendamos evaluar y, si ésta tiene variables libres, dar una asignación a dichas variables libres, por medio de una lista, en caso de que no las tenga, simplemente se usa la lista vacía. Y por supuesto pedirle alguna polaridad, ya sea `pos`, si queremos ver que la fórmula sea satisfacible o bien `neg`, si queremos ver que no sea satisfacible.

Veamos un ejemplo de una consulta. Supongamos que queremos evaluar la fórmula $\exists x(old(x) \wedge man(x))$ que tiene representación en Prolog `some(X,and(old(X),man(X)))`, en el modelo que hemos estado manejando. Entonces lo único que tenemos que hacer es hacer la consulta

```
?- sat(some(X,and(old(X),man(X))),M,[],pos).
```

donde `M` es el modelo tecleado en su representación en Prolog, la lista vacía corresponde a que no hay variables libres y la polaridad `pos` a que deseamos ver si la fórmula es satisfacible. Entonces Prolog dirá

```
false
```

Ahora, estas implementaciones tienen un pequeño problema técnico. Por la forma en que trabaja Prolog, sin discernir si los argumentos con los que se realiza una consulta están bien instanciados, podemos pasarle una expresión mal construida y aun así Prolog tratará de ver si se satisfacen las cláusulas. Esto es un detalle que se arregla de forma sencilla, pero que no daremos explícitamente en el texto. El código final ya incluye el manejo de dichas dificultades. Baste decir que lo que se hace, es verificar que la expresión sea una expresión bien construida, antes de intentar la evaluación.

1.3. Límites de la Lógica de Primer Orden

Evaluar los límites expresivos de la FOL no es sencillo, sin embargo, podemos darnos una buena idea de lo que sucede.

Muy a pesar de las muchas objeciones que se imputan sobre la FOL, la mayoría de ellas son equivocadas. A menudo se dice que la FOL es incapaz

de representar el tiempo (en nuestro caso las conjugaciones verbales), o los estados epistémicos, o los fenómenos modales, u otras cosas. Pero debemos recordar que los símbolos de relación son extremadamente generales, siendo capaces de representar casi cualquier cosa, incluido el tiempo. Si se le da el tratamiento adecuado al modelo, se pueden representar muchas de las cosas que inicialmente se considera imposible lograr. En su libro [BB05], Blackburn-Bos logran destacar la flexibilidad de la FOL para representar intervalos de tiempo discretos que con una ligera y sencilla modificación podrían convertirse en intervalos continuos de tiempo. El modelo es como sigue:
$A = \{mia, monday, tuesday, wednesday, person, day, precede, happy\}$, $M = (D, F)$, $D = \{a, b, c, d\}$, $F(mia) = a$, $F(monday) = b$, $F(tuesday) = c$, $F(wednesday) = d$, $F(person) = \{a\}$, $F(day) = \{a, b, c\}$, $F(precede) = \{(b, c), (c, d), (b, d)\}$, $F(happy) = \{(a, b), (a, d)\}$. Modelo que pretende (y consigue) modelar una situación donde Mia esté feliz en los días Lunes y Miércoles pero no en los Martes, donde el Lunes (como esperaríamos) precede al Martes y al Miércoles, mientras que el Martes sucede al Lunes y precede al Miércoles, el Miércoles sucediendo a ambos, Lunes y Martes. En resumen, modela una situación temporal, del estado emocional de Mia. Con una ligera modificación podemos introducir la hora del día (convirtiendo el modelo discreto original en algo continuo). $F(happy) = \{(a, x, y) : x = b \vee x = d$, $10{:}00 \leq y \leq 17{:}00$ $\}$. Lo que nos diría que Mia está feliz sólo en ciertas horas de los días Lunes y Miércoles (presumiblemente en la noche algo la inquieta). [BB05] también deja en claro que no sólo este tipo de tratamientos son posibles sino que además puede dársele un tratamiento de primer orden a las lógicas de orden superior.

Sin embargo, algo de lo que sí adolece la FOL es un tratamiento para los llamados *cuantificadores generalizados* como lo serían: algunos, la mayoría, cerca, lejos, muchos, pocos, grande, chico, mediano, y otros más. Este tipo de cuantificadores están presentes en los lenguajes naturales con mucha frecuencia. Así que un tratamiento formal para el lenguaje natural debe ser capaz de manejar este tipo de fenómenos. Más aun, hay fenómenos que aunque podrían tratarse en FOL, pierden mucho del sentido común de las nociones detrás de estos fenómenos, al hacerlo. Como son las construcciones con verbos modales que incluyen, pero no se limitan a, creer o necesitar. Por ejemplo, la oración *Bill knows Cat dances* y *Dina needs her medicine before noon*, son oraciones que pierden la intuición que poseemos de las nociones de conocer y necesitar si las plasmamos directamente en FOL. Veamos esto con un poco más de detalle. Si representamos *Bill knows Cat dances* por $know(bill, x) \wedge dance(x) \wedge cat = x$ y *Cat's web nickname is LovelyCat1990* por $Cat = LovelyCat1990$, entonces tendríamos que aceptar la fórmula $know(bill, x) \wedge dance(x) \wedge lovelycat1990 = x$. Lo que nos diría que

Bill es consciente de cada descripción con la que se identifique a Cat, incluyendo nombres de la infancia, apodos de la escuela, apelativos, etcétera. Esto no parece muy congruente con lo que uno piensa normalmente es el concepto de conocimiento de una persona, peor es si reemplazamos el conocimiento por la creencia. Este último fenómeno nos lleva a tener que salir del terreno de la lógica clásica para introducirnos en el interesante ámbito de las lógicas intensionales, lógicas donde el principio de extensionalidad no se cumple.

Pero esto no significa que hemos desperdiciado el tiempo al revisar la lógica clásica, ni que vayan a ser inútiles las representaciones que construimos en Prolog para ella. Por el contrario, nos será de gran utilidad, pues el método que seguiremos nos permitirá analizar una oración en el lenguaje natural, traducirla en una fórmula de la FOL, para llevar a cabo la evaluación de su verdad, respecto de una situación, y regresar después una respuesta en lenguaje natural. Además, aun cuando sugiriéramos un plan de acción distinto (por ejemplo, basado en una lógica modal o multimodal), al final de cuentas deberíamos ser capaces de evaluar fórmulas de la FOL, repecto de un modelo (sólo que en esos casos se haría varias veces, una vez por cada *mundo posible*). Sin mencionar, que la experiencia que trae analizar la FOL, nos prepara para el tipo de ideas que son empleadas en las lógicas no clásicas.

Ahora bien, antes mencionamos algunos otros "problemas" que la FOL presenta, postergando una discusión al respecto. Por ejemplo, la oración (1.2) tiene asociadas las fórmulas (1.3) y (1.4). Lo que presenta el inconveniente de escoger alguna de estas fórmulas para representar a la oración[2]. Y al hacerlo, indefectiblemente perdemos la fórmula que descartemos. Es decir, perdemos alguna de las *lecturas* de la oración original. Revisemos esto con más detalle. La oración original es:

(1.2) *Every woman loves some rock-star.*

Una de las lecturas de la oración nos dice que cada mujer ama a alguna estrella del rock, puede ser que cada una ame a alguna estrella diferente, esta lectura va intuitivamente asociada a la fórmula

(1.3) $\forall x(woman(x) \rightarrow \exists y(rock\text{-}star(y) \wedge love(x, y)))$

donde el cuantificador existencial está bajo el alcance del cuantificador universal. La otra lectura de la oración nos dice que cada mujer ama a una misma estrella de rock, esta lectura esta dada por la fórmula

(1.4) $\exists y(rock\text{-}star(y) \wedge \forall x(woman(x) \rightarrow love(x, y)))$

[2]Si consideráramos ambas fórmulas asociadas a la oración caeríamos en la profunda sima de la ambigüedad

donde el cuantificador universal y el existencial tienen alcances invertidos. Este problema, es clásico en la literatura concerniente a la CS, y existen varias formas de solventar la situación. Una forma de hacer esto, es darnos cuenta de que la lectura (1.4) implica a la lectura (1.3). Esta relación hace que la primera lectura reciba el nombre de más *fuerte*, mientras que la segunda recibe el nombre de más *débil*. La relación de implicación que existe entre ellas nos permite en ciertos casos usar la pragmática en el momento de hacer inferencias. De manera que en una situación donde la segunda lectura sea verdadera, también la primera lectura lo será. El primer problema que esto presenta es que no habría oportunidad de que se amara a diferentes "estrellas del rock". Pero los problemas no terminan allí. Si consideramos más cuantificadores dentro de la oración y jugamos con ellos revolviéndolos (permutándolos), obtenemos diferentes oraciones las cuales se implican unas a otras. El problema yace en que no todas las oraciones se implican de una sóla lectura (fórmula). De manera que aun cuando se asignara una sola fórmula a la oración, perderíamos alguna de las lecturas asociadas a la oración original. Peor aun, este fenómeno no sólo se presenta en cuantificadores, también las estructuras de cláusulas relativas, entre otras construcciones, causan este fenómeno. Así que lo ideal sería poder encontrar un procedimiento que dé un tratamiento uniforme a estos problemas.

Otra complicación se presenta al tratar de cuantificar predicados en las expresiones. Como por ejemplo

(1.8) *Obama has all the attributes of a democrat.*

En principio la FOL es incapaz de manejar este tipo de situaciones. La solución más directa para este inconveniente es introducir la lógica de segundo orden. Y aunque ya mencionamos la posibilidad de darle un tratamiento con la FOL a las lógicas de orden superior, en particular a la de segundo orden, no está de más revisar cómo la logica de orden superior resuelve estos inconvenientes. Principalmente porque al final del día estaremos utilizando una lógica sobre la teoría de tipos para nuestro modelo del lenguaje natural. Pero también cabe mencionar que el tratamiento de primer orden (llamado semántica de Henkin) para segundo orden no es equivalente a segundo orden (ver el capítulo 4 de [Sha99]). En lo que sigue exploraremos algunos otros lenguajes lógicos que usaremos para el modelado del lenguaje natural.

Capítulo 2

Lógica Intensional

Toca el turno de revisar algunos lenguajes de lógicas intensionales. Introduciremos la *lógica modal* y una de sus derivadas la *lógica temporal*, en sus dos vertientes, la del lenguaje proposicional y la de primer orden. Veremos también como implementar en Prolog el lenguaje de la lógica modal de primer orden.

2.1. Lógica Intensional Proposicional

Nuestro interés en la presente sección es el de revisar el lenguaje de la lógica intensional proposicional (del inglés intensional logic) y dar una rápida aproximación al modelado de expresiones del lenguaje natural mediante la lógica intensional proposicional que construiremos.

La forma más sencilla de obtener una lógica intensional, es introduciendo lo que se conoce como *operadores modales*, éstos, pueden interpretarse de diferentes maneras, dando lugar a distintas expresiones intensionales, que son hasta cierto punto, parecidas a las expresiones del lenguaje natural que pretenden modelar. Esta será la forma que seguiremos a continuación.

2.1.1. Lógica Modal y Temporal Proposicional

Con el fin de proceder requerimos de algunas definiciones que nos permitan establecer el lenguaje que queremos manejar. Damos primero las reglas sintácticas.

2.1.1 **Definición**.
Consideremos el alfabeto: (,), $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$ y letras proposicionales:

(i) Cada letra proposicional es una fórmula.

(ii) Si ϕ y ψ son fórmulas, entonces $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \rightarrow \psi$ y $\phi \leftrightarrow \psi$ son también formulas.

(iii) Dada una fórmula ϕ, entonces $\Box\phi$ y $\Diamond\phi$ son formulas.

(iv) Sólo las expresiones obtenidas aplicando un número finito de veces las reglas (i)–(iii) son fórmulas.

Las reglas (i)–(ii) son las mismas que las de la lógica proposicional clásica. La regla (iii) introduce $\Box$ y $\Diamond$, que son los antes mencionados, operadores modales. Veamos algunos ejemplos. Las siguientes son fórmulas de la lógica modal proposicional $\Diamond\phi$, $\phi \wedge \psi$, $\phi \vee \Box\psi$, $\Diamond\phi \rightarrow \Box\Diamond\psi$.

Ahora necesitamos una forma que nos permita decidir cuándo una fórmula de la lógica modal proposicional será verdadera y cuándo no. Debemos, además, dar alguna interpretación para los símbolos $\Box$ y $\Diamond$, preferentemente, de manera que coincida con alguna noción del lenguaje natural, que no se pueda capturar en la lógica proposicional tradicional (de hecho esta interpretación será la que le dé la característica intensional a este lenguaje, pero esto lo discutiremos más adelante).

Digamos que $\Diamond$ significa posiblemente, o es posible que. De manera que $\Diamond\phi$ significará es posible que ϕ, o posiblemente ϕ. De acuerdo a la concepción de posibilidad de Leibniz, para que *algo* sea posible en el mundo actual, significa que ese *algo* en realidad ocurre en un mundo *posible* (entenderemos un mundo posible como aquél en el que ocurren eventos en relación con el mundo actual, en breve formalizaremos esta noción). Digamos también que $\Box$ significa necesariamente, o es necesario que. De modo que $\Box\phi$ significará es necesario que ϕ, o necesariamente ϕ.

Si entendemos lo necesario como aquello que no puede dejar de evitar ocurrir, entonces tenemos una relación entre $\Box$ y $\Diamond$. A saber, que $\Box\phi \equiv \neg\Diamond\neg\phi$. Recíprocamente, podemos entender que lo posible es aquello que no es necesario que no suceda. De nuevo llegamos a una relación. Esta vez, $\Diamond\phi \equiv \neg\Box\neg\phi$. Esta pequeña discusión nos deja ver que en realidad sólo uno de los operadores es *básico* y que el otro puede escribirse en términos de aquél que se escoja como básico.

Así, la expresividad del lenguaje de la lógica proposicional tradicional crece, ahora somos capaces de describir situaciones como:

(2.1) *It's possible you don't understand me, but it's not necessary.*

(2.2) *If it can be raining, then it must be possible it's raining.*

(2.3) *It's possible that if it may be raining, then it must be raining.*

(2.4) *If it may be necessary that it's raining, then it must be raining.*

(2.5) *Perhaps it's raining, and maybe this is necessary.*

Escritas en símbolos de nuestro lenguaje, las oraciones (2.1)-(2.4) son:

(2.6) $\Diamond\neg e \wedge \neg\Box e$

(2.7) $\Diamond p \rightarrow \Box\Diamond p$

(2.8) $\Diamond(\Diamond p \rightarrow \Box p)$

(2.9) $\Diamond\Box p \rightarrow \Box p$

Para la oración (2.5) tenemos dos opciones, ya sea que la traduzcamos como:

(2.10) $\Diamond p \wedge \Diamond\Box p$

pero la siguiente también sería aceptable:

(2.11) $\Diamond p \wedge \Diamond\Box\Diamond p$

Esta oración muestra la escasez de expresividad que aun tenemos en este lenguaje. Sin embargo, esto lo trataremos después.

El método para encontrar la(s) fórmula(s) que le corresponden a una oración seguirá siendo intuitivo. Por ejemplo, para la oración (2.1), pensando en $\Diamond$ como en lo posible, en $\Box$ como lo necesario, usando la letra proposicional e: *you understand me*, y asociando los conectivos de la misma forma que hicimos en el capítulo anterior, obtenemos la fórmula (2.6).

Aun así, conviene declarar explícitamente las reglas que seguiremos para la traducción de las oraciones en fórmulas de este nuevo lenguaje. A parte de las reglas dadas en el capítulo anterior. Si estamos hablando de la noción de necesidad-posibilidad

- $\Diamond$ El adverbio *necessarily* (y algunos de sus "sinónimos", como *must*, *needs*) se asocia con el operador $\Box$.
- $\Diamond$ El adverbio *possibly* (y algunos de sus "sinónimos", como *maybe*, *perhaps*) se asocia con el operador $\Diamond$.

Esta será por lo general la noción que utilizaremos en nuestros ejemplos.

Ahora necesitamos una manera de decidir cuándo consideraremos que una fórmula de nuestro lenguaje será verdadera y cuándo falsa.

Identificando a los mundos posibles, en nuestra discusión anterior, con contextos de un discurso (después de todo nuestra mira está en tratar las expresiones del lenguaje natural, así que una representación de este tipo parece razonable), llegamos a la siguiente definición:

2.1.2 **Definición (Modelo de Kripke).**
Un modelo M consiste de:

(i) Un conjunto no vacío de contextos K.

(ii) Una relación binaria R en K llamada relación de accesibilidad.

(iii) Una función de evaluación V tal que para cada letra proposicional p y cada contexto $k \in K$ asigna un valor de verdad $V_k(p)$.

Usualmente al conjunto K se lo denota W pensando en la interpretación de mundos posibles que ya discutimos. Pero como ya mencionamos, los consideraremos mejor como contextos de un discurso.
Revisemos un ejemplo. Queremos modelar un contexto que considere dos letras proposicionales p y q que corresponden a las situaciones siguientes: p: *it is raining* y q: *you understand me*. Un posible modelo de Kripke para las oraciones (2.1)-(2.5) es el siguiente: $K = \{k_1, k_2\}$, $R = \{(k_1, k_2)\}$, $V_{k_1}(p) = 1$, $V_{k_1}(q) = 0$, $V_{k_2}(p) = 0$, $V_{k_2}(q) = 1$.

Ahora que sabemos lo que es un modelo en la lógica intensional, podemos establecer cuándo diremos que una fórmula será verdadera.

2.1.3 **Definición.**
Dado un modelo M con K como su conjunto de contextos, R su relación de accesibilidad, y V su evaluación, entonces $V_{M,k}(\phi)$ es el valor de verdad de ϕ en $k \in K$, dado M, definido por:

(i) $V_{M,k}(p) = V_k(p)$, para cada letra proposicional p.

(ii) $V_{M,k}(\neg\phi) = 1$ si y sólo si $V_{M,k}(\phi) = 0$

(iii) $V_{M,k}(\phi \to \psi) = 1$ si y sólo si $V_{M,k}(\phi) = 0$ o $V_{M,k}(\psi) = 1$

(iv) $V_{M,k}(\Box\phi) = 1$ si y sólo si $\forall k' \in K$ tal que kRk' y $V_{M,k'}(\phi) = 1$

(v) $V_{M,k}(\Diamond\phi) = 1$ si y sólo si $\exists k' \in K$ tal que kRk' y $V_{M,k'}(\phi) = 1$

Si $V_{M,k}(\phi) = 1$ escribimos $M, k \models \phi$. Notamos que (i)–(iii) son las mismas definiciones que para la lógica proposicional tradicional. Lo novedoso son las

reglas (iv) y (v), que plasman nuestra discusión acerca de de la posibilidad y la necesidad.

Claro está, que esta noción depende intrínsecamente de cómo pensemos la relación de accesibilidad R. Si identificamos los contextos con mundos posibles, y la relación de accesibilidad como pasar de un mundo posible a otro, entonces estamos representando precisamente la intuición de necesidad-posibilidad de Leibniz.

Aquí justamente yace la versatilidad de los operadores modales, pues dependiendo la interpretación que queramos, podemos plasmar distintos tipos de nociones. Como *deber*, *conocimiento*, *creencia*, *obligación*. Por ejemplo, podemos pensar en la relación R como una relación lineal, sin extremos, sobre el eje del tiempo, que exprese la noción de "antes que" o "anterior a" (denotémosla por $<$), y podemos denotar a los contextos como momentos en el tiempo. Esta interpretación da lugar a lo que se conoce como *lógica temporal* (en inglés *tense logic* [1]), originada por Arthur Prior en 1957[2]

En esta lógica, se suele distinguir entre la parte hacia adelante de la relación y la parte anterior a un momento dado. Así, se da lugar a cuatro operadores modales (aunque extraordinariamente sólo uno será básico), haciendo que $\Box$ se reemplace por H, si se considera la parte anterior de la relación, o bien por G, si se considera la parte hacia adelante. Para $\Diamond$, se usa P para la parte hacia atrás de la relación, y F para la parte hacia adelante. De modo que, con estas consideraciones, $H\phi$ ($G\phi$) ahora representa algo como "en todos los tiempos pasados (futuros) ϕ", mientras que $P\phi$ ($F\phi$) representa "en algún tiempo pasado (futuro) ϕ". Los operadores H, P, G y F, se conocen como *operadores temporales*, de aquí el nombre de *lógica temporal*. Esta lógica nos permite expresar conjugaciones verbales como el siguiente ejemplo sugiere:

(2.12)

p	Lola dances
Fp	Lola will dance
Pp	Lola danced
PPp	Lola had danced
FPp	Lola will have danced
PFp	Lola would dance

Con conjugaciones a nuestra disposición ahora podemos expresar cosas como:

[1] Curiosamente la palabra *tense* puede traducirse como *temporal* en algunos contextos siendo el tiempo de conjugación verbal el más común de estos contextos.

[2] Time and Modality. Oxford University Press. basado en sus cursos de 1956 sobre John Locke.

(2.13) *You are still naive, but some day you will no longer be.*

(2.14) *I'm your friend and I will always be.*

(2.15) *Martin has read King Lear, and Xavier has too.*

(2.16) *When she stroke, the knight had raised his shield.*

(2.17) *When she stroke, the knight was about to raise his shield.*

(2.18) *The castle will be put under siege or not. And if it is sieged, this will always be the case.*

(2.19) *Only if you stay forever with me shall I be entirely happy.*

en símbolos (13)–(19) quedan:

(2.20) $p \wedge F\neg p$

(2.21) $p \wedge Gp$

(2.22) $Pp \wedge Pq$

(2.23) $P(p \wedge Pq)$

(2.24) $P(p \wedge Fq)$

(2.25) $(Fp \vee F\neg p) \wedge (Fp \rightarrow HFp)$

(2.26) $Gq \rightarrow Gp$

La manera de obtener estas fórmulas a partir de las oraciones originales sigue las siguientes "reglas". Si estamos hablando de la noción temporal

- ♡ Un verbo conjugado en futuro se asocia con el operador F....
- ♡ Un verbo conjugado en antefuturo se asocia a los operadores FP... en ese orden.
- ♡ Un verbo conjugado en pasado se asocia al operador P....
- ♡ Un verbo conjugado en antepasado se asocia a los operadores PP.
- ♡ Un verbo conjugado en postpretérito se asocia a los operadores PF en ese orden.

Como podemos ver, estas "reglas" coinciden con lo mostrado en (2.12). Las fórmulas (2.20)–(2.26) se obtienen usando las letras proposicionales adecuadas junto con estas reglas. Por ejemplo, (2.20) se obtiene con p: *you are still naive* y la regla para la conjugación en tiempo futuro.

Desafortunadamente no todas las conjugaciones verbales del lenguaje natural se pueden producir de esta manera, ni tampoco toda combinación de los símbolos H, P, G, y F corresponde a una conjugación. Por lo que de nueva cuenta, notamos una falta en el poder expresivo de este lenguaje.

Por supuesto, podemos combinar las dos interpretaciones que dimos antes (considerando ambas, la relación de mundos posibles, y la relación de anterior a), para ser capaces de modelar construcciones contrafactuales, como por ejemplo:

(2.27) *If I had scored one more goal I could have won the game.*

en símbolos:

(2.28) $Pp \rightarrow \Diamond Pq$

Así, vemos que podemos incluir más de un operador modal en el lenguaje intensional. En cuanto a las otras formas de interpretar $\Box$ y $\Diamond$, algunas, nos permiten expresar algunas otras nociones que sirven en el modelado del lenguaje natural como ya lo mencionamos: *deber* u *obligación*, *creencia*, etc. Una de las más útiles es la que conlleva al lenguaje de la lógica epistémica. No entraremos en muchos detalles pero suponiendo que el operador † represente a la creencia, podríamos representar oraciones como

(2.29) *I believe tomorrow may rain.*

representada por

(2.30) $\dagger F \Diamond r$

donde r:*rain*, $\Diamond$:*may*, F:*tomorrow*.

2.1.2. Intensionalidad de la Lógica Modal Proposicional

Hasta ahora, hemos revisado cómo varias nociones del lenguaje natural pueden representarse mediante el lenguaje de la lógica modal y temporal. Pero no hemos discutido acerca de la intensionalidad de estos lenguajes. Ni siquiera hemos dicho qué entendemos por intensionalidad. Para tal propósito, necesitamos definir el principio de extensionalidad, también conocido como principio de Leibniz. Y también lo que son los contextos opacos y transparentes.
Antes de eso necesitamos dar una definición más

2.1.4 **Definición**.
Sea $M = (K, R, V)$. Una fórmula ϕ es inferencia lógica de una fórmula ψ si, para todo modelo M, tal que $M \models \psi$, se tiene que $M \models \phi$. Escribimos $\psi \models \phi$.

Donde $M \models \phi$ significa que $M, k \models \phi$ para todo contexto $k \in K$.

2.1.5 **Definición**.
Decimos que un contexto es opaco si no satisface el principio de extensionalidad (débil):

(i) $s \leftrightarrow t \models \phi \leftrightarrow \phi^{[t/s]}$

Donde, $\phi^{[t/s]}$ significa sustituir las ocurrencias de s en ϕ por t. Si (i) se satisface en un contexto, decimos que el contexto es transparente.

2.1.6 **Definición**.
Una lógica intensional es una lógica que contiene contextos opacos.

Ahora podemos ver por qué hemos asegurado al principio que la lógica modal proposicional es una lógica intensional. Probablemente en los ejemplos pasados se haya podido ver que si en algún contexto k se tiene que $p \leftrightarrow q$ es verdadera, eso no garantiza que $\Box p \leftrightarrow \Box q$ también lo sea, pues esto depende de todos los contextos k' accesibles a partir de k, y la verdad de $p \leftrightarrow q$ no nos dice nada acerca de esto.

Un ejemplo esclarece las cosas. Supongamos que tenemos lo siguiente:

(2.31) *The brightest girl of the classroom is the richest girl of the school.*

Traduzcámosla por $p \leftrightarrow q$, donde p:*the brigthest girl of the classroom*, q:*the richest girl of the school*, y consideraremos al verbo de identidad *is* como al conectivo $\leftrightarrow$.

(2.32) *Maybe the brightest girl of the classroom is possibly the richest girl of the school.*

La representación de esta última sería $\Diamond p \leftrightarrow \Diamond q$. Donde consideraremos tanto a *maybe* como a *possibly*, representados por el símbolo $\Diamond$. Ahora necesitamos un modelo para efectuar la evaluación. Tomemos $M = (K, R, V)$, con $K = \{w_1, w_2\}$, $R = \{(w_2, w_1)\}$ y $V_{w_1}(p) = V_{w_2}(p) = V_{w_1}(q) = V_{w_2}(q) = 1$. Entonces de acuerdo a la semántica, $p \leftrightarrow q$ es verdadera tanto en w_1 como en w_2, sin embargo, la fórmula $\Diamond p \leftrightarrow \Diamond q$ sólo es verdadera en w_2, mientras que en w_1 es falsa. Pues el mundo w_1 no está relacionado con algún mundo en el cual sea verdadera $p \leftrightarrow q$. Por lo tanto $V_{M,w1}(\Diamond p \leftrightarrow \Diamond q) = 0$.

Pero precisamente esta característica es lo que permite al lenguaje de la lógica intensional su poder expresivo, y lo que nos permite representar

mediante ella situaciones del lenguaje natural complicadas. Por ejemplo, introduzcamos la oración

(2.33) *Billie likes being the boyfriend of the richest girl of the school.*

Entonces parecería que la siguiente es una conclusión válida:

(2.34) *Billie likes being the boyfriend of the brightest girl of the classroom.*

La lógica tradicional así lo permitiría. Sin embargo, en el lenguaje natural puede no ser una conclusión deseada. Si lo único que le interesa a Billie es el dinero, entonces podríamos dudar de la validez del esquema. Es decir, la conclusión está condicionada a la intención[3] que Billie tenga respecto a su novia. Esto viola el principio de extensionalidad, convirtiéndolo en un contexto opaco, y por lo tanto, en una expresión intensional. La cual puede ser representada con la maquinaria expuesta hasta el momento.

Al igual que este ejemplo, podemos construir muchos otros, basándonos en ciertas nociones del lenguaje natural. En los siguientes ejemplos las primeras dos oraciones no derivan a la tercera:

- Discurso Indirecto:

  (2.35) *Lola said that John is brave.*

  (2.36) *John is tall.*

  (2.37) *Lola said that John is tall.*

- Citación:

  (2.38) *The king spoke saying: "Today we remember the fallen heros of our land".*

  (2.39) *Today is July 4th.*

  (2.40) *The king spoke saying: "July 4th we remember the fallen heros of our land".*

- Actitudes proposicionales, como descubrir, creer, sospechar, conocer, intuir:

  (2.41) *Greason suspects the blood spots belong to the murderer.*

  (2.42) *Anna is the murderer.*

  (2.43) *Greason suspects the blood spots belong to Anna.*

[3]No confundir con lo que hemos traducido por *intensión*.

- Intensiones como, buscar, desear:

  (2.44) *Derek wishes for not making mistakes this year.*

  (2.45) *Not Making mistakes this year is equal to winning the gold glove.*

  (2.46) *Derek wishes for winning the gold glove.*

- Designación temporal:

  (2.47) *Mary is the queen.*

  (2.48) *In the fifthteen century Columbus was sent by the queen to find new lands.*

  (2.49) *In the fifthteen century Columbus was sent by Mary to find new lands.*

- Modalidad:

  (2.50) *Two necessarily is the succesor of one.*

  (2.51) *Two is the number of mars'moons.*

  (2.52) *The number of mars'moons necessarily is the succesor of one.*

Además de éstas, hay muchas otras construcciones dando lugar a contextos opacos. Casi cualquier categoría de expresiones contiene elementos que pueden construir contextos opacos, por ejemplo, adjetivos como, presunto y supuesto, adverbios como, aparentemente y normalmente, etcétera. Después regresaremos a esas cuestiones con más detalle.

2.2. Lógica de Predicados Intensional

Desviamos nuestra atención en esta sección hacia la lógica de primer orden y sus capacidades para modelar expresiones del lenguaje natural. Naturalmente, dotaremos al lenguaje de la lógica de predicados con un operador intensional, revisaremos algunos ejemplos para ver cómo funciona, qué tipo de expresiones están fuera de nuestro alcance, y si nuestras intuiciones se satisfacen adecuadamente.

2.2.1. Lógica de Predicados Modal

El lenguaje de la lógica de predicados modal se obtiene añadiendo los operadores modales $\Box$ y $\Diamond$ al lenguaje de la lógica de predicados tradicional.

La sintaxis es como sigue:

2.2.1 **Definición**.
Consideremos el alfabeto: $(,)$, $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$, $\exists$, $\forall$, variables $x, y, z, \ldots$, símbolos de constante $a, b, c, \ldots$, y símbolos de relación $P, Q, R, \ldots$.

(i) Cada símbolo de constante y cada variable es un término.

(ii) Si t_1, t_2 son términos, entonces $t_1 = t_2$ es una fórmula.

(iii) Si R es un símbolo de *n-relación* y $t_1, \ldots, t_n$ son términos, entonces $R(t_1, \ldots, t_n)$ es una fórmula.

(iv) Si ϕ y ψ son fórmulas, entonces $\neg\phi$, $\phi \vee \psi$, $\phi \wedge \psi$, $\phi \rightarrow \psi$ y $\phi \leftrightarrow \psi$ son fórmulas.

(v) Si ϕ es una fórmula y x una variable, entonces $\exists x\phi(x)$ y $\forall x\phi(x)$ son fórmulas.

(vi) Si ϕ es una fórmula, entonces $\Box\phi$ y $\Diamond\phi$ son fórmulas.

(vii) Sólo son fórmulas las expresiones que se obtengan usando un número finito de veces las reglas (ii)–(vii).

Lo primero es ver ejemplos de expresiones bien construidas.

$\forall x \exists y(R_2(x, y) \rightarrow Q_2(x, y))$,
$\forall x \Box R_1(x)$,
$\Box \forall x R_1(x)$,
$\Box \Diamond \exists x(Q(x) \vee R(x))$.

Por supuesto que este lenguaje nos proporciona de una maquinaria más poderosa, que nos permite expresiones cuantificadas, de manera que ahora podemos construir oraciones como:

(2.53) *Necessarily every man is mortal.*

(2.54) *Possibly there is an element which has one proton in its core.*

(2.55) *Each molecule necessarily is constituted of atoms.*

Una vez más, la forma de obtener las expresiones del lenguaje de la lógica de primer orden intensional (denotémoslo IFOL), que corresponden a las oraciones, es usar la intuición junto con las reglas mencionadas antes.

Antes de dar la semántica necesitamos definir lo siguiente:

2.2.2 **Definición**.
Un modelo de la lógica intensional de predicados es un cuarteto $\langle K, R, D, V, \rangle$, donde, K es un conjunto no vacío de contextos, $R \subset K^2$ es la relación de accesibilidad, D en un conjunto no vacío llamado el dominio del modelo, y V es una función de interpretación tal que:

(i) Si c es una constante del lenguaje de la lógica de predicados modal, entonces $V_k(c) \in D$

(ii) Si $k \in K$, y Q es un símbolo de *n- relación*, entonces $V_k(Q) \subset D^n$.

2.2.3 **Definición**.
Una asignación es una función que a cada variable del lenguaje de la lógica de predicados modal, le asocia un elemento de D.

2.2.4 **Definición**.
Sean α, β dos asignaciones y x una variable del lenguaje de la lógica de predicados modal. Si para toda variable y distinta de x se tiene que $\alpha(y) = \beta(y)$ decimos que β es una x-*variante* de α.

La semántica del lenguaje es como sigue:

2.2.5 **Definición**.
Sean $M = \langle K, R, D, V \rangle$ un modelo de la lógica modal, $k \in K$ un contexto, y α una asignación. Definimos el valor de verdad de una fórmula ϕ en k, respecto de M y α, denotado $V_{M,k,\alpha}(\phi)$, como:

(i) Si Q es un símbolo de n-*relación*, y t_1,...,t_n son constantes o variables y $\phi = Q(t_1, \ldots, t_n)$, entonces $V_{M,k,\alpha}(\phi) = 1$ si $Q(V_{M,k,\alpha}(t_1),\ldots,V_{M,k,\alpha}(t_n)) \in V_{M,k,\alpha}(Q)$.

(ii) Si ϕ es una fórmula, entonces $V_{M,k,\alpha}(\neg\phi) = 1$ si y sólo si $V_{M,k,\alpha}(\phi) = 0$

(iii) Si ϕ y ψ son fórmulas, entonces $V_{M,k,\alpha}(\phi \wedge \psi) = 1$ si y sólo si $V_{M,k,\alpha}(\phi) = 1$ y $V_{M,k,\alpha}(\psi) = 1$

(iv) Si ϕ y ψ son fórmulas, entonces $V_{M,k,\alpha}(\phi \vee \psi) = 1$ si y sólo si $V_{M,k,\alpha}(\phi) = 1$ o $V_{M,k,\alpha}(\psi) = 1$

(v) Si ϕ y ψ son fórmulas, entonces $V_{M,k,\alpha}(\phi \rightarrow \psi) = 1$ si y sólo si $V_{M,k,\alpha}(\phi) = 0$ o $V_{M,k,\alpha}(\psi) = 1$

(vi) Si ϕ y ψ son fórmulas, entonces $V_{M,k,\alpha}(\phi \leftrightarrow \psi) = 1$ si y sólo si $V_{M,k,\alpha}(\phi) = 1$ y $V_{M,k,\alpha}(\psi) = 1$ o $V_{M,k,\alpha}(\phi) = 0$ y $V_{M,k,\alpha}(\psi) = 0$.

(vii) Si ϕ es una fórmula y x es una variable, entonces $V_{M,k,\alpha}(\forall x\phi(x)) = 1$ si y sólo si $V_{M,k,\beta}(\phi) = 1$ para toda β x-*variante* de α.

(viii) Si ϕ es una fórmula y x es una variable, entonces $V_{M,k,\alpha}(\exists x\phi(x)) = 1$ si y sólo si hay una β x-*variante* de α tal que $V_{M,k,\beta}(\phi) = 1$.

(ix) Si ϕ es una fórmula, entonces $V_{M,k,\alpha}(\Box\phi) = 1$ si y sólo si $V_{M,k',\alpha}(\phi) = 1$ para todo $k' \in K$ tal que Rkk'.

(x) Si ϕ es una fórmula, entonces $V_{M,k,\alpha}(\Diamond\phi) = 1$ si y sólo si hay un $k' \in K$ tal que Rkk' y $V_{M,k',\alpha}(\phi) = 1$.

donde:
Si c es una constante, $V_{M,k,\alpha}(c) = V_k(c)$, y
si x es una variable, $V_{M,k,\alpha}(x) = \alpha_k(x)$.

Se puede notar que los valores que reciben las constantes y las variables no son propiamente valores de verdad, motivo por el que se excluyen dentro de los incisos correspondientes en la definición. Sin embargo, es útil darles un valor a los *términos* bajo $V_{M,k,\alpha}$ porque en lo por venir seguiremos una estrategia similar.

Cuando el modelo que se trabaja está claro, omitiremos el índice M en $V_{M,k,\alpha}$ escribiendo solamente $V_{k,\alpha}$.

Un último comentario al respecto de esta definición, es que en ella hemos dado un caso particular que debe mencionarse, y es, que estamos considerando un dominio D constante, para cada contexto $k \in K$. Esto se traduce en que las constantes de D se denominan *designadores rígidos*, y ya habrá tiempo para discutir las consecuencias de esta decisión.

Ahora introduciremos los conceptos de lecturas *de dicto* y *de re* de una oración. Para esto, regresemos a la discusión de modalidad que dejamos pendiente en la sección pasada. Allí vimos que la oraciones (2.50) y (2.51) no concluyen la oración (2.52). Por lo menos no en la lectura más clara de esta oración, sin embargo, también podemos entender (2.52) de la siguiente manera:

(2.56) *The number which is the number of mars'moons is necessarily the succesor of one.*

En esta lectura de la oración (2.52) podemos pensar que en efecto se sigue a partir de (2.50) y (2.51).

Sean N el número de lunas de Marte y s la función sucesor, entonces podemos escribir en símbolos (2.52) y (2.56) respectivamente como:

(2.57) $\Box\exists x(x = N \wedge x = s(1))$.

(2.58) $\exists x(x = N \wedge \Box(x = s(1)))$.

(2.57) es la lectura *de dicto* de la oración (pues la modalidad está sobre una proposición o *dictum*), mientras que (2.58) es la lectura *de re* (la modalidad está sobre una entidad o *res*). Entonces vemos que la diferencia entre las lecturas yace en el alcance del operador $\Box$. Mientras que en (2.57) el alcance de $\Box$ es $\exists x(x = N \wedge x = s(1))$ en (2.58) es solamente $x = s(1)$.

Si todas las variables dentro del alcance de $\Box$ están acotadas por cuantificadores también dentro de este alcance. La modalidad se llama *de dicto* (como en (2.57)). Si, por otro lado, hay una varible libre dentro del alcance de $\Box$ y acotada fuera de este alcance por un cuantificador, la modalidad se llama *de re* (como en (2.58)).

La razón para mencionar estas modalidades, es que aparecen con cierta frecuencia en el lenguaje natural, por ejemplo la modalidad *de re* se puede ejemplificar con la oración:

(2.59) *Each one of the born crabs may die in the first days.*

En símbolos:

(2.60) $\forall x \Diamond D(x)$

Es claro que no es lo mismo que la lectura *de dicto*:

(2.61) *It may be that each of the born crabs die in the first days.*

En símbolos:

(2.62) $\Diamond \forall x D(x)$

Otra forma de construir una lógica intensional de predicados es usar operadores temporales en vez de modales. E igualmente, estos operadores presentan los dos tipos de lectura que aparecen con los operadores modales. Veamos un ejemplo:

(2.63) *The moon is going to be full.*

En símbolos:

(2.64) $\mathbf{F}\exists x(\forall y(Moon(y)) \leftrightarrow y = x \wedge Full(x))$

es la lectura de dicto y:

(2.65) $\exists x(\forall y(Moon(y) \leftrightarrow y = x) \wedge \mathbf{F}Full(x))$

es la lectura de re.

También, se pueden combinar los operadores modales y temporales para modelar oraciones como:

(2.66) *Djokovic may one day win Wimbledon.*

que en símbolos del lenguaje sería:

(2.67) $\Diamond \mathbf{F} Wimbledon(d)$

Manejo de la Identidad

Las decisiones que tomamos anteriormente, de tomar un dominio constante D y de considerar a las constantes como designadores rígidos resulta en la validez de algunos principios como:

(2.68) $b = c \rightarrow \Box(b = c)$

Pensando en $\Box$ con la noción de necesidad, no tenemos problema con esta regla, pero algo muy distinto sucede si tratamos a $\Box$ con la noción de creencia. Pensemos en las siguientes oraciones:

(2.69) *Neptune is the god of sea.*

(2.70) *The Greeks thought the god of sea is the god of sea.*

(2.71) *The Greeks thought Neptune is the god of sea.*

En este caso no parece que la noción de creencia del lenguaje natural esté plasmada correctamente.

Algo similar ocurre si pensamos en $\Box$ como deber:

(2.72) *John is innocent.*

(2.73) *The judge must declare innocent people as innocent.*

(2.74) *The judge must declare John as innocent.*

Así, vemos que la noción de deber no queda bien representada.

Por último, revisemos algunas oraciones que carecen de traducción directa en el lenguaje de la lógica de predicados modal.

(2.75) *If Mary is wise, then she may notice there is a thing John and Peter have in common.*

Como podemos ver, en esta oración se está cuantificando sobre una propiedad, no sobre individuos. Algo que no se puede tratar con el lenguaje de

predicados modal. La forma en que podemos pensar esta oración es introduciendo una variable de predicado X, así, la oración podría escribirse con símbolos de nuestro lenguaje como:

(2.76) $W(m) \rightarrow \Diamond(\exists X(X(j) \wedge X(p)))$

Pero esta notación está fuera del lenguaje de la lógica de predicados modal. Algo similar pasa con la siguiente oración:

(2.77) *Obama has all the attibutes of a democrat.*

Usando variables de predicado quedaría:

(2.78) $\forall X(\forall x(D(x) \rightarrow X(x)) \rightarrow X(o))$

Hay además, oraciones como:

(2.79) *Carl is running quickly.*

que en nuestro lenguaje tendría que asociarse de la siguiente manera para poder representarse con símbolos:

(2.80) *Carl is running and Carl is quick.*

Sin embargo, hemos perdido el significado original de la oración. Sobre todo si comparamos a un Carl que corra la maratón, con uno que esté inscrito en la competencia de los 100 metros planos. ¿Quién de ellos será el rápido? Lo mismo ocurre con algunos adjetivos relativos como grande y pequeño, por ejemplo, la oración:

(2.81) *Christie is a large ant.*

tiene que asociarse de la manera:

(2.82) *Christie is large and Christie is an ant.*

para poder representarse en nuestro lenguaje. Pero igualmente, podemos ver que no representa lo mismo que la oración original. Pues nos preguntamos qué tan grande en realidad podrá ser Christie si es una hormiga. En el siguiente capítulo mostraremos cómo salvar estos problemas, considerando un lenguaje más poderoso que la lógica de predicados modal. Por lo pronto dejaremos nuestra discusión momentáneamente y pasaremos a ver como se podría representar lo que hemos visto hasta el momento en Prolog.

2.3. Representando la lógica modal en Prolog

En esta parte revisaremos la manera de representar la IFOL en Prolog. Empezando con las nuevas fórmulas que aquí aparecen, luego veremos como representar los modelos de Kripke y terminaremos definiendo la satisfacibilidad de las fórmulas de la IFOL respecto de un modelo de Kripke, un mundo, y una asignación.

2.3.1. Representación de las fórmulas y modelos de la lógica modal

La manera que emplearemos para representar a la lógica modal en Prolog seguirá principios muy similares a los que utilizamos en la representación de la FOL. Los símbolos de constante y relación, las variables, y las fórmulas de la FOL que reciben una representación idéntica en la IFOL permanecerán con la misma representación. Las nuevas representaciones corresponderan a las fórmulas con operadores modales. Usaremos el átomo `poss` para representar la fórmula con operador $\Diamond$ y el átomo `nec` para representar a la fórmula con operador $\Box$. De modo que los predicados de Prolog que representan a las fórmulas $\Box\phi$ y $\Diamond\phi$, serán `nec(PHI)`, `poss(PHI)`, respectivamente. La única representación que sufre un cambio significativo es la del modelo. Pues recordemos que para la FOL se tenía un solo mundo y una sola función de interpretación, mientras que ahora, en el modelo de Kripke se tienen varios mundos en los que la función de interpretación puede tomar diferentes valores. Usaremos el átomo `kmodel` para saber que nos referimos a un modelo de Kripke, y el predicado `kmodel/3` llevará en su primer argumento el conjunto de los mundos posibles, representado por una lista, en su segundo argumento la relación de accesibilidad, representada por una lista de parejas de elementos de la primera lista, y en su tercer argumento una lista de modelos tradicionales de la FOL, en la misma forma en que lo veníamos manejando, por cada mundo del conjunto de mundos. Esto quedará mejor entendido con un ejemplo. Representemos el modelo de Kripke $M = (W, R, D, F)$, sobre el alfabeto A, donde: $A = \{alis, bill, cat, dina, old_1, beverage_1, dancer_1, dance_1, man_1, woman_1, rock\text{-}star_1, walk_1, run_1, love_2, like_2, drink_2, give_3, offer_3\}$, $W = \{w_1, w_2\}$, $R = \{(w_1, w_2), (w_2, w_2)\}$, $D = \{a, b, c, d, e, f, g\}$, y F dada por:

$F_{w_1}(alis) = a$	$F_{w_1}(woman_1) = \{a, c, d\}$
$F_{w_1}(bill) = b$	$F_{w_1}(rock\text{-}star_1) = \{b, e\}$
$F_{w_1}(cat) = c$	$F_{w_1}(walk_1) = \{a, d\}$
$F_{w_1}(dina) = d$	$F_{w_1}(run_1) = \{b\}$
$F_{w_1}(old_1) = \{d, e\}$	$F_{w_1}(love_2) = \{(b, a), (c, b)\}$
$F_{w_1}(beverage_1) = \{f, g\}$	$F_{w_1}(like_2) = \{(a, b), (a, e), (c, e), (d, e)\}$
$F_{w_1}(dancer_1) = \{c\}$	$F_{w_1}(drink_2) = \emptyset$
$F_{w_1}(dance_1) = \emptyset$	$F_{w_1}(give_3) = \emptyset$
$F_{w_1}(man_1) = \{b\}$	$F_{w_1}(offer_3) = \{(a, b, g)\}$
$F_{w_2}(alis) = a$	$F_{w_2}(woman_1) = \{a, c, d\}$
$F_{w_2}(bill) = b$	$F_{w_2}(rock\text{-}star_1) = \{b, e\}$
$F_{w_2}(cat) = c$	$F_{w_2}(walk_1) = \{a, e\}$
$F_{w_2}(dina) = d$	$F_{w_2}(run_1) = \emptyset$
$F_{w_2}(old_1) = \{d, e\}$	$F_{w_2}(love_2) = \{(b, b), (c, b)\}$
$F_{w_2}(beverage_1) = \{f, g\}$	$F_{w_2}(like_2) = \{(a, b), (a, e), (c, b), (c, e), (d, e)\}$
$F_{w_2}(dancer_1) = \{c\}$	$F_{w_2}(drink_2) = \{(a, f), (c, g)\}$
$F_{w_2}(dance_1) = \{c\}$	$F_{w_2}(give_3) = \{\}$
$F_{w_2}(man_1) = \{b, e\}$	$F_{w_2}(offer_3) = \{(d, c, f)\}$

La división es para facilitar la lectura de los valores que toma la función F en w_1 y w_2. Entonces este modelo quedaría representado de la siguiente manera

```
kmodel([w1,w2],
       [(w1,w2),(w2,w2)],
       [w1([a,b,c,d,e,f,g],
          [f(0,alis,a),
          f(0,bill,b),
          f(0,cat,c),
          f(0,dina,d),
          f(1,old,[d,e]),
          f(1,beverage,[f,g]),
          f(1,dancer,[c]),
          f(1,dance,[]),
          f(1,man,[b]),
          f(1,woman,[a,c,d]),
          f(1,rock_star,[b,e]),
          f(1,walk,[a,d]),
          f(1,run,[b]),
          f(2,love,[(b,a),(c,b)]),
          f(2,like,[(a,b),(a,e),(c,e),(d,e)]),
          f(2,drink,[]),
          f(3,give,[]),
```

```
    f(3,offer,[(a,b,g)])
]),
w2([a,b,c,d,e,f,g],
    [f(0,alis,a),
    f(0,bill,b),
    f(0,cat,c),
    f(0,dina,d),
    f(1,old,[d,e]),
    f(1,beverage,[f,g]),
    f(1,dancer,[c]),
    f(1,dance,[c]),
    f(1,man,[b,e]),
    f(1,woman,[a,c,d]),
    f(1,rock_star,[b,e]),
    f(1,walk,[a,e]),
    f(1,run,[]),
    f(2,love,[(b,b),(c,b)]),
    f(2,like,[(a,b),(a,e),(c,b),(c,e),(d,e)]),
    f(2,drink,[(a,f),(c,g)]),
    f(3,give,[]),
    f(3,offer,[(d,c,f)])
])
]
).
```

2.3.2. Representando la satisfacibilidad de la lógica modal de primer orden

Ahora nos dedicaremos a plasmar la definición de satisfacibilidad de una fórmula de la IFOL en Prolog. La forma de hacerlo es muy similar a la que se usó al representar la satisfacibilidad de la FOL. La única diferencia radica en que esta vez, aparte de una fórmula, un mundo y una asignación, requerimos de un *mundo posible* para efectuar la evaluación. Nombraremos al predicado que representará la satisfacción mediante `satisface/5`, cuyo primer argumento será una fórmula a evaluar, el segundo un modelo de Kripke, el tercero un mundo, el cuarto una asignación, y el quinto, como antes, la bandera que nos indica la polaridad en la que deseamos llevar a cabo la evaluación. De manera que una consulta usual sería algo como

```
?- satisface(FORMULA,KMODEL,WORLD,ASSIGNMENT,POLARITY).
```

Empecemos pues con nuestra tarea, manejando el conectivo más sencillo: la

negación. Para ello nos son suficientes las cláusulas

```
satisface(not(Formula),KModel,World,G,pos):-
      satisface(Formula,KModel,World,G,neg).
satisface(not(Formula),KModel,World,G,neg):-
      satisface(Formula,KModel,World,G,pos).
```

Podemos ver que las cláusulas son idénticas a las que representaban a la negación para la FOL, sólo que usando el predicado `satisface` en vez de `sat`. Para los demás conectivos la estrategia es completamente similar. Veamos las cláusulas para la conjunción por ejemplo.

```
satisface(and(Formula1,Formula2),KModel,World,G,pos):-
   satisface(Formula1,KModel,World,G,pos),
   satisface(Formula2,KModel,World,G,pos).
satisface(and(Formula1,Formula2),KModel,World,G,neg):-
   satisface(Formula1,KModel,World,G,neg);
   satisface(Formula2,KModel,World,G,neg).
```

En cuanto a las fórmulas cuantificadas la descripción es también similar. Igualmente ocurre para los símbolos de relación. Las fórmulas que sí introducen cambios en la manera de hacer las cosas son las fórmulas con operadores modales, veamos por ejemplo para el operador $\Diamond$

```
satisface(poss(Formula),KModel,World,G,pos):-
   extractRel(KModel,Rel),
   member((World,RWorld),Rel),
   satisface(Formula,KModel,RWorld,G,pos).
```

Básicamente lo que hacen lás cláusulas es imitar la definición de satisfacción de una fórmula con operador rombo. Primero, el predicado `extractRel/2`, extrae la relación de accesibilidad del modelo. Luego, el predicado `member/2` busca por algún mundo `RWorld`, relacionado con `World`, a partir de la relación previamente extraída. La última cláusula trata entonces de ver si la fórmula sin el operador rombo es satisfacible en ese mundo relacionado. Esto es la versión para la polaridad positiva. Para la polaridad negativa hay más casos a considerar. Pues si la relación de accesibilidad es vacía la formula $\Diamond\phi$ es falsa en cualquier mundo. Lo mismo sucede si el mundo en el que se está llevando a cabo la evaluación está aislado (no hay mundos relacionados con él). O bien si simplemente la fórmula ϕ es falsa en cualquier mundo relacionado con aquél en el que se efectúa la evaluación. Esta discusión se plasma en las cláusulas que siguen

```
satisface(poss(Formula),KModel,World,G,neg):-
   extractRel(KModel,Rel),
   Rel = [].
satisface(poss(Formula),KModel,World,G,neg):-
   extractRel(KModel,Rel),
```

```
   \+ member((World,RWorld),Rel).
satisface(poss(Formula),KModel,World,G,neg):-
   extractRel(KModel,Rel),
   setof(X,member((World,X),Rel),ALL),
   setof(X,
      (
      member((World,X),Rel),
      satisface(Formula,KModel,X,G,neg)
      ),
      ALL
   ).
```

Para el operador de necesitación $\Box$ tenemos dos opciones. Usar la relación que existe entre los operadores $\Box\phi \equiv \neg\Diamond\neg\phi$, y emplear la representacion que ya construimos para el rombo y la negación. O establecer directamente las condiciones de satisfacibilidad como hemos hecho para el resto de las fórmulas. Como ésta es la forma que seguiremos, debemos ver que las condiciones excepcionales sobre la relación de accesibilidad afecten a la polaridad positiva de la evaluación.

```
satisface(nec(Formula),KModel,_,_,pos):-
   extractRel(KModel,Rel),
   Rel = [].
satisface(nec(Formula),KModel,World,_,pos):-
   extractRel(KModel,Rel),
   \+ member((World,RWorld),Rel).
satisface(nec(Formula),KModel,World,G,pos):-
   extractRel(KModel,Rel),
   \+ Rel = [],
   setof(X,member((World,X),Rel),ALL),
   setof(X,
      (
      member((World,X),Rel),
      satisface(Formula,KModel,X,G,pos)
      ),
      ALL
   ).
```

Una vez representada la satisfacibilidad, el resto es sencillo. Si queremos verificar la satisfacibilidad de una fórmula de la IFOL en un modelo de Kripke, dados un mundo y una asignación, basta realizar la consulta

```
?- satisface(FORMULA,MODELO_KRIPKE,MUNDO,ASIGNACION,pos).
```

Una vez más debemos mencionar algunos inconvenientes técnicos, que se

presentan al dejar el código exactamente de la manera en que lo presentamos arriba. Pues, en principio, se requiere que la fórmula sea una expresión bien construida de la IFOL. El código final realiza estos chequeos y se encarga de producir el resultado esperado.

Capítulo 3

Teoría de Tipos y Cálculo Lambda

En esta parte revisaremos la aplicación de la *teoría de tipos* en expresiones del lenguaje natural. También discurriremos sobre el cálculo λ. Una extraordinaria herramienta que nos servirá para formalizar un proceso que venimos manejando con cierta dificultad hasta el momento: traducir una oración del lenguaje natural en una fórmula de un lenguaje lógico, por ejemplo, de la FOL.

3.1. Teoría de Tipos Estándar

La *teoría de tipos* es un sistema lógico de orden superior. En este lenguaje se permite cuantificar no sólo sobre individuos sino sobre variables de cualquier *tipo* (en breve diremos a que nos referimos con esto). Esto es en particular útil cuando nuestro objetivo son expresiones del lenguaje natural, que contiene relaciones entre propiedades, por ejemplo, las oraciones (2.75) y (2.77) de la sección anterior. Pero el lenguaje natural también contiene expresiones que atribuyen propiedades a las propiedades de entidades. Por ejemplo, la propiedad, *color*, expresa una propiedad de propiedades de individuos, mientras que *morado* es una propiedad de individuos, a la que se puede aplicar *color*. Por tanto, en oraciones como:

(3.1) *Purple is a color.*

podemos representar *color* mediante $\mathscr{C}$ y *morado* mediante P, y escribir con símbolos $\mathscr{C}(P)$ como representación de (3.1), donde aparece explícitamente que *color* se aplica a *morado*. Curiosamente en la mayoría de los textos que tratan esta punto ponen este mismo ejemplo del predicado *color*. Pero hay

muchas más propiedades de propiedades: *temperatura*, como en *el horno alcanza una alta temperatura*, *personalidad*, como en *la personalidad de Bill es pasiva*, *carácter*, como en *tiene un carácter recio*, *constitución*, como en *la constitución del edificio es sólida*. Adjetivos calificativos o sustantivos que se componen de diversas propiedades. Por ejemplo, la oración

(3.2) *The United States of America has only two different political parties.*

Donde podemos ver que el sustantivo *political parties* no es en sí una sola propiedad, sino que en realidad comprende una serie (conjunto) de propiedades, en este caso de sólo dos elementos. Un ejercicio interesante se produce al examinar algunos títulos que usamos normalmente al referirmos a ciertas personas, animales o cosas, y tratar de describir los componentes que caracterizan dicho título. Por ejemplo, podemos pensar que una "estrella del rock" se compone de una persona, que es músico, a veces desempeña labores de vocalista, tiene conciertos, compone música, etcétera. Al hacer esto estamos creando lo que en otras áreas de AI se conoce como la red (o marco) semántica(o). El por qué tomar a algunas como propiedades simples y otras como compuestas es un tópico de amplia discusión tanto de sintáctica como de semántica del lenguaje, así que no ahondaremos mucho más en este tema. Pero nuestra pequeña incursión sirve para ver que en realidad hay muchas más propiedades de propiedades de las que se piensa normalmente.

También hay muchos modificadores, como los adverbios y los adjetivos relativos, como los revisados en los ejemplos (2.79) y (2.81), que son fáciles de representar en el lenguaje de la teoría de tipos, sin perder la noción que tienen en el lenguaje natural (como ocurría en la lógica de predicados modal).

La teoría de tipos supone que las entidades están localizadas en niveles o estratos, así, dos expresiones que se refieran a entidades en diferentes estratos, se dice que son de diferentes tipos. Con el fin de emplear la teoría de tipos, para representar expresiones del lenguaje natural, solamente requeriremos de dos tipos básicos, el tipo de las expresiones que se refiere a las entidades, que representaremos por e, y el tipo que se refiere a expresiones que toman valores de verdad, que representaremos por t. El resto de los tipos se construirán a partir de estos. El conjunto de todos los tipos se define por:

3.1.1 **Definición**.
El conjunto de los tipos, $\mathbf{T}$, es el conjunto que satisface:

(i) $e, t \in \mathbf{T}$

(ii) Si $a, b \in \mathbf{T}$, entonces $\langle a, b \rangle \in \mathbf{T}$

(iii) Nada es elemento de $\mathbf{T}$ excepto aquello que satisface (i) o (ii)

La cláusula (ii) es la que se encarga de generar a los tipos "compuestos", o sea, el resto de tipos que necesitaremos para nuestro lenguaje. En principio, un número infinito (aunque en la práctica sólo necesitaremos algunos de ellos). La idea en la notación que manejamos es: si una expresión α, tiene tipo $\langle a, b\rangle$, entonces puede aplicarse a una expresión β, de tipo a, para dar una expresión $\alpha(\beta)$, de tipo b. Este proceso se conoce como *aplicación funcional* de α a β (a veces sólo *aplicación*). A la expresión α se la llama el functor denotado generalmente por $\mathscr{F}$, y a la expresión β se la denomina argumento, denotado por $\mathscr{A}$.

En lo subsecuente, denotaremos por L a la teoría de tipos. Ahora demos la sintaxis:

3.1.2 **Definición**.
Consideremos el vocabulario:

- Para cada tipo a, un conjunto infinito $\mathrm{VAR_a}$ de variables del tipo a
- Los conectivos usuales: $\neg$ $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$
- Los cuantificadores: $\exists$ y $\forall$
- Los paréntesis (y)
- El símbolo de identidad $=$
- Para cada tipo a, un conjunto (puede ser vacío) de constantes de tipo a, $\mathrm{CON_a}$

(i) Si α es una constante o variable de tipo a de L, entonces α es una expresión de tipo a de L

(ii) Si α es una expresión de tipo $\langle a, b\rangle$ de L, y β es una expresión de tipo a de L, entonces $(\alpha(\beta))$ es una expresión de L

(iii) Si ϕ y ψ son expresiones de tipo t, entonces también lo son: $\neg\phi$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$ y $(\phi \leftrightarrow \psi)$

(iv) Si ϕ es una expresión de tipo t, y v es una variable de tipo a, entonces $\forall v\phi$ y $\exists v\phi$ son expresiones de tipo t.

(v) Si α y β son expresiones del mismo tipo, entonces $\alpha = \beta$ es una expresión de tipo t

(vi) Sólo las expresiones obtenidas usando las reglas (i)–(v) un número finito de veces son expresiones de L.

Al conjunto de expresiones bien formadas del tipo a de L lo denotaremos por WE_a, en esta notación, el conjunto de fórmulas es WE_t. Revisemos como representaríamos el incansable ejemplo de siempre, en este lenguaje. Tomemos el vocabulario A con conjunto de constantes, $\mathrm{CON}_e = \{alis, bill, cat, dina\}$, $\mathrm{CON}_{\langle e,t\rangle} = \{man, walk, woman, old, beverage, rock\text{-}star, dance, dancer, run\}$, $\mathrm{CON}_{\langle e,\langle e,t\rangle\rangle} = \{love, like, drink\}$, $\mathrm{CON}_{\langle e,\langle e,\langle e,t\rangle\rangle\rangle} = \{give, offer\}$, entonces las expresiones *alis*, *dancer(cat)*, *old(bill)*$\vee$*dance(cat)*, $\forall$*v(woman(x)*$\wedge$ *man(x))*, *like(bill)*, *like(bill)(alis)*, $\exists$*v (beverage(v)*$\wedge$*offer(cat)(bill)(v))* son expresiones bien formadas, aunque no todas del mismo tipo.

Una cosa peculiar de este lenguaje, es la manera en que se construyen las fórmulas, mediante la aplicación funcional; muy diferente de la lógica de predicados.

Revisemos algunos ejemplos de expresiones derivadas. Por mencionar alguno, $\langle e,t\rangle$ es un tipo que al aplicarse a una expresión de tipo e nos da algo del tipo t, o sea, una fórmula. Un ejemplo de una expresión de este tipo es un predicado de primer orden (como *dance*), pues al aplicarse a un individuo (como *cat*), nos da un valor de verdad (asociado a la fórmula *dance(cat)*, que representa la oración *Cat dances*). Otra manera de obtener una fórmula es aplicando algo del tipo $\langle\langle e,t\rangle,t\rangle$ a algo del tipo $\langle e,t\rangle$, en este caso podemos pensar en una propiedad (o predicado de segundo orden, como *temperature*), pues una propiedad aplicada a un predicado de primer orden (como *high*), nos da, de nueva cuenta, un valor de verdad (para nuestro ejemplo, asociado a la fórmula *temperature(high)*, que representa a la frase *high temperature*).

Por otra parte, un modificador de predicado (como *rápido*), puede verse como algo del tipo $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$, pues al aplicarse a un predicado de primer orden, nos da de nuevo un predicado de primer orden (por ejemplo, al predicado *correr* se le puede añadir el modificador *rápido*, resultando en el predicado *correr rápido*).

Un último caso que conviene revisar es el del tipo $\langle e,\langle e,t\rangle\rangle$, que al aplicarse a una expresión del tipo e, resulta en un predicado de primer orden. Un ejemplo de algo de este tipo, sería una relación entre dos individuos (como *love*), pues al aplicarse primero a un individuo (digamos *Mary*), resulta en un predicado de primer orden (en esta ocasión, *to love Mary*), que tiene tipo $\langle e,t\rangle$ por lo que puede aplicarse a otra entidad (digamos *John*), resultando ahora en un valor de verdad (asociado a *John loves Mary*). Es curioso notar que los individuos pueden entrar a formar parte de un predicado de primer orden. En nuestro ejemplo, el predicado resultaba ser *to love Mary*. En la lógica de primer orden para obtener un resultado similar, tendríamos

Tipo	Tipo de expresión	Ejemplo
e	Entidad o expresión individual	*Alis, Bill, Cat, Dina.*
$\langle e,t\rangle$	Predicado de primer orden	*Runs, dances, is a: rock-star/dancer.*
t	Fórmula	*Cat is a dancer and dances.*
$\langle t,t\rangle$	Modificador sentencial	*Does not...*
$\langle e,e\rangle$	Función de individuos a individuos	*His girlfriend.*
$\langle\langle e,t\rangle,\langle e,t\rangle\rangle$	Modificador de predicado de primer orden	*Slowly, beautifully.*
$\langle e,\langle e,t\rangle\rangle$	Relación binaria de primer orden	*Likes, loves, drinks.*
$\langle e,\langle e,\langle e,t\rangle\rangle\rangle$	Relación ternaria de primer orden	*Offers, gives.*
$\langle\langle e,t\rangle,t\rangle$	Relación unaria de segundo orden	*Is a color, has...personality.*
$\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle$	Relación binaria de orden dos	*...personality is better than...*
$\langle e,\langle\langle e,t\rangle,t\rangle\rangle$	Relación binaria entre individuos y predicados de primer orden	*He does that nicely.*
$\langle\langle\langle e,t\rangle,t\rangle,t\rangle$	Relación unaria de orden tres	*Is a second order predicate, human attributes.*

Cuadro 3.1: Tipos y sus Expresiones Equivalentes

que añadir una letra de predicado por cada individuo con el que quisiéramos construir una relación *to love* a ese individuo, en este lenguaje, basta aplicar la expresión *love* a cada individuo para obtener la relación *to love* con ese individuo.

La tabla (3.1) muestra algunos de los tipos que se pueden construir a partir de los tipos básicos e y t, y algunos ejemplos de qué cosa sería una expresión de ese tipo.

De aquí en adelante denotaremos a las constantes de individuo por letras minúsculas a, b, c,..., a las constantes de predicado de primer orden por mayúsculas A, B, C,..., y a las constantes de predicado de segundo orden por caligráficas $\mathscr{A}$, $\mathscr{B}$, $\mathscr{C}$,..., y lo mismo para las variables.

Ahora, para dar la semántica, necesitamos definir los dominios de interpretación que tiene asociado cada tipo, que denotaremos mediante $\mathbf{D}_{a,D}$, dado un tipo a y un dominio D.

3.1.3 Definición.
Sea D un conjunto no vacío.

(i) $\mathbf{D}_{e,D} = D$

(ii) $\mathbf{D}_{t,D} = \{0, 1\}$

(iii) $\mathbf{D}_{\langle a,b\rangle,D} = \mathbf{D}_{b,D}^{\mathbf{D}_{a,D}}$

Como generalmente el domino D está claro, podemos quitar tal subíndice y escribir solamente $\mathbf{D}_a$, cuando no haya riesgo de ambigüedad. Con esta definición, ya podemos decir como serán nuestros modelos para este lenguaje.

3.1.4 Definición.
Un modelo $\mathbf{M}$ para el lenguaje de la teoría de tipos L consiste de:

(i) Un conjunto dominio D no vacío.

(ii) Una familia de funciones de interpretación $\{I_i\}$, que a cada expresión constante de tipo a, le asigne un elemento de su respectivo dominio $\mathbf{D}_a$, es decir, para cada tipo a, I_a es una función, $I_a : CON_a \to \mathbf{D}_a$.

Para llevar a cabo la evaluación de una expresión, también necesitamos de una familia de funciones de asignación $\{g_i\}$, que a cada variable de tipo a, le asocie un elemento de su respectivo dominio $\mathbf{D}_a$, es decir, para cada tipo a, g_a es una función $g_a : VAR_a \to \mathbf{D}_a$. Para evitar una notación complicada, el subíndice de las interpretaciones y las asignaciones se puede quitar sin riesgo de confusión, sólo es necesario recordar que la interpretación asocia a cada constante de *cierto* tipo, un elemento de el dominio de *ese mismo* tipo. Lo mismo para las asignaciones y las variables.

Ahora sí, damos la semántica de la teoría de tipos. Cabe mencionar que en esta parte cambiaremos la notación que habíamos venido manejando. En vez de utilizar $V_{M,g}$ como un evaluación a valores de verdad, respecto del modelo M y la asignación g, usaremos $[\![\cdot]\!]_{M,g}$. Ahora es más claro que no todas las expresiones son fórmulas (o expresiones de tipo t), por lo que no toda expresión tendrá un valor en el dominio $\mathbf{D}_t = \{0, 1\}$. En general, dada una expresión del tipo a, la interpretación respecto de un modelo M y una asignación g, deberá ser un elemento de $\mathbf{D}_a$. Además, debemos asegurarnos de que el principio de composicionalidad se respete, es decir, que la interpretación de una expresión esté dada por la unión de las interpretaciones de las partes que la constituyen. La definición que funciona es como sigue:

3.1.5 Definición.
Dados un modelo M, una asignación g, y una expresión de L, definimos la interpretación de la expresión con respecto a M y g como:

(i) Si $\alpha \in CON_a$, entonces $[\![\alpha]\!]_{M,g} = I(\alpha)$
y si $\alpha \in VAR_a$, entonces $[\![\alpha]\!]_{M,g} = g(\alpha)$

(ii) Si $\alpha \in WE_{\langle a,b\rangle}$, $\beta \in WE_a$, entonces $[\![\alpha(\beta)]\!]_{M,g} = [\![\alpha]\!]_{M,g}([\![\beta]\!]_{M,g})$

(iii) Si $\phi, \psi \in WE_t$, entonces
$[\![\neg\phi]\!]_{M,g} = 1$ si y sólo si $[\![\phi]\!]_{M,g} = 0$
$[\![\phi \wedge \psi]\!]_{M,g} = 1$ si y sólo si $[\![\phi]\!]_{M,g} = 1$ y $[\![\psi]\!]_{M,g} = 1$
$[\![\phi \vee \psi]\!]_{M,g} = 1$ si y sólo si $[\![\phi]\!]_{M,g} = 1$ o $[\![\psi]\!]_{M,g} = 1$
$[\![\phi \rightarrow \psi]\!]_{M,g} = 0$ si y sólo si $[\![\phi]\!]_{M,g} = 1$ y $[\![\psi]\!]_{M,g} = 0$
$[\![\phi \leftrightarrow \psi]\!]_{M,g} = 1$ si y sólo si $[\![\phi]\!]_{M,g} = [\![\psi]\!]_{M,g}$

(iv) Si $\phi \in WE_t$, $v \in VAR_a$, entonces
$[\![\forall v\phi]\!]_{M,g} = 1$ si y sólo si para toda v-variante g' de g se tiene que $[\![\phi]\!]_{M,g'} = 1$
$[\![\exists v\phi]\!]_{M,g} = 1$ si y sólo si hay al menos una v-variante g' tal que $[\![\phi]\!]_{M,g'} = 1$

(v) Si $\alpha, \beta \in WE_a$, entonces $[\![\alpha = \beta]\!]_{M,g} = 1$ si y sólo si $[\![\alpha]\!]_{M,g} = [\![\beta]\!]_{M,g}$

De esta definición vemos que, tanto las constantes como las variables del tipo a, se interpretan de la misma forma, como elementos del dominio D_a, la diferencia radica en que el elemento que le corresponda estará designado mediante la función de interpretación I para las constantes, mientras que para las variables esta designación se hace por medio de la función de asignación g. Para las expresiones compuestas, la interpretación estará dada siguiendo el principio de composicionalidad, del que ya hemos hecho meción. Para las expresiones del tipo t, que resultan ser las fórmulas, los conectivos y cuantificadores tradicionales tienen la interpretación usual. La manera en que definimos los dominios de interpretación de los distintos tipos, nos dejan ver como es la función $[\![\cdot]\!]$ en algunos casos sencillos. Por ejemplo, supongamos que queremos ver cuál es la interpretación de la expresión *Lola sings*. En nuestro lenguaje, la expresión se representa mediante los símbolos $S(l)$. Con l el símbolo que representa a Lola, y S el símbolo que representa a cantar. Ahora, debemos ver qué cláusula aplica, en este caso es la número (ii), como S es constante de tipo $\langle e, t\rangle$ y l es constante de tipo e, $S(l)$ es de tipo t, de manera que al final debemos obtener una interpretación a manera de valor de verdad. La cláusula (ii) nos dice que debemos aplicar la interpretación de S a la interpretación de l, es decir, que $[\![S(l)]\!] = [\![S]\!]([\![l]\!])$. S y l son constantes, así que se interpretan, de acuerdo a (i), mediante la función de interpretación I, asociada a un modelo M (para l que es de tipo e, $I(l)$ es simplemente un elemento de $\mathbf{D}_e = D$, mientras que para S que es de tipo $\langle e, t\rangle$, $I(S) : D_e \rightarrow D_t$ o bien $I(S) : D \rightarrow \{0, 1\}$), asi que aplicar $I(S)(I(l))$

Tipo	Interpretación
e	Entidad
t	Valor de verdad
$\langle e,t\rangle$	Función de entidades a valores de verdad (o función característica de) conjunto de entidades
$\langle t,t\rangle$	Función de valores de verdad a valores de verdad
$\langle e,e\rangle$	Función de entidades a entidades
$\langle\langle e,t\rangle,\langle e,t\rangle\rangle$	Función de conjuntos de entidades a conjuntos de entidades
$\langle e,\langle e,t\rangle\rangle$	Función de entidades a conjuntos de entidades; conjunto pares ordenados de entidades
$\langle e,\langle e,\langle e,t\rangle\rangle\rangle$	Función de entidades a, funciones de entidades a conjuntos de entidades, conjunto de tercias ordenadas de entidades
$\langle\langle e,t\rangle,t\rangle$	(Función característica de) un conjunto de conjuntos de entidades
$\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle$	Función de conjuntos de entidades a, conjuntos de conjuntos de entidades, conjunto de parejas ordenadas de conjuntos
$\langle e,\langle\langle e,t\rangle,t\rangle\rangle$	Función de entidades a conjuntos de conjuntos de entidades
$\langle\langle\langle e,t\rangle,t\rangle,t\rangle$	(Función característica de) un conjunto de conjuntos de conjuntos de entidades

Cuadro 3.2: Tipos y sus Interpretaciones

resulta en aplicar $I(S)$ al individuo al que l denota. Si este valor resulta ser 1, entonces ese individuo tiene la propiedad expresada por S (en este caso cantar).

Debido a que $[\![\cdot]\!]$ le asigna un valor a todo tipo de expresiones, el lenguaje de la teoría de tipos nos permite dar una definición de equivalencia más amplia. Así, dos expresiones α y β, del mismo tipo, son equivalentes, si $[\![\alpha]\!]_{M,g} = [\![\beta]\!]_{M,g}$.

La tabla (3.2) muestra algunos tipos y sus interpretaciones.

Ahora debe ser claro que las oraciones (2.79) y (2.81), que carecían de una traducción en la lógica de predicados, ya pueden representarse en este lenguaje. Lo mismo ocurre con las oración (2.77).

(3.3) $(\mathscr{Q}(R))(c)$

sería la traducción de (2.79), donde $\mathscr{Q}$ representa *quickly*, R: *running* y c a *Carl*, y

(3.4) $(\mathscr{L}(A))(c)$

sería la traducción de (2.81), donde $\mathscr{L}$ representa a *large*, A: *ant* y c: *Christie*. Las traducciones de (2.75) y (2.77), ya las habíamos dado en el capítulo anterior, aunque entonces enfatizábamos que no pertenecían al lenguaje de la IFOL. En este lenguaje L, sólo la traducción de (2.77) pertenece al lenguaje, la otra expresión aun queda fuera de nustro alcance.

Sin embargo, a pesar de las ventajas que presenta este lenguaje, aun existen varias nociones que no quedan completamente plasmadas. Por ejemplo el siguiente argumento aun no queda garantizado en la semántica que dimos.

(3.5) *John runs quickly*

no implica que

(3.6) *John runs*

Una manera de solventar esta dificultad, es introduciendo los *postulados del significado*, como hizo Richard Montague en su trabajo original de PTQ. Por ahora conviene revisar un ejemplo de lo recién expuesto, exponer en él todas las categorías que podrían usarse en el lenguaje natural sería complicado, así que revisaremos sólo algunas de ellas, aunque trataremos que haya exponentes de las más notables.

Consideremos el vocabulario A, con conjunto de constantes, $\mathrm{CON}_e = \{alis, bill, cat, dina\}$, $\mathrm{CON}_{\langle e,t\rangle} = \{passive, hasty, high, low, man, walk, woman, old, beverage, rock\text{-}star, dance, dancer, run\}$, $\mathrm{CON}_{\langle e,\langle e,t\rangle\rangle} = \{love, like, drink\}$, $\mathrm{CON}_{\langle e,\langle e,\langle e,t\rangle\rangle\rangle} = \{give, offer\}$, $\mathrm{CON}_{\langle\langle e,t\rangle,t\rangle} = \{temperature, personality\}$, $\mathrm{CON}_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle} = \{quickly, slowly, enjoy\}$, $\mathrm{CON}_{\langle\langle\langle e,t\rangle,\langle e,t\rangle\rangle,\langle\langle e,t\rangle,\langle e,t\rangle\rangle\rangle} = \{very\}$. Y el modelo $M = (D, I)$, donde $D = \{a, b, c, d, e, f, g\}$, e I dada por

$I_e(alis) = a$ $\quad$ $I_e(cat)$=c

$I_e(bill) = b$ $\quad$ $I_e(dina)$=d

$I_{\langle e,t\rangle}(passive) = \{a, d\}$ $\quad$ $I_{\langle e,t\rangle}(old) = \{d, e\}$

$I_{\langle e,t\rangle}(hasty) = \{b, c\}$ $\quad$ $I_{\langle e,t\rangle}(beverage) = \{f, g\}$

$I_{\langle e,t\rangle}(high) = \{f\}$ $\quad$ $I_{\langle e,t\rangle}(rock\text{-}star) = \{b, e\}$

$I_{\langle e,t\rangle}(low) = \{g\}$ $\quad$ $I_{\langle e,t\rangle}(dance) = \emptyset$

$I_{\langle e,t\rangle}(man) = \{b, e\}$ $\quad$ $I_{\langle e,t\rangle}(dancer) = \{c\}$

$I_{\langle e,t\rangle}(walk) = \{a, d\}$ $\quad$ $I_{\langle e,t\rangle}(run) = \{b\}$

$I_{\langle e,t\rangle}(woman) = \{a, c, d\}$

$I_{\langle e,\langle e,t\rangle\rangle}(love) = \{(b, a), (c, b)\}$ $\quad$ $I_{\langle e,\langle e,t\rangle\rangle}(like) = \{(a, b), (a, e), (c, e), (d, e)\}$

$I_{\langle e,\langle e,t\rangle\rangle}(drink) = \emptyset$

$I_{\langle e,\langle e,\langle e,t\rangle\rangle\rangle}(offer) = \{(a, b, g)\}$ $\quad$ $I_{\langle e,\langle e,\langle e,t\rangle\rangle\rangle}(give) = \emptyset$

$I_{\langle\langle e,t\rangle,t\rangle}(temperature) = \{\{f\}, \{g\}\}$ $\quad$ $I_{\langle\langle e,t\rangle,t\rangle}(personality) = \{\{a, d\}, \{b, c\}\}$

$I_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}(quickly) = \{\{a, d\} \mapsto \{a\}\}$ $\quad$ $I_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}(slowly) = \{\{a, d\} \mapsto \{d\}, \{c\} \mapsto \{c\}\}$

$I_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}(enjoy) = \{\{a, d\} \mapsto \{b, c\}\}$

$$I_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle,\langle\langle e,t\rangle,\langle e,t\rangle\rangle\rangle}(very) = \{\{\{a,d\} \mapsto \{a\}\} \mapsto \{\{a,d\} \mapsto \emptyset\}\}$$

La notación $\{x_i\} \mapsto \{x_j\}$, debe leerse: la función característica f', cuyos valores son x_i, tiene como imagen a la función característica g', con valores x_j. Sería kilométrico poner todos los valores asociados a las funciones, así que aquellos símbolos que reciben un valor en un conjunto diferente de $\{0,1\}$ o D, los trataremos dando sólo algunos valores, y consideraremos que el resto de los valores van al conjunto vacío. Ahora tratemos de evaluar algunas oraciones compuestas

(3.7) *Alis walks.*

(3.8) *Alis walks quickly.*

(3.9) *Alis walks very quickly.*

(3.10) *Alis has passive personality.*

Representadas por las expresiones de L

(3.11) *walk(alis)*

(3.12) *(quickly(walk))(alis)*

(3.13) *((very(quickly))(walk))(alis)*

(3.14) *(personality(passive))(alis)*

Omitiremos los subíndices M y g, por estar sobreentendidos. Para ver el valor de $[\![walk(alis)]\!]$ debemos usar el inciso (ii) de la definición. Como $[\![walk]\!] = \{a,d\}$ (debemos ver este conjunto como una función característica), y como $[\![alis]\!] = a$, entonces hay que aplicar esta función al valor a, lo que nos da como resultado 1, de donde concluimos que en M la expresión es verdadera. Ahora para, $[\![(quickly(walk))(alis)]\!]$, usamos otra vez el inciso (ii) dos veces. Primero empezamos calculando el valor de $[\![quickly]\!]$, que es de acuerdo al modelo, la función que asigna a la función característica $\{a,d\}$ la función característica $\{a\}$, y a las demás funciones les asigna el conjunto vacío. Por el cálculo que hicimos en la expresión anterior, sabemos que $[\![walk]\!] = \{a,d\}$, por lo tanto, $[\![quickly]\!]([\![walk]\!]) = \{a\}$, por último, al aplicar esta nueva función al valor a, que es el valor de $[\![alis]\!]$, obtenemos el valor 1. De modo que la expresión es verdadera en M.
En el caso de $[\![((very(quickly))(walk))(alis)]\!]$ habrá que utilizar tres veces el inciso (ii). Iniciamos calculando $[\![very]\!]$ que resulta ser la función que asigna a la función $\{\{a,d\} \mapsto \{a\}\}$ la función $\{\{a,d\} \mapsto \emptyset\}$, de nueva cuenta recurrimos al cálculo de la expresión anterior donde vimos que $[\![quickly]\!] =$

$\{\{a,d\} \mapsto \{a\}\}$, por lo tanto, $[\![very]\!]([\![quickly]\!])$ es la función $\{\{a,d\} \mapsto \emptyset\}$. El siguiente paso es aplicar esta función a la función $\{a,d\}$, que es $[\![walk]\!]$, el resultado de hacer esto es la función característica vacía. Finalmente debemos aplicar la función vacía al elemento a, claramente el resultado de hacer esto es 0. Así, esta expresión es falsa en M.
Para terminar con las expresiones compuestas, veamos el caso de *(personality(passive))(alis)*. Primero, hay que buscar $[\![personality]\!]$, que es de acuerdo al modelo dado, la función que asigna a la función característica $\{a,d\}$, la función característica $\{b,c\}$, como $[\![passive]\!]$ es precisamente la función $\{a,d\}$, entonces el resultado de aplicar $[\![personality]\!]([\![passive]\!])$ es justamente la función característica $\{b,c\}$. Y ya que $[\![alis]\!] = a$, entonces el valor de verdad de la expresión es 0, o sea que es falsa en M.
Antes de pasar al cálculo de otro tipo de expresiones, conviene destacar la diversidad de tipos, que al aplicarse unos con otros, nos dan expresiones de WE_t. Como ejemplo, vimos *walk(alis)*, *(quickly(walk))(alis)*, *((very(quickly))(walk))(alis)* y *(personality(passive))(alis)*. Naturalmente, hay una gran diferencia entre aplicar un verbo a una entidad, aplicar un verbo adverbiado a una entidad, aplicar un doble adverbio a un verbo que se aplica a una entidad, y aplicar un adjetivo compuesto a una entidad. Pero con el lenguaje L ya somos capaces de hablar de todos estos tipos de relaciones. Antes mencionamos que en L los adverbios no implican el verbo (contrariamente a lo que pasa en el lenguaje natural), en el ejemplo es posible divisar este fenómeno, pues vimos que *(quickly(walk))(alis)* era verdadera, pero no lo era *((very(quickly))(walk))(alis)* (aunque aquí la consecuecia sí podría ser consistente con la intuición del lenguaje natural). Esto se debe al valor que las funciones asignen a otras funciones, y fácilmente podríamos haber hecho que el valor que tomara *quickly(walk)* fuera un conjunto del cual a sea miembro, mientras que a no fuera miembro de *walk*, lo cual causaría que el adverbio no implique al verbo, algo antinatural para verbos como *walk*. Pasemos a calcular expresiones más complicadas. A continuación ponemos una expresión de L y la oración que pretende modelar

(3.15) $(walk)(alis) \vee dance(cat)$

(3.16) *Alis walks or Cat dances.*

(3.17) $\exists x(dancer(x) \wedge \neg dance(x))$

(3.18) *There is some dancer that does not dance.*

(3.19) $\forall y(woman(y) \rightarrow rock\text{-}star(y))$

(3.20) *Every woman is a rock-star.*

(3.21) $\exists z(beverage(z) \wedge temperature(low(z)), offer(a,b,z))$

(3.22) *Alis offers a cold beverage to Bill.*

Para no complicar los cálculos haremos un poco más informalmente el proceso. Para (3.15) debemos calcular ambos disyuntos, el primero sabemos que es verdadero por las cuentas que hicimos antes, así que la expresión es verdadera. Si quisiéramos saber el valor de verdad del otro disyunto, *dances(cat)*, basta ver que c no pertenece al $\emptyset$, que es el valor de $[\![dance]\!]$, para saber que es falso. En cuanto a (3.17), hay que ver que exista un elemento de $[\![dancer]\!] = \{c\}$ tal que no sea elemento de $[\![dance]\!] = \emptyset$, lo cual se cumple, por lo tanto, el valor de (3.17) es verdadero. La expresión (3.19) nos hace buscar si todo elemento de $[\![woman]\!] = \{a, c, d\}$ es miembro de $[\![rock\text{-}star]\!] = \{b, e\}$, como esto no es así, la expresión es falsa. Finalmente para (3.21), debemos revisar si hay un elemento en $[\![beverage]\!] = \{f, g\}$, que sea además miembro de $[\![temperature(low)]\!] = \{g\}$, y que también aparezca en la tercera componente de alguna tercia en $[\![offer]\!] = \{(a, b, g)\}$. Al hacer esta revisión, vemos que la expresión es verdadera.

Todo va bien con este lenguaje. Salvo los pequeños detalles que ya mencionamos, parece ser bastante más expresivo que la FOL. Falta, sin embargo, añadirle operadores intensionales, para llevar a L al territorio de las lógicas intensionales. Pero antes de hacer eso, nos conviene introducir un operador especial, que recibe el nombre de operador lambda (λ), y que nos será de gran ayuda en lo posterior.

3.2. Cálculo Lambda

En esta parte añadimos el operador λ, proveniente del cálculo λ desarrollado por Church, para luego ver en qué nos puede servir. Después veremos una manera de llevar el lenguaje que hemos construido a Prolog.

Antes de dar las reglas para el operador λ, daremos una pequeña motivación para introducir un operador que resulta en ocasiones extraño y en cierto modo complicado.
Ya mencionamos que usando un símbolo del tipo $\langle e, \langle e, t\rangle\rangle$, y aplicándolo a una entidad, podemos construir una relación de orden uno de entidades. En esa ocasión dimos el ejemplo de *love(lola)* que resultaba en la relación de amar a *Lola*. Luego, al aplicar esta relación a otra entidad (*Ed*) creábamos la expresión de WE_t, *love(lola)(ed)*, que representa la oración *Ed loves Lola*. Nuestro interés ahora, es ver si podemos realizar el proceso inverso, es decir, dada una expresión ya aplicada a entidades o funciones, obtener una relación

entre elementos del tipo adecuado. En otras palabras, queremos abstraer sobre los parámetros que ya están instanciados en una expresión, para llegar a una expresión más general. De modo que si empleamos estas ideas en una oración como *Ed loves Lola* y abstraemos sobre *Ed*, llegamos a la oración *to love Lola*, mientras que si abstraemos sobre *Lola*, llegamos a la oración *Ed loves "someone"*. Como ambas oraciones ya sólo tienen un parámetro instanciado, si volvemos a abstraer sobre cualquiera de estas oraciones, obtenemos el predicado *love*. La pregunta importante es si en verdad este proceso nos será de alguna utilidad. Para tratar de contestarla, recordemos la forma del ejemplo planteado en el capítulo 1. Donde sugeríamos una manera de llevar el seguimiento de una charla, surgida a partir de un trámite burocrático en una dependencia. Supongamos que las primeras dos preguntas de la empleada de la dependencia hacia Cat, que desea realizar un viaje fuera de su país de origen, fueron: *Do you currently have any illness?* y *Have you ever done anything illegal?* Y que Cat las responde sinceramente como *no*. Una posible pregunta subsecuente es: *Are you married?* Si Cat responde de una manera específica: *I'm married to Bill*, la persona de la dependencia puede hacer el siguiente razonamiento:
Pregunta: *Is Cat married with someone?*
Fórmula: $\exists x$ *married(cat,x)*
Hecho: *Cat said she's married with Bill.*
Fórmula: *married(cat,bill)*
Deducción: *If she's married with Bill then she's married with someone.*
Las reglas de la FOL permiten dar este mismo razonamiento, en la forma de deducción: *married(cat,bill)* $\vdash \exists x$ *married(cat,x)*. Entonces la FOL nos permitiría llevar el seguimiento del discurso, actualizando el modelo conforme nueva información vaya entrando en él. Este mismo procedimiento puede aplicarse a situaciones diversas en diferentes contextos, con diferentes relaciones y diferentes individuos. En el caso de la relación *married*, no hay mucha diferencia sobre cuál individuo se abstraiga, se obtiene una relación simétrica. Algo distinto pasa si pensamos en una relación que no es necesariamente simétrica como *love*. Pues en un caso se obtiene la relación *to love someone* (al abstraer sobre *ed*), mientras que en otro se obtiene *someone loves* (haciéndolo sobre *lola*). El orden en el que se abstraiga es esencial en obtener uno u otro resultado. Sin embargo, no está clara la manera en la que la empleada de la dependencia llegó a abstraer sobre *Bill* y no sobre *Cat*. Pues si la relación hubiera sido más sensible (no simétrica, como *love*), la deducción no se hubiera efectuado exitosamente. Las reglas de deducción de la FOL no nos dicen nada al respecto de sobre cual variable anteponer el cuantificador existencial en el caso de relaciones de orden dos, sólo se nos indica que hacerlo en cualquiera de ellas es igualmente válido. Pero si tuviéramos un operador que

nos permitiera indicar el orden de abstracción, no sólo podríamos decir exactamente sobre cuál variable anteponer el cuantificador existencial, sino que además tendríamos un procedimiento para construir las fórmulas asociadas a las oraciones a partir de su representación en símbolos de L

3.2.1. Sintaxis y Semántica del Cálculo Lambda

Para pasar del lenguaje L, al lenguaje de la teoría de tipos junto con el operador lambda, que denotaremos en adelante L_λ. Primero debemos añadir una regla a la sintaxis de L. Que es como sigue.

3.2.1 Definición.

(vii) Si α es una expresión de tipo a, y v es una variable de tipo b, entonces $\lambda v.\alpha$ es una expresión de tipo $\langle b, a\rangle$

Ya hemos ejemplificado como generalizar expresiones particulares intuitivamente. Formalmente, el método para hacer esto es usar la regla sintáctica nueva (vii). Este proceso, de usar el operador λ se conoce como *abstracción funcional* o λ *abstracción*. Veamos algunos ejemplos: *Bill walks*, asociada a *walk(bill)*, tiene tipo t, usando una variable v, de tipo e, podemos crear la expresión $\lambda v.$*(walk(v))* que es de tipo $\langle e, t\rangle$, es decir, el mismo tipo que el símbolo *walk*, que es lo que pretendíamos obtener. Ahora, ya hemos hecho explícito varias veces que *Ed loves Lola* se representa por *(love(ed))(lola)*, tomando una variable x del tipo e, podemos crear la expresión $\lambda x.$*(love(x))(lola)*, o bien la expresión $\lambda x.$*(love(ed))(x)*, ambas de tipo $\langle e, t\rangle$. Pero la lambda abstracción se puede usar con variables de cualquier tipo (para fines prácticos, en tanto que aparezcan en la expresión cosas de ese tipo), así que podemos usar una variable X, de tipo $\langle e, t\rangle$ y abstraer en *walk(bill)*, obteniendo como resultado $\lambda X.(X(bill))$ que nos diría algo como *Bill has a property*, o abstrayendo con una variable V, del tipo $\langle e, \langle e, t\rangle\rangle$, en *(love(ed))(lola)* llegamos a $\lambda V.((V(ed))(lola))$, lo que nos dice que hay una relación binaria entre *ed* y *lola*.
Lo que sigue es dar la semántica del nuevo operador.

3.2.2 Definición.

(vii) Si $\alpha \in WE_a$ y $v \in VAR_b$, entonces $[\![\lambda v.\alpha]\!]_{M,g}$ es la función $h \in \mathbf{D}_a^{\mathbf{D}_b}$ tal que para toda $d \in \mathbf{D}_b : h(d) = [\![\alpha]\!]_{M,g^{[v/d]}}$

La interpretación parece más complicada de lo que es en realidad. La idea

es que la abstracción generaliza una expresión α de un tipo a, usando una variable v de otro tipo b, obteniendo así una expresión de tipo $\langle b, a\rangle$ (o sea, una expresión que recibe cosas del tipo b para dar como resultado cosas del tipo a). Entonces, para que la interpretación sea compatible con este proceso, debemos darle a cada elemento d del dominio sobre el que corre la variable v ($\mathbf{D}_b$), justo la interpretación que tendría la expresión α, si en su argumento estuviera dicho elemento d. Esto es precisamente lo que logramos al usar la asignación $g^{[v/d]}$ (fijar el argumento), y luego al calcular $[\![\alpha]\!]_{M,g[v/d]}$ (la interpretación de la expresión compuesta de α con argumento d). Este proceso debe hacerse para cada d, al juntar todos los casos es como se obtiene la función h. En pocas palabras, como vimos en el ejemplo de *walk(bill)*, quisiéramos que el símbolo *walk* y $\lambda v.walk(v)$ sean exactamente la misma función. La definición de la semántica garantiza precisamente esto. Esto quiere decir que tenemos dos notaciones para representar la misma expresión, por ejemplo, para decir que *Bill walks*, podemos escribir *walk(bill)*, o bien $\lambda x.walk(x)(bill)$, sin embargo, en vez de escribir $\lambda x.walk(x)(bill)$, queremos usar simplemente la expresión *walk(bill)*. Este proceso se puede formalizar para cualquier tipo de expresiones si hacemos las consideraciones pertinentes. En general, en vez de escribir una expresión como $\lambda x.\alpha(\beta)$, escribiremos simplemente ${}^{[x/\beta]}\alpha$, donde ${}^{[x/\beta]}$ significa reemplazar las ocurrencias libres de la variable x en α por β. Este proceso se llama λ-conversión, β-conversión, o λ-reducción. Pero al realizar este procedimiento debemos ser cuidadosos, pues existe un problema similar al que ocurre en la FOL, al sustituir una subfórmula, por otra equivalente, en una fórmula. Puede suceder que las variables de una de ellas queden accidentalmente acotadas por un cuantificador, lo que ocasiona que la fórmula cambie radicalmente. Afortunadamente, podemos reetiquetar las variables de una de ellas para que no haya problema al efectuar la λ-reducción lo que nos garantiza el resultado siguiente

3.2.3 Proposición (β-conversión).
Si toda variable que ocurre libre en γ es libre para v en β, entonces $\lambda v.\beta(\gamma)$ es equivalente a ${}^{[\gamma/v]}\beta$.

Donde, una variable v' es libre para v en una expresión β si y sólo si ninguna ocurrencia libre de v en β está bajo el alcance (acotada) de un cuantificador $\exists v'$, $\forall v'$, o un operador $\lambda v'$. Al proceso de reetiquetar las variables lo llamaremos α-conversión.
Ahora discutamos un poco acerca de la utilidad de introducir el operador λ. Primero, una limitante de la teoría de tipos que veníamos manejando es que no podemos emplear los conectivos lógicos como $\wedge$ o $\neg$ en cosas que no sean del tipo t. Esto se ve reflejado en el hecho de que ciertas oraciones no se pue-

dan expresar en el lenguaje de la teoría de tipos, por ejemplo, oraciones con un constituyente de predicado coordinado. Como un ejemplo consideremos:

(3.23) *Gambling and fighting is senseless.*

que no se puede expresar en el lenguaje de la teoría de tipos, porque la conjunción está al nivel de predicados de primer orden (o cosas de tipo $\langle e, t\rangle$), no en fórmulas (tipo t). Por lo tanto, carecemos de una traducción directa. Sin embargo, con la ayuda del operador λ podemos traducir la oración como:

(3.24) $\lambda X \neg \mathscr{S}(X)(\lambda x(G(x) \wedge F(x)))$

donde G, F y $\mathscr{S}$ se presentan como *gambling*, *fighting* y *sense* repectivamente. En particular, el predicado de segundo orden *senseless*, también requiere el uso del operador λ para su traducción, pues la negación no puede usarse al nivel de predicados de segundo orden. Por lo que, debemos transformarlo en algo del tipo t. Esto se logra convirtiendo el predicado *sense* en una fórmula (las cuales se pueden negar con $\neg$), negándola, y luego abstrayendo sobre una variable de tipo $\langle e, t\rangle$ en el término *sense* (de tipo $\langle\langle e, t\rangle, t\rangle$), para al final obtener $\lambda X \neg \mathscr{S}(X)$.

Hay que señalar que al usar la λ abstracción se debe tener cuidado en el orden en que se abstraen las variables. Regresemos al ejemplo, *Bill loves Lola*. Si abstraemos sobre Lola, obtenemos la expresión *Bill loves x*, que representa algo como *loving someone*. Si por el contrario abstraemos sobre Bill, obtenemos la expresión *x loves Lola*, que representa algo como *being loved by*.

Aun no hemos revisado expresiones cuantificadas, pero es claro que se pueden expresar en el lenguaje de la teoría de tipos aumentada con el operador lambda. Por ejemplo,

(3.25) *Every man breathes.*

(3.26) $\lambda Y \forall x(M(x) \rightarrow Y(x))(B)$

(3.27) *A boy plays.*

(3.28) $\lambda Y \exists x(B(x) \rightarrow Y(x))(P)$

No jugaremos mucho más con este lenguaje en cuanto a revisar cuantiosos ejemplos de oraciones. Mejor procederemos a construir una representación en Prolog del leguaje L_λ, y a formalizar el procedimiento para llevar una oración del lenguaje natural a una fórmula de la FOL, algo que nos será de gran utilidad en lo porvenir.

3.2.2. Lenguaje Natural, Gramática y Cálculo Lambda

Toca el turno de revisar el proceso de llevar un fragmento del lenguaje natural al lenguaje L_λ. El objetivo último será llevar un fragmento de lenguaje natural a la FOL de manera algorítmica. Para esto, necesitamos pasar por un proceso de traducción intermedio a L_λ. Ahora bien, la manera más sencilla de hacerlo será usando una gramática de frases estructuradas (PSG)[1]. Esto nos será especialmente fácil por el lenguaje de programación que elegimos para nuestras implementaciones: Prolog. Pues éste, ya viene con un mecanismo integrado que permite la declaración de PSGs, en el formato de *Gramáticas de Cláusulas Definidas*. Empero antes de revisar las declaraciones en Prolog de dichos mecanismos, conviene argüir en la razón para emplear PSGs como proceso de traducción entre el lenguaje natural y FOL.
Las PSGs surgieron a partir de los desarrollos de Noam Chomsky por formalizar los mecanismos que permiten generar las infinitas frases de un lenguaje natural, a partir de un número finito de símbolos y reglas. Debido a que la sintaxis de un lenguaje formal puede introducirse mediante una PSG, y a que las PSGs son muy intuitivas en la manera en que presentan las reglas de generación de frases, resulta ser una buena manera de intentar representar la sintaxis del lenguaje natural; sin mencionar que la terminología entre las PSGs y la gramática lingüística es similar, lo que también ayuda en parte. Así, usando reglas muy sencillas e intuitivas, de generación de frases, seremos capaces de generar una gran cantidad de oraciones, no tendremos que preocuparmos en demasía por la sintaxis, ni por implementar los llamados analizadores gramaticales (del inglés *parsers*), ni de los generadores de oraciones (porque Prolog y su mecanismo de gramáticas de cláusulas definidas harán todo este trabajo, algo que se complica al usar cualquier otro lenguaje de programación). Por otro lado, el por qué escoger la FOL como lenguaje al cual traducir las oraciones de una lengua, reside en el hecho de que es un lenguaje formal muy estudiado, motivo por el que hay desarrollos formales en métodos de deducción, que son implementables computacionalmente sin mucha dificultad[2]. Por lo tanto, podremos concentrarnos en la parte semántica del lenguaje natural, y resolver allí los conflictos que se presentan, de los cuales ya hemos hablado.

Árboles de análisis

Procedamos entonces en nuestra tarea. Algo que nos será de ayuda son

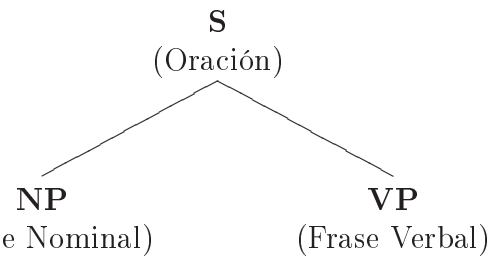
[1] Introdujimos en su momento las PSGs, con un ejemplo.

[2] Prolog mismo está desarrollado en base a uno de tales métodos, el método de *resolución binaria de Robinson*.

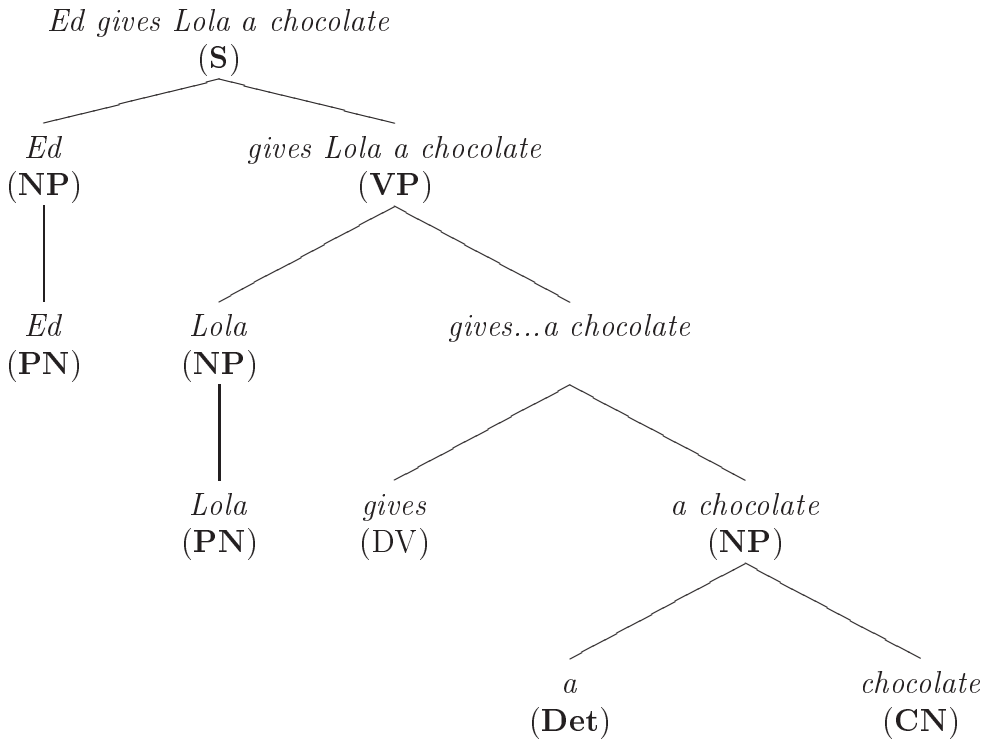
los conocidísimos, sobre todo en lingüística, árboles de análisis (*parse trees*). Estos árboles se obtienen descomponiendo una oración en los elementos de las categorías que la conforman, elementos que son dictados por las reglas de una PSG. Por ejemplo, si tenemos una regla de la forma

$$\mathbf{S} \longrightarrow \mathbf{NP\ VP}$$

que nos indica que algo de la categoría **S** se constituye por algo de la categoría **NP** seguido por algo de la categoría **VP**. Donde, **S** representa la categoría de las oraciones, **NP** la de las frases nominales, y **VP** la de las frases verbales.
Entonces podemos representar la regla mediante el árbol



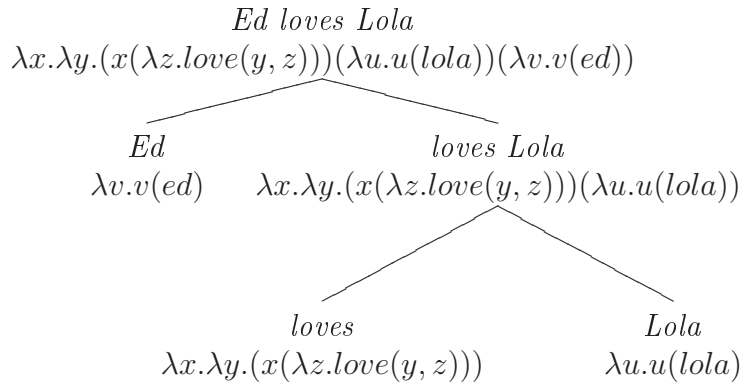
El árbol servirá para dos propósitos fundamentales: generación y análisis. En el primer caso, con el árbol podremos revisar si una oración es generada por la gramática: si al descomponer todos los nodos, los nodos hoja tienen solamente elementos terminales en cada una de ellas. Y en el segundo, el árbol nos permitirá construir, a partir de los elementos lexicales, una fórmula de la FOL asociada a la oración que se obtiene en el nodo raíz, de hecho en cada paso iremos obteniendo expresiones de L_λ, pero sólo en el nodo raíz se tendrá una expresión de WE_t.
Evidentemente si una regla puede representarse mediante un árbol, asimismo una oración generada por la PSG puede representarse por un árbol. Basta continuar este proceso de descomposición en cada nodo categoría hasta llegar a los símbolos terminales. Por ejemplo, la oración *Ed gives Lola a chocolate*, quedaría representada por el árbol que sigue, donde además de las categorías mencionadas arriba, tenemos **PN** para los nombres propios, **CN** para los sustantivos comunes, **Det** para los determinadores, y DV representa al subconjunto de la categoría **V** (de verbos), que contiene a los verbos ditransitivos



El árbol, además, muestra el proceso composicional de cómo se va formando la oración. Notemos que por ejemplo, el determinador *a* junto con el sustantivo *chocolate* forman, al juntarse, la frase nominal *a chocolate*. Esto desde la perspectiva de la PSG. Pero desde una perspectiva diferente, a saber la de L_λ, podemos ver al símbolo *a* como algo del tipo determinador, tal que al aplicarse a algo del tipo sustantivo común, nos da algo del tipo de frase nominal. Así que una asociación adecuada de los tipos de L_λ con las categorías sintácticas del lenguaje, nos permitiría construir la traducción en L_λ, de las oraciones generadas por una PSG que usemos para modelar al lenguaje natural. Si ahora añadimos la información adecuada en los símbolos terminales (que no son otra cosa que los elementos lexicales), podemos obtener un proceso para obtener una fórmula, asociada a una oración, que se va construyendo a la par del árbol de análisis.
Veamos un ejemplo. Traduzcamos la oración *Ed loves Lola*. Para hacerlo requerimos una PSG, usemos las reglas

$\mathbf{S} \longrightarrow \mathbf{NP}\ \mathbf{VP}$
$\mathbf{NP} \longrightarrow \mathbf{PN}$
$\mathbf{VP} \longrightarrow \mathit{TV}\ \mathbf{NP}$
$\mathbf{PN} \longrightarrow \mathit{Ed} \mid \mathit{Lola}$
$\mathit{TV} \longrightarrow \mathit{loves}$

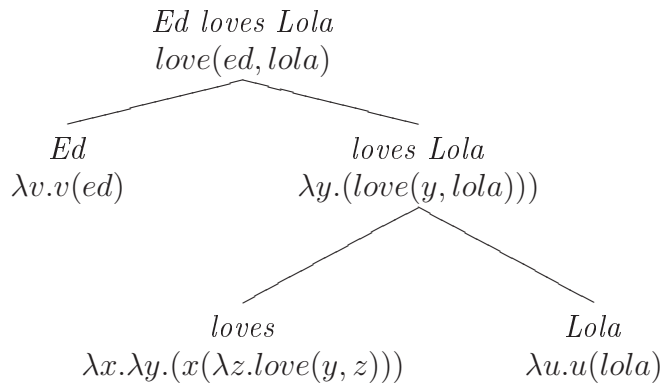
Esta PSG genera la oración *Ed loves Lola*, como muestra el árbol siguiente



Por otro lado, si asociamos a los elementos lexicales con las expresiones

Ed --→ $\lambda x.x(ed)$
Lola --→ $\lambda x.x(lola)$
loves --→ $\lambda x.\lambda y.(x(\lambda z.love(y, z)))$

y aplicamos las expresiones en los nodos correspondientes, siguiendo el orden en el que aparecen en el árbol, vamos obteniendo expresiones que pueden aplicarse y reducirse hasta obtener una fórmula de la FOL, que es precisamente la que uno asociaría a una oración como la que estamos ejemplificando. El árbol anterior mostraba las expresiones sin reducir. Contrástese con el árbol siguiente



que presenta las expresiones ya reducidas con las aplicaciones necesarias de α-conversión y λ-reducción.
Ahora estamos listos para implementar estas ideas en Prolog.

3.2.3. Representando el Lambda Cálculo en Prolog

Hasta ahora hemos venido manejando los ejemplos de expresiones de L y L_λ, en una sola "línea". Pero pronto veremos que si queremos usar la lambda abstracción con el fin de algoritmizar el proceso de llevar una oración en lenguaje natural a su representación en la FOL, deberemos hacer explícitos los varios pasos intermedios que surgen de aplicaciones sucesivas de la λ-reducción y la α-conversión. Una ventaja que tenemos al usar las reglas de sintaxis de L_λ, es que podemos reciclar todas las representaciones que dimos para la FOL en Prolog. Sólo nos falta ver la manera de representar las aplicaciones y las lambda abstracciones.
Para representar las primeras usaremos un término `app/2`, para representar a las aplicaciones, donde ambos argumentos son expresiones bien formadas que pueden aplicarse, esto es, el primer argumento deberá tener tipo $\langle a, b\rangle$ y el segundo tipo a, para que el resultado sea una expresión del tipo b.
Para las abstracciones usaremos el término `lam/2`, donde el primer argumento corresponde a la variable sobre la que queremos abstraer y el segundo es una expresión en la que se quiere realizar la abstracción.
De manera que la expresión $(love(ed))(lola)$ quedaría representada como `app(app(love,ed),lola).`
mientras que la expresión $\lambda x.walk(x)$, quedaría
`lam(X,app(walk,X).`
Ahora necesitamos de programas que realicen tanto la α-conversión como la λ-reducción.

α-conversión

Tratemos primero la implementación de un programa que se encargue de reetiquetar las variables en una expresión. La idea es llevar una lista de las variables que van apareciendo en una expresión, e irlas reetiquetando (renombrando) si éstas se encuentran acotadas. Esto lo lograremos con una lista de substituciones, que será una lista que contenga un término `sub(X,Y)`, por cada variable x que deba substituirse, y una diferencia de listas de variables libres. Por supuesto, estas listas deberán estar vacías al inicio del procedimiento. Manejaremos entonces dos predicados: `alphaConvert/2` y `alphaConvert/4`. El primero se encargará de inicializar el proceso por nosotros cada vez que necesitemos hacer la α-conversión, mientras que el segundo será el predicado que lleve a cabo la conversión, descomponiendo la expresión en partes más pequeñas, y llevando una lista con aquellas variables que deban ser reetiquetadas. Así que lo que necesitamos es definir cláusulas que cubran todas las posibles expresiones que podemos encontrar. La más sencilla es el caso de una

variable, pues si encontramos que la variable es parte de la lista de substituciones, simplemente realizamos la substitución, mientras que si la variable es libre, se añade a la lista variables libres

```
alphaConvert(X,SubList,Free1-Free2,Y):-
        var(X),
        (
           member(sub(Z,Y),SubList),
           X==Z, !,
           Free2=Free1
        ;
           Y=X,
           Free2=[X|Free1]
        ).
```

Para las expresiones que acotan variables como es el caso de los cuantificadores y el operador lambda, debemos realizar la substitución correspondiente de x por y

```
alphaConvert(Expression,SubList,Free1-Free2,some(Y,Formula2)):-
   nonvar(Expression),
   Expression = some(X,Formula1),
   alphaConvert(Formula1,[sub(X,Y)|SubList],Free1-Free2,Formula2).

alphaConvert(Expression,SubList,Free1-Free2,some(Y,Formula2)):-
   nonvar(Expression),
   Expression = all(X,Formula1),
   alphaConvert(Formula1,[sub(X,Y)|SubList],Free1-Free2,Formula2).

alphaConvert(Expression,SubList,Free1-Free2,some(Y,Formula2)):-
   nonvar(Expression),
   Expression = lam(X,Formula1),
   alphaConvert(Formula1,[sub(X,Y)|SubList],Free1-Free2,Formula2).
```

El resto de las expresiones se descomponen en subexpresiones usando el predicado `compose/3`, luego, usando el predicado `alphaConvertList/4`, se llevan a cabo las α-conversiones en cada subexpresión (esto es lo que hace este predicado), y finalmente se reconstruye la expresión original (pero con variables ya renombradas) usando nuevamente el predicado `compose/3`.

```
alphaConvert(Exp1,SubList,Free1-Free2,Exp2):-
        nonvar(Exp1),
        \+ Exp1 = some(_,_),
        \+ Exp1 = all(_,_),
```

```
        \+ Exp1 = lam(_,_),
        compose(Exp1,Symbol,Args1),
        alphaConvertList(Args1,SubList,Free1-Free2,Args2),
        compose(Exp2,Symbol,Args2).

alphaConvertList([],_,Free1-Free2,[]).

alphaConvertList([Exp1|Rest1],SubList,Free1-Free2,[Exp2|Rest2]):-
        alphaConvert(Exp1,SubList,Free1-Free2,Exp2),
        alphaConvertList(Rest1,SubList,Free1-Free2,Rest2).
```

Como puede apreciarse, el predicado `alphaConvertList/4` se encarga de α-convertir cada elemento de la lista que se le pase en su primer argumento, devolviendo una lista con elementos ya α-convertidos en su cuarto argumento.

λ-reducción

Para implementar el predicado que lleve a cabo las λ-reducciones, de expresiones como *λu.dance(u)(λv.v(lola))*, dando como resultado la expresión *dance(lola)*, usaremos el predicado `lamReduction/3`, que recibirá como primer argumento la expresión a reducir, regresará en su segundo argumento la expresión reducida y en el tercer argumento llevará el seguimiento de la variable que debe sustituirse en el momento adecuado a manera de lista. Es decir, cada vez que encuentre una expresión, resultado de la aplicación de un functor con un argumento ($\mathscr{F}(\mathscr{A})$), agregará a la cabeza de la lista dicho argumento ($\mathscr{A}$), quitará la aplicación y el argumento de la expresión quedándose solamente con la expresión functor ($\mathscr{F}$) y substituirá el argumento que tiene guardado únicamente hasta que encuentre en la expresión functor una expresión del tipo abstracción. Este proceso nos permite controlar que las substituciones de los argumentos no se hagan desordenadamente, sino que entren en el momento que les toca, pues si en algún instante del procedimiento se halla otra aplicación, el argumento de esta nueva aplicación entrará en la cabeza de la lista, relegando el argumento que haya sido guardado antes, y obteniendo prioridad en la substitución. De manera que en la primera abstracción que se presente, se substituirá este último argumento y no el primero que entró en la lista, que es justamente el procedimiento que seguimos al reducir algo como *[(λy.λx.like(x,y))(lola)][ed]*. Notemos que aquí no podemos reducir directamente la expresión sustituyendo *ed*, pues la expresión functor es en sí misma una aplicación de *λy.λx.like(x,y)* con *lola* como argumento, así que uno posterga la reducción con el argumento *ed* y procede

primero a reducir la expresión interior, substituyendo *lola* en la abstracción más exterior en la expresión, en este caso la que acota a la variable y, por lo que al reducir obtenemos la expresión $\lambda x.like(x,lola)$. Ahora la expresión ya no se puede reducir más, así que uno recuerda que faltó reducir usando el argumento *ed*, como el functor es una abstracción ya podemos hacerlo, substituyendo la ocurrencia de x, y obteniendo finalmente la expresión *like(ed,lola)*. Por supuesto, lo que entre en la "lista de espera" no necesita ser un individuo, puede ser una expresión compleja como $\lambda u.u(lola)$, $\lambda v.dance(v)$, etc. Además de esto, debemos tomar en cuenta que la expresión puede no ser ni una aplicación ni una abstracción. En esos casos deberemos descomponer en las subexpresiones que conforman la expresión global, con el fin de revisar si en las subexpresiones hay aplicaciones o abstracciones, y en caso necesario realizar los pasos para reducir. Una última cosa que deberemos de cercioranos es de hacer uso del predicado que ya construimos para la α-conversión, con el fin de evitar acotamientos indeseables de variables. Esto lo haremos justo antes de pasarle la expresión al predicado de la λ-reducción.

```
lamReduction(Expression,Result,Stack):-
   nonvar(Expression),
   Expression = app(Functor,Argument),
   nonvar(Functor),
   alphaConvert(Functor,ConvertedFunctor),
   lamReduction(ConvertedFunctor,Result,[Argument|Stack]).

lamReduction(Expression,Result,[X|Stack]):-
   nonvar(Expression),
   Expression = lam(X,Formula),
   lamReduction(ConvertedFor,Result,Stack).

lamReduction(Expresssion,Result,[]):-
   nonvar(Expression),
   \+ (Expression = app(X,_),nonvar(X)),
   compose(Expression,Functor,SubExpressions),
   lamReductionList(SubExpressions,ResultSubExpressions),
   compose(Result,Functor,ResultSubExpressions).
```

Por último, queremos una versión de la reducción en la que no tengamos que preocuparnos de la lista, sino solamente de la expresión de entrada y la de salida, que es la expresión ya reducida.

```
lamReduction(X,Y):-
   lamReduction(X,Y,[]).
```

Este será el predicado que usaremos en las aplicaciones.

NL --→ FOL

Ahora sí, estamos equipados con las herramientas necesarias para desarrollar un proceso que nos permita pasar de una oración del lenguaje natural (denotado en lo que sigue por NL) a la FOL. Haremos esto con un pequeño ejemplo, donde podremos ver en acción a los verbos, sustantivos, nombres propios y los determinadores "un(a)" y "cada".
Lo primero que necesitamos es establecer el vocabulario A para construir $L(A)$. Aquí es donde entran las PSGs que nos ayudarán a ligar los símbolos de NL con FOL. Supongamos que queremos traducir las oraciones

Bill sings.
Ed likes lola.
Cat offers Dan a beverage.
Every woman loves a rock-star.

Eso quiere decir que A deberá contener los símbolos para los nombres propios *Ed*, *Lola*, *Cat* y *Bill*, los símbolos para los verbos *offers*, *loves*, *likes*, y los símbolos para los sustantivos *woman* y *rock-star*. Empezamos extrayendo los elementos lexicales de las oraciones y dándoles una representación vía Lambda Cálculo, los nombres propios se representarán

Ed--→ λx.x(ed)
Lola--→ λx.x(lola)

los sustantivos comunes y los verbos intransitivos reciben una representación similar[3]

rock-star--→ λy.rock-star(y)
sings--→ λy.sing(y)

para los verbos transitivos usamos

likes--→ λu.λv.(uλx.like(v,x))

los verbos ditransitivos se representan por

offers--→ λx.λy.λz.(xλw.(yλu.offers(z,w,u)))

[3]Recordemos que en FOL usábamos símbolos de relación de aridad uno tanto para sustantivos comunes como para verbos intransitivos. Esto es una consecuencia de esa decisión.

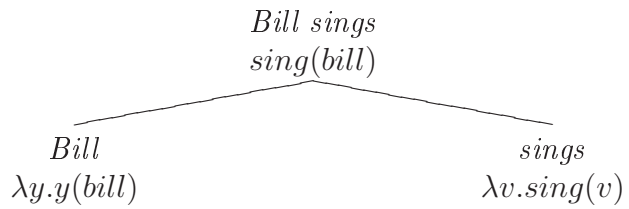
los determinadores se representan mediante

$every \dashrightarrow \lambda u.\lambda v.\forall(u(x) \rightarrow v(x))$
$a \dashrightarrow \lambda u.\lambda v.\exists(u(x) \wedge v(x))$

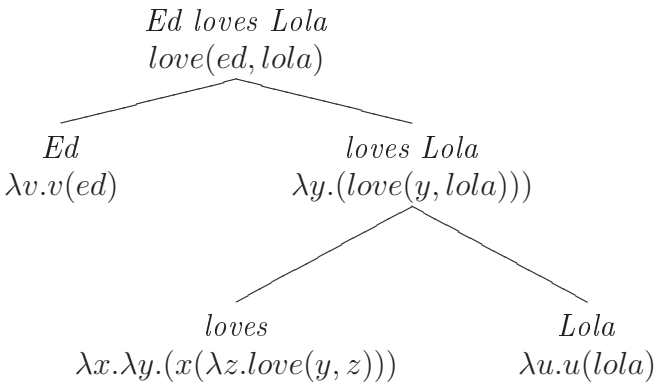
Ahora necesitamos una manera de representar las oraciones con una PSG, usemos la siguiente

S	$\longrightarrow$	**NP VP**
NP	$\longrightarrow$	PN
NP	$\longrightarrow$	**Det** Noun
VP	$\longrightarrow$	IV
VP	$\longrightarrow$	TV **NP**
VP	$\longrightarrow$	IO DO
IO	$\longrightarrow$	DV **NP**
DO	$\longrightarrow$	**Det** Noun
IV	$\longrightarrow$	*sings*
TV	$\longrightarrow$	*likes*|*loves*
DT	$\longrightarrow$	*offers*
Noun	$\longrightarrow$	*woman*|*rock-star*
PN	$\longrightarrow$	*Ed*|*Lola*|*Cat*|*Bill*|*Dan*
Det	$\longrightarrow$	*a*|*every*

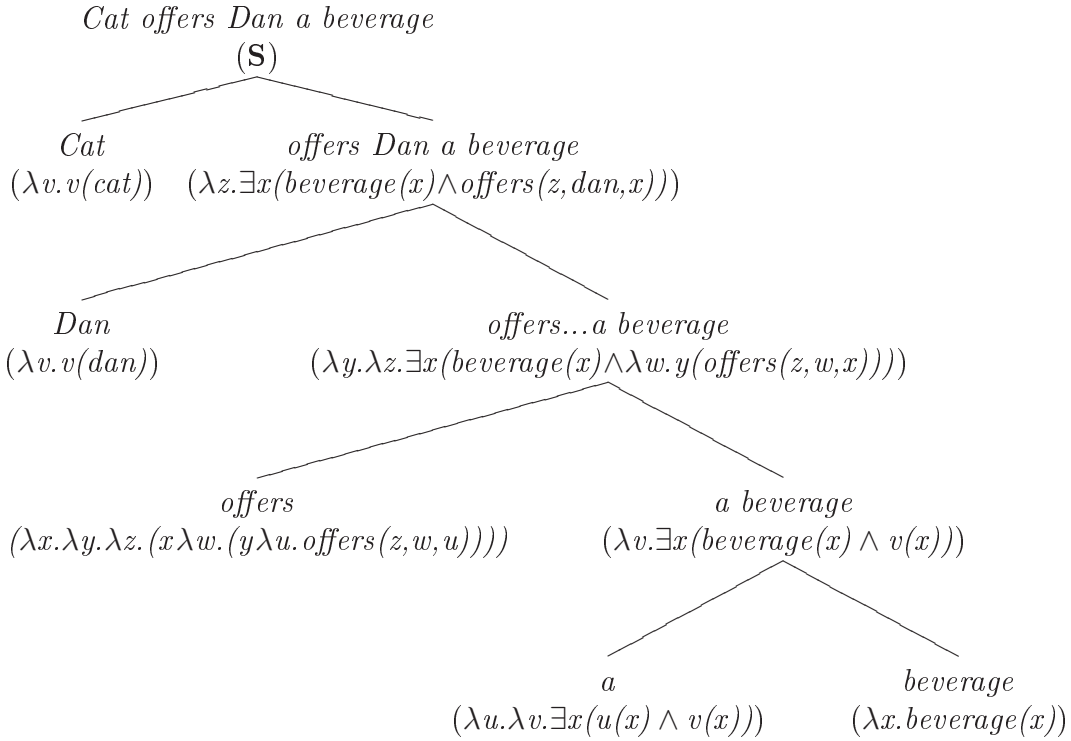
Esta PSG genera las oraciones que pretendemos modelar por lo que asocia el árbol de análisis que necesitamos para el proceso de traducción. Por ejemplo para la oración *Bill sings* tenemos



Para la oración obtenemos*Ed loves Lola*



Y para la oración *Cat offers Dan a beverage* se tiene



Donde cada nodo muestra la expresión obtenida al aplicar sus hijos y reducirlos con la λ-reducción y en caso necesario habiendo usado la α-conversión. Este mismo procedimiento lo podemos llevar a Prolog muy fácilmente, transcribiendo la PSG en una gramática de cláusulas definidas (DCG), mecanismo que ya viene integrado en Prolog. Por ejemplo, la PSG que usamos quedaría

```
s(app(NP,VP))-> np(NP), vp(VP).
np(PN)-> pn(PN).
np(app(Det,Noun))-> det(Det), noun(Noun).
```

```
vp(IV)-> iv(IV).
vp(app(TV,NP))-> tv(TV), np(NP).
vp(app(app(DV,IO),DO))-> io(DV,IO), do(DO).
do(app(Det,Noun))-> det(Det), noun(Noun).
io(DV,IO)-> dv(DV), np(IO).
```

El argumento extra, en el lado izquierdo de la regla, nos sirve para pasar la representación en lambda cálculo a los predicados de la derecha, de manera que aquí es donde indicamos en que forma deben aplicarse el functor y el argumento para conseguir una expresión del tipo del lado izquierdo de la regla. La información lexical la podemos representar también como reglas de una DCG, en la forma siguiente

```
noun(lam(X,beverage(X)))--> [beverage].
noun(lam(X,woman(X)))--> [woman].
noun(lam(X,rock_star(X)))--> [rock,star].
iv(lam(X,sing(X)))--> [sings].
pn(lam(X,app(X,ed)))-> [ed].
pn(lam(X,app(X,lola)))--> [lola].
det(lam(Y,lam(Z,all(X,imp(app(Y,X),app(Z,X))))))--> [every].
tv(lam(X,lam(Y,app(X,lam(Z,love(Y,Z))))))--> [loves].
dtv(lam(X,lam(Y,lam(Z,app(X,lam(W,app(Y,lam(U,offer(Z,W,U)))))))))-->
[offers].
```

como podemos ver, simplemente estamos dando la codificación en Prolog usando la representación que ya revisamos de lambda cálculo, junto con reglas de una DCG. Y eso es todo. Con esta representación ya podemos obtener las fórmulas deseadas. Entonces, si realizamos la consulta

```
?- s(Z,[ed,loves,lola],[]),lamReduction(Z,Res).
```

Obtenemos como respuesta

```
Z = app(lam(_G27,app(_G27,ed)),app(lam(_G36,lam(_G39,app(_G36,lam
(_G45,love(_G39,_G45))))),lam(_G51,app(_G51,lola)))),
Res = love(ed,lola) .
```

La primera consulta nos da la representación en lambda cálculo de la oración *Ed loves Lola*, que se le da al predicado `s/3`, en forma de una diferencia de listas. El resultado de hacer esto instancia a la variable `Z` con esta representación. La segunda consulta, del predicado `lamReduction/2` liga a la variable `Res` con la expresión ya λ-reducida que es la fórmula de FOL que buscábamos.

Capítulo 4

Teoría de Tipos Intensional

Aquí presentamos la teoría de tipos aumentada con los operadores modales que ya hemos visto anteriormente, el operador lambda, y un par de nuevos operadores que nos ayudan para construir la base de lo que será el lenguaje de PTQ (llamado así por las siglas en inglés de Proper Treatment of Quantifiers), originalmente creado por Montague para analizar el lenguaje natural, desde una aproximación de *teoría de modelos*. Veremos crecer nuestra capacidad expresiva y cómo se solucionan muchas de las inconveniencias que la teoría de tipos de la sección previa presentaba.

4.1. Construcciones Intensionales

Empezamos con una definición de los tipos.

4.1.1 Definición.
El conjunto de los tipos T, de la teoría de tipos intensional, se define por:

(i) $e, t \in \mathrm{T}$

(ii) Si $a, b \in \mathrm{T}$, entonces $\langle a, b\rangle \in \mathrm{T}$

(iii) Si $a \in \mathrm{T}$, entonces $\langle s, a\rangle \in \mathrm{T}$

Aquí de nueva cuenta e y t son nuestros tipos básicos, con e refiriéndose a las entidades y t a las expresiones que toman un valor de verdad, la cláusula (ii) para tipos compuestos también es igual que antes, lo nuevo es la cláusula (iii), que nos permite formar un nuevo tipo ($\langle s, a\rangle$) por cada tipo a. Notemos que el propósito de s, es crear un nuevo tipo compuesto a, a partir de un tipo a, y que no es en sí mismo un tipo. Las expresiones del tipo $\langle s, a\rangle$ veremos que serán las que representen la intensionalidad del lenguaje, jugando el papel

de funciones de mundos posibles a entidades de tipo a. En este lenguaje, consideraremos el siguiente vocabulario:

para cada tipo a un conjunto infinito VAR_a de variables del tipo a

los conectivos $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$

los cuantificadores $\forall$ y $\exists$

el símbolo de identidad $=$

los operadores $\Box$, $\Diamond$, $^\wedge$ y $^\vee$

los paréntesis (y)

para cada tipo a, un conjunto (puede ser vacío), CON_a de constantes de tipo a

La sintaxis es prácticamente la misma de la teoría de tipos de la sección anterior.

4.1.2 Definición.
Denotemos por WE_a al conjunto de expresiones bien formadas del tipo a.

(i) Si $\alpha \in VAR_a$ o $\alpha \in CON_a$, entonces $\alpha \in WE_a$

(ii) Si $\alpha \in WE_{\langle a,b\rangle}$ y $\beta \in WE_a$, entonces $(\alpha(\beta)) \in WE_b$

(iii) Si $\phi, \psi \in WE_t$, entonces $\neg\phi, \phi \wedge \psi, \phi \vee \psi, \phi \rightarrow \psi, \phi \leftrightarrow \psi \in WE_t$

(iv) Si $\phi \in WE_t$ y $v \in VAR_a$, entonces $\forall v\phi, \exists v\phi \in WE_t$

(v) Si $\alpha, \beta \in WE_a$, entonces $(\alpha = \beta) \in WE_t$

(vi) Si $\alpha \in WE_a$ y $v \in VAR_b$, entonces $\lambda v.\alpha \in WE_{\langle b,a\rangle}$

(vii) Si $\phi \in WE_t$, entonces $\Box\phi, \Diamond\phi \in WE_t$

(viii) Si $\alpha \in WE_a$, entonces $^\wedge\alpha \in WE_{\langle s,a\rangle}$

(ix) Si $\alpha \in WE_{\langle s,a\rangle}$, entonces $^\vee\alpha \in WE_a$

(x) Sólo son expresiones bien formadas las que se obtengan por medio del uso de las reglas (i)–(ix) un número finito de veces.

Las cláusulas (i)–(vi) son idénticas a las que ya teníamos del lenguaje de la teoría de tipos no intensional. El inciso (vii) introduce los operadores modales como lo habíamos hecho para los primeros dos lenguajes intensionales revisados. Notemos que los operadores $\Box$ y $\Diamond$ sólo se pueden aplicar a cosas del tipo t (o fórmulas). Los operadores que pueden resultarnos extraños son los presentados en los incisos (viii) y (ix). La idea detrás de éstos es la siguiente.

Hasta ahora hemos venido cargando con una complicación al tratar los contextos opacos, y las construcciones basadas en ellos. Vimos que el principio de extensionalidad no se cumplía en estos casos, y que había que ver más allá de una identidad para permitir la sustitución de "iguales por iguales'". Resulta que una manera de garantizar el principio de extensionalidad, en contextos opacos, es definiendo lo que llamaremos más tarde *intensión* de una expresión. De manera que, si dos expresiones tienen la misma *intensión*, se pueden sustituir, respetando la interpretación de la expresión en la que se sustituya.

El operador $^\wedge$, llamado "gorro" ó "arriba", será el encargado de obtener dicha *intensión* de una expresión por nosotros. Pero también, debemos tener una forma de obtener la expresión (o *extensión* como llamaremos después), de la que procede una *intensión*. El operador $^\vee$, llamado "taza" o "abajo" será el que esté a cargo de dicha tarea.

Ya desde este punto, podemos ver como será el dominio de interpretación de algo del tipo $\langle s, a\rangle$, que si nos damos cuenta, es el involucrado en la sintaxis del operador $^\wedge$. La interpretación de un tipo intensional $\langle s, a\rangle$ deberá ser una función que vaya de mundos posibles a elementos del dominio de interpretación del tipo a

Ahora necesitamos ver los dominios de interpretación que tendrán las expresiones de diferentes tipos.

4.1.3 Definición.
Sean D un conjunto no vacío y W un conjunto no vacío de mundos posibles. El dominio de interpretación del tipo a denotado $\mathbf{D}_{a,D,W}$ se define por:

(i) $\mathbf{D}_{e,D,W} = D$

(ii) $\mathbf{D}_{t,D,W} = \{0, 1\}$

(iii) $\mathbf{D}_{\langle a,b\rangle,D,W} = \mathbf{D}_{b,D,W}^{\mathbf{D}_{a,D,W}}$

(iv) $\mathbf{D}_{\langle s,a\rangle,D,W} = \mathbf{D}_{a,D,W}^{W}$

En esta definición hemos establecido lo que ya veníamos discutiendo. De nuevo, omitiremos el subíndice D, e incluso el W, cuando no haya riesgo de

Tipo	Interpretación
$\langle s, e\rangle$	Función de mundos a entidades: concepto individual
$\langle s, t\rangle$	Función de mundos a valores de verdad: proposición
$\langle s, \langle e, t\rangle\rangle$	Función de mundos a conjuntos de entidades: propiedad de primer orden
$\langle s, \langle e, \langle e, t\rangle\rangle\rangle$	Función de mundos a funciones de entidades a conjuntos de entidades: relación binaria de primer orden
$\langle s, \langle\langle e, t\rangle, t\rangle\rangle$	Función de mundos a conjuntos de conjuntos de entidades
$\langle\langle s, e\rangle, t\rangle$	Función de conceptos individuales a valores de verdad: (función característica de) conjunto de conceptos individuales
$\langle s, \langle\langle s, e\rangle, t\rangle\rangle$	Función de mundos a conjuntos de conceptos individuales: propiedad de conceptos individuales
$\langle\langle s, t\rangle, t\rangle$	Función de proposiciones a valores de verdad: (función característica de) conjunto de proposiciones

Cuadro 4.1: Tipos Intensionales e Interpretaciones

confusión. Veamos un ejemplo de como funciona. El dominio de interpretación del tipo $\langle s, t\rangle$, de acuerdo al inciso (iv), es $\mathbf{D}_{\langle s,t\rangle} = \mathbf{D}_t^W = \{0, 1\}^W$, que es el conjunto de funciones de W en los valores de verdad, entonces, una expresión del tipo $\langle s, t\rangle$ se refiere a una función de mundos posibles a valores de verdad. Llamaremos a dichas expresiones, *proposiciones*. Así, $\mathbf{D}_{\langle s,t\rangle}$ es el conjunto de proposiciones. Ahora bien, $\mathbf{D}_{\langle s,\langle e,t\rangle\rangle} = \mathbf{D}_{\langle e,t\rangle}^W = (\{0, 1\}^D)^W$ es el conjunto de funciones de mundos posibles a (funciones características de) conjuntos de individuos. De modo que una expresión de tipo $\langle s, \langle e, t\rangle\rangle$ se refiere a una función de mundos posibles a conjuntos de individuos. Ya que los conjuntos de individuos sirven como interpretaciones de los predicados, y los predicados toman diferentes conjuntos en cada mundo posible, esta función puede tomarse como la intensión del predicado. Llamaremos a tal intensión, *propiedad*.

La tabla (4.1) muestra las interpretaciones de los tipos nuevos $\langle s, a\rangle$, para algunos casos y los nombres con los que nos referiremos a expresiones de ese tipo. La tabla debe completarse junto con la tabla (3.2) para obtener una descripción completa de los tipos.

Con esto dicho, podemos decir lo que será un modelo de la teoría de tipos intensional.

4.1.4 Definición.
Un modelo $\mathbf{M}$ de la teoría de tipos intensional consiste de:

(i) Un conjunto dominio no vacío D

(ii) Un conjunto no vacío W de mundos posibles

(iii) Una familia de funciones de interpretación $\{I_i\}$ tal que a cada constante del tipo a le asigna una función que va del conjunto de mundos posibles W a el dominio de interpretación $\mathbf{D}_a$

También necesitamos de una familia de funciones de asignación $\{g_i\}$, tal que a cada variable del tipo a le asigna un elemento del conjunto $\mathbf{D}_a$.

Nuevamente podemos pensar en una sola función de interpretación I, la cual consiste, para cada tipo a, en la función I_a. Lo mismo en las asignaciones. Así, podemos omitir el subíndice. Observemos que en esta definición no hemos hecho mención de la relación de accesibilidad, esto se debe a que consideraremos la relación universal que contiene las parejas de todos los mundos posibles. Otra cosa que conviene decir, es que la función de interpretación I, ahora no trata a las constantes como designadores rígidos (aunque un tratamiento de este tipo todavía es posible, si tomamos una función constante de W en $\mathbf{D}_a$). En cambio, les da distintos valores en cada mundo posible. De manera que si α es una expresión del tipo a, y $w \in W$, entonces $I(\alpha)(w) \in \mathbf{D}_a$, llamaremos a este elemento la *referencia* de α en w. Ahora sí, podemos dar la semántica de este nuevo lenguaje. Dado un modelo $\mathbf{M}$, un mundo w, una expresión α y una asignación g, denotaremos por $[\![\alpha]\!]_{M,w,g}$ a la *extensión* o *referencia* de α en w dados $\mathbf{M}$ y g.

4.1.5 Definición.
Sea $\mathbf{M}$ un modelo de la teoría de tipos intensional, $w \in W$ y g una asignación.

(i) Si $\alpha \in CON_a$, entonces $[\![\alpha]\!]_{M,w,g} = I(\alpha)(w)$
Si $\alpha \in VAR_a$, entonces $[\![\alpha]\!]_{M,w,g} = g(\alpha)$

(ii) Si $\alpha \in WE_{\langle a,b\rangle}$ y $\beta \in WE_a$, entonces $[\![\alpha(\beta)]\!]_{M,w,g} = [\![\alpha]\!]_{M,w,g}([\![\beta]\!]_{M,w,g})$

(iii) Si $\phi, \psi \in WE_t$, entonces
$[\![\neg\phi]\!]_{M,w,g} = 1$ si y sólo si $[\![\phi]\!]_{M,w,g} = 0$
$[\![\phi \wedge \psi]\!]_{M,w,g} = 1$ si y sólo si $[\![\phi]\!]_{M,w,g} = 1$ y $[\![\psi]\!]_{M,w,g} = 1$
$[\![\phi \vee \psi]\!]_{M,w,g} = 1$ si y sólo si $[\![\phi]\!]_{M,w,g} = 1$ o $[\![\psi]\!]_{M,w,g} = 1$
$[\![\phi \rightarrow \psi]\!]_{M,w,g} = 0$ si sólo si $[\![\phi]\!]_{M,w,g} = 1$ y $[\![\psi]\!]_{M,w,g} = 0$
$[\![\phi \leftrightarrow \psi]\!]_{M,w,g} = 1$ si y sólo si $[\![\phi]\!]_{M,w,g} = [\![\psi]\!]_{M,w,g}$

(iv) Si $\phi \in WE_t$ y $v \in VAR_a$, entonces
$[\![\forall v\phi]\!]_{M,w,g} = 1$ si y sólo si para todo $d \in \mathbf{D}_a : [\![\phi]\!]_{M,w,g^{[v/d]}} = 1$
$[\![\exists v\phi]\!]_{M,w,g} = 1$ si y sólo si hay un $d \in \mathbf{D}_a : [\![\phi]\!]_{M,w,g^{[v/d]}} = 1$

(v) Si $\alpha, \beta \in WE_a$, entonces
$[\![\alpha = \beta]\!]_{M,w,g} = 1$ si y sólo si $[\![\alpha]\!]_{M,w,g} = [\![\beta]\!]_{M,w,g}$

(vi) Si $\alpha \in WE_a$ y $v \in VAR_b$, entonces $[\![\lambda v \alpha]\!]_{M,w,g}$ es la función h en $\mathbf{D}_a$ tal que para todo $d \in \mathbf{D}_b$ se tiene que $h(d) = [\![\alpha]\!]_{M,w,g^{[v/d]}}$

(vii) Si $\phi \in WE_t$, entonces
$[\![\Box\phi]\!]_{M,w,g} = 1$ si y sólo si para todo $w' \in W$ se tiene que $[\![\phi]\!]_{M,w',g} = 1$
$[\![\Diamond\phi]\!]_{M,w,g} = 1$ si y sólo si hay un $w' \in W$ tal que $[\![\phi]\!]_{M,w,g} = 1$

(viii) Si $\alpha \in WE_a$, entonces $[\![^\wedge\alpha]\!]_{M,w,g}$ es la función h en $\mathbf{D}_a^W$ tal que para todo $w' \in W$ se tiene que $h(w') = [\![\alpha]\!]_{M,w,g}$

(ix) Si $\alpha \in WE_{\langle s,a\rangle}$, entonces $[\![^\vee\alpha]\!]_{M,w,g} = [\![\alpha]\!]_{M,w,g}(w)$

Las referencias que pueden parecer complicadas son las relativas a los incisos (viii) y (ix). Habíamos dicho que una expresión de tipo $\langle s, a\rangle$ tiene como dominio de interpretación una función de mundos posibles en cosas del dominio $\mathbf{D}_a$. Si α tiene tipo a, entonces $^\wedge\alpha$ tiene tipo $\langle s, a\rangle$, de modo que le corresponde alguna de esas funciones. Para saber exactamente cuál le toca, tomamos un mundo $w' \in W, w' \neq w$, y vemos que extensión tiene allí α, juntando todos estos valores es como obtenemos h, la extensión de $^\wedge\alpha$. Por ejemplo, si R es un predicado de primer orden (de tipo $\langle e, t\rangle$), entonces $^\wedge R$ es de tipo $\langle s, \langle e, t\rangle\rangle$ por lo que su extensión debe ser una función de mundos posibles a cosas del tipo $\langle e, t\rangle$, qué mejor que tomar lo que el predicado R significa en cada mundo w, esto es justo lo que nos dice la definición que será su valor, pues h es la función que en cada mundo w toma como valor $[\![R]\!]_{M,w,g}$.

En cuanto a las expresiones del tipo $\langle s, a\rangle$. Sus extensiones, como acabamos de ver, son funciones de mundos posibles en cosas del tipo a, ya que $^\vee$ aplicado a una expresión de este tipo nos deja algo de tipo a, no se nos ocurre algo mejor, que asignarle lo que le toca bajo la función, que es la extensión de la expresión original, en ese mundo. Un ejemplo, si l es una constante de tipo $\langle s, e\rangle$, entonces su extensión $[\![l]\!]_{M,w,g}$ es una función de mundos posibles a individuos, por lo tanto, podemos aplicarle a w dicha función, es decir, $[\![l]\!](w)$, esto es justo lo que dice la definición que es la extensión de la expresión $^\vee l$.

Hasta ahora hemos lidiado solamente con las extensiones de las expresiones. Es hora de introducir la definición de la intensión de una expresión. Lo que yace detrás de esta definición es tratar de resolver el problema de la sustitución en contextos opacos, el problema radica en que un individuo puede tener una referencia en un contexto y otra distinta en otro. Si tuviéramos una manera de revisar la referencia de una expresión en cada contexto, podríamos ver en qué mundos dos expresiones α y β son iguales y en cuáles

no. Pero esto es precisamente lo que la función de mundos posibles en extensiones hace. Denotaremos a la intensión de una expresión α, relativa a un modelo M y una asignación g como $Int_{M,g}(\alpha)$.

4.1.6 Definición.
Si $\alpha \in WE_a$, entonces $Int_{M,g}(\alpha)$ es la función $h \in \mathbf{D}_a^W$ tal que para todo $w' \in W$ se tiene que $h(w') = [\![\alpha]\!]_{M,w',g}$.

Con esta definición, podemos revisar las referencias de una expresión en cada mundo. Ahora, si dos expresiones α y β tienen la misma intensión, significa que las funciones $Int_{M,g}(\alpha) = Int_{M,g}(\beta)$ son iguales, en otras palabras, significa que las referencias de α y β son las mismas en cada mundo, es decir, que $[\![\alpha]\!]_{M,w,g} = [\![\beta]\!]_{M,w,g}$ en todo mundo $(Int_{M,g}(\alpha)(w) = Int_{M,g}(\beta)(w))$. El recíproco, no se cumple en general, y era esto precisamente lo que ocasionaba problemas con el principio de extensionalidad, en contextos opacos. Además, debemos notar que la semántica del operador $^\wedge$ deliberadamente se dio de manera que coincida con la intensión de una expresión, esto es, que $[\![^\wedge\alpha]\!]_{M,w,g} = Int_{M,g}(\alpha)$. Todo esto, con el fin de que el operador "gorro" nos permita obtener la intensión de una expresión α en términos de la extensión de la expresión $^\wedge\alpha$.

Finalmente podemos enunciar un teorema, que establece el resultado que queríamos obtener, desde que revisamos los problemas que presentan las construcciones intensionales, en contextos opacos, con el principio de Leibniz.

4.1.7 Teorema.
Sean α y β expresiones de la teoría intensional de tipos. Entonces
$^\wedge\alpha =^\wedge \beta \models \gamma = [\beta/\alpha]\gamma$

4.2. Interacción de $^\wedge$, $^\vee$ y Lambda Conversión

El hecho de que la teoría intensional de tipos tiene expresiones cuya extensión varía de mundo en mundo, nos forza a revisar de nueva cuenta bajo qué condiciones la lambda conversión satisface que $\lambda v\beta(\gamma) = [\gamma/v]\beta$.

Antes, conviene hacer notar que $^\vee{}^\wedge\alpha$ es equivalente a α para toda expresión α de la teoría intensional de tipos (denotémosla L para abreviar). En cuanto a $^\wedge{}^\vee\alpha$ se refiere, se necesita dar cierta condición para garantizar un resultado similar. Pero para dar esta condición, primero necesitamos revisar algunas cosas.

4.2.1 Definición.

Definimos el conjunto de las expresiones intensionalmente cerradas, ICE^{L}, como el mínimo subconjunto de WE^{L} tal que:

(i) Si $v \in VAR_a$, entonces $v \in ICE^{\mathrm{L}}$

(ii) Si $\alpha \in WE_a$, entonces $^\wedge\alpha \in ICE^{\mathrm{L}}$

(iii) Si $\phi \in WE_t$, entonces $\Box\phi, \Diamond\phi \in ICE^{\mathrm{L}}$

(iv) Si α está construida de elementos de ICE^{L} usando solamente conectivos, cuantificadores, y el operador λ, entonces $\alpha \in ICE^{\mathrm{L}}$

4.2.2 Teorema.
Si $\alpha \in ICE^{\mathrm{L}}$, entonces $[\![\alpha]\!]_{M,w,g} = [\![\alpha]\!]_{M,w',g}$ para todo M, w, w'.

Una consecuencia de (4.2) es el siguiente.

4.2.3 Teorema.
Si $\alpha \in ICE^{\mathrm{L}}$, entonces $^\wedge{}^\vee\alpha$ es equivalente a α.

Bajo las mismas suposiciones de (4.2) es que podemos formular el siguiente resultado.

4.2.4 Teorema.
Supongamos que $\beta \in \mathrm{L}, \gamma \in ICE^{\mathrm{L}}$ y v una variable tales que:

(i) todas las variables libres en γ son libres para v en β.

(ii) $\gamma \in ICE^{\mathrm{L}}$, o bien no hay ocurrencias libres de v en β que estén bajo el alcance de $\Box, \Diamond$ o $^\wedge$.

Entonces, $\lambda v\beta(\gamma)$ es equivalente a $[\gamma/v]\beta$.

En estos momentos, la teoría de tipos intensional que hemos venido definiendo, es prácticamente la misma de PTQ, solamente falta incluir los operadores temporales (que ya hemos revisado con anterioridad en el capítulo 2), y los postulados del significado de los que desafortunadamente no podremos hablar con propiedad, aunque procuraremos mencionar algo al respecto en la siguiente sección.

Al considerar los operadores temporales P, H, F y G, la única complicación que se presenta es el cambio que debe realizarse es en los dominios de interpretación. En realidad, sólo la cláusula (iv) sufre una pequeña modificación, convirtiéndose en

4.2.5 Definición.
(v) $\mathbf{D}_{\langle s,a\rangle,D,W,T} = \mathbf{D}^{W,T}_{a,D,W,T}$

Ya que ahora debemos considerar tanto mundos posibles como momentos en el tiempo, tenemos que usar el conjunto $W \times T$, para representar mundos posibles en un momento dado. Las expresiones del tipo intensional $\langle s, a\rangle$ se convierten así en funciones de índices (que representan mundos en momentos) a cosas del tipo a. Por supuesto que la función de interpretacón I también debe adaptarse. Ahora asignará a las constantes α del tipo a un elemento del dominio $D_a^{W\times T}$ en cada índice $(w, t) \in W \times T$. Esta redefinición afecta a las cláusulas

4.2.6 Definición.
Bajo las hipótesis de (4.1). Las siguientes reemplazan a los respectivos incisos.

(vii) Si $\phi \in WE_t$, entonces
$[\![\Box\phi]\!]_{M,(w,t),g} = 1$ si y sólo si para todo $w' \in W$ y $t' \in T$, $[\![\phi]\!]_{M,(w',t'),g} = 1$
$[\![\Diamond\phi]\!]_{M,(w,t),g} = 1$ si y sólo si hay algún $w' \in W$ y $t' \in T$, tal que $[\![\phi]\!]_{M,(w',t'),g} = 1$

(viii) Si $\alpha \in WE_a$, entonces $[\![^\wedge\alpha]\!]_{M,(w,t),g}$ es la función $h \in \mathbf{D}_a^{W\times T}$ tal que para todo $w' \in W$ y todo $t' \in T$, se tiene que $h((w', t')) = [\![\alpha]\!]_{M,(w',t'),g}$

(ix) Si $\alpha \in WE_{\langle s,a\rangle}$, entonces $[\![^\wedge\alpha]\!]_{M,(w,t),g} = [\![\alpha]\!]_{M,(w,t),g}((w, t))$

Lo primero a notar en la definición es que $\Box$ significa ahora necesidad en todo tiempo posible. Además hay que añadir las cláusulas respectivas para P, H, F y G.

4.2.7 Definición.
(x) Si $\alpha \in WE_t$, entonces
$[\![H\phi]\!]_{M,(w,t),g} = 1$ si y sólo si para todo $w' \in W$ y todo $t' \in T, t' < t$, $[\![\phi]\!]_{M,(w',t'),g} = 1$
$[\![P\phi]\!]_{M,(w,t),g} = 1$ si y sólo si hay algún $w' \in W$ y $t' \in T, t' < t$, tal que $[\![\phi]\!]_{M,(w',t'),g} = 1$
$[\![G\phi]\!]_{M,(w,t),g} = 1$ si y sólo si para todo $w' \in W$ y todo $t' \in T, t' > t$, $[\![\phi]\!]_{M,(w',t'),g} = 1$
$[\![F\phi]\!]_{M,(w,t),g} = 1$ si y sólo si hay algún $w' \in W$ y $t' \in T, t' > t$, tal que $[\![\phi]\!]_{M,(w',t'),g} = 1$

4.3. Gramática de Montague

En esta parte revisaremos el sistema PTQ de Montague, no revisaremos propiamente la gramática de Montague como tal, sino solamente el lenguaje que solventa las dificultades surgidas a partir de las ambigüedades de alcance, por ese motivo nos hemos referido al sistema de Montague como PTQ. Veremos algunas aplicaciones que tiene este sistema en el modelado del lenguaje natural. Platicaremos sobre la manera que Montague introdujo para contrarrestar los efectos de la dependencia lexical en el significado de ciertas oraciones y revisaremos una aproximación más moderna a este respecto, basada en lo que se ha denominado en los últimos días como *representaciones semánticas subespecificadas* (*semantic underspecified representations*).

4.3.1. PTQ

Como ya mencionamos, el sistema PTQ de Montague es el lenguaje de la teoría intensional de tipos con los operadores lambda, gorro y taza. Un problema que Montague logró resolver al usar PTQ fue, entre otros, el que se presenta con las ambigüedades de alcance. Ambigüedades que aparecen al tratar oraciones como *every woman loves a rock-star* y *every priest does not smoke*. En el capítulo 1 mencionamos que la primera oración puede representarse por las fórmulas (1.3), y (1.4) que repetimos abajo en ese mismo orden

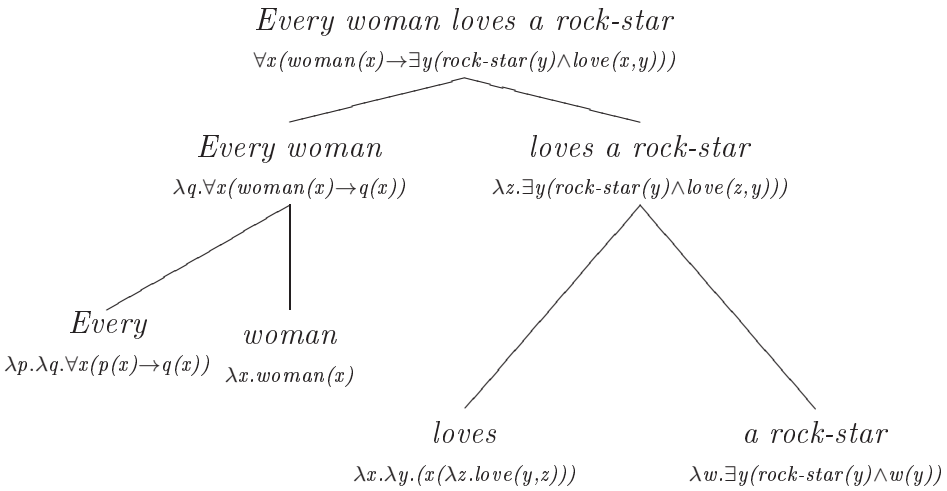
(1.3) $\forall x(woman(x) \rightarrow \exists y(rock\text{-}star(y) \wedge love(x, y)))$

(1.4) $\exists y(rock\text{-}star(y) \wedge \forall x(woman(x) \rightarrow love(x, y)))$

sintácticamente no hay problema con las oraciones, ambas son fórmulas de la FOL. El problema se presenta al interpretarlas semánticamente. Pero antes de adentrarnos en más detalles necesitamos hacer ciertos acuerdos. La estrategia a seguir es similar a la que tomamos en (3.2.3), donde definíamos un conjunto de categorías básicas y derivadas, que empatábamos con algunas categorías lingüísticas, dábamos representaciones en lambda cálculo a los elementos lexicales, y usando una PSG construíamos árboles de análisis para las oraciones. Usaremos esas mismas representaciones, junto con la PSG allí expuesta. Hasta ahora, hemos venido manejando las PSGs pensando en que el lado derecho de la regla, compone o conforma al lado izquierdo de la regla con las categorías que allí aparecen por simple concatenación. Pero puede decirnos aun más. Algo como la operación mediante la cual se obtiene la expresión final. En este caso pensaremos que una regla de la forma $\mathbf{S} \longrightarrow \mathbf{NP\ VP}$ nos dice adicionalmente, que la categoría **S** se obtiene al aplicar algo de la categoría **NP** como functor, con algo de la categoría **VP** como

argumento. Ahora sí, revisemos el ejemplo de la oración *every woman loves a rock-star*. Tenemos que asociar a los elementos lexicales con expresiones de L_λ, hagámoslo de la misma manera en que lo hicimos antes
rock-star $\dashrightarrow \lambda x.rock\text{-}star(x)$
woman $\dashrightarrow \lambda x.woman(x)$
love $\dashrightarrow \lambda x.\lambda y.(x(\lambda z.love(y,z)))$
every $\dashrightarrow \lambda u.\lambda v.\forall(u(x)\rightarrow v(x))$
a $\dashrightarrow \lambda u.\lambda v.\exists(u(x)\wedge v(x))$

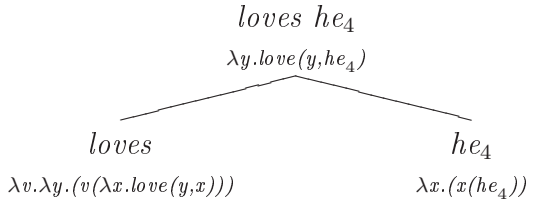
Entonces usando la regla **NP** $\longrightarrow$ **Det Noun** podemos construir la parte de la oración que corresponde a *every woman*, identificando a *woman* como **Noun** y a *every* con **Det**. Para la parte de *loves a rock-star*, usamos primero la regla **NP** $\longrightarrow$ **Det Noun** nuevamente, identificando ahora *rock-star* como **Noun** y *a* como **Det**, y después la regla **VP** $\longrightarrow$ TV **NP** identificando *love* como TV y *a rock-star* como **NP**. Finalmente la regla **S** $\longrightarrow$ **NP VP** nos permite formar la oración final. El proceso realiza aplicaciones respectivas en cada paso, si además, realizamos las lambda reducciones en cada paso, obtenemos el árbol de análisis



No parece que haya alguna inconveniencia en este proceder, sin embargo, debido a nuestra dependencia de la PSG para generar las fórmulas, esta forma es la única manera de construir una representación para la oración *every woman loves a rock-star*. Esto quiere decir que hemos perdido la lectura en la cual sólo una estrella del rock es amada por cada mujer. Empero esta imposibilidad es también una consecuencia de haberle asignado las expresiones lambda anteriores a los cuantificadores, *every* y *a*. Podríamos haberlo evitado al cambiar las lambda expresiones a dichos elementos lexicales. Si

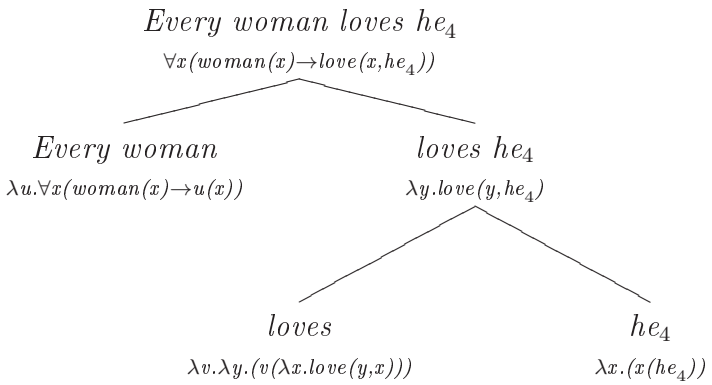
hacemos esto, imponiendo expresiones lambda adecuadas, para obtener la otra lectura de la oración, perdemos esta lectura, donde cada mujer ama a (posiblemente) diferentes estrellas del rock. La cuestión central yace en por qué es tan importante que cada elemento lexical tenga una representación en el lambda cálculo. Tal vez podríamos quedarnos con aquellas expresiones de L_λ que correspondan a fórmulas de la FOL y que además sean representaciones de oraciones en el NL. Pero ¿cómo discriminar aquellas expresiones que representan oraciones del NL de las que no? Este es el problema que se ha tratado desde que el programa de Montague surgió. La manera aquí esbozada es una de muchas formas ideadas al respecto. En nuestro caso, la idea de que sea la sintaxis, mediante la PSG, la que dirija el proceso, evita que no haya expresiones que correspondan a oraciones agramaticales en el NL. Pero pareciera ser que perdemos muchas expresiones que también corresponden a oraciones del NL. Abrir un poco la sintaxis suena a una buena idea, esto podría hacerse añadiendo más reglas[1], pero entonces se pierde la intuición que provee la PSG. Es aquí donde aparecen las novedosas ideas de las representaciones subespecificadas, de las que hablaremos en breve.

Antes, platiquemos fugazmente la idea de Montague (en otras palabras: el corazón de PTQ). La meta es obtener la fórmula $\exists y(\textit{rock-star(y)} \wedge \forall x(\textit{woman}(x) \rightarrow \textit{love}(x, y)))$. En lugar de usar *a rock-star*, usemos un pronombre he_4 que consideraremos ser de categoría **NP**, y cuya representación será la de un nombre propio $\lambda x.x(he_4)$. Entonces podemos al usar la regla **VP** $\longrightarrow$ TV **NP** para crear el árbol

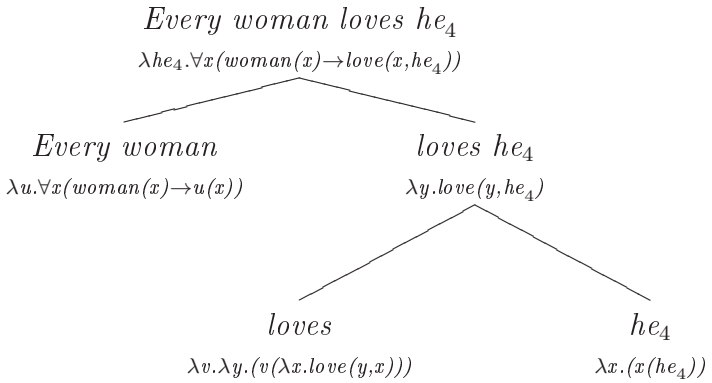


Usando ahora la regla, **S** $\longrightarrow$ **NP VP**, podemos continuar el árbol como sigue

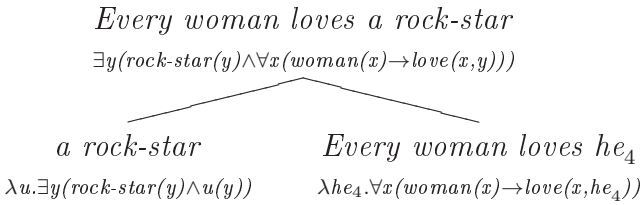
[1]Precisamente esto es lo que hizo Montague originalmente.



Podemos ver que todo va como en el proceso que ya habíamos hecho antes para construir la representación de la oración, excepto que no hemos introducido aun el cuantificador existencial. Esta es la ventaja de la que nos ha provisto la variable de pronombre he_4: retrasar la introducción del cuantificador existencial. Introduzcamos ahora una nueva regla que nos permite anteponer un operador lambda a la oración obtenida, abstrayendo respecto de la variable pronombre. Entonces obtenemos



Si ahora nos permitimos substituir la variable de pronombre he_4 por una **NP** como *a rock-star* y pensamos que la representación de la oración final se obtiene al aplicar (y reducir) la representación de la **NP** *a rock-star* con la representación de *Every woman loves* he_4 obtenemos



Este último paso nos da la lectura de la oración buscada. El único precio

que tuvimos que pagar fue introcucir una variable "rara" de pronombre, y la introducción de un par de reglas sintácticas para tratar con estas nuevas construcciones, que contengan a las variables de pronombre. Y eso es todo, esta es la novedosa idea de Montague para tratar los cuantificadores formalmente. Puede verse que un proceso similar puede tratar también a la oración *every priest does not smoke*, oración que presenta ambigüedad por la introducción de la negación en dos partes distintas: a nivel sentencial (donde se niega toda la fórmula), o bien a nivel del sustantivo *priest* (donde se niega a aquellos dentro de la relación *priest*).

En su momento, esta original idea permitió un crecimiento en la metodología de la traduccion de NL a un lenguaje lógico. Y en breve veremos que la idea sigue prevaleciendo en los mecanismos modernos para tratar este problema, sólo que un tanto disfrazada.

4.3.2. Representaciones Subespecificadas

Llega el momento de ver los mecanismos actuales para resolver las ambigüedeades de alcance. Estos métodos, tienen su base en los siguientes inconvenientes que presenta el tratamiento con el que PTQ solventa las ambigüedades de alcance. Primero, como ya mencionamos, al introducir nuevas reglas en la PSG, perdemos la intuición gramatical sobre el NL en cuestión, que el mecanismo de la PSG nos provee inicialmente. Nuestra idea de una gramática para un NL es que nos provea de reglas generales para la formación de las oraciones, en términos sencillos, a partir de las categorías más básicas, y no que sean reglas raras que trabajen para resolver las ambigüedades. La segunda cuestión es que cada distinta fuente de ambigüedad nos obliga a añadir reglas extra en la PSG, es decir, tendríamos que añadir reglas para tratar las ambigüedades surgidas a partir de la negación y el cuantificador universal, otras más para el caso del cuantificador existencial y el universal, etc. En pocas palabras no hay un tratamiento uniforme para las ambigüedades de alcance. Una última inconveniencia reside en el hecho de que al momento de implementar se vuelve engorroso el mantenimiento de los programas, pues es difícil darle un tratamiento modular directo a PTQ.

Los primeros intentos de implementar modularmente las ideas de PTQ fueron creciendo en grado de complejidad, al principio se intentó concentrarse en una sola fuente de ambigüedad y dar métodos para resolverla. Los métodos de almacenaje como los de Cooper y Keller pueden verse como intentos de este tipo. Pero conforme se presentaban inconvenientes con los métodos desarrollados, se divisó la posibilidad de dar un tratamiento uniforme a las fuentes de ambigüedad. El método que aquí presentamos de representaciones subespecificadas es uno de los muchos presentados recientemente.

Dieñado originalmente por Johan Bos a finales de los 90, y descrito con detalle y ejemplos en [BB05], este método en principio sirve para su propósito original: resolver ambigüedades de alcance de varios tipos. Pero, aunque los autores no lo dicen, la manera en que está pensado podría ser más flexible aun y resolver otro tipo de ambigüedades. La idea central es diseñar un *metalenguaje* formal (en nuestro caso será la *semántica de hoyos*) que nos sirva para llevar el proceso de construcción (árbol) de las expresiones en el lenguaje objeto. El metalenguaje trabajará sobre lo que se define como representaciones subespecificadas, que no son otra cosa que expresiones "parciales" de algún lenguaje lógico objeto. Estas expresiones estarán sujetas a ciertas restricciones, restricciones que deben estar explícitamente dadas en el metalenguaje. Luego, una función se encargará de llenar los espacios vacíos de la expresión, y al final del proceso se podrán obtener las diferentes fórmulas del lenguaje objeto asociadas a una sola expresión del NL.

Cabe mencionar que uno de los puntos importantes en este desarrollo es el de representaciones semánticas. A medida que se ha ido desarrollando la teoría, se ha visto que la manera en que la información semántica es representada, juega un papel crucial en la complejidad que se tenga a la hora de describir la información. Con esto dicho procedemos en nuestra tarea.

4.3.3. Semántica de Hoyos

Llamaremos en lo subsiguiente SRL al lenguaje lógico objeto en el que estamos representando las oraciones de un NL. Las iniciales SRL vienen de *lenguaje de representaciones semánticas*. Definiremos un metalenguaje que llamaremos URL por las iniciales de *lenguaje de representaciones subespecificadas*. Como la idea es llevar el seguimiento de la construcción de una fórmula en SRL, debemos tener el vocabulario adecuado. Sea SRL el lenguaje lógico objeto (por ejemplo FOL) sobre el que estamos trabajando, con vocabulario A. El vocabulario del lenguaje URL se define como sigue:

> Cada símbolo de constante de SRL es también un símbolo de constante de URL.
>
> Por cada símbolo de predicado de aridad n *symbol*, de SRL, hay un símbolo de predicado de aridad $n+1$ *:symbol*, en URL.
>
> Los símbolos de predicado de aridad 2 :NOT y $\leq$.
>
> Los símbolos de aridad 3 :AND, :OR, :IMP :IFF, :ALL, :SOME y :EQ.

Esto es, tenemos en URL los símbolos para referirnos a los símbolos de SRL, pues es claro que emplearemos a los símbolos :AND, :OR, : IMP, :IFF, :ALL, :SOME y :EQ para referirnos a las ocurrencias de los símbolos $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$,

$\forall$, $\exists$ e $=$ en una fórmula de SRL. El único símbolo adicional que hemos introducido en URL es el $\leq$, que nos servirá para declarar las restricciones a las que estará sujeta la fórmula. Las cosas que podrán ser comparadas mediante este símbolo son de tres distintas formas: algo que llamaremos *hoyos* que denotaremos h, h$'$, h$_i$,..., *etiquetas* que denotaremos l, l$'$, l$_j$,... y *meta-variables*[2] que denotaremos v, v$'$,v$_k$,.... Entenderemos por *meta-término* una variable o constante de URL, y un *nodo* será un hoyo o una etiqueta. Ahora diremos qué tipo de expresiones pueden construirse en URL. Empezamos con las representaciones subespecificadas (USRs) básicas.

(i) Si l es una etiqueta y h un hoyo, entonces l$\leq$h es una USR básica.

(ii) Si l es una etiqueta, y n, n$'$ son nodos, entonces l:NOT(n), l:AND(n,n$'$), l:OR(n,n$'$), l:IMP(n,n$'$) y l:IFF(n,n$'$) son USRs básicas.

(iii) Si l es una etiqueta, y t, t$'$ son meta-términos, entonces l:EQ(t,t$'$) es una USR básica.

(iv) Si l es una etiqueta, y S es un símbolo de SRL con aridad n, y t$_1$,...,t$_n$ son meta-términos, entonces l:S(t$_1$,...,t$_n$) es una USR básica.

(v) Si l es una etiqueta, v es una meta-variable, y n es un nodo, entonces l:SOME(v,n) y l:ALL(v,n) son USRs básicas.

(vi) Sólo son USRs básicas las que se obtengan aplicando (i)–(v) un número finito de veces.

Los incisos (ii)–(v) nos indican la forma que tiene el nodo en cuestión, es decir, qué símbolo participa. Notemos que las expresiones que se obtienen de estos incisos tienen una l al inicio, a estas expresiones las llamaremos *fórmulas etiquetadas*. En cuanto al inciso (i), este es el que nos dice las retricciones que la fórmula de SRL deberá cumplir, por ejemplo, una USR de la forma l$\leq$h nos dice que un hoyo h domina a la posición con etiqueta l. Estas USRs las llamaremos *restricciones de dominancia*. El resto de las USRs se dan a continuación.

(i) Toda expresión USR básica es una USR.

(ii) Si ϕ es una USR, y n es un nodo, entonces $\exists n\phi$ es una USR.

(iii) Si ϕ es una USR, y v es una meta-variable, entonces $\exists$vϕ es una USR.

(iv) Si ϕ y ψ son USRs, entonces $\phi \wedge \psi$ es una USR.

(v) Sólo son USR las que se obtengan al usar un número finito de veces (i)–(iv).

[2]Es decir, variables de URL.

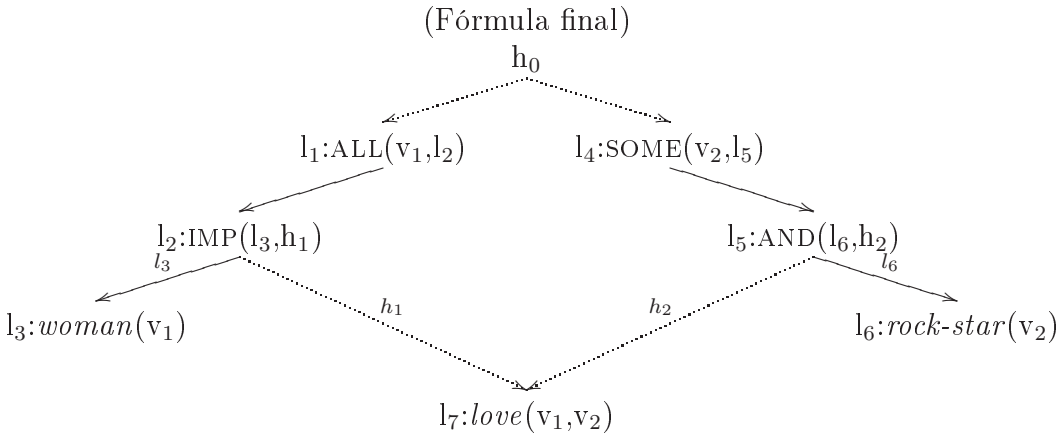
Como podemos ver el meta-lenguaje URL sólo emplea fórmulas en forma conjuntiva cerradas existencialmente. Revisemos como aplicar esta sintaxis en un ejemplo. Vamos a representar la oración *every woman loves a rock-star*. Lo primero será asociar las palabras individuales con su respectivo símbolo en URL, esto lo hacemos así:
every$\dashrightarrow \exists l_1 \exists l_2 \exists v_1 (l_1{:}\textsc{all}(v_1,l_2) \wedge \exists l_3 \exists h_1 (l_2{:}\textsc{imp}(l_3,h_1)))$
woman$\dashrightarrow \exists l_3 \exists v_1 (l_3{:}woman(v_1))$
loves$\dashrightarrow \exists l_7 \exists v_1 \exists v_2 (l_7{:}love(v_1,v_2))$
a$\dashrightarrow \exists l_4 \exists l_5 \exists v_2 (l_4{:}\textsc{some}(v_2,l_5) \wedge \exists l_6 \exists h_2 (l_5{:}\textsc{and}(l_6,h_2)))$
rock-star$\dashrightarrow \exists l_6 \exists v_2 (l_6{:}rock\text{-}star(v_2))$
Esto parece ser muy complicado, pero si lo examinamos con ciudado, podemos ver que estamos siguiendo la misma estrategia que hemos venido manejando desde que usamos la FOL para representar a esta oración. Además intencionalmente hemos hecho que los subíndices empaten para lo que sería la construcción de la fórmula final. Ahora veamos por qué esta representación es adecuada. Primero, la etiqueta antes de cada símbolo, nos sirve para identificar la parte de la fórmula sobre la que se hace una afirmación, así vemos que por ejemplo *woman* se lo identifica con la etiqueta l_3, mientras que a *every* se lo identifica con las etiquetas l_1 y l_2, pues su representación está dada por dos símbolos: :ALL e :IMP. Nuestro objetivo es decir en principio que toda mujer ama "algo", es por esto que la variable v_1, que corresponde al primer argumento de :ALL, es la misma que aparece en el argumento de :*woman*, mientras que el segundo argumento de :ALL es llenado por la etiqueta que hace referencia al símbolo :IMP, l_2, (nada nuevo hasta ahora). Lo novedoso está presente en esta parte de la fórmula, pues aunque el primer argumento de :IMP es lo que esperaríamos, esto es, la etiqueta l_3 que nos refiere al símbolo :*woman*, el segundo argumento lo ocupa un hoyo h_1. Esto nos dice intuitivamente que el consecuente de la implicación puede ser de muchas formas. Similarmente podemos ver que la representación de *a rock-star* deja subespecificada el segundo conyunto de :AND, mediante el hoyo h_2, poniendo solamente la información del primer conyunto, donde se nos indica que el símbolo :*rock-star* participa. En principio, estos hoyos podrían ser llenados con cualquier etiqueta presente dentro de la representación, pero esto puede causar lecturas indeseadas[3], es aquí donde las restricciones nos ayudan. Como queremos que *:love* tenga la menor prioridad (ocurra primero en la construcción del árbol), y este símbolo tiene la etiqueta l_7, imponemos las restricciones $l_7 \leq h_1$ y $l_7 \leq h_2$. Ya que las representaciones del cuantificador universal y el existencial están en estas etiquetas, lo que estamos pidiendo en el fondo es que el símbolo :*love* sea alcanzado por estos dos cuantificadores.

[3]Que no tengan nada que ver con la oración original a la que pretendemos representar.

Pensando en que la fórmula final está asociada con un hoyo h_0, también ponemos la restricción de que esta representación contenga a las representaciones del cuantificador universal y el existencial, lo que nos da $\exists h_0(l_1 \leq h_0 \wedge l_4 \leq h_0)$. No tenemos que inmiscuir a las otras etiquetas porque el juego está en la relación de los cuantificadores y sus respectivas representaciones, y la manera en que alcancen al símbolo :*love*. La representación final será entonces tomar la conjunción de cada representación individual y formar la conjunción también repecto de las restricciones. Por supuesto, la forma conjuntiva nos ayuda a eliminar repeticiones del cuantificador existencial, por lo que podemos quedarnos con la representacion

$\exists h_0 \exists h_1 \exists h_2 \exists l_1 \exists l_2 \exists l_3 \exists l_4 \exists l_5 \exists l_6 \exists l_7 \exists v_1 \exists v_2($
l_1:ALL(v_1,l_2)$\wedge$
l_2:IMP(l_3,h_1)$\wedge$
l_3:*woman*(v_1)$\wedge$
l_4:SOME(v_2,l_5)$\wedge$
l_5:AND(l_6,h_2)$\wedge$
l_6:*rock-star*(v_2)$\wedge$
l_7:*love*(v_1,v_2)$\wedge$
$l_7 \leq h_1 \wedge$
$l_7 \leq h_2 \wedge$
$l_1 \leq h_0 \wedge$
$l_4 \leq h_0$).

Aunque esta enorme cadena de símbolos es mejor entendible si se la representa de manera gráfica

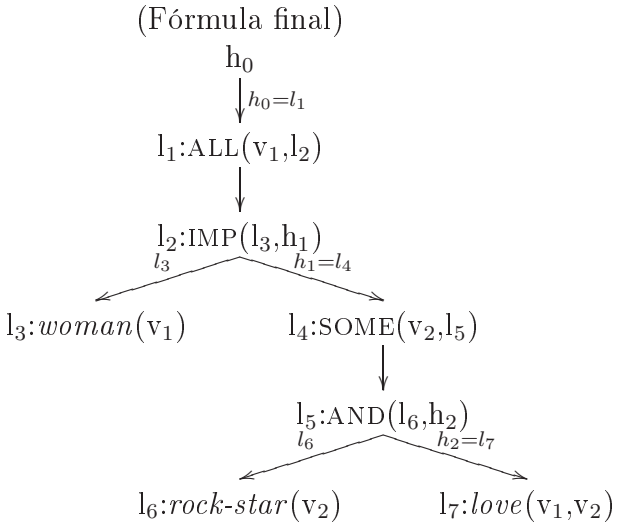


Las líneas sólidas indican que el nodo en cuestión se compone de su(s) hijo(s), mientras que las líneas punteadas indican las restricciones que hay entre los

hoyos y las etiquetas. Notemos por ejemplo, que la etiqueta l_7 es dominada por los hoyos h_1 y h_2 (y por tanto por h_0), mientras que las etiquetas l_1 y l_4 están dominadas por h_0. En este sentido, podemos pensar que las líneas sólidas indican que el "camino" ya está definido o dado, mientras que las líneas punteadas nos dicen que el camino no está definido todavía (en principio uno podría tomar cualquiera de ellos). El hecho de que los hoyos aparezcan, evita que nos comprometamos en alguna elección de algún cuantificador por sobre otro. Aunque al final queremos poder llenar los hoyos de alguna manera, de modo que lleguemos a una representación semántica en SRL. Es decir, necesitamos llenar los hoyos con una fórmula (etiqueta) de manera que se satisfagan todas las restricciones. Por supuesto que no podemos permitir que haya una etiqueta asignada a más de un hoyo. Por lo tanto, podemos ver el mecanismo que nos permite llenar los hoyos (llamémosle *enchufador*) como una función inyectiva cuyo dominio es el conjunto de hoyos y codominio el conjunto de etiquetas. Un enchufador será *admisible* para una USR si al sustituir las etiquetas de la manera que nos indica el enchufador, se satisfacen todas las retricciones en la USR. Por ejemplo, la USR que representa a la oración *every woman loves a rock-star*, tiene dos enchufadores posibles, P y P′:

$\mathrm{P}(\mathrm{h}_0) = l_1 \quad \mathrm{P}(\mathrm{h}_1) = l_4 \quad \mathrm{P}(\mathrm{h}_2) = l_7$
$\mathrm{P}'(\mathrm{h}_0) = l_4 \quad \mathrm{P}'(\mathrm{h}_1) = l_7 \quad \mathrm{P}'(\mathrm{h}_2) = l_1$

Para verificar que en efecto son admisibles, podemos sustituir en los hoyos de la representación gráfica de la USR las etiquetas señaladas por el enchufador. Por ejemplo, para P obtendríamos la figura



Obsevamos que no hay más líneas punteadas (el camino ya está definido del todo), pues cada hoyo se ha identificado con una etiqueta de acuerdo a lo que

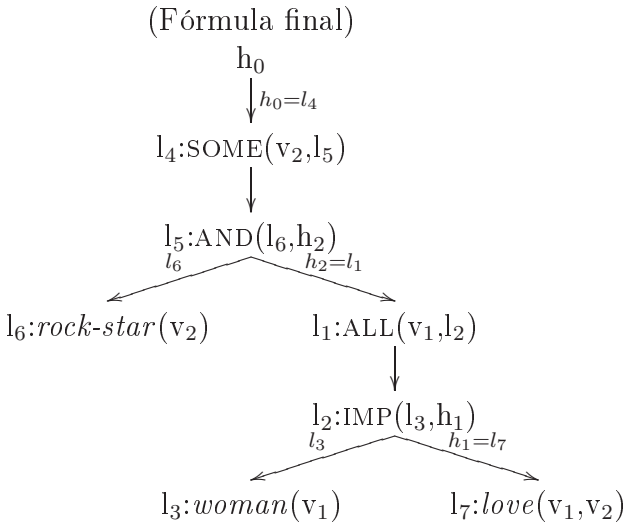
indica P. Si el árbol se linealiza en este momento, obtenemos una expresión de SRL

ALL(v_1,IMP($woman$(v_1),SOME(v_2,AND($rock\text{-}star$(v_2),$love$(v_1,v_2))))),

que corresponde a la fórmula de FOL

$\forall x(woman(x) \rightarrow \exists y(rock\text{-}star(y) \wedge love(x,y)))$.

lo que nos señala que hemos logrado construir la fórmula donde el cuatificador universal tiene un alcance mayor. En cuanto al otro enchufador, P′, tiene asociada la figura



que no es otra cosa que el árbol de una fórmula de SRL la cual podemos linealizar para leer como

SOME(v_2,AND($rock\text{-}star$(v_2),ALL(v_1,IMP($woman$(v_1),$love$(v_1,v_2))))).

La representación en SRL de la fórmula de FOL

$\exists y(rock\text{-}star(y) \wedge \forall x(woman(x) \rightarrow love(x,y)))$,

la lectura donde el cuantificador existencial tiene un alcance mayor.

Este novedoso método, que en principio parece sumamente intrincado, puede

implementarse computacionalmente[4]. En su libro [BB05], los autores presentan una implementación en Prolog, que permite ver lo satisfactorio de esta aproximación en el tratamiento de diversas ambigüedades.

4.4. Lenguaje Natural: Representación, Inferencia y Uso

Una vez que hemos logrado representar las oraciones de un NL y logrado resolver los problemas de las ambigüedades (al menos algunas de ellas). Lo que sigue es usar métodos de deducción para tratar de modelar cosas de interés en el NL. Por ejemplo, la toma de decisiones, las respuestas automáticas, el seguimiento de una conversación, etc. Esta es una área de intensa investigación actualmente y que ha llevado a tratar problemas interesanes de otras disciplinas, como por ejemplo, desarrollar un constructor de modelos (digamos de FOL) para una fórmula dada, y ver si la fórmula es satisfacible. Pero la CS va más allá. Después de todo uno de los objetivos es modelar el NL lo que tiene severas complicaciones. Y también se ha tratado de darle representación al uso cotidiano del lenguaje (lo que comúnmente se denomina pragmática). Por lo pronto nuestro cometido de presentar algunas formas de representación de oraciones del NL en algunos lenguajes lógicos se ha cumplido. Resta hacer solamente unos comentarios finales que reservamos para la siguiente sección.

[4] Aunque hacerlo requiere cierta habilidad en programación.

Capítulo 5

Conclusiones

Gottlob Frege inició un programa para ligar a una expresión con su significado, en términos de las partes que la componían. Aunque su intento no fue del todo fructífero, abrió las puertas al método de composicionalidad que actualmente predomina en prácticamente todas las áreas de la CS. Años después, Alonzo Church diseñó un lenguaje que empataba con estas ideas de manera impecable y que era posible implementar computacionalmente. Más tarde, Saul Kripke daba una semántica a la lógica modal, que podía servir de marco a la descripción de posibilidades. Finalmente en 1973 Richard Montague usó todos estos elementos para dar un tratamiento formal a algo que parecía imposible, un (fragmento de) NL. Desde su programa a la fecha, muchas aproximaciones se han dado, tratando de cumplir su programa original. En el camino se han visto muchas complicaciones, que poco a poco se han ido resuelto. Sin embargo, aun hay mucho por hacer en cuanto a esto se refiere. Y muchas interesantes preguntas aun quedan sin respuesta.

En la presente obra hemos revisado algunos lenguajes lógicos que se han usado para representar el lenguaje natural, y algunas formas de representaciones semánticas para las oraciones, que tratan de capturar el significado original de las mismas. Pero quedan inconclusas varias cosas, como el método para relacionar a las oraciones y consecuencias lógicas de éstas. Los métodos para hacer esto, son muy variados, y constituyen uno de los objetivos de la CS, por lo que mucho tiempo y esfuerzo se ha dirigido en esta dirección. Una forma de hacerlo, es usar alguno de los métodos disponibles de deducción en FOL, como resolución o tablas semánticas, que además, pueden ser implementados computacionalmente. Aunque esta aproximación sólo sería válida para la FOL. En IFOL, el desarrollo de tales métodos de deducción, todavía es una sima inexplorada actualmente. Además, estos métodos están en constante cambio y muchos de ellos todavía requieren de mayor refinamiento.

Otro de los objetivos de la CS de los que no hablamos es la pragmática, algo

muy reciente y que requiere nuevos tratamientos, pues de hacerlo con los métodos dados aquí, muy seguramente crearán soluciones inaceptables desde el punto de vista de NL.
Pero los mismos métodos aquí descritos tienen cuestiones sin repuesta hasta el momento. En las siguientes líneas mencionaremos algunos. El primero es el que concierne al problema de la expresividad. Hemos visto que tanto la FOL, la IFOL, la TT y L_λ, todos presentan este problema. En FOL los verbos modales son muy complicados de representar con todo detalle, es decir, respetando las propiedades que presentan intuitivamente en el NL, y aunque pueden tratar de usarse algunos métodos para resolver esto, por ejemplo, la semántica de Henkin, el resultado es de difícil tratamiento. Mucho más intuitivo y sencillo es usar la IFOL. Sin embargo, la IFOL no está excenta de problemas de expresividad, siendo las cláusulas relativas, de las cuales hablaremos en breve, una de las fuentes mayores de estos problemas. Los cuantificadores generalizados son también un reto enorme, presente en todos los lenguajes lógicos usados en la presente, y a pesar de que algunos de ellos pueden plasmarse en un lenguaje lógico, como el de la FOL[1], aun no reciben un tratamiento uniforme, lo que cuesta mucho computacionalmente hablando. Tenemos que admitir que en este sentido los métodos heurísticos, tan usados en NLP, son mucho mejores en cuanto a resultados y desempeño computacional, que hacerlo directamente. Pero un problema de expresividad inmenso y que no ha recibido tanta atención, presumiblemente debido a su complejidad, es el de las cláusulas relativas. Hablemos un poco de este problema. En el capítulo 2, revisamos un ejemplo en la parte de lógica modal proposicional que no era claro con respecto a qué fórmula asociarle. La oración era

(2.5) *Pherhaps it's raining, and maybe this is necessary.*

con representaciones

(2.10) $\Diamond p \wedge \Diamond p \Box p$

y

(2.11) $\Diamond p \wedge \Diamond p \Box \Diamond p$

Ambas parecen expresar cierto sentido de la oración (2.5), pero no podemos comprometernos con alguna de ellas al cien. El problema yace en la cláusula relativa indicada por *this*.
Veamos un ejemplo más intrincado. La oración

[1] En [Gam1b] se muestra cómo hacer esto en algunos casos sencillos.

He says that Melissa and Bill were at the party, Lola was near the house too, but she didn't see him there,
las preguntas son a quién se refiere el pronombre *she*, a *Lola* o a *Melissa*, a quién se refiere el pronombre *him* a *Bill* o al pronombre *He*, y a qué lugar se refiere el adverbio locativo *there*, a *the party* o a *near the house*. En la práctica, pareciera que *she* es una referencia a *Lola*, por aparecer al final de la oración, justo después de la mención a *Lola*. Pero si se entonara con más énfasis a *Melissa* que a *Lola*, y la parte de *Lola* se pronunciara rápidamente (tal vez la charla es acerca de si *Melissa* fue a la fiesta), ya no estaríamos tan seguros de nuestra primera afirmación. Problemas similares se presentan con *he* y *there*. Como puede apreciarse, este es un problema difícil, sobre todo si consideramos una aproximación directa[2]. De nueva cuenta resulta ser más útil hasta el momento usar los métodos probabilistas desarrollados en el ámbito del NLP.
Claro está que no nos ayudaría mucho usar IFOL o L_λ, como lenguajes de representación, para tratar de resolver estos problemas de cláusulas relativas.
El segundo problema mayor es el de la complejidad computacional, y para mostrarlo nada mejor que referirnos a la IFOL. En la representación del lenguaje natural mediante IFOL, quedan varios inconvenientes computacionales en el tratamiento que dimos. En principio, si deseáramos dar un tratamiento completo al NL, tendríamos que incluir un operador modal por cada verbo modal, esto ocasionaría que el algoritmo fuera muy complicado aun para situaciones sencillas. Experimentos en esta parte mostraron que un anidamiento de tres operadores modales, en la fórmula a evaluar, ya causa una pequeña pausa en la velocidad de respuesta de una consulta, en un modelo que considera ocho mundos posibles y algunos símbolos de contante y relación. Así que añadir un operador para la creencia, otro para la necesidad, otro para el deber, etc. Incrementaría el grado de complejidad del algoritmo y los cálculos necesarios para dar una respuesta. Y si pensamos que no hay oraciones con este número de operadores basta revisar la siguiente oración donde se marcan los verbos modales (y eso que no hemos incluido el fenómeno temporal que tendría que respresentarse también mediante IFOL):
Bill ***sabe*** *que mañana* ***debe*** *tramitar su visa o* ***tendrá*** *que hacer la cita en la embajada nuevamente.*

El cálculo lambda también posee detalles que pulir, y un camino no explorado todavía es usar métodos de deducción en la teoría de tipos. Esto último sería muy interesante, pues nos diría que cada palabra individual lleva una

[2] Aquí precisamente es donde el método de representaciones subespecificadas podría ser de utilidad. Pero resta que haya más interés por investigar al respecto.

carga semántica inherente, que al asociarse en la oración completa, contribuye de algún modo en el significado de la oración, aún antes de la formación de la oración misma. Es decir, tendríamos alguna idea de qué esperar en la oración final en base a sus componentes lexicales, lo que suena ciertamente intuitivo. Pero sin alejarnos tanto, el tratamiento que dimos a la teoría de tipos posee inconvenientes, por ejemplo, el uso de conectivos lógicos sólo se puede aplicar en cosas del tipo t. Si añadimos más conectivos, uno para cada tipo, podríamos construir oraciones con conjunciones de frases nominales por mencionar alguna. Aunque queda por ver qué tan conveniente es hacer esto.

Con respecto a las representaciones subespecificadas, es un tratamiento muy reciente, que está en constante cambio, pero lo que sí es notorio, es la versatilidad con la que se pueden manejar diferentes representaciones semánticas y las consecuencias de hacerlo de una u otra forma. Una incógnita que merece pronta respuesta en la parte de semántica de hoyos es qué tan adecuado es usar un metalenguaje en el proceso de descripción de la fórmula del lenguaje objeto. Se supone que la idea es tener un mecanismo claro para generar las fórmulas del lenguaje, pero pareciera que el proceso es increíbemente complicado para tales fines, y lo que uno se ahorra en la sintaxis, lo termina pagando en la representación semántica. Además, las (muchas pero aun escasas) ambigüedades que se pueden resolver con este método pertenecen al ámbito de la FOL, no se sabe qué ocurra en otros territorios como la IFOL. Por ejemplo, ya ejemplificamos que (2.5) tiene dos posibles fórmulas asociadas en el lenguaje de la lógica proposicional modal, (2.10) y (2.11), y aun no se sabe de un mecanismo que permita hacer la desambiguación de este tipo de sentencias para este lenguaje.
En cuanto al lenguaje de programación usado, se ha podido ver la ventaja que resulta de emplear un lenguaje declarativo como Prolog, ya que hemos podido presentar el código de los programas de manera casi transparente, algo imposible de hacer si usáramos un lenguaje imperativo. No hemos tenido que preocuparnos por implementar analizadores o generadores gramaticales por el mecanismo interno de Prolog de gramáticas de cláusulas definidas. Y la representación semántica de los operadores lógicos es una copia fiel de las definiciones de satisfacción.
Otra cosa que no ha sido discutida en detalle es que los métodos aquí expuestos (y muchos de los usados en la CS) van ligados a una sintaxis de por medio (por este motivo hay quienes catalogan este tipo de aproximaciones como *sintax-driven semantics*). Y hacer esto conlleva una serie de cuestiones que van ligadas a las relaciones de ser constituyente en un árbol de análisis para una oración y que son amplio objeto de discusión en la sintaxis (gramática) de un NL. Por último discutiremos un poco de una de las muchas

posibles aplicaciones de estos desarrollos, algo que iniciamos desde el capítulo 1. En dicha sección discutíamos la posibilidad de llevar el seguimiento de una conversación entre Cat y la funcionaria de una dependencia. Claro que entre otras cosas, tendríamos que haber desarrollado módulos que lleven a cabo la deducción y la pragmática. Pero suponiendo que los tuviéramos, seríamos capaces de automatizar el proceso de dicho trámite mediante una máquina a la que se le podría contestar en NL, y no solamente mediante una serie de opciones de respuesta (como se puede hacer actualmente). Más aun, la computadora podría estar dotada con una base de conocimiento relativa al área de trámites de migración, de donde en principio, podría deducir si la información que se le da es suficiente para considerar expedir el documento en cuestión o no hacerlo. Pero llevar el estado de conocimiento nos vuelve a regresar al tratamiento con IFOL, y ya comentamos lo costoso que es esto computacionalmente. De llevarlo a cabo deberíamos al menos tener un proceso que nos permitiera reducir el modelo de Kripke a solamente algunos mundos posibles para no hacer una búsqueda tan general.

Por supuesto, hacer un programa que cumpla, al menos en cierta medida, estos objetivos, sería un ambicioso proyecto que requeriría de un desarrollo ordenado, tomar prestados una buena cantidad de métodos tanto de computación como de lógica y lingüística, pulir los procedimientos actuales, algunos de los cuales hemos presentado, y desarrollar aproximaciones a fenómenos del lenguaje que aun permanecen en oscuridad.

Apéndices

Apéndice A

A.1.

En esta sección mostramos la manera de hacer consultas Con los programas de revisión de satisfacción de una fórmula de IFOL en un modelo de Kripke con respecto a una asignación y un mundo. Todas las implementaciones se realizaron y probaron en SWI-Prolog (Linux y Windows), y algunas corren también en GProlog (Linux). Los cuatro archivos principales son
kripkeModelChecker.pl,
modelosEjemplo.pl,
modelCheckerTestSuite.pl, y
predSemCom.pl.
Estos archivos se muestran en apéndices separados por conveniencia. El primero que mencionamos, **kripkeModelChecker.pl**[1] contiene las implementaciones de la semántica de mundos posibles tal como se mostró en el texto, añadiendo el chequeo de que la expresión esté bien formada. Aquí se encuentran las definiciones de los operadores modales (cuadrado y rombo), los cuantificadores (existe y para todo), los conectivos lógicos (negación, conjunción, disyunción, implicación y equivalencia), y las fórmulas atómicas (igualdad de términos y relaciones instanciadas con términos). Para hacer consultas en el motor de inferencia modal desarrollado, basta iniciar Prolog (en GNU-Linux esto sería dar `swipl` en el prompt de una terminal, y luego cargar este archivo con la instrucción `['kripkeModelChecker.pl']`. mientras que en Windows basta dar doble click en el icono del archivo), pero debe notarse que todos los archivos deben estar en el directorio de trabajo. El programa se encargará de hacer los enlaces con los otros archivos para poder hacer consultas.
Ahora, tal vez se quiera saber el contenido de los otros archivos, sobre todo porque estos son los que se modifican para probar las fórmulas y los

[1] Apéndice D.

modelos. El archivo **modelosEjemplo.pl**[2], como su nombre indica, contiene algunos ejemplos de modelo de Kripke con la forma que se dio en el texto, como segundo argumento del predicado `ejemplo(NumModelo,Kmodelo)`, el primer argumento sirve para discriminar en el ejemplo que se quiera trabajar. Este archivo puede editarse para cambiar los modelos: añadir mundos, cambiar la relación de accesibilidad, los símbolos, las relaciones, etc.
El archivo **modelCheckerTestSuite.pl**[3] contiene las fórmulas de la IFOL que queremos probar, en la forma dada en el texto, por ejemplo, `poss(woman(lola))`, como primer argumento del predicado
`prueba(Formula,NumModelo,Mundo,Asignacion,Polaridad).`
Los otros argumentos indican el numero de modelo sobre el que se quiere trabajar (este es el que empata con el número del predicado
`ejemplo(NumModelo,Kmodelo))`,
el mundo en el que se quiere hacer la evaluación, la asignación de variables g y la polaridad con la que se quiere evaluar la fórmula. Este archivo puede editarse para probar diferentes fórmulas de la IFOL, o probarlas en otros modelos, etc.
El último archivo, **predSemCom.pl**[4] contiene definiciones de predicados auxiliares, como el que descompone una expresión en sus partes, y no necesita modificarse para el funcionamiento del programa principal.
Una vez que se inició el programa kripkeModelTestChecker, se puede hacer la evaluación de las fórmulas en el archivo **modelCheckerTestSuite.pl** en los ejemplos que contiene **modelosEjemplo.pl** simplemente tecleando en el prompt de Prolog
`?-modelCheckerTestSuite.`
y dando enter.
La siguiente es un fragmento de la corrida.

```
Fórmula de Entrada:
1 woman(cindy)
Mundo: w1
Ejemplo de Modelo: 1
Estado: No satisfacible en el modelo.
Checador de Modelos dice: Satisfacible en el modelo. ¡RESULTADO INESPERADO!


Fórmula de Entrada:
1 some(A,woman(A))
Mundo: w1
```

[2] Apéndice E
[3] C.
[4] Apéndice B.

```
Ejemplo de Modelo: 1
Estado: Satisfacible en el modelo.
Checador de Modelos dice: Satisfacible en el modelo. ¡CORRECTO!

Fórmula de Entrada:
1 not(some(A,woman(A)))
Mundo: w1
Ejemplo de Modelo: 1
Estado: Satisfacible en el modelo.
Checador de Modelos dice: No satisfacible en el modelo. ¡RESULTADO INESPERADO!

Fórmula de Entrada:
1 some(A,nurse(A))
Mundo: w1
Ejemplo de Modelo: 1
Estado: Satisfacible en el modelo.
Checador de Modelos dice: Satisfacible en el modelo. ¡CORRECTO!
```

Notemos que se nos da información de qué fórmula pretendemos evaluar, el mundo en el que queremos hacerlo, el ejemplo de modelo en el que estamos evaluando, y la polaridad que le pedimos. Al final, se nos dice si la fórmula es satisfacible en el modelo o no de acuerdo a lo que el programa determine.

Apéndice B

B.1.

Lo siguiente es el código del archivo **predSemCom.pl**

```
:- module(predSemCom,
      tractDomain/2,
extractRel/2,
extractWorldModel/3,
/* appendLists/3,*/
/* basicFormula/1,*/
/**/ descomponer/3,
/* concatStrings/2,*/
/* executeCommand/1,*/
/**/ infija/0,
/**//* memberList/2,*/
/* newFunctionCounter/1,*/
/**/ prefija/0,
/**/ imprimeRepresentacion/1,
probarUnica/1,
/* removeFirst/3,*/
/* removeDuplicates/2,*/
/* reverseList/2,*/
selectFromList/3
/* simpleTerms/1,*/
/* substitute/4,*/
/* unionSets/3,
variablesInTerm/2 */

).


/***** descomponer *****/


descomponer(Termino,Simbolo,ListaArg):-
Termino =.. [Simbolo|ListaArg].


/***** infija-prefija *****/


:- dynamic bbmode/1.
```

```
bbmode(prefix).
infija:- retractall(bbmode(_)), assert(bbmode(infix)).
prefija:- retractall(bbmode(_)), assert(bbmode(prefix)).

/***** imprimeRepresentacion *****/

imprimeRepresentacion(Lecturas):-
imprimeRep(Lecturas,0).
imprimeRep([],_):- nl.
imprimeRep([Lectura|OtrasLecturas],M):-
N is M + 1, nl, write(N), tab(1),
\+ \+ (numbervars(Lectura,0,_), print(Lectura)),
imprimeRep(OtrasLecturas,N).

/********** probarUnica **********/

probarUnica(Meta):- call(Meta), !.

/********** Extraer el dominio de un modelo de Kripke **********/

extractDomain(KModel,Dominio):-
KModel = kmodel(Dominio,_,_).

/********** Extraer la relacion de accesibilidad de un modelo de Kripke **********/

extractRel(KModel,Relacion):-
KModel = kmodel(_,Relacion,_).

/********** Extraer el modelo asociado a un mundo de un modelo de Kripke **********/

extractWorldModel(KModel,Mundo,Model):-
KModel = kmodel(Dominio,_,ListaMundos),
member(X,ListaMundos),
arg(1,X,Model),
functor(X,Mundo,1),
member(Mundo,Dominio).

/********** Remover un elemento de una lista. **********/

selectFromList(X,[X|L],L).
selectFromList(X,[Y|L1],[Y|L2]):-
```

```
selectFromList(X,L1,L2).
```

Apéndice C

C.1.

El siguiente es el código del archivo **modelCheckerTestSuite.pl**

```
:- module(modelCheckerTestSuite,[prueba/5]).

/********** Ejemplos de fórmulas para probar. **********/

prueba(woman(cindy),1,w1,[],neg).

prueba(some(X,woman(X)),1,w1,[],pos).

prueba(not(some(X,woman(X))),1,w1,[],pos).

prueba(some(X,nurse(X)),1,w1,[],pos).

prueba(not(some(X,and(nurse(X),woman(X)))),1,w1,[],neg).

prueba(all(X,imp(nurse(X),woman(X))),1,w1,[],pos).

prueba(all(X,imp(woman(X),nurse(X))),1,w1,[],neg).

prueba(all(X,iff(nurse(X),woman(X))),1,w1,[],pos).

prueba(not(some(X,examine(X,alis))),1,w1,[],neg).

prueba(some(X,and(examine(X,alis),medic(X))),1,w1,[],pos).

prueba(poss(some(X,some(Y,and(nurse(X),inject(X,Y))))),1,w1,[],pos).

prueba(poss(nurse(cindy)),1,w1,[],pos).
```

```
prueba(nec(nurse(cindy)),1,w1,[],pos).

prueba(nec(some(X,some(Y,and(nurse(cindy),and(medicine(Y),give(cindy,X,Y)))))),1,w1,[],pos).

prueba(nec(poss(nurse(cindy))),1,w1,[],pos).

prueba(poss(nec(nurse(cindy))),1,w1,[],pos).

prueba(poss(nec(and(patient(alis),nurse(cindy)))),1,w1,[],pos).

prueba(poss(some(X,nec(nurse(X)))),1,w1,[],pos).

prueba(nec(some(X,poss(nurse(X)))),1,w1,[],pos).

prueba(nec(some(X,some(Y,poss(and(and(medicine(X),patient(Y)),give(cindy,Y,X)))))),1,w1,[],pos).

prueba(nec(some(X,some(Y,some(Z,poss(and(and(medicine(X),patient(Y)),give(Z,Y,X))))))),1,w1,[],pos).

prueba(nec(some(X,some(Y,some(Z,give(Z,Y,X))))),1,w1,[],pos).

prueba(some(X,some(Y,some(Z,nec(give(Z,Y,X))))),1,w1,[],pos).

prueba(nec(some(X,dancer(X))),2,k1,[],pos).

prueba(nec(some(X,and(dancer(X),woman(X)))),2,k1,[],pos).

prueba(nec(some(X,and(dancer(X),man(X)))),2,k1,[],pos).

prueba(poss(some(X,and(dancer(X),man(X)))),2,k1,[],pos).

prueba(poss(some(X,and(dancer(X),man(X)))),2,k2,[],pos).

/*

prueba(and(some(X,man(X)),some(X,woman(X))),3,w1,[],pos).

prueba(not(some(X,woman(X))),3,w1,[],neg).

prueba(some(X,and(tasty(X),burger(X))),3,w1,[],indef).
```

```
prueba(not(some(X,and(tasty(X),burger(X)))),3,w1,[],indef).

prueba(some(X,and(man(X),not(some(Y,woman(Y))))),3,w1,[],neg).

prueba(some(X,and(man(X),not(some(X,woman(X))))),3,w1,[],neg).

prueba(some(X,and(woman(X),not(some(X,customer(X))))),2,w1,[],indef).
*/
```

Apéndice D

D.1.

Lo que sigue es el código del archivo **kripkeModelChecker.pl**

```
/********** Modulo de chequeo de satisfacibilidad en un modelo de Kripke. **********/

/********** donde dice modelCheckerTestSuite en realidad son ejemplos de formulas **/

:- module(kmodelChecker,[modelCheckerTestSuite/0,
info/0,
infija/0,
prefija/0,
evaluar/2,
evaluar/3,
satisface/4,
satisface/5]).

/* Carga de modulos necesarios. */

:- use_module(predSemCom,[
extractRel/2,
extractDomain/2,
extractWorldModel/3,
probarUnica/1,
descomponer/3,
infija/0,
prefija/0,
imprimeRepresentacion/1,
selectFromList/3]).

:- use_module(modelosEjemplo,[ejemplo/2]).
```

```
:- use_module(modelCheckerTestSuite,[prueba/5]).

/********** Evaluar una fórmula en un ejemplo de modelo. **********/

evaluar(Formula,Ejemplo):-
evaluar(Formula,Ejemplo,[]).

/********** Evaluar una fórmula en un ejemplo de modelo con respecto a una asignación. **********/

evaluar(Formula,Ejemplo,Asignacion):-
ejemplo(Ejemplo,Modelo),
satisface(Formula,Modelo,Asignacion,Resultado),
imprimeEstado(Resultado).

/***** Evaluar una formula en un ejemplo de modelo de Kripke. *****/

evaluarKM(Formula,Ejemplo,Mundo):-
evaluarKM(Formula,Ejemplo,Mundo,[]).

/***** Evaluar una formula en un ejemplo de modelo de Kripke respecto de un mundo y una asigna-
cion. *****/

/***** el mundo debe ser parte del conjunto de mundos del modelo. *****/

evaluarKM(Formula,Ejemplo,Mundo,Asignacion):-
ejemplo(Ejemplo,KModelo),
satisface(Formula,KModelo,Mundo,Asignacion,Resultado),
imprimeEstado(Resultado).

/********** Suite de Pruebas. **********/

modelCheckerTestSuite:-
format('~n***** CHECADOR DE MODELOS DE KRIPKE 2 *****~n',[]),
prueba(Formula,Ejemplo,Mundo,Asignacion,Estado),
format('~n~nFórmula de Entrada:',[]),
imprimeRepresentacion([Formula]),
format('Mundo: ~p~n',[Mundo]),
format(Éjemplo de Modelo: ~p~nEstado: ',[Ejemplo]),
imprimeEstado(Estado),
ejemplo(Ejemplo,KModelo),
```

```
probarUnica(kmodelChecker3:satisface(Formula,KModelo,Mundo,Asignacion,Resultado)),
format('~nChecador de Modelos dice: ',[]),
imprimeEstado(Resultado),
imprimeComparacion(Estado,Resultado),
fail.

modelCheckerTestSuite.

/********** Imprime el estado de un ejemplo de fórmula. **********/

imprimeEstado(pos):- write('Satisfacible en el modelo. ').
imprimeEstado(neg):- write(Ño satisfacible en el modelo. ').
imprimeEstado(indef):- write(Ímposible evaluar. ').

/********** Imprime comparación del resultado obtenido y el esperado. **********/

imprimeComparacion(Esperado,Resultado):-
\+ Esperado=Resultado,
write('¡RESULTADO INESPERADO!').

imprimeComparacion(Esperado,Resultado):-
Esperado=Resultado,
write('¡CORRECTO!').

/*****************************************************************************************/

/********** Satisfacibilidad respecto de un modelo de Kripke un mundo y una asignacion **********/

/*****************************************************************************************/

/********** Satisfacibilidad de formula con operador rombo **********/

satisface(Formula,KModelo,Mundo,Asignacion,pos):-
nonvar(Formula),
Formula = poss(Subformula),
extractDomain(KModelo,Dominio),
extractRel(KModelo,Rel),
\+ Rel = [],
member(Mundo,Dominio),
member((Mundo,RMundo),Rel),
satisface(Subformula,KModelo,RMundo,Asignacion,pos).
```

```
satisface(Formula,KModelo,_,_,neg):-
nonvar(Formula),
Formula = poss(_),
extractRel(KModelo,Rel),
Rel = [].

satisface(Formula,KModelo,Mundo,_,neg):-
nonvar(Formula),
Formula = poss(_),
extractDomain(KModelo,Dominio),
extractRel(KModelo,Rel),
\+ Rel = [],
member(Mundo,Dominio),
\+ setof(X,member((Mundo,X),Rel),_).

satisface(Formula,KModelo,Mundo,Asignacion,neg):-
nonvar(Formula),
Formula = poss(Subformula),
extractDomain(KModelo,Dominio),
extractRel(KModelo,Rel),
\+ Rel = [],
member(Mundo,Dominio),
setof(X,member((Mundo,X),Rel),ALL),
setof(X,
(
member((Mundo,X),Rel),
satisface(Subformula,KModelo,X,Asignacion,neg)
),
ALL
).

satisface(Formula,KModelo,Mundo,Asignacion,indef):-
nonvar(Formula),
Formula = poss(Subformula),
extractDomain(KModelo,Dominio),
extractRel(KModelo,Rel),
member(X,Dominio),
member((Mundo,X),Rel),
satisface(Subformula,KModelo,X,Asignacion,indef).
```

```
/********** Formula con operador cuadrado. **********/

satisface(Formula,KModelo,_,_,pos):-
nonvar(Formula),
Formula = nec(_),
extractRel(KModelo,Rel),
Rel = [].

satisface(Formula,KModelo,Mundo,_,pos):-
nonvar(Formula),
Formula = nec(_),
extractDomain(KModelo,Dominio),
extractRel(KModelo,Rel),
\+ Rel = [],
member(Mundo,Dominio),
\+ setof(X,member((Mundo,X),Rel),_).

satisface(Formula,KModelo,Mundo,Asignacion,pos):-
nonvar(Formula),
Formula = nec(Subformula),
extractDomain(KModelo,Dominio),
extractRel(KModelo,Rel),
\+ Rel = [],
member(Mundo,Dominio),
setof(X,member((Mundo,X),Rel),ALL),
setof(X,
(
member((Mundo,X),Rel),
satisface(Subformula,KModelo,X,Asignacion,pos)
),
ALL
).

satisface(Formula,KModelo,Mundo,Asignacion,neg):-
nonvar(Formula),
Formula = nec(Subformula),
extractDomain(KModelo,Dominio),
extractRel(KModelo,Rel),
\+ Rel = [],
member(Mundo,Dominio),
member((Mundo,RMundo),Rel),
```

```
satisface(Subformula,KModelo,RMundo,Asignacion,neg).

satisface(Formula,KModelo,Mundo,Asignacion,indef):-
nonvar(Formula),
Formula = nec(Subformula),
extractDomain(KModelo,Dominio),
extractRel(KModelo,Rel),
member(X,Dominio),
member((Mundo,X),Rel),
satisface(Subformula,KModelo,X,Asignacion,indef).

/********** Verificacion de constantes y variables respecto de un mundo. **********/

satisface(X,_,_,_,indef):-
var(X), !.

satisface(X,_,_,_,indef):-
atomic(X), !.

/********** Cuantificador existencial repecto de un mundo. **********/

satisface(Formula,KModelo,Mundo,Asignacion,pos):-
nonvar(Formula),
Formula = some(X,SubFormula),
var(X),
extractWorldModel(KModelo,Mundo,model(Dominio,_)),
member(V,Dominio),
satisface(SubFormula,KModelo,Mundo,[g(X,V)|Asignacion],pos).

satisface(Formula,KModelo,Mundo,Asignacion,neg):-
nonvar(Formula),
Formula = some(X,SubFormula),
var(X),
extractWorldModel(KModelo,Mundo,model(Dominio,_)),
setof(V,member(V,Dominio),Dom),
setof(V,
(
member(V,Dominio),
satisface(SubFormula,KModelo,Mundo,[g(X,V)|Asignacion],neg)
),
Dom
```

```
).

satisface(Formula,KModelo,Mundo,Asignacion,indef):-
nonvar(Formula),
Formula = some(X,SubFormula),
extractWorldModel(KModelo,Mundo,model(Dominio,_)),
(
nonvar(X)
;
var(X),
member(V,Dominio),
satisface(SubFormula,KModelo,Mundo,[g(X,V)|Asignacion],indef)
).

/********** Cuantificador universal respecto de un mundo. **********/

satisface(Formula,KModelo,Mundo,Asignacion,pos):-
nonvar(Formula),
Formula = all(X,SubFormula),
var(X),
extractWorldModel(KModelo,Mundo,model(Dominio,_)),
setof(V,
(
member(V,Dominio),
satisface(SubFormula,KModelo,Mundo,[g(X,V)|Asignacion],pos)
),
Dom),
setof(V,member(V,Dominio),Dom).

satisface(Formula,KModelo,Mundo,Asignacion,neg):-
nonvar(Formula),
Formula = all(X,SubFormula),
var(X),
extractWorldModel(KModelo,Mundo,model(Dominio,_)),
member(V,Dominio),
satisface(SubFormula,KModelo,Mundo,[g(X,V)|Asignacion],neg).

satisface(formula,KModelo,Mundo,Asignacion,indef):-
nonvar(Formula),
Formula = all(X,SubFormula),
extractWorldModel(KModelo,Mundo,model(Dominio,_)),
```

```
(
nonvar(X)
;
member(V,Dominio),
satisface(SubFormula,KModelo,Mundo,[g(X,V)|Asignacion],indef)
).

/*********************************/
/********** **********/
/********** CONECTIVOS **********/
/********** **********/
/*********************************/

/********** Conjuncion respecto de un mundo **********/

satisface(Formula,KModel,Mundo,Asignacion,pos):-
nonvar(Formula),
Formula = and(Formula1,Formula2),
satisface(Formula1,KModel,Mundo,Asignacion,pos),
satisface(Formula2,KModel,Mundo,Asignacion,pos).

satisface(Formula,KModel,Mundo,Asignacion,neg):-
nonvar(Formula),
Formula = and(Formula1,Formula2),
satisface(Formula1,KModel,Mundo,Asignacion,neg),
satisface(Formula2,KModel,Mundo,Asignacion,pos).

satisface(Formula,KModel,Mundo,Asignacion,neg):-
nonvar(Formula),
Formula = and(Formula1,Formula2),
satisface(Formula1,KModel,Mundo,Asignacion,pos),
satisface(Formula2,KModel,Mundo,Asignacion,neg).

satisface(Formula,KModel,Mundo,Asignacion,neg):-
nonvar(Formula),
Formula = and(Formula1,Formula2),
satisface(Formula1,KModel,Mundo,Asignacion,neg),
satisface(Formula2,KModel,Mundo,Asignacion,neg).

satisface(Formula,KModel,Mundo,Asignacion,indef):-
nonvar(Formula),
```

```
Formula = and(Formula1,Formula2),
(
satisface(Formula1,KModel,Mundo,Asignacion,indef)
;
satisface(Formula2,KModel,Mundo,Asignacion,indef)
).

/********** Disyuncion respecto de un mundo. **********/

satisface(Formula,KModel,Mundo,Asignacion,pos):-
nonvar(Formula),
Formula = or(Formula1,Formula2),
satisface(Formula1,KModel,Mundo,Asignacion,pos),
satisface(Formula2,KModel,Mundo,Asignacion,pos).

satisface(Formula,KModel,Mundo,Asignacion,pos):-
nonvar(Formula),
Formula = or(Formula1,Formula2),
satisface(Formula1,KModel,Mundo,Asignacion,neg),
satisface(Formula2,KModel,Mundo,Asignacion,pos).

satisface(Formula,KModel,Mundo,Asignacion,pos):-
nonvar(Formula),
Formula = or(Formula1,Formula2),
satisface(Formula1,KModel,Mundo,Asignacion,pos),
satisface(Formula2,KModel,Mundo,Asignacion,neg).

satisface(Formula,KModel,Mundo,Asignacion,neg):-
nonvar(Formula),
Formula = or(Formula1,Formula2),
satisface(Formula1,KModel,Mundo,Asignacion,neg),
satisface(Formula2,KModel,Mundo,Asignacion,neg).

satisface(Formula,KModel,Mundo,Asignacion,indef):-
nonvar(Formula),
Formula = or(Formula1,Formula2),
(
satisface(Formula1,KModel,Mundo,Asignacion,indef)
;
satisface(Formula2,KModel,Mundo,Asignacion,indef)
).
```

```
/********** Implicacion respecto de un mundo **********/

satisface(Formula,KModel,Mundo,Asignacion,pos):-
nonvar(Formula),
Formula = imp(Formula1,Formula2),
satisface(Formula1,KModel,Mundo,Asignacion,pos),
satisface(Formula2,KModel,Mundo,Asignacion,pos).

satisface(Formula,KModel,Mundo,Asignacion,pos):-
nonvar(Formula),
Formula = imp(Formula1,Formula2),
satisface(Formula1,KModel,Mundo,Asignacion,neg),
satisface(Formula2,KModel,Mundo,Asignacion,pos).

satisface(Formula,KModel,Mundo,Asignacion,neg):-
nonvar(Formula),
Formula = imp(Formula1,Formula2),
satisface(Formula1,KModel,Mundo,Asignacion,pos),
satisface(Formula2,KModel,Mundo,Asignacion,neg).

satisface(Formula,KModel,Mundo,Asignacion,pos):-
nonvar(Formula),
Formula = imp(Formula1,Formula2),
satisface(Formula1,KModel,Mundo,Asignacion,neg),
satisface(Formula2,KModel,Mundo,Asignacion,neg).

satisface(Formula,KModel,Mundo,Asignacion,indef):-
nonvar(Formula),
Formula = imp(Formula1,Formula2),
(
satisface(Formula1,KModel,Mundo,Asignacion,indef)
;
satisface(Formula2,KModel,Mundo,Asignacion,indef)
).

/********** Doble Implicacion respecto de un mundo. **********/

satisface(Formula,KModel,Mundo,Asignacion,pos):-
nonvar(Formula),
Formula = iff(Formula1,Formula2),
```

```
satisface(Formula1,KModel,Mundo,Asignacion,pos),
satisface(Formula2,KModel,Mundo,Asignacion,pos).

satisface(Formula,KModel,Mundo,Asignacion,neg):-
nonvar(Formula),
Formula = iff(Formula1,Formula2),
satisface(Formula1,KModel,Mundo,Asignacion,neg),
satisface(Formula2,KModel,Mundo,Asignacion,pos).

satisface(Formula,KModel,Mundo,Asignacion,neg):-
nonvar(Formula),
Formula = iff(Formula1,Formula2),
satisface(Formula1,KModel,Mundo,Asignacion,pos),
satisface(Formula2,KModel,Mundo,Asignacion,neg).

satisface(Formula,KModel,Mundo,Asignacion,pos):-
nonvar(Formula),
Formula = iff(Formula1,Formula2),
satisface(Formula1,KModel,Mundo,Asignacion,neg),
satisface(Formula2,KModel,Mundo,Asignacion,neg).

satisface(Formula,KModel,Mundo,Asignacion,indef):-
nonvar(Formula),
Formula = iff(Formula1,Formula2),
(
satisface(Formula1,KModel,Mundo,Asignacion,indef)
;
satisface(Formula2,KModel,Mundo,Asignacion,indef)
).

/********** Negacion respecto de un mundo. **********/

satisface(Formula,KModel,Mundo,Asignacion,pos):-
nonvar(Formula),
Formula = not(SubFormula),
satisface(SubFormula,KModel,Mundo,Asignacion,neg).

satisface(Formula,KModel,Mundo,Asignacion,neg):-
nonvar(Formula),
Formula = not(SubFormula),
satisface(SubFormula,KModel,Mundo,Asignacion,pos).
```

```
satisface(Formula,KModel,Mundo,Asignacion,indef):-
nonvar(Formula),
Formula = not(SubFormula),
satisface(SubFormula,KModel,Mundo,Asignacion,indef).

/*********************************/
/********** **********/
/********** RELACIONES **********/
/********** **********/
/*********************************/

/********** Igualdad respecto de un mundo. **********/

satisface(Formula,KModel,Mundo,Asignacion,pos):-
nonvar(Formula),
Formula = eq(X,Y),
interpreta(X,KModel,Mundo,Asignacion,Valor1),
interpreta(Y,KModel,Mundo,Asignacion,Valor2),
Valor1=Valor2.

satisface(Formula,KModel,Mundo,Asignacion,neg):-
nonvar(Formula),
Formula = eq(X,Y),
interpreta(X,KModel,Mundo,Asignacion,Valor1),
interpreta(Y,KModel,Mundo,Asignacion,Valor2),
\+ Valor1=Valor2.

satisface(Formula,KModel,Mundo,Asignacion,indef):-
(
nonvar(Formula),
Formula = eq(X,Y),
\+ interpreta(X,KModel,Mundo,Asignacion,_)
);
(
nonvar(Formula),
Formula = eq(X,Y),
\+ interpreta(Y,KModel,Mundo,Asignacion,_)
).

/********** Relaciones de orden uno respecto de un mundo. **********/
```

```
satisface(Formula,KModel,Mundo,Asignacion,pos):-
nonvar(Formula),
descomponer(Formula,Simbolo,[Argumento]),
\+ member(Simbolo,[not,poss,nec]),
interpreta(Argumento,KModel,Mundo,Asignacion,Valor),
extractWorldModel(KModel,Mundo,model(_,F)),
member(f(1,Simbolo,ValoresP),F),
member(Valor,ValoresP).

satisface(Formula,KModel,Mundo,Asignacion,neg):-
nonvar(Formula),
descomponer(Formula,Simbolo,[Argumento]),
\+ member(Simbolo,[not,poss,nec]),
interpreta(Argumento,KModel,Mundo,Asignacion,Valor),
extractWorldModel(KModel,Mundo,model(_,F)),
member(f(1,Simbolo,ValoresP),F),
\+ member(Valor,ValoresP).

satisface(Formula,KModel,Mundo,Asignacion,indef):-
nonvar(Formula),
descomponer(Formula,Simbolo,[Argumento]),
\+ member(Simbolo,[not,poss,nec]),
extractWorldModel(KModel,Mundo,model(_,F)),
(
\+ var(Argumento),
\+ atom(Argumento)
;
var(Argumento),
\+ interpreta(Argumento,KModel,Mundo,Asignacion,_)
;
atom(Argumento),
\+ interpreta(Argumento,KModel,Mundo,Asignacion,_)
;
\+ member(f(1,Simbolo,_),F)
).

/********** Relaciones binarias respecto de un mundo. **********/

satisface(Formula, KModelo,Mundo,Asignacion,pos):-
nonvar(Formula),
```

```
descomponer(Formula,Simbolo,[Argumento1,Argumento2]),
\+ member(Simbolo,[eq,and,or,imp,iff,some,all]),
interpreta(Argumento1,KModelo,Mundo,Asignacion,Valor1),
interpreta(Argumento2,KModelo,Mundo,Asignacion,Valor2),
extractWorldModel(KModelo,Mundo,model(_,F)),
member(f(2,Simbolo,ValoresP),F),
member((Valor1,Valor2),ValoresP).

satisface(Formula,KModelo,Mundo,Asignacion,neg):-
nonvar(Formula),
descomponer(Formula,Simbolo,[Argumento1,Argumento2]),
\+ member(Simbolo,[eq,and,or,imp,iff,some,all]),
interpreta(Argumento1,KModelo,Mundo,Asignacion,Valor1),
interpreta(Argumento2,KModelo,Mundo,Asignacion,Valor2),
extractWorldModel(KModelo,Mundo,model(_,F)),
member(f(2,Simbolo,ValoresP),F),
\+ member((Valor1,Valor2),ValoresP).

satisface(Formula,KModelo,Mundo,Asignacion,indef):-
nonvar(Formula),
descomponer(Formula,Simbolo,[Argumento1,Argumento2]),
\+ member(Simbolo,[eq,and,or,imp,iff,some,all]),
extractWorldModel(KModelo,Mundo,model(_,F)),
(
\+ var(Argumento1),
\+ atom(Argumento1)
;
\ var(Argumento2),
\ atom(Argumento2)
;
var(Argumento1),
\+ interpreta(Argumento1,KModelo,Mundo,Asignacion,_)
;
var(Argumento2),
\+ interpreta(Argumento2,KModelo,Mundo,Asignacion,_)
;
atom(Argumento1),
\+ interpreta(Argumento1,KModelo,Mundo,Asignacion,_)
;
atom(Argumento2),
\+ interpreta(Argumento2,KModelo,Mundo,Asignacion,_)
```

```
;
\+ member(f(2,Simbolo,_),F)
).

/********** Relaciones ternarias respecto de un mundo. **********/

satisface(Formula,KModelo,Mundo,Asignacion,pos):-
nonvar(Formula),
descomponer(Formula,Simbolo,[Argumento1,Argumento2,Argumento3]),
interpreta(Argumento1,KModelo,Mundo,Asignacion,Valor1),
interpreta(Argumento2,KModelo,Mundo,Asignacion,Valor2),
interpreta(Argumento3,KModelo,Mundo,Asignacion,Valor3),
extractWorldModel(KModelo,Mundo,model(_,F)),
member(f(3,Simbolo,ValoresP),F),
member((Valor1,Valor2,Valor3),ValoresP).

satisface(Formula,KModelo,Mundo,Asignacion,neg):-
nonvar(Formula),
descomponer(Formula,Simbolo,[Argumento1,Argumento2,Argumento3]),
interpreta(Argumento1,KModelo,Mundo,Asignacion,Valor1),
interpreta(Argumento2,KModelo,Mundo,Asignacion,Valor2),
interpreta(Argumento3,KModelo,Mundo,Asignacion,Valor3),
extractWorldModel(KModelo,Mundo,model(_,F)),
member(f(3,Simbolo,ValoresP),F),
\+ member((Valor1,Valor2,Valor3),ValoresP).

satisface(Formula,KModelo,Mundo,Asignacion,pos):-
nonvar(Formula),
descomponer(Formula,Simbolo,[Argumento1,Argumento2,Argumento3]),
extractWorldModel(KModelo,Mundo,model(_,F)),
(
\+ var(Argumento1),
\+ atom(Argumento1)
;
\+ var(Argumento2),
\+ atom(Argumento2)
;
\+ var(Argumento3),
\+ atom(Argumento3)
;
var(Argumento1),
```

```
\+ interpreta(Argumento1,KModelo,Mundo,Asignacion,_)
;
var(Argumento2),
\+ interpreta(Argumento2,KModelo,Mundo,Asignacion,_)
;
var(Argumento3),
\+ interpreta(Argumento3,KModelo,Mundo,Asignacion,_)
;
atom(Argumento1),
\+ interpreta(Argumento1,KModelo,Mundo,Asignacion,_)
;
atom(Argumento2),
\+ interpreta(Argumento2,KModelo,Mundo,Asignacion,_)
;
atom(Argumento3),
\+ interpreta(Argumento3,KModelo,Mundo,Asignacion,_)
;
\+ member(f(3,Simbolo,_),F)
).

/********** Interpretacion de terminos en un mundo **********/

interpreta(Termino,KModel,Mundo,G,Valor):-
(
var(Termino),
member(g(Y,Valor),G),
Y == Termino,
!,
extractWorldModel(KModel,Mundo,model(D,F)),
member(Valor,D)
;
atom(Termino),
extractWorldModel(KModel,Mundo,model(D,F)),
member(f(0,Termino,Valor),F)
).
```

Apéndice E

E.1.

Lo que sigue es el código del archivo **modelosEjemplo.pl**

```
:- module(modelosEjemplo,[ejemplo/2]).

% Ejemplo: Hospital

ejemplo(1,
    kmodel([w1,w2,w3,w4,w5,w6,w7,w8],
    [(w1,w2),(w1,w3),(w1,w4),(w2,w4),(w3,w4),(w4,w5),(w4,w6),(w4,w7),(w5,w7),(w6,w7),(w7,w8)],
    [w1(model([a,b,c,d,e,f,he1,he2,she1,she2,she3,m1,m2,m3,m4],
    [f(0,alis,a),f(0,brad,b),f(0,cindy,c),f(0,donna,d),f(0,elliot,e),f(0,fina,f),
    f(1,woman,[a,c,d,f,she1,she2,she3]),f(1,man,[b,e,he1,he2,he3]),
    f(1,nurse,[c,f,she1]),f(1,medic,[d,e,he2]),f(1,patient,[a,he1,he3,she2]),
    f(1,medicine,[m1,m2,m3,m4]),f(1,syrup,[m1]),f(1,injection,[m2]),f(1,pill,[m3,m4]),
    f(1,wounded,[a,she2]),f(1,healthy,[he3]),f(1,ill,[he1]),
    f(2,examine,[(d,a),(e,he3)]),f(2,operate,[(he2,she2)]),
    f(3,prescribe,[]),f(2,inject,[(f,he1)]),
    f(3,give,[(c,she1,m4)])
    ])),
    w2(model([a,b,c,d,e,f,he1,he2,she1,she2,she3,m1,m2,m3,m4],
    [f(0,alis,a),f(0,brad,b),f(0,cindy,c),f(0,donna,d),f(0,elliot,e),f(0,fina,f),
    f(1,woman,[a,c,d,f,she1,she2,she3]),f(1,man,[b,e,he1,he2,he3]),
    f(1,nurse,[c,f,she1]),f(1,medic,[d,e,he2]),f(1,patient,[a,he1,he3,she2]),
    f(1,medicine,[m1,m2,m3,m4]),f(1,syrup,[m1]),f(1,injection,[m2]),f(1,pill,[m3,m4]),
    f(1,wounded,[a,she2]),f(1,healthy,[he3]),f(1,ill,[he1]),
    f(2,examine,[(e,he3)]),f(2,operate,[(he2,she2)]),
    f(3,prescribe,[(d,a,m4)]),f(2,inject,[(f,he1)]),
    f(3,give,[(c,she1,m4)])
    ])),
```

```
w3(model([a,b,c,d,e,f,he1,he2,she1,she2,she3,m1,m2,m3,m4],
[f(0,alis,a),f(0,brad,b),f(0,cindy,c),f(0,donna,d),f(0,elliot,e),f(0,fina,f),
f(1,woman,[a,c,d,f,she1,she2,she3]),f(1,man,[b,e,he1,he2,he3]),
f(1,nurse,[c,f,she1]),f(1,medic,[d,e,he2]),f(1,patient,[a,he1,he3,she2]),
f(1,medicine,[m1,m2,m3,m4]),f(1,syrup,[m1]),f(1,injection,[m2]),f(1,pill,[m3,m4]),
f(1,wounded,[a,she2]),f(1,healthy,[he3]),f(1,ill,[he1]),
f(2,examine,[(d,a)]),f(2,operate,[(he2,she2)]),
f(2,prescribe,[(e,he3,m3)]),f(2,inject,[(f,he1)]),
f(3,give,[(c,she1,m4)])
])),
w4(model([a,b,c,d,e,f,he1,he2,she1,she2,she3,m1,m2,m3,m4],
[f(0,alis,a),f(0,brad,b),f(0,cindy,c),f(0,donna,d),f(0,elliot,e),f(0,fina,f),
f(1,woman,[a,c,d,f,she1,she2,she3]),f(1,man,[b,e,he1,he2,he3]),
f(1,nurse,[c,f,she1]),f(1,medic,[d,e,he2]),f(1,patient,[a,he1,he3,she2]),
f(1,medicine,[m1,m2,m3,m4]),f(1,syrup,[m1]),f(1,injection,[m2]),f(1,pill,[m3,m4]),
f(1,wounded,[a,she2]),f(1,healthy,[he3]),f(1,ill,[he1]),
f(2,examine,[]),f(2,operate,[(he2,she2)]),
f(2,prescribe,[(d,a,m1),(e,he3,m3)]),f(2,inject,[(f,he1)]),
f(3,give,[(c,she1,m4)])
])),
w5(model([a,b,c,d,e,f,he1,he2,she1,she2,she3,m1,m2,m3,m4],
[f(0,alis,a),f(0,brad,b),f(0,cindy,c),f(0,donna,d),f(0,elliot,e),f(0,fina,f),
f(1,woman,[a,c,d,f,she1,she2,she3]),f(1,man,[b,e,he1,he2,he3]),
f(1,nurse,[c,f,she1]),f(1,medic,[d,e,he2]),f(1,patient,[a,he1,he3,she2]),
f(1,medicine,[m1,m2,m3,m4]),f(1,syrup,[m1]),f(1,injection,[m2]),f(1,pill,[m3,m4]),
f(1,wounded,[a,she2]),f(1,healthy,[he3]),f(1,ill,[he1]),
f(2,examine,[(d,a),(e,he3)]),f(2,operate,[(he2,she2)]),
f(2,prescribe,[]),f(2,inject,[(f,he1)]),
f(3,give,[(c,she1,m4)])
])),
w6(model([a,b,c,d,e,f,he1,he2,she1,she2,she3,m1,m2,m3,m4],
[f(0,alis,a),f(0,brad,b),f(0,cindy,c),f(0,donna,d),f(0,elliot,e),f(0,fina,f),
f(1,woman,[a,c,d,f,she1,she2,she3]),f(1,man,[b,e,he1,he2,he3]),
f(1,nurse,[c,f,she1]),f(1,medic,[d,e,he2]),f(1,patient,[a,he1,he3,she2]),
f(1,medicine,[m1,m2,m3,m4]),f(1,syrup,[m1]),f(1,injection,[m2]),f(1,pill,[m3,m4]),
f(1,wounded,[a,she2]),f(1,healthy,[he3]),f(1,ill,[he1]),
f(2,examine,[(d,a),(e,he3)]),f(2,operate,[(he2,she2)]),
f(2,prescribe,[]),f(2,inject,[(f,he1)]),
f(3,give,[(c,she1,m4)])
])),
w7(model([a,b,c,d,e,f,he1,he2,she1,she2,she3,m1,m2,m3,m4],
```

```
    [f(0,alis,a),f(0,brad,b),f(0,cindy,c),f(0,donna,d),f(0,elliot,e),f(0,fina,f),
    f(1,woman,[a,c,d,f,she1,she2,she3]),f(1,man,[b,e,he1,he2,he3]),
    f(1,nurse,[c,f,she1]),f(1,medic,[d,e,he2]),f(1,patient,[a,he1,he3,she2]),
    f(1,medicine,[m1,m2,m3,m4]),f(1,syrup,[m1]),f(1,injection,[m2]),f(1,pill,[m3,m4]),
    f(1,wounded,[a,she2]),f(1,healthy,[he3]),f(1,ill,[he1]),
    f(2,examine,[]),f(2,operate,[(he2,she2)]),
    f(2,prescribe,[]),f(2,inject,[(f,he1)]),
    f(3,give,[(d,a,m4),(e,he3,m3)])
    ])),
    w8(model([a,b,c,d,e,f,he1,he2,she1,she2,she3,m1,m2,m3,m4],
    [f(0,alis,a),f(0,brad,b),f(0,cindy,c),f(0,donna,d),f(0,elliot,e),f(0,fina,f),
    f(1,woman,[a,c,d,f,she1,she2,she3]),f(1,man,[b,e,he1,he2,he3]),
    f(1,nurse,[c,f,she1]),f(1,medic,[d,e,he2]),f(1,patient,[a,he1,he3,she2]),
    f(1,medicine,[m1,m2,m3,m4]),f(1,syrup,[m1]),f(1,injection,[m2]),f(1,pill,[m3,m4]),
    f(1,wounded,[a,she2]),f(1,healthy,[he3]),f(1,ill,[he1]),
    f(2,examine,[(d,a),(e,he3)]),f(2,operate,[(he2,she2)]),
    f(2,prescribe,[]),f(2,inject,[]),
    f(3,give,[])
    ]))
    ]
    )
    ).

ejemplo(2,
    kmodel(
    % Conjunto de Mundos:
    [k1,k2,k3],
    % Relación de Accesibilidad:
    [(k1,k2),(k2,k3),(k3,k1)],
    % Conjunto de Modelos de Primer Orden:
    [k1(model(
    %Dominio del contexto k1
    [a,b,c],
    %Función de Interpretación para k1
    [f(0,ann,a),
    f(0,bob,b),
    f(0,cat,c),
    f(1,woman,[a,c]),
    f(1,man,[b]),
    f(1,dancer,[c])
    ]
```

```
    )
    ),
    k2(model(
     %Dominio del contexto k2
    [a,b,c],
     %Función de Interpretación para k2
    [f(0,ann,a),
    f(0,bob,b),
    f(0,cat,c),
    f(1,woman,[a,c]),
    f(1,man,[b]),
    f(1,dancer,[a])
    ]
    )
    ),
    k3(model(
     %Dominio del contexto k3
    [a,b,c],
     %Función de Interpretación para k3
    [f(0,ann,a),
    f(0,bob,b),
    f(0,cat,c),
    f(1,woman,[a,c]),
    f(1,man,[b]),
    f(1,dancer,[b,c])
    ]
    )
    )
    ]
    )
    ).

%Ejemplo: Rigid Designation Test over S5

ejemplo(3,
    kmodel([w1,w2,w3],
    [(w1,w1),(w1,w2),(w1,w3),(w2,w1),(w2,w2),(w2,w3),(w3,w1),(w3,w2),(w3,w3)],
    [w1(model([a,b,c],
    [f(0,anna,a),
    f(0,bill,b),
    f(0,cat,c),
```

```
f(1,man,[b]),
f(1,woman,[a,c]),
f(1,dance,[a,b,c]),
f(1,rest,[])
])),
w2(model([a,b,c],
[f(0,alex,a),
f(0,bred,b),
f(0,caitlin,c),
f(1,man,[a,b]),
f(1,woman,[c]),
f(1,dance,[a,c]),
f(1,rest,[b])
])),
w3(model([a,b,c],
[f(0,andrew,a),
f(0,beth,b),
f(0,casey,c),
f(1,man,[a]),
f(1,woman,[b,c]),
f(1,dance,[b,c]),
f(1,rest,[])
]))
]
)
).
```

Bibliografía

[BB05] Patrick Blackburn and Johan Bos. *Representation and Inference for Natural Language. A First Course in Computational Semantics*. CSLI Publications, 2005.

[BBS06] Patrick Blackburn, Johan Bos, and Kristina Striegnitz. *Learn Prolog Now!* College Publications, 2006.

[BRV01] P. Blackburn, M. De Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

[CMG92] Gennaro Chierchia and Sally McConnell-Ginet. *Meaning and Grammar. An Introduction to Semantics*. MIT Press, 1992.

[Cru00] Alan Cruse. *Meaning in Language. An Introduction to Semantics and Pragmatics*. Oxford University Press, 2000.

[DWP81] D.R. Dowty, R.E. Wall, and S. Peters. *Introduction to Montague Semantics*. Dordrecht: Reidel, 1981.

[EU10] Jan van Eijck and Christina Unger. *Computational Semantics with Functional Programming*. Cambridge University Press, 2010.

[Gam1a] L.T.F. Gamut. *Logic, Language, and Meaning. VolumeI. Introduction to Logic*. Chicago and London: The University of Chicago Press, 1991a.

[Gam1b] L.T.F. Gamut. *Logic, Language, and Meaning. VolumeII. Intensional Logic and Logical Grammar*. Chicago and London: The University of Chicago Press, 1991b.

[GM89] G. Gazdar and C. Mellish. *Natural Language Processing in Prolog*. Addison-Wesley Publishing Company, 1989.

[JM00] D. Jurafsky and J. Martin. *Speech and Language Processing*. Prentice-Hall, 2000.

[Pri01] Graham Priest. *An Introduction to Non-Classical Logic*. Cambridge University Press, 2001.

[Sha99] S. Shapiro. *Foundations Without Foundationalism: A Case for Second-order Logic*. Oxford University Press, 1999.

[VF09] L.M. Villegas and Max Fernández. *Lógica Matemática I*. UAMI, 2009.

[VF11] L.M. Villegas and Max Fernández. *Lógica Matemática II*. UAMI, 2011.