DESDE EL CÁLCULO DE PREDICADOS A LAS GRAMÁTICAS MODULARES

Antonio Menchén / Víctor Díaz

Área de Lenguajes y Sistemas Informáticos
Facultad de Informática y Estadística
Universidad de Sevilla

Resumen

Las Gramáticas Lógicas Modulares (MLGs), se corresponden con las Gramáticas Atribuidas del modelo no-lógico, y permiten haciendo uso de una serie de operadores y símbolos terminales lógicos, obtener a partir del árbol de análisis de una frase su forma lógica.

Hemos observado que puede construirse, desde PROLOG , un sistema de Procesamiento del Lenguaje Natural, modular y sistemático, tratándose en este trabajo de aplicarlo a los problemas propios del castellano, y mostrando a través del acceso a bases de datos relacionales la forma de utilizarlo en otros entornos.

Este trabajo se divide en cuatro partes: CALCULO DE PREDICADOS Y REPRESENTACION SEMANTICA, GRAMATICAS LOGICAS, IMPLEMENTACION DE LAS MLGs y MEJORAS DEL FORMALISMO. En la primera se hace un planteamiento del problema que nos ocupa: representación del lenguaje natural en un lenguaje de ordenador, y constituye la justificación de la segunda y tercera partes. En la segunda se muestra el progreso que trae la utilización de una MLG frente a una Gramática de Clásulas Definidas (DCG), en la tercera se hace una exposición detallada de todos los problemas que plantea la implementación de estos formalismos y en la cuarta se hace una propuesta de futuras investigaciones en este terreno.

1. Cálculo de Predicados y Representación Semántica

Es evidente que sólo una parte de nuestro conocimiento se trasmite a través del lenguaje natural. Siempre han existido un gran número de lenguajes, que llamaremos formales, desprovistos de ambigüedades y que obedecen a reglas estrictas de construcción. Entre estos lenguajes están los de programación en los que habremos de 'convertir' cualquier conocimiento que se desee procesar.

En [1] se apunta que los humanos tenemos una gran capacidad de desambigüación y que esto es debido, simplemente, a que implícitamente poseemos un conocimiento (más o menos exacto, por otra parte), del contexto en que debe de entenderse un párrafo, una frase o una palabra.

Parece pues plausible que tomando un lenguaje formal (conveniente), que represente una frase en LN y añadiéndole la representación del contexto, podremos disponer de programas que: permitan acceder a datos, a bases de conocimiento, a entornos de sistemas operativos, etc, sin que el usuario tenga que conocer SQL, PROLOG o UNIX, pongamos por caso.

Tomando como lenguaje formal el cálculo de predicados, llamaremos estructura lógica a la representación en este lenguaje de la frase sin tener en cuenta el contexto. El lenguaje de programación que utilizaremos para procesar la frase es PROLOG.

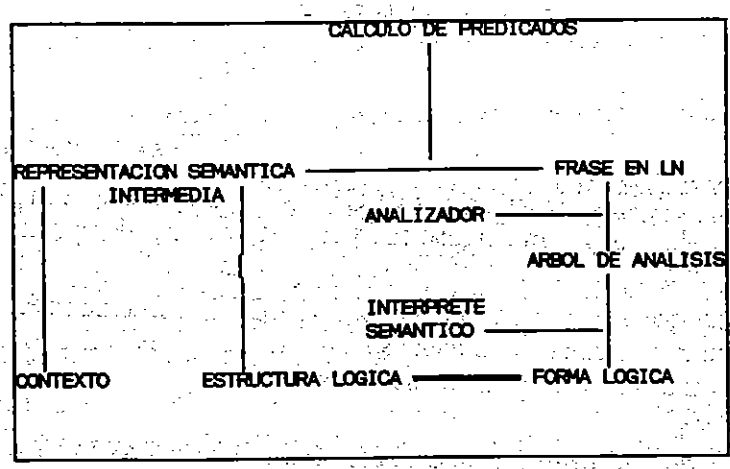
La forma lógica es la representación en PROLOG de la estructura lógica.

Además PROLOG presenta la ventaja de que pueden implementarse fácilmente bases de datos relacionales en él.

Necesitamos, pues, un programa PROLOG que a partir de una sentencia en LN obtenga su forma lógica.

Ese programa se puede obtener de la gramática del lenguaje escrita mediante símbolos no-terminales que son predicados con argumentos, es decir, mediante una gramática lógica, la cual se traduce a un programa PROLOG que analice las sentencias aplicándose otro programa que se denomina «compilador de reglas», obteniéndose por fin el árbol de análisis. Este árbol es una estructura que representa, bien el árbol sintáctico, bien la forma lógica directamente o bien estas dos informaciones aunque de una forma implícita.

Todo lo dicho se resume en el siguiente cuadro:



que se completa con:

1.1. El Lenguaje del Cálculo de Predicados

Es el lenguaje que representa fórmulas lógicas. Esencialmente son nombres de predicados, que representan las relaciones entre otros predicados, variables o constantes, unidos por las llamadas conectivas:

$$\neg, \wedge, \vee, \rightarrow, \equiv$$

Los paréntesis encierran a las expresiones que están relacionadas y separan las distintas fórmulas lógicas, y las comas separan a las expresiones.

Además dispondremos de los cuantificadores:

$$E\ 'A$$

Hay otros elementos que no son de lógica de primer orden pero que se utilizarán para poder dar el sentido correcto a las frases: notación tipada, términos proposicionales y cuantificadores generalizados.

La notación tipada nos permite más poder de expresión del cálculo de predicados de primer orden, para aplicarlo al LN:

De la fórmula:

$$(\forall x)\ (\exists y)\ q(x,y)$$

es tipada como:

$$(\forall x | t1(x))\ (\exists y | t2(y))\ q(x,y)$$

donde se quiere significar que para todos los x de un dominio t1, existe al menos un y de un dominio t2 que verifican la relación q.

Esto hace que sea más precisa la fórmula lógica expresando el significado. Por ejemplo, la frase: Cada hombre ama a una *mujer*, se representa:

$$(\forall x | man(x))\ (\exists y | woman(y))\ loves(x,y)$$

(Como puede observarse a los predicados se les ha dado el nombre en inglés para representar, el significado de la palabra más que la palabra en un idioma determinado.)

Puede verse la equivalencia de las fórmulas que siguen y se aplicarán en el resto del trabajo:

$$(\forall x | t1(X))\ p(x) \equiv (\forall x)\ (t1(x) \rightarrow p(x))$$

$$(\exists x | t1(x))\ p(x) \equiv (\exists x)\ (t1(x) \wedge p(x))$$

Esto permite tratar la fórmula inicial como su equivalente:

$$(\forall x)\ (t1(x) \rightarrow (\exists y)\ (t2(y) \wedge q(x,y)))$$

Así la frase: Cada hombre ama *a una mujer*, se representa:

$$(\forall x)\ (man(x) \rightarrow (\exists y)\ (woman(y) \wedge loves(x,y)))$$

1.2 Representación en Estructuras Lógicas

En [2] se enuncia «un*a frase en lenguaje natural está formada gramaticalmente por un verbo y un cierto número de sintagmas nominales*».

Nos basaremos en el siguiente principio de asociación: A toda estructura gramatical se le puede asociar una estructura lógica estableciendo las dos transformaciones:

a) El verbo por un predicado.

b) Cada sintagma nominal por una cuantificación restringida al contenido de ese sintagma.

La frase: Un p*erro persigue a un gato*, $(\exists x | dog(x))\ (\exists y | cat(y))\ follows(x,y)$

que es equivalente a: $(\exists x)\ (dog(x) \wedge (\exists y)\ (cat(y) \wedge follows(x,y)))$

De esta forma, de una manera más o menos afortunada, podremosrepresentar todo tipo de frases:

Así la frase: García *trabaja en todos los proyectos dirigidos por López*, queda representada:

$$(\forall x)\ (project(x) \wedge leads(LOPEZ_1, x) \rightarrow works_{on}(GARCIA_1, x))$$

Esta representación distingue entre «*proposición*» y «*descripción*». El verbo expresa, principalmente, una proposición que puede ser afirmada, negada, o ser la base de una pregunta o una orden, el sintagma nominal expresa una descripción de un conjunto de objetos.

1.3 Equivalencia entre Estructura Lógica y Forma Lógica

Según hemos visto ya, un determinante que se corresponde perfectamente con el cuantificador universal es cada.

La representación:

$$(\forall x)\ (man(x) \rightarrow (\exists y)\ (woman(y) \wedge loves(x,y))$$

Vamos a ponerla de la forma:

$$(\forall x)\ (P)$$

donde x es una variable sobre la que se cuantifica y P es la proposición que es cierta para todo valor de x. Sabemos que si P tiene la forma:

$$A \rightarrow B$$

su equivalente es:

$$\neg A \vee B$$

que equivale a:

$$\neg (A \wedge \neg B)$$

En [3], McCord propone como forma lógica de la anterior frase

each(man(X), ex(woman(Y),loves(X,Y))).

donde each(P,Q) :- not((P , not(Q))).

Con lo que está perfectamente justificado el decir que la forma lógica anterior representa la estructura lógica de la frase en LN.

De igual manera, para el cuantificador existencial, se obtendría:

ex(P,Q) :- P , Q.

1.4 Formas Lógicas asociadas a distintas Categorías Gramaticales

Tal como se apunta en [4], se llama forma lógica, LF, a la representación del significado de una frase en un lenguaje lógico. Además, se dice algo muy importante: la *estructura de la frase en subfrases, determina la construcción de la forma lógica en subformas lógicas*. Sin darnos cuenta hemos dado un paso de gigante en cuanto a la implementación, ya que a partir de ahora prescindiremos de la representación en el cálculo de predicados (aunque siempre partiremos de él), traduciendo directamente una frase a su forma lógica. Ahora estamos en disposición de escribir la gramática de un LN y obtener por el principio de composición de formas lógicas, la forma lógica de una frase, pero este es un tema que dejaremos para más adelante.

Cuantificadores como muchos, unos pocos o la mayoría pueden tratarse de la misma forma que cada o uno, sin más que definir convenientemente su forma lógica.

La frase: Unos *pocos tigres son mansos*, tendría como forma lógica:

few(tiger(X),tame(X)).

Al primer argumento, del predicado, se le llama base: especifica la base o el rango de items sobre lo que se cuantifica. Al segundo se le llama foco: especifica aquella parte de la sentencia sobre la que se hace énfasis (en los adverbios se verá más claro este segundo concepto).

Además se verifica:

```
few(Base,Foco):-
card(Base,B),
card((Base,Foco),BF),
small(BF,B).
```

donde:

```
card(P,N):-
setof(P,P,S),
length(S,N).
```

El predicado setof(X,P,S) nos devuelve en S el conjunto ordenado (lista) de todos los términos X que hacen que P se cumpla, y length(S,N) devuelve en N el número de elementos de S. El predicado small(M,N) se cumple para valores de M/N menores a uno dado.

Con muchos se realizaría algo análogo. La diferencia estaría en el valor límite de la fracción anterior. Para el cuantificador muchos, es necesario tener datos con que comparar este valor, ya que es imposible determinar en valor absoluto el límite de la fracción M/N.

Otro cuantificador que aparece a menudo es el determinante definido. Su forma lógica será:

the(P,Q):- card(P,1), P, Q.

En cuanto a los adverbios, si tomamos la frase: Juan compró ayer un gato, el adverbio nos dice el tiempo de validez de la sentencia.

En [3] la traducción a forma lógica es:

yesterday(ex(cat(X),buy(juan,X))),

Luego el argumento es una forma lógica. Cuando esto ocurre se dice que la categoría gramatical es intencional (en oposición a extensional). La mayoría de los adverbios tienen dos argumentos, por lo que la forma lógica es:

adverbio(Base,Foco).

En frases adverbiales aparece con frecuencia el fenómeno de la **enfatización**, su importancia es tal, que determina en gran medida la forma lógica.

Así la frase: Ju*an siempre compra libros en el Corte Inglés*, si se enfatiza dónde compra los libros, tiene como forma lógica:

always((book(X),buy(juan,X)):E, at(corte_ingles,E)).

Mientras que, si se enfatiza qué compra, dará la forma lógica:

always(at(corte_ingles,buy(juan,X)), book(X)).

La mayoría de los adjetivos son extensionales, y son semánticamente nombres:
Así la frase: Juan ve una casa ro*ja*, tiene como forma lógica:

ex((house(X),red(X)), see(juan,X)).

Otros adjetivos son intencionales: anterior, futuro, falso, verdadero, etc. No modifican entidades, como en el caso anterior, sino a tipos de entidades (No se puede decir anterior(X) o falso(X), siendo X una entidad)

Como ejemplo, tenemos la frase: Juan *se encuentra con su anterior maestro*, dando como forma lógica:

ex(former(teacher(X,juan)), meet(juan,X)).

1.5 El Conocimiento del Contexto

En [4] se define la representación semántica intermedia como una combinación de elementos que son necesarios para la interpretación semántica. Esta representación semántica intermedia estaría formada por las siguientes partes:

La estructura lógica, que se corresponde con la forma lógica de [3].
El contenido conceptual.
El acto de discurso.
Las anotaciones pragmáticas.

1.5.1. El Contenido Conceptual

Tomemos la frase: J*uan lee un libro*, cuya forma lógica es:

ex(book(X),read(juan,X)).

Es evidente que Juan puede leer un libro perfectamente (a menos que sea analfabeto), pero lo que no está claro, al menos en un contexto normal, es que el libro pueda leer a Juan. Este conocimiento es posible añadirlo a la estructura lógica. Vamos a proponer una forma sistemática y que servirá para la mayoría de los casos.

En [2] se propone que cualquiera que sea el dominio se haga uso de estos dos predicados:

instance(<objeto>,<clase>),
propval(<objeto>,<propiedad>,<valor>).

El segundo predicado nos va a servir para indicar que es Juan quien lee, y el libro el objeto leído:

propval(r,agent,JUAN_1),
propval(r,object,x).

donde se ha realizado la abstracción instance(r,read): r es la instancia de read.

Con esta notación la representación intermedia queda:

$$(\exists x|instance(x, book))(\exists r|instance(r, read)), propval(r, agent, JUAN), propval(r, object, x)$$

Estableciendo un sistema de jerarquía de clases dotadas de propiedades que heredan unas de otras, se podrá deducir que el agente de una instancia de «read» es una persona, porque «read» es una subclase de «process» y una propiedad de «process» es «agent» que tiene como tipo de valor «person».

En [3], McCord implementa esta información conceptual en lo que llama ranura (slot), exigiendo, por ejemplo, que el agente del verbo give sea humano, y que un hombre (como el caso de Juan), tenga como tipo semántico el ser humano.

Por lo tanto allí no se puede hablar de representación intermedia desde el momento en que el contenido conceptual no se interpreta, sino que son restricciones semánticas dentro de la gramática.

1.5.2 Actos de Discurso

Supone el incorporar a la representación intermedia que se obtiene de la estructura lógica y los contenidos conceptuales, la intención del locutor al emitir una frase, de esta forma podemos tratar más adecuadamente preguntas u órdenes. En [3], el acto de discurso se incorpora a la forma lógica. Así una pregunta como: *¿Cada hombre ama a María?*, da la forma lógica:

yesno(each(man(X), E), love(X,mary):E).

1.5.3. Anotaciones Pragmáticas

Permiten resolver ambigüedades que aun quedan para entender el sentidode la frase. Pueden referirse a la cuantificación, a informaciones que completan el contexto o, a las que sitúan el tiempo o el modo.

2. Gramáticas Lógicas

Una característica muy apreciable de la programación lógica es que las gramáticas para los lenguajes (naturales y formales), pueden ser expresadas fácilmente como programas lógicos y que pueden ser ejecutados para el análisis y la síntesis de estos lenguajes. Dentro de esta idea PROLOG resulta ser el lenguaje lógico más indicado.

2.1. Gramáticas Lógicas Modulares (MLGs)

Las gramáticas lógicas más extendidas y que se encuentran implementadas en cualquier versión de PROLOG, son las llamadas Gramáticas de Cláusulas Definidas (DCGs). Consisten en una generalización de las gramáticas de estructura de frase libre de contexto, en la que los símbolos no-terminales pueden contener argumentos, donde, posteriormente, se obtendrán las estructuras de árbol sintáctico o representación semántica.

Las DCGs presentan el problema de la imposibilidad de separar el árbol de análisis de la representación semántica de la frase. Para el LN, donde la acción semántica depende muchas veces del contexto o del énfasis, este formalismo se muestra poco adecuado y potente. El problema que se presenta a menudo es que la conexión entre la estructura lógica y la regla de producción es muy remota y complicada, por lo que diseñar el analizador a partir de una DCG es imposible.

Se hace necesario separar el componente sintáctico del semántico, para poderlos manipular por separado.

En las Gramáticas Lógicas Modulares (MLGs), debidas a Michael McCord, del Watson Research Center, las reglas gramaticales son:

```
S→a : OP-LF
```

que se traduce en la separación entre el componente sintáctico y el semántico.

A OP-LF se le llama terminal lógico, y está formado por el operador OP, que determina como se combinarán las representaciones semánticas parciales, y LF es una forma lógica que establece una relación entre los argumentos que intervienen en la regla gramatical.

En una DCG para cada no-terminal de la forma:

```
nt(...)→B1:B2:...
```

se obtiene un nodo de un árbol:

nt(X1,X2,...).

donde las Xis son los nodos asociados a los Bis.

Para las MLGs el nodo del árbol asociado con cada regla va a ser de la forma:

syn(nt:Arg, Hijos).

donde nt es un no-terminal, Arg es el primer argumento de nt(...) (si no tiene argumentos es []), e Hijos es la lista de nodos producidos por el cuerpo de la regla. Cada no-terminal contribuye con un simple syn, mientras que cada terminal lógico contribuye con un elemento simple, llamado igual que el mismo.

No todos los símbolos no-terminales van a contribuir con un nodo en el árbol de análisis. Esto se hace con el fin de evitar árboles de derivación innecesariamente grandes. A los que contribuyen se les llama no-terminales fuertes, mientras que el resto se llamarán no-terminales débiles.

El intérprete semántico tomará este árbol y obtendrá por diversas manipulaciones la forma lógica.

El componente de interpretación semántica actúa generalmente con los items semánticos, los cuales son de la misma forma que los terminales lógicos, excepto que pueden ser formas lógicas complejas (resultado de la combinación de varios items semánticos).

Ejemplos de terminales lógicas son:

```
l-hombre(X).
Q/P-cada(P,Q).
```

donde l (unificación izquierda) hace que la forma lógica asociada se una a la izquierda (durante la interpretación semántica) con la forma lógica del ítem semántico con el que se esta recombinando. El operador Q/P hace que P sea unificado con la forma lógica del ítem semántico con el que se está recombinando. Por ejemplo: si el ítem «cada» modifica al ítem «hombre» (como en la frase nominal «cada hombre»), el resultado es el ítem: @P-cada(hombre(X),P) donde @P es nuevo operador que hace que P sea unificado con la forma lógica del ítem con el que se está recombinando.

Ejemplo: cuando @P-cada(hombre(X),P) modifica a l-vive(X), el resultado es el ítem semántico l-cada(hombre(X),vive(X)), que en el último paso de la interpretación queda como cada(hombre(X),vive(X)) y puede interpretarse como «cada hombre vive».

Para escribir el programa de traslación (el compilador de reglas), hemos de tener en cuenta que cada no-terminal, además de los dos argumentos diferencia de listas, ha de tener dos argumentos representando a las estructuras de análisis. Si el no-terminal es fuerte habrá que tomar un sólo argumento de la estructura de análisis que represente el syn asociado con la expansión de este no-terminal (el otro argumento será nil). Si es débil los dos argumentos representarán la diferencia de listas de los nodos producidos por la expansión del no-terminal.

3. Implementación de las MLGs

Para poder abordar una aplicación de procesamiento de lenguaje natural es necesario determinar dos aspectos: cuáles son el tipo de oraciones que serán reconocidas y la definición del campo semántico de la palabras que aparecerán en dichas oraciones.

Al determinar estos dos aspectos podremos restringir el número de oraciones entendibles por la aplicación y reducir los problemas de ambigüedad.

El estudio sintáctico nos permitirá definir la gramática que, aunque depende de la aplicación particular, siguen las mismas pautas para su definición independientemente de la aplicación.

Podemos hacer estudios separados de: los sintagmas nominales, los tipos de oraciones compuestas, los sintagmas verbales y los sintagmas preposicionales.

También es importante resolver los problemas de aquellos sintagmas que no estén colocados en su situación natural, que normalmente serán resueltos utilizando variables que recojan información sobre ellos para recolocarlos en la estructura.

El campo semántico de las palabras, en especial de los nombres y verbos, puede ser útil para desechar oraciones sin sentido y para resolver ambigüedades sintácticas, ya que obligan a que además de cumplir la estructura sintáctica coincidan los tipos que pueden combinarse.

Podemos definir una jerarquía de tipos semánticos y asociar a cada palabra del lexicón un tipo, e incluso si esta palabra pudiera tener modificadores, a qué tipo semántico deben de pertenecer. Los tipos los representaremos con dos puntos seguido de un identificador de tipo.

Ej: perro:animal

4. Aplicación a la Consulta de Bases de datos Relacionales

El proceso de trasformar una oración a una forma objetivo entendible por la aplicación particular que se haga, se divide en una serie de fases, según el siguiente cuadro:

Fase	Transformación	
1 División en tokens:	Oración	-> Lista de tokens
2 Análisis sintáctico:	Lista de tokens	-> Arbol sintáctico
3 Análisis semántico:	Arbol sintáctico	-> Forma lógica
4 Análisis objeto:	Forma lógica	-> Forma objetivo

4.1. División de Tokens

En ella se divide la oración en palabras y se le asocia un token que es ella misma, aunque en algunas se refleja en el token algunas características como empezar por mayúsculas o ser un número. También pueden descomponerse las contracciones y eliminar los signos de puntuación. La oración será un tipo cadena de caracteres y se devolverá un lista PROLOG de tokens.

Ej: Oración: «Juan *llegó a las 10.*»

Lista de tokens: [may(juan),llego,a,las,num(10)]

4.2. Análisis sintáctico

Se obtiene la estructura de árbol ya comentada. Este árbol que contiene la estructura sintáctica de la oración es generado automáticamente por la gramática, e incluye los operadores de cuyo funcionamiento hablaremos en la fase tercera. La gramática será definida en función del tipo de oraciones que se pretendan interpretar.

Ej: Queremos obtener una gramática que reconozca los siguientes tipos de oraciones relacionadas con una universidad:
1 - ¿ Qué asignaturas tiene informática ?.
2 - ¿ Cuantos estudiantes de álgebra obtuvieron un notable en física ?.
3 - ¿ Aprobó cada estudiante 3 asignaturas ?.

Vemos que las oraciones son todas interrogativas y en tercera persona.

Simplifica considerar que una oración está formada por un verbo con una serie de modificadores de dicho verbo, en dichos modificadores se incluyen el objeto directo, el objeto indirecto, el sujeto e incluso cualquier sintagma nominal, con o sin preposición, que regularmente aperezca con dicho verbo.

Siguiendo este método tenemos que sintácticamente tendríamos la regla prototipo:

Oración -> Verbo + Modificadores.

De esta manera en la primera oración «asignaturas» e «informática» modifican al verbo «tiene», y en la segunda «física» modifica a «obtuvieron».

La idea es, por tanto, reflejar en el lexicón los verbos con todos los modificadores y sus tipos, quedándonos en el lexicón los siguientes predicados que recogen en sus argumentos: la palabra, su forma lógica, la variable y tipo asociada al sujeto y una lista con las variables, tipos e información sobre qué preposición rige de todos los que consideraremos modificadores verbales, a este último argumento se le denomina ranura. Separamos de la ranura el modificador sujeto por facilidad a la hora de implementar. Con las ranuras permitimos que haya una misma forma verbal pero con distintos tipos de modificadores, es deseable mantener el mismo identificador, a ser posible su forma en infinitivo, para la forma lógica del verbo incorporándose un número para distinguirlos. También usaremos variables que nos unifiquen los núcleos de los modificadores con los argumentos de la forma lógica del verbo.

Ej: verbo(tiene,tener1(X,Y),X:fac,[(Y:asig)]).
verbo(obtuvieron,obtener1(X,Y,Z),X:est,[(Y:not),(Z:prep(en):asig)]).
verbo(aprobo,aprobar1(X,Y),X:est,[(Y:asig)]).

Vemos que asociamos como tipo, el tipo del núcleo del sintagma modificador. La inclusión de los tipos nos permitirá desechar modificadores que no sean coherentes con el verbo, pues al girar sobre el verbo la construcción de la gramática, leído éste sabremos cuántos modificadores tendremos y qué tipo tiene por su predicado asociado en el lexicón.

Un problema sería que puede haber delante del verbo modificadores y ésto según nuestra regla nos obliga a recolocar ese premodificador detrás del verbo. Quizás la manera más sencilla de abordar este problema es obligar a escribir las oraciones según unos patrones, aunque intentando que estos patrones sean naturales. Unas normas podrían ser:

– escribir el sujeto siempre al lado del verbo,
– permitir que sólo haya un modificador como máximo delante del verbo,
– si no hay pronombres interrogativos, colocar primero el verbo y después el sujeto en las oraciones interrogativas.

Aunque parezcan muy rígidas, estas normas se corresponden con las usadas normalmente al hablar. Las ventajas de usarlas son:

– descubrir fácilmente el sujeto asociado al verbo. La relación sintáctica entre el sujeto y verbo es muy fuerte, ya que deben coincidir en número, persona y tipo,
– tener que recolocar como máximo un modificador, esto es fácil usando un argumento Recol que recoja las características de ese modificador y una vez obtenido el verbo, ver que modificador se adapta a él. Si no fuera el sujeto, entonces por las normas anteriores el siguiente modificador debería ser el sujeto.

Queda también estudiar los sintagmas nominales. Distinguiremos los sintagmas nominales según el tipo de palabra que sea su núcleo, nos reduciremos a dos posibles casos: nombre propio y nombre común.

En un sintagma nominal nos interesa saber el número Num, persona Pers y género Gen para resolver problemas sintácticos, como dijimos antes nuestras oraciones son en tercera persona y de ahí podemos obviar la marca de persona. Consideraremos que el tipo de un sintagma nominal Tipo es el tipo semántico de su núcleo y éste nos servirá para resolver las ambigüedades ya expuestas. Puesto que los sintagmas nominales se refieren a entidades, éstas pueden ser cuantificadas mediante presentadores del sintagma, por tanto, también nos interesa saber qué tipo de cuantificación Dtipo tiene. Suponindo que X es la variable asociada al sintagma nominal, estas informaciones pueden ser recogidas mediante una etiqueta de la siguiente forma:

X:DTipo:Num:Gen:Tipo.

Atendiendo a los distintos tipos de núcleos podemos ver que:

Si el núcleo es un nombre común no estará cuantificado ni normalmente modificado y le asociaremos la etiqueta con sólo la información de su género y tipo. Para definir una entidad no cuantificada usaremos el identificador de tipo def, así la etiqueta sería:

X:def:sg:Genero:TipoN.

Si el núcleo es un nombre propio puede estar cuantificado y puede tener modificadores. Así vemos que en la oración 2 álgebra modifica a estudiantes. Estos modificadores se tratan análogamente a los de los verbos con la particularidad de que no se da el caso de la recolocación. Su etiqueta estará formada por su tipo, número y género al que se le suma el tipo del cuantificador que será unificado mediante otra regla.

Con estas ideas la regla prototipo sería:
Sintagmanominal -> Nombrepropio.
Sintagmanominal -> Determinante + Nombrecomún + Modificadores.

Los determinantes cuantifican a los nombres y tienen un operador especial P/Q para determinar el ámbito de cuantificación que será explicado en la fase 3. Esta cuantificación puede ser de distintos tipos dependiendo de su significado. Una característica es que son una clase cerrada de palabras por lo tanto pueden estudiarse particularmente cada uno. En las oraciones ejemplo vemos que funcionan como determinates: que, 3, cuantos ,cada, y un. La regla prototipo de un determinante será reconocer el determinate y devolver su género, número y tipo.

Los modificadores tienen un tratamiento iterativo consistente en elegir una ranura de las ranuras e intentar reconocer un modificador de esas características, normalmente un sintagma nominal con o sin preposición, hasta que se hayan reconocidas todas. Para su buen funcionamiento es necesaria la correcta definición de las ranuras para los verbos y los nombres, mientras que los modificadores verbales suelen llevar preposiciones a o en, en los modificadores nominales suelen ser precedidos por la preposición de, pero esta regla es quebrantada a menudo. A esta regla compete recolocar si hubiera algún precomplemento, para ello usamos un par de argumentos (Recol,Recol1) cuyo significado es el siguiente:

- si los dos son la lista vacía, o sea ([],[]) entonces o no ha habido precomplemento o este ha sido un precomplemento pero ya ha sido recolocado,
- si son de la forma (recoloca(_),[]) entonces ha habido precomplemento que se ha conseguido recolocar,
- si es de la forma (recoloca(_),recoloca(_)) entonces ha habido un precomplemento pero no ha podido ser recolocado.

Otro aspecto a destacar son los adjuntos que son otro tipo de modificadores, como por ejemplo los adverbios, las complementos circunstanciales, las cláusulas de relativo y otros más que no son recogidos en los argumentos de las formas lógicas de los verbos. Estos son también reconocidos mediante esta regla de tal manera que si existe un modificador que no es coherente con ninguna ranura debe intentarse reconocer como estos otros modificadores. Denominaremos complementos a los modificadores recogidos en las ranuras.

Debido a que las cláusulas de relativo pueden o no tener su propio sujeto es necesario incrementar el número de argumentos de modificadores con XRel, que guarda las características del núcleo nominal ya leído para ver si coincide o no con el verbo subordinado. Los adverbios que semánticamente destacan una parte de la oración o foco, llamados adverbios focalizadores tienen un tratamiento especial que será detallado en la fase 3.

La regla prototipo para los modificadores sería:

Modificadores -> Elegirranura + Complemento + OtrosModificadores.
Modificadores -> Adjunto + OtrosModificadores.
Modificadores -> Noquedanranuras.

Queda el aspecto de definir qué no no-terminales serán fuertes, ya que es fundamental para el buen funcionamiento del análisis semántico. Estos se corresponden con los elementos de la oración más importantes como los sintagmas nominales, sintagmas preposicionales, determinantes, oraciones subordinadas y adverbios. No se definen como tal los núcleos de los sintagmas preposiciones y reglas auxiliares.

La gramática definitiva quedaría:

```
oracion ->
   sujetoverbo(Ranuras,Recol)
   : modificadores(sv,Ranuras,Recol,[],_).
sujetoverbo(Ranuras,Recol) ->
   premodificador(X,Numn,Tipon,Prep)
    : nucleoverbal(Y,Numv,Tipov,Ranuras)
    : coincide(Recol,Prep,(X:Numn:Tipon),(Y:Numv:Tipov)).
sujetoverbo(Ranuras,[]) ->
   (@P)-si_no(P)
   : nucleoverbal(X,Num,Tipo,Ranuras)
   : sintagmanominal((X:Dtipo:Num:Gen:Tipo)).
premodificador(X,Num,Tipo,Prep) ->
   sintagmanominal((X:Dtipo:Num:Gen:Tipo)).
```

```
premodificador(X,Num,Tipo,Prep) ->
   +Prep
   : preposicion(Prep)
   : sintagmanominal((X:pi:Num:Gen:Tipo)).
coincide([],[],Etiqueta,Etiqueta) ->
   [].
coincide(recoloca(X:Tipo,Prep),Prep,(X:N:T),(Y:N1:T1)) ->
   sintagmanominal((Y:Dtipo:N1:Gen:T1)).
nucleoverbal(X,Num,Tipo,Ranuras) ->
   +V
   : verbo(V,Pred,X:Tipo,Num,Ranuras)
   : 1-Pred.

sintagmanominal((X:def:sg:Gen:Tipo)) ->
  +Tnoun
  : nombrepropio(Tnoun,Noun,Gen,Tipo)
  : 1-(X=Noun).
sintagmanominal((X:Dtipo:Num:Gen:Tipo)) ->
   det((X:Dtipo:Num:Gen))
   : +N
   : nombrecomun(N,P,X:Tipo,Num,Gen,Ranuras)
   : 1-P.
   : modificadores(sn,Ranuras,[],[],(X:_:Num:_:Type).

det((X:Dtipo:Num:Gen)) ->
   +D
   : dt(D,B,F,P,X,Num,Gen,Dtipo)
   : F/B-P.

modificadores(Cat,RanurasI,Recol,Recol2,XRel) rightarrow
     elegir((Tiporanura:X:Tipo),RanurasI,RanurasF)
     : complemento(Tiporanura,X:Tipo,Recol,Recol1)
   : modificadores(Cat,RanurasF,Recol1,Recol2,XRel).
modificadores(Cat,Ranuras,Recol,Recol,XRel) ->
   adjunto(Cat,XRel)
   : modificadores(Cat,Ranuras,Recol,Recol,XRel).
modificadores(Cat,[],Recol,Recol,XRel) ->
   [].

adjunto(sv,_) ->
   pp((X:_),(@X)).
adjunto(sv,_) ->
   avp.
adjunto(sn,XRel) ->
   pp(XRel,r).
adjunto(sn,XRel) ->
   relclse(XRel).
pp((X:_),Op) ->
   +Prep
   : prep(Prep,Pred,Y:Tipo,X)
   : Op-Pred
   : sintagmanominal((Y:Dtipo:Num:Gen:Tipo)).
avp ->
   +Adv
   : adv(Adv,Pred,Op)
   : Op-Pred.
relclse(X:_:Num:Gen:Tipo) ->
   +que
   : vhead(X,Num,Tipo,Ranuras)
  : modificadores(vp,Ranuras,[],[],_).
relclse(X:_:Num:Gen:Tipo) ->
```

```
    +que
     : sintagmanominal(Y:_:Numn:Genn:Tipon)
     : vhead(Y,Numn,Tipon,Ranuras)
     : postmods(vp,Ranuras,recoloca(X:Tipo,[]),[],_).

complemento(cd,(X:Tipo),recoloca(X:Tipo,[]),[]) ->
    [].
complemento(cd(a),(X:Tipo),recoloca(X:Tipo,a),[]) ->
    [].
complemento(ci,(X:Tipo),recoloca(X:Tipo,a),[]) ->
    [].
complemento(cp(Prep),(X:Tipo),recoloca(X:Tipo,Prep),[]) ->
    [].
complemento(cd,(X:Tipo),E,E) ->
    sintagmanominal((X:_:_:_:Tipo)).
complemento(cd(a),(X:Tipo),E,E) ->
    +a
    : sintagmanominal((X:_:_:_:Tipo)).
complemento(ci,(X:Tipo),E,E) ->
    +a
    : sintagmanominal((X:_:_:_:Tipo)).
complemento(cp(Prep),(X:Tipo),E,E) ->
    +Prep
    : sintagmanominal((X:_:_:_:Tipo)).
```

4.3. Análisis Semántico

En ella se hace un recorrido tipo preorden en el árbol de la oración de tal manera que se van generando análisis semánticos parciales para cada subárbol mediante las operaciones que son detalladas más adelante, una vez llegada a la raíz del árbol tendremos el análisis semántico de toda la oración, o sea, la forma lógica de toda la oración.

Para realizar el análisis semántico asociamos a cada nodo del árbol sintáctico un ítem semántico aumentado que consta de tres elementos: una etiqueta, un operador y una subforma lógica, estos items semánticos aumentados los representaremos de la siguiente manera:

Etiqueta Operador Subforma lógica.

De una manera automática, a las hojas del árbol, que se les denomina items semánticos, le asociaremos la etiqueta terminal:[], y a los nodos intermedios que son las raíces de los distintos subárboles, que se les denomina items sintácticos, les asociaremos el operador id y la subforma lógica true. El análisis será por tanto obtener la subforma lógica del ítem semántico aumentado de la raíz.

El análisis irá modificando sólo los operadores y subformas lógicas, pues la etiqueta será siempre la correspondiente a la raíz del subárbol que es analizado.

Para obtener esta representación semántica usamos dos tipos de operaciones con el árbol sintáctico: operación de modificación, operación de transformación.

4.3.1. Operación de Modificación

Obtiene un ítem semántico aumentado como la combinación de dos items semánticos aumentados. Con esta operación vamos encajando unas subformas lógicas con otras. Para realizar esta operación nos fijamos sólo en los operadores y subformas lógicas de los items semánticos aumentados.

Ej: Supongamos un subárbol con raíz R y cuyos hijos de izquierda a derecha son H1, H2, y H3, entonces:

R se modifica con H3 dando H3*,
H3* se modifica con H2 dando H2*,
H2* se modifica con H1 dando H1*,
H1* es la representación semántica del subárbol con raíz R.

4.3.2. Operación de Transformación

Transforma la estructura del árbol sintáctico, para ello utiliza información recogida en las etiquetas e incluso en los operadores de los items semánticos aumentados. Esta estructura es transformada de dos maneras:

Operación de reordenación: Cambia el orden de los hijos de un nivel.
Con esta operación ordenamos las subformas lógicas, de tal manera que la operación de modificación se efectúe correctamente. Esta operación es realizada previamente a la de modificación para cada nivel. Para ordenarlas se le asigna un número de orden a los distintos tipos de etiquetas. Con los adverbios focalizadores se debe primero establecer cuál es su foco, una vez colocado el adverbio y su foco al final se establece la ordenación de los demás items.

Ej: Supongamos un subárbol con raíz R y cuyos hijos de izquierda a derecha son H1, H2, y H3, entonces mediante la reordenación podría quedar H2, H1 Y H3.

Operación de elevación: Eleva un nodo hijo al nivel del padre, colocándolo como su hermano izquierdo más cercano. La elevación es necesaria para resolver los problemas de ámbitos de cuantificadores. Antes de modificar un ítem se comprueba si debe ser elevado.

Ej: Supongamos un subárbol con raíz R y cuyos hijos de izquierda a derecha son H1, H2, y R1, y a su vez R1 tiene los hijos S1 y S2, entonces al elevar S2, el resultado sería la raíz R con los hijos H1, H2, S1 y R1*, donde R1* es la representación semántica de R1 sin contar con S1.

Esta operaciones de transformación permiten disociar el orden de escritura sintáctico con el orden en el que aparecen en la forma lógica, lo cual tiene especial aplicación en la búsqueda del ámbito de los determinantes y del foco de los adverbios focalizadores.

4.3.3. Operadores Semánticos

Para entender mejor la operación de modificación haremos un estudio de los operadores más comunes:

Operador l : Está asociado a los núcleos de los sintagmas nominales y a los verbos. Es un operador de conjunción a la izquierda y simplemente une por la izquierda con el operador & su forma lógica asociada con la forma lógica obtenida hasta el momento.

Operador r: Esta asociado a complementos preposicionales de núcleos nominales. Su comportamiento es similar al operador l, pero es de conjunción a la derecha.

Operador id: Asociado al nodo raíz del subárbol que se está operando. Funciona como operador identidad.

Operador F/B: Es usado para los determinantes y algunos pronombres. Funciona como un predicado con dos argumentos: B llamado base y F llamado foco. Unifica su base B

con la forma lógica obtenida en su mismo nivel a su derecha y sube al nivel superior convertido en el operador @F, para poder unificar su foco F. Este operador aparece con asiduidad como nodo más a la izquierda de un nivel.

Operador @P : Es usado con adverbios no focalizadores y aditamentos. Unifica con P la forma lógica obtenida en ese nivel a su derecha.

Operador B<F: Es usado para los adverbios focalizadores. Funciona como un predicado con dos argumentos: B llamado base y F llamado foco. Unifica su foco F con la forma lógica obtenida en su mismo nivel a su derecha y se modifica convertido en el operador focal(B,Pred,OpB), para poder unificar su base B con la forma lógica obtenida sucesivamente en su nivel pero a la izquierda, por eso utiliza el operador con esos tres argumentos: B en donde unifica al final la base definitiva, OpB que utiliza para ir construyendo progresivamente B y el argumento Pred en donde obtendrá la forma lógica definitiva.

Operador focal(Base,Pred,OpB): Ver operador B<F.

4.4. Ejemplo de construcción de una forma Lógica

Expondremos un ejemplo de la obtención de una forma lógica antes de adentrarnos en su transformación a forma objetivo para una aplicación, supongamos que queremos analizar la siguiente oración: «Di*eron clases todos los profesores de cada departamento*»

El árbol sintáctico generado será el siguiente:

```
s:[]
   @P - sino(P)
   1 - darclase(X)
   np:X:todo:singular:masculino:profesor
      det:X:todo:singular:masculino
         F/B - cada(B,F)
      1 - profesor(X,Y)
      np:Y:todo:singular:masculino:departamento
         det:Y:todo:singular:masculino
            F1/B1 - cada(B1,F1)
         1 - departamento(Y)
```

Las subformas lógicas han sido obtenidas de la oración salvo la subforma sino(P), que se usa para representar oraciones interrogativas cuyo resultado es o verdadero o falso. Tenemos un nombre común profesor que está modificado por otro que es departamento, un verbo dar clases con un solo complemento sujeto y dos determinantes cuantificadores universales, esto es reflejado como de tipo todo en la etiquetas para det y para np. La gramática expuesta debe modificarse en las reglas nucleoverbal y determinante para recoger el verbo dar clase y el determinante todos los.

En el análisis semántico prescindiremos de la información de las etiquetas asociadas al género, número y tipo semántico de los núcleos de los sintagmas nominales.

Se irán analizando sucesivamente y por este orden los subárboles cuya raíz está etiquetada como det:X:todo, det:Y:todo, np:Y:todo, np:X:todo y s:[]. Como mencionamos antes los items semánticos asociados a estos nodos se modifican con sus hijos de derecha a izquierda. Veamos detalladamente el análisis:

Subárbol con etiqueta det:X:todo: Tenemos un hijo, luego no hay operación de transformación, sólo la operación de modificación [mod]. Cualquier operador operado con id es él mismo y su subforma lógica. La etiqueta como dijimos es la del padre [padre].

terminal:[] F/B cada(B,F)
[padre] det:X:todo id true
[mod] det:X:todo F/B cada(B,F)

Subárbol con etiqueta det:Y:todo: Análogo al anterior.

terminal:[] F1/B1 cada(B1,F1)
[padre] det:Y:todo id true
[mod] det:Y:todo F1/B1 cada(B1,F1)

Subárbol con etiqueta np:Y:todo: En líneas generales se pretende que los núcleos nominales o verbales queden mediante la operación de ordenación [ord] lo más a la derecha posible de un nivel, dejando más a la izquierda los cuantificadores, por tanto en este nivel no hay ordenación, al operar el nodo np con terminal obtenemos np:Y:todo l departamento(Y), que debe ser modificado con F1/B1. Como dijimos esta modificación unifica la base B1 y sube al nivel superior convertido en @F1, para unificar su foco F1.

det:Y:todo F1/B1 cada(B1,F1)
terminal:[] l departamento(Y)
[padre] np:Y:todo id true
[mod] np:Y:todo l departamento(Y)
[mod] np:Y:todo @F1 cada(departamento(Y),F1)

Subárbol con etiqueta np:X:todo: Hay reordenación [ord], para que el núcleo profesor sea el primero en ser modificado. Una vez ordenados, los items son modificados, pero al haber dos cuantificadores, nos encontramos con que uno de ellos tiene ya su base unificada pues su operador es @F1, como el padre tiene etiqueta todo, implica que vamos a unificar la base de otro cuantificador, por lo que este ítem np:Y:todo debe ser elevado. Al elevarse el ítem anterior se modifica profesor(X,Y) con la base B de det:X:todo de manera similar a la ya dicha.

det:X:todo F/B cada(B,F)
terminal:[] l profesor(X,Y)
np:Y:todo @F1 cada(departamento(Y),F1)

[ord]

det:X:todo F/B cada(B,F)
np:Y:todo @F1 cada(departamento(Y),F1)
terminal:[] l profesor(X,Y)

[padre] np:X:todo id true
[mod] np:X:todo l profesor(X,Y)
[elev] np:Y:todo @F1 cada(departamento(Y),F1)
[mod] np:X:todo @F cada(profesor(X,Y),F)

Subárbol con etiqueta s:[]: Nuevamente hay una ordenación de items para colocar el núcleo verbal al final, la subforma sino tiene mayor prioridad que los cuantificadores pues se está preguntando por la validez de toda la oración. Vemos como el ítem elevado se convierte en hermano izquierdo del ítem semántico aumentado de su nivel. En este nivel se unifican las variable F y F1 sucesivamente con las subformas parciales obtenidas. El operador @F al ser modificado con cualquier otro operador Op, se convierte en el operador Op.

terminal:[] @P sino(P)
terminal:[] 1 darclase(X)
np:Y:todo @F1 cada(departamento(Y),F1)
np:X:todo @F cada(profesor(X,Y),F)
[ord]
terminal:[] @P sino(P)
np:Y:todo @F1 cada(departamento(Y),F1)
np:X:todo @F cada(profesor(X,Y),F)
terminal:[] 1 darclase(X)
[padre] s:[] id true

[mod] s:[] 1 darclase(X)
[mod] s:[] 1 cada(profesor(X,Y),darclase(X))
[mod] s:[] 1 cada(departamento(Y),cada(profesor(X,Y),darclase(X)))
[mod] s:[] 1 sino(cada(departamento(Y),cada(profesor(X,Y),darclase(X))))

Quedando el análisis de la oración:

sino(cada(departamento(Y),cada(profesor(X,Y),darclase(X))))

4.5. Ejemplo de construcción de una Forma Objetivo

Las formas lógica son generales y puede ser usadas como punto de partida de cualquier aplicación particular. Supongamos que queremos hacer una interfaz en lenguaje natural para consultas a base de datos, para ello una vez obtenida la forma lógica de la oración correspondiente a la pregunta tenemos que transformarla en una sentencia válida que reconozca el sistema gestor de base de datos SGBD. Veremos como transformarla para ser reconocida para un base de datos PROLOG.

En la fase de traducción necesitamos:

- definir los predicados asociados a los cuantificadores, adverbios y predicados no léxicos. En nuestro ejemplo sino y cada.
- descomponer aquellos predicados que contienen a su vez predicados que no sean reconocidos como tales por prolog. Este problema aparece asociado a los complementos circunstanciales o predicados cuyos argumentos puedan ser formas lógicas a su vez.
- tener la traducción a relaciones de la base de datos de todos los posibles predicados reconocibles, especialmente nombres y verbos. En nuestro ejemplo departamento, profesor y darclase.

El predicado sino puede ser definido:

```
sino(P) :-
  P,!,write(«Si, es cierto»).

sino(_) :-
  write(«No, no es cierto»).
```

El predicado cada(P,Q) como se mencionó puede ser definido como:

```
cada(P,Q) :-
  not( (P , not(Q)) ).
```

Si tenémos las siguientes relaciones reflejadas como hechos PROLOG, donde recogemos la información sobre profesores, departamentos y las asignaturas dadas por los profesores de una universidad:

```
departamen(Codigodepartamento,Nombredepartamento)
profesor(Codigoprofesor,Nombreprofesor,Codigodepartamento)
perfilprof(Codigoprofesor,Asignatura)
```

podríamos traducir:

```
departamento(X)  -> departamen(X,_)
profesor(X,Y)    -> profesor(X,_,Y)
darclase(X)      -> perfilprof(X,_).
```

Con estas definiciones y traducciones quedaría la forma objetivo:

```
sino(cada(departamen(X,_),cada(profesor(X,_,Y,_),perfilprof(X,_)))),
```

que sería un predicado directamente ejecutable.

5 Mejoras del Formalismo

En un futuro nos planteamos adecuar todas las técnicas desarrolladas por Michael McCord para implementar las MLGs, a una estructura más sistemática siguiendo el modelo LOQUI, desarrollado en [2].

Creemos que de la gramática se debe obtener la representación intermedia. Pensamos que esto supondría colocar en los sitios adecuados los operadores y terminales lógicos que hiciesen falta.

También pensamos que habrá que profundizar en la representación del conocimiento, y tratar de aplicar este formalismo al tratamiento de textos amplios.

Referencias

[1] PATRICK SAINT-DIZIER AND STAN SZPAKOWICZ, 1190 Logic programming, Logic grammars, Language processing en Logic and logic grammars for language processing Ellis Horwood series in artificial intelligence

[2] JEAN-LOUIS BINOT, 1990 Traitement de la langue naturelle, logique et programmation en Approche logique de líntelligence artificielle, V.3 Ed. DUNOD

[3] MICHAEL MCCORD, 1990 Natural Language ptocessing in Prolog. en Knowledge Systems an Prolog 2ª ed. Ed. Addison-Wesley Publishing Company, Inc.

[4] FERNANDO PEREIRA and STUART M. SHIEBER, 1987. Prolog and natural-language analysis. Ed. CSLI.

También se han usado los siguientes libros y artículos para la elaboración del presente trabajo:

Traitement de la langue naturelle, logique et programmation en Approche logique de líntelligence artificielle, V.1 y V. 2 Ed. DUNOD

MCCORD M. C. (1982). Using slots and modifiers in logic grammars for natural language, Artificial Intelligence, Vol. 18, 327, 367.

DAHL, V. and M. C. MCCORD (1983). Treating coordination in logic grammars, American Journal of Computational Linguistics, Vol 9, 69-91.

MCCORD M. C. (1984). Semantic Interpretation for the EPISTLE system. Proccedings of the Second International Logic Programming Conference, Uppsala, Sweden, 65-76.

MCCORD, M. C. (1985). Modular Logic Grammars, Proccedings 23rd Annual Meeting of the Association for Computational Linguistics, Chicago, 104-117.

MCCORD, M. C. (1985). Desing of a Prolog-based machine translation system, Proccedings of the Third International Logic Programming Conference, London.

MCCORD, M. C. (1985). Semantics of Natural Language: LFL, Presentation at IBM Europe Institute, Oberlech, Austria.

MCCORD, M. C. (1985). LMT: A Prolog-based machine translation system (extended abastract) In: Nirenburg, S. ed., Proceedings of the Conference on Theorical and Methodological Issues in Machine Translation of Natural Languages, Colgate University, Hamilton, 179-182.