11
2 ej.

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

TRADUCTOR SINTACTICO - SEMANTICO BIDIRECCIONAL INGLES - ESPAÑOL BASADO EN GRAMATICAS DE CLAUSULA DEFINIDA Y GRAMATICAS DE ESTRUCTURA DE FRASE GENERALIZADA

T E S I S

Que Para Obtener el Título de:

INGENIERO EN COMPUTACION

P r e s e n t a :

ALFREDO MASANAO DIAZ MAEDA

Dirigida por el M. en C. Alejandro Jimenez Garcia

México, D.F. 1988

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

CONTENIDO

INTRODUCCION

Se sabe, a través de vestigios y documentos, que el hombre ha tenido desde siempre la inquietud de fabricar artefactos que imiten ciertos comportamientos de los animales y de los seres humanos. Esta inquietud se ha hecho cada vez más ambiciosa, de hecho en los siglos XVI-XVII algunos relojeros asombraron al mundo con mecanismos tales como un pato que "comía" granos para más tarde excretarlos (por supuesto los granos originales permanecían dentro del pato, y el excremento se colocaba de antemano dentro de él); o los muñecos que tocaban piezas musicales, y los que eran capaces de escribir con letra muy estilizada.

Con el gran avance científico y tecnológico que ha envuelto al mundo desde la segunda mitad del siglo XIX, el hombre ha retomado estas tareas pero con el objetivo de desarrollar artefactos que sean más bien útiles.

Una de las tareas que suelen considerarse como exclusivas de los seres humanos (aspecto con el que no concuerdo) es la capacidad de comunicarse mediante el habla. Ahora bien, la invención de las computadoras hizo que algunas personas se tomaran en serio la idea de desarrollar una máquina o un sistema capáz de llevar a cabo la labor de traducción entre dos idiomas (lenguajes naturales).

Lograr que una máquina lleve a cabo tal tarea resultó ser mucho mas difícil de lo que se pensó originalmente, principalmente porque no se contaba con una teoría lingüística lo suficientemente depurada como para formalizar el lenguaje de tal manera que fuera posible su formulación en algún código de computadora. También se tenía la desventaja de que los códigos de computadora eran muy deficientes y que las computadoras mismas no tenían el poderío suficiente para soportar tal tipo de tarea.

Afortunadamente las cosas han cambiado y gracias a los grandes avances en la lingüística y al desarrollo tecnológico, es posible llevar a cabo la creación de sistemas capáces de entender el lenguaje natural y de efectuar lo que se conoce como traducción automática. Por traducción automática se entiende la traducción por computadora de textos de un lenguaje natural a otro lenguaje natural.

Los estudios actuales en cuestión de traducción automática se enfocan casi por completo en la obtención de sistemas capaces de manejar la representación semántica de

los lenguajes involucrados, esto se debe principalmente a que la representaciòn sintàctica ya ha sido superada casi por completo.

Casi todos los formalismos desarrollados hasta ahora no se basan en la gramàtica libre de contexto por ser considerada no apta para la interpretaciòn de lenguaje natural. Sin embargo recientemente se han realizado estudios en el formalismo de las Gramàticas de Clàusula Definida (Definite Clause Grammar, DCG) y en el formalismo de las Gramàticas de Estructura de Frase Generalizada (Generalized Phrase Structure Grammar, GPSG); estos formalismos establecen que es posible obtener sistemas de lenguaje natural de gràn calidad, considerando a los lenguajes naturales como libres de contexto.

El presente trabajo de tesis tiene como objetivos primarios los siguientes:

- Describir los aspectos principales de las DCGs y GPSGs como formalismos para la creaciòn de sistemas de traducciòn automàtica y su interpretaciòn en el lenguaje de programaciòn Prolog, esto último porque el sistema implementado se desarrollò en Prolog.
- Desarrollar un sistema de traducciòn automàtica entre los idiomas Español e Inglès con anàlisis sintàctico y semàntico, basado en las DGCs y las GPSGs.
  Se toman los idiomas Español e Inglès porque el primero es el idioma oficial en Mèxico, y el Inglès por que es el idioma de difusiòn por excelencia.
  No se pretende obtener un sistema de traducciòn completo entre los dos idiomas entre otras cosas por cuestiòn de volumen.
- Verificar si las Gramàticas de Clàusula Definida y las Gramàticas de Estructura de Frase Generalizada son viables como formalismos para la interpretaciòn sintàctica y semàntica en sistemas de traducciòn automàtica.

Los objetivos secundarios son:

- Mostrar un panorama general de lo que se conoce bajo el nombre de Lenguaje Natural y Traducciòn Automàtica como àreas de estudio de la Inteligencia Artificial.
- Mostrar un panorama general sobre la Gramàtica, enfocado a dirigirla hacia el contexto de la Inteligencia Artificial. Esto último debido a que existen muchos estudios en gramàtica que no generan puntos directos de conexiòn entre èsta y la Inteligencia Arttificial.

Este trabajo esta organizado de tal forma que se recomienda que aquella persona no familiarizada con la Inteligencia Artificial o en especial con el campo del

lenguaje natural y traducción automàtica lea el capitulo I. Si la persona tampoco està familiarizada con aspectos correspondientes a la gramàtica entonces se recomienda que lea el capitulo II. Esto se debe a que los dos primeros temas tienen el proposito de ambientar a la persona en tales tòpicos.

Los capitulos III y IV describen los formalismos en que se basa el sistema de traducciòn automàtica desarrollado para èsta tesis, por lo que se recomienda a toda persona que los consulte. El capitulo V describe la implementaciòn del sistema desarrollado, por lo que es esencial su consulta a fin de entender el diseño y desarrollo del sistema de traducciòn automàtica.

Al final se incluyen las concluciones del trabajo de investigaciòn y del sistema de traducciòn automàtica desarrollado, tomando en cuenta los objetivos mencionados.

El apèndice A contiene el listado completo del sistema implementado, y el apèndice B contiene algunas pruebas hechas al sistema.

CAPITULO I

LENGUAJE NATURAL
Y
TRADUCCION AUTOMATICA

> ... la sociedad solo puede ser entendida a través de un estudio de los mensajes y de las facilidades de comunicación que le pertenecen.
>
> Norbert Wiener.

I.1 INTRODUCCION

Una de las áreas de estudio de mayor interés dentro del campo de la Inteligencia Artificial (IA, en inglés Artificial Intelligence AI) es la dedicada al Procesamiento de Lenguaje Natural (PLN, en inglés Natural Language Processing NLP), cuyo objetivo es lograr que la computadora sea capáz de "entender" el lenguaje aprendido y utilizado por los seres humanos.

El objetivo primordial de los sistemas de procesamiento de lenguaje natural es lograr una comunicación hombre-máquina eficiente, donde el esfuerzo de la interpretación sea más bien hecha por la computadora y no por el hombre. Tal tipo de sistemas permitirá que prácticamente cualquier persona sea capáz de comunicarse con una computadora y de explotar de manera eficiente sus recursos sin la necesidad de conocimientos previos en computación y/o programación.

El estudio del lenguaje natural dentro de la Inteligencia Artificial se divide en dos áreas principales: reconocimiento del lenguaje y entendimiento del lenguaje [KIM85]. En el reconocimiento del lenguaje se pretende lograr identificar la gramaticalidad de una oración identificando sus elementos y su correcta estructuración. En el entendimiento del lenguaje se pretende lograr la interpretación correcta de la oración desde el punto de vista de su significado, tomando en cuenta la complejidad de

la representación destino, el tipo de mapeo a realizar y el nivel de interacción de los componentes de la representación original [RIC85].

Desde el punto de vista aplicativo, el estudio del lenguaje natural se divide en dos áreas: La interpretación del lenguaje natural y la traducción automática. Ambas requieren tanto de reconocimiento, como de entendimiento del lenguaje a procesar (o lenguajes en el caso de la traducción automática).

Resulta obvio que el desarrollo de esta área de estudio de la Inteligencia Artificial ha obligado la cooperación tanto de expertos en esta área, como de expertos en lingüística, para el estudio y desarrollo de formalismos y sistemas útiles y eficientes de procesamiento de lenguaje natural.

I.2 HISTORIA

De acuerdo con Yehoshua Bar-Hillel, la concepción de los sistèmas de traducción automática fué concebida a principios de los años 30, por el ruso P. P. Smirnov-Troyansky y por el francès G. B. Artsouni [BAR81]. Sin embargo su trabajo fuè ignorado principalmente por no existir aún màquinas en las cuales desarrollar (o al menos intentar desarrollar) tal tipo de tareas, es decir màquinas computadoras.

Una dècada despuès, en 1946-49, a pesar de que la teoria lingüistica estaba muy poco desarrollada y de hecho no existian siquiera los conceptos esenciales que conocemos actualmente. Warren Weaver y A. Donald Booth que se dedicaban a la generación y manejo de còdigos de computadora, concibieron que se podrian emplear las mismas tècnicas para sistemas de traducción en el que el principal problema era la incorporación de un diccionario completo de los dos lenguajes (idiomas) a traducir ([BAR81], [MAC85]).

El resultado de la investigación de lo que ellos denominaron Traducción Automàtica (Machine Translation), que pretendia simular en la computadora las funciones

desempeñadas por un traductor humano, fué el fracaso. A raiz de esa y otras investigaciones surgieron un sin número de dificultades prácticamente insuperables tanto por los escasos conocimientos sobre teoria lingüistica de aquella época como por lo rudimentario de los sistemas de cómputo [BAR81].

Los problemas más representativos a los que se enfrentaron en el proceso de traducción fueron: [BAR81].

- Muchas palabras tienen varias traducciones dependientes del contexto.
- El orden de las palabras varia de un idioma a otro.
- Las expresiones idiomáticas no pueden traducirse palabra a palabra.

Estos problemas obligaron a Weaver a abandonar la idea inicial de tener un sistema que pudiera efectuar traducciones entre dos idiomas cualesquiera, mediante el uso de un lenguaje intermedio universal al cual llamó "interlingua" [BAR81].

Muchos fracasos hicieron que el interès por los sistemas de traducción automàtica se perdiera y sólo unos cuantos (principalmente en la IBM y la Universidad de Georgetown) continuaron realizando algunos estudios, principalmente sobre la traducción Ruso-Inglès [BAR81]. En lugar de intentar realizar las traducciones palabra a palabra, los investigadores se dedicaron al estudio de sistemas de traducción oración a oración y frase a frase. El sistema de traducción Ruso-Inglès usaba un diccionario de 250 palabras y seis reglas sintàcticas.

El experimento realizado en la U. de Georgetown demostró que la traducción automàtica es factible [CHA86]. Surgieron entonces en Harvard, el MIT, la Universidad de Pennsylvania y otros lugares, proyectos cuya meta era la obtención de traducciones útiles, tipicamente del Ruso.

En 1966 Bar-Hillel anunció en el reporte de la ALPAC que "La traducción automàtica significa tentadoramente poder pasar por medio de un algoritmo de un texto fuente que pueda entender la màquina a un texto destino útil, sin recurrir a la traducción o edición humana. En este contexto, no ha existido la traducción automàtica de texto cientifico general y no hay ninguna en proyecto por el momento..." ([BAR81], [MAC85]). Además dijo que no existia manera de resolver las ambigüedades de sentido sin un entendimiento profundo de lo que una palabra significa [CHA86]. En ese mismo año, la "National Science Foundation" publicó el "Reporte Pierce", en la cual Pierce dice que no hay manera en que la traducción automàtica pueda justificarse en tèrminos de resultados prácticos, las traducciones producidas por los sistemas eran muy dificiles de leer.

Inclusive la post-edición no lograba que las traducciones automàticas se semejaran siquiera un poco a las traducciones humanas.

Afortunadamente las investigaciones realizadas por Noam Chomsky sobre teoria lingüistica, publicadas en su libro "Syntactic Structures" [CHO62], en las que introduce su gramàtica transformacional, permitieron un nuevo desarrollo en el anàlisis de la sintaxis. Los cientificos aprovecharon estas nuevas aportaciones y se desarrollaron nuevos lenguajes de programaciòn y nuevas estructuras de datos (Algol, Lisp, estructuras de àrbol, etc.). Esto permitiò el resurgimiento de la investigaciòn en el campo del entendimiento del lenguaje natural en los años 60 y la traducciòn automàtica en los años 70 [BAR81]. Wilks y Schank fueron de los màs destacados iniciadores en esta nueva generaciòn de sistemas de traducciòn automàtica.

Los primeros sistemas que se desarrollaron se enfocaban ùnicamente en el anàlisis sintàctico por ser màs sencillo, y se basaban en el manejo de patrones, evitando asi un anàlisis sintàctico exhaustivo [MAC85]. De los sistemas de èste tipo el màs conocido es ELIZA, un sistema que modelaba el diàlogo entre un paciente (el usuario) y su psiquiatra (ELIZA), fuè desarro-llado por Joseph Weizenbaum en 1966.

Uno de los primeros sistemas de traducciòn automàtica que dió resultados alentadores fuè el sistema de traducciòn automàtica de Yorick Wilks (1973) [BAR81]. Su sistema efectuaba traducciones aceptables del Inglès al Francès. Se basa en un proceso basado en semàntica a partir de un anàlisis sintàctico convencional, haciendo uso ademàs del formalismo de la interlingua ideado por Weaver. Segùn Wilks su representaciòn semàntica fuè diseñada ùnicamente para sistemas de traducciòn automàtica, por lo que es inapropiada para otro tipo de aplicaciòn en el procesamiento de lenguaje natural.

Estos son los sistemas precursores considerados como clàsicos dentro del campo del procesamiento de lenguaje natural, por lo que se pueden contextualizar como històricos. Algunos de los trabajos realizados posteriormente (hasta la actualidad) seràn descritos en los capitulos I.8 y I.9.

I.3 INTERPRETACION DE LENGUAJE NATURAL

La interpretación y generación en el procesamiento de lenguaje natural requiere de la aplicación cooperativa del conocimiento especifico del lenguaje a interpretar, del conocimiento acerca del uso y orden de las palabras, su identificación y caracterización, la estructura de las frases y conocimiento del mundo real sobre situaciones, eventos, contextos, etc.; Para ello se hace necesario poder obtener una representación adecuada del lenguaje a interpretar, tener un sistema de reconocimiento o ajuste de patrones eficiente, una representación semántica completa y un diccionario ó léxico lo más completo posible ([CHA85], [GRO86], [POL86], [RIC85]).

Un procesador de lenguaje natural debe resolver los problemas de ambigüedad. Tal labor es muy complicada debido a que el contexto del discurso que restringe la interpretación del texto se encuentra especificada mayormente en el texto mismo [GRO86]. El conocimiento del contexto extralingüistico hace que la solución de ambigüedades sea más fácil, sin embargo siempre será necesaria la manipulación del conocimiento acerca del mundo real.

Dado que el procesamiento de lenguaje natural es sumamente complicado, resulta prácticamente imposible la realización de sistemas que efectúen un análisis completo a nivel primordialemente semántico, los sistemas actuales procesan textos en lenguaje natural de áreas especificas, como por ejemplo electrónica, y diversos campos de la ciencia de la computación, [NAG85].

Los principales factores que influyen en la complejidad del procesamiento de lenguaje natural son [RIC85]:

- La complejidad de la representación final en la que el ajuste se efectúa.
- El tipo de mapeo (uno-uno, muchos-uno, etc.).
- El nivel de interacción de los componentes de la representación original.

Estos factores obligaron que los primeros sistemas útiles desarrollados efectuaran únicamente un análisis superficial de las oraciones, como en el caso de ELIZA; y que el conocimiento representado se ubicara en un contexto altamente restringido en su semántica, como en los sistemas STUDENT y SHRDLU [RIC85].

El procedimiento general para el procesamiento de lenguaje natural consta de 3 etapas [RIC85]:

A) Construir una estructura que represente la información a ser comunicada, es decir, decidir lo que se va a decir.
B) Aplicar las reglas del texto y la estructura del diàlogo para construir secuencias de oraciones congruentes.
C) Aplicar la información lèxica y las reglas sintàcticas que generen oraciones reales o verdaderas.

Para que una oración pueda ser interpretada es necesario entender en primer lugar todas las palabras que contiene la oración y, en segundo lugar juntar las palabras de manera que formen una estructura que represente el significado de la oración. El proceso de determinación del significado correcto de las palabras se conoce como desambigüedad lèxica o desambigüedad del sentido de la palabra [RIC85].

Para poder interpretar un conjunto de oraciones (texto) es necesario encontrar la(s) relación(es) entre esas oraciones, pero para ello es necesario tener un alto nivel de conocimiento sobre el mundo (contexto) del que se discute. A fin de facilitar esta tarea es necesario que durante el proceso de utilización del conocimiento se enfatice la atención en la parte relevante de la base de datos del conocimiento disponible y que se utilice tal conocimiento para resolver las ambigüedades y la realización de las conexiones entre las cosas que se dicen (escriben) [RIC85].

I.4 TRADUCCION AUTOMATICA

En la traducción automàtica las oraciones del lenguaje origen son mapeadas (por lo general) directamente en las oraciones correspondientes del lenguaje destino mediante la ayuda de un diccionario del lenguaje fuente, ó del lenguaje destino (o de ambos), y un modelo simple de las estructuras gramaticales de los dos lenguajes. Para ello es necesario entender el significado de la oración [RIC85].

En su libro "Teoria de la Información, del Lenguaje y de la Cibernética", Jagjit Singh [SIN82] comenta en su ensayo sobre màquinas traductoras que: "... La dificultad està en que no existe una clave sencilla y obvia que una los dos lenguajes naturales y asi el traducir, distinto de convertir en còdigos, no es una transformaciòn letra por letra. Como resultado de esto, inevitablemente, es propensa a sufrir pèrdidas de conversiòn màs o menos serias en cuanto a sentido y contenido."

Algunos de los problemas a los que se enfrentan los sistemas que no analizan el significado son los siguientes:

A) Desambigüedad Lèxica: El lenguaje origen puede tener una palabra de dos o màs significados, mientras que el lenguaje destino puede usar dos palabras distintas. Para seleccionar la palabra correcta es necesario conocer el significado que se le pretende dar a la palabra.
B) Ambiguedad Gramatical: Cada uno de los significados posibles de una oraciòn gramaticalmente ambigua en el lenguaje origen puede ser representado por una estructura gramatical distinta en el lenguaje destino.
C) Referencias Anafòricas: En el lenguaje origen, un solo pronombre puede requerir distintos pronombres en el lenguaje destino. Para elegir el adecuado debe conocerse la referencia exacta.
D) Modismos: Un modismo en el lenguaje origen debe ser reconocido y no debe traducirse directamente en el lenguaje destino.

Segùn Bruderer, un sistema de traducciòn puede describirse como consistente en cuatro elementos [JIM86]:

1) El diccionario (lèxico).
2) Las gramàticas (para los lenguajes fuente y destino).
3) El programa de traducciòn.
4) El equipo de procesamiento de datos (hardware).

De acuerdo con Jimènez [JIM86], se tienen dos enfoques en el proceso de traducciòn que dependen de la relaciòn existente entre los programas y las gramàticas:

- Lèxico-programa.
- Lèxico-gramàtica-programa.

La segunda de ellas es la màs utilizada por ofrecer las siguientes ventajas [JIM86]:

- La descripciòn de las gramàticas no dependen de los programas de traducciòn.
- Se puede utilizar el mismo programa para traducir distintos pares de lenguajes fuente-destino debido a que el programa de traducciòn es independiente de la descripciòn de los lenguajes.

- Si las reglas gramaticales o las palabras del lèxico son cambiadas, el programa de traducciòn no es afectado. Gracias a eso podemos actualizar y modificar el proceso de traducciòn.
- Se puede dividir la tarea en dos àreas distintas, una para el experto en lingüistica y otra para el experto en computaciòn.

De acuerdo a la evolución de estos sistemas, se pueden identificar distintos niveles en la traducciòn automàtica [JIM86]:

1.- Nivel lèxico.
2.- Nivel sintàctico.
3.- Nivel semàntico.
4.- Nivel Pragmàtico.

El nivel lèxico se basa en una traducciòn palabra-a-palabra. Este fuè el enfoque utilizado en los primeros sistemas de traducciòn [JIM86].

El nivel sintàctico fuè el màs utilizado en los años 70 y principios de los años 80, y normalmente se limita a traducir una palabra a la vez y mostrando còmo una palabra se relaciona con otra, por lo que muy dificilmente se pueden resolver ambigüedades y otros problemas comùnes debido a la falta de informaciòn que tiene este enfoque. De hecho, solamente genera rechazo si alguna regla del lenguaje es violada ([JIM86], [RIC85]).

El nivel semàntico involucra la traducciòn de oraciones tomando en cuenta su significado, normalmente mediante el mapeo de las estructuras sintàcticas y los objetos en el dominio correspondiente. Las estructuras que no pueden ser mapeadas son rechazadas ([JIM86], [RIC85]).

El nivel pragmàtico se basa en el conocimiento especifico del mundo, es decir, lo que se dice en un momento dado es reinterpretado para determinar que es lo que se quiere decir o dar a entender. Se usa para construir sistemas de alta calidad ([JIM86], [RIC85]).

De acuerdo con Jimènez [JIM86], el proceso de traducciòn se puede dividir en 7 estados:

1.- Pre-ediciòn (codificaciòn de informaciòn).
2.- Entrada.
3.- Anàlisis.
4.- Transformaciòn (traducciòn).
5.- Sintesis.
6.- Salida.
7.- Post-ediciòn.

1.- La pre y post-ediciòn normalmente se efectùan bajo intervenciòn humana. En su libro "Teoria de la

Información, el Lenguaje y la Cibernética" [SIN82], Singh dice que "... las traducciones automáticas en alta fidelidad probablemente nùnca estarán al alcance de la computadora digital normal, aunque éstas serán posibles si el hombre y la màquina colaboran apropiadamente, colaboración en la cual el hombre puede preeditar el texto de entrada para hacérselo màs asequible al manejo de la màquina o posteditar el de salida para hacerlo màs inteligible al lector".

2.- El proceso de entrada consiste bàsicamente en la lectura y almacenamiento de la oración en la computadora. La oración es separada en palabras y almacenada en una lista para ser presentada de esta forma al proceso de análisis.

3.- El proceso de análisis es el más importante de la traducción y se divide en tres etapas diferentes:

i.- Busqueda en el lèxico.
ii.- Análisis Sintáctico.
iii.- Análisis Semántico.

i) La busqueda en el lèxico puede ser de dos tipos: El análisis morfológico y el análisis de ajuste.

El Análisis morfológico, que consiste en el reconocimiento de una oración y su desglosamiento en palabras y morfemas para ser comparadas contra el lèxico, el cual contiene únicamente la forma básica de las palabras, y por lo tanto los afijos, inflexiones, etc. son eliminados. Este enfoque es muy apropiado para lenguajes como el Español, en el que sí tienen por ejemplo, conjugaciones diferentes para cada tiempo, e inclusive las terminaciones de las palabras son diferentes de acuerdo al número y género. Sin embargo, no es apropiado en lenguajes con pocas terminaciones gramaticales como el Inglès.

El análisis de ajuste es muy apropiado para ser utilizado en lenguajes como Prolog. Es necesario que el lèxico incluya todas las formas posibles de las palabras para que la palabra completa sea comparada con el lèxico. Este enfoque es adecuado para lenguajes como el Inglès. Inclusive en lenguajes como el Español es la mejor alternativa al intentar implementaciones en Prolog.

ii) El objetivo primordial de este estado es determinar la estructura sintáctica de la oración, por lo que cada palabra se asocia con un elemento específico de la gramática. Para ello el analizador sintáctico construye un árbol

de anàlisis (parse tree) que puede ser analizado en base a las distintas tècnicas de bùsqueda (depth-first, breadth-first, etc.).

iii) El anàlisis semàntico intenta categorizar las palabras de acuerdo a elementos semànticos que nos pueden dar informaciòn sobre cómo utilizar las palabras en la oraciòn, es decir, informaciòn sobre la compatibilidad de las palabras. Por ùltimo se deben resolver las distintas ambigüedades encontradas en el texto.

4.- El objetivo del estado de traducciòn es la transformaciòn de las estructuras del lenguaje fuente a las estructuras equivalentes en el lenguaje destino. La transformaciòn puede ser mediante reajustes, inversiones de elementos, etc., o bien una transformaciòn lèxica en la que se efectùa el reemplazo de palabras. Este proceso, a diferencia del anàlisis y sintesis, sì es gobernado por los idiomas involucrados. La transformaciòn normalmente se lleva a cabo en un sòlo sentido (direcciòn) debido a que las rutinas de anàlisis y conversiòn de gramàticas son muy complejas, y a que generalmente se requieren distintas rutinas para la traducciòn en uno y otro sentido. Este inconveniente se supera fàcilmente con el uso de lenguajes como el Prolog, debido a que permite que el ajuste de valores en las clàusulas se haga de manera automàtica, por lo que el proceso de traducciòn se puede hacer en ambas direcciones con las mismas rutinas.

5.- En la sintesis se busca obtener, a partir de la estructura obtenida en el proceso de transferencia, las oraciones en el lenguaje destino. Por lo general este proceso se desarrolla menos que los modelos de anàlisis. En el caso del sistema mostrado en èsta tesis no sucede asì, ya que debido a las caracteristicas de Prolog, el estado de anàlisis resulta ser el mismo que el de sintesis.

6.- El estado de salida consiste simplemente en desplegar en algùn dispositivo de salida la oraciòn correspondiente del lenguaje fuente en el lenguaje destino.

"... las instrucciònes del programa de la màquina deben ser escritas en el propio lenguaje interno de èsta que es mucho màs preciso, ordenado y descifrable que cualquier lenguaje humano. Este acuerdo, entre el lenguaje ordenado de la màquina y los lenguajes naturales del hombre, genera casi todas las dificultades de la traducciòn a màquina. En cuanto

podamos convertir nuestro conocimiento actual sobre la morfologia, la sintaxis y la semàntica de los lenguajes naturales en rutinas de acuerdo a un plan, o procedimientos de decisiòn, podrà ser mecanizado el proceso de traducir de un lenguaje natural a otro", Jagjit Singh [SIN82].

I.5 SINTAXIS

El àrea màs estudiada y desarrollada del procesamiento de lenguaje natural (PLN) es el correspondiente a la sintaxis de los lenguajes naturales. La investigaciòn en esta àrea concierne dos aspectos relacionados: las gramàticas y los algoritmos de anàlisis gramatical (parsing algorithms).

Se puede definir a la gramàtica (en este contexto) como una especificaciòn finita de un posible lenguaje infinito que capta sus regularidades. Puede usarse en la generaciòn de las oraciones del lenguaje, en la determinaciòn de si una cadena dada pertenece o no al lenguaje y, en caso de pertenecer a èste, determinar su estructura de acuerdo a la gramàtica. Los estudios realizados por Chomsky contribuyeron mayormente al desarrollo de la teoria gramatical que se tomò como base para el desarrollo de los lenguajes de computadora ([CHO76], [GRI71], [LEW78], [TRE85]).

De entre las gramàticas usadas para el PLN destacan ([BAR81], [GRO86]): las gramàticas de estructura de frase (phrase structure grammars) y sus variantes, las gramàticas transformacionales (transformational grammars), las gramàticas sistèmicas (systemic grammars) y las gramàticas de agregaciòn arbòrea (tree adjunction grammars).

De acuerdo con Grosz, Sparck y Lynn [GRO86], los algoritmos de anàlisis gramatical especifican còmo se aplica la gramàtica a una secuencia de palabras para producir una representaciòn estructurada o àrbol de anàlisis gramatical (parsing tree). Un analizador gramatical (parser) comprende una gramàtica y un algoritmo de anàlisis gramatical. En algunos sistemas de PLN el analizador gramatical incluye un proceso (integrado o separado) que produce una representaciòn del significado de la expresiòn de entrada.

LENGUAJE NATURAL Y TRADUCCION AUTOMATICA

La teoria de lenguajes formales introducida por Chomsky en los años 50 fuè desarrollada como un estudio matemàtico, sin embargo influenciò grandemente el diseño de lenguajes de computadora y posteriormente los sistemas de PLN [BAR81]. Un lenguaje formal se define como un conjunto (posiblemente infinito) de cadenas de longitud finita formado a partir de un vocabulario finito de simbolos. La gramàtica de un lenguaje formal se especifica en tèrminos de los siguientes conceptos:

- Las categorias sintàcticas, referenciadas como simbolos no terminales o variables.
- Los simbolos terminales del lenguaje.
- Las reglas de reescritura o producciones, especifican las relaciones entre ciertas cadenas de simbolos terminales y no-terminales.
- El simbolo inicial.

Para la creaciòn de los lenguajes de computadora se hizo necesario la formulaciòn de nuevos tipos de gramàticas propias.

Tales gramàticas (formales), definidas por Chomsky, son de cuatro clases o tipos bàsicos y se diferencian en la forma en que las reglas de reescritura pueden usarse o son permitidas, y son ([BAR81], [GRI71], [LEW78], [RIC85], [TRE85]):

- Tipo 0: Gramàtica de Estructura de Frase (Phrase Structure Grammar), muy poco estudiada hasta finales de la dècada pasada. Sus reglas de reescritura son de la forma:

  u ::= v

  donde: u està en V+
  v està en V*
  V es un alfabeto
  V+ = A^1 unièn A^2 unièn... A^n unièn...
  V* = A^0 unièn A+

  Es decir, la parte izquierda puede ser una secuencia de simbolos, y la parte derecha puede ser vacia.

  Nòtese que este tipo de gramàticas no tienen restricciones en la forma que pueden tomar las reglas de reescritura.

- Tipo 1: Gramàtica Sensitiva al Contexto ò Dependiente de Contexto (Context Sensitive), sus reglas de reescritura son de la forma:

xUy ::= xuy

donde: U está en V-T
T = alfabeto de símbolos terminales
x,y están en V*
u está en V+

El término sensitivo al contexto se refiere al hecho de que se permite reescribir U como u solamente dentro del contexto x...y.

- Tipo 2: Gramática Libre de Contexto (Context Free), surgió a partir de las gramáticas sensitivas al contexto, y sus reglas de reescritura son de la forma:

U ::= u

donde: U está en V-T
u está en V*

Se le llama libre de contexto porque U puede reescribirse como u independientemente del contexto en el cual aparezca. En este tipo de gramática, una regla puede tener la forma:

U ::= &

donde: & es la cadena vacía.

Si la gramática carece de &, entonces se le conoce como gramática libre de & (&-free grammar).

- Tipo 3: Gramática Regular (Regular Grammar), cuyas reglas de reescritura son de la forma:

U ::= N ó U ::= WN

donde: N está en T
U,W están en V-T

Este tipo de gramática es muy importante en el estudio de la teoría del lenguaje y la teoría de autómatas, pues el conjunto de cadenas que genera puede ser aceptado por una máquina de estado finito y viceversa ([GRI71], [RAL76], [TRE85]).

Las cuatro clases están incrementalmente restringidas, esto es, existen lenguajes de estructura de frase que no son sensitivos al contexto, lenguajes sensitivos al contexto que

no son libres de contexto, y lenguajes libres de contexto que no son regulares. La forma de las reglas de reescritura en las gramáticas tipo 1, 2 y 3, por ser más restringida, genera lenguajes correspondientemente más simples.

De éstas gramáticas, la más usada ha sido sin duda alguna la gramática tipo 3 ó libre de contexto por ser la gramática más versátil y poderosa para la definición de lenguajes de computadora, (le sigue en uso la gramática regular). A pesar de que los lenguajes naturales son lenguajes regulares, y no libres de contexto, Chomsky ([CHO62], [CHO76], [BAR81]) ha demostrado mediante su gramática transformativa que la representación libre de contexto es una alternativa prometedora en el procesamiento de lenguaje natural. De hecho, la mayoría de los formalismos más eficientes se basan en este tipo de gramáticas. Existen sin embargo algunos formalismos basados en representaciones no-libres de contexto, como por ejemplo, las redes de transición aumentada (RTA, o ATN del inglés Augmented Transition Network), que siguen siendo estudiadas y usadas en el desarrollo de sistemas de PLN por su buen comportamiento.

I.6 SEMANTICA

Tradicionalmente la semántica se toma como un mapeo entre el lenguaje y un modelo que corresponde a un mundo real o imaginario. Desafortunadamente los sistemas de NLP no pueden llevar a cabo el mapeo de manera directa. Necesitan primeramente representar el significado de una expresión, y que dicha representación sea entonces interpretada en el mundo del modelo ([GRO86], [KAT79]).

Para poder llevar a cabo la interpretación semántica es necesario entender primeramente los conceptos de dominio, contexto y tarea [GRO86]. El dominio es una parte del mundo (real o imaginario) sobre la cual el sistema tiene conocimiento general y/o particular. El contexto de una expresión se determina por la combinación de las expresiones previas en el discurso, su localización en el espacio, tiempo, creencias, deseos e intenciones de los participantes del discurso. La tarea del sistema es el servicio que ofrece

al usuario. Es decir, el dominio està asociado con los sistemas y el tipo de textos que pueden entender; el contexto està asociado con las expresiones y la forma còmo son entendidas las expresiones; y la tarea està asociada con los sistemas y los usuarios, o sea con lo que el usuario intenta hacer con sus expresiones.

Grosz, Sparck y Lynn en su libro "Readings in Natural Language Processing" [GRO86], establecen que "La representaciòn completa del significado de una expresiòn refleja todos los aspectos del dominio, contexto y tarea, y puede ser interpretado directamente en el mundo o modelo al que el sistema està conectado. La representaciòn completa del significado de una expresiòn refleja su semàntica lèxica, las entidades particulares denotadas por sus expresiones referenciantes, los argumentos de sus relaciones, las relaciones adicionales sobre las explicitamente especificadas en la expresiòn, asi como la intenciòn del hablante al producir la expresiòn. Mientras que las decisiones sobre algunos de estos aspectos pueden hacerse localmente, basados simplemente en el conocimiento sintàctico y lèxico semàntico, otras decisiones requeriràn: informaciòn del discurso, conocimiento real particular o de sentido comùn del mundo del modelo, o conocimiento acerca de las intenciones del hablante".

Un problema comùn es que las expresiones en lenguaje natural pueden ser ambiguas e indeterminadas, en formas que no pueden ser resueltas localmente. Pueden ser ambiguas con respecto a tres cosas: estructura, lèxico y alcance.

Una expresiòn puede ser indeterminada con respecto a la informaciòn que no ha sido explicitamente especificada. Dada la ambigüedad e indeterminaciòn de las expresiones de lenguaje natural y dado que resolverlas puede requerir informaciòn que no es local a la expresiòn, si un proceso de interpretaciòn semàntica particular obtiene una interpretaciòn intermedia, dicha representaciòn intermedia puede a su vez ser ambigua y/o indeterminada en varias formas.

En sus libros "Syntactic Structures" y "Aspects of the Theory of Syntax" ([CHO62], [CHO76]), Chomsky propone que las gramàticas pueden tener una organizaciòn tripartita. La primera parte consiste en una gramàtica de estructura de frase que genera cadenas de morfemas que representan oraciones activas declarativas simples, cada una con un marcador de frase o àrbol de derivaciòn asociado. La segunda parte consta de una secuencia de reglas transformacionales que forman representaciones de toda la variedad de oraciones. La tercera parte consta de una secuencia de reglas morfofonèmicas que mapean la representaciòn de cada oraciòn a una cadena de fonemas.

En "Aspects of the Teory of Syntax", Chomsky revisa estos aspectos conocidos bajo el nombre de "teoria estandar de la gramàtica generativa", donde la generaciòn de oraciones parte de una gramàtica libre de contexto que genera la estructura de la oraciòn y es seguida por una selecciòn de palabras de una estructura a partir de un lèxico. Se dice que la gramàtica libre de contexto y el lèxico forman la base de la gramàtica, y su salida es precisamente la estructura profunda. El sistema de reglas transformacionales mapea las estructuras profundas a estructuras superficiales. Las partes base y transformacional de la gramàtica forman su componente sintàctico. El sonido de la oraciòn se determina por su estructura superficial y es interpretada por el componente fonològico de la gramàtica. La estructura profunda, interpretada por el componente semàntico, determina el significado de la oraciòn. De ahi que la aplicaciòn de reglas transformacionales a estructuras profundas deba preservar el significado.

La interpretaciòn semàntica es indispensable para poder utilizar la informaciòn ya sea para interpretaciòn o bièn para su traduccìòn de forma apropiada, y debe contemplar tambièn los aspectos pragmàticos a fin de superar las ambigüedades a que deba enfrentarse [CHA86]. Si tenemos por ejemplo:

> Pedro fuè a la tienda. Encontrò la leche en el pasillo. Pagò por ella y se fuè.

El problema a que nos enfrentamos aquì està en decidir a què se refiere el "ella" de la tercera oraciòn, a la tienda o a la leche. Se pueden tener casos mucho màs complejos por tener varias interpretaciones vàlidas posibles y no muy obvias dentro del texto.

Las transformaciones que se lleven a cabo en una gramàtica no deben cambiar el significado, pues si el significado cambiara, entonces la estructura profunda no representarla todo el significado pues algunos serian creados por las transformaciones [CHA86]. Esta suposiciòn controversial simplifica mucho el manejo de la interpretaciòn semàntica, pues permite ignorar las reglas de transformaciòn al considerar como se hace la interpretaciòn semàntica. Es necesario que el significado se encuentre especificado en el diccionario o lèxico.

La teoria de la semàntica composicional [CHA86] establece que la razòn por la cual somos capaces de entender un nùmero grande y arbitrario de oraciones, muchas de las cuales puede que jamàs hayamos escuchado, es porque en primer lugar somos capaces de analizarlas sintàcticamente utilizando un nùmero relativamente pequeño de reglas

sintácticas y, en segundo lugar podemos derivar el significado de todo a partir de los significados de las partes. Para ello necesitamos especificar tres cosas.

- El significado de las palabras individuales.
- Cómo los significados de las palabras individuales se combinan para formar el significado de los grupos de palabras.
- Cómo se enlaza todo esto con nuestro analizador sintáctico.

El enlace entre las reglas semánticas y el análisis sintáctico se lleva a cabo tipicamente de dos formas [CHA86]: Una de ellas es crear la representación interna despues de efectuar el análisis sintáctico. Ello tiene ciertas ventajas, tales como el no desperdicio de tiempo en la creación de una representación interna para un análisis gramatical que eventualmente falla. Sin embargo, cuando una oración tiene más de una forma sintáctica, es posible que la semántica guíe al analizador gramatical, es decir, la decisión de si una cierta regla transformacional debe o no ser aplicada descansa fuertemente en la información semántica que se tenga. La otra forma es construir la representación interna al mismo tiempo que se establecen las relaciones sintácticas. Esto facilita la interacción entre la semántica y la sintaxis.

I.7 FORMALISMOS PARA EL PROCESAMIENTO DE LENGUAJE NATURAL

A continuación se describen brevemente las propiedades de algunos de los formalismos más usados en el procesamiento de lenguaje natural.

I.7.a.- Comparación de Palabras (Keyword Matching):

Este es un formalismo sencillo y austero que consiste simplemente en comparar las palabras dadas, con una serie de patrones (templates) de palabras que la computadora es capáz de interpretar [RIC85]. Tienen la ventaja de permitir el uso de oraciones de gramática incierta o poco usual.

I.7.b.- Diagramas de Transición de Estado Finito (Finite State-Transition Diagram):

Una (Finite-State Transition Diagram) FSTD consiste en una serie de estados (nodos) con arcos que enlazan un estado con otro [BAR81]. Un estado es designado como el estado inicial y un subconjunto de estados como final; los arcos se etiquetan con los terminales de la gramática. Se dice entonces que el dispositivo acepta una cadena de palabras solamente si, partiendo del estado inicial, siguiendo las palabras, se puede alcanzar un estado final al terminar la oración. Las FSTD únicamente pueden reconocer gramáticas regulares (tipo 3).

I.7.c.- Redes de Transición Recursiva (Recursive Transition Network):

Dado que las FSTD únicamente pueden reconocer gramáticas regulares (tipo 3), se desarrolló una extensión de las FSTD con un mecanismo de recursión que permitía manejar gramáticas libres de contexto y se les denominó Redes de Transición Recursiva (Recursive Transition Networks, RTN) y su característica principal es que las etiquetas de los arcos pueden incluir tanto símbolos terminales como no-terminales, estos últimos denotando el nombre de otra subred a la cual se le da el control temporal sobre el proceso de análisis ([BAR81], [WOO86]).

I.7.d- Redes de Transición Aumentada (Augmented Transition Network):

Algunos estudiosos de la materia concuerdan con Schank en que las gramáticas regulares y las gramáticas libres de contexto son insuficientes para manejar el lenguaje natural [BAR81].

Debido a esta consideración, las RTN se extendieron para generar un analizador más poderoso, las redes de transición aumentada (Augmented Transition Network, ATN), o Gramáticas de Redes de Transición Aumentada (ATN Grammars) desarrolladas por William Woods en 1970 [BAR81].

Las ATN son RTNs extendidas en tres formas [BAR81] [WOO86]:

1) Se adicionó un conjunto de registros para almacenar información tal como árboles de derivación parcialmente formados, para los saltos entre subredes.

ii) Los arcos pueden tener además pruebas asociadas a ellos que deben ser satisfechas previamente a la toma del arco.
iii) Se le pueden adicionar ciertas acciones al arco que se pueden ejecutar cuando èste se toma, generalmente para modificar la estructura de datos.

Las ATN resultan por tanto más poderosas por su grado de naturalidad y expresividad, sumamente adecuados para el análisis de lenguaje natural. Son muy apropiadas para el manejo de gramáticas transformacionales. Tienen un alto poder generativo, su representación y operatividad es muy eficiente, además de resultar ser un formalismo muy flexible para la experimentación, por lo que es muy usado por lingüistas investigadores. Las condiciones arbitrarias que se pueden poner en los arcos la hacen sensitiva al contexto. Una fuerte limitante de las ATN es su alta dependencia en la sintaxis, además el mecanismo tipo 'top-down' (de arriba a abajo) no determinístico es ineficiente cuando sigue trayectorias de derivación erróneas ([BAR81], [MAC87], [RIC85], [WOO86]).

Las ventajas que proporcionan las ATN como formalismo para el PLN sobre formalismos anteriores son [WOO86]:
- Lucidez.
- Poder generativo.
- Eficiencia en su representación.
- Captura regularidades del lenguaje.
- Eficiencia de operación.
- Flexibilidad para experimentación.

Shapiro [SHA82] propone una extensión para las ATN a las que denomina "Generalized ATNs", tal extensión permite una fácil generación y análisis de redes semánticas.

I.7.e.- Redes de Transición Aumentada en Cascada (Cascaded ATN Grammars):

Las Redes de Transición Aumentada en Cascada (RTAC, en inglés CATN) desarrolladas por Woods [WOO80], son una extensión del formalismo de las redes de transición aumentadas. Están basadas en las gramáticas de estructura de frase (Phrase Structure Grammars).

Las CATN permiten descomponer un ambiente complejo de entendimiento de lenguaje natural en una secuencia de Redes de Transición Aumentada que cooperan bajo dominios separados de responsabilidad y acción [WOO80].

Se les denomina en cascada porque cada estado o transductor ATN toma su entrada a partir de la salida del

estado o transductor ATN previo, haciendo más sencillo, versátil y poderoso el proceso de análisis gramatical (parsing) del lenguaje. Tienen además la ventaja de que en general trabaja mejor y tiene mayor poderío que una ATN recursiva.

Woods asegura además [WOO80], que "...la experiencia con las gramáticas de ATN para el lenguaje natural indica que todo aquello que una gramática transformacional del lenguaje natural hace, puede ser hecho con una ATN simple, por lo que parece ser que no se necesita otra cosa mas que un número finito de estados de una cascada".

I.7.f.- Marcos de Minsky (Minsky's Frames):

Marvin Minsky propuso en 1975 una metodología de representación conocida como teoría de marcos (frame theory) [MAC85]. Un marco es una estructura utilizada para representar una situación estereotipada (contexto), el marco contiene una serie de nodos interconectados jerárquicamente. Los nodos superiores del marco contienen la información más estable del marco y conforme se desciende en jerarquía se obtienen detalles específicos. El marco debe contener además, datos que indiquen qué cosas se pueden esperar y qué hacer si una situación no es como se esperaba originalmente.

I.7.g.- Libretos (Scripts):

Schank, Abelson, et al. desarrollaron en 1977 el formalismo conocido como Libretos (Scripts). Los Libretos se basan en la teoría de las Dependencias Conceptuales de Schank [MAC85].

De acuerdo con Schank, "... un libreto es una estructura que describe secuencias propias de eventos en un contexto particular. Un libreto está formado por compartimientos y requisitos sobre lo que puede entrar en los compartimientos. La estrucura es una unidad interconectada, y lo que hay en un compartimiento afecta a lo que puede haber en otro. Los libretos manejan sucesos diarios. No están sujetos a muchos cambios ni proporcionan los elementos para manejar situaciones completamente nuevas. Así, un libreto está predeterminado, esto es, consiste en una secuencia estereotipada de acciones que define una situación bien conocida".

I.7.h.- Redes Semànticas (Semantic Networks):

La red semàntica desarrollada por Ross Quillian [MAC85], es una gràfica dirigida en la cual los nodos representan conceptos y los arcos relaciones entre ellos. Dos caracteristicas importantes de las redes semànticas son:

- Es intuitivamente sencillo.
- Se adapta especialmente bien a la recuperaciòn de informaciòn.

I.7.i.- Procesador Sintàctico General (General Syntactic Processor):

El Procesador Sintàctico General (PSG, en inglès GSP) desarrollado por Ronald Kaplan es un sistema versàtil para la generaciòn y anàlisis de cadenas en lenguaje natural [BAR81]. Sus estructuras de datos son intuitivas y sus estructuras de control relativamente fàciles de implementar. El GSP puede emular directamente otros analizadores, incluyendo las ATN y el generador de texto de Friedman. El GSP sintetiza las caracteristicas formales de diferentes mètodos de anàlisis.

La estructura de datos bàsica del GSP es el marco (frame), en esta estructura se puede representar tanto la gramàtica como la oraciòn de entrada. Un marco puede ser representado como un àrbol modificado en dos aspectos [BAR81]:

i) Los arcos del àrbol han sido rearreglados para producir un àrbol binario.
ii) Los nodos y los arcos han sido intercambiados.

El marco tambièn puede usarse para representar una cadena de àrboles o bosque, es decir, un conjunto de àrboles disjuntos, dando como resultado un conjunto de mùltiples interpretaciones para una palabra o frase dada.

El marco es solamente la estructura de datos y no està directamente relacionada con la gramàtica, ùnicamente sirve como el "pizarròn" bajo el cual opera la gramàtica. Tiene caracteristicas tales como recursiòn, "backtracking" y movimiento del apuntador a travès de la oraciòn de entrada.

El GSP provee una estructura simple dentro de la cual se pueden describir muchos sistemas de procesamiento de lenguaje, y sus primitivos son apenas suficientes para la representaciòn de la mayoria de los elementos de un analizador basado en sintaxis. Su estructura permite que se use como herramienta para comparar y evaluar diferentes sistemas.

I.7.j.- DIAGRAM

DIAGRAM es una gramática de estructura de frase aumentada (sensitiva al contexto) utilizada en sistemas de inteligencia artificial para la interpretación de diálogos en inglés, [ROB86]. Se usa como una herramienta en investigación para encontrar qué estructuras y procesos son necesarios para la interpretación de diálogos.

Una premisa básica es que los participantes del diálogo interpreten las expresiones de cada uno de los demás tomando en cuenta no solamente los significados de condicionales ciertas de lo que se dice, sino también su intención, meta, plan, creencias, estado de conocimiento y foco de atención, según el texto y contexto.

DIAGRAM es solamente una gramática incompleta del inglés. Analiza tanto frases simples como compuestas y puede extenderse independientemente de un dominio particular de aplicación. Además el sistema realiza todo el análisis sintáctico que corresponde a las ambiguedades semánticas que no provienen de ambigüedades léxicas.

I.7.k.- Lógica de Predicados de Primer Orden:

La idea es utilizar la lógica como un formalismo que sirva para la construcción de una estructura conceptual para el procesamiento de lenguaje natural. Esto se logra traduciendo la información que está en lenguaje natural a una representación de la forma de lógica de predicados de primer orden; también se les conoce como Gramáticas Lógicas (Logic Grammars, LG) ([CHA77], [DAH81], [SAN71]).

Tiene la ventaja de la fácil representación en la computadora, puede usarse como una teoría de inferencia y conocimiento y tiene la posibilidad de manejo tanto de sintaxis como de semántica. Además, es facilmente implementable en los lenguajes Lisp y Prolog, dadas las propiedades naturales de los mismos.

I.7.l.- Gramáticas de Metamorfosis
(Metamorphosis Grammars):

La Gramática de Metamorfosis (Metamorphosis Grammars, MG) es un formalismo basado en la lógica de predicados y se ha demostrado que es muy útil en el procesamiento de lenguaje natural. Fué desarrollada por Colmerauer en 1975, y está basada en la gramática tipo-0 de Chomsky ([DAH81], [PER80]). Son una extensión de las Gramáticas Lógicas.

Se basa en reglas de sustitución muy poderosas, y tiene las siguientes características [DAHB1]:

- Se pueden describir reglas de sustitución sensitivas al contexto.
- Cualquier subcadena (substring) de cualquier longitud se puede sustituir en cualquier otra cadena (string).
- Los símbolos de la gramática pueden incluir argumentos.
- Se deben especificar las condiciones para la aplicación de reglas.
- La generación y análisis gramatical de las oraciones son provistas por el procesador.

Las gramáticas de Metamorfosis pueden ser consideradas como una alternativa conveniente al formalismo de las Gramáticas Lógicas (Logic Grammars). Sus reglas (libres de contexto) se pueden expresar como cláusulas de Horn o definida (ver DCG a continuación), lo cual implica que son fácilmente representables en Prolog.

I.7.m.- Gramáticas de Cláusula Definida (Definite Clause Grammars):

Las Gramáticas de Cláusula Definida (GCD, en inglés DCG) o de cláusulas de Horn son un formalismo apropiado para el procesamiento tanto de lenguaje natural como artificial, descrito por Colmerauer y Kowalsky en 1975. Expresan la gramática en forma lógica y son una extensión de las Gramáticas Libres de Contexto (Context-Free Grammars, CFG) ([JIM86], [PER80]). Forman parte de las Gramáticas Lógicas.

Entre sus principales ventajas se cuenta con que, dado que las DCG son una extensión natural de las CFG, adopta todas sus características de la teoría de lenguaje, lo cual implica que la traducción de una CFG en DCG sigue una generalización simple. Además, Prolog resulta ser un lenguaje que funciona adecuadamente como analizador de los lenguajes descritos por una DCG ([JIM86], [PER80]).

Pereira y Warren [PER80], afirman que las DCG son al menos tan eficientes como las ATN, con la ventaja de que las DCG son más claras, concisas y poderosas.

Dado que las DCG son uno de los formalismos base para esta tesis, en el capítulo III se dará una explicación mas completa de su filosofía y características.

I.7.n.- Gramàticas de Extraposiciòn (Extraposition Grammars):

Este formalismo es una extensiòn de las Gramàticas de Clàusula Definda (DCG), y son tambièn descritas en base a clàusulas lògicas de Horn. Su caracterìstica principal consiste en que los constituyentes de extraposiciòn izquierda (conocidas asì en la gramàtica transformacional) son màs fàciles de describir que en los otros formalismos basados en clàusulas definidas [PER81]. Tambièn son una extensiòn de las LG.

La extraposiciòn a la izquierda ha sido usada por los lingüistas para describir la forma de las oraciones interrogativas en clàusulas relativas, por lo menos en lenguajes tales como Francès, Inglès, Portuguès y EspaÑol. El propòsito de las Gramàticas de Extraposiciòn (XG) es expresar dicha extraposiciòn de manera clara y concisa [PER81].

Pereira indica en su articulo "Extraposition Grammars" [PER81] que "La extraposiciòn a la izquierda en una oraciòn de lenguaje natural ocurre cuando un subconstituyente de algùn constituyente se pierde y algùn otro constituyente a la izquierda del incompleto, representa de alguna forma el constituyente perdido".

La diferencia fundamental entre las reglas de las Gramàticas de Extraposiciòn (XG) y las reglas de las Gramàticas de Clàusula Definida (DCG) consiste en que la parte izquierda de una regla de una Gramàtica de Extraposiciòn (XG) puede contener varios simbolos.

I.7.o.- Gramàticas de Estructura Modificante (Modifier Structure Grammars):

Las Gramàticas de Estructura Modificante (GEM, en inglès MSG) son un formalismo para la representaciòn en lògica de gramàticas para el procesamiento de lenguaje natural [DAH83].

Las MSG, al igual que las DCG, las XG y las MG son gramàticas lògicas fàcilmente implementables en Prolog y han demostrado ser una base sòlida para trabajar construcciones lògicas complejas del lenguaje natural, tàles como la coordinaciòn (la construcciòn con conjunciones tales como 'y') [DAH83].

Verònica Dahl [DAH83], establece que este formalismo es ùtil aùn fuera del procesamiento de lenguaje natural "... porque las nociones de estructura sintàctica e

interpretaciòn semàntica estàn màs restringidas que en muchos sistemas previos...".

Las MSG (basadas en las XG), son verdaderas gramàticas lògicas, puès se pueden traducir directamente en sistemas de clàusulas de Horn, es decir, el formato de Prolog. Un nuevo elemento de las MSG es que la formaciòn de las estructuras de anàlisis de las oraciones se hace mayormente implicita en el formalismo de la gramàtica. La estructura de anàlisis asociada a la oraciòn en una MSG puede ser considerada un àrbol de estructura de frase, por lo que se pretende que el tèrmino "Gramàtica de Estructura Modificante" sea paralela al tèrmino "Gramàtica de Estructura de Frase"; y si se le restringe la extraposiciòn y la coordinaciòn, entonces se tratarà de una Gramàtica de Estructura de Frase libre de contexto.

I.7.p.- Gramàticas de Lògica Restringida (Restricting Logic Grammars):

Las Gramàticas de Lògica Restringida (GLR, en inglès RLG), son un formalismo basado en las gramàticas lògicas que imponen algunas de las restricciones sugeridas en la teoria lingüistica Chomskiana reciente (ver [LEE85], capitulo 17), y se puede representar eficientemente en Prolog [STA87].

El formalismo se basa en las XG. Su extenciòn incluye el uso de un nuevo tipo de reglas denominadas reglas de cambio (switch rules),que son una extensiòn de las reglas de movimiento izquierdo, con la finalidad de permitir una mayor construcciòn lingüistica, y un movimiento a la derecha restringido, [STA87].

I.7.q.- Gramàticas de Ranura (Slot Grammars):

A este formalismo se le denomina Gramàtica de Ranura (GR, en inglès SG) porque està organizada alrededor de ciertas relaciones gramaticales denominadas ranuras (slots) y las reglas necesarias para llenarlas [McC80]. El analizador trabaja en base a una filosofia de abajo-hacia-arriba (Bottom-up), y mantiene para cada frase una lista denominada 'ranuras disponibles' (available slots, ASLOTS), por lo que la frase puede crecer teniendo una de las ranuras en la lista de ASLOTS llena por una frase adjunta permisible.

Los sistemas creados bajo este formalismo pueden manejar una gràn variedad de fenòmenos gramaticales, tales

como: dependencias de verbo y estructuras WH- (why, when, where, etc.) [McC80].

Los SG se basan en las Gramàticas de Estructura de Frase Aumentada (APSG) desarrolladas por Heidorn en 1972-75, en las Gramàticas Sistèmicas y en las ATN.

I.7.r.- Gramàtica de Estructura de Frase Generalizada (Generalized Phrase Structure Grammar):

La Gramàtica de Estructura de Frase Generalizada (GEFG, en inglès GPSG) es un formalismo propuesto por Gazdar en 1978. Las GPSG incrementan el poder descriptivo sin incrementar el poder generativo de las Gramàticas de Estructura de Frase ([GAZ85], [JOS82]). De hecho, Gazdar pretende restringir el poderio de las GPSG al de los lenguajes libres de contexto. Gazdar introduce dos nociones importantes en sus GPSG:

1.- Categorias con huecos (gaps) y un conjunto asociado de reglas derivadas y reglas enlazadas.
2.- Metareglas para derivar reglas a partir de otras.

Las categorias con huecos (gaps) y las reglas asociadas, pràcticamente no incrementan el poder generativo con respecto al de las Gramàticas Libres de Contexto.

Las metareglas, a pesar de estar en cierto modo restringidas, sì incrementan el poder generativo debido a que, por ejemplo, una metaregla puede generar un conjunto infinito de reglas libres de contexto que pueden generar un lenguaje sensitivo de contexto restringido. Las metareglas en las gramàticas actuales escritas en base a una estructura de GPSG estàn lo suficientemente restringidas para no incrementar el poder generativo.

Se considera que las GPSG son un formalismo adecuado para llevar a cabo la representaciòn y anàlisis sintàctico-semàntico de los lenguajes naturales.

La GPSG es el segundo formalismo usado en el desarrollo de esta tesis, por lo que se discutirà con mayor detalle en el capitulo IV.

I.7.s.- Arboles de Estructura de Frase (Phrase Structure Trees):

Joshi y Levy [JOS82], proponen un formalismo que consiste en una extensiòn de las GPSG de Gazdar, los Arboles

de Estructura de Frase (Phrase Structure Trees, PST). Este formalismo incrementa el poder descriptivo de las GPSG.

Los PST tienen la caracteristica de retener toda la informaciòn estructural que tiene grandes implicaciones en la generaciòn de procedimientos de inferencia gramatical.

I.7.t.- Gramàticas de Caso (Case Grammars):

La Gramàtica de Caso se fundamenta en que existen casos denominados "deep cases", que son la forma bàsica en que una frase nominal o una frase preposicional pueden relacionar un verbo, por lo tanto todos los verbos deben seleccionar su(s) caso(s) de entre estos cuatro.

Un caso es el que indica o señala a la persona o "Agente" responsable de la acciòn. Otro caso es el "beneficiario" o la persona que se beneficia de la acciòn. Los casos "fuente y destino" son donde la informaciòn inicia o termina. Y el caso "paciente" es el objeto afectado de alguna forma por la acciòn.

El concepto de la gramàtica de caso ha sido aceptada en gràn medida para el anàlisis del idioma japonès, debido a que, segùn ellos, no han logrado encontrar la manera de abarcar muchos de los fenòmenos lingüisticos (excepciones) de manera sistemàtica para la traducciòn automàtica con otros formalismos.

I.8 SISTEMAS DE INTERPRETACION DE LENGUAJE NATURAL

En èste tema se mencionan brevemente algunos de los sistemas clàsicos y los sistemas de desarrollo reciente en esta àrea. De ser posible se mencionarà en que tipo de formalismo se basa el sistema y sus principales caracteristicas.

Robert Lindsay desarrollò en 1963 el sistema SAD-SAM (Syntactic Applier and Diagrammer-Semantic Analyzing Machine) [BAR81]. El sistema de pregunta-respuesta acepta oraciones en inglès, està basado en las CFG y fuè programado en IPL-V. y manejaba alrededor de 1700. palabras. SAD construia un àrbol de derivaciòn y se lo enviaba a SAM, el cual lo analizaba para obtener la informaciòn semàntica relevante a fin de construir àrboles genealògicos y contestar las preguntas formuladas.

En 1964 se desarrollaron en el Instituto de Tecnologia de Massachusetts (Massachusetts Institute of Technology, MIT) el sistema STUDENT (por Daniel Borrow) dedicado a la soluciòn de pequeños problemas algebràicos a partir de un sistema de patrones; y el sistema SIR ò "Semantic Information Processing" (por Bertram Raphael) que realizaba deducciones sencillas a partir de una serie de hechos ([BAR81], [MAC85]).

A principios de los años 60, Bert Green et al. desarrollaron el sistema BASEBALL que consiste bàsicamente en un sistema de recuperaciòn de informaciòn que acepta oraciones en inglès restringido y accesa una base de datos para obtener la informaciòn requerida. Està escrito en IPL-V ([BAR81], [GRE86]). Tambièn a principios de los años 60, Terry Winograd desarrollo en el MIT el sistema SRHDLU que pretendia ser un sistema de PLN de universo limitado, especificamente la manipulaciòn de un brazo mecànico. El sistema recibia las ordenes en inglès. Fuè implementado en LISP y MICRO-PLANNER. Desafortunadamente resultò ser tan complicado que solamente fuè utilizado por Winograd como investigaciòn doctoral ([BAR81], [MAC85]).

Joseph Weizenbaum desarrollò en 1966 el sistema ELIZA, basado en un esquema de patrones y enfocado a modelar un diàlogo entre un paciente (usuario) y su psiquiatra (ELIZA). Este sistema detectaba contextos en base a las respuestas dadas por el usuario. A pesar de que el diàlogo establecido resultaba estable y convincente, por sus caracteristicas (muy pobre anàlisis sintàctico y semàntico principalmente) ha dejado de ser considerado por muchos como sistema de PLN ([BAR81], [MAC85]).

William Woods desarrollò en 1967-72 el sistema LUNAR. Este es un sistema de pregunta-respuesta que se dedica a obtener la informaciòn de un banco de datos sobre muestras lunares en la NASA. Està basado en las ATN y lleva a cabo un anàlisis sintàctico exhaustivo. Fuè implementado en LISP ([BAR81], [MAC85]).

Isard y Longuet-Higgins [ISA71], desarrollaron en la Universidad de Edinburgo un sistema capàz de responder preguntas en inglès que pueden involucrar ciertos

cuantificadores (some, every, and, no). Las preguntas son traducidas a forma de cálculo de predicados de primer orden e interpretadas bajo una representación de estructura profunda, haciendo únicamente su análisis sintáctico. Sólo trabaja con oraciones activas simples.

Roger Schank et al. desarrollaron en 1975 el sistema MARGIE (Meaning Analysis, Response Generation, and Inference on English) en el laboratorio de IA de Stanford [BAR81]. El sistema pretendía proveer un modelo intuitivo de los procesos de entendimiento de lenguaje natural. Este sistema sirvió para demostrar la viabilidad de su teoría de las dependencias conceptuales. En 1977 desarrolló junto con Robert Abelson et al., los sistemas SAM (Script Applier Mechanism) y PAM (Plan Applier Mechanism) en la Universidad de Yale para demostrar el uso de libretos (scripts) y planes (plans) en el entendimiento de historias sencillas. Los libretos de Schank son similares a los marcos de Minsky o los esquemas de Bartlett desde el punto de vista de que pueden usarse para anticiparse a aspectos de los eventos que representa ([BAR81], [CUL86]).

Verónica Dahl et al. [DAH81] (Universidad de Buenos Aires), proponen un sistema en el que se utiliza la lógica (sistémas lógicos) como un lenguaje de preguntas (query language) interno y como una herramienta de programación. El sistéma lógico incluye algunas extensiones de la lógica de predicados estandar que permite manejar características sintácticas y semánticas.

A su vez, Dahl y sus colaboradores desarrollaron también un analizador lógico-programado que traduce el español a dicho sistema. Analiza la concordancia semántica y la correcta formación sintáctica, además de detectar ciertas presuposiciones, resolver algunas ambigüedades y reflejar relaciones entre conjuntos. Se basa en el formalismo de las Gramáticas de Metamorfosis (Metamorphosis Grammars, MG). El sistema está desarrollado en Prolog.

Warren y Pereira [WAR82], desarrollaron un sistema basado en las Gramáticas de Extraposición (XG) e implementado en Prolog. El sistema se llama Chat-80, y está dedicado a contestar preguntas formuladas en inglés. El diseño lo hace fácilmente adaptable y eficiente en una gran variedad de aplicaciones. La filosofía del sistema consiste en traducir la pregunta formulada en inglés al subconjunto lógico de Prolog, y la expresión lógica resultante la transforma mediante un algoritmo en lo que denominaron optimización de preguntas (query optimisation) en una base de datos relacional. La representación en Prolog se ejecuta finalmente para poducir la respuesta.

Michael Main [MAI83] (Universidad del estado de Washington), también desarrolló un sistema de pregunta-respuesta para lenguaje natural basandose en la Semántica Denotacional propuesta por Scott-Strachey, y se enfoca a la comunicación con ingenieros y programadores. El sistema se basa en el idioma inglés y hace énfasis en la interpretación semántica de las preguntas, la cual se toma como una función del conjunto de universos disponibles al conjunto de respuestas posibles. Para llevar a cabo la interpretación semántica hace uso del Algebra Lambda, categorizaciones sintáctico-semánticas y el manejo de objetos semánticos.

Weischedel y Sondheimer [WEI83], proponen el uso de metareglas y una estructura de control bajo la cual llamarlas a modo de estructura para el procesamiento de entradas mal-formadas (ill-formed) desde el punto de vista léxico, sintáctico, semántico y pragmático. La parte izquierda de una metaregla diagnostica un problema como una regla violada y la parte derecha suaviza la violación y establece la forma en que el proceso puede reducirse, de ser posible, en su totalidad. Los errores que contempla son básicamente la omisión de artículos, homónimos, errores tipográficos, palabras desconocidas, restricciones, personificación, etc.. Por otro lado, Fass y Wilks [FAS83], analizan también las oraciones mal-formadas y su relación con la semántica preferencial (Preference Semantics).

Granger [GRA83] desarrolló en la Universidad de California el sistema NOMAD. Este sistema se dedica al entendimiento de textos "defectuosos", es decir, es un sistema que detecta y corrige errores cometidos durante la interpretación de texto mal-formado sintáctica y semánticamente. Esta diseñado para operar en el dominio restringido de los mensajes barco-tierra (ship-to-shore). Trabaja interactivamente con el usuario para codificar el mensaje en una forma legible de base de datos.

Masaru Tomita [TOM87], desarrolló en la Universidad de Carnegie-Mellon un algoritmo para el análisis de gramáticas libres de contexto aumentadas. El algoritmo es básicamente un analizador gramátical de izquierda-a-derecha (Left-to-Right, LR) que genera inicialmente una tabla de análisis gramatical LR de desplazamiento reducido a partir de una gramática libre de contexto aumentada. Una característica importante es que puede manejar gramáticas ambiguas. Su eficiencia se debe principalmente a la introducción de un nuevo concepto, el 'stack' gráfico-estructurado (graph-structured stack), que permite al analizador LR mantener múltiples análisis sin analizar ninguna parte de la entrada dos veces de la misma forma. El análisis lo realiza en-línea (on-line), es decir en cuanto el usuario teclea la primera palabra de una oración, sin esperar que la teclee por completo. Está implementado en Lisp.

I.9 SISTEMAS DE TRADUCCION AUTOMATICA

En este tema, se describirán algunos de los sistemas existentes enfocados a la traducción automàtica. Al igual que en el capitulo anterior, se mencionarán las caracteristicas principales de los sistemas y de ser posible, el formalismo en que se basan.

David Weber y William Mann [WEB81], desarrollaron un proyecto muy interesante que consistió en investigar la factibilidad de utilizar la computadora para que realice la mayoria de los cambios necesarios para adaptar los textos de un dialecto dado a otros dialectos, en este caso del Quechua. El sistema se ha utilizado para la traducción entre cinco dialectos. Los resultados obtenidos demuestran que el sistema (i) permite que los informantes no bidialectos puedan producir las adaptaciones adecuadas sin gran ayuda del traductor; (ii) mejora la calidad del texto resultante y; (iii) reduce el tiempo y esfuerzo tanto en la adaptación como en la preparación del manuscrito.

Bennet y Slocum [BEN85] desarrollaron en el Centro de Investigaciones Linguisticas (LRC) de la Universidad de Texas en Austin, el sistema de traducción por computadora de alta calidad totalmente automàtico conocido como METAL. El sistema traduce ùnicamente textos del Alemàn al Inglès, pero se està trabajando actualmente en la extensión a los lenguajes Español y Chino.

Biewer, Fènegrol, Ritzke y Stegentritt [BIE85], de la Universitat des Saar Saarlandes (Alemania Federal), desarrollaron un sistema modular multinivel para la traducción Francès-Alemàn. El sistema se llama ASCOF (Analysis and Syntesis Of French by means of COMSKEE (COMputing and String Keeping language)). ASCOF adopta la división clàsica del proceso de traducción, es decir, anàlisis-traducción-síntesis. El anàlisis se realiza en tres fases: (i) anàlisis morfològico, (ii) identificación de frases sintácticas no complejas y de la macroestructura de la oración y, (iii) la determinación de la estructura de frases sintácticas complejas y las funciones sintácticas en que el criterio sintáctico y semántico se usan. Se apoya en redes semánticas y realiza ùnicamente traducciones entre el Francès y el Alemàn.

Nagao, Tsujii y Nakamura, de la Universidad de Kyoto, describen en su articulo "The Japanese Government Project for Machine Translation" [NAG85], el proyecto fundado por la Agencia de Ciencia y Tecnologia conocido como "Research on Fast Information Services between Japanese and English for Scientific and Engineering Literature". El propósito del

proyecto es demostrar la factibilidad de la traducción por computadora de artículos de índole científico y técnico entre los dos idiomas, y como resultado establecer un sistema de intercambio de información rápida para ese tipo de artículos. Este proyecto se está llevando a cabo bajo una estrecha cooperación entre cuatro organizaciones:

i) La Universidad de Kyoto, responsable de desarrollar el software del sistema para la parte central del proceso de traducción (el sistema de escritura de la gramática y el sistema de ejecución); los sistemas de gramáticas para el análisis-traducción-síntesis; la especificación detallada sobre la información escrita en los diccionarios de palabras y los manuales de trabajo para la construcción de estos diccionarios.

ii) La "Electrotechnical Laboratories" (ETL) es la responsable de la entrada-salida del texto de la máquina de traducción, el análisis y síntesis morfológico, y la construcción de los diccionarios de verbos y adjetivos basados en los manuales de trabajo preparados en la Univ. de Kyoto.

iii) El Centro de Información de Japón para la Ciencia y la Tecnología (JICST), está encargada del diccionario de nombres (noun dictionary) y del procesamiento de términos técnicos especiales en los campos científico y técnico.

iv) El Sistema de Investigación de Procesamiento de la Información (RIPS) bajo el auspicio de la Agencia de Tecnología Ingenieril, es la responsable de completar el sistema de la máquina de traducción, incluyendo las interfaces hombre-máquina del sistema desarrollado en la Univ. de Kyoto que permitan la pre y postedición, el acceso a las reglas gramaticales y el mantenimiento del diccionario.

Actualmente el proyecto procesa únicamente textos de las áreas de electrónica, ingeniería eléctrica y campos de la ciencia de la computación.

Vasconcellos y León [VAS85] de la Organización Panamericana de la Salud, desarrollaron un sistema de traducción Español-Inglés, y otro sistema de traducción Inglés-Español, conocidos como SPANAM y ENGSPAN respectivamente. SPANAM ha estado en operación en dicha organización desde 1980 y ENGSPAN desde 1985. Los dos sistemas están escritos en PL/1, se basan en el formalismo de las ATN y efectúan un análisis sintáctico y semántico de la información. Son capaces de procesar alrededor de 50,000 palabras. SPANAM fué concebido originalmente como un sistema de traducción directa, mientras que ENGSPAN es un sistema de transferencia sintáctica y léxica.

Johnson, King y Tombe [JOH85] desarrollaron, junto con otros colaboradores, el sistema EUROTRA. El desarrollo de este sistema es un proyecto de la Comunidad Económica Europea que pretende producir un prototipo operacional relativamente pequeño de cada uno de los siete idiomas oficiales de la comunidad a cualquier otro de los mismos siete idiomas. El proyecto está dividido en tres fases:

i) 1983-1984: La definición del software básico a ser usado por todos los grupos y una definición preliminar de la estructura lingüística que servirá como la base del trabajo de la segunda fase.

ii) 1985-1986: La producción del primer sistema de trabajo, tomando los siete idiomas como lenguajes potenciales fuente y destino. El objetivo es demostrar la factibilidad del enfoque adoptado y, redifinir y estabilizar la estructura lingüística.

iii) 1987-1988: Construir el sistema prototipo real que cubra los siete idiomas y maneje un vocabulario de aproximadamente 20,000 entidades lèxicas por lenguaje.

Jimenez [JIM86], desarrolló en el Departamento de Ciencias de la Computación de la Universidad de Essex, un sistema de traducción Inglès-Español bidireccional. El sistema realiza la traducción de oraciones compuestas mediante análisis sintàctico, y está basado en el formalismo de las GPSG y principalmente de las DCG. Además propone una alternativa de extensión del sistema en base a las GPSG para llevar a cabo el anàlisis semántico de las oraciones a fin de obtener un sistema de traducción más eficiente. El sistema trabaja en una DEC-10 y está implementado en Prolog. De hecho este trabajo es el principio y la base para el desarrollo de la presente tesis.

I.10 SUMARIO

En este capitulo se mencionaron los aspectos generales referentes a la interpretación de lenguaje natural y a la traducción automàtica. Se habló brevemente sobre los antecedentes históricos de la traducción automàtica. Se mostraron los elementos sintàcticos y semánticos más

importantes a considerar para la creación de los formalismos usados como base para el desarrollo de los sistemas de procesamiento del lenguaje natural y los sistemas de traducción automática.

Se mencionaron además algunos de los primeros sistemas y de los sistemas actuales más significativos o representativos de procesamiento del lenguaje natural y de traducción automática, señalando sus principales características, su filosofía y en algunos casos sus formalismos base.

CAPITULO II

GRAMATICA

> Se puede hablar, sin embargo, sin conocer la gramàtica, y el que habla sin cononerla posee realmente una gramàtica, y habla de acuerdo con sus reglas, de las cuales, sin embargo, no tiene conciencia.
>
> Immanuel Kant.

II.1 INTRODUCCION

El anàlisis gramatical de cualquier idioma resulta siempre una labor extenuante y complicada, sin embargo, es posible ilustrar de manera muy aceptable, la naturaleza y comportamiento de un idioma (lenguaje) describiendo las caracteristicas màs generales e importantes de su gramàtica.

Naturalmente, los idiomas inglès y espaÑol no tienen el mismo origen. Mientras que el espaÑol es una lengua romance de raices primordialmente grecolatinas [TOR76], el inglès es una lengua perteneciente al grupo germànico de los idiomas indoeuropeos (anglosajòn). Sin embargo, al realizar el anàlisis gramatical de ambos idiomas es posible apreciar la semejanza entre sus estructuras. Tal semejanza resulta sumamente ventajosa para el estudio objeto de esta tesis, debido a que tanto el anàlisis de la semejanza y compatibilidad de ambas gramàticas, asi como la generaciòn de los procedimientos necesarios para la traducciòn resultan ser menos complicados de lo que en un principio se podria concebir.

Los estudios que han realizado los lingüistas y filòsofos primordialmente, han traido consigo el desarrollo de diversas teorias y tipos de gramàticas; de entre ellas,

la Gramàtica Generativa Transformativa, mejor conocida como Gramàtica Transformativa, ha sido la màs aceptada por describir de manera muy acertada la estructura de la lengua. La Gramàtica Transformativa fuè desarrollada por Noam Chomsky y fuè publicada en su obra "Syntactic Structures" en 1957 [CHO62].

Tanto los estudios de la sintàxis como de la semàntica (esta ùltima, estudiada a fondo muy recientemente) nos han permitido caracterizar acertadamente la estructura y el significado de los lenguajes. Desafortunadamente la semàntica sigue siendo un campo poco desarrollado actualmente. Por ello, un anàlisis semàntico completo resulta ser una labor algo màs que imposible, [NAG85].

En este capitulo se verà a grandes razgos la gramàtica transformacional tomando como base los idiomas inglès y espaÑol, asi como una breve descripciòn de la sintàxis, la semàntica y los elementos gramaticales de dichos idiomas.

II.2 ANALISIS GRAMATICAL

Para llevar a cabo el anàlisis gramatical adoptarè primero una definiciòn formal de gramàtica y posteriormente describirè las caracteristicas mas importantes que debe cumplir una gramàtica de este tipo, basada en la teoria transformacionalista de Chomsky.

Existen un sin nùmero de definiciones sobre lo que es una gramàtica (ver glosario), y considero como la màs completa la siguiente:

> "Gramàtica es un conjunto de reglas de estructura de frase para la derivaciòn de oraciones declarativas activas simples, combinadas con un conjunto de reglas transformacionales que, cuando se aplican a las oraciones derivadas por las reglas de estructura de frase, agregan, sustraen o modifican el orden dentro de ellas, o se combinan en formas complejas". [STO65].

Se puede decir que esencialmente una gramàtica es el conjunto de reglas que determinan el comportamiento de un lenguaje. La definiciòn formal proporciona elementos que permiten apreciar las caracteristicas de tales reglas.

Desde el punto de vista del uso de la lengua, se puede decir que la gramàtica es descriptiva; y desde el punto de vista de la estructura gramatical es prescriptiva.

Desde el punto de vista taxonómico [CHO76], la gramàtica consiste en una descripción de los aspectos superficiales en que una oración difiere de otra y una clasificación de sus diferencias. Nótese que nos dice poco o nada acerca de como tales oraciones son formadas. Algunas de las diferencias gramaticales del inglès y el español son [STO65]:

i) Orden de las palabras:

Tengo algo que hacer	I have something to do
Tengo que hacer algo	I have to do something
La bonita hija de Don Juan es amiga mia.	Don Juan's pretty daughter is a friend of mine.
La hija bonita de Don Juan es amiga mia.	Don Juan's pretty daughter is a friend of mine.

"Tengo algo que hacer" implica una obligación o compromiso. "Tengo que hacer algo" no implica una obligación, sino màs bièn un fuerte deseo.

ii) Entonación. Tiene tres señales principales:
(a) Une las frases en una sola unidad.
(b) Separa las frases que no son una sola unidad.
(c) Marca ciertos tipos de oraciones, tales como los interrogativos.

iii) Palabras de función. Cada palabra de una oración tiene probablemente una función que se puede especificar. Esta función es en ocasiones màs semàntica que gramatical, y en otras, màs gramatical que semàntica.

iv) Afijación. Este es un proceso por el cual los elementos de la palabra (prefijos, sufijos) estàn sujetos a una unidad lèxica, ya sea para señalar una función sintàctica (la 's de boy's) o para permitirle a una palabra funcionar en diferentes clases de palabras (en inglès: able, el -ity de ability; en español: capaz, la -idad de capacidad).

Según Stockwell [STO65], desde el punto de vista de los grados de agrupamiento de las unidades lèxicas en secuencia se le puede denominar a la gramàtica como Gramàtica de

Estructura de Frase (Phrase Structure Grammar). Esta postula que dada una secuencia de unidades lèxicas que constituyen una oración, es posible determinar què elementos estàn cercanamente relacionados con cuales otros, asi como especificar su grado de relación.

Por ejemplo, dada la oración:

The two boys were playing in the yard.

Una descripción de estructura de frase postula que "the two boys" y "were playing in the yard" son dos constituyentes inmediatos de la oración, y cada uno de ellos, a su vez, contiene constituyentes aùn màs inmediatos.

Las reglas de estructura de frase pueden verse como un modelo parcial de las reglas de formación por la cual se restringe la construcción de oraciones.

Por ejemplo, si para la oración anterior se tienen los siguientes elementos:

```
 NP = Noun Phrase
 VP = Full Verb Phrase
VP1 = Verb Phrase
AUX = Auxiliary
  D = Determiner
ADJ = Adjective
  N = Noun
 VI = Intransitive Verb
ADV = Adverb
PREP = Preposition
 TE = Tense
```

Y las reglas:

```
  S ---) NP + VP
 NP ---) D + (ADJ) + N +(-s)
  D ---) the, this
ADJ ---) two, new
  N ---) boy, girl, game, yard
 VP ---) AUX + VP1
AUX ---) TE (be + ing)
 TE ---) {PAST}
         {PRES}
VP1 ---) Vi + (ADV)
 VI ---) play, run
ADV ---) PREP + NP
PREP ---) in, on, at
```

donde los simbolos con mayùsculas indican simbolos no terminales, los simbolos con minùsculas indican simbolos

terminales, los parèntesis "()" implican que el elemento es opcional, y las llaves "{}" indican que se debe adoptar por uno u otro elemento.

El àrbol generado serà:

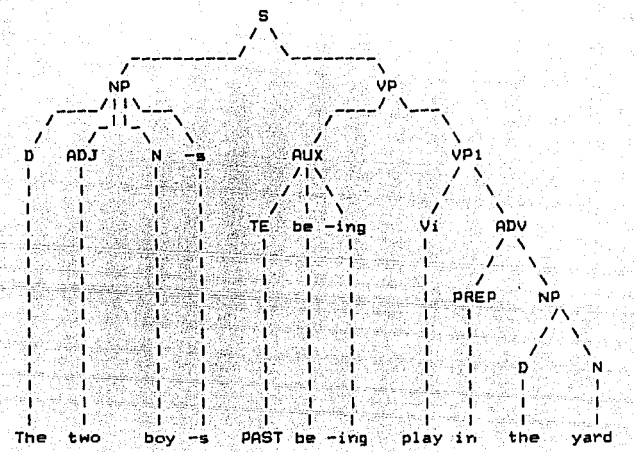


Figura II.2 -1-

Una gramàtica de estructura de frase es un conjunto ordenado de tales reglas, y constituyen una de las herramientas màs poderosas desarrollads para la descripciòn lingüistica.

Con la inclusiòn de reglas transformacionales se gana poder para mostrar las relaciones entre las palabras, y se eliminan muchas de las limitaciones significativas de los modelos gramaticales màs simples.

II.3 GRAMATICA TRANSFORMATIVA

II.3.1 CONCEPTOS

Como se mencionó en la introducción de este capitulo, la Gramàtica Generativa Transformativa, o Gramàtica Transformativa (GT) fue desarrollada por Noam Chomsky y la publicó en 1957 en su obra "Syntactic Structures". Según Chomsky [CHO76], "Una gramàtica de una lengua pretende ser una descripción de la competencia intrinseca del hablante-oyente ideal. Si la gramàtica es, además, perfectamente explicita, es decir, si no depende de la inteligencia y comprensión del lector; antes al contrario, proporciona un anàlisis explicito de lo que el lector pondria de su parte; podemos llamarla Gramàtica Generativa".

Al hablar de competencia, se hace referencia al nivel de conocimiento y dominio que de su lengua (gramàtica) tiene el hablante-oyente nativo.

En su libro, Chomsky hace dos aportaciones importantes ([AIT74], [CHO76]):

i) Cuestionó las metas aceptadas a través de las cuales la teoria lingüistica estaba orientada y, redefinió el propósito y funciones de una gramàtica.
ii) Especificó la forma que este nuevo tipo de gramàtica debe tener, y la llamó Gramàtica Transformacional.

Chomsky establece también que el sistema de reglas de la gramàtica puede dividirse en tres componentes, a saber: el componente sintàctico, el fonológico y el semàntico.

El componente sintàctico especifica un conjunto infinito de objetos formales abstractos, cada uno de los cuales incorpora toda la información correspondiente a una interpretación única de una oración concreta, y consta de dos partes:

(a) El componente de estructura sintagmàtica, cuyas reglas operan con el lèxico (diccionario) y sirven de instrucciones para la producción de la estructura profunda.
(b) El conjunto de reglas transformativas (reglas T) que realizan operaciones diversas sobre las estructuras profundas, convirtiéndolas en estructuras superficiales y aproximándolas a su forma definitiva.

El componente fonológico (no contemplado en esta tesis) determina la forma fonética de una oración generada por las reglas sintácticas, es decir, relaciona una estructura

generada por el componente sintàctico con una señal representada fonèticamente.

El componente semàntico determina la interpretación semàntica de una oración, es decir, relaciona una estructura generada por el componente sintàctico con una cierta representación semàntica. Opera sobre la estructura profunda de las oraciones.

Asi pues, tanto el componente fonológico como el semàntico son puramente interpretativos ([CHO76], [KAT79]).

II.3.2 GRAMATICA TRANSFORMACIONAL

La mayoria de los lenguajes tienen construcciones de estructura superficial diferente y estructura profunda similar [AIT74].

Si tenemos por ejemplo:

Pedro patea la pelota.

y por otro lado las derivaciones:

La pelota es pateada por Pedro.

Pedro no patea la pelota.

En una gramàtica transformativa, tales oraciones se derivan siempre de un solo núcleo o 'kernel'.

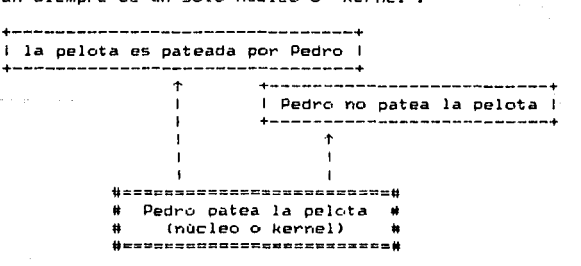


Fig. II.3 -1-

Como se observa, la estructura profunda 'kernel' (Pedro patea la pelota) es convertida o transformada en diferentes estructuras superficiales (la pelota es pateada por Pedro, etc.). Es decir, la gramática transformativa es una gramática que convierte estructuras profundas (representación lógica formular del significado) en estructuras superficiales (lo que oimos o vemos escrito) por medio de transformaciones.

Si se tienen por ejemplo los siguientes sintagmas:

Un vaso de leche.

Un vaso de cristal.

Ambos tienen la misma estructura superficial, pero en realidad son diferentes. Mientras que la primera se refiere a lo que contiene el vaso, la segunda se refiere al material del que está hecho el vaso.

Resumiendo, las reglas transformacionales permiten hacer lo siguiente [CHA86]:

1.- Poner elementos en los constituyentes de una oración.
2.- Mover constituyentes.
3.- Agregar constituyentes.
4.- Borrar constituyentes.

Una diferencia fundamental entre la teoría de la gramática transformativa y otras teorías [HAD75] es la convicción de que el propósito de una gramática no es simplemente el de clasificar los hechos observados, lo que de hecho habla la gente, sino caracterizar la capacidad intelectual que subyace a su uso de la lengua. Es decir, buscar la descripción del conocimiento intuitivo que de su propia lengua posee el hablante.

El término "generativa" se refiere al hecho de asignarle a cualquier oración de la lengua la descripción estructural correcta que muestra la clase de información que el hablante utiliza al interpretar esa oración, [CHO76].

Nota.- El desarrollo de la sintaxis y la semántica tratados en esta tesis, así como las teorías en las que se basa, se fundamentan en la gramática transformativa, por lo que es probable que se encuentren algunas discrepancias con textos que difieran de la teoría transformacional de Chomsky.

II.4 SINTAXIS

Para realizar el análisis sintáctico basado en la gramática transformacional, es suficiente considerar una gramática sencilla cuyas reglas constituyan el componente de Estructura Sintagmática (ES) a utilizar [HAD75].

```
1.        O ---) SN SV
2.       SV ---) aux verbal
3.      aux ---) t (M)
4.        t ---) {pasado}
                 {presente}
5.   verbal ---) {V (SN (pasiva))}
                 {cop prednom}
6. prednom ---) {adj}
                 {SN}
7.       SN ---) (det) N (pl).
```

donde:

```
O = oración
SN = sintagma nominal
SV = sintagma verbal (o predicativo)
aux = auxiliar
t = tiempo
M = modal
V = verbo
cop = cópula
prednom = predicado nominal
adj = adjetivo
det = determinante
N = nombre
pl = plural
```

Nótese que todas las reglas de la estructura sintagmática tienen la forma: X ---) Y Z , que significa: reescríbase (o reemplaze) a X como Y seguida de Z.

La función de las oraciones es simplificar la descripción reduciendo varias reglas a una.

Si se toma por ejemplo la oración:

i) Pedro pateó la pelota

y se realiza la sustitución en la gramática dada se obtiene:

ii) Pedro pasado patear l- pelota

La representación (ii) resulta inaceptable como oración en español (obsérvese que el verbo no concuerda con el sujeto, el artículo determinante no concuerda con su nombre, y el tiempo está representado por un símbolo en vez de por la terminación verbal abierta -eó). Esta representación resulta de la aplicación de reglas de estructura sintagmática y constituye una derivación de la estructura profunda "Pedro pateó la pelota".

El árbol estructural correspondiente es:

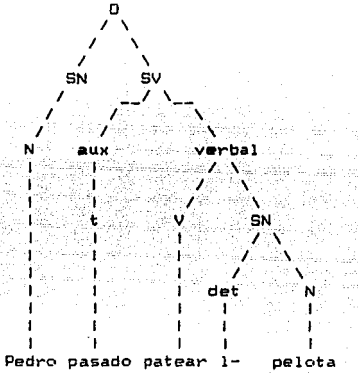


Fig. II.4 -1-

Si las reglas se escriben correctamente, entonces no solamente se producirá la estructura profunda de todas las oraciones aceptables (del español por ejemplo), sino que también se asignará la descripción correcta de su estructura gramatical.

A continuación se muestra como la oración:

Pedro puede patear la pelota

difiere de su estructura profunda:

Pedro pateó la pelota

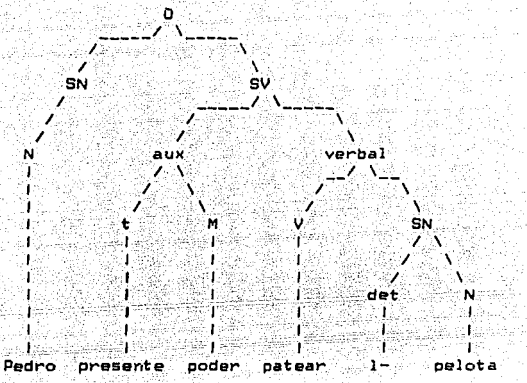


Fig. II.4 -2-

Obsérvese que de las reglas se puede derivar, de igual modo, la estructura profunda de la nueva oración.

Las principales caracteristicas de la estructura sintagmàtica son [HAD75]:

1) Introducen las categorias gramaticales necesarias para caracterizar las oraciones (del Español y el Inglès) y muestran la forma en que van juntas estas categorias para formar sintagmas y oraciones.
2) Introducen los elementos lèxicos y gramaticales.
3) Se ocupan de la estructura profunda.
4) Pueden utilizarse para mostrar la estructura profunda correcta (normalmente en forma de àrbol) de cualquier oración (del Español y el Inglès).

Las reglas transformativas (reglas T), al igual que las reglas de estructura sintagmàtica, son reglas de reescritura, pero difieren de éstas en varios aspectos. Raramente las reglas T tienen un sòlo elemento a la izquierda y no proporcionan anàlisis de constituyentes [HAD75]. Las operaciones que realizan las reglas T son de cuatro tipos básicos: adición (o unión), en la que se agrega un elemento a la oración; remoción, en la que se suprime un elemento; permutación, en la que se altera el orden de los elementos; y sustitución, en la que un elemento sustituye a otro.

Las tres partes básicas de una regla T son [HAD75]:

a) La Descripción Estructural (DE), que especifica la estructura profunda de las oraciones a las que se aplica la regla.
b) El Cambio Estructural (CE), que señala què cambios hay que realizar.
c) La Condición, que identifica cualquier situación especial que pueda haber presente si se va a aplicar la regla.

Si se tiene por ejemplo la siguiente regla transformativa:

PASIVA:

DE: SN X V SN pasiva
1 2 3 4 5

Condición: ninguna

CE: 1 2 3 4 5 ---) 4 2 ser -d- 3 por 1

y la oración:

Pedro pateó la pelota

y le aplicamos la transformación de 'pasiva':

PASIVA:

	Pedro	pasado	patear	1- pelota	pasiva
DE:	SN	X	V	SN	pasiva
	1	2	3	4	5

Condición: ninguna

	1- pelota	pasado	ser -d-	patear	por	Pedro
CE:	4	2	ser -d-	3	por	1

obtenemos:

La pelota fué pateada por Pedro

II.5 SEMANTICA

II.5.1 CONCEPTOS

La semántica, a diferencia de la sintaxis que se ocupa de analizar la estructrura de la oración y su interrelación con otras oraciones, se encarga de estudiar el significado o sentido de las palabras de manera aislada o dentro de un contexto.

Es importante tener presente que la teoria semántica es la menos desarrollada de las distintas áreas del análisis lingüistico y la naturaleza de las propiedades formales de las reglas para la interpretación de los significados ce las palabras es la que con menos claridad se ha comprendido.

Existen numerosas oraciones que son consideradas como gramaticalmente válidas, pero son carentes de significado o validez semántica, como por ejemplo:

La flor corre alegremente mientras silba. (*)

La aceptación o rechazo de oraciones por parte de una persona se relaciona con la habilidad de parafrasear, de detectar ambigüedades, etc., asi como también la habilidad de resolver tales ambigüedades haciendo referencia al contexto y que tiene lugar cuando se contradice lo que sabemos hacerca del "mundo real".

En una oración como:

La edificación está a punto de caer.

Se pueden detectar elementos léxicos individuales que tienen contenido, tales como estar y caer, y se le conoce como significado léxico. A los elementos gramaticales tales como la terminación '-ción' y las particulas 'a', 'de', asi como las interrogativas, imperativas, etc. son denominadas Significado Gramatical [AIT74].

Aitchison ha sugerido que "...el léxico ligado a la gramática transformacional debe consistir en elementos formados por un conjunto de elementos semánticos. La implicación es que tales elementos semánticos son universales y la razón de que los lenguajes varien de vocabulario es porque los elementos se juntan en diferentes combinaciones".

Los elementos léxicos de un lenguaje forman un patrón coherente. No se organizan tan rigidamente como las reglas de la gramática, pero los patrones son suficientes para

permitir que cada elemento tenga una relación definible con todos los demàs.

De acuerdo a Aitchison, el hombre no acumula en su cerebro un inventario de todas las posibles oraciones, sino una serie de reglas semànticas que permiten darle significado a las oraciones. Estas reglas parecen ser en parte, reglas de compatibilidad aplicables tanto entre elementos lèxicos, como entre elementos lèxicos y gramaticales. Por ejemplo, el tiempo pasado es compatible con la palabra 'ayer', pero incompatible con la palabra 'mañana'. De hecho se ha sugerido que cada elemento lèxico tiene instrucciones inherentes a su uso, y dichas instrucciones pueden ser tanto semànticas como gramaticales.

Las instrucciones gramaticales pueden especificar ciertas condiciones gramaticales que se aplican donde quiera que un elemento es usado. Por ejemplo, un verbo (patear) debe ir acompañado de un sujeto (Pedro) y un objeto (pelota).

sujeto (--- verbo ---) objeto

Pedro (--- patea ---) la pelota

Las instrucciones semànticas pueden indicar elementos que deben estar presentes en los elementos asociados mencionados en las instrucciones gramaticales [QUI80]. Por ejemplo, determinar que para el verbo leer, el sujeto debe ser animado, y el objeto debe ser un texto. Sin embargo, el estudio del significado de una palabra no debe tratarse sòlo como un sistema de relaciones internas, sino también en relación con el mundo exterior.

Desde el punto de vista transformacionalista se puede decir que el componente semàntico de una gramàtica transformativa se ocupa de asignar significados a las oraciones generadas por el componente sintàctico. El componente semàntico opera sobre estructuras profundas. Esto se debe a que los elementos de significación han sido asignados mediante las reglas de Estructura Sintagmàtica (ES). No hay que olvidar que lo único que hacen las reglas de transformación (reglas T) es señalar la forma en que se relacionan la estructura profunda y la superficial, es decir, las reglas T no aportan significado a la oración ([AIT74], [CHO76], [HAD75]). El punto fundamental es que lo que va incrustado dentro de la estructura profunda, cuando un artículo lèxico individual se ha seleccionado para una posición, no son las letras normalmente asociadas a una palabra, sino realmente todos los rasgos sintàcticos, semànticos y fonològicos los que caracterizan esa palabra.

Según Katz [KAT79], la situación actual es primordialmente el resultado del cambio de concepción producido por la gramática transformacional. Con la lingüística concebida como estudio del sistema de reglas interiorizadas que constituyen la competencia lingüística del hablante nativo, el significado tiene un lugar en la estructura gramatical.

II.5.2 SIGNIFICADO GRAMATICAL

Todos los elementos gramaticales de la estructura profunda contribuyen algo al significado de la oración y estos significados los utilizamos y entendemos los hablantes del español, por ejemplo, más o menos en el mismo sentido.

Una solución al problema de reflejar el significado gramatical en nuestra gramática es la de suponer que cada elemento gramatical introducido en las reglas de estructura sintagmática (ES) va relacionado en el diccionario con un significado (o una serie de significados, si es necesario). El significado gramatical debe ser válido de modo que el componente semántico pueda asignar significados a las oraciones.

La operación real de la asignación de significados mediante los componentes semánticos todavía no se ha descrito por completo. Katz y Fodor [KAT79] describen el proceso como realizado por una serie de reglas de proyección, que actúan sobre las estructuras profundas.

Se debe considerar al componente semántico de una gramática transformativa como un procedimiento, claramente formalizado en una serie de reglas de proyección para la asignación de una serie de significados aceptables a cualquier oración generada por el componente semántico de la gramática.

En el caso especial de la capacidad para comunicarse en una lengua natural, de acuerdo con Chomsky, tenemos que distinguir entre los aspectos que conciernen a la competencia lingüística del hablante, es decir, aquello que un hablante ideal sabe acerca de la estructura gramatical de su lengua y que le permite comunicarse haciendo uso de ella, y su actuación lingüística, es decir, el modo en que utiliza su competencia lingüística para comunicarse con otros hablantes en situaciones concretas. De ahí que la teoría

general de la comunicación lingüística se componga de dos teorías separadas pero relacionadas: una de ellas es la teoría de la competencia y la otra la teoría de la actuación.

La teoría de la competencia lingüística, enuncia el sistema de reglas que representa formalmente las estructuras lingüísticas ideales que subyacen en las locuciones del habla natural.

La teoría de la actuación lingüística trata de explicar los principios que usan los hablantes en la producción y entendimiento concretos del habla natural.

Leech establece en su obra "Semantica", 7 tipos de significado, el primero de ellos es el significado conceptual y sostiene básicamente lo que Katz describe como significado gramatical. Los otros tipos son: [LEE85].

- Significado Connotativo:
  Es el valor comunicativo que tiene una expresión atendiendo solo a lo que ella se refiere, haciendo a un lado su contenido puramente conceptual.
- Significado Social:
  Es lo que un elemento de la lengua expresa acerca de las circunstancias sociales de su empleo.
- Significado Afectivo:
  Es la consideración de cómo el lenguaje refleja las opiniones y las creencias personales del hablante, incluyendo su actitud para con el oyente o su postura ante algo de lo que se está hablando, por lo tanto se transmite a menudo explícitamente a través del contenido conceptual o connotativo de las palabras.
- Significado Reflejo:
  Es aquel que se da en los casos de significado conceptual múltiple, es decir, cuando un sentido de una palabra forma parte de nuestra respuesta a otro sentido.
- El Significado Conlocativo:
  Consiste en las asociaciones que una palabra adquiere al tener en cuenta los significados de las palabras que suelen aparecer en su entorno.
- Significado Temático:
  Es lo que se comunica por la forma en que el mensaje está organizado respecto del orden y del énfasis.

II.5.3 TEORIA SEMANTICA

Según Leech, una teoria semántica debe considerar lo siguiente[LEE85]:

a) Predecir los enunciados básicos de sinonimia, entraNe, tautologia, contradicción, etc. para una lengua.
b) Relacionar el significado con la sintaxis dentro de una teoria global sobre el funcionamiento de la lengua.
c) Relacionar el significado con la pragmática.
d) Tiene que ser parte de una teoria más general que defina la naturaleza de las descripciones semánticas, y que constituyen por tanto una especificación de los universales lingüisticos.

Según Katz, la teoria semántica debe contener [KAT79]:

a) Un esquema de representación semántica consistente en un vocabulario teorético del cual puedan extraerse los constructos semánticos requeridos para la formulación de interpretaciones semánticas particulares.
b) Un modelo del componente semántico de una gramática que debe describir la manera en que las interpretaciones semánticas se proyectan en ahormantes subyacentes. Debe explicar la competencia semántica subyacente en la capacidad del hablante para entender el significado de nuevas oraciones elegidas arbitrariamente entre la gama infinita de oraciones.
c) Una especificación de la forma del diccionario y una especificación de la forma de las reglas que proyectan representaciones semánticas para constituyentes sintácticos complejos partiendo de las representaciones del diccionario con respecto a los sentidos de sus partes sintácticas minimas.
d) Una especificación de la forma del componente semántico, de la relación entre el diccionario y las reglas de proyección, y de la manera en que dichas reglas se aplican para la asignación de representaciones semánticas.

El término ahormante semántico se refiere a la representación semántica de uno u otro de los conceptos que aparecen como partes de los sentidos. Los ahormantes representan a los constituyentes sintácticos de las oraciones.

Los conceptos de la gramática tradicional están siendo reemplazados por términos técnicos de vocabulario más convenientes para la explicación y descripción formales

dentro de la gramàtica generativa, esto se debe a que los fenòmenos gramaticales se dividen en un sentido amplio en tres clases (fonològicos, sintàcticos y semànticos), mientras que los conceptos ordinarios son una mezcla de màs de una clase [KAT79].

Tal redefinición, establece Katz, pretende obtener un vocabulario tècnico de constructos sintàcticos que refleje los modos en que los fenòmenos sintàcticos difieren de los fenòmenos semànticos y de los fenòmenos fonològicos; asi como también un vocabulario tècnico de contructos semànticos que refleje la diferencia de los fenòmenos semànticos con los sintàcticos y fonològicos.

Los fenòmenos sintàcticos tienen que ver con la disposición de objetos fonèticos u ortogràficos en concatenaciones que constituyen secuencias lineales bien-formadas. Los fenòmenos semànticos tienen que ver con el significado o contenido conceptual de tales secuencias y sus partes. Y los fonològicos tienen que ver con su pronunciación.

Si los fenòmenos semànticos tienen que ver con el contenido conceptual, entonces esos fenòmenos consisten en cosas tales como la sinonimia, la ambigüedad, la significación, la analiticidad, la redundancia, la contradicción, la antonimia, y el entrañe (ver glosario).

Principio para determinar si un concepto pertenece a la teoria sintàctica o semàntica [KAT79]:

a) Si la representación de la estructura sintàctica debe contener cierto concepto gramatical @ con objeto de que tal o cual fenòmeno sintàctico sea explicado del modo màs simple y revelador, pero la representación de la estructura semàntica no puede contener @ porque su aparición impediria que tal o cual fenòmeno semàntico fuese explicado del modo màs simple y revelador, entonces @ pertenece exclusivamente al vocabulario de la teoria sintàctica.
b) Si la representación de la estructura semàntica debe contener cierto concepto gramatical @ con objeto de que tal o cual fenòmeno semàntico sea explicado del modo màs simple y revelador, pero la representación de la estructura sintàctica no puede contener @ porque su aparición impediria que tal o cual fenòmeno sintàctico fuese explicado de modo màs simple y revelador, entonces @ pertenece exclusivamente al vocabulario de la teoria semàntica.
c) Si la representación de la estructura sintàctica y de la semàntica debe contener cierto concepto gramatical @ con objeto de explicar sus respectivos

fenómenos del modo más simple y revelador, entonces @ pertenece tanto a la teoría sintáctica como semántica.

Al adoptar este principio adoptamos el punto de vista de que lo que distingue a los componentes sintácticos y semánticos es fundamentalmente su vocabulario técnico respectivo y no, por ejemplo, los tipos de reglas que se usan en esos componentes o las condiciones que restringen la operación de las reglas.

Acerca de esta cuestión, partiendo de los conceptos sintácticos y, por ejemplo, de los géneros masculino, femenino y neutro. Los constructos semánticos de virilidad, feminidad y asexualidad, que en muchas lenguas tienen una distribución similar a la de los rasgos sintácticos, representan su papel en la determinación de propiedades y relaciones semánticas como la de analiticidad, sinonimia y entrañe.

La tradicional clasificación del género masculino, femenino y neutro puede expresarse, según sugiere Chomsky, por medio de los rasgos '1género', '2género' y '3género' respectivamente. Si el artículo léxico para un morfema contiene uno de estos tres rasgos, se especifica por el género; por ejemplo, el artículo léxico para el morfema "muchacho" adoptará la forma [KAT79]:

(muchacho; [+N, +det_, +cont, +anim, +hum, 1género])

El artículo léxico para el morfema "muchacha" adoptará la forma:

(muchacha; [+N, +det_, +cont, +anim, +hum, 2género])

Y el artículo léxico para el morfema "piedra" adoptará la forma:

(piedra; [+N, +det_, +cont, -anim, 3género])

Cualesquiera que sean los medios que elijamos para especificar el género, la gramática debe dar cuenta también del hecho de que las oraciones i-iii son gramaticalmente bien-formadas:

i) The baby drank its bottle.
ii) The baby drank her bottle.
iii) The baby drank his bottle.

Estos hechos ponen de manifiesto que no puede haber un artículo léxico único para 'baby' en el que solamente aparezca uno de los rasgos del género. Un modo natural de especificar el género equivale a dotar al léxico de tres

artículos para 'baby', cada uno de los cuales contiene uno diferente de los tres rasgos del género o quiza un artículo que sea una abreviatura conveniente de los tres.

Por otra parte, existen ciertas relaciones y propiedades lógicas, entre las cuales están las propiedades de reflexividad, irreflexividad y no-reflexividad [KAT79]. Se puede postular que su representación en la gramática puede manejarse mediante el par de rasgos [+refl] y [-refl], correspondiendo el primero a los reflexivos, el segundo a los irreflexivos, y a los no-reflexivos ninguno de ellos. Este rasgo servirá para diferenciar la gramaticalidad de :

i) Pedro se incriminó a sí mismo

y la ingramaticalidad de:

ii) Pedro se suicidó a sí mismo (*)

Si estos rasgos sintácticos permiten distinguir entre los objetos reflexivos, irreflexivos y no-reflexivos, entonces [-refl] también puede construirse como representando la información semántica de que la actividad expresada por un verbo que recibe este rasgo es una actividad que el agente no puede realizar sobre sí mismo y [+refl] puede construirse como representando la información semántica de que la actividad expresada por un verbo que reciba este rasgo es una actividad que el agente puede realizar solamente sobre sí mismo [KAT79]. Por tanto estos rasgos sintácticos no solo sirven para marcar la aberración sintáctica, sino también para determinar las propiedades lógicas de la reflexividad, la irreflexividad y la no-reflexividad. Sin embargo, semejante posición no puede mantenerse sin predecir incorrectamente algunas propiedades semánticas y dejar otras inexplicadas.

Está claro que (ii) es gramaticalmente aberrante, de ahí que la aberración gramatical en estos casos tenga que ser explicada como semánticamente anómala mediante la introducción de restricciones de selección adecuadas. Pero, si tales restricciones se agregan a las lecciones de verbos como "suicidar" e "incriminar", deja de estar claro que los rasgos sintácticos [+refl] y [-refl] sean necesarios [KAT79]. La distinción [refl] parece no tener ahora papel en la representación semántica de las propiedades lógicas de reflexividad, irreflexividad y no-reflexividad.

Resulta claro que el análisis de significancia de una oración no se puede realizar analizando las palabras que componen la oración de manera aislada. Según Leech: "... la hipótesis más sencilla e ingenua a este respecto dice que el significado de una oración es la suma de los significados de las palabras y de los demás elementos que las componen, esto

es falso debido a que genera entre otras cosas asignificancias", [LEE85].

Leech propone dejar de acoplar el análisis semántico al molde de las unidades sintácticas y buscar en su lugar unidades y estructuras que funcionen a nivel semántico sin descuidar sus correlaciones (sintáctico-semántico), lo cual se logra tomando como punto de partida el análisis componencial. Entiéndase a éste como la descripción del significado en base a combinaciones de rasgos contrastantes. El análisis componencial se puede aplicar al significado de la palabra pero no al de la oración completa.

Esto no es una limitante fuerte debido a que es posible tomar por ejemplo sintagmas y sinonimias. Por lo tanto la unidad semántica en la que se aplica el análisis componencial es inferior a la de la oración, pero es también potencialmente superior al de la palabra; a esta unidad superior se le denomina predicación.

El análisis componencial de predicaciones permite manejar de manera más eficiente aspectos como entrañe, inconsistencia, argumentos nulos, restricciones selectivas, etc.

Para construir un modelo del significado oracional, al análisis de predicaciones aún le falta un elemento esencial, esto es, el elemento lógico que relaciona las predicaciones con las afirmaciones sobre el mundo real. Esto se debe a que la "lógica natural" que se intenta describir en la lingüística no es necesariamente lo mismo que la lógica formal ha desarrollado en la filosofía. Por ello se hace necesario encontrar una relación entre "palabras lógicas" y "palabras de contenido" mediante el uso de operadores lógicos y cuantificadores, por ejemplo: [LEE85]

	ELEMENTO LOGICO	ELEMENTO DE CONTENIDO
nunca	negación	+tiempo
en alguna parte	cuantificación	+lugar
quién	interrogación	+persona
venir	demostratividad	+movimiento

Tabla. II.4 -1-

La sintáxis, al igual que la semántica, está organizada a base de oposiciones contrastantes, pero en primer lugar, la sintaxis es mucho menos rica en cuanto a dimensiones de

contraposición que la semántica debido a que gran parte de la estructura contrastante de la semántica queda absorbida por la lexicalización, y en segundo lugar, mientras que las alternativas sintácticas son obligatorias, las semánticas son frecuentemente opcionales.

Una teoría lingüística debe especificar para cualquier lengua al menos tres cosas (excluyendo el nivel fonológico): [LEE85]

a) El conjunto de representaciones semánticas bien formadas.
b) El conjunto de representaciones sintácticas bien forma-das.
c) Las reglas (de expresión) para emparejar las representaciones semánticas con las sintácticas.

II.6 LEXICON

El diccionario de una lengua se ha caracterizado generalmente como un repertorio de todos lo hechos específicos de esa lengua, es decir, aquellos que no se pueden generalizar mediante reglas. Los hechos específicos incluyen también irregularidades, o casos que son excepciones a una regla determinada. Tales hechos específicos pueden enunciarse en relación a ciertos elementos (piezas léxicas) que corresponden por lo general a las unidades gramaticales que conocemos con el nombre de "palabras"; sin embargo en algunos casos, la pieza léxica abarca un fragmento de sintaxis mayor que una palabra, en cuyo caso la denominamos idiotismo, pero en cualquier caso la pieza léxica ha de tener su pronunciación y su significado (definición) perfectamente delimitados, y por tanto, el diccionario atañe a los tres niveles: el sintáctico, el semántico y el fonológico (éste último no se contempla en la tesis) [LEE85].

Las palabras tienen asociados pares léxicos mutuamente excluyentes, y se marcan como "+" ó "-". De esa forma, la relación del léxico para una palabra (por ejemplo: hombre) incluirá no solamente los rasgos del diccionario tradicional ([+N], [+masc]), sino también Rasgos Lexicológicos Precisos ([+común, +contable, +animado, +humano]) ([CHO76], [HAD75], [LEE85]).

Este tipo de clasificaciòn permite establecer la distinciòn entre, por ejemplo nombres, y especificar con mayor detalle què tipos de nombre pueden insertarse en la posiciòn correspondiente a 'N' en cualquier àrbol estructural.

	com	anim	cont	hum	..etc.
hombre, niño, etc...	+	+	+	+	
perro, caballo, etc...	+	+	+	-	
casa, pistola, etc...	+	-	+	-	
vino, trigo, etc...	+	-	-	-	
Jose, Marco, etc...	-	+	-	+	
Rocinante, Fido, etc...	-	+	-	-	
Francia, India, etc...	-	-	-	-	

Tabla. II.4 -2-

A esas fòrmulas se les llama Definiciones Componenciales y a las dimensiones mismas del significado se les suele llamar Oposiciones Semànticas. Es recomendable omitir la definiciòn y uso de dimensiones neutrales a fin de evitar un trabajo de enumeraciòn exhaustivo no fructifero. A este mètodo de anàlisis se le denomina Anàlisis Componencial [LEE85].

Para evitar la generaciòn de estructuras profundas incorrectas (marcadas con un asterisco "*"), que son igualmente probables que una estructura profunda correcta como por ejemplo:

Pedro pasado comer 1- manzana

1- manzana pasado comer Pedro (*)

Se especifican, por ejemplo para cada 'N' de la oraciòn, comenzando con el sujeto, los rasgos que son apropiados para su posiciòn en la oraciòn, y son aplicados mediante una serie de reglas que obligan que se haga una elecciòn entre los rasgos sintàcticos posibles de acuerdo a su posiciòn.

La regla lèxica general que abarca estas operaciones es [HAD75]:

> "Cualquier palabra del lèxico puede ser sustituida por la serie de rasgos en la terminaciòn de una rama de un àrbol estructural, siempre y cuando no exista contradicciòn de especificaciones de rasgo

entre el artículo léxico y la serie terminal de rasgos".

En el español, por ejemplo, la subcategorización de 'N' se realiza antes que las subcategorizaciones de las demás categorías, porque la categoría 'nombre' se considera elemento básico de la oración en español.

Dadas las reglas de estructura sintagmática que se ocupan de la selección léxica, se añade al conjunto de rasgos inherentes dentro del árbol, los rasgos contextuales que sean necesarios para especificar, en ese conjunto de rasgos, su función gramatical en la oración. Esto se logra añadiendo a cada conjunto de rasgos léxicos del árbol, rasgos que establecen:

a) Las categorías por encima de él en el árbol.
b) Sus categorías hermanas.

Si añadimos por ejemplo [+O, +SN, +_] a la matriz de rasgo del sujeto, en nuestra oración ejemplo "Pedro pasado comer 1- manzana" hemos añadido la información de que el nombre en cuestión es el sujeto de una oración y que no va acompañado de "det". Los rasgos añadidos de esta forma a la matriz de "manzana" serán [+O, +SV, +verbal, +SN, +det_] y los de la matriz de "comer" serán [+O, +SV, +verbal, +V, +_SN]. La línea de rasgos como [+_SN], [+det_], y [+_] señala la posición de la matriz que estamos considerando. Estos rasgos se leerían respectivamente, "aparece seguido de SN", "aparece precedido de un det" y "aparece solo".

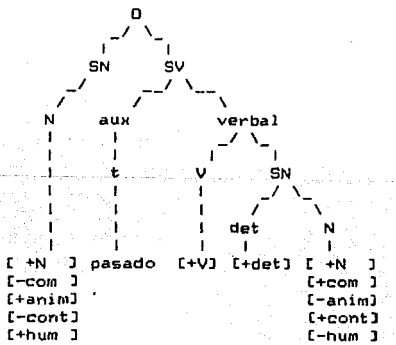


Fig. II.4 -4-

Obsèrvese que para el simbolo 'N', se tiene:

i)	N --->	[+N], S.C.
ii)	[+N] --->	[+-com, +-anim]
iii)	[+anim] --->	[+-hum]
iv)	[+com] --->	[+-cont]

La primera regla (i) sustituye el simbolo de categoria 'N' por el rasgo [+N], y mediante las reglas de Simbolo Complejo (S.C.) establece el hecho de que ya no describimos la estructura constituyente, sino màs bien la estructura de las palabras [HAD75]. Una de las funciones del simbolo complejo (S.C.) es indicar que las reglas que siguen, cuando son aplicadas, sirven no para sustituir un elemento por otro, sino para aÑadir lo que sobre èl ya se ha decidido [CHO76]. Esto es, para el ejemplo, la regla (ii) requiere que al rasgo [+N] le sea aÑadido por una parte [+com] ò [-com], y por otra [+anim] ò [-anim].

A cada posiciòn del nombre de un àrbol se le asigna por las reglas i-iv una cierta combinaciòn de rasgos inherentes. Desde el momento en que esos mismos rasgos son utilizados para caracterizar palabras en el lèxico, ya es posible sustituir palabras en un lugar apropiado del àrbol.

Tanto la derivaciòn morfològica como la transferencia semàntica se pueden entender como casos distintos del mismo proceso bàsico, consistente en ampliar el repertorio de las rùbricas lèxicas [LEE85].

Observaciones adicionales sobre las reglas lèxicas: [LEE85]

a) La capacidad de estas reglas para generar nuevas rùbricas lèxicas es extraordinariamente poderosa.
b) El presentar conjuntamente la derivaciòn morfològica, la conversiòn y la transferencia semàntica como casos del mismo fenòmeno general està motivado por otras caracteristicas comùnes, aparte de por su productividad parcial.
   - diversidad.
   - no definitiva cerrazòn semàntica.
   - recursividad.
   - bidireccionalidad.
   - tendencia a la deformaciòn de las rùbricas lèxicas.

II.6 SUMARIO

En este capitulo se mostraron los aspectos generales de la teoria gramatical, asi como la sintaxis y la semàntica tomadas como base de estudio para el desarrollo de esta tesis. Se describe superficialmente la Gramàtica Generativa Transformacional de Chomsky y la Teoria Semàntica de Katz por ser formalismos gramaticales base para el desarrollo de las Gramàticas de Clausula Definitiva (DCG) y de las Gramàticas Estructuradas de Frase Generalizada (GPSG) usadas en esta tesis.

Esto deja claro que no se puede contar, al menos por ahora, con una manera bien definida que permita determinar el limite exacto entre el anàlisis sintàctico y semàntico de la oraciòn. Parece existir siempre una zona de traslape entre ellos que hace sumamente dificil su estudio.

Cabe cuestionarse por què el estudio y anàlisis del lenguaje natural es tan dificil si el trabajo que implica es anàlogo al que se realiza con los lenguajes artificiales tales como los lenguajes de computadora. La razòn es simplemente que los lenguajes artificiales han sido concebidos dentro de un contexto perfectamente predeterminado y se han desarrollado a partir de una gramàtica dada. En el caso de los lenguajes naturales no sucede asi, pues la gramàtica surgiò despuès de la lengua, lo cual hace dificil una adaptaciòn total y directa, tomando en cuenta ademàs que los lenguajes naturales son ilimitados en cuanto al dominio de la informaciòn que puede ser comunicada a travès de sus oraciones.

CAPITULO III

GRAMATICA DE CLAUSULA DEFINIDA

> Alice was too much puzzled to say anything, so after a minute Humpty Dumpty began again. "They've a temper, some of them, particularly verbs, they're the proudest - adjectives you can do anything with, but not verbs - however, I can manage the whole lot!".
>
> Lewis Carroll.

III.1 INTRODUCCION

Las Gramàticas de Clàusula Definida (GCD, en inglès Definite Clause Grammars DCG), son un formalismo sencillo y poderoso para la descripciòn de lenguajes formales y naturales. Este formalismo fuè descrito originalmente por Colmerauer y Kowalski en 1975, como un mètodo para expresar gramàticas en forma lògica, y posteriormente por Pereira y Warren [PER80] en 1980.

El formalismo de las DCG es una extensiòn natural de las Gramàticas Libres de Contexto (ver cap. I.5), y provee ademàs de la descripciòn del lenguaje, un medio favorable para el anàlisis de las cadenas (strings) que se pueden generar con ese lenguaje. Ademàs resulta ser pràcticamente un programa ejecutable en el lenguaje de programaciòn Prolog (PROgramming in LOGic).

Dado que las DCGs son pràcticamente programas ejecutables del lenguaje de programaciòn Prolog, podemos obtener de ellas un còdigo de màquina eficiente, por lo que se pueden implementar procesos de lenguaje natural directamente como DCGs.

III.2 CONCEPTOS

Como se vió en el capitulo I.5, las gramàticas se pueden dividir segùn Chomsky ([CHO76] [GRI71]) en 4 tipos o clases; de ellas el tipo de gramàtica fundamental en la teoria de los lenguajes formales es la Gramàtica Libre de Contexto (Context Free Grammar CFG). El nombre se debe a que el elemento a reescribir puede ser reemplazado independientemente del contexto en el cual aparezca, como una secuencia de simbolos terminales y no-terminales.

Colmerauer y Kowalski formularon la idea de traducir el formalismo de las Gramàticas Libres de Contexto (CFG) en uno de propósito general denominado "lògica de predicados de primer orden". A partir de ahí, idearon un mètodo para expresar reglas libres de contexto como reglas lògicas restringidas y les denominaron "clàusulas definidas", "clàusulas de Horn" o "clàusulas regulares". La ventaja que presenta esta nueva formulaciòn es que el problema de anàlisis de una cadena (string) de un lenguaje se transforma en un problema de comprobaciòn de que un cierto teorema parte de los axiomas de clàusula definida que describen el lenguaje [PER80].

La Programaciòn Lògica se inició a principios de los años 70 como una herramienta para resolver problemas de demostraciòn de teoremas e inteligencia artificial. En 1972 Kowalski y Colmerauer crearon la idea fundamental de que la lògica se puede utilizar como un lenguaje de programaciòn, fuè entonces cuando Roussel implementò el primer interprete PROLOG en la Universitè de Marseille en 1972. PROLOG se basa en la lògica de predicados de primer orden (Clàusulas de Horn) [LLO84].

En su articulo "Definite Clause Grammars for Language Analysis" [PER80], Pereira y Warren establecen que "... si una Gramàtica Libre de Contexto (CFG) es expresada en Clàusulas Definidas de acuerdo con el mètodo de Colmerauer y Kowalski, y ejecutada como un programa Prolog, el programa resulta ser un analizador "top-down" (de arriba a abajo) para el lenguaje que la CFG describe". También establecen que "... la tècnica para traducir las CFGs en Clàusulas Definidas tiene una generalizaciòn simple, resultando en un formalismo màs poderoso que las CFGs, pero igualmente tratables en Prolog". A èste formalismo le denominaron Gramàtica de Clàusula Definida (Definite Clause Grammar DCG).

Algunas de las razones por las que las CFGs son importantes en la teoria de los lenguajes son [PER80]:

- Las formas posibles para las oraciones de un lenguaje están descritas de forma modular y clara.
- Es posible representar el anidamiento recursivo de frases característico de prácticamente todos los lenguajes.
- Se tiene un cuerpo establecido de resultados en las CFGs que son muy útiles en el diseño de algoritmos de análisis gramatical.

Recordemos que Chomsky [CHO76] demuestra que las CFGs no son del todo apropiadas para la descripción de los lenguajes naturales. Sin embargo, el formalismo de las DCGs supera en mucho tal desventaja extendiendo las CFGs en tres formas [PER80]:

1.- Las DCGs suministran dependencia de contexto a la gramática. De ésta forma únicamente aquellas formas de una frase que dependen del contexto de la oración son permitidas en ciertas partes de ella, lo cual resulta ser de gran ayuda para la solución de problemas tales como la ambigüedad.
2.- Las DCGs permiten la construcción de árboles de estructura en el transcurso del análisis gramatical, lo cual permite proveer una representación del significado de la oración.
3.- Las DCGs permiten la inclusión de condiciones extra en las reglas gramaticales. Ello permite modificar el curso del proceso de análisis gramatical dependiendo de los resultados intermedios obtenidos.

La extensión que de las CFGs hacen las DCGs es básicamente mediante la adición de símbolos no-terminales con argumentos, por lo que se tiene un medio propicio para el análisis gramatical de las oraciones de un lenguaje.

III.3 LA GRAMATICA DE CLAUSULA DEFINIDA Y LAS GRAMATICAS LIBRES DE CONTEXTO

Los conceptos que contempla este tema fueron tomados principalmente del artículo de Pereira y Warren: "Definite Clause Grammars for Language Analisis - A Survey of the Formalism and a Comparison with Augmented Transition

Networks" [PER80] y de la disertación de M. en C. de Jimenez "Low-Quality Fully Automatic Machine Translation by Syntax Only" [JIM86].

En una Gramática Libre de Contexto (GLC, o Context Free Grammar CFG) se puede observar que cada regla tiene la forma:

nt --) body

donde: nt es un simbolo no-terminal.
body es una secuencia de uno o màs elementos (items) separados por comas.

nota: la terminologia està tomada tal cual de los textos originales.

"Cada elemento (item) es, o un simbolo no-terminal o una secuencia de simbolos terminales. El significado de la regla es que 'body' es una forma posible para una frase de tipo 'nt'. Un simbolo no-terminal se escribe como un àtomo en Prolog, mientras que una secuencia de simbolos terminales se escribe como una lista en Prolog, donde un simbolo terminal puede ser cualquier tèrmino en Prolog".

A fin de poder observar como implementar y expresar en DCGs la definición anterior, considèrese la siguiente gramàtica (libre de contexto) simple:

```
sentence --) noun_phrase, verb_phrase
noun_phrase --) proper_noun
noun_phrase --) determiner, common_noun
verb_phrase --) transitive_verb, noun_phrase

proper_noun --) [pedro]
determiner --) [la]
common_noun --) [pelota]
transitive_verb --) [patea]
```

La gramàtica anterior permite generar la oración:

pedro patea la pelota.

III.3.1 CONSTRUCCION DE ESTRUCTURAS

Cuando se lleva a cabo la expansión de simbolos no-terminales, las estructuras se construyen progresivamente durante el proceso de unificación de Prolog ([BOI87], [CLO84], [JIM86], [PER80], [PER81]).

Para obtener la traducción de la gramática anterior a Cláusulas Definidas, se le asocia a cada símbolo no-terminal un predicado que contenga como parámetros los puntos inicial y final de la frase; las reglas gramaticales anteriores quedarán como sigue:

```
        sentence(S0,S) --) noun_phrase(S0,S1),
                           verb_phrase(S1,S)
    noun_phrase(S0,S) --) proper_noun(S0,S)
    noun_phrase(S0,S) --) determiner(S0,S1),
                           common_noun(S1,S)
    verb_phrase(S0,S) --) transitive_verb(S0,S1),
                           noun_phrase(S1,S)

        proper_noun --) [pedro]
         determiner --) [la]
        common_noun --) [pelota]
    transitive_verb --) [patea]
```

Las reglas anteriores se pueden leer como, por ejemplo para la primera regla (sentence): "una 'sentence' (oración) se extiende de S0 a S si existe una 'noun_phrase' (frase nominal) de S0 a S1 y una 'verb_phrase' (frase verbal) de S1 a S"; o como en el caso de la segunda regla (noun_phrase): "una 'noun_phrase' (frase nominal) se extiende de S0 a S si existe un 'proper_noun' (nombre propio) de S0 a S", etc.

La representación de los simbolos terminales en las reglas se hace de forma similar pero con un predicado de 3 parámetros que tiene la siguiente forma:

```
connects(S0,T,S)
```

donde: connects es el nombre del predicado.
S0, S son los puntos inicial y final.
T es el simbolo terminal.

Por lo tanto, las reglas correspondientes a los simbolos terminales de la gramática quedan como:

```
    proper_noun(S0,S) --) connects(S0,pedro,S)
     determiner(S0,S) --) connects(S0,la,S)
    common_noun(S0,S) --) connects(S0,pelota,S)
transitive_verb(S0,S) --) connects(S0,patea,S)
```

Se puede observar que la representación de una CFG por medio de cláusulas es independiente de datos (data-independent) desde el punto de vista de que la representación de la cadena (string) actual a ser analizada gramaticalmente no es "conocida" por las cláusulas. de hecho solamente el predicado 'connects' y la meta a ser probada lo toman en cuenta. Si se desea entonces representar el último conjunto de reglas de forma general e independiente de datos, se redefinirá entonces la cláusula 'connects' de la siguiente forma:

```
connects([W|S],W,S)
```

Esta cláusula puede leerse como: "La posición de la cadena etiquetada por la lista con la cabeza W y la cola S está conectada por el símbolo W a la posición de la cadena etiquetada con S". Por lo que el conjunto de reglas de los terminales serán entonces:

```
proper_noun(S0,S) --> connects(S0,W,S),
                      prop_noun(W)
determiner(S0,S)  --> connects(S0,W,S),
                      det(W)
common_noun(S0,S) --> connects(S0,W,S),
                      com_noun(W)
transitive_verb(S0,S) --> connects(S0,W,S),
                          trans_verb(W)
```

De esta forma se logra la independencia de datos, por lo que las mismas reglas sirven para un lèxico más amplio, como por ejemplo:

```
prop_noun(pedro)
prop_noun(luis)
prop_noun(lucia)
det(la)
det(los)
com_noun(pelota)
com_noun(vidrios)
com_noun(platos)
trans_verb(patea)
trans_verb(rompe)
trans_verb(lava)
```

Con el conjunto de reglas dadas y el lèxico anterior se pueden formar oraciones tales como:

```
pedro patea la pelota
lucia lava los platos
luis rompe los vidrios
luis lava la pelota
pedro rompe los platos
lucia patea los vidrios
```

luis patea la platos (*)
lucia rompe los pelota (*)
etc.

Dado que las CFGs pueden ser expresadas como clàusulas definidas, se verà ahora la conceptualizaciòn de las DCGs de acuerdo con la definiciòn dada por Pereira y Warren en [PER80].

Las DCGs extienden la notaciòn de las CFGs de dos formas:

i) Se permite que los simbolos no-terminales sean tèrminos compuestos en adiciòn a los àtomos simples permitidos en las CFGs.

ii) En el lado derecho de una regla se pueden tener llamadas a procedimientos para expresar ciertas condiciones que deben ser satisfechas por la regla a validar. Estas condiciones extras se encierran entre llaves ('{' y '}').

De esta forma, si se tiene una regla de CFG como la siguiente:

```
noun_phrase(N) --> [W],{rootform(W,N), is_noun(N)}
```

y cuya interpretaciòn es: "Una frase identificada como el nombre N puede consistir en una palabra simple W, donde N es la forma raiz de W y N es un nombre".

Los simbolos terminales, no-terminales y llamadas a procedimientos que se encuentran en la parte derecha de una regla son referenciados como 'metas' (goals).

La representaciòn de la regla anterior en DCG serà de la siguiente forma:

```
noun_phrase(N,S0,S):- connects(S0,W,S),
                      rootform(W,N),
                      is_noun(N).
```

Esta forma es a su vez, la forma final de las reglas de DCG codificadas como clàusulas Prolog. Nòtese que los simbolos terminales de aridad N se traducen en predicados de aridad N+2 con el mismo nombre, donde los primeros N argumentos son los especificados en el no-terminal y los 2 argumentos extra son los utilizados en la traducciòn de un no-terminal libre-de-contexto (context-free non-terminal); las llamadas a procedimientos del lado derecho se traducen directamente, sin cambios.

La expansiòn de reglas gramaticales con argumentos extra como paràmetros incrementa el poderio del formalismo

de las DCGs por medio de 3 mecanismos importantes necesarios para el análisis de lenguajes:

1) La posibilidad de construir estructuras del tipo árbol de análisis gramatical (parse tree) simultáneamente con el proceso de análisis gramatical mismo.
2) La posibilidad de adicionar condiciones extra a los elementos de una frase sintáctica.
3) La posibilidad de tratar la dependencia de contexto para evitar problemas de ambigüedad.

A fin de obtener una gramática más eficiente, las reglas de los símbolos no-terminales adoptan la siguiente forma:

```
sentence([s,NP,VP],S0,S) --> noun_phrase(NP,S0,S1),
                             verb_phrase(VP,S1,S)
noun_phrase([np,PN],S0,S) --> proper_noun(PN,S0,S)
noun_phrase([np,DET,CN],S0,S) --> determiner(DET,S0,S1),
                                  common_noun(CN,S1,S)
verb_phrase([vp,TV,NP],S0,S) --> transitive_verb(TV,S0,S1),
                                 noun_phrase(NP,S1,S)
```

Esta forma permite manejar con mayor claridad y eficiencia los componentes a ser analizados por cada regla, además de formar un árbol de análisis gramatical facilmente interpretable.

III.3.2 REPRESENTACION DE CONDICIONES ADICIONALES

Las condiciones adicionales o extra, son las restricciones impuestas en los constituyentes de una regla.

Si se tiene por ejemplo una regla que considera la concordancia en tiempo entre un verbo auxiliar y un verbo, tal regla será:

```
sentence([s,NP,[aux,TENSE,ROOT],VP],S0,S):-
        noun_phrase(NP,S0,S1),
        auxiliary([aux,TENSE,ROOT],S1,S2),
        aux_tense(ROOT,TENSE,AGR),
        verb_phrase(VP,AGR,S2,S).

aux_tense(ser,presente,gerundio).
```

Tales reglas gramaticales permiten que oraciones que contengan por ejemplo el verbo patear, sean compatibles o se ajusten en tiempo con la forma del verbo estar, por lo que una oración válida sería:

Pedro está pateando la pelota.

III.3.3 DEPENDENCIA DE CONTEXTO

Los argumentos de los símbolos no-terminales en una DCG se utilizan además de en la construcción de estructuras, en el manejo de información contextual, es decir en el proceso de análisis necesario para evitar o al menos disminuir las ambigüedades sintácticas en la oración mediante el análisis de las palabras de entorno. Esto se logra utilizando argumentos que contengan información acerca de elementos sintácticos tales como número, género, persona, etc.

Para poder utilizar la información contextual es necesario agregar a los símbolos no-terminales correspondientes, argumentos extra de tal forma que, por ejemplo una gramática que contenga:

```
noun_phrase([np,DET,CN,NUM]
            ,S0,S)              --) determiner(NUM,DET,S0,S1),
                                    common_noun(NUM,CN,S1,S)

  determiner(NUM,DET,S0,S)      --) connects(S0,NUM,DET,S),
                                    det(NUM,DET)

  common_noun(NUM,CN,S0,S)      --) connects(S0,NUM,CN,S),
                                    com_noun(NUM,CN)

                                    det(singular,la)
                                    det(plural,las)
                                    com_noun(singular,pelota)
                                    com_noun(plural,pelotas)
```

permitirá el ajuste correcto entre, por ejemplo el determinante "la" y el nombre común "pelota" o entre "las" y "pelotas", pero no entre "la" y "pelotas" o "las" y "pelota" debido a que no son gramaticales (*) por no ser compatibles en número.

la pelota	la pelotas (*)
las pelotas	las pelota (*)

III.4 LA GRAMATICA DE CLAUSULA DEFINIDA (DCG) Y EL LENGUAJE DE PROGRAMACION PROLOG

Se ha demostrado que la DCG no es más que un conjunto de clàusulas definidas, y su significado es independiente de cualquier mecanismo de ejecuciòn en que se aplique.

A partir de la semàntica de Prolog se puede observar que para analizar una oraciòn, se tienen que utilizar las reglas gramaticales en base a una politica de arriba-hacia-abajo (top-down), una a la vez y de izquierda a derecha. En caso de existir reglas alternativas, el mecanismo de "backtracking" las analizarà todas. La forma en que todo este proceso de anàlisis se lleve a cabo dependerà de la forma y el orden en que se haya escrito la gramàtica y el control que se tenga sobre el mecanismo de "backtracking".

Para poder entender còmo se ejecuta una Gramàtica de Clàusula Definida (DCG), se describirà a continuaciòn la forma en que una oraciòn es anàlizada gramaticalmente.

Considèrese la siguiente gramàtica:

```
english_dcg_parser(SENTENCE,TREE):-
      sentence(TREE,SENTENCE,[]).                    ... (a)

sentence([s,NP,VP],S0,S):-
      noun_phrase(NUM,PER,NP,S0,S1),
      verb_phrase(NUM,PER,TENSE,VP,S1,S).            ... (b)

noun_phrase(singular,3,[np,PROP_NOUN],S0,S):-
      proper_noun(PROP_NOUN,S0,S).                   ... (c)

noun_phrase(NUM,PER,[np,DET,COM_NOUN],S0,S):-
      determiner(NUM,PER,DET,S0,S1),
      common_noun(NUM,PER,COM_NOUN,S1,S).            ... (d)

verb_phrase(NUM,PER,TENSE,[vp,TV,NP],S0,S):-
      transitive_verb(NUM,PER,TENSE_AGR,TV,S0,S1),
      noun_phrase(NUM1,PER1,NP,S1,S).                ... (e)

determiner(NUM,PER,[det,W],S0,S):-
      connects(S0,W,S),
      lex(det,W,NUM,PER).                            ... (f)

common_noun(NUM,PER,[com,W],S0,S):-
      connects(S0,W,S),
      lex(com,W,NUM,PER,_,_).                        ... (g)
```

```
proper_noun([pn,W],S0,S):-
     connects(S0,W,S),
     lex(pn,W,_,_).                                    ... (h)

transitive_verb(NUM,PER,TENSE,[tv,ROOT],S0,S):-
     connects(S0,W,S),
     lex(tv,ROOT,W,NUM,PER,TENSE,_,_,_).               ... (i)

connects([W|S],W,S).                                   ... (j)
```

Siguiendo la notaciòn adoptada por Pereira y Warren [PER80], las metas (goals) en Prolog se escribiràn como:

simbolo de punto1 a punto2

donde: simbolo, es un no-terminal o una lista de terminales.
punto1 (P1)
y punto2 (P2), son posiciones en las oraciones.

Entonces, para la oraciòn:

```
Pedro patea la pelota
1     2     3  4     5
```

La regla inicial (a):

```
english_dcg_parser(SENTENCE,TREE):-
```

instancia SENTENCE como [pedro,patea,la,pelota], lo cual significa que se desea una estructura denominada TREE para dicha oraciòn desde 1 hasta 5. La regla (a) llamarà a la regla (b) mediante:

```
sentence(TREE,SENTENCE,[])
```

la cual mediante el ajuste (match) crearà las instanciaciones:

```
TREE = [s,NP,VP]
  S0 = [pedro,patea,la,pelota]
   S = []
```

y las metas (goals):

```
noun_phrase(NUM,PER,NP,S0,S1)    desde 1 hasta P1
verb_phrase(NUM,PER,TENSE,VP,S1,S)  desde P1 hasta 5
```

la primera de estas reglas se ajusta (match) con la regla (c) produciendo las instanciaciones siguientes:

```
NP = [np, PROP_NOUN]
NUM = singular
PER = 3
```

y la meta:

```
proper_noun(PROP_NOUN, S0, S).
```

Se ajusta a la regla (h) y se expande como:

```
connects(S0, W, S)   desde 1 hasta P1
lex(pn, W, _, _, _)
```

Ambas resultan exitosas debido a que la palabra de la posición 1 en la oración es "pedro", y se tiene esa palabra en el lèxico como un nombre propio; por lo tanto, la solución de la meta proper_noun es:

```
proper_noun(pedro, S0, S)   desde 1 hasta 2
```

y NP está instanciada como: [np, [pn, pedro]], y la meta noun_phrase se realiza como:

```
noun_phrase(singular, 3, [np, [pn, pedro]], S0, S1)
```

Prolog procede entonces con la meta verb_phrase desde P1 hasta 5. Ajustando la regla (e), se creará la instanciación siguiente:

```
VP = [vp, TV, NP]
```

y las metas:

```
transitive_verb(singular, 3, _, TV, S0, S1)  desde 2 hasta P2
noun_phrase(N1, P1, NP, S1, S)               desde P2 hasta 5
```

La primera de èstas metas se ajusta con la regla (i) instanciando:

```
TV = [tv, ROOT]
```

y las metas:

```
connects(S0, W, S)
lex(tv, ROOT, W, singular, 3, TENSE, _, _, _)
```

la cual será exitosa, regresando la forma raiz del verbo "patear". Por lo tanto:

```
TV = [tv, ROOT]
```

y transitive_verb se realizarán como:

transitive_verb(singular,3,_,[tv,patear],S0,S1)

entonces la segunda meta de verb_phrase se ajustarà a la regla (d) creando la instanciaciòn:

NP = [np,DET,COM_NOUN] desde 3 hasta 5

y las metas:

determiner(NUM1,PER1,DET,S0,S1) desde 3 hasta P3
common_noun(NUM1,PER1,COM_NOUN,S1,S) desde P3 hasta 5

Se puede observar que el noun_phrase desde P2 hasta 5 no necesita concordar en nùmero o persona con las frases previas de la oraciòn, por lo que se utilizan NUM1 y PER1 como otros valores para nùmero y persona. Estas dos ùltimas metas se ajustarán a las reglas (f) y (g), resultando exitosas y regresando:

NP = [np,[det,la],[com,pelota]]

por lo que la clàusula verb_phrase se completarà como:

VP = [vp,[tv,patear],
[np,[det,la],[com,pelota]]]

y finalmente, la estructura completa para la oraciòn serà:

TREE = [s,[np,singular,3,[pn,pedro]]
[vp,singular,3,present,[tv,patear],
[np,[det,la],[com,pelota]]]]

el cual es el àrbol de anàlisis gramatical (parse tree) de la oraciòn.

III.5 VENTAJAS DE LAS GRAMATICAS DE CLAUSULA DEFINIDA (DCG) SOBRE OTROS FORMALISMOS

"La 'variable lògica' es un elemento de los programas lògicos que hace de las DCGs un formalismo poderoso para la implementaciòn de analizadores pràcticos de lenguaje. Las estructuras se pueden escribir por partes, permitiendo que las partes no especificadas sean tratadas como variables.

Al efectuar el anàlisis de la estructura hace uso del proceso de unificaciòn para llenar las estructuras representadas por variables", [PER80].

nota: si se desea detallar màs sobre el proceso de unificaciòn se recomienda consultar [RIC85] cap. 5.4 o bien [LLO84] cap. 1.

Los textos siguientes son extractos del articulo de Pereira y Warren: "Definite Clause Grammars- A Survey of the Formalism and a Comparison with Augmented Transition Networks" [PER80].

Los sistemas pràcticos para el anàlisis de lenguaje natural son necesariamente largos y complejos. La DCG es un formalismo fàcilmente entendible como una màquina para el anàlisis y la descripciòn de un lenguaje en particular, èsta propiedad la comparten con las Gramàticas Libres de Contexto (CFG). Por lo tanto las DCGs son un formalismo lùcido.

El poderio y grado de generalizaciòn que alcanza un formalismo por medio de la consideraciòn de lo que puede o no ser expresado en dicho formalismo se puede juzgar desde los puntos de vista teòrico y pràctico. Teòricamente las DCGs tienen el poderio de una màquina de Turing, por lo que en ese sentido es lo màs general posible.

Las DCGs son concisas bàsicamente porque son una forma de programaciòn lògica, y los programas lògicos son en general màs concisos que los programas hechos en lenguajes convencionales. El factor principal es el uso de reconocimiento de patrones (pattern matching) en lugar de operaciones explicitas para la prueba de registros y la construcciòn de estructuras.

La ejecuciòn de las DCGs en Prolog provee un mecanismo de anàlisis gramatical tipo "top-down, left-to-right, depth-first" (arriba-hacia-abajo, de izquierda-a-derecha, de profundidad-primero). Las DCGs se expresan directamente en el lenguaje de programaciòn Prolog, por lo que se puede decir que una DCG es un programa Prolog. Las DCGs no requieren de un intèrprete o compilador especial. Por lo tanto, para trabajar eficientemente las DCGs hay que trabajar eficientemente Prolog.

A fin de poder proveer una estructura adecuada para el anàlisis de lenguaje, un formalismo no debe ser tan restrictivo que impida la experimentaciòn con nuevas ideas. Las DCGs tienen acceso total al subconjunto de la lògica de clàusulas definidas como un lenguaje de programaciòn de propòsito general, por lo que la prueba de gramàticas con DCGs es mas fàcil de experimentar inclusive con estrategias de anàlisis (parsing) radicalmente diferentes.

Las DCGs son también un formalismo útil para los estudios teóricos del lenguaje, y como consecuencia proveen un puente potencial entre el trabajo que realizan los lingüistas teóricos, los filósofos y aquellos dedicados a la ingeniería de sistemas de lenguaje natural.

En una DCG, la estructura obtenida del análisis de una frase puede depender de elementos que no han sido encontrados durante el transcurso del análisis gramatical de la oración (concepto de variable lógica).

Desde el punto de vista práctico, la claridad y modularidad de las DCGs son una ayuda vital en el desarrollo de sistemas de tamaño y complejidad necesarias para el análisis real del lenguaje natural.

Debido a que las DCGs consisten en pequeñas reglas independientes, es más fácil extender el sistema con nuevos constructos lingüísticos o modificar el tipo de estructuras que ya se tienen.

III.6 SUMARIO

En éste capitulo se discutieron los conceptos principales del formalismo de las Gramáticas de Cláusula Definida (DCG), que es uno de los formalismos (junto con las Gramáticas Estructuradas de Frase Generalizada (GPSG)) en que se basa éste trabajo de tesis.

Se mostró la relación que existe entre la DCG y la Gramática Libre de Contexto (CFG), así como la forma en que las DCGs trabajan. Se mostró también una metodología básica para la construcción de estructuras y la inclución de elementos adicionales para el análisis de lenguajes considerando el contexto, su implementación en el lenguaje de programación Prolog, y sus ventajas.

CAPITULO IV

GRAMATICA DE ESTRUCTURA DE FRASE GENERALIZADA (GPSG)

> La estructura gramatical de todos los idiomas implica la elaboración perfectamente organizada de pilas de desplazamiento descendente (push-down stacks).
>
> D.R. Hofstadter

IV.1 INTRODUCCION

El estudio y la comprensión de la Gramàtica de Estructura de Frase Generalizada es en extremo difícil. Como èsta tesis no persigue adquirir un conocimiento completo y un dominio total de la teoría, el tema se desarrolla en èste capítulo de la manera más sencilla posible, abarcando tan sòlo los aspectos de interès y con la profundidad necesaria para el desarrollo de la tesis.

El tema se desarrollò principalmente en base al libro "Generalized Phrase Structure Grammar" de Gazdar, Klein, Pullum y Sag [GAZ85], el artículo "From English to Logic: Context-Free Computation of 'Conventional' Logical Translation" de Schubert y Pelletier [SCH82], y el artículo "Phrase Structure Trees Bear More Fruit Than You Would Have Thought" de Joshi y Levy [JOS82].

La Gramàtica de Estructura de Frase Generalizada (GEFG, en inglès Generalized Phrase Structure Grammar GPSG) es un formalismo basado en la gramàtica generativa, y fuè propuesto por Gazdar en 1978 como una alternativa a las gramàticas transformativas. Las GPSG son una extensiòn de las Gramàticas de Estructura de Frase (GEF, en inglès Phrase Structure Grammars PSG). Los objetivos perseguidos por Gazdar al desarrollar las GPSG fueron principalmente:

a) Obtener un metalenguaje restringido capaz de definir las gramàticas de los lenguajes naturales.
b) Desarrollar un formalismo que incrementara el poder descriptivo de las PSG sin incrementar el poder generativo.
c) Restringir tal poder descriptivo-generativo ùnicamente a los lenguajes libres de contexto (context-free).

Las GPSG son capaces de manipular informaciòn tanto sintàctica como semàntica, permitiendo asi obtener traducciones de mejor calidad que aquellas realizadas ùnicamente con componentes sintàcticos. Pero no debe esperarse que las GPSG realicen un anàlisis semàntico complejo, debido a que el formalismo no logra adoptar todos los aspectos que implica el tratar la semàntica. Inclusive debe tomarse en cuenta que la interpretaciòn de un lenguaje natural està regido tambièn y en forma considerable por la pragmàtica y la fonètica.

Otra caracteristica importante de las GPSG es la facilidad de generaciòn de definiciones recursivas. Ello permite una màs fàcil representaciòn e implementaciòn en computadora digital, asi como un anàlisis màs eficiente para determinar la validez o rechazo de una oraciòn, ademàs de un anàlisis màs simple de las estructuras (reglas) que lo forman.

IV.2 CONCEPTOS

La gramàtica transformacional parece ser una teoria limitada inclusive para la descripciòn sintàctica de los lenguajes naturales, debido a que las reglas de transformaciòn y otros elementos tales como filtros y restricciones de movimiento no son del todo poderosos.

Montague desarrollò una serie de trabajos sobre la lògica del idioma Inglès. Sus trabajos se formalizaron en lo que suele denominarse Gramàtica de Montague, y de acuerdo a ellos, la correspondencia entre la estructura sintàctica y la forma lògica es màs sencilla de lo supuesto. Establece que a cada lexema le corresponde un tèrmino lògico o 'functor', y a cada regla de composiciòn sintàctica le

corresponde una regla semàntica de composiciòn lògica estructuralmente anàloga. Se le denominò Hipòtesis regla-a-regla (rule-to-rule).

La limitante principal de la gramàtica de Montague es que trata ùnicamente fragmentos sintàcticos simples del Inglès.

La teoria de Estructura de Frase desarrollada por Gazdar y otros surgiò a partir de los desarrollos de Montague, incorporando principalmente el enfoque semàntico. El segundo apoyo para el desarrollo de la teoria de Estructura de Frase es el desarrollo de la teoria transformativa de Chomsky, pues se han adoptado diversos conceptos de ella, como por ejemplo: la subcategorizaciòn, la coordinaciòn, etc.

A pesar de que las GPSG tienen su fundamento sintàctico principalmente en la teoria transformativa de Chomsky, no hacen uso de transformaciones. Las GPSG utilizan lo que los autores denominan como 'Traducciones'; su modo de operar se explicarà a detalle en el siguiente tema (cap. IV.3).

La semàntica que manejan las GPSG no pretende determinar la mala-formaciòn (ill-formedness) de las frases, sino màs bien indicar còmo las estructuras de una parte de la frase analizada se interpreta por medio de cierta regla.

IV.3 BASE TEORICA Y ESTRUCTURA

IV.3.1 DOMINANCIA INMEDIATA Y PRECEDENCIA LINEAL

Toda regla de estructura de frase especifica dos tipos de relaciones:

i) Dominancia inmediata (ID).
ii) Precedencia lineal (LP).

Si se tiene por ejemplo, la regla:

S --) NP VP

La dominancia inmediata nos indica que los nodos NP y VP están dominados de manera inmediata por S, es decir, se forma un árbol local (de profundidad uno) donde la raiz es el elemento S. El árbol puede ser cualquiera de los que se muestran:

Fig. IV.3 -1-

Una gramática debe imponer restricciones sobre la precedencia lineal de relaciones hermanas. En notación de GPSG se utiliza el operador '(' para indicar tal precedencia. Por ejemplo:

VP (NP

indica que VP debe preceder a NP. Por lo tanto, para que la regla dada sea interpretada adecuadamente, debe adicionarsele la condición de precedencia lineal (normalmente se agrega a modo de regla adicional), quedando:

i) S --) NP, VP

ii) NP (VP

IV.3.2 SUBCATEGORIZACION

Existen restricciones en contextos de ocurrencia de elementos léxicos que la gramática debe especificar, pero que no pueden reducirse exclusivamente a hechos referentes al significado.

Por ejemplo, para la siguiente regla de estructura de frase:

VP --) V (NP) (PP)

se presenta el problema de poder determinar cuando expander V con un NP, cuando expanderlo con un PP o cuando expanderlo con ambos elementos (o no expanderlo).

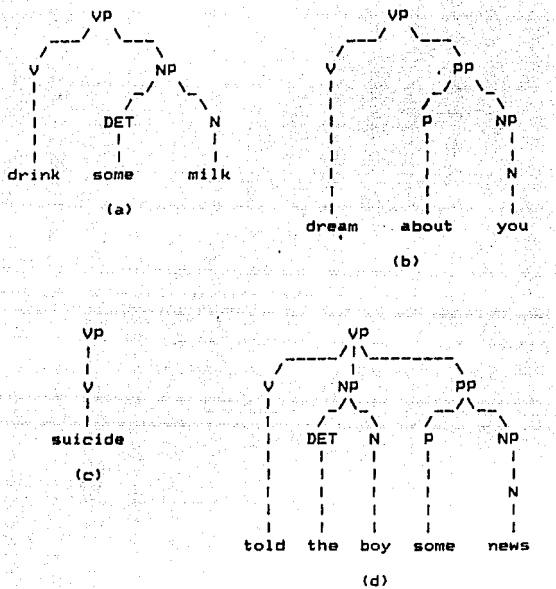


Fig. IV.3 -2-

El problema se puede resolver expandiendo la regla de acuerdo a subcategorizaciones. Esto se logra introduciendo en las reglas de estructura de frase libres de contexto de la gramàtica, símbolos preterminales tales como (adj, det, prep, etc.). Estos símbolos determinan las posibles categorizaciones de las palabras en las estructuras.

Gazdar introduce las subcategorizaciones de manera transparente agregando ùnicamente índices enteros al elemento SUBCAT de los elementos preterminales. Los índices enteros actùan como apuntadores a las reglas al interactuar con el lexicòn, por lo tanto para la regla de estructura de frase anterior se tendrà:

```
VP --) V[1]
VP --) V[2], NP
VP --) V[3], PP
VP --) V[4], NP, PP
```

IV.3.3 ESTRUCTURA

Es importante indicar que la gramàtica de Estructura de Frase no hace uso de transformaciones. Consiste completamente en reglas de estructura de frase, con aceptaciones nodales en lugar de interpretaciones generativas, suele denominàrseles traducciones. Por ejemplo, la regla:

```
[(S) --) (NP) (VP)]
```

establece que un fragmento con raiz S, rama izquierda NP y rama derecha VP es un fragmento admisible de un àrbol sintàctico. Tales reglas de estructura de frase son fàciles de entender y permiten el uso de mètodos de anàlisis (parsing) eficientes sobre gramàticas libres-de-contexto (context-free). Ademàs, la gramàtica dispone de la hipòtesis regla-a-regla emparentando cada regla sintàctica con una regla semàntica tipo Montague, la cual aporta la traducciòn lògica admitida por la regla sintàctica.

Gazdar sustituye las transformaciones con dos elementos metagramaticales:

- Esquemas de reglas.
- Metareglas.

Dichos elementos permiten operar las generaciones lingüisticas que supuestamente podian ser manejadas ùnicamente mediante transformaciones.

El uso de categorias con huecos (gaps) es una innovaciòn que consiste en eliminar un elemento de una categoria dada. Por ejemplo, NP/PP denota un NP al cuàl se le ha eliminado el PP. Tal tipo de categorizaciòn se introduce con el uso de esquemas de reglas y metareglas.

En la GPSG, argumentos tàles como NP y N son asumidos como denotadores de conjuntos de propiedades en lugar de denotadores de individuos.

GRAMATICA DE ESTRUCTURA DE FRASE GENERALIZADA

Considèrese la siguiente regla de Gazdar:

1. ((S) --) (NP) (VP); (VP' NP'))

El primer elemento indica el nùmero de regla, el segundo elemento indica la regla sintàctica, y el tercero la regla semàntica. La regla semàntica establece que la traducciòn en lògica intencional (lògica de predicados de 1er. y 2o. orden) del constituyente S està compuesto de la traducciòn VP (como functor) y la traducciòn NP (como operando).

Como se mencionò anteriormente, Gazdar indrodujo las categorias con huecos y las reglas asociadas. La finalidad principal es permitir la generaciòn de dependencias ilimitadas (unbounded dependencies).

Una construcciòn de dependencia ilimitada es aquella en la cual:

a) Una relaciòn sintàctica de algùn tipo existe entre las subestructuras, y
b) La distancia estructural entre èstas dos subestructuras no està restringida a un dominio finito, como por ejemplo que ambas subestructuras sean de la misma clàusula.

Tales construcciones deben conceptualizarse en tèrminos de geometria de Àrboles como constituyentes en tres partes: la parte superior, la media y la inferior. La parte superior es la subestructura que introduce la dependencia, la parte media es el dominio de la estructura que la dependencia expande (manejo de los huecos o gaps), y la parte inferior es la subestructura en que la dependencia termina o es eliminada.

Sea Vn el conjunto de simbolos no-terminales bàsicos. A partir de èl se define al conjunto D(Vn) de simbolos no-terminales derivado como:

$$D(Vn) = \{ @/\& \mid @, \& \quad Vn\}$$

Si se tiene por ejemplo que: si S y NP son los ùnicos dos simbolos no-terminales, entonces D(Vn) consistirà en S/S, S/NP, NP/NP, y NP/S. La interpretaciòn de la categoria derivada (categoria diagonalizada (slashed) o categoria con hueco (gap)) es como sigue: Un nodo etiquetado @/& dominarà los subàrboles de igual forma que aquellos que pueden ser dominados por @, con la excepciòn de que en algùn lugar de cada subàrbol de tipo @/& ocurrirà un nodo de la forma &/& dominando un presunto pronombre o la cadena vacia, y cada nodo que enlace @/& y &/& serà de la forma v/&. De ahi que @/& etiquete a un nodo de tipo &. De ahi que @/& etiquete a un nodo de tipo @ que domina el material que contiene un

hueco de tipo &. Por ejemplo, S/NP es una oración que tiene un NP extraviado en alguna parte.

Las reglas derivadas permiten la propagación de un hueco y las reglas de enlace permiten la introducción de una categoría con el hueco. Por ejemplo, dada la regla:

[S NP VP]

podemos obtener dos reglas derivadas:

[S/NP NP/NP VP]
[S/NP NP VP/NP]

Un ejemplo de regla de enlace es una regla (esquema de regla) que introduce un categoría con un hueco de la forma requerida para la actualización.

[S @ S/@]

para @ = PP , se convierte en:

[S PP S/PP]

Esta regla introduce una estructura como la que se muestra a continuación:

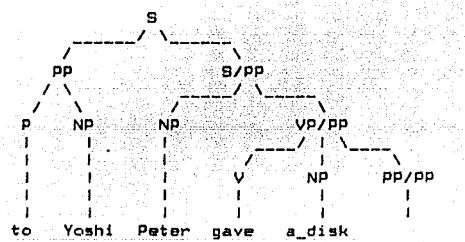


Fig. IV.4 -1-

Esta técnica de uso de categorías con huecos y el manejo de reglas derivadas y de enlace permite que las dependencias ilimitadas (unbounded dependencies) sean utilizadas en la representación de estructura de frase. El problema de conllevan es la gran proliferación de categorías tipo @/&.

IV.4 INTERPRETACION SEMANTICA

Gazdar et.al toman como base para la interpretación de la formación semántica correcta del lenguaje, el modelo desarrollado por Montague, y que se puede considerar como consistente en dos tareas:

i) La especificación de las denotaciones posibles de cada categoría de expresión sintácticamente determinada.
ii) Especificar la manera en que las denotaciones de expresiones complejas son producidas como una función de las denotaciones de sus constituyentes.

Por ejemplo para la oración:

Pedro juega

se puede determinar la interpretación analizándola de la siguiente forma:

[s [np Pedro][vp juega]]

lo que se pretende es darle a dicha forma una interpretación relativa al lenguaje natural involucrado. Por tanto y de acuerdo a (i) se debe decidir qué cosas serán denotadas por cada componente (S, NP, VP). Pero para que las denotaciones de las categorías de la gramática sean congruentes con la realidad (universo) que se desea representar es necesario especificar cómo la denotación de un elemento se debe combinar con otro elemento válido de (i) de acuerdo a (ii). Para ello se puede hacer uso de la denominada aplicación función-argumento.

El modelo nos permite entonces manejar dos opciones: que las denotaciones de NP sean aplicadas como función a las denotaciones de VP o viceversa.

Si se tiene por ejemplo la siguiente regla:

S --) NP, VP ; VP' NP"

ésta regla permite, tomando en cuenta la dominancia inmediata y la precedencia lineal, la construcción del árbol local que se muestra a continuación:

Fig. IV.5 -1-

Si se toman en cuenta las traducciones semànticas especificadas en la misma regla, el àrbol local tendrà asignado en S un VP' aplicado como 'functor' a NP", dando como resultado el àrbol interpretado siguiente:

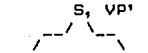


Fig. IV.5 -2-

Para ilustrar mejor esto, se puede considerar la siguiente gramàtica [RAD86]:

S --) NP, AUX, VP; VP' NP"
VP --) V, NP; NP' V"
NP --) DET, N

La cual genera estructuras con la que se pueden formar oraciones como la siguiente:

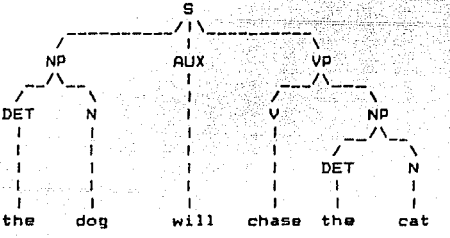


Fig IV.5 -3-

La oración generada es correcta, pero si en lugar del verbo chase se hubiera utilizado por ejemplo, el verbo V fall, o el nombre N book, las oraciònes generadas:

The dog will fall the cat *
The book will chase the cat *

resulta ser una oración incorrecta. Por lo tanto, para superar estos inconvenientes se puede hacer uso de las subcategorizaciones, de tal forma que la subcategorización

asociada a, por ejemplo el verbo V permita especificar y determinar adecuadamente en que tipo de estructura puede ser utilizado, por ejemplo:

```
chase: V, [+NP]      (-- V[1]
fall: V, [-NP]       (-- V[2]
```

En èste caso, V[1] es una categorizaciòn para el verbo transitivo chase y V[2] es una subcategorizaciòn para el verbo intransitivo fall.

Obviamente para la generaciòn apropiada de oraciones con sentido gramatical semàntico se requiere que las subcategorizaciones sean màs completas y se incluyan ademàs rasgos semànticos en el lexicòn como se ilustrò en el capitulo II.5. En el ejemplo anterior, para los nombres N book y dog la subcategorizaciòn en base a rasgos semànticos puede ser:

```
dog [+anim, +cont, -hum]      (-- N[1]
book [-anim, +cont, -hum]     (-- N[2]
```

y para el verbo V chase:

```
chase [+NP, +anim, +cont]
```

de èsta forma book no serà aceptado en la estructura para el verbo chase por no ser animado.

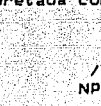
IV.5 LA GRAMATICA DE ESTRUCTURA DE FRASE GENERALIZADA Y EL PROLOG.

Se describirà ahora una manera en que se pueden adaptar los principios de la teoria de GPSGs en el lenguaje de programaciòn Prolog de acuerdo a sus caracteristicas, y a la forma en que se ha tomado en la implementaciòn del sistema de traducciòn presentado en èsta tesis.

La dominancia inmediata y la precedencia lineal es manejada por Prolog de manera totalmente transparente, puès mientras que la teoria de las GPSG establece que para que la regla de traduccion:

```
S --) NP VP
```

sea interpretada como:



es necesario especificarlo de acuerdo a la ID y LP como:

```
S --) NP, VP
NP ( VP
```

En Prolog se puede especificar la traducción como una regla de la siguiente forma:

```
S :- NP,
     VP.
```

La semántica de Prolog interpretará tal regla (traducción) de acuerdo a como lo exigen la Dominancia Inmediata y la Precedencia Lineal, es decir, para que S sea aceptada deben evaluarse satisfactoriamente NP y VP en ese orden. Recuérdese que Prolog evalúa sus cláusulas de arriba a abajo y de izquierda a derecha.

La subcategorización se maneja prácticamente de la misma forma en la teoría de GPSGs y en Prolog. En las GPSGs se establece que para que la regla de estructura de frase:

```
VP --) V (NP) (PP)
```

sea analizada apropiadamente es necesaria su subcategorización de la siguiente forma:

```
VP --) V[1]

VP --) V[2], NP
V[2] ( NP

VP --) V[3], PP
V[3] ( PP

VP --) V[4], NP, PP
V[4] ( NP ( PP
```

En Prolog únicamente se evita la indización (de V en éste caso). El desmenuzamiento de la regla de estructura de frase es necesaria para que el analizador Prolog considere todos los casos, quedando:

```
VP :- V.

VP :- V,
      NP.

VP :- V,
      PP.

VP :- V,
      NP,
      VP.
```

En las GPSGs se establece que una regla de traducción es a su vez un fragmento de otra regla de traducción. En Prolog es inherente tal tipo de manejo de reglas, pues una regla puede contener referencias a otras reglas que son evaluadas al momento de su invocación.

El uso de categorias con huecos (gaps) en las GPSGs es analizado do forma distinta en Prolog. Mientras que en las GPSGs es necesario separar el elemento que se desea analizar por separado e integrarlo posteriormente a la estructura en el hueco correspondiente, en Prolog el elemento a analizar es analizado total e independientemente, y una vez concluido su análisis es integrado al resto de la regla de traducción. Por ejemplo para:

```
VP --) TV, NP, PP
TV ( NP ( PP
```

se puede establecer:

```
VP/PP
```

indicando que el VP contiene un hueco para PP.

En Prolog la cláusula equivalente será:

```
VP :- TV,
      NP,
      PP.
```

como se sabe, Prolog evalua en primer lugar el argumento TV, posteriormente el NP de manera independiente y total; y finalmente el PP, que es integrado igualmente a la regla una vez evaluado de manera independiente y total.

IV.6 SUMARIO

El segundo formalismo en que se basa el desarrollo del sistema presentado en esta tesis es la Gramàtica de Estructura de Frase Generalizada (GPSG), por lo que en este capitulo se hace una descripciòn del mismo tomando en cuenta los fundamentos del formalismo adoptados en el desarrollo del sistema.

Se describen la dominancia inmediata y la precedencia lineal como bases teòricas, la estructura de las reglas de traducciòn y la semàntica del formalismo enfocada a la interpretaciòn misma de las estructuras y no a la determinaciòn de malformaciones semànticas a nivel interpretaciòn de lenguaje.

Por ùltimo se describe como se adaptaron los fundamentos de la GPSG en el lenguaje de programaciòn Prolog, lenguaje en que se desarrollò el sistema de traducciòn automàtica aqui presentado.

CAPITULO V

IMPLEMENTACION

> Algùn dia podremos registrar argumentos en una màquina del mismo modo que ahora hacemos los ingresos de ventas en una registradora.
>
> Vannevar Bush

V.1 INTRODUCCION

En este capitulo se describirá a grandes rasgos la implementaciòn del sistema que se presenta en esta tesis.

El sistema es un traductor Inglès-Español bidireccional que analiza las oraciones sintàctica y semànticamente. El procedimiento principal, los procedimientos de E/S y los analizadores gramaticales (Inglès y Español) abarcan aproximadamente 2100 lineas de còdigo, y el lexicòn aproximadamente 1400 lineas, ocupando en total casi 130 KBy. Està desarrollado en Prolog (Arity/Prolog V. 4.0) y trabaja en una microcomputadora tipo IBM PC con disco duro (hard disk) y 640 KBy de memoria principal.

V.2 ESTRUCTURA DEL SISTEMA

La estructura del sistema es del tipo propuesto por Jiménez (ver cap. I.4) y consta de 5 etapas:

1.- Entrada.
Consiste ùnicamente en la interface con el usuario para la lectura de la oraciòn que se desea (analizar y) traducir.

2.- Análisis.
Esta etapa efectúa el desmenuzamiento de la oración para poder identificar si las palabras que la comprenden forman parte del léxico del sistema. Lleva a cabo el análisis sintáctico de acuerdo a la gramática establecida y a la concordancia de rasgos sintácticos (como número y persona) a fin de determinar si la oración es gramatical y sintácticamente válida.
Realiza también el análisis semántico a fin de determinar si la oración es congruente con los conocimientos del mundo real que contiene el sistema y que son determinados fundamentalmente en base a los rasgos semánticos implementados y a la concordancia semántica establecida.

3.- Traducción.
En esta etapa se realiza, a partir de la estructura obtenida en el análisis de la oración en el idioma origen, la transformación necesaria para generar la estructura de lo que concluirá como la oración en el idioma destino.

4.- Síntesis.
Consiste en tomar la estructura generada en el proceso de traducción o transformación y obtener a partir de ella junto con el lexicón la oración en el idioma destino.

5.- Salida.
Es la interface con el usuario destinada a desplegar las oraciones fuente y destino.

V.3 CONTROL SINTACTICO-SEMANTICO

Como se mencionó anteriormente, se pretende que el sistema sea capáz de detectar las anomalías sintácticas que se presenten en la oración y de determinar la validéz semántica de la misma en base al conocimiento que el sistema tenga sobre el mundo real.

Se puede decir que el dominio del sistema está basado en un léxico de propósito general altamente restringido. Esta característica se puede justificar dado que es imposible obtener un sistema de traducción completo por diversas razones, entre ellas:

- Dado que los lenguajes naturales no son formales es muy dificil obtener una gramàtica òptima y completa de ellos.
- La generación de un lexicón que comprenda, además de todas las palabras del lenguaje natural (lenguajes en el caso de sistemas de traducción), todos los rasgos sintctico-semánticos necesarios para su correcta interpretación resulta ser una tarea extenuante.
- La generación de las reglas de traducción para la gramàtica 'completa', incluyendo el control de las excepciones del lenguaje y el manejo adecuado de los rasgos sintàctico_semànticos es también una labor sumamente complicada.
- No se cuenta con equipo de còmputo que sea capàz de soportar eficientemente el volumen de información que comprendería el lexicón y el conjunto de reglas de traducción, además de poder procesar la información de tal forma que proporcione resultados de manera no ambigua, ràpida, económica y confiable, es decir, altamente eficiente. Parece ser que el procesamiento en paralelo es por ahora el camino màs prometedor, sin embargo aún no es un recurso al alcance de muchos.
- El sistema mostrado en èsta tesis se desarrollò en un equipo de còmputo muy pequeño e ineficiente (computadora personal),
- No es objetivo de la tesis la generación del sistema completo sino ùnicamente la realización de un sistema de traducción que permita apreciar el control sintàctico-semántico y determinar la viabilidad de los formalismos como adecuados para el reconocimiento y entendimiento del lenguaje natural en un sistema de traducción automàtica.

Dentro del contexto que nos interesa, ello implica que una oración que quede comprendida dentro de la gramàtica del sistema resultarà invàlida por el solo hecho de no formar parte de su lèxico.

La validez sintáctica se determina, además de la gramaticalidad, mediante el uso de tres rasgos sintácticos:

- nùmero.
- persona.
- género.

Estos rasgos permiten operar adecuadamente las conjugaciones verbales, asi como los adjetivos determinativos, pronombres, auxiliares y cuantificadores. De esa forma se pueden operar y detectar anomalias en oraciones tales como:

pedro juega en el parque
los niños chicos cantan
la mujer vive con el ladrón

pocos hombres fumaron
pedro juegan en el parque (*)
los niño chicas cantan (*)
la mujer vive con la ladrón (*)
pocos hombre fumaron (*)

Pero no es posible detectar anomalias de tipo semántico como por ejemplo:

pedro juega en la pelota (?)
los perros chicos cantan (?)
la hamburguesa vive con el ladrón (?)
pocos pajaros fumaron (?)

Estos rasgos se incluyen en el lexicón y pueden ser transmitidos y heredados por los componentes gramaticales y determinar así su concordancia sintáctica.

Por otra parte, la validéz semántica se determina únicamente mediante la concordancia de rasgos semánticos. Los rasgos semánticos utilizados en el sistema son:

- animado.
- humano.
- genero.
- alas.
- localidad.
- alimento.

Estos rasgos permiten que las oraciones incorrectas que el analizador sintáctico no detecta (?), sean rechazadas por el analizador semántico. Al igual que los rasgos sintácticos, los rasgos semánticos se establecen en el lexicón como una lista de elementos. Por ejemplo, el rasgo semántico 'animado' impedirá que la oración:

la hamburguesa vive con el ladrón (*)

sea aceptada como válida debido a que en el lexicón se ha establecido que el verbo transitivo vivir debe consistir tanto en el sujeto activo como en el pasivo de un ser animado, y como 'hamburguesa' contiene el rasgo semántico correspondiente como inanimado, la oración es rechazada.

Los rasgos semánticos señalados se adoptaron porque se considera que con ellos es suficiente para el control del léxico seleccionado. Existen otros rasgos semánticos como por ejemplo 'contable' que permite diferenciar entre cosas que se pueden contabilizar (libro) y cosas que no pueden ser contabilizadas (agua).

En el lexicón y en el sistema los rasgos sintácticos y semánticos son operados como una sola lista ordenada de la siguiente manera:

```
[ número, persona, animado, humano, género,
  alas, localidad, alimento ]
```

bajo la siguiente notación y dominio:

```
          número : singular, plural
         persona : 1, 2, 3
         animado : anim_y, anim_n
          humano : hum_y, hum_n
          género : masc_y, masc_n
            alas : wing_y, wing_n
       localidad : loc_y, loc_n
        alimento : meal_y, meal_n
```

Por ejemplo:

```
lex_adj(american,[singular,3,_,_,masc_y,_,_,_],americano).

lex_com_noun(airplane,[singular,3,anim_n,hum_n,masc_y,wing_y,
                       loc_n,meal_n],avion).
lex_com_noun(boy,[singular,3,anim_y,hum_y,masc_y,wing_n,
                  loc_n,meal_n],nino).
lex_com_noun(tacos,[plural,3,anim_n,hum_n,masc_y,wing_n,
                    loc_n,meal_y],tacos).

lex_adj(tasty,[singular,3,_,hum_n,masc_y,_,loc_n,meal_y],sabroso).

lex_det(our,[singular,3,_,_,masc_n,_,_,_],nuestra).

lex_pn(peter,[anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],pedro).
lex_pn(fido,[anim_y,hum_n,masc_y,wing_n,loc_n,meal_n],fido).
lex_pn(cancun,[anim_n,hum_n,_,_,loc_y,meal_n],cancun).

lex_tv(eat,eat,present,[singular,1,anim_y,_,_,wing_n,_,_],
                       [_,_,_,_,_,_,_,meal_y],comer,como).
lex_tv(eat,eat,present,[singular,2,anim_y,_,_,wing_n,_,_],
                       [_,_,_,_,_,_,_,meal_y],comer,comes).
lex_tv(eat,eats,present,[singular,3,anim_y,_,_,wing_n,_,_],
                        [_,_,_,_,_,_,_,meal_y],comer,come).
lex_tv(eat,eat,present,[plural,1,anim_y,_,_,wing_n,_,_],
                       [_,_,_,_,_,_,_,meal_y],comer,comemos).
lex_tv(eat,eat,present,[plural,2,anim_y,_,_,wing_n,_,_],
                       [_,_,_,_,_,_,_,meal_y],comer,comeis).
lex_tv(eat,eat,present,[plural,3,anim_y,_,_,wing_n,_,_],
                       [_,_,_,_,_,_,_,meal_y],comer,comen).
```

V.4 MODULO PRINCIPAL

El módulo principal consiste en un procedimiento de control de todo el proceso (translator), un procedimiento de control del proceso de traducción Ingles-Español (translate), un procedimiento de control del proceso de traducción Español-Inglés (traduce), y los procedimientos de manejo de pantalla y entrada/salida.

El procedimiento principal (translator) gobierna a los procedimientos de manejo de pantalla, el menù y la invocación a los procedimientos de control: translate y traduce.

```
/*                  */
/* Main Procedure */
/*                  */


translator:-
    cls,
    box(0,0,24,79),
    repeat,
        main_screen,
        [!
          read_option(OPTION),
          case([OPTION == $F$ -) assert(finish),
                OPTION == $f$ -) assert(finish),
                OPTION == $1$ -) translate,
                OPTION == $2$ -) traduce ! fail ])
        !],
    finish,
    retract(finish),
    cls.
```

Fig. V.4.1 Procedimientto Principal (translator)
(Arity/Prolog V. 4.0)

Este procedimiento mantiene todo el sistema dentro de un hambiente de menù que itera hasta que el usuario seleccione la opción de finalización.

Los procedimientos de control del proceso de traducción (translate y traduce) accesan los procedimientos de entrada/salida, los procedimientos de anàlisis sintàctico-

semántico de los idiomas fuente y destino, y la procedimiento de traducción. Los dos procedimientos se muestran a continuación:

```
/*                    */
/*  English-Spanish   */
/*                    */


translate:-
    retract(end),
    retract(fin),
    tscroll(0,(2,2),(22,78)),
    tmove(4,23),
    wa(33,112),
    write($ ENGLISH - SPANISH   TRANSLATION $),
    repeat,
        [! english_spanish !],
    end.


english_spanish:-
    input_sentence1(ENGLISH_SENTENCE),
    [!
      tmove(15,3),
      write_list(ENGLISH_SENTENCE)
    !],
    english_parser(ENGLISH_SENTENCE,ENGLISH_PARSE_TREE),
    translation(ENGLISH_PARSE_TREE,SPANISH_PARSE_TREE), !,
    spanish_parser(SPANISH_SENTENCE,SPANISH_PARSE_TREE), !,
    tmove(18,3),
    write_list(SPANISH_SENTENCE),
    tmove(22,3),
    write($[ENTER]$),
    read_line(0,_),
    tscroll(0,(7,2),(22,78)).
```

Fig. V.4.2 Procedimiento de Traducción (Inglès-Español)

(Arity/Prolog V. 4.0)

```
/*                    */
/* Spanish-English    */
/*                    */


traduce:-
    retract(fin),
    retract(end),
    tscroll(0,(2,2),(22,78)),
    tmove(4,23),
    wa(31,112),
    write($ TRADUCCION  ESPAxOL - INGLES $),
    repeat,
        [! spanish_english !],
    fin.


spanish_english:-
    input_sentence2(SPANISH_SENTENCE),
    [!
      tmove(15,3),
      write_list(SPANISH_SENTENCE)
    !],
    spanish_parser(SPANISH_SENTENCE,SPANISH_PARSE_TREE),
    translation(ENGLISH_PARSE_TREE,SPANISH_PARSE_TREE), !,
    english_parser(ENGLISH_SENTENCE,ENGLISH_PARSE_TREE), !,
    tmove(18,3),
    write_list(ENGLISH_SENTENCE),
    tmove(22,3),
    write($[ENTER]$),
    read_line(0,_),
    tscroll(0,(7,2),(22,78)).
```

Fig. V.4.3 Procedimiento de Traducción
(Español-Inglés)
(Arity/Prolog V. 4.0)

Ambos procedimientos contienen la llamada al procedimiento de entrada (input_sentence) que se encarga de leer la oración dada por el usuario y corresponde a la primer etapa del proceso de traducción. Las llamadas a los procedimientos de análisis sintáctico-semántico tanto para análisis como para síntesis (english_parser y spanish_parser) según el caso corresponden a las etapas 2 y 4 del proceso de traducción y son invocadas en el siguiente orden: la de análisis después de haber leído y validado la

oración dada por el usuario (1a. etapa), y la de síntesis después del proceso de traducción. Por último contienen las llamadas al procedimiento de salida (write_list) que corresponde a la 5a. y última etapa. Estos procedimientos también iteran, por lo que para salir de ellas es necesario que el usuario teclee como oración de entrada la palabra 'end' o 'fin' según el caso.

Se puede observar que ambos procedimientos trabajan de igual forma, la diferencia radica tan solo en el orden en que los procedimientos son llamados por la procedimiento.

V.5 ANALIZADOR SINTACTICO-SEMANTICO, INGLES

Este módulo es el encargado de la generación de la estructura de transformación (árbol sintáctico-semántico) de las oraciones dadas en inglés durante el proceso de análisis. Así mismo efectúa el proceso de síntesis de la estructura obtenida del proceso de transformación para la extracción de la oración resultante en inglés.

El analizador está basado en la teoría de las DCGs descrita en el capítulo III, la teoría de las GPSGs descrita en el capítulo IV y en las teorías semánticas de Leech y katz descritas en el capítulo II.

La gramática utilizada se obtuvo del libro: "Transformational Syntax" de A. Radford, [RAD86] (ver fig. V.5.1). Esta gramática, a pesar de ser tan solo un subconjunto de la gramática del inglés más completa que se ha desarrollado, permite cubrir una variedad significativa de las oraciones del idioma. Debe tomarse en cuenta también que el sistema no puede analizar oraciones comprendidas dentro de la gramática que contengan palabras no incluidas en el lexicón. Todo esto limita en gran medida la magnitud de oraciones que el sistema puede analizar, pero debe recordarse que no es objetivo de la tesis la generación de un sistema "completo" de traducción.

```
S --) NP (AUX) VP

NP --) | PN              |
       | PRON            | (RC)
       | DET (ADJ) CN    | (RC PP)
       | Q (ADJ) CN      |

VP --) IV (ADVP) (PP)
VP --) TV NP (ADVP) (PP)

ADVP --) (DEG) ADV
PP --) PREP NP
RC --) REL VP
```

Fig. V.5.1 Gramàtica del Inglès.
(solo simbolos no terminales)

donde:

ADJ = Adjective
ADV = Adverb
ADVP = Adverbial Phrase
AUX = Auxiliary

CN = Common Noun

DEG = Degree Adverb
DET = Determiner

IV = Intransitive Verb

NP = Noun Phrase

PN = Proper Noun
PP = Prepòsitional Phrase
PREP = Preposition
PRON = Pronoun

Q = Quantifier

RC = Relational Clause
REL = Relational

S = Sentence

TV = Transitive Verb

VP = Verb Phrase

Fig. V.5.2 Simbologia de la Gramàtica

A fin de mostrar cómo funciona la implementación tomando en cuenta la teoria de las DCGs y GPSGs, asi como el manejo de rasgos semánticos desarrollados por Chomsky, Katz y Leech, se analizará la oración:

Peter cut an apple.

Las cláusulas del analizador que interesan para ilustrar el proceso son las que se muestran a continuación:

```
sentence([s,NP,VP],S0,S):-
    noun_phrase(NP,S0,S1),
    verb_phrase(NP,VP,S1,S).


noun_phrase([np,PROP_NOUN],S0,S):-
    proper_noun(PROP_NOUN,S0,S).

noun_phrase([np,DET,COM_NOUN],S0,S):-
    common_noun(COM_NOUN,S0,S),
    determiner(COM_NOUN,DET,S0,S1).


verb_phrase(NP1,[vp,TV,NP2],S0,S):-
    [!
      transitive_verb(NP1,TV,S0,S1),
      noun_phrase(NP2,S1,S)
    !],
    not(same(NP1,NP2)),
    match_tv_np(TV,NP2).


proper_noun([pn,W,PN_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S)
    !],
    lex_pn(W,PN_SSFEAT,_),
    append([singular,3],PN_SSFEAT,PN_SUBFEAT).


determiner(ELEM,[det,W,DET_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S),
      lex_eng_det(W,DET_SUBFEAT)
    !],
    check_a_an_det(ELEM,W),
    last(ELEM,ELEM_SUBFEAT),
    same(DET_SUBFEAT,ELEM_SUBFEAT).
```

```
determiner(ELEM,[det,W,DET_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S),
      lex_eng_det(W,DET_SUBFEAT)
    !],
    W \== 'a',
    W \== 'an',
    last(ELEM,ELEM_SUBFEAT),
    same(DET_SUBFEAT,ELEM_SUBFEAT).


check_a_an_det([_,WORD|_],W):-
    W == 'an',
    atom_string(WORD,STRING),
    nth_char(0,STRING,ASCII), !,
    case([ASCII == 97 -> true,
          ASCII == 101 -> true,
          ASCII == 105 -> true,
          ASCII == 111 -> true,
          ASCII == 117 -> true| fail]).

check_a_an_det([_,WORD|_],W):-
    W == 'a',
    atom_string(WORD,STRING),
    nth_char(0,STRING,ASCII), !,
    case([ASCII == 97 -> fail,
          ASCII == 101 -> fail,
          ASCII == 105 -> fail,
          ASCII == 111 -> fail,
          ASCII == 117 -> fail| true]).


common_noun([com_noun,W,CN_SUBFEAT],S0,S):-
    [!
      connects(S0,_,S1),
      connects(S1,W,S)
    !],
    lex_com_noun(W,CN_SUBFEAT,_).

common_noun(ELEM,[com_noun,W,CN_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S),
      last(ELEM,ELEM_SUBFEAT)
    !],
    lex_com_noun(W,CN_SUBFEAT,_),
    same(ELEM_SUBFEAT,CN_SUBFEAT).


transitive_verb(NP,[tv,ROOT,TENSE,
                    TV_SUBFEAT1,TV_SUBFEAT2],S0,S):-
    [!
      get_subfeat(NP,NP_SUBFEAT),
      connects(S0,W,S)
```

```
        !],
        lex_tv(ROOT,W,TENSE,TV_SUBFEAT1,TV_SUBFEAT2,_,_),
        same(NP_SUBFEAT,TV_SUBFEAT1).


match_tv_np(TV,NP):-
        [!
          last(TV,TV_SUBFEAT),
          get_subfeat(NP,NP_SUBFEAT)
        !],
        same(TV_SUBFEAT,NP_SUBFEAT).


same(SUBFEAT,SUBFEAT).
```

Fig. V.5.3 Cláusulas del Inglés

Las cláusulas anteriores pueden anàlizar oraciones que se encuentren dentro del dominio de la siguiente gramàtica:

```
S --) NP VP

NP --) PN
NP --) DET CN

VP --) TV NP
```

Fig. V.5.4 Fragmento de la gramàtica del sistema

```
lex_com_noun(apple,[singular,3,anim_n,hum_n,masc_n,wing_n,
                    loc_n,meal_y],manzana).

lex_det(an,[singular,3,_,_,masc_y,_,_,_],un).
lex_det(an,[singular,3,_,_,masc_n,_,_,_],una).

lex_eng_det(a,[singular,3,_,_,_,_,_,_]).
lex_eng_det(an,[singular,3,_,_,_,_,_,_]).

lex_sp_det(un,[singular,3,_,_,masc_y,_,_,_]).
lex_sp_det(una,[singular,3,_,_,masc_n,_,_,_]).

lex_pn(peter,[anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],pedro).
```

```
lex_tv(cut,cut,past,[singular,1,_,hum_y,_,_,_,_],
                    [_,_,_,_,_,_,_,meal_y],cortar,corte).
lex_tv(cut,cut,past,[singular,2,_,hum_y,_,_,_,_],
                    [_,_,_,_,_,_,_,meal_y],cortar,cortaste).
lex_tv(cut,cut,past,[singular,3,_,hum_y,_,_,_,_],
                    [_,_,_,_,_,_,_,meal_y],cortar,corto).
```

Fig. V.5.5 Fragmento del Lexicòn del sistema

El proceso es bàsicamente el mismo que se muestra en el capitulo III.4 adicionandole las siguientes caracteristicas:

- Los rasgos sintacticos y semànticos son transmitidos a modo de lista como un paràmetro dentro de los paràmetros que van conformando el àrbol sintàctico-semàntico de la oraciòn.
- Las condiciones extra estàn ubicadas no al final de la clàusula, sino en el lugar que se considera màs apropiado para la correcta interpretaciòn semàntica (en ocasiones sintàctica).
- Los 'parametros hereditarios', cuando existan, seràn siempre el primer paràmetro de la lista.
- Los componentes de los elementos gramaticales se encuentran en ocasiones en orden distinto al indicado por la gramàtica debido a que tal orden de anàlisis permite que la interpretaciòn semàntica sea màs eficiente y apropiada.
- En caso de existir casos especiales para la interpretaciòn correcta de las oraciones, son controlados en las clàusulas correspondientes a los simbolos terminales de la gramàtica.

El proceso de 'backtracking' de Prolog permite que sean rastreadas las clàusulas necesarias (todas de ser necesario) para la interpretaciòn y generaciòn del àrbol de anàlisis (parse tree). A continuaciòn se muestra el rastreo de las clàusulas anteriores para el anàlisis de la oraciòn citada, nòtese la transmisiòn de los paràmetros hereditarios, el orden de ejecuciòn y las condiciones extra:

nota: los paràmetros con formato '_XXXX' hex. son la representaciòn en Arity/Prolog de las variables anònimas.

```
CALL: english_parser([peter,cut,an,apple],_015D) ? )
CALL: sentence(_015D,[peter,cut,an,apple],[]) ? )
CALL: noun_phrase(_234D,[peter,cut,an,apple],_2379) ? )
CALL: proper_noun(_24C1,[peter,cut,an,apple],_2379) ? )
CALL: connects([peter,cut,an,apple],_2609,_2379) ? )
EXIT: connects([peter,cut,an,apple],peter,[cut,an,apple])
```

```
CALL: lex_pn(peter, _2655, _2659) ? )
EXIT: lex_pn(peter, [anim_y, hum_y, masc_y, wing_n, loc_n, meal_n]
          , pedro)
CALL: append([singular, 3],
             [anim_y, hum_y, masc_y, wing_n, loc_n, meal_n],
             _2615) ? )
EXIT: append([singular, 3],
             [anim_y, hum_y, masc_y, wing_n, loc_n, meal_n],
             [singular, 3, anim_y, hum_y, masc_y, wing_n, loc_n,
              meal_n])
EXIT: proper_noun([pn, peter,
                   [singular, 3, anim_y, hum_y, masc_y, wing_n,
                    loc_n, meal_n]],
                  [peter, cut, an, apple], [cut, an, apple])
EXIT: noun_phrase([np, [pn, peter,
                        [singular, 3, anim_y, hum_y, masc_y,
                         wing_n, loc_n, meal_n]]],
                  [peter, cut, an, apple], [cut, an, apple])
CALL: verb_phrase([np, [pn, peter, [singular, 3, anim_y, hum_y,
                       masc_y, wing_n, loc_n, meal_n]]],
                  -2359, [cut, an, apple], []) ? )
CALL: transitive_verb([np, [pn, peter,
                            [singular, 3, anim_y, hum_y, masc_y,
                             wing_n, loc_n, meal_n]]],
                      -3279, [cut, an, apple], _32BD) ? )
CALL: get_subfeat([np, [pn, peter,
                        [singular, 3, anim_y, hum_y, masc_y,
                         wing_n, loc_n, meal_n]]], _3485) ? )
EXIT: get_subfeat([np, [pn, peter,
                        [singular, 3, anim_y, hum_y, masc_y,
                         wing_n, loc_n, meal_n]]],
                  [singular, 3, anim_y, hum_y, masc_y, wing_n,
                   loc_n, meal_n])
CALL: connects([cut, an, apple], _3491, _32BD) ? )
EXIT: connects([cut, an, apple], cut, [an, apple])
CALL: lex_tv(_3431, cut, _343D, _3449, _3455, _34BD, _34C1) ? )
EXIT: lex_tv(cut, cut, past,
             [singular, 3, _4801, hum_y, _4819, _4825, _4831, _483D],
             [_4849, _4855, _4861, _486D, _4879, _4885, _4891,
              meal_y], cortar, corto)
CALL: same([singular, 3, anim_y, hum_y, masc_y, wing_n, loc_n,
            meal_n],
           [singular, 3, _4801, hum_y, _4819, _4825, _4831, _483D])
EXIT: same([singular, 3, anim_y, hum_y, masc_y, wing_n, loc_n,
            meal_n],
           [singular, 3, anim_y, hum_y, masc_y, wing_n, loc_n,
            meal_n])

EXIT: transitive_verb([np, [pn, peter,
                            [singular, 3, anim_y, hum_y, masc_y,
                             wing_n, loc_n, meal_n]]],
                      [tv, cut, past,
                       [singular, 3, anim_y, hum_y, masc_y,
```

```
                                wing_n, loc_n, meal_n],
                               [_4849, _4855, _4861, _486D, _4879, _4885,
                                _4891, meal_y]],
                              [cut, an, apple], [an, apple])
     CALL: noun_phrase(_3285, [an, apple], []) ? )
     CALL: proper_noun(_4CDD, [an, apple], []) ? )
     CALL: connects([an, apple], _4E25, []) ? )
     FAIL: connects([an, apple], _4E25, [])
     FAIL: proper_noun(_4CDD, [an, apple], [])
     CALL: common_noun(_4CF5, [an, apple], []) ? )
     CALL: connects([an, apple], _4E99, 4E9D) ? )
     EXIT: connects([an, apple], an, [apple])
     CALL: connects([apple], _4E5D, []) ? )
     EXIT: connects([apple], apple, [])
     CALL: lex_com_noun(apple, _4E69, 4EBD) ? )
     EXIT: lex_com_noun(apple, [singular, 3, anim_n, hum_n, masc_n,
                                wing_n, loc_n, meal_y],
                        manzana)
     EXIT: common_noun([com_noun, apple, [singular, 3, anim_n, hum_n,
                                          masc_n, wing_n, loc_n,
                                          meal_y]],
                       [an, apple], [])
     CALL: determiner([com_noun, apple, [singular, 3, anim_n, hum_n,
                                          masc_n, wing_n, loc_n,
                                          meal_y]],
                      _4CE9, [an, apple], _4D29) ? )
     CALL: connects([an, apple], _4E99, 4E9D) ? )
     EXIT: connects([an, apple], an, [apple])
     CALL: lex_eng_det(an, _5669) ? )
     EXIT: lex_eng_det(an, [singular, 3, _59F1, _59FD, _5A09, _5A15,
                            _5A21, _5A2D])
     CALL: check_a_an_det([com_noun, apple,
                           [singular, 3, anim_n, hum_n, masc_n,
                            wing_n, loc_n, meal_y]], an) ? )
     EXIT: check_a_an_det([com_noun, apple,
                           [singular, 3, anim_n, hum_n, masc_n,
                            wing_n, loc_n, meal_y]], an)
     CALL: last([com_noun, apple, [singular, 3, anim_n, hum_n, masc_n,
                                   wing_n, loc_n, meal_y]], _56D9) ? )
     EXIT: last([com_noun, apple, [singular, 3, anim_n, hum_n, masc_n,
                                   wing_n, loc_n, meal_y]],
                                  [singular, 3, anim_n, hum_n, masc_n,
                                   wing_n, loc_n, meal_y]])
     CALL: same([singular, 3, _59F1, _59FD, _5A09, _5A15, _5A21, _5A2D],
                [singular, 3, anim_n, hum_n, masc_n, wing_n, loc_n,
                 meal_y]) ? )
     EXIT: same([singular, 3, anim_n, hum_n, masc_n, wing_n, loc_n,
                 meal_y],
                [singular, 3, anim_n, hum_n, masc_n, wing_n, loc_n,
                 meal_y])
     EXIT: determiner([com_noun, apple, [singular, 3, anim_n, hum_n,
                                          masc_n, wing_n, loc_n,
                                          meal_y]],
```

```
                        [det,an,[singular,3,anim_n,hum_n,masc_n,
                                 wing_n,loc_n,meal_y]],
                        [an,apple],[apple])
EXIT: noun_phrase([np,[det,an,[singular,3,anim_n,hum_n,
                               masc_n,wing_n,loc_n,meal_y]],
                      [com_noun,apple,
                       [singular,3,anim_n,hum_n,masc_n,
                        wing_n,loc_n,meal_y]]],
                  [an,apple],[])
CALL: not same([np,[pn,peter,
                    [singular,3,anim_y,hum_y,masc_y,wing_n,
                     loc_n,meal_n]]],
               [np,[det,an,[singular,3,anim_n,hum_n,
                            masc_n,wing_n,loc_n,meal_y]],
                   [com_noun,apple,
                    [singular,3,anim_n,hum_n,masc_n,
                     wing_n,loc_n,meal_y]]] ? )
EXIT: not same([np,[pn,peter,
                    [singular,3,anim_y,hum_y,masc_y,wing_n,
                     loc_n,meal_n]]],
               [np,[det,an,[singular,3,anim_n,hum_n,
                            masc_n,wing_n,loc_n,meal_y]],
                   [com_noun,apple,
                    [singular,3,anim_n,hum_n,masc_n,
                     wing_n,loc_n,meal_y]]])
CALL: match_tv_np([tv,cut,past,
                   [singular,3,anim_y,hum_y,masc_y,wing_n,
                    loc_n,meal_n],
                   [_4849,_4855,_4861,_486D,_4879,_4885,
                    _4891,meal_y]],
                  [np,[det,an,[singular,3,anim_n,hum_n,
                               masc_n,wing_n,loc_n,meal_y]],
                      [com_noun,apple,
                       [singular,3,anim_n,hum_n,masc_n,
                        wing_n,loc_n,meal_y]]] ? )
EXIT: match_tv_np([tv,cut,past,
                   [singular,3,anim_y,hum_y,masc_y,wing_n,
                    loc_n,meal_n],
                   [singular,3,anim_n,hum_n,
                    masc_n,wing_n,loc_n,meal_y]],
                  [np,[det,an,[singular,3,anim_n,hum_n,
                               masc_n,wing_n,loc_n,meal_y]],
                      [com_noun,apple,
                       [singular,3,anim_n,hum_n,masc_n,
                        wing_n,loc_n,meal_y]]])
EXIT: verb_phrase([np,[pn,peter,
                       [singular,3,anim_y,hum_y,masc_y,
                        wing_n,loc_n,meal_n]]],
                  [vp,[tv,cut,past,
                       [singular,3,anim_y,hum_y,masc_y,
                        wing_n,loc_n,meal_n],
                       [singular,3,anim_n,hum_n,
                        masc_n,wing_n,loc_n,meal_y]],
```

```
                        [np, [det, an, [singular, 3, anim_n,
                                        hum_n, masc_n, wing_n,
                                        loc_n, meal_y]],
                             [com_noun, apple,
                             [singular, 3, anim_n, hum_n, masc_n,
                              wing_n, loc_n, meal_y]]]],
                   [cut, an, apple], [])
EXIT: sentence([s, [np, [pn, peter,
                         [singular, 3, anim_y, hum_y, masc_y,
                          wing_n, loc_n, meal_n]]],
                   [vp, [tv, cut, past,
                         [singular, 3, anim_y, hum_y, masc_y,
                          wing_n, loc_n, meal_n],
                         [singular, 3, anim_n, hum_n,
                          masc_n, wing_n, loc_n, meal_y]],
                        [np, [det, an, [singular, 3, anim_n,
                                        hum_n, masc_n, wing_n,
                                        loc_n, meal_y]],
                             [com_noun, apple,
                             [singular, 3, anim_n, hum_n, masc_n,
                              wing_n, loc_n, meal_y]]]],
                   [peter, cut, an, apple], [])

EXIT: english_parser([peter, cut, an, apple],
                     [s,
                      [np, [pn, peter,
                            [singular, 3, anim_y, hum_y,
                             masc_y, wing_n, loc_n, meal_n]]],
                      [vp, [tv, cut, past,
                            [singular, 3, anim_y, hum_y, masc_y,
                             wing_n, loc_n, meal_n],
                            [singular, 3, anim_n, hum_n,
                             masc_n, wing_n, loc_n, meal_y]],
                           [np, [det, an, [singular, 3, anim_n,
                                           hum_n, masc_n, wing_n,
                                           loc_n, meal_y]],
                                [com_noun, apple,
                                [singular, 3, anim_n, hum_n,
                                 masc_n, wing_n, loc_n, meal_y]
                                ]]]],
                      [peter, cut, an, apple], [])
```

Fig. V.5.6 Rstreo de una oración en Inglès

A mayor complejidad de la oración, mayor tiempo de anàlisis es necesario debido a que el recorrido del Árbol gramatical crece en proporción al numero de palabras que contiene la oración y al numero de elementos lèxicos que se deben recorrer.

Si una oración es incorrecta (lo cual implica un recorrido extenuante del árbol gramatical), ya sea por error de entrada, por la mala construcción de la oración, por incongruencia semántica, o porque alguna de las palabras no está incluida en el lexicón; el sistema desplegará un mensaje indicando que la oración es incorrecta.

V.6 TRADUCTOR BIDIRECCIONAL

El módulo de traducción consiste básicamente en tomar la estructura de árbol sintáctico-semántico generado por el analizador (parser) del lenguaje fuente y generar a partir de él y el lexicón la estructura de árbol correspondiente al lenguaje destino.

El traductor funciona de manera similar a los analizadores (parsers). Separa el árbol por categorías gramaticales de manera sucesiva hasta alcanzar las hojas del árbol. En ese momento realiza las consultas necesarias al lexicón para obtener la traducción correspondiente y construir mediante la integración de los elementos gramaticales el árbol correspondiente al lenguaje destino.

Por ejemplo para la oración analizada en el tema anterior:

Peter cut an apple.

y para la cual el analizador sintáctico-semántico Inglés genera la estructura de árbol:

```
english_parser([peter,cut,an,apple],
               [s,
                [np,[pn,peter,
                     [singular,3,anim_y,hum_y,
                      masc_y,wing_n,loc_n,meal_n]]],
                [vp,[tv,cut,past,
                     [singular,3,anim_y,hum_y,masc_y,
                      wing_n,loc_n,meal_n],
```

```
                    [singular,3,anim_n,hum_n,
                     masc_n,wing_n,loc_n,meal_y]],
                  [np,[det,an,[singular,3,anim_n,hum_n,
                              masc_n,wing_n,loc_n,meal_y]],
                      [com_noun,apple,
                       [singular,3,anim_n,hum_n,masc_n,
                        wing_n,loc_n,meal_y]]]]],
            [peter,cut,an,apple],[])
```

Fig. V.6.1 Arbol de anàlisi Inglès

Inicialmente la estructura se ajustarà (match) a la regla de traducciòn:

```
translation([s,[np|NP],[vp|VP]],[s,[np|S_NP],[vp|S_VP]]):-
    trans_np(NP,S_NP),
    trans_vp(VP,S_VP).
```

la cual instanciarà:

```
NP (-- [np,[pn,peter,[singular,3,anim_y,hum_y,masc_y,
                      wing_n,loc_n,meal_n]]]

VP (-- [vp,[tv,cut,past,[singular,3,anim_y,hum_y,masc_y,
                         wing_n,loc_n,meal_n],
                       [singular,3,anim_n,hum_n,masc_n,
                        wing_n,loc_n,meal_y]],
             [np,[det,an,[singular,3,anim_n,hum_n,masc_n,
                         wing_n,loc_n,meal_y]],
                 [com_noun,apple,[singular,3,anim_n,hum_n,
                                  masc_n,wing_n,loc_n,
                                  meal_y]]]]
```

La clàusula instanciada para 'trans_np' serà:

```
trans_np([[pn,PROP_NOUN,SFEAT]],[[pn,S_PROP_NOUN,SFEAT]]):-
    trans_pn(PROP_NOUN,SFEAT,S_PROP_NOUN).
```

de la siguiente forma:

```
trans_np([[pn,peter,[singular,3,anim_y,hum_y,masc_y,
                                wing_n,loc_n,meal_n]]],
        [[pn,S_PROP_NOUN,SFEAT]]):-
    trans_pn(peter,[singular,3,anim_y,hum_y,masc_y,wing_n,
                    loc_n,meal_n],S_PROP_NOUN).
```

la cual instanciarà a su vez a la clàusula:

```
trans_pn(W,[_,_|SSFEAT],SW):-
    lex_pn(W,SSFEAT,SW).
```

de la siguiente forma:

```
trans_pn(peter,[_,_|[anim_y,hum_y,masc_y,wing_n,loc_n,
                        meal_n]],SW):-
    lex_pn(peter,[anim_y,hum_y,masc_y,wing_n,loc_n,
                    meal_n],SW).
```

que es el nivel hoja de la regla de traducciòn para NP --> PN, y es en ese momento cuando se obtiene la traducciòn:

```
lex_pn(peter,[anim_y,hum_y,masc_y,wing_n,loc_n,
                meal_n],pedro).
```

El proceso es esencialmente el mismo para la traducciòn de la frase verbal (verb_phrase) y los demàs elementos del àrbol sintàctico-semàntico.

La clàusula instanciada para 'trans_vp' serà:

```
trans_vp([[[tv,ROOT,TENSE,SFEAT1,SFEAT2,],[np|NP]]],
        [[tv,S_ROOT,TENSE,SFEAT1,SFEAT2],
        ,[np|S_NP]]):-
    [!
      trans_tv(ROOT,TENSE,SFEAT1,SFEAT2,S_ROOT)
    !],
    trans_check_tv(S_ROOT),
    trans_np(NP,S_NP).
```

de la forma:

```
trans_vp([[tv,cut,past,
           [singular,3,anim_y,hum_y,masc_y,wing_n,
            loc_n,meal_n],
           [singular,3,anim_n,hum_n,masc_n,wing_n,
            loc_n,meal_y]],
          [np,[det,an,[singular,3,anim_n,hum_n,masc_n,
                       wing_n,loc_n,meal_y]],
              [com_noun,apple,[singular,3,anim_n,hum_n,
                               masc_n,wing_n,loc_n,
                               meal_y]]]],
          [[tv,S_ROOT,past,
           [singular,3,anim_y,hum_y,masc_y,wing_n,
            loc_n,meal_n],
           [singular,3,anim_n,hum_n,masc_n,wing_n,
            loc_n,meal_y]],
          ,[np|S_NP]]):-
     [!
       trans_tv(cut,past,[singular,3,anim_y,hum_y,masc_y,
                          wing_n,loc_n,meal_n],
                         [singular,3,anim_n,hum_n,masc_n,
                          wing_n,loc_n,meal_y],S_ROOT)
     !],
     trans_check_tv(S_ROOT),
     trans_np([[det,an,[singular,3,anim_n,hum_n,masc_n,
                        wing_n,loc_n,meal_y]],
               [com_noun,apple,[singular,3,anim_n,hum_n,
                                masc_n,wing_n,loc_n,
                                meal_y]]],S_NP).
```

que utilizando las cláusulas:

```
trans_np([[det,DET,DET_SUBFEAT],
          [com_noun,COM_NOUN,CN_SUBFEAT]],
         [[det,S_DET,DET_SUBFEAT],
          [com_noun,S_COM_NOUN,CN_SUBFEAT]]):-
     trans_com(COM_NOUN,CN_SUBFEAT,S_COM_NOUN),
     trans_det(COM_NOUN,DET,DET_SUBFEAT,S_DET).

trans_det(CN,W,SFEAT,SW):-
     lex_det(W,SFEAT,SW),
     W == 'an',
     atom_string(CN,STRING),
     nth_char(0,STRING,ASCII),
     case([ASCII == 97 -> true,
           ASCII == 101 -> true,
           ASCII == 105 -> true,
           ASCII == 111 -> true,
           ASCII == 117 -> true| fail]).

trans_det(_,W,SFEAT,SW):-
     lex_det(W,SFEAT,SW).
```

```
trans_com(W,SFEAT,SW):-
    lex_com_noun(W,SFEAT,SW).

trans_tv(W,TENSE,SFEAT1,SFEAT2,SW):-
    lex_tv(W,_,TENSE,SFEAT1,SFEAT2,SW,_).
```

Dando como resultado final:

```
translation([s,[np,[pn,peter,[singular,3,anim_y,hum_y,masc_y,
                              wing_n,loc_n,meal_n]]],
             [vp,[tv,cut,past,[singular,3,anim_y,hum_y
                               masc_y,wing_n,loc_n,meal_n],
                              [singular,3,anim_n,hum_n,
                               masc_ywing_n,loc_n,meal_y]],
                 [np,[det,an,[singular,3,anim_n,hum_n,
                              masc_n,wing_n,loc_n,meal_y]],
                     [com_noun,apple,[singular,3,anim_n,
                                      hum_n,masc_n,wing_n,
                                      loc_n,meal_y]]]],
            [s,[np,[pn,pedro,[singular,3,anim_y,hum_y,masc_y,
                              wing_n,loc_n,meal_n]]],
             [vp,[tv,corto,past,[singular,3,anim_y,hum_y
                                 masc_y,wing_n,loc_n,meal_n],
                                [singular,3,anim_n,hum_n,
                                 masc_ywing_n,loc_n,meal_y]],
                 [np,[det,una,[singular,3,anim_n,hum_n,
                              masc_n,wing_n,loc_n,meal_y]],
                     [com_noun,manzana,[singular,3,anim_n,
                                        hum_n,masc_n,wing_n,
                                        loc_n,meal_y]]]]])
```

Que son los árboles de análisis (parser trees) sintáctico-semántico de los lenguajes fuente y destino.

V.7 ANALIZADOR SINTACTICO-SEMANTICO, ESPAÑOL

Como se mencionó en el capitulo II, las gramàticas del Inglès y del Español son de orígenes distintos, sin embargo existe una gràn semejanza en, al menos, las estructuras gramaticales bàsicas; de hecho solamente varia la posicion de ciertos elementos gramaticales.

En las gramàticas seleccionadas para el desarrollo de èsta tesis, las diferencias son:

1) El Adjetivo (ADJ) en la gramàtica del Inglès se encuentra antes del nombre comùn (CN), mientras que en la gramàtica del Español se encuentra despuès.
2) En la frase verbal (VP) que analiza los verbos transitivos (TV), se tiene un elemento gramatical adicional, el conectivo (CONN). La función del conectivo (en este caso 'a') consiste en darle adecuaciòn gramatical a las oraciones que requieren de èl despuès del TV.

Pedro admira las aves en el parque.

Pedro ama a las mujeres hermosas.
↑
|___ conectivo

Es evidente que el Español es un idioma muy rico en expresiones y vocabulario, por lo que su gramàtica es igualmente diversa, sin embargo con las reglas gramaticales seleccionadas es posible analizar un número aceptable de oraciones comunes.

A continuación se muestra la gramàtica seleccionada para el anàlisis del Español:

```
S --) NP (AUX) VP

NP --) | PN             |
       | PRON           | (RC)
       | DET CN (ADJ)   | (RC PP)
       | Q CN (ADJ)     |

VP --) IV (ADVP) (PP)
VP --) TV (CONN) NP (ADVP) (PP)

ADVP --) (DEG) ADV
PP --) PREP NP
RC --) REL VP
```

Fig. V.7.1 Gramàtica del Español (solo simbolos no terminales)

donde:

```
ADJ  = Adjective
ADV  = Adverb
ADVP = Adverbial Phrase
AUX  = Auxiliary

CN   = Common Noun
CONN = Connective

DEG  = Degree Adverb
DET  = Determiner

IV   = Intransitive Verb

NP   = Noun Phrase

PN   = Proper Noun
PP   = Prepositional Phrase
PREP = Preposition
PRON = Pronoun

Q    = Quantifier

RC   = Relational Clause
REL  = Relational

S    = Sentence

TV   = Transitive Verb

VP   = Verb Phrase
```

Fig. V.7.2 Simbología de la gramática

Dado que las gramáticas de ambos idiomas son similares, es de suponer que el conjunto de reglas de traducción o cláusulas sea también similar. A continuación se muestra el conjunto de cláusulas del Español utilizadas para el análisis de la oración:

Pedro cortó una manzana.

```
sp_sentence([s,NP,VP],S0,S):-
    sp_noun_phrase(NP,S0,S1),
    sp_verb_phrase(NP,VP,S1,S).

sp_noun_phrase([np,PROP_NOUN],S0,S):-
    sp_proper_noun(PROP_NOUN,S0,S).
```

```
sp_noun_phrase([np,DET,COM_NOUN],S0,S):-
    sp_determiner(DET,S0,S1),
    sp_common_noun(DET,COM_NOUN,S1,S).

sp_verb_phrase(NP1,[vp,TV,NP2],S0,S):-
    [!
      sp_transitive_verb(NP1,TV,S0,S1),
      check_tv(TV),
      sp_noun_phrase(NP2,S1,S)
    !],
    not(same(NP1,NP2)),
    match_tv_np(TV,NP2).

sp_proper_noun([pn,W,PN_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S)
    !],
    lex_pn(_,PN_SSFEAT,W),
    append([singular,3],PN_SSFEAT,PN_SUBFEAT).

sp_determiner([det,W,DET_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S)
    !],
    lex_sp_det(W,DET_SUBFEAT).

sp_common_noun(ELEM,[com_noun,W,CN_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S),
      last(ELEM,ELEM_SUBFEAT)
    !],
    lex_com_noun(_,CN_SUBFEAT,W),
    same(ELEM_SUBFEAT,CN_SUBFEAT).

sp_transitive_verb(NP,[tv,ROOT,TENSE,TV_SUBFEAT1,
                                TV_SUBFEAT2],S0,S):-
    [!
      get_subfeat(NP,NP_SUBFEAT),
      connects(S0,W,S)
    !],
    lex_tv(_,_,TENSE,TV_SUBFEAT1,TV_SUBFEAT2,ROOT,W),
    same(NP_SUBFEAT,TV_SUBFEAT1).
```

Fig. V.7.3 Cláusulas del Español

Las cláusulas anteriores pueden analizar oraciones que se encuentren dentro del dominio de la siguiente gramática:

```
S --> NP VP

NP --> PN
NP --> DET CN

VP --> TV NP
```

Fig. V.7.4 Fragmento de la gramática del sistema

```
lex_com_noun(apple,[singular,3,anim_n,hum_n,masc_n,
                    wing_n,loc_n,meal_y],manzana).

lex_det(an,[singular,3,_,_,masc_n,_,_,_],una).

lex_sp_det(una,[singular,3,_,_,masc_n,_,_,_]).

lex_pn(peter,[anim_y,hum_y,masc_y,wing_n,loc_n,
              meal_n],pedro).

lex_tv(cut,cut,past,[singular,3,_,hum_y,_,_,_,_],
                    [_,_,_,_,_,_,_,meal_y],
       cortar,corto).
```

Fig. V.7.5 Fragmento del lexicón del sistema

Dado que el proceso de análisis es esencialmente el mismo que en el caso del analizador (parser) del Inglès no se mostrarà el proceso correspondiente en Español.

Si se analizan con detalle los rasgos sintàcticos utilizados en el sistema desde el punto de vista del Español, resultarà sencillo entender el porque de la necesidad de ellos màs que en el caso del Inglès. Un ejemplo de ello son los verbos debido a que su conjugaciòn requiere del dominio del gènero, persona y nùmero. Ademàs de que la concordancia sintàctica entre la frase nominal (NP) y la frase verbal (VP) està controlada por los rasgos mismos.

V.8 EL LEXICON

El lexicón o diccionario del sistema consiste en elementos lèxicos que constan de 3 partes:

1) La versión inglesa como primer parámetro.
2) El conjunto de rasgos sintàctico-semànticos, a modo de lista como segundo parámetro.
3) La versión española como tercer y último parámetro.

Existen 3 excepciones:

1) Se tienen 3 conjuntos de adjetivos determinativos (determiners):
   - Uno que cumple con el formato establecido (lex_adj).
   - dos que contienen solamente alguna de las versiones (español o inglès) y la lista de rasgos sintàctico-semànticos (lex_eng_det y lex_sp_det).
2) Los verbos intransitivos y el auxiliar contienen:
   - El verbo inglès en modo infinitivo.
   - El verbo inglès en modo conjugado.
   - El tiempo o el modo (no incluido en los rasgos debido a que solamnente se utiliza esta caracteristica para la conjugación verbal).
   - El verbo español en modo infinitivo.
   - El verbo español en modo conjugado.
3) Los verbos transitivos contienen los mismos elementos que los verbos intransitivos, con la modalidad que en lugar de una lista de rasgos sintàctico-semànticos cuentan con dos listas de rasgos. Esto se debe a que la primer lista de rasgos se utiliza para verificar la compatibilidad sintàctico-semàntica entre la frase nominal (NP) y el verbo transitivo (TV), es decir entre el agente ejecutor de la acción y el verbo; y la segunda lista se utiliza para verificar la compatibilidad sintàctico-semàntica entre el verbo transitivo (TV) y la frase nominal (NP) contenida en la frase verbal (VP), es decir entre el verbo y el agente sobre el que se ejecuta la acción.

El lexicón es muy sencillo de analizar y actualizar debido a que los rasgos están claramente definidos y especificados. Para su actualización solamente se debe encontrar el lugar dentro de la clasificación en que debe ir el nuevo elemento, determinar sus rasgos e incluirlo con el formato dado.

El lexicón del sistema, por las razones que se expusieron al inicio de èste capitulo es muy pequeño, consiste únicamente en

- 15 adjetivos.
- 5 adverbios.

- 1 auxiliar.
- 57 nombres comunes.
- 1 conectivo.
- 2 adverbios de grado.
- 18 adjetivos determinativos.
- 10 verbos intransitivos.
- 40 nombres propios.
- 2 preposiciones.
- 13 pronombres personales.
- 2 cuantificativos.
- 1 relacional.
- 16 verbos transitivos.

A continuación se muestra un fragmento del lexicón implementado en el sistema

```
lex_com_noun(apple, [singular, 3, anim_n, hum_n, masc_n, wing_n,
                     loc_n, meal_y], manzana).

lex_det(an, [singular, 3, _, _, masc_y, _, _, _], un).
lex_det(an, [singular, 3, _, _, masc_n, _, _, _], una).

lex_eng_det(a, [singular, 3, _, _, _, _, _, _]).
lex_eng_det(an, [singular, 3, _, _, _, _, _, _]).

lex_sp_det(un, [singular, 3, _, _, masc_y, _, _, _]).
lex_sp_det(una, [singular, 3, _, _, masc_n, _, _, _]).

lex_pn(peter, [anim_y, hum_y, masc_y, wing_n, loc_n, meal_n], pedro).

lex_tv(cut, cut, past, [singular, 1, _, hum_y, _, _, _, _],
                       [_, _, _, _, _, _, _, meal_y], cortar, corte).
lex_tv(cut, cut, past, [singular, 2, _, hum_y, _, _, _, _],
                       [_, _, _, _, _, _, _, meal_y], cortar, cortaste).
lex_tv(cut, cut, past, [singular, 3, _, hum_y, _, _, _, _],
                       [_, _, _, _, _, _, _, meal_y], cortar, corto).
```

Fig. V.8.1 Fragmento del Lexicón del Sistema

A continuación se muestra la simbología y una serie de cuadros mostrando el lèxico implementado en el sistema:

Simbología:

```
[ number    {singular, plural}
  person    {1, 2, 3}
  animate   {y, n}
```

```
        human       {y, n}
        masculine   {y, n}    (-- gender
        wings       {y, n}
        location    {y, n}
        meal        {y, n}
    ]
```

@ =) tiene algùn valor del dominio correspondiente dependiendo del caso en que se encuentre.

_ =) hereda cualquier valor del dominio correspondiente dependiendo de la interpretaciòn sintàctico-semàntica.

ADJ = Adjective

--)	american	[@ 3 _ _ @ _ _ _]	americano
	bad	[@ 3 _ _ @ _ n _]	malo
	beautiful	[@ 3 _ _ @ _ _ _]	hermoso
	big	[@ 3 _ _ _ _ _ _]	grande
	black	[@ 3 _ _ @ _ n n]	negro
	good	[@ 3 _ _ @ _ n _]	bueno
	intelligent	[@ 3 y _ _ _ n n]	inteligente
	little	[@ 3 y _ _ _ n n]	pequeno
	lovely	[@ 3 _ _ _ _ _ _]	adorable
	old	[@ 3 _ _ @ _ _ _]	viejo
	pretty	[@ 3 _ _ @ _ _ _]	bonito
	red	[@ 3 n _ _ _ n n]	rojo
	small	[@ 3 n _ _ _ _ _]	chico
	tasty	[@ 3 _ n @ _ n y]	sabroso
	ugly	[@ 3 _ _ @ _ _ _]	feo

ADV = Adverb

--)	carefully	cuidadosamente
	sincerely	sinceramente
	slowly	lentamente
	well	bien
	gladly	alegremente

AUX = Auxiliary

--) be estar

IMPLEMENTACION

CN = Common Noun

--)	airplane	[	s	3	n	n	y	y	n	n	] avion
	apple	[	s	3	n	n	n	n	n	y	] manzana
	apples	[	p	3	n	n	n	n	n	y	] manzanas
	ball	[	s	3	n	n	n	n	n	n	] pelota
	balls	[	p	3	n	n	n	n	n	n	] pelotas
	bed	[	s	3	n	n	n	n	y	n	] cama
	bird	[	s	3	y	n	y	y	n	n	] ave, pajaro
	birds	[	p	3	y	n	y	y	n	n	] aves, pajaros
	boy	[	s	3	y	y	y	n	n	n	] nino
	boys	[	p	3	y	y	y	n	n	n	] ninos
	burger	[	s	3	n	n	n	n	n	y	] hamburgesa
	burgers	[	p	3	n	n	n	n	n	y	] hamburgesas
	butterfly	[	s	3	y	n	n	y	n	n	] mariposa
	butterflies	[	p	3	y	n	n	y	n	n	] mariposas
	car	[	s	3	n	n	y	n	n	n	] auto, coche
	cars	[	p	3	n	n	y	n	n	n	] autos, coches
	cat	[	s	3	y	n	y	n	n	n	] gato
	cats	[	p	3	y	n	y	n	n	n	] gatos
	coke	[	s	3	n	n	n	n	n	y	] coca_cola
	cream	[	_	3	n	n	n	n	n	y	] crema
	doll	[	s	3	n	n	n	n	n	n	] muneca
	dog	[	s	3	y	n	y	n	n	n	] perro
	dogs	[	p	3	y	n	y	n	n	n	] perros
	garden	[	s	3	n	n	y	n	y	n	] jardin
	girl	[	s	3	y	y	n	n	n	n	] nina
	girls	[	p	3	y	y	n	n	n	n	] ninas
	glass	[	s	3	n	n	y	n	n	n	] vidrio
	glasses	[	p	3	n	n	y	n	n	n	] vidrios
	hammer	[	s	3	n	n	y	n	n	n	] martillo
	house	[	s	3	n	n	n	n	y	n	] casa
	knife	[	s	3	n	n	y	n	n	n	] cuchillo
	library	[	s	3	n	n	n	n	y	n	] biblioteca
	man	[	s	3	y	y	y	n	n	n	] hombre
	men	[	p	3	y	y	y	n	n	n	] hombres
	milk_shake	[	s	3	n	n	n	n	n	y	] malteada
	park	[	s	3	n	n	y	n	y	n	] parque
	policeman	[	s	3	y	y	y	n	n	n	] policia
	policemen	[	p	3	y	y	y	n	n	n	] policias
	school	[	s	3	n	n	n	n	y	n	] escuela
	salad	[	_	3	n	n	n	n	n	y	] ensalada
	sofa	[	s	3	n	n	y	n	y	n	] sofa
	stone	[	s	3	n	n	n	n	n	n	] piedra
	student	[	s	3	y	y	y	n	n	n	] estudiante
	students	[	p	3	y	y	y	n	n	n	] estudiantes
	taco	[	s	3	n	n	y	n	n	y	] taco
	tacos	[	p	3	n	n	y	n	n	y	] tacos
	thief	[	s	3	y	y	y	n	n	n	] ladron
	toy	[	s	3	n	n	y	n	n	n	] juguete
	toys	[	p	3	n	n	y	n	n	n	] juguetes
	university	[	s	3	n	n	n	n	y	n	] universidad
	woman	[	s	3	y	y	n	n	n	n	] mujer

```
        women       [ p 3 y y n n n n ]  mujeres
        yard        [ s 3 n n y n y n ]  patio
```

CONN = Connective

```
    --) a     (admirar, atrapar, cazar, amar)
```

DEG = Degree Adverb

```
    --) so        tan
        very      muy
```

DET = Determiner

```
    --) a         [ s 3 _ _ _ _ _ _ ]  un (a)
        an        [ s 3 _ _ _ _ _ _ ]  un (a)
        all       [ s 3 _ _ _ _ _ _ ]  todo (a) (os) (as)
        every     [ s 3 _ _ _ _ _ _ ]  cada
        her       [ _ 3 _ _ _ _ _ _ ]  su (s)
        his       [ _ 3 _ _ _ _ _ _ ]  su (s)
        its       [ _ 3 _ _ _ _ _ _ ]  su (s)
        my        [ _ 3 _ _ _ _ _ _ ]  mi (s)
        our       [ _ 3 _ _ _ _ _ _ ]  nuestro (a) (os) (as)
        some      [ p 3 _ _ _ _ _ _ ]  algunos (as)
        that      [ s 3 _ _ _ _ _ _ ]  ese (a)
        the       [ _ 3 _ _ _ _ _ _ ]  el, la, los, las
        theirs    [ p 3 _ _ _ _ _ _ ]  sus
        these     [ p 3 _ _ _ _ _ _ ]  estos (as)
        this      [ s 3 _ _ _ _ _ _ ]  este (a)
        those     [ p 3 _ _ _ _ _ _ ]  aquellos (as)
        your      [ s 3 _ _ _ _ _ _ ]  tu
        yours     [ p 3 _ _ _ _ _ _ ]  tus,
                                       vuestro (a) (os) (as)
```

IV = Intransitive Verb

```
    --) disappear [ @ @ _ _ _ _ _ _ ]  desaparecer
        fly       [ @ @ @ @ _ y n @ ]  volar
        live      [ @ @ y _ _ _ n _ ]  vivir
        play      [ @ @ y _ _ _ n n ]  jugar
        run       [ @ @ y _ _ n n n ]  correr
        sing      [ @ @ y @ _ @ n _ ]  cantar
        sleep     [ @ @ y _ _ _ _ n ]  dormir
        smoke     [ @ @ y y _ n n n ]  fumar
```

```
          speak      [ @ @ y y _ n n n ]  hablar
          study      [ @ @ y y _ n n n ]  estudiar


PN   = Proper Noun

     --) alex       [ s 3 y y y n n n ]  alejando
         charles    [ s 3 y y y n n n ]  carlos
         edward     [ s 3 y y y n n n ]  eduardo
         eduard     [ s 3 y y y n n n ]  lalo
         eugene     [ s 3 y y y n n n ]  eugenio
         fred       [ s 3 y y y n n n ]  federico
         gerard     [ s 3 y y y n n n ]  gerardo
         jim        [ s 3 y y y n n n ]  jim
         john       [ s 3 y y y n n n ]  juan
         joseph     [ s 3 y y y n n n ]  jose
         billy      [ s 3 y y y n n n ]  memo
         pete       [ s 4 y y y n n n ]  pedro
         peter      [ s 4 y y y n n n ]  pedro
         sam        [ s 3 y y y n n n ]  samuel
         william    [ s 3 y y y n n n ]  guillermo

         alice      [ s 3 y y n n n n ]  alicia
         carmen     [ s 3 y y n n n n ]  carmen
         chris      [ s 3 y y n n n n ]  cristina
         guadalupe  [ s 3 y y n n n n ]  guadalupe
         laura      [ s 3 y y n n n n ]  laura
         lucy       [ s 3 y y n n n n ]  lucia
         lupe       [ s 3 y y n n n n ]  lupe
         mary       [ s 3 y y n n n n ]  maria
         norma      [ s 3 y y n n n n ]  norma
         paty       [ s 3 y y n n n n ]  patricia
         priss      [ s 3 y y n n n n ]  priscila
         rose       [ s 3 y y n n n n ]  rosa
         susan      [ s 3 y y n n n n ]  susana

         fido       [ s 3 y n y n n n ]  fido
         lassie     [ s 3 y n n n n n ]  lassie
         neron      [ s 3 y n y n n n ]  neron
         piolin     [ s 3 y n y y n n ]  piolin
         silvester  [ s 3 y n y n n n ]  silvestre

         california [ s 3 n n _ _ y n ]  california
         cancun     [ s 3 n n _ _ y n ]  cancun
         cuernavaca [ s 3 n n _ _ y n ]  cuernavaca
         london     [ s 3 n n _ _ y n ]  londres
         madrid     [ s 3 n n _ _ y n ]  madrid
         mexico     [ s 3 n n _ _ y n ]  mexico
         new_york   [ s 3 n n _ _ y n ]  vueva_york
```

PREP = Preposition

```
--) in     [ _ _ _ _ _ _ y _ ]  en
    with   [ @ @ _ _ _ _ _ _ ]  con, conmigo, contigo
```

PRON = Pronoun

```
--) i      [ s 1 y y _ n n n ]  yo
    me     [ s 1 y y _ n n n ]  conmigo
    you    [ s 2 y y _ n n n ]  tu, contigo
    he     [ s 3 y y y n n n ]  el
    him    [ s 3 y y y n n n ]  el
    she    [ s 3 y y n n n n ]  ella
    her    [ s 3 y y n n n n ]  ella
    it     [ s 3 _ _ _ _ _ _ ]  ello
    we     [ p 1 y y @ n n n ]  nosotros
    us     [ p 1 y y @ n n n ]  nosotros
    you    [ p 2 y y @ n n n ]  vosotros
    they   [ p 3 y y @ n n n ]  ellos
    them   [ p 3 y y @ n n n ]  ellos
```

Q = Quantifier

```
--) many   [ p 3 _ _ @ _ _ _ ]  muchos
    few    [ p 3 _ _ @ _ _ _ ]  algunos, pocos
```

REL = Relational

```
--) that    que
```

TV = Transitive Verb

```
--) admire [ @ @ _ y _ _ _ _ ]
           [ _ _ _ _ _ _ _ _ ]  admirar    (-- a
    adopt  [ @ @ _ y _ _ _ _ ]
           [ _ _ y _ _ _ _ _ ]  adoptar    (-- a
    break  [ @ @ y _ _ _ _ _ ]
           [ _ _ n _ _ _ n n ]  romper
    buy    [ @ @ _ y _ _ _ _ ]
           [ _ _ _ n _ _ _ _ ]  comprar
    catch  [ @ @ y _ _ n _ _ ]
           [ _ _ _ _ _ _ n _ ]  atrapar    (-- a
    chase  [ @ @ y _ _ _ _ _ ]
           [ _ _ y n _ _ _ _ ]  cazar      (-- a
    clean  [ @ @ _ y _ _ _ _ ]
           [ _ _ n _ _ _ _ _ ]  limpiar
```

```
cut     [ @ @ _ y _ _ _ _ ]
        [ _ _ _ _ _ _ _ y ] cortar
defend  [ @ @ y _ _ n _ _ ]
        [ _ _ y _ _ _ _ _ ] defender  (-- a
eat     [ @ @ y _ _ n _ _ ]
        [ _ _ _ _ _ _ _ y ] comer
find    [ @ @ y _ _ n _ _ ]
        [ _ _ _ _ _ _ _ _ ] encontrar (-- a
hear    [ @ @ y _ _ _ _ _ ]
        [ _ _ _ _ _ _ n n ] oir       (-- a
listen  [ @ @ _ y _ _ _ _ ]
        [ @ @ _ y _ _ _ _ ] escuchar  (-- a
love    [ @ @ _ y _ _ _ _ ]
        [ _ _ _ _ _ _ _ _ ] amar      (-- a
see     [ @ @ y _ _ _ _ _ ]
        [ _ _ _ _ _ _ _ _ ] ver       (-- a
study   [ @ @ _ y _ _ _ _ ]
        [ _ _ _ n _ _ n n ] estudiar
```

Fig. V.8.2 Lexicòn del Sistema

V.9 CONSIDERACIONES DE LA IMPLEMENTACION

El sistema està basado originalmente en el formalismo de las Gramàticas de Clàusula Definida (DCG) de la manera descrita por Pereira y Warren en [PER80], en el formalismo de las Gramàticas de Estructura de Frase Generalizada (GPSG) descrito por Gazdar et al. en [GAZ85], y en el sistema de traducciòn automàtica desarrollado por Jimenez [JIM86].

Durante el estudio de los formalismos concluì que si las bases para el desarrollo del sistema se tomaban ùnicamente de las fuentes citadas, no se podria realizar el anàlisis semàntico a nivel de determinaciòn de formaciones congruentes con el mundo real, por lo que tendria entonces que desarrollar el sistema sin considerar la malformaciòn semàntica. Como tal tipo de anàlisis es uno de los objetivos primordiales del sistema, me vì en la necesidad de tomar fuentes de documentaciòn que me permitieran adquirir los elementos y conocimientos necesarios para formular una manera de superar esta limitante.

De las teorías estudiadas como base de fundamentación teórica sobre gramática, sintaxis y semántica, las teorías de Katz [KAT79] y de Leech [LEE85] (además del antecedente adquirido en 'Theory of Aspects of Syntax' de Chomsky [CHO76]) son las que me proporcionaron los elementos teóricos que me permitieron resolver el problema de congruencia semántica mediante la formulación de rasgos semánticos (ver. cap. II).

La generación, control y manejo de los rasgos semánticos de acuerdo a las formulaciones de Leech y Katz es fácilmente implementable en Prolog dado la facilidad de representación de listas que ofrece el lenguaje; además de adecuarse perfectamente a los formalismos utilizados.

Las condiciones extra son utilizadas de distinta forma a la propuesta por Pereira y Warren [PER81] (utilizada por Jimenez [JIM86]). En ésta implementación, las condiciones extra son colocadas dentro de las reglas de traducción en el lugar que resulta más conveniente primordialmente para la buena interpretación semántica; la interpretación sintáctica se lleva a cabo en base a la gramática y la compatibilidad de rasgos sintácticos.

Los elementos de compatibilidad semántica (en raras ocasiones sintáctica) son transferidos entre las reglas de traducción mediante un parámetro que, en caso de existir, es el primero de la lista. Por ejemplo, todas las reglas de traducción que analizan las frases verbales (VP) contienen dicho parámetro a fin de poder determinar la compatibilidad semántica de ésta con la frase nominal.

El análisis de las frases preposicionales en una de las partes más delicadas del sistema, desde el punto de vista de que debe llevarse un control muy detallado de la compatibilidad entre las frases nominales de oración y el verbo involucrado.

En la gramática del español, la discriminación para el uso del conectivo 'a' se hace únicamente en base al verbo involucrado y no en base a la parte derecha de la frase verbal (VP).

Los verbos transitivos contienen dos listas de rasgos sintáctico-semánticos, a diferencia de los demás elementos léxicos que contienen una sola lista. El propósito de éstas es que la primer lista permita determinar la compatibilidad sintáctico-semántica entre el ejecutor de la acción y el verbo, y la segunda lista permita determinar la compatibilidad entre el verbo y el receptor de la acción.

En algunas de las conjugaciones verbales en el lexicòn se dan dos especificaciones para la 3a. persona del singular y del plural; esto es porque el verbo es vàlido para frases nominales distintas, cuyos rasgos impiden su especificaciòn unitaria dentro del lexicòn.

Dado que los rasgos sintàcticos y semànticos se encuentran especificados en el lexicòn, la extensiòn del mismo requiere exclusivamente de la inclusiòn del nuevo elemento con sus rasgos correspondientes. De igual forma, para incluir en el sistema la consideraciòn de nuevos rasgos semànticos, basta con incluir èstos en el lexicòn. En ambos casos no se requiere modificar el còdigo correspondiente a los analizadores (parsers) de cualquiera de los idiomas ni el traductor; salvo contadas excepciones en que los rasgos adicionados requieran del control de casos especiales dentro del los analizadores.

Para evitar que se requiera en un momento dado de la modificaciòn de còdigo en los analizadores (parsers) y/o en el traductor de sistema cuando se agregen nuevos elementos lèxicos, se puede buscar una clasificaciòn adicional de elementos sintàcticos y/o semànticos que se incluyan en las listas existentes y que eviten la necesidad de modificaciòn de còdigo para el funcionamiento correcto del sistema.

V.10 SUMARIO

Se mencionò cual es la estructura general del sistema en base a las etapas que lo comprenden dentro del proceso de traducciòn de lenguaje natural, con sus caracteristicas.

Dado que el anàlisis sintàctico-semàntico es de vital importancia en el sistema, se mencionò el criterio de selecciòn de rasgos, su justificaciòn y su representaciòn dentro del còdigo del sistema.

Se describieron cada uno de los mòdulos que comprenden el sistema: el mòdulo principal, los analizadores inglès y español, el traductor y el lexicòn. Describiendo ademàs su operatividad.

Por ùltimo se hacen comentarios acerca de las consideraciones especiales de la implementaciòn del sistema.

CONCLUSIONES

En èste trabajo se presentò un sistema de traducción a nivel sintàctico-semàntico bidireccional entre los idiomas Español e Inglès, desarrollado en base al formalismo de las Gramàticas de Clàusula Definida (DCGs), al formalismo de las Gramàticas de Estructura de Frase Generalizada (GPSGs) y a las teorias semànticas de Leech y de Katz. Inicialmente no se consideró la inclusión de las teorias semànticas para el desarrollo del sistema, pero sin esas bases teòricas habria sido imposible llevar a cabo el anàlisis semàntico de las oraciones en cualquiera de los idiomas involucrados.

El Prolog y los formalismos de las DCGs y GPSGs permitieron que el diseño del sistema tuviera las siguientes ventajas:

- Modularidad: Cada parte del sistema: el módulo principal, los módulos de anàlisis del Español y del Inglès, el módulo de traducción y el módulo del lexicón, son totalmente independientes; por lo que el ambiente de desarrollo y actualización es totalmente flexible. Además, cada módulo està a su vez perfectamente modularizado, de tal manera que su diseño, actualización y anàlisis es muy sencillo.

- Bidireccionalidad: Una de las ventajas del Prolog es la bidireccionalidad en la instanciación de sus variables, gracias a ello, la traducción en cualquier sentido es automàtica y no se requiere de módulos adicionales para que se lleve a cabo. Esto implica que los módulos de 'parsing' del Inglès y del Español sirven tanto para el anàlisis como para la síntesis de las oraciones y el módulo de traducción es uno solo.

La viabilidad de las Gramàticas de Clàusula Definida y las Gramàticas de Estructura de Frase Generalizada como formalismos para el desarrollo de sistemas de traducción automàtica queda demostrada al observarse que el anàlisis sintàctico-semàntico del sistema cumple con la orientación que se pretende implementar en la base de conocimientos y las clàusulas, asi como la manera en que la semàntica de las

DCGs y GPSGs es manejada de manera prácticamente transparente por el Prolog. Además las oraciones obtenidas son congruentes con el mundo real.

Es importante señalar que en realidad las GPSGs no mejoran en mucho el sistema de traducción, por lo que se puede concluir también que con las bases de las DCGs habría sido suficiente para elaborar el traductor en Prolog sin afectar los resultados obtenidos. De hecho, las DCGs resultan ser una extensión práctica de las bases teóricas de las GPSGs. El único aspecto importante de las GPSGs que no se implemento en el sistema fué el uso de metareglas, tal ves su implementación le habría dado un giro a la conclusión referente a ellas.

El sistema adolece aún de muchas deficiencias, pero no se pretendió en ningún momento obtener la mejor y más completa versión del sistema.

En el diseño del sistema se manejan algunos casos especiales, como por ejemplo el control de las frases preposicionales, que pueden dificultar la extensión del sistema. Una alternativa de solución a este problema puede ser la generación de ciertos rasgos semánticos especiales enfocados al control de las preposiciones, a fin de eliminar la necesidad de rutinas de control de casos especiales que pueden crecer en cierta proporción con el crecimiento del lexicón y de la ampliación de la gramática.

Considero que un futuro desarrollo puede ser utilizar los elementos aquí utilizados para intentar obtener un sistema que considere en lugar de oraciones, frases completas y detecte el contexto a fin de poder obtener un traductor más sofisticado, o bién un sistema de pregunta-respuesta.

Por último creo conveniente indicar que las bases de conocimiento personal son en el area de computación y no de lingüística, por lo que una parte considerable del tiempo dedicado al desarrollo de éste trabajo consistió en conseguir e investigar tópicos relacionados con la linguistica, además de tener que seleccionar el material que considerara apropiado para el enfoque deseado, por lo que con seguridad el trabajo debe adolecer de muchas deficiencias en esa area, pero considero (y espero) que el trabajo puede servir como incentivo para que personas de ambos enfoques se inquieten y lleven a cabo conjuntamente futuros desarrollos que superen los bemóles de ésta tesis.

G L O S A R I O

Actuación.- La conducta lingüística factual, el uso real del lenguaje en situaciones concretas, que no puede, sin embargo, entenderse como reflejo directo de la competencia.

Adjetivo.- Es la parte de la oración que completa ó precisa la significación del sustantivo al cual se refiere.

Adjetivo: Calificativo		(viejo, grande)
Determinativo:	Numerales	(uno, tercero)
	Posesivos	(mi, tu)
	Indefinidos	(este, aquel)

Adverbio.- Son las palabras que modifican el verbo o el adjetivo ó a otro adverbio, determinando circunstancias de lugar, modo, tiempo, cantidad, que aclaran su significado, o denotando el carácter afirmativo, negativo ó dubitativo de la oración.
(aquí, ahí, donde, cuando, como)

Afijación.- Proceso morfológico, sintáctico y semántico. Adición de afijos a las raíces de las palabras para desarrollar formas de flexión o de formación de palabras por medio de la derivación.

Afijo.- Partícula que se pone al principio o al final de las palabras para modificar su significado (prefijo, sufijo, infijo).

Ahormante.- Ahormacional = Phrase Structure.

Ambiguedad.- Indeterminación gramatical, sintáctica, semántica.

Analiticidad.- Según Kant, es la aserción vacía que se origina cuando el significado de un predicado contiene solo atributos que son componentes del significado del sujeto.

Anáfora.- En oposición a catáfora, la recurrencia hacia atrás en un texto.

Anomalía.- Irregularidad.
Una oración es anómala cuando presenta divergencias frente a las reglas de la lengua. Para las anomalías gramaticales se emplea preferentemente los términos agramaticalidad y de grados de gramaticalidad, y se reserva el de anomalía para designar la desviación semántica.

GLOSARIO

Antonimia.- Oposición de dos voces diferentes.
Oposición de palabras.
Calidad de antónimo.

Artículo.- Circunscribe la extensión en que ha de tomarse el pronombre al cual se antepone.

Auxiliar.- Constituyentes de los que se derivan los elementos de los tiempos simples y compuestos y de los modos.

Caso.- Función que desempeñan los sustantivos, adjetivos y pronombres.

Competencia.- Capacidad lingüística del hablante de la lengua materna, su conocimiento interior, inconsciente de su lengua. Los conocimientos del hablante ideal.

Comunicación.- Es el intercambio verbal entre un hablante que produce un enunciado destinado a otro hablante, a un interlocutor del que solicita la escucha y/o una respuesta explícita o implícita.

Conjunción.- Parte invariable de la oración que tiene la función de unir dos o más palabras u oraciones.
(o, u, y, pero, excepto, luego, porque).

Constituyente.- Elementos, componentes de formas complejas.
Construcciones que funcionan como unidades en construcciones superiores.

Constructo.- Técnica que designa un objeto no observado que se supone para la explicación de los datos, hechos y observaciones; los constructos forman el plano genotípico de la descripción lingüística.

Contradicción.- Enunciado necesariamente falso.

Cópula.- Son las formas finitas de ser, estar parecer y a veces quedar, que unen el sujeto de la frase con el predicado nominal.

Declinación.- Serie ordenada de los casos gramaticales.

Derivación.- Proceso de generación de la frase, esto es, de la sucesiva aplicación de reglas de reescritura de una gramática generativa desde la cadena inicial a través de cadenas intermedias hasta la cadena terminal.

Entonación.- Línea que determina la serie de sonidos que constituyen una palabra, frase u oración.

GLOSARIO

Entrañe.- Si en una oración dada se interpreta un hecho (estado de las cosas) descrito como afirmado y otro hecho es deducible de èste, entonces èste està entrañado.

Estructura Sintagmàtica.- La relación bàsica de la linealidad en el sentido de contiguidad asociativa no conmutativa, análisis de constituyentes

Expresión Idiomàtica.- Creación lingüística peculiar de una lengua o combinación de palabras.
Unidad fraseológica cuyo significado total no se deduce de los diferentes significados lexicos que la componen.

Fonètica.- Parte de la lingüistica que se ocupa de describir los sonidos de una lengua o de estudiar la evolución de los mismos.

Fonología.- Ciencia que estudia la relación de los fonèmas con su significación lògica o afectiva.

Gènero.- El gènero de los sustantivos proviene de la concordancia necesaria que deben guardar con una u otra de las terminaciones de los adjetivos.
(masculino, femenino, neutro)
Gènero epiceno: liebre, rinoceronte, àguila.

Gramàtica.- Es un conjunto de reglas que discriminan entre el uso educado y otras formas de uso de la palabra.

Es una clasificación de señales que diferencian una oración de otra.

Es un número finito de patrones de oraciones, cada patrón conteniendo una o màs ranuras en las que una clase correspondiente de unidades lèxicas pueda reemplazar una a la otra.

Es una descripción de las relaciones probables entre unidades lèxicas en secuencia.

Es una descripción de los grados de agrupamiento de las unidades lèxicas en secuencia.

Es un conjunto de reglas de estructura de frase para la derivación de oraciones declarativas activas simples combinadas con un conjunto de reglas transformacionales que, cuando se aplican a las oraciones derivadas por las reglas de estructura de frase, agregan, sustraen o modifican el orden dentro de ellas, o se combinan en formas complejas.

GLOSARIO

Homógrafo.- Aplícase a las palbras de distinta significación que se escriben de igual manera

Homónimo.- Dícese de dos o mas personas o cosas que llevan el mismo nombre.

Idiotismo.- Toda construcción que pertenece a una lengua determinada y que no posee ningún correspondiente sintáctico en otra lengua (por si las moscas).

Imperativo.- Modo de expresión de ejecución de acciones.

Inconsistencia.- Cuando dos oraciones no son ni verdaderas ni falsas, por contradecirse una a la otra.

Inflexión.- Cambio realizado en la forma de una palabra para señalar su función sintáctica en la oración, generalmente se señala por afijación.

Lenguaje.- Conjunto de sonidos articulados con los que el hombre expresa lo que piensa o siente.

Lexema.- Unidad léxica, "palabra".
Elemento del vocabulario que puede estar compuesto de uno o más morfemas libres.
Unidad del lexicón con un significado lexicográfico relativamente independiente.

Lexicalización.- La transformación de un elemento a una unión de elementos en un elemento léxico o conceptual único y estable de la lengua, p.ej. ferrocarril.

Léxico.- Diccionario de una lengua o región (lexicón).
La totalidad de las palabras o el vocabulario de una lengua.

Modelo.- Representación idealizante, esquemática y simplificada de un objeto complejo o un conjunto de objetos, especialmente en cuanto a sus propiedades estructurales y funcionales, así como su comportamiento.

Morfema.- Elemento lingüístico cuya función es relacionar los semantémas de la oración o fijar y delimitar la función y significado de las voces.

Unidad lingüística recurrente mínima que tiene un significado y no puede ser analizada en unidades significativas recurrentes menores.

Elemento terminal de la estructura profúnda sintáctica que se define por la derivación sintáctica del término inicial oración.

Modal.- Clase de unidades lingüísticas invariables que modifican no el contenido del verbo, sino la concepción de la oración.

Modo.- Categoría gramatical asociada en general al verbo y que traduce:
1) El tipo de comunicación instituido por el locutor entre él y su interlocutor.
2) La actitud del hablante con respecto a sus propios enunciados.

Morfología.- El estudio de las formas, de las alteraciones formales de las palabras, de las formas de flexión y de las clases de palabras, que se encuentra estrechamente ligado con la sintáxis.

Nombre.- (Noun), Expresión que designa un objeto.

Nominal.- (Complejo (Fn); Noun Phrase):
Asume en las oraciones la función del nombre. Es el nudo por el que se ponen en relación el adjetivo y el nombre.

Norma.- La norma lingüística contiene las diferencias usuales, normales, tradicionales, no necesariamente funcionales del sistema.

Número.- Es el accidente gramatical de los nombres por el cual adoptan forma diversa según signifiquen ó se refieran a uno o más objetos.
(singular, plural)

Parafrasis.- Interpretación amplificativa de un texto.

Paralogismo.- Razonamiento falso, pero sin intención.

Polisémico.- Aplícase a palabras con pluralidad de significaciones

Pragmático.- Que funda las Teorías en el estudio de los textos.

Pragmatismo.- Doctrina filosófica según la cual nuestro conocimiento de las cosas solo consiste en sus efectos o en el uso que de ellas podamos hacer.

Prefijo.- Partícula antepuesta a ciertas palabras para modificar su significado.

Preposición.- Partícula que subordina un elemento sintáctico a otro.
(a, ante, bajo, con, contra, de, desde)

GLOSARIO

Presuposición.- Las suposiciones que hace implicitamente un hablante con una manifestación se denominan supuestos de la oración o presuposiciones. Las presuposiciones fijan, además, què condiciones hay para el uso correcto de una oración dentro de un conjunto lingüistico.

Pronombre Personal.- Son las palabras con que se designa a las personas gramaticales.
(yo, tu, el)

Pronombre Posesivo.- Indican que la posesión de un objeto corresponde a determinada persona gramatical.
(mio, tuyo)

Pronombre Demostrativo.- Son los que denotan el lugar donde se encuentra un objeto respecto de una de las personas gramaticales.
(este, ese, aquel)

Quid.- Razòn, punto principal de una cosa.

Rasgo.- Constructo conceptual, concepto imprescindible para la correcta comprensión de la estructuración del lenguaje.

Rasgo Semàntico.- Atomo de significado, elemento del concepto o del contenido, que considerado como (micro)estructurado en si, es el elemento de base y constructo de una teoria semàntica.

Rasgo Sintàctico.- Rasgo que especifica la conducta sintàctica de las unidades lèxicas.

Recursividad.- La posibilidad de utilizar repetidamente reglas en un apartado de reglas.

Redundancia.- Repetitividad del habla, presentación repetida o múltiple, explicita o implicita del mismo contenido o hecho.

Reflexividad.- Propiedad de una clase verbal o de una construcción sintàctica en la que sujeto y complemento son idènticos referencialmente, al dirigirse al sujeto autor.

Regla de Reescritura.- Una regla de reescritura tiene la forma:

$$A \dashrightarrow Z\,/\,X+@$$

donde: X, @ son cadenas de simbolos (posiblemente vacias) y Z es una cadena no vacia.

GLOSARIO

Semantema.- Parte del vocablo (raiz o radical) que encierra su significado (p.ej. blanc-, en: blanco, blancura).

Semàntica.- Disciplina que se ocupa del estudio de la significaciòn de las palabras o vocablos.

Semàntica Filosòfica.- Disciplina que analiza las relaciones del signo con la realidad, es decir, en què condiciones se aplica un signo a un objeto, que normas pueden y deben observarse para asegurar una significaciòn verdadera, etc..

Semàntica General.- Disciplina que estudia el porquè, el còmo y el para què del actò de la comunicaciòn, a base de los signos utilizados.

Semàntica Linguistica.- Toma el lenguaje, la lengua y el habla, no como debieran ser o se quisiera que fuesen, sino como son, para establecer p.ej.: què es una palabra, què relaciones median entre la forma y el sentido de una palabra, porquè cambian las palabras de significaciòn, cual es el mecanismo de sus cambios, etc..
De acuerdo con esto, la semàntica lingüistica parte de varios supuestos aportados por la lingüistica general:
1) El contenido del signo lingüistico (esto es, el contenido de una palabra, vocablo, tèrmino o dicciòn) no es de orden necesario, sino arbitrario y convencional.
2) Estos significados no son estàticos e inmutables, sino dinàmicos y variables.
3) Estas variaciones no pueden reducirse a leyes, como las fìdicas, o como las correspondencias fonètico-fonològicas, pero sus causas y modos pueden, sì, organizarse y clasificarse con criterio cientifico.

Semiòtica.- Teoria general de los signos y de los lenguajes, abarca la pragmàtica, la semàntica y la sintaxis.

Significado Gramatical.- de una palabra, frase o texto es lo que se expone y se dice, lo que se convierte en lenguaje en el proceso de comunicaciòn.

Sintagma.- Grupo o encadenamiento de palabras.

GLOSARIO

Sintaxis.- Parte de la gramática que estudia la ordenación de las palabras y su relación en la oración, y la relación de unas oraciones con otras.

Sinónimo.- Aplícase a dos vocablos que tienen una significación completamente idéntica o muy parecida (gusto, placer)

Sinonimia.- Circunstancia de ser sinónimos dos o más vocablos.

Subcategorización.- Subclasificación de las categorías lexicas en subclases para poder describir la conducta sintáctica de las correspondientes unidades léxicas.

Sufijo.- Partículas que se añaden a los radicales de algunas palabras variando su significado.

Sustantivo.- Palabra con que se designan los objetos. (hombre, piedra, lobo)

Tautología.- Enunciado que ha de ser correcto en virtud de su mismo significado.

Taxonómico.- Propiedad de la gramática que se limita a la segmentación y clasificación del habla y por eso sólo es capáz de dar cuenta de la estructura superficial sintáctica.

Verbo.- Es una parte del discurso que no tiene flexión casual, sino que se conjuga según tiempo, persona y número y designa una actividad o bién el padecer de una actividad.

Voz.- Palabra, vocablo.
La voz es una categoría gramatical asociada al verbo y a su auxiliar, que indica la relación gramatical entre el verbo, el sujeto o el agente y el objeto, cada voz se manifiesta por flexiones verbales específicas (desinencias o prefijos, formas diferentes de los auxiliares, etc.).

REFERENCIAS

CAPITULO I:

[BAR81] Barr, Avron. and Feigenbaum, Edward A. The Handbook of Artificial Intelligence Vol.1. Addison Wesley. 1981, USA.

[BEN85] Bennet, Winfield S. and Slocum, Jonathan. The LRC Machine Translation System. pp.111-121. American Journal of Computacional Linguistics. 11(2-3) Apr-Sep 1985, USA.

[BIE85] Biewer, Axel., Feneyrol, Christian., Ritzke, Johannes. and Stegentritt, Erwin. ASCOF - A Modular Multilevel System for French-German Translation.

[CHA77] Charniak, Eugene. Inference and Knowledge in Language Comprehension. pp. 541-574. Machine Intelligence 8 (Elcock, E.W., Michie, Donald). Ellis Horwood Ltd. 1977, Great Britain.

[CHA86] Charniak, Eugene. and McDermott, Drew. Introduction to Artificial Intelligence. Addison-Wesley. 1986, USA.

[CHO62] Chomsky, Noam. Syntactic Structures. Mouton & Co. 1962, Netherlands.

[CHO76] Chomsky, Noam. Aspectos de la Teoria de la Sintàxis. Aguilar. 1976, España.

[CUL86] Cullingford, Richard. SAM. pp.627-649. Readings in Natural Language Processing; edited by Grosz, Sparck Jones y Webber. Morgan Kaufmann Publishers Inc. 1986, USA.

[DAH81] Dahl, Verònica. Translating Spanish Into Logic Through Logic. pp. 149-164. American Journal of Computational Linguistic. 7(3) Jul-Sept 1981. USA.

[DAH83] Dahl, Verònica. Treating Coordination in Logic Grammars. pp. 69-91. American Journal of Computational Linguistics. 9(2) Apr-Jun 1983. USA.

[FAS83] Fass, Dan and Wills Yorick. Preference Semantics, Ill-Formedness, and Metaphor. pp.178-187. American Journal of Computational Linguistics. 9(3-4) Jul-Dec 1983, USA.

[GAZ85] Gazdar, G., Klein, E., Pullum, G.K. and Sag, I.A. Generalized Phrase Structure Grammar. Basil Blackwell Publ. Ltd. 1985, Oxford.

[GRA83] Granger, Richard H. The NOMAD System: Expectation-Based Detection and Correction of Errors during Understanding of Syntactically and Semantically Ill-Formed Text. pp.188-196. American Journal of Computational Linguistics. 9(3-4) Jul-Dec 1983, USA.

[GRE86] Green, Bert F.Jr., Wolf, Alice K., Chomsky, Carol. BASEBALL: An Automatic Question Answerer. pp.545-549. Readings in Natural Language Processing; Edited by Grosz, Sparck Jones & Webber. Morgan Kaufmann Publishers Inc. 1986, USA.

[GRI71] Gries, David. Compiler Construction for Digital Computers. John Wiley & Sons Inc. 1976, USA.

[GRO86] Grosz, Barbara J., Sparck, Karen Jones and Lynn Bonnie Webber. Readings in Natural Language Processing. Morgan Kaufmann. 1986, USA.

[ISA71] Isard, S. and Longuet-Higgins, L. Question-answering in English. pp. 243-254. Machine Intelligence 6 (Meltzer, Bernard., Michie, Donald). Edinburgh University Press. 1971, Great Britain.

[JIM86] Jiménez, Alejandro. Low-Quality Fully Automatic Machine Translation by Syntax Only. M.Sc. Dissertation. University of Essex. 1986, England.

[JOH85] Johnson, Rod. EUROTRA: A Multilingual System under Development. pp. 155-169. American Journal of Computational Linguistics. 11(2-3) Apr-Sept 1985. USA.

[JOS82] Joshi, Aravind K. Phrase Structure Trees Bear More Fruit than You Would Have Thought. pp. 1-11. American Journal of Computational Linguistics. 8(1) Jan-Mar 1982. USA.

[KAT79] Katz, Jerrold J. Teoría Semántica. Aguilar. 1979, Madrid.

[KIM85] Kimbrell, Roy E. English Recognition. pp. 125-140. Byte 10(13) December 1985, USA.

REFERENCIAS

[LEE85] Leech, Geoffrey. Semàntica. Colecciòn AU 187. Alianza Editorial. 1985, Madrid.

[LEW78] Lewis, II P.M., Rosenkrantz, D.J. and Stearns, R.E., Compiler Design Theory. The Systems Programming Series. Addison Wesley. 1978, USA.

[MAC85] Macias, Benjamin. Lenguaje Natural por Computadora. pp. 123-130. Ciencia y Desarrollo 6(65) Nov/Dic 1985, Mèxico.

[MAC87] Macias, Benjamin. Investigaciòn en Traductores de Lenguaje Natural por Computadora., IV Reuniòn Nacional de Inteligencia Artificial- memorias. pp. 54-58. 1987, Puebla.

[MAI83] Main, Michael G. and Benson, David B. Denotational Semantics for "Natural" Language Question-Answering Programs. pp. 11-21. American Journal of Computational Linguistics. 9(1) Jan-Mar 1983, USA.

[McC80] McCord, Michael C. Slot Grammars. pp.31-43. American Journal of Computational Linguistics. 6(1) Jan-Mar 1980, USA.

[NAG85] Nagao, Makoto., Tsujii, Jun-ichi. and Nakamura, Jun-ichi. The Japanese Government Proyect for Machine Translation. pp. 91-110. American Journal of Computational Linguistics. 11(2-3) Apr-Sept 1985, USA.

[PER80] Pereira, F.C.N. and Warren, D.H.D. Definite Clause Grammars for Language Anlysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks. Artificial Intelligence, 13. pp. 231-278. North Holland Publ. Co. 1980.

[PER81] Pereira, Fernando. Extraposition Grammars. pp. 243-256. American Journal of Computational Linguistics. 7(4) Oct-Dec 1981. USA.

[POL86] Pollack, Jordan. and Waltz, David L. Interpretation of Natural Language. pp. 189-198. Byte 11(2) February 1986, USA.

[RAL76] Ralston, Anthony. Encyclopedia of Computer Science. Van Nostrand Reinhold. 1976, USA.

[RIC85] Rich, Elaine. Artificial Intelligence. McGraw-Hill. 1985, Singapore.

REFERENCIAS

[ROB86] Robinson, Jane J. DIAGRAM: A Grammar for Dialogues. pp.139-159. Readings in Natural LAnguage Processing; edited by Grosz, Sparck Jones and Webber. Morgan Kaufmann Publishers Inc. 1986, USA.

[SAN71] Sandewall, E. Representing Natural Language Information In Predicate Calculus. pp. 255-277. Machine Intelligence 6 (Meltzer, Bernard., Michie, Donald). Edinburgh University Press. 1971, Great Britain.

[SHA82] Shapiro, Stuart C. Generalized Augmented Transition Network Grammars for Generation From Semantic Networks. pp. 12-25. American Journal of Computational Linguistics. 8(1) Jan-Mar 1982, USA.

[SIN82] Singh, Jagjit. Teoria de la Informaciòn el Lenguaje y la Cibernètica. Alianza Editorial. 1982, Madrid.

[STA87] Stabler, Edward P. Jr. Restricting Logic Grammars With Government-Binding Theory. pp. 1-10. American Journal of Computational Linguistics. 13(1-2) Jan-Jun 1987, USA.

[TOM87] Tomita, Masaru. An Efficient Augmented-Context-Free Parsing Algorithm. pp.31-46. American Journal of Computational Linguistics. 13(1-2) Jan-Jun 1987. USA.

[TRE85] Tremblay, Jean-Paul. and Sorenson, Paul G. The Theory and Practice of Compiler Writing. McGraw-Hill. 1985, Singapore.

[VAS85] Vasconcellos, Muriel. and Leòn, Marjorie. SPANAM and ENGSPAN: Machine Translation at the Pan American Health Organitation. pp. 122-136. American Journal of Computational Linguistics. 11(2-3) Apr-Sept 1985, USA.

[WAR82] Warren, David H.D. and Pereira, Fernando C.N. An Efficient Easily Adaptable System for Interpreting Natural Language Queries. pp. 110-122. American Journal of Computational Linguistics. 8(3-4) Jul-Dec 1982, USA.

[WEB81] Weber, David J. Prospects for Computer-Asisted Dialect Adaptation. pp. 165-177. American Journal of Computational Linguistics. 7(3) Jul-Sept 1981, USA.

[WEI83] Wieschedel, Ralph M. and Sondheimer, Norman K. Meta-rules as a Basis for Processing Ill-Formed Output. pp.161-177. American Journal of Computational Linguistics. 9(3-4) Jul-Dec 1983, USA.

REFERENCIAS

[WOO80] Woods, William A. Cascaded ATN Grammars. pp. 1-12. American Journal of Computational Linguistics. 6(1) Jan-Mar 1980, USA.

[WOO86] Woods, William A. Transition Network Grammars for Natural Language Analysis. pp. 71-87. Readings in Natural Language Processing; edited by Grosz, Sparck Jones and Webber. Morgan Kaufmann Publishers Inc. 1986, USA.

CAPITULO II

[AIT74] Aitchison, Jean. General Linguistics. Teach Yoursef Books. 1974, Great Britain.

[CHA86] Charniak, Eugene. and McDermott, Drew. Introduction to Artificial Intelligence. Addison-Wesley. 1986, USA.

[CHO62] Chomsky, Noam. Syntactic Structures. Mouton & Co. 1962, Netherlands.

[CHO76] Chomsky, Noam. Aspectos de la Teoria de la Sintàxis. Aguilar. 1976, EspaÑa.

[HAD75] Hadlich, Roger. Gramática Transformativa del español. Gredos. 1975, Madrid.

[KAT79] Katz, Jerrold J. Teoria Semntica. Aguilar. 1979, Madrid.

[LEE85] Leech, Geoffrey. Semàntica. Colección AU 187. Alianza Editorial. 1985, Madrid.

[NAG85] Nagao, Makoto., Tsujii, Jun-ichi. and Nakamura, Jun-ichi. The Japanese Government Proyect for Machine Translation. pp. 91-110. American Journal of Computational Linguistics. 11(2-3) Apr-Sept 1985, USA.

[STO65] Stockwell, Robert P., Bowen J.Donald. and Martin John W. The Grammatical Structures of English and Spanish. University of Chicago Press. 1965, USA.

[TOR76] Torres Lemus, Alfonso. Filologia segundo curso. Porrùa. 1976, Mèxico.

REFERENCIAS

CAPITULO III

[BOI87] Boisen, Sean. Language Processing.Using Definite Clause Grammars. pp. 46-56. A.I. Expert. 2(6) June 1987. Miller Freeman Publications. USA.

[CHO76] Chomsky, Noam. Aspectos de la Teoria de la Sintàxis. Aguilar. 1976, España.

[CLO84] Clocksin, William F. and Mellish, Chistopher S. Programming in Prolog. Springer-Verlag. 1981, Germany

[GAZ85] Gazdar, G., Klein, E., Pullum, G.K. and Sag, I.A. Generalized Phrase Structure Grammar. Basil Blackwell Publ. Ltd. 1985, Oxford.

[GRI71] Gries, David. Compiler Construction for Digital Computers. John Wiley & Sons Inc. 1976, USA.

[JIM86] Jiménez, Alejandro. Low-Quality Fully Automatic Machine Translation by Syntax Only. M.Sc. Dissertation. University of Essex. 1986, England.

[JOS82] Joshi, Aravind K. Phrase Structure Trees Bear More Fruit than You Would Have Thought. pp. 1-11. American Journal of Computational Linguistics. 8(1) Jan-Mar 1982. USA.

[LLO84] Lloyd, John W. Foundations of Logic Programming. Springer-Verlag. 1984, Germany.

[PER80] Pereira, F.C.N. and Warren, D.H.D. Definite Clause Grammars for Language Anlysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks. Artificial Intelligence, 13. pp. 231-278. North Holland Publ. Co. 1980.

[PER81] Pereira, Fernando. Extraposition Grammars. pp. 243-256. American Journal of Computational Linguistics. 7(4) Oct-Dec 1981. USA.

[RIC85] Rich, Elaine. Artificial Intelligence. McGraw-Hill. 1985, Singapore.

[SCH82] Schubert, Lenhart K. From English to Logic: Context-Free Computation of 'Conventional' Logical Translation. pp. 26-44. American Journal of Computational Linguistics. 8(1) Jan-Mar 1982, USA.

CAPITULO V

[RAD86] Radford, Andrew. Transformational Syntax. Cambridge University Press. 1986, Great Britain.

B I B L I O G R A F I A

Anderson, John R. y Bower, Gordon H. Memoria Asociativa. Limusa. 1977, México.

Beristáin, Helena. Gramática Estructural de la Lengua Española. UNAM. 1984, México.

Carbonell, Jaime G. and Hayes, Philip J. Recovery Strategies for Parsing Extragrammatical Language. pp. 123-146. American Journal of Computational Linguistics. 9(3-4) Jul-Dec 1983, USA.

Charniak, Eugene. Computational Semantics. Fundmental Studies in Computer Science. North-Holland. 1978, Netherlands.

Clocksin, William. A Prolog Primer. pp. 147-158. Byte 12(9) August 1987. McGraw-Hill. USA.

Cohen, Robin. Analyzing the Structure of Augmentative Discourse. pp. 11-24. American Journal of Computational Linguistics. 13(1-2) Jan-Jun 1987, USA.

Colmerauer, Alain. Prolog, Lenguaje de la Inteligencia Artificial. pp. 1072-1082. Mundo Cientifico. 4(41) Nov. 1984. Fontalba. Barcelona.

Colmerauer, Alain. Opening the Prolog III Universe. pp. 177-182. Byte 12(9) August 1987. McGraw-Hill. USA.

Cuadrado, Clara Y. and Cuadrado, John L. Prolog Goes to Work. pp. 151-158. Byte 10(8) August 1985. McGraw-Hill. USA.

Cuadrado, Clara Y. and Cuadrado, John L. Handling Conflicts In Data. pp. 193-202. Byte 11(12) November 1986. McGraw-Hill. USA.

Diaz-Guerrero, Rogelio. y Salas, Miguel. El Diferencial Semántico del Idioma Español. Trillas. 1975. México.

Dubois, Jean et al. Diccionario de Linguistica. Alianza Editorial. 1979, México.

Eisenbach, Susan. and Sadler, Chris. Declarative Languages: An Overview. pp. 181-197. Byte 10(8) August 1985. McGraw-Hill. USA.

BIBLIOGRAFIA

Emond, J.C. and Paulissen, A. The Art of Deduction. pp. 207-214. Byte 11(12) November 1986. McGraw-Hill. USA.

Greniewski, H. Cibernètica sin Matemàticas. Fondo de Cultura Econòmica. 1978, Mèxico.

Gunji, Takao. Subcategorization and Word Order. pp.51-73. Language and Artificial Intelligence. Makoto Nagao, editor. North-Holland. 1987, Netherlands.

Halliday, M.A.K. Text as Semantic Choice in Social Contexts. pp. 176-225. Grammars and Descriptions. Edited By Van Dijk, Teun A., Petofi and Jànos S. Walter de Gruyter & Co. 1977, Germany.

Hayes, Philip J. Flexible Parsing. pp. 232-242. American Journal of Computational Linguistics. 7(4) Oct-Dec 1981. USA.

Hendrix, Gary G. Semantic Aspects of Translation. pp.267-283. Readings in Natural Language Processing; edited by Grosz, Sparck Jones y Webber. Morgan Kaufmann Publications Inc. 1986, USA.

Hobbs, Jerry R. Resolving Pronoun References. pp.339-352. Readings in Natural Language Processing; edited by Grosz, Sparck Jones and Webber. Morgan Kaufmann Publishers Inc. 1986, USA.

Hobbs, Jerry R. and Shieber, Stuart M. An Algorithm for Generating Quantifier Scopings. pp.47-63. American Journal of Computational Linguistics. 13(1-2) Jan-Jun 1987. USA.

ICOT Institute for New Generation Computer Technology. Fifth Generation Computer. 1984, Tokyo.

Jimènez, Alejandro. Porque PROLOG. pp. 17-23. Revista de Computaciòn Ø10. 7(9) 1987, Mèxico.

Kac, Michael B., Manaster-Ramer, Alexis. and Rounds, William C. Simultaneous-Distributive Coordination and Context-Freeness. pp.25-30. American Journal of Computational Linguistics. 13(1-2) Jan-Jun 1987. USA.

Kowalsky, Robert. Logic Programming. pp. 161-177. Byte 10(8) August 1985, USA.

Lane, Alex. Simulating a Microprocessor. pp.161-168. Byte 12(9) August 1987, USA.

Lewandowsky, Theodor. Diccionario de Linguistica. Catedra. Madrid, 1982.

BIBLIOGRAFIA

Lewis, Robert. A Prolog to the Future. pp. 284-295. PC World. December 1986, USA.

Marcus, Claudia. Prolog Programming, Applications for Database Systems, Expert Systems and Natural Language Systems. Addison Wesley. 1986, USA.

Matthews, M.Haytham. Prolog and C Join Forces. pp.34-44. Computer Language. July 1987. Miller Freeman Publ. San Francisco Ca.

Mora, Alejandro de la. Las Partes de la Oración. ANUIES. 1975, México.

Moto-oka, Tohru. Los Ordenadores de la Quinta Generación. pp. 648-657. Mundo Científico. 6(37) Junio 1984. Fontalba. Barcelona.

Nagao, Makoto; (editor). Language and Artificial Intelligence. North-Holland, 1987, Netherlands.

Palmer, Harold. and Blandford, F.G. A Grammar of Spoken English. Cambridge University Press. 1976, London.

Pereira, Fernando. The Indiscipline of PROLOG Programming. pp. 7-8. A.I. Expert. 2(6) June 1987, USA.

Piñeiro, Jaime. VOX Compendio de Dificultades de la Lengua Inglesa. Bibliograf. 1978, Barcelona.

Pylyshyn, Zenon W. Perspectivas de la Revolución de los Computadores. Alianza Editorial. 1975, Madrid.

Quirk, Randolph., Greenbaum, Sidney., Leech, Geoffrey., and Svartvick, Jan. A Grammar of Contemporary English. Longman Group Ltd. 1980, London.

Sánchez, Antonio A. Paradigmas y Controversias en la traducción Automática de textos. IV Reunión Nacional de Inteligencia Artificial- memorias. pp. 48-53. 1987, Puebla.

Schank, Roger. Representation and Understanding of Text. pp. 575-619. Machine Intelligence 8 (Elcock, E.W., Michie, Donald). Ellis Horwood Ltd. 1977, Great Britain.

Seuren, Pieter A.M. Semantic Syntax. Oxford University Press. 1974, Great Britain.

Szpakowicz, Stan. Logic Grammars. pp. 185-195. Byte 12(9) August 1987. McGraw-Hill. USA.

Tanaka, Hozumi. DCKR -- Knowledge Representation in Prolog and Its Application to Natural Language Processing. pp.353-366. Language and Artificial Intelligence. Makoto Nagao, editor. North-Holland. 1987, Netherlands.

Tucker, Allen B. Programming Languages. pp. 230-252 350-379 382-409. McGraw-Hill. 1986, Singapore.

Van Dijk, Teun A. Connectives in Text Grammar and Text Logic. pp. 11-61. Grammars and Descriptions. Edited By Van Dijk, Teun A. and Petofi, János S. Walter de Gruyter & Co. 1977, Germany.

Werner, Abraham. Diccionario de Terminologia Linguistica Actual. Gredos. Madrid, 1981.

Winston, Patrick H. Artificial Intelligence. Addison Wesley. 1984, USA.

APENDICE A

LISTADO DEL SISTEMA DE
TRADUCCION AUTOMATICA

```
/*********************************************************/
/*                                                       */
/*    Traductor Sintáctico-Semántico Bidireccional       */
/*    Inglés-Español Basado en Gramáticas de Cláusula    */
/*    Definida (DCG) y Gramáticas de Estructura de       */
/*    Frase Generalizada (GPSG).                         */
/*                                                       */
/*    PARTE I: Programa Principal y                      */
/*             Rutinas de Entrada/Salida                 */
/*                                                       */
/*                        Alfredo Masanao D. Maeda       */
/*                        Ing. en Computación, UNAM      */
/*                        Abril, 1988     Méx. D.F.      */
/*                                                       */
/*********************************************************/



/*******************/
/*                 */
/*  Main Program   */
/*                 */

translator:-
    cls,
    box(0,0,24,79),
    repeat,
        main_screen,
        [!
          read_option(OPTION),
          case([[OPTION == 0F0 -) assert(finish),
                OPTION == 0f0 -) assert(finish),
                OPTION == 0I0 -) translate,
                OPTION == 0Z0 -) traduce ! fail ])
        !],
    finish,
    retract(finish),
    cls.

box(Y,X,Y1,X1):-
    C is X1-X,
    tmove(Y,X),
    mc(C,196),
    tmove(Y,X1),
    mc(1,191),
    C1 is Y1-Y,
    Y2 is Y+1,
    vert(X1,Y2,Y1),
    tmove(Y,X),
    mc(1,218),
    tmove(Y2,X),
    vert(X,Y2,Y1),
```

```
    tmove(Y1,X),
    mc(1,192),
    X2 is X+1,
    tmove(Y1,X2),
    mc(C,196),
    tmove(Y1,X1),
    mc(1,217),!.

vert(X,Y,Y1):-
    ctr_set(0,Y),
    repeat,
        ctr_inc(0,Z),
        tmove(Z,X),
        mc(1,179),
    Z == Y1.

main_screen:-
    tscroll(0,(2,2),(22,78)),
    tmove(5,18),
    wa(38,112),
    write(0 T R A N S L A T I O N   S Y S T E M $),
    tmove(10,25),
    wa(24,120),
    write(0 1)  English - Spanish. $),
    tmove(13,25),
    wa(24,120),
    write(0 2)  Spanish - English. $),
    tmove(17,25),
    wa(10,80),
    write(0 F)inish. $).

read_option(OPTION):-
    tmove(21,5),
    wa(22,29),
    write(0 Choose an option:      $),
    tmove(21,25),
    read_line(0,OPTION).


/*                  */
/*  English-Spanish */
/*                  */

translate:-
    retract(end),
    retract(fin),
    tscroll(0,(2,2),(22,78)),
    tmove(4,23),
    wa(33,112),
    write(0 ENGLISH - SPANISH  TRANSLATION $),
    repeat,
        !! english_spanish !),
    end.
```

```
english_spanish:-
    input_sentence1(ENGLISH_SENTENCE),
    [!
      tmove(15,3),
      write_list(ENGLISH_SENTENCE)
    !],
    english_parser(ENGLISH_SENTENCE,ENGLISH_PARSE_TREE),
    translation(ENGLISH_PARSE_TREE,SPANISH_PARSE_TREE), !,
    spanish_parser(SPANISH_SENTENCE,SPANISH_PARSE_TREE), !,
    tmove(18,3),
    write_list(SPANISH_SENTENCE),
    tmove(22,3),
    write($[ENTER]$),
    read_line(0,_),
    tscroll(0,(7,2),(22,78)).



/*                  */
/* Spanish-English */
/*                  */


traduce:-
    retract(fin),
    retract(end),
    tscroll(0,(2,2),(22,78)),
    tmove(4,23),
    wa(31,112),
    write($ TRADUCCION  ESPAÑOL - INGLES $),
    repeat,
        [! spanish_english !],
    fin.


spanish_english:-
    input_sentence2(SPANISH_SENTENCE),
    [!
      tmove(15,3),
      write_list(SPANISH_SENTENCE)
    !],
    spanish_parser(SPANISH_SENTENCE,SPANISH_PARSE_TREE),
    translation(ENGLISH_PARSE_TREE,SPANISH_PARSE_TREE), !,
    english_parser(ENGLISH_SENTENCE,ENGLISH_PARSE_TREE), !,
    tmove(18,3),
    write_list(ENGLISH_SENTENCE),
    tmove(22,3),
    write($[ENTER]$),
    read_line(0,_),
    tscroll(0,(7,2),(22,78)).
```

```
/*****************************/
/*                           */
/*    Input/Output Routines  */
/*                           */


end.          /* created to finish E-S translation */
fin.          /* created to finish S-E translation */


/*                                                 */
/* read the sentence and formalize it as a list */
/*                                                 */

input_sentence1(SENTENCE_LIST):-
    gc(full),
    tmove(8,2),
    write(' Please type in the sentence: '),
    tmove(9,2),
    write('     ( 'end' to finish. ) '),
    tmove(10,2),
    read_line(0,SENTENCE), !,
    create_list(SENTENCE,LIST,ELEMENT,SENTENCE_LIST), !.

input_sentence2(SENTENCE_LIST):-
    gc(full),
    tmove(8,2),
    write(' Por favor teclee la oraci'n: '),
    tmove(9,2),
    write('     ( 'fin' para terminar. ) '),
    tmove(10,2),
    read_line(0,SENTENCE), !,
    create_list(SENTENCE,LIST,ELEMENT,SENTENCE_LIST), !.

/* create the list structure from string */
create_list(SENTENCE,LIST,ELEMENT,SENTENCE_LIST):-
    quit_spaces(SENTENCE,STRING),
    string_search(' ',STRING,LOCATION),        /* locate a word */
    substring(STRING,0,LOCATION,SUBSTRING),
    chng_lcase(SUBSTRING,STRING_WORD),         /* change to lowercase */
    atom_string(ATOM_WORD,STRING_WORD),
    append(ELEMENT,[ATOM_WORD],LIST),          /* add it to the list */
    string_length(STRING,LENGTH),
    TO is LENGTH-LOCATION-1,
    FROM is LOCATION+1,
    substring(STRING,FROM,TO,REST_STRING), !,
    create_list(REST_STRING,_,LIST,SENTENCE_LIST).

create_list(REST_STRING,LIST,ELEMENT,LIST):-
    quit_spaces(REST_STRING,STRING),
    chng_lcase(STRING,STRING_WORD),
    quit_point(STRING_WORD,ATOM_WORD),       /* remove final point */
    append(ELEMENT,[ATOM_WORD],LIST).
```

```
create_list(REST_STRING,LIST,ELEMENT,LIST):-
    quit_spaces(REST_STRING,STRING),
    chng_lcase(STRING,STRING_WORD),
    atom_string(ATOM_WORD,STRING_WORD),
    append(ELEMENT,[ATOM_WORD],LIST).


/* eliminate extra spaces */
quit_spaces(SENTENCE,SENTENCE):-
    substring(SENTENCE,B,1,CHAR),
    CHAR \== ' '.

quit_spaces(SENTENCE,STRING):-
    string_length(SENTENCE,LENGTH),
    NEW_LENGTH is LENGTH-1,
    substring(SENTENCE,1,NEW_LENGTH,PRESTRING),
    quit_spaces(PRESTRING,STRING).


/* change uppercase to lowercase */
chng_lcase(SUBSTRING,STRING_WORD):-
    nth_char(0,SUBSTRING,CODE),
    CODE ) 64,                          /* uppercase */
    CODE ( 91,
    NEW_CODE is CODE+32,                /* change it to lowercase */
    name(CHAR,[NEW_CODE]),
    string_length(SUBSTRING,LENGTH),
    NEW_LENGTH is LENGTH-1,
    substring(SUBSTRING,1,NEW_LENGTH,NEW_STRING),
    concat(CHAR,NEW_STRING,STRING_WORD).

chng_lcase(SUBSTRING,SUBSTRING):-
    nth_char(0,SUBSTRING,CODE),
    CODE ) 96,                          /* lowercase */
    CODE ( 123.


/* remove final point (if any) */
quit_point(REST_SENTENCE,ATOM_WORD):-
    string_length(REST_SENTENCE,LENGTH),
    FROM_TO is LENGTH-1,
    substring(REST_SENTENCE,FROM_TO,1,POINT), !,
    POINT == '.',
    substring(REST_SENTENCE,0,FROM_TO,STRING_WORD),
    atom_string(ATOM_WORD,STRING_WORD).


/* add the new word to the list */
append([],L,L).
append([X|L1],L2,[X|L3]):-
    append(L1,L2,L3).
```

```
/* write the list on screen as a sentence */
write_list([]):- !.

write_list([H|T]):-
    write(H),
    write(' '),
    write_list(T),!.
```

```
/*********************************************************/
/*                                                       */
/*    Traductor Sintáctico-Semántico Bidireccional       */
/*    Inglés-Español Basado en Gramáticas de Cláusula    */
/*    Definida (DCG) y Gramáticas de Estructura de       */
/*    Frase Generalizada (GPSG).                         */
/*                                                       */
/*    PARTE 2: Analizador Sintáctico-Semántico           */
/*             Inglés                                    */
/*                                                       */
/*                                                       */
/*                        Alfredo Mazamao D. Maeda       */
/*                        Ing. en Computación, UNAM      */
/*                        Abril, 1988     Méx. D.F.      */
/*                                                       */
/*********************************************************/



/*********************************************/
/*                                           */
/*  This routine performs the DCG-GPSG based */
/*  syntactical and semantical analysis.     */
/*                                           */


/*             */
/* English Parser */
/*             */


english_parser(SENTENCE,TREE):-
    sentence(TREE,SENTENCE,[]).

english_parser(SENTENCE,TREE):-
    tmove(18,3),
    wa(37,110),
    write(0 **** Invalid English Sentence **** 0),
    tmove(20,3),
    wa(3,140),
    write('(=) '),
    write_list(SENTENCE),
    tmove(22,3),
    write(0[ENTER]0),
    read_line(0,_),
    tscroll(0,(7,2),(22,78)),
    flush,
    !, fail.
```

```
/*          */
/* Sentence */
/*          */

/* S */
sentence([end],[end],_).          /* used only to finish */

sentence([s,NP,VP],S0,S):-
    noun_phrase(NP,S0,S1),
    verb_phrase(NP,VP,S1,S).

sentence([s,NP,AUX,VP],S0,S):-    /* (VP (-- gerund mode) */
    [!
      noun_phrase(NP,S0,S1),
      auxiliary(NP,AUX,S1,S2)
    !],
    verb_phrase(NP,VP,S2,S), !,
    check_gerund(VP).


/*             */
/* Noun Phrase */
/*             */

/* NP --) PN */
noun_phrase([np,PROP_NOUN],S0,S):-
    proper_noun(PROP_NOUN,S0,S).

/* NP --) PN, RC */
noun_phrase([np,PROP_NOUN,REL],S0,S):-
    proper_noun(PROP_NOUN,S0,S1),
    relative_clause(PROP_NOUN,REL,S1,S).

/* NP --) PN, RC, PP */
noun_phrase([np,PROP_NOUN,REL,PREP_PH],S0,S):-
    proper_noun(PROP_NOUN,S0,S1),
    relative_clause(PROP_NOUN,REL,S1,S2),
    prepositional_phrase(REL,PREP_PH,S2,S).

/* NP --) PRON */
noun_phrase([np,PER_PRON],S0,S):-
    personal_pronoun(PER_PRON,S0,S).

/* NP --) PRON, RC */
noun_phrase([np,PER_PRON,REL],S0,S):-
    personal_pronoun(PER_PRON,S0,S1),
    relative_clause(PER_PRON,REL,S1,S).

/* NP --) PRON, RC, PP */
noun_phrase([np,PER_PRON,REL,PREP_PH],S0,S):-
    personal_pronoun(PER_PRON,S0,S1),
    relative_clause(PER_PRON,REL,S1,S2),
    prepositional_phrase(REL,PREP_PH,S2,S).
```

```
/* NP --) DET, CN */
noun_phrase([np,DET,COM_NOUN],S0,S):-
    common_noun(COM_NOUN,S0,S),
    determiner(COM_NOUN,DET,S0,S1).

/* NP --) DET, ADJ, CN */
noun_phrase([np,DET,ADJ,COM_NOUN],S0,S):-
    adjective(ADJ,S0,S2),
    determiner(ADJ,DET,S0,S1),
    common_noun(ADJ,COM_NOUN,S2,S).

/* NP --) DET, CN, RC */
noun_phrase([np,DET,COM_NOUN,REL],S0,S):-
    common_noun(COM_NOUN,S0,S2),
    determiner(COM_NOUN,DET,S0,S1),
    relative_clause(COM_NOUN,REL,S2,S).

/* NP --) DET, ADJ, CN, RC */
noun_phrase([np,DET,ADJ,COM_NOUN,REL],S0,S):-
    adjective(ADJ,S0,S2),
    determiner(ADJ,DET,S0,S1),
    common_noun(ADJ,COM_NOUN,S2,S3),
    relative_clause(COM_NOUN,REL,S3,S).

/* NP --) DET, CN, RC, PP */
noun_phrase([np,DET,COM_NOUN,REL,PREP_PHI],S0,S):-
    common_noun(COM_NOUN,S0,S2),
    determiner(COM_NOUN,DET,S0,S1),
    relative_clause(COM_NOUN,REL,S2,S3),
    prepositional_phrase(REL,PREP_PH,S3,S).

/* NP --) DET, ADJ, CN, RC, PP */
noun_phrase([np,DET,ADJ,COM_NOUN,REL,PREP_PHI],S0,S):-
    adjective(ADJ,S0,S2),
    determiner(ADJ,DET,S0,S1),
    common_noun(ADJ,COM_NOUN,S2,S3),
    relative_clause(COM_NOUN,REL,S3,S4),
    prepositional_phrase(REL,PREP_PH,S4,S).

/* NP --) Q, CN */
noun_phrase([np,QUANT,COM_NOUN],S0,S):-
    quantifier(QUANT,S0,S1),
    common_noun(QUANT,COM_NOUN,S1,S).

/* NP --) Q, ADJ, CN */
noun_phrase([np,QUANT,ADJ,COM_NOUN],S0,S):-
    quantifier(QUANT,S0,S1),
    adjective(QUANT,ADJ,S1,S2),
    common_noun(ADJ,COM_NOUN,S2,S).
```

```
/* NP --> Q, CN, RC */
noun_phrase([np,QUANT,COM_NOUN,REL],S0,S):-
    quantifier(QUANT,S0,S1),
    common_noun(QUANT,COM_NOUN,S1,S2),
    relative_clause(COM_NOUN,REL,S2,S).

/* NP --> Q, ADJ, CN, RC */
noun_phrase([np,QUANT,ADJ,COM_NOUN,REL],S0,S):-
    quantifier(QUANT,S0,S1),
    adjective(QUANT,ADJ,S1,S2),
    common_noun(ADJ,COM_NOUN,S2,S3),
    relative_clause(COM_NOUN,REL,S3,S).

/* NP --> Q, CN, RC, PP */
noun_phrase([np,QUANT,COM_NOUN,REL,PREP_PH],S0,S):-
    quantifier(QUANT,S0,S1),
    common_noun(QUANT,COM_NOUN,S1,S2),
    relative_clause(COM_NOUN,REL,S2,S3),
    prepositional_phrase(REL,PREP_PH,S3,S).

/* NP --> Q, ADJ, CN, RC, PP */
noun_phrase([np,QUANT,ADJ,COM_NOUN,REL,PREP_PH],S0,S):-
    quantifier(QUANT,S0,S1),
    adjective(QUANT,ADJ,S1,S2),
    common_noun(ADJ,COM_NOUN,S2,S3),
    relative_clause(COM_NOUN,REL,S3,S4),
    prepositional_phrase(REL,PREP_PH,S4,S).



/*              */
/* Verb Phrase */
/*              */

/* VP --> IV */
verb_phrase(NP,[vp,IV],S0,S):-
    intransitive_verb(NP,IV,S0,S).

/* VP --> IV, ADV */
verb_phrase(NP,[vp,IV,ADV],S0,S):-
    intransitive_verb(NP,IV,S0,S1),
    adverb(ADV,S1,S).

/* VP --> IV, ADVP */
verb_phrase(NP,[vp,IV,ADVP],S0,S):-
    intransitive_verb(NP,IV,S0,S1),
    adverbial_phrase(ADVP,S1,S).

/* VP --> IV, PP */
verb_phrase(NP,[vp,IV,PREP_PH],S0,S):-
    intransitive_verb(NP,IV,S0,S1),
    prepositional_phrase(IV,PREP_PH,S1,S).
```

```
/* VP --> IV, ADV, PP */
verb_phrase(NP,[vp,IV,ADV,PREP_PH],S0,S):-
    intransitive_verb(NP,IV,S0,S1),
    adverb(ADV,S1,S2),
    prepositional_phrase(IV,PREP_PH,S2,S).

/* VP --> IV, ADVP, PP */
verb_phrase(NP,[vp,IV,ADVP,PREP_PH],S0,S):-
    intransitive_verb(NP,IV,S0,S1),
    adverbial_phrase(ADVP,S1,S2),
    prepositional_phrase(IV,PREP_PH,S2,S).


/* VP --> TV, NP  (not(NP1=NP2)) (TV(-)NP2) */
verb_phrase(NP1,[vp,TV,NP2],S0,S):-
    [!
      transitive_verb(NP1,TV,S0,S1),
      noun_phrase(NP2,S1,S)
    !],
    not(same(NP1,NP2)),
    match_tv_np(TV,NP2).

/* VP --> TV, NP, ADV  (not(NP1=NP2)) (TV(-)NP2) */
verb_phrase(NP1,[vp,TV,NP2,ADV],S0,S):-
    [!
      transitive_verb(NP1,TV,S0,S1),
      noun_phrase(NP2,S1,S2)
    !],
    not(same(NP1,NP2)),
    match_tv_np(TV,NP2),
    adverb(ADV,S2,S).

/* VP --> TV, NP, ADVP  (not(NP1=NP2)) (TV(-)NP2) */
verb_phrase(NP1,[vp,TV,NP2,ADVP],S0,S):-
    [!
      transitive_verb(NP1,TV,S0,S1),
      noun_phrase(NP2,S1,S2)
    !],
    not(same(NP1,NP2)),
    match_tv_np(TV,NP2),
    adverbial_phrase(ADVP,S2,S).


/* VP --> TV, NP, PP  (not(NP1=NP2)) (TV(-)NP2) */
verb_phrase(NP1,[vp,TV,NP2,PREP_PH],S0,S):-
    [!
      transitive_verb(NP1,TV,S0,S1),
      noun_phrase(NP2,S1,S2)
    !],
    not(same(NP1,NP2)),
    match_tv_np(TV,NP2),
    prepositional_phrase(TV,PREP_PH,S2,S).
```

```
/* VP --) TV, NP, ADV, PP  (not(NP1=NP2)) (TV(-)NP2) */
verb_phrase(NP1,[vp,TV,NP2,ADV,PREP_PH],S0,S):-
    [!
      transitive_verb(NP1,TV,S0,S1),
      noun_phrase(NP2,S1,S2)
    !],
    not(same(NP1,NP2)),
    match_tv_np(TV,NP2),
    adverb(ADV,S2,S3),
    prepositional_phrase(TV,PREP_PH,S3,S).

/* VP --) TV, NP, ADVP, PP  (not(NP1=NP2)) (TV(-)NP2) */
verb_phrase(NP1,[vp,TV,NP2,ADVP,PREP_PH],S0,S):-
    [!
      transitive_verb(NP1,TV,S0,S1),
      noun_phrase(NP2,S1,S2)
    !],
    not(same(NP1,NP2)),
    match_tv_np(TV,NP2),
    adverbial_phrase(ADVP,S2,S3),
    prepositional_phrase(TV,PREP_PH,S3,S).


/*                          */
/* Complementary  Routines */
/*                          */

/* PN */
proper_noun([pn,W,PN_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S)
    !],
    lex_pn(W,PN_SSFEAT,_),
    append([singular,3],PN_SSFEAT,PN_SUBFEAT).


/* PRON */
personal_pronoun([ppron,W,PPRON_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S)
    !],
    lex_pron(W,PPRON_SUBFEAT,_).

/* DET */
determiner(ELEM,[det,W,DET_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S),
      lex_eng_det(W,DET_SUBFEAT)
    !],
    check_a_an_det(ELEM,W),
    last(ELEM,ELEM_SUBFEAT),
    same(DET_SUBFEAT,ELEM_SUBFEAT).
```

```
determiner(ELEM,[det,W,DET_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S),
      lex_eng_det(W,DET_SUBFEAT)
    !],
    W \== 'a',                          /* for correct use of */
    W \== 'an',                         /* a, an determiners  */
    last(ELEM,ELEM_SUBFEAT),
    same(DET_SUBFEAT,ELEM_SUBFEAT).

check_a_an_det([_,WORD|_],W):-
    W == 'an',
    atom_string(WORD,STRING),
    nth_char(0,STRING,ASCII), !,
    case([ASCII == 97 -> true,
          ASCII == 101 -> true,
          ASCII == 105 -> true,
          ASCII == 111 -> true,
          ASCII == 117 -> true| fail]).

check_a_an_det([_,WORD|_],W):-
    W == 'a',
    atom_string(WORD,STRING),
    nth_char(0,STRING,ASCII), !,
    case([ASCII == 97 -> fail,
          ASCII == 101 -> fail,
          ASCII == 105 -> fail,
          ASCII == 111 -> fail,
          ASCII == 117 -> fail| true]).


/* CN */
common_noun([com_noun,W,CN_SUBFEAT],S0,S):-
    [!
      connects(S0,_,S1),
      connects(S1,W,S)
    !],
    lex_com_noun(W,CN_SUBFEAT,_).

common_noun(ELEM,[com_noun,W,CN_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S),
      last(ELEM,ELEM_SUBFEAT)
    !],
    lex_com_noun(W,CN_SUBFEAT,_),
    same(ELEM_SUBFEAT,CN_SUBFEAT).


/* ADJ */
adjective([adj,W,ADJ_SUBFEAT],S0,S):-
    [!
      connects(S0,_,S1),
      connects(S1,W,S)
```

```
    !],
    lex_adj(W,ADJ_SUBFEAT,_).

adjective(ELEM,[adj,W,ADJ_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S),
      last(ELEM,ELEM_SUBFEAT)
    !],
    lex_adj(W,ADJ_SUBFEAT,_),
    same(ELEM_SUBFEAT,ADJ_SUBFEAT).


/* Q */
quantifier([quant,W,Q_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S)
    !],
    lex_quant(W,Q_SUBFEAT,_).


/* RC */
relative_clause(ELEM,[rclause,W,VP],[WIS0],S):-
    lex_rel(W,_),
    verb_phrase([_,ELEM],VP,S0,S).


/* pp */
prepositional_phrase(REL,[prep_ph,[prep,PREP,PREP_SUBFEAT],NP],S0,S):-
    [!
      connects(S0,PREP,S1),
      PREP == 'with'
    !],
    lex_prep(PREP,PREP_SUBFEAT,_),
    [!
      noun_phrase(NP,S1,S),
      connects(S1,PRON,_),
      compare_pron(PRON),
      get_subfeat(NP,NP_SUBFEAT),
      same(PREP_SUBFEAT,NP_SUBFEAT)
    !],
    compare_prep_np(PREP_SUBFEAT,NP_SUBFEAT),
    compare_rel_np(REL,NP_SUBFEAT).

prepositional_phrase(IV,[prep_ph,[prep,PREP,PREP_SUBFEAT],NP],S0,S):-
    [!
      connects(S0,PREP,S1),
      PREP == 'with'
    !],
    lex_prep(PREP,PREP_SUBFEAT,_),
    [!
      noun_phrase(NP,S1,S),
      connects(S1,PRON,_),
      compare_pron(PRON),
```

```
      get_subfeat(NP,NP_SUBFEAT),
      same(PREP_SUBFEAT,NP_SUBFEAT)
    !),
    compare_prep_np(PREP_SUBFEAT,NP_SUBFEAT),
    compare_iv_np(IV,NP_SUBFEAT).

prepositional_phrase(TV,[prep_ph,[prep,PREP,PREP_SUBFEAT],NP],S0,S):-
    [!
      connects(S0,PREP,S1),
      PREP == 'with'
    !],
    lex_prep(PREP,PREP_SUBFEAT,_),
    [!
      noun_phrase(NP,S1,S),
      connects(S1,PRON,_),
      compare_pron(PRON),
      get_subfeat(NP,NP_SUBFEAT),
      same(PREP_SUBFEAT,NP_SUBFEAT)
    !],
    compare_tv_np(TV,NP_SUBFEAT).

prepositional_phrase(_,[prep_ph,[prep,PREP,PREP_SUBFEAT],NP],S0,S):-
    [!
      connects(S0,PREP,S1),
      PREP == 'in',
      lex_prep(PREP,PREP_SUBFEAT,_),
      noun_phrase(NP,S1,S),
      get_subfeat(NP,NP_SUBFEAT),
      same(PREP_SUBFEAT,NP_SUBFEAT),
      connects(S1,LOC,_)
    !],
    check_loc(LOC).

prepositional_phrase(_,[prep_ph,[prep,PREP,PREP_SUBFEAT],NP],S0,S):-
    [!
      connects(S0,PREP,S1),
      PREP == 'in',
      lex_prep(PREP,PREP_SUBFEAT,_),
      noun_phrase(NP,S1,S),
      get_subfeat(NP,NP_SUBFEAT),
      same(PREP_SUBFEAT,NP_SUBFEAT),
      connects(S1,_,S2),
      connects(S2,LOC,_)
    !],
    check_loc(LOC).

/* check syntactic concordance between 'with' PREP and PRON */
compare_pron(PRON):-                          /* for correct use of */
    case([PRON == 'i' -> fail,                 /* 'with' preposition */
          PRON == 'he' -> fail,
          PRON == 'she' -> fail,
          PRON == 'we' -> fail,
          PRON == 'they' -> fail! true]).
```

```
/* check semantic concordance between PREP and NP in both parsers */
compare_prep_np([P_SSFEAT1,P_SSFEAT2,_,_,_,_,_,_,_],[NP_SSFEAT1,NP_SSFEAT2,_,_,_,_,_,_,_]):-
    same(P_SSFEAT1,NP_SSFEAT1), !,
    same(P_SSFEAT2,NP_SSFEAT2).

/* check semantic concordance between NP and PP verbs */
compare_rel_np(REL,NP_SUBFEAT):-
    last(REL,TEMP1), !,
    tail(TEMP1,[[_,VERB,_,REL_SUBFEAT]]), !,
    case([VERB == disappear -) true,
          VERB == fly -) compare_verb1(REL_SUBFEAT,NP_SUBFEAT),
          VERB == live -) compare_verb2(REL_SUBFEAT,NP_SUBFEAT),
          VERB == play -) compare_verb3(REL_SUBFEAT,NP_SUBFEAT),
          VERB == run -) compare_verb1(REL_SUBFEAT,NP_SUBFEAT),
          VERB == sleep -) compare_verb3(REL_SUBFEAT,NP_SUBFEAT),
          VERB == smoke -) compare_verb4(REL_SUBFEAT,NP_SUBFEAT),
          VERB == speak -) compare_verb4(REL_SUBFEAT,NP_SUBFEAT),
          VERB == sing -) compare_verb4(REL_SUBFEAT,NP_SUBFEAT),
          VERB == study -) compare_verb4(REL_SUBFEAT,NP_SUBFEAT)| fail]).

/* check semantic concordance between IV and PP verbs */
compare_iv_np([IV,NP_SUBFEAT]):-
    tail(IV,[VERB,_,IV_SUBFEAT]), !,
    case([VERB == disappear -) true,
          VERB == fly -) compare_verb1(REL_SUBFEAT,NP_SUBFEAT),
          VERB == live -) compare_verb2(REL_SUBFEAT,NP_SUBFEAT),
          VERB == play -) compare_verb3(REL_SUBFEAT,NP_SUBFEAT),
          VERB == run -) compare_verb1(REL_SUBFEAT,NP_SUBFEAT),
          VERB == sleep -) compare_verb3(REL_SUBFEAT,NP_SUBFEAT),
          VERB == smoke -) compare_verb4(REL_SUBFEAT,NP_SUBFEAT),
          VERB == speak -) compare_verb4(REL_SUBFEAT,NP_SUBFEAT),
          VERB == sing -) compare_verb4(REL_SUBFEAT,NP_SUBFEAT),
          VERB == study -) compare_verb4(REL_SUBFEAT,NP_SUBFEAT)| fail]).

/* check semantic concordance between TV and PP verbs */
compare_tv_np([_,VERB,_,_,[_,_,TV_SSFEAT]],[_,_,NP_SSFEAT]):-
    case([VERB == adopt -) same(NP_SSFEAT,[anim_y,_,_,_,_,_]),
          VERB == admire -) same(NP_SSFEAT,_),
          VERB == break -) same(NP_SSFEAT,[anim_n,_,_,wing_n,loc_n,_]),
          VERB == buy -) same(NP_SSFEAT,TV_SSFEAT),
          VERB == catch -) same(NP_SSFEAT,[_,_,_,_,_,loc_n,_]),
          VERB == chase -) same(NP_SSFEAT,[_,_,_,_,_,loc_n,_]),
          VERB == clean -) same(NP_SSFEAT,[anim_n,_,_,_,_,_,_]),
          VERB == cut -) same(NP_SSFEAT,[anim_n,hum_n,masc_y,wing_n,loc_n,meal_n]),
          VERB == defend -) same(NP_SSFEAT,[anim_y,_,_,_,_,_,_]),
          VERB == eat -) same(NP_SSFEAT,[_,_,_,_,_,_,meal_y]),
          VERB == find -) same(NP_SSFEAT,_),
          VERB == hear -) same(NP_SSFEAT,[_,_,_,_,_,loc_n,meal_n]),
          VERB == listen -) same(NP_SSFEAT,[anim_y,_,_,_,_,_,_]),
          VERB == love -) same(NP_SSFEAT,[_,_,_,_,_,loc_n,_]),
          VERB == see -) same(NP_SSFEAT,_),
          VERB == study -) same(NP_SSFEAT,[_,hum_n,_,_,_,loc_n,meal_n])| fail]).
```

```
/* semantic concordance rules in both parsers */
compare_verb1([_,_,_,_,_,_,_,SSFEAT,_],[_,_,_,_,_,_,_,NP_SSFEAT,_]):-
    same(SSFEAT,NP_SSFEAT).

compare_verb2([_,_,SSFEAT1,_,_,_,_,_],[_,_,NP_SSFEAT1,_,_,_,_,_]):-
    same(SSFEAT1,NP_SSFEAT1).

compare_verb3([_,_,_,_,_,_,_,SSFEAT1,SSFEAT2],[_,_,_,_,_,_,_,NP_SSFEAT1,NP_SSFEAT2]):-
    same(SSFEAT1,NP_SSFEAT1), !,
    same(SSFEAT2,NP_SSFEAT2).

compare_verb4([_,_,_,SSFEAT,_,_,_,_],[_,_,_,NP_SSFEAT,_,_,_,_]):-
    same(SSFEAT,NP_SSFEAT).


/* check semantic concordance between verb and 'in' PREP */
check_loc(LOC):-
    lex_pn(LOC,[_,_,_,_,loc_y,_],_).

check_loc(LOC):-
    lex_com_noun(LOC,[_,_,_,_,_,_,loc_y,_],_).


/* IV */
intransitive_verb(NP,[iv,ROOT,TENSE,IV_SUBFEAT],S0,S):-
    [!
      get_subfeat(NP,NP_SUBFEAT),
      connects(S0,W,S)
    !],
    lex_iv(ROOT,W,TENSE,IV_SUBFEAT,_,_),
    same(NP_SUBFEAT,IV_SUBFEAT).


/* ADV */
adverb([adv,W],S0,S):-
    [!
      connects(S0,W,S)
    !],
    lex_adv(W,_).


/* ADVP */
adverbial_phrase([advp,DEG,ADV],S0,S):-
    [!
      degree(DEG,S0,S1)
    !],
    adverb(ADV,S1,S).


/* DEG */
degree([deg,W],S0,S):-
    [!
```

```
        connects(S0,W,S)
    !),
    lex_deg(W,_).


/* TV */
transitive_verb(NP,[tv,ROOT,TENSE,TV_SUBFEAT1,TV_SUBFEAT2),S0,S):-
    (!
      get_subfeat(NP,NP_SUBFEAT),
      connects(S0,W,S)
    !),
    lex_tv(ROOT,W,TENSE,TV_SUBFEAT1,TV_SUBFEAT2,_,_),
    same(NP_SUBFEAT,TV_SUBFEAT1).


/* check semantic concordance between TV and NP in both parsers */
match_tv_np(TV,NP):-
    (!
      last(TV,TV_SUBFEAT),
      get_subfeat(NP,NP_SUBFEAT)
    !),
    same(TV_SUBFEAT,NP_SUBFEAT).


/* AUX */
auxiliary(NP,[aux,ROOT,TENSE],S0,S):-
    (!
      get_subfeat(NP,NP_SUBFEAT),
      connects(S0,W,S)
    !),
    lex_aux(ROOT,W,TENSE,AUX_SUBFEAT,_,_),
    same(NP_SUBFEAT,AUX_SUBFEAT).


/* check syntactic concordance between AUX and verb */
check_gerund([_,[_,_,gerund|_]|_]).


/*                          */
/* Miscelaneous Routines */
/* for both parsers.        */
/*                          */

connects([W|S],W,S).

same(SUBFEAT,SUBFEAT).

first([X|_],X).

last([X],X):- !.
last([_|Y],X):-
    last(Y,X).
```

```
tail([_|Y],Y).

get_subfeat(ELEM,SUBFEAT):-
    tail(ELEM,TEMP1), !,
    first(TEMP1,TEMP2), !,
    last(TEMP2,SUBFEAT).
```

```
/*************************************************************/
/*                                                           */
/*    Traductor Sintáctico-Semántico Bidireccional           */
/*    Inglés-Español Basado en Gramáticas de Cláusula        */
/*    Definida (DCG) y Gramáticas de Estructura de           */
/*    Frase Generalizada (GPSG).                             */
/*                                                           */
/*    PARTE 3: Proceso de Traducción                         */
/*                                                           */
/*                          Alfredo Masanao D. Maeda         */
/*                          Ing. en Computación, UNAM        */
/*                          Abril, 1988     Méx. D.F.        */
/*                                                           */
/*************************************************************/



/*****************************/
/* This routine performs the */
/* English  (--)  Spanish    */
/* translation process.      */
/*                           */

translation([end],[fin]):-        /* this is only to finish */
    assert(end),
    assert(fin).

/* S --) NP, VP  (--)  S --) NP, VP */
translation([s,[np|NP],[vp|VP]],[s,[np|S_NP],[vp|S_VP]]):-
    trans_np(NP,S_NP),
    trans_vp(VP,S_VP).

/* S --) NP, AUX, VP  (--)  S --) NP, AUX, VP */
translation([s,[np|NP],[aux|AUX],[vp|VP]],[s,[np|S_NP],[aux|S_AUX],[vp|S_VP]]):-
     trans_np(NP,S_NP),
     trans_aux(NP,AUX,S_AUX),
     trans_vp(VP,S_VP).


/* NP --) PN  (--)  NP --) PN */
trans_np([[pn,PROP_NOUN,SFEAT]],[[pn,S_PROP_NOUN,SFEAT]]):-
    trans_pn(PROP_NOUN,SFEAT,S_PROP_NOUN).

/* NP --) PRON  (--)  NP --) PRON */
trans_np([[ppron,PER_PRON,SFEAT]],[[ppron,S_PER_PRON,SFEAT]]):-
    trans_pron(PER_PRON,SFEAT,S_PER_PRON).

/* NP --) DET, CN  (--)  NP --) DET, CN */
trans_np([[det,DET,DET_SUBFEAT],[com_noun,COM_NOUN,CN_SUBFEAT]],
         [[det,S_DET,DET_SUBFEAT],[com_noun,S_COM_NOUN,CN_SUBFEAT]]):-
    trans_com(COM_NOUN,CN_SUBFEAT,S_COM_NOUN),
    trans_det(COM_NOUN,DET,DET_SUBFEAT,S_DET).
```

```
/* NP --) DET, ADJ, CN  (--)  NP --) DET, CN, ADJ */
trans_np([[det,DET,DET_SUBFEAT],[adj,ADJ,ADJ_SUBFEAT],[com_noun,COM_NOUN,CN_SUBFEAT]],
         [[det,S_DET,DET_SUBFEAT],[com_noun,S_COM_NOUN,CN_SUBFEAT],[adj,S_ADJ,ADJ_SUBFEAT]]):-
    trans_com(COM_NOUN,CN_SUBFEAT,S_COM_NOUN),
    trans_det(COM_NOUN,DET,DET_SUBFEAT,S_DET),
    trans_adj(ADJ,ADJ_SUBFEAT,S_ADJ).

/* NP --) Q, CN  (--)  NP --) Q, CN */
trans_np([[quant,QUANT,QUANT_SUBFEAT],[com_noun,COM_NOUN,CN_SUBFEAT]],
         [[quant,S_QUANT,QUANT_SUBFEAT],[com_noun,S_COM_NOUN,CN_SUBFEAT]]):-
    trans_quant(QUANT,QUANT_SUBFEAT,S_QUANT),
    trans_com(COM_NOUN,CN_SUBFEAT,S_COM_NOUN).

/* NP --) Q, ADJ, CN  (--)  Q, CN, ADJ */
trans_np([[quant,QUANT,QUANT_SUBFEAT],[adj,ADJ,ADJ_SUBFEAT],[com_noun,COM_NOUN,CN_SUBFEAT]],
         [[quant,S_QUANT,QUANT_SUBFEAT],[com_noun,S_COM_NOUN,CN_SUBFEAT],[adj,S_ADJ,ADJ_SUBFEAT]]):-
    trans_quant(QUANT,QUANT_SUBFEAT,S_QUANT),
    trans_com(COM_NOUN,CN_SUBFEAT,S_COM_NOUN),
    trans_adj(ADJ,ADJ_SUBFEAT,S_ADJ).


/* NP --) PN, RC  (--)  NP --) PN, RC */
trans_np([[pn,PROP_NOUN,SFEAT],[rclause,REL,[vp|VP]]],
         [[pn,S_PROP_NOUN,SFEAT],[rclause,S_REL,[vp|S_VP]]]):-
    trans_pn(PROP_NOUN,SFEAT,S_PROP_NOUN),
    trans_rel(REL,VP,S_REL,S_VP).


/* NP --) PRON, RC  (--)  NP --) PRON, RC */
trans_np([[ppron,PER_PRON,SFEAT],[rclause,REL,[vp|VP]]],
         [[ppron,S_PER_PRON,SFEAT],[rclause,S_REL,[vp|S_VP]]]):-
    trans_pron(PER_PRON,SFEAT,S_PER_PRON),
    trans_rel(REL,VP,S_REL,S_VP).


/* NP --) DET, CN, RC  (--)  NP --) DET, CN, RC */
trans_np([[det,DET,DET_SUBFEAT],[com_noun,COM_NOUN,CN_SUBFEAT],[rclause,REL,[vp|VP]]],
         [[det,S_DET,DET_SUBFEAT],[com_noun,S_COM_NOUN,CN_SUBFEAT],[rclause,S_REL,[vp|S_VP]]]):-
    trans_com(COM_NOUN,CN_SUBFEAT,S_COM_NOUN),
    trans_det(COM_NOUN,DET,DET_SUBFEAT,S_DET),
    trans_rel(REL,VP,S_REL,S_VP).

/* NP --) DET, ADJ, CN, RC  (--)  NP --) DET, CN, ADJ, RC */
trans_np([[det,DET,DET_SUBFEAT],[adj,ADJ,ADJ_SUBFEAT],[com_noun,COM_NOUN,CN_SUBFEAT],
          [rclause,REL,[vp|VP]]],
         [[det,S_DET,DET_SUBFEAT],[com_noun,S_COM_NOUN,CN_SUBFEAT],[adj,S_ADJ,ADJ_SUBFEAT],
          [rclause,S_REL,[vp|S_VP]]]):-
    trans_com(COM_NOUN,CN_SUBFEAT,S_COM_NOUN),
    trans_det(COM_NOUN,DET,DET_SUBFEAT,S_DET),
    trans_adj(ADJ,ADJ_SUBFEAT,S_ADJ),
    trans_rel(REL,VP,S_REL,S_VP).
```

```
/* NP --) Q, CN, RC  (--)  NP --) Q, CN, RC */
trans_np([[quant,QUANT,QUANT_SUBFEAT],[com_noun,COM_NOUN,CN_SUBFEAT],
          [rclause,REL,[vp|VP]]],
         [[quant,S_QUANT,QUANT_SUBFEAT],[com_noun,S_COM_NOUN,CN_SUBFEAT],
          [rclause,S_REL,[vp|S_VP]]]):-
    trans_quant(QUANT,QUANT_SUBFEAT,S_QUANT),
    trans_com(COM_NOUN,CN_SUBFEAT,S_COM_NOUN),
    trans_rel(REL,VP,S_REL,S_VP).

/* NP --) Q, ADJ, CN, RC  (--)  Q, CN, ADJ, RC */
trans_np([[quant,QUANT,QUANT_SUBFEAT],[adj,ADJ,ADJ_SUBFEAT],[com_noun,COM_NOUN,CN_SUBFEAT],
          [rclause,REL,[vp|VP]]],
         [[quant,S_QUANT,QUANT_SUBFEAT],[com_noun,S_COM_NOUN,CN_SUBFEAT],[adj,S_ADJ,ADJ_SUBFEAT],
          [rclause,S_REL,[vp|S_VP]]]):-
    trans_quant(QUANT,QUANT_SUBFEAT,S_QUANT),
    trans_com(COM_NOUN,CN_SUBFEAT,S_COM_NOUN),
    trans_adj(ADJ,ADJ_SUBFEAT,S_ADJ),
    trans_rel(REL,VP,S_REL,S_VP).

/* NP --) PN, RC, PP  (--)  NP --) PN, RC, PP */
trans_np([[pn,PROP_NOUN,SFEAT],[rclause,REL,[vp|VP]],[prep_ph|PREP_PH]],
         [[pn,S_PROP_NOUN,SFEAT],[rclause,S_REL,[vp|S_VP]],[prep_ph|S_PREP_PH]]):-
    trans_pn(PROP_NOUN,SFEAT,S_PROP_NOUN),
    trans_rel(REL,VP,S_REL,S_VP),
    trans_prep_ph(PREP_PH,S_PREP_PH).

/* NP --) PRON, RC, PP  (--)  NP --) PRON, RC, PP */
trans_np([[ppron,PER_PRON,SFEAT],[rclause,REL,[vp|VP]],[prep_ph|PREP_PH]],
         [[ppron,S_PER_PRON,SFEAT],[rclause,S_REL,[vp|S_VP]],[prep_ph|S_PREP_PH]]):-
    trans_pron(PER_PRON,SFEAT,S_PER_PRON),
    trans_rel(REL,VP,S_REL,S_VP),
    trans_prep_ph(PREP_PH,S_PREP_PH).

/* NP --) DET, CN, RC, PP  (--)  NP --) DET, CN, RC, PP */
trans_np([[det,DET,DET_SUBFEAT],[com_noun,COM_NOUN,CN_SUBFEAT],[rclause,REL,[vp|VP]],
          [prep_ph|PREP_PH]],
         [[det,S_DET,DET_SUBFEAT],[com_noun,S_COM_NOUN,CN_SUBFEAT],[rclause,S_REL,[vp|S_VP]],
          [prep_ph|S_PREP_PH]]):-
    trans_com(COM_NOUN,CN_SUBFEAT,S_COM_NOUN),
    trans_det(COM_NOUN,DET,DET_SUBFEAT,S_DET),
    trans_rel(REL,VP,S_REL,S_VP),
    trans_prep_ph(PREP_PH,S_PREP_PH).

/* NP --) DET, ADJ, CN, RC, PP  (--)  NP --) DET, CN, ADJ, RC, PP */
trans_np([[det,DET,DET_SUBFEAT],[adj,ADJ,ADJ_SUBFEAT],[com_noun,COM_NOUN,CN_SUBFEAT],
          [rclause,REL,[vp|VP]],[prep_ph|PREP_PH]],
         [[det,S_DET,DET_SUBFEAT],[com_noun,S_COM_NOUN,CN_SUBFEAT],[adj,S_ADJ,ADJ_SUBFEAT],
          [rclause,S_REL,[vp|S_VP]],[prep_ph|S_PREP_PH]]):-
    trans_com(COM_NOUN,CN_SUBFEAT,S_COM_NOUN),
    trans_det(COM_NOUN,DET,DET_SUBFEAT,S_DET),
    trans_adj(ADJ,ADJ_SUBFEAT,S_ADJ),
    trans_rel(REL,VP,S_REL,S_VP),
    trans_prep_ph(PREP_PH,S_PREP_PH).
```

```
/* NP --> Q, CN, RC, PP  (--)  NP --> Q, CN, RC, PP */
trans_np([[quant,QUANT,QUANT_SUBFEAT],[com_noun,COM_NOUN,CN_SUBFEAT],[rclause,REL,[vp|VP]],
         [prep_ph|PREP_PH]],
        [[quant,S_QUANT,QUANT_SUBFEAT],[com_noun,S_COM_NOUN,CN_SUBFEAT],[rclause,S_REL,[vp|S_VP]],
         [prep_ph|S_PREP_PH]]):-
    trans_quant(QUANT,QUANT_SUBFEAT,S_QUANT),
    trans_com(COM_NOUN,CN_SUBFEAT,S_COM_NOUN),
    trans_rel(REL,VP,S_REL,S_VP),
    trans_prep_ph(PREP_PH,S_PREP_PH).

/* NP --> Q, ADJ, CN, RC, PP  (--)  Q, CN, ADJ, RC, PP */
trans_np([[quant,QUANT,QUANT_SUBFEAT],[adj,ADJ,ADJ_SUBFEAT],[com_noun,COM_NOUN,CN_SUBFEAT],
         [rclause,REL,[vp|VP]],[prep_ph|PREP_PH]],
        [[quant,S_QUANT,QUANT_SUBFEAT],[com_noun,S_COM_NOUN,CN_SUBFEAT],[adj,S_ADJ,ADJ_SUBFEAT],
         [rclause,S_REL,[vp|S_VP]],[prep_ph|S_PREP_PH]]):-
    trans_quant(QUANT,QUANT_SUBFEAT,S_QUANT),
    trans_com(COM_NOUN,CN_SUBFEAT,S_COM_NOUN),
    trans_adj(ADJ,ADJ_SUBFEAT,S_ADJ),
    trans_rel(REL,VP,S_REL,S_VP),
    trans_prep_ph(PREP_PH,S_PREP_PH).


/* VP --> IV */
trans_vp([[iv,ROOT,TENSE,SFEAT]],[[iv,S_ROOT,TENSE,SFEAT]]):-
    trans_iv(ROOT,TENSE,SFEAT,S_ROOT).

/* VP --> IV, ADV  (--)  VP --> IV, ADV */
trans_vp([[iv,ROOT,TENSE,SFEAT],[adv,ADV]],
        [[iv,S_ROOT,TENSE,SFEAT],[adv,S_ADV]]):-
    trans_iv(ROOT,TENSE,SFEAT,S_ROOT),
    trans_adv(ADV,S_ADV).

/* VP --> IV, ADVP  (--)  VP --> IV, ADVP */
trans_vp([[iv,ROOT,TENSE,SFEAT],[advp|ADVP]],
        [[iv,S_ROOT,TENSE,SFEAT],[advp|S_ADVP]]):-
    trans_iv(ROOT,TENSE,SFEAT,S_ROOT),
    trans_adv_ph(ADVP,S_ADVP).

/* VP --> IV, PP */
trans_vp([[iv,ROOT,TENSE,SFEAT],[prep_ph|PREP_PH]],
        [[iv,S_ROOT,TENSE,SFEAT],[prep_ph|S_PREP_PH]]):-
    trans_iv(ROOT,TENSE,SFEAT,S_ROOT),
    trans_prep_ph(PREP_PH,S_PREP_PH).

/* VP --> IV, ADV, PP  (--)  VP --> IV, ADV, PP */
trans_vp([[iv,ROOT,TENSE,SFEAT],[adv,ADV],[prep_ph|PREP_PH]],
         [[iv,S_ROOT,TENSE,SFEAT],[adv,S_ADV],[prep_ph|S_PREP_PH]]):-
    trans_iv(ROOT,TENSE,SFEAT,S_ROOT),
    trans_adv(ADV,S_ADV),
    trans_prep_ph(PREP_PH,S_PREP_PH).
```

```
/* VP --) IV, ADVP, PP  (--)  VP --) IV, ADVP, PP */
trans_vp([[iv,ROOT,TENSE,SFEAT],[advp|ADVP],[prep_ph|PREP_PH]],
         [[iv,S_ROOT,TENSE,SFEAT],[advp|S_ADVP],[prep_ph|S_PREP_PH]]):-
    trans_iv(ROOT,TENSE,SFEAT,S_ROOT),
    trans_adv_ph(ADVP,S_ADVP),
    trans_prep_ph(PREP_PH,S_PREP_PH).

/* VP --) TV, NP  (--)  VP --) TV, NP */
trans_vp([[tv,ROOT,TENSE,SFEAT1,SFEAT2],[np|NP]],
         [[tv,S_ROOT,TENSE,SFEAT1,SFEAT2],[np|S_NP]]):-
    [!
      trans_tv(ROOT,TENSE,SFEAT1,SFEAT2,S_ROOT)
    !],
    trans_check_tv(S_ROOT),
    trans_np(NP,S_NP).

/* VP --) TV, NP  (--)  VP --) TV, CONN, NP */
trans_vp([[tv,ROOT,TENSE,SFEAT1,SFEAT2],[np|NP]],
         [[tv,S_ROOT,TENSE,SFEAT1,SFEAT2],[conn,CONN],[np|S_NP]]):-
    [!
      trans_tv(ROOT,TENSE,SFEAT1,SFEAT2,S_ROOT)
    !],
    not(trans_check_tv(S_ROOT)),
    trans_conn(CONN),
    trans_np(NP,S_NP).

/* VP --) TV, NP, ADV  (--)  VP --) TV, NP, ADV */
trans_vp([[tv,ROOT,TENSE,SFEAT1,SFEAT2],[np|NP],[adv,ADV]],
         [[tv,S_ROOT,TENSE,SFEAT1,SFEAT2],[np|S_NP],[adv,S_ADV]]):-
    [!
      trans_tv(ROOT,TENSE,SFEAT1,SFEAT2,S_ROOT)
    !],
    trans_check_tv(S_ROOT),
    trans_np(NP,S_NP),
    trans_adv(ADV,S_ADV).

/* VP --) TV, NP, ADV  (--)  VP --) TV, CONN, NP, ADV */
trans_vp([[tv,ROOT,TENSE,SFEAT1,SFEAT2],[np|NP],[adv,ADV]],
         [[tv,S_ROOT,TENSE,SFEAT1,SFEAT2],[conn,CONN],[np|S_NP],[adv,S_ADV]]):-
    [!
      trans_tv(ROOT,TENSE,SFEAT1,SFEAT2,S_ROOT)
    !],
    not(trans_check_tv(S_ROOT)),
    trans_conn(CONN),
    trans_np(NP,S_NP),
    trans_adv(ADV,S_ADV).

/* VP --) TV, NP, ADVP  (--)  VP --) TV, NP, ADVP */
trans_vp([[tv,ROOT,TENSE,SFEAT1,SFEAT2],[np|NP],[advp|ADVP]],
         [[tv,S_ROOT,TENSE,SFEAT1,SFEAT2],[np|S_NP],[advp|S_ADVP]]):-
    [!
      trans_tv(ROOT,TENSE,SFEAT1,SFEAT2,S_ROOT)
    !],
```

```
    trans_check_tv(S_ROOT),
    trans_np(NP,S_NP),
    trans_adv_ph(ADVP,S_ADVP).

/* VP --) TV, NP, ADVP  (--)  VP --) TV, CONN, NP, ADVP */
trans_vp([[tv,ROOT,TENSE,SFEAT1,SFEAT2],[np|NP],[advp|ADVP]],
         [[tv,S_ROOT,TENSE,SFEAT1,SFEAT2],[conn,CONN],[np|S_NP],[advp|S_ADVP]]):-
    [!
      trans_tv(ROOT,TENSE,SFEAT1,SFEAT2,S_ROOT)
    !],
    not(trans_check_tv(S_ROOT)),
    trans_conn(CONN),
    trans_np(NP,S_NP),
    trans_adv_ph(ADVP,S_ADVP).

/* VP --) TV, NP, PP  (--)  VP --) TV, NP, PP */
trans_vp([[tv,ROOT,TENSE,SFEAT1,SFEAT2],[np|NP],[prep_ph|PREP_PH]],
         [[tv,S_ROOT,TENSE,SFEAT1,SFEAT2],[np|S_NP],[prep_ph|S_PREP_PH]]):-
    [!
      trans_tv(ROOT,TENSE,SFEAT1,SFEAT2,S_ROOT)
    !],
    trans_check_tv(S_ROOT),
    trans_np(NP,S_NP),
    trans_prep_ph(PREP_PH,S_PREP_PH).

/* VP --) TV, NP, PP  (--)  VP --) TV, CONN, NP, PP */
trans_vp([[tv,ROOT,TENSE,SFEAT1,SFEAT2],[np|NP],[prep_ph|PREP_PH]],
         [[tv,S_ROOT,TENSE,SFEAT1,SFEAT2],[conn,CONN],[np|S_NP],[prep_ph|S_PREP_PH]]):-
    [!
      trans_tv(ROOT,TENSE,SFEAT1,SFEAT2,S_ROOT)
    !],
    not(trans_check_tv(S_ROOT)),
    trans_conn(CONN),
    trans_np(NP,S_NP),
    trans_prep_ph(PREP_PH,S_PREP_PH).

/* VP --) TV, NP, ADV, PP  (--)  VP --) TV, NP, ADV, PP */
trans_vp([[tv,ROOT,TENSE,SFEAT1,SFEAT2],[np|NP],[adv,ADV],[prep_ph|PREP_PH]],
         [[tv,S_ROOT,TENSE,SFEAT1,SFEAT2],[np|S_NP],[adv,S_ADV],[prep_ph|S_PREP_PH]]):-
    [!
      trans_tv(ROOT,TENSE,SFEAT1,SFEAT2,S_ROOT)
    !],
    trans_check_tv(S_ROOT),
    trans_np(NP,S_NP),
    trans_adv(ADV,S_ADV),
    trans_prep_ph(PREP_PH,S_PREP_PH).

/* VP --) TV, NP, ADV, PP  (--)  VP --) TV, CONN, NP, ADV, PP */
trans_vp([[tv,ROOT,TENSE,SFEAT1,SFEAT2],[np|NP],[adv,ADV],[prep_ph|PREP_PH]],
         [[tv,S_ROOT,TENSE,SFEAT1,SFEAT2],[conn,CONN],[np|S_NP],[adv,S_ADV],[prep_ph|S_PREP_PH]]):-
    [!
      trans_tv(ROOT,TENSE,SFEAT1,SFEAT2,S_ROOT)
    !],
```

```
    not(trans_check_tv(S_ROOT)),
    trans_conn(CONN),
    trans_np(NP,S_NP),
    trans_adv(ADV,S_ADV),
    trans_prep_ph(PREP_PH,S_PREP_PH).

/* VP --) TV, NP, ADVP, PP  (--)  VP --) TV, NP, ADVP, PP */
trans_vp([[tv,ROOT,TENSE,SFEAT1,SFEAT2],[np|NP],[advp|ADVP],[prep_ph|PREP_PH]],
        [[tv,S_ROOT,TENSE,SFEAT1,SFEAT2],[np|S_NP],[advp|S_ADVP],[prep_ph|S_PREP_PH]]):-
    ['
      trans_tv(ROOT,TENSE,SFEAT1,SFEAT2,S_ROOT)
    '],
    trans_check_tv(S_ROOT),
    trans_np(NP,S_NP),
    trans_adv_ph(ADVP,S_ADVP),
    trans_prep_ph(PREP_PH,S_PREP_PH).

/* VP --) TV, NP, ADVP, PP  (--)  VP --) TV, CONN, NP, ADVP, PP */
trans_vp([[tv,ROOT,TENSE,SFEAT1,SFEAT2],[np|NP],[advp|ADVP],[prep_ph|PREP_PH]],
        [[tv,S_ROOT,TENSE,SFEAT1,SFEAT2],[conn,CONN],[np|S_NP],[advp|S_ADVP],[prep_ph|S_PREP_PH]]):-
    ['
      trans_tv(ROOT,TENSE,SFEAT1,SFEAT2,S_ROOT)
    '],
    not(trans_check_tv(S_ROOT)),
    trans_conn(CONN),
    trans_np(NP,S_NP),
    trans_adv_ph(ADVP,S_ADVP),
    trans_prep_ph(PREP_PH,S_PREP_PH).


/* PN */
trans_pn(W,[_,_|SSFEAT],SW):-
    lex_pn(W,SSFEAT,SW).


/* PRON */
trans_pron(W,SFEAT,SW):-
    lex_pron(W,SFEAT,SW).


/* DET */
trans_det(_,W,SFEAT,SW):-
    ['
      lex_det(W,SFEAT,SW)
    '],
    W \== 'a',
    W \== 'an'.

trans_det(CN,W,SFEAT,SW):-
    lex_det(W,SFEAT,SW),
    W == 'an',
    atom_string(CN,STRING),
    nth_char(0,STRING,ASCII),
```

```
    case([ASCII == 97 -> true,
          ASCII == 101 -> true,
          ASCII == 105 -> true,
          ASCII == 111 -> true,
          ASCII == 117 -> true| fail]).

trans_det(_,W,SFEAT,SW):-
    lex_det(W,SFEAT,SW).


/* CN */
trans_com(W,SFEAT,SW):-
    lex_com_noun(W,SFEAT,SW).


/* ADJ */
trans_adj(W,SFEAT,SW):-
    lex_adj(W,SFEAT,SW).


/* Q */
trans_quant(W,SFEAT,SW):-
    lex_quant(W,SFEAT,SW).

/* RC */
trans_rel(W,VP,SW,S_VP):-
    lex_rel(W,SW),
    trans_vp(VP,S_VP).

/* PP */
trans_prep_ph([[prep,PREP,SFEAT],[np|NP]],[[prep,S_PREP,SFEAT],[np|S_NP]]):-
    [!
      lex_prep(PREP,SFEAT,S_PREP)
    !],
    PREP == 'in',
    trans_np(NP,S_NP).

trans_prep_ph([[prep,PREP,SFEAT],[np|NP]],[[prep,S_PREP,SFEAT],[np|S_NP]]):-
    [!
      PREP == 'with',
      get_elem(NP,ELEM)
    !],
    ELEM \== 'me',
    ELEM \== 'you',
    lex_prep(PREP,SFEAT,S_PREP),
    trans_np(NP,S_NP).

trans_prep_ph([[prep,PREP,SFEAT],[np|NP]],[[prep,S_PREP,SFEAT],[np|S_NP]]):-
    [!
      S_PREP == 'con',
      get_elem(S_NP,ELEM),
      trans_check_pron(ELEM,PPRON),
      tail(S_NP,TAIL)
```

```
    !),
    TAIL == [],
    trans_np_elem(PPRON,_,NP,S_NP),
    lex_prep(PREP,SFEAT,S_PREP).

trans_prep_ph([[prep,PREP,SFEAT],[np|NP]],[[np|S_NP]]):-
    [!
      trans_np(NP,S_NP),
      get_elem(S_NP,ELEM)
    !],
    ELEM = 'conmigo',
    get_sfeat(NP,NP_SFEAT),
    same(NP_SFEAT,SFEAT),
    lex_prep(PREP,SFEAT,ELEM).

trans_prep_ph([[prep,PREP,SFEAT],[np|NP]],[[np|S_NP]]):
    [!
      get_elem(NP,ELEM),
      ELEM = 'you',
      trans_np_elem(_,'contigo',NP,S_NP),
      get_sfeat(NP,NP_SFEAT)
    !],
    same(NP_SFEAT,SFEAT),
    lex_prep(PREP,SFEAT,'contigo').

trans_prep_ph([[prep,PREP,SFEAT],[np|NP]],[[prep,S_PREP,SFEAT],[np|S_NP]]):-
    lex_prep(PREP,SFEAT,S_PREP),
    S_PREP == 'con',
    trans_np(NP,S_NP).


get_elem([FIRST|_],ELEM):-
    tail(FIRST,TEMP),
    first(TEMP,ELEM).

trans_check_pron(ELEM,PPRON):-
    case([ELEM == 'el' -> PPRON = 'him',
          ELEM == 'ella' -> PPRON = 'her',
          ELEM == 'nosotros' -> PPRON = 'us',
          ELEM == 'nosotras' -> PPRON = 'us',
          ELEM == 'ellos' -> PPRON = 'them',
          ELEM == 'ellas' -> PPRON = 'them' | false]).

trans_np_elem(PER_PRON,S_PER_PRON,[[ppron,PER_PRON,SFEAT]],[[ppron,S_PER_PRON,SFEAT]]):-
    trans_pron(PER_PRON,SFEAT,S_PER_PRON).

get_sfeat([LIST],SFEAT):-
    last(LIST,SFEAT).


/* IV s/
trans_iv(W,TENSE,SFEAT,SW):-
    lex_iv(W,_,TENSE,SFEAT,SW,_).
```

```
/* ADV */
trans_adv(W,SW):-
    lex_adv(W,SW).


/* ADVP */
trans_adv_ph([[deg,DEG],[adv,ADV]],[[deg,S_DEG],[adv,S_ADV]]):-
    trans_deg(DEG,S_DEG),
    trans_adv(ADV,S_ADV).


/* DEG */
trans_deg(W,SW):-
    lex_deg(W,SW).


/* TV */
trans_tv(W,TENSE,SFEAT1,SFEAT2,SW):-
    lex_tv(W,_,TENSE,SFEAT1,SFEAT2,SW,_).


/* check if not (a) connective needed */
trans_check_tv(S_ROOT):-
    case([S_ROOT == romper -) true,
          S_ROOT == comprar -) true,
          S_ROOT == limpiar -) true,
          S_ROOT == cortar -) true,
          S_ROOT == comer -) true,
          S_ROOT == estudiar -) true| fail]).


/* CONN */
trans_conn(W):-
    lex_conn(_,W).


/* AUX */
trans_aux([NP],[ROOT,TENSE],[S_ROOT,TENSE]):-
    [!
      last(NP,SFEAT)
    !],
    lex_aux(ROOT,_,TENSE,SFEAT,S_ROOT,_).
```

```
/*************************************************************/
/*                                                           */
/*    Traductor Sintáctico-Semántico Bidireccional           */
/*    Inglés-Español Basado en Gramáticas de Cláusula        */
/*    Definida (DCG) y Gramáticas de Estructura de           */
/*    Frase Generalizada (GPSG).                             */
/*                                                           */
/*    PARTE 4: Analizador Sintáctico-Semántico               */
/*             Español.                                      */
/*                                                           */
/*                                                           */
/*                          Alfredo Hasanao D. Maeda         */
/*                          Ing. en Computación, UNAM        */
/*                          Abril, 1988     Méx. D.F.        */
/*                                                           */
/*************************************************************/



/* ***************************************** */
/*                                           */
/* This routine performs the syntactic-      */
/* semantic analysis based on DCG and GPSG.  */
/*                                           */


/*             */
/* Spanish Parser */
/*             */


spanish_parser(SENTENCE,TREE):-
    sp_sentence(TREE,SENTENCE,[]).

spanish_parser(SENTENCE,TREE):-
    tmove(18,3),
    wa(39,118),
    write(@ **** Oraci"n Española incorrecta **** @),
    tmove(20,3),
    wa(3,148),
    write('(e) '),
    write_list(SENTENCE),
    write(@[ENTER]@),
    read_line(0,_),
    tscroll(0,(7,2),(22,78)),
    flush,
    !, fail.
```

```
/*          */
/* Sentence */
/*          */

/* S */
sp_sentence([fin],[fin],_).          /* used only to finish */

sp_sentence([s,NP,VP],S0,S):-
    sp_noun_phrase(NP,S0,S1),
    sp_verb_phrase(NP,VP,S1,S).

sp_sentence([s,NP,AUX,VP],S0,S):-    /* (VP (- gerund mode) */
    [!
      sp_noun_phrase(NP,S0,S1),
      sp_auxiliary(NP,AUX,S1,S2)
    !],
    sp_verb_phrase(NP,VP,S2,S),
    check_gerund(VP).

/*             */
/* Noun Phrase */
/*             */

/* NP --) PN */
sp_noun_phrase([np,PROP_NOUN],S0,S):-
    sp_proper_noun(PROP_NOUN,S0,S).

/* NP --) PN, RC */
sp_noun_phrase([np,PROP_NOUN,REL],S0,S):-
    sp_proper_noun(PROP_NOUN,S0,S1),
    sp_relative_clause(PROP_NOUN,REL,S1,S).

/* NP --) PN, RC, PP */
sp_noun_phrase([np,PROP_NOUN,REL,PREP_PH],S0,S):-
    sp_proper_noun(PROP_NOUN,S0,S1),
    sp_relative_clause(PROP_NOUN,REL,S1,S2),
    sp_prepositional_phrase(REL,PREP_PH,S2,S).

/* NP --) PRON, RC */
sp_noun_phrase([np,PER_PRON,REL],S0,S):-
    sp_personal_pronoun(PER_PRON,S0,S1),
    sp_relative_clause(PER_PRON,REL,S1,S).

/* NP --) PRON, RC, PP */
sp_noun_phrase([np,PER_PRON,REL,PREP_PH],S0,S):-
    sp_personal_pronoun(PER_PRON,S0,S1),
    sp_relative_clause(PER_PRON,REL,S1,S2),
    sp_prepositional_phrase(REL,PREP_PH,S2,S).

/* NP --) DET, CN */
sp_noun_phrase([np,DET,COM_NOUN],S0,S):-
    sp_determiner(DET,S0,S1),
    sp_common_noun(DET,COM_NOUN,S1,S).
```

```
/* NP --> DET, CN, ADJ */
sp_noun_phrase([np,DET,COM_NOUN,ADJ],S0,S):-
    sp_determiner(DET,S0,S1),
    sp_common_noun(DET,COM_NOUN,S1,S2),
    sp_adjective(COM_NOUN,ADJ,S2,S).

/* NP --> DET, CN, RC */
sp_noun_phrase([np,DET,COM_NOUN,REL],S0,S):-
    sp_determiner(DET,S0,S1),
    sp_common_noun(DET,COM_NOUN,S1,S2),
    sp_relative_clause(COM_NOUN,REL,S2,S).

/* NP --> DET, CN, ADJ, RC */
sp_noun_phrase([np,DET,COM_NOUN,ADJ,REL],S0,S):-
    sp_determiner(DET,S0,S1),
    sp_common_noun(DET,COM_NOUN,S1,S2),
    sp_adjective(COM_NOUN,ADJ,S2,S3),
    sp_relative_clause(COM_NOUN,REL,S3,S).

/* NP --> DET, CN, RC, PP */
sp_noun_phrase([np,DET,COM_NOUN,REL,PREP_PHI],S0,S):-
    sp_determiner(DET,S0,S1),
    sp_common_noun(DET,COM_NOUN,S1,S2),
    sp_relative_clause(COM_NOUN,REL,S2,S3),
    sp_prepositional_phrase(REL,PREP_PH,S3,S).

/* NP --> DET, CN, ADJ, RC, PP */
sp_noun_phrase([np,DET,COM_NOUN,ADJ,REL,PREP_PHI],S0,S):-
    sp_determiner(DET,S0,S1),
    sp_common_noun(DET,COM_NOUN,S1,S2),
    sp_adjective(COM_NOUN,ADJ,S2,S3),
    sp_relative_clause(COM_NOUN,REL,S3,S4),
    sp_prepositional_phrase(REL,PREP_PH,S4,S).

/* NP --> PRON */
sp_noun_phrase([np,PER_PRON],S0,S):-
    sp_personal_pronoun(PER_PRON,S0,S).

/* NP --> Q, CN */
sp_noun_phrase([np,QUANT,COM_NOUN],S0,S):-
    sp_quantifier(QUANT,S0,S1),
    sp_common_noun(QUANT,COM_NOUN,S1,S).

/* NP --> Q, CN, ADJ */
sp_noun_phrase([np,QUANT,COM_NOUN,ADJ],S0,S):-
    sp_quantifier(QUANT,S0,S1),
    sp_common_noun(QUANT,COM_NOUN,S1,S2),
    sp_adjective(COM_NOUN,ADJ,S2,S).

/* NP --> Q, CN, RC */
sp_noun_phrase([np,QUANT,COM_NOUN,REL],S0,S):-
    sp_quantifier(QUANT,S0,S1),
```

```
    sp_common_noun(QUANT,COM_NOUN,S1,S2),
    sp_relative_clause(COM_NOUN,REL,S2,S).

/* NP --) Q, CN, ADJ, RC */
sp_noun_phrase([np,QUANT,COM_NOUN,ADJ,REL],S0,S):-
    sp_quantifier(QUANT,S0,S1),
    sp_common_noun(QUANT,COM_NOUN,S1,S2),
    sp_adjective(COM_NOUN,ADJ,S2,S3),
    sp_relative_clause(COM_NOUN,REL,S3,S).

/* NP --) Q, CN, RC, PP */
sp_noun_phrase([np,QUANT,COM_NOUN,REL,PREP_PH],S0,S):-
    sp_quantifier(QUANT,S0,S1),
    sp_common_noun(QUANT,COM_NOUN,S1,S2),
    sp_relative_clause(COM_NOUN,REL,S2,S3),
    sp_prepositional_phrase(REL,PREP_PH,S3,S).

/* NP --) Q, CN, ADJ, RC, PP */
sp_noun_phrase([np,QUANT,COM_NOUN,ADJ,REL,PREP_PH],S0,S):-
    sp_quantifier(QUANT,S0,S1),
    sp_common_noun(QUANT,COM_NOUN,S1,S2),
    sp_adjective(COM_NOUN,ADJ,S2,S3),
    sp_relative_clause(COM_NOUN,REL,S3,S4),
    sp_prepositional_phrase(REL,PREP_PH,S4,S).

/*            */
/* Verb Phrase */
/*            */

/* VP --) IV */
sp_verb_phrase(NP,[vp,IV],S0,S):-
    sp_intransitive_verb(NP,IV,S0,S).

/* VP --) IV, ADV */
sp_verb_phrase(NP,[vp,IV,ADV],S0,S):-
    sp_intransitive_verb(NP,IV,S0,S1),
    sp_adverb(ADV,S1,S).

/* VP --) IV, ADVP */
sp_verb_phrase(NP,[vp,IV,ADVP],S0,S):-
    sp_intransitive_verb(NP,IV,S0,S1),
    sp_adverbial_phrase(ADVP,S1,S).

/* VP --) IV, PP */
sp_verb_phrase(NP,[vp,IV,PREP_PH],S0,S):-
    sp_intransitive_verb(NP,IV,S0,S1),
    sp_prepositional_phrase(IV,PREP_PH,S1,S).

/* VP --) IV, ADV, PP */
sp_verb_phrase(NP,[vp,IV,ADV,PREP_PH],S0,S):-
    sp_intransitive_verb(NP,IV,S0,S1),
    sp_adverb(ADV,S1,S2),
    sp_prepositional_phrase(IV,PREP_PH,S2,S).
```

```
/* VP --> IV, ADVP, PP */
sp_verb_phrase(NP, [vp, IV, ADVP, PREP_PH], S0, S):-
    sp_intransitive_verb(NP, IV, S0, S1),
    sp_adverbial_phrase(ADVP, S1, S2),
    sp_prepositional_phrase(IV, PREP_PH, S2, S).


/* VP --> TV, NP  (not(TV = 'a' connective)) (not(NP1=NP2)) (TV (-) NP2) */
sp_verb_phrase(NP1, [vp, TV, NP2], S0, S):-
    [!
      sp_transitive_verb(NP1, TV, S0, S1),
      check_tv(TV),
      sp_noun_phrase(NP2, S1, S)
    !],
    not(same(NP1, NP2)),
    match_tv_np(TV, NP2).

/* VP --> TV, CONN, NP  (TV = 'a' connective) (not(NP1=NP2)) (TV (-) NP2) */
sp_verb_phrase(NP1, [vp, TV, CONN, NP2], S0, S):-
    [!
      sp_transitive_verb(NP1, TV, S0, S1),
      not(check_tv(TV)),
      connective(CONN, S1, S2),
      sp_noun_phrase(NP2, S2, S)
    !],
    not(same(NP1, NP2)),
    match_tv_np(TV, NP2).

/* VP --> TV, NP, ADV  (not(TV = 'a' connective)) (not(NP1=NP2)) (TV (-) NP2) */
sp_verb_phrase(NP1, [vp, TV, NP2, ADV], S0, S):-
    [!
      sp_transitive_verb(NP1, TV, S0, S1),
      check_tv(TV),
      sp_noun_phrase(NP2, S1, S2)
    !],
    not(same(NP1, NP2)),
    match_tv_np(TV, NP2),
    sp_adverb(ADV, S2, S).

/* VP --> TV, CONN, NP, ADV  (TV = 'a' connective) (not(NP1=NP2)) (TV (-) NP2) */
sp_verb_phrase(NP1, [vp, TV, CONN, NP2, ADV], S0, S):-
    [!
      sp_transitive_verb(NP1, TV, S0, S1),
      not(check_tv(TV)),
      connective(CONN, S1, S2),
      sp_noun_phrase(NP2, S2, S3)
    !],
    not(same(NP1, NP2)),
    match_tv_np(TV, NP2),
    sp_adverb(ADV, S3, S).
```

```
/* VP --) TV, NP, ADVP  (not(TV =) 'a' connective)) (not(NP1=NP2)) (TV (-) NP2) */
sp_verb_phrase(NP1,[vp,TV,NP2,ADVP],S0,S):-
    [!
      sp_transitive_verb(NP1,TV,S0,S1),
      check_tv(TV),
      sp_noun_phrase(NP2,S1,S2)
    !],
    not(same(NP1,NP2)),
    match_tv_np(TV,NP2),
    sp_adverbial_phrase(ADVP,S2,S).

/* VP --) TV, CONN, NP, ADVP  (TV =) 'a' connective) (not(NP1=NP2)) (TV (-) NP2) */
sp_verb_phrase(NP1,[vp,TV,CONN,NP2,ADVP],S0,S):-
    [!
      sp_transitive_verb(NP1,TV,S0,S1),
      not(check_tv(TV)),
      connective(CONN,S1,S2),
      sp_noun_phrase(NP2,S2,S3)
    !],
    not(same(NP1,NP2)),
    match_tv_np(TV,NP2),
    sp_adverbial_phrase(ADVP,S3,S).

/* VP --) TV, NP, PP  (not(TV =) 'a' connective)) (not(NP1=NP2)) (TV (-) NP2) */
sp_verb_phrase(NP1,[vp,TV,NP2,PREP_PH],S0,S):-
    [!
      sp_transitive_verb(NP1,TV,S0,S1),
      check_tv(TV),
      sp_noun_phrase(NP2,S1,S2)
    !],
    not(same(NP1,NP2)),
    match_tv_np(TV,NP2),
    sp_prepositional_phrase(TV,PREP_PH,S2,S).

/* VP --) TV, CONN, NP, PP  (TV =) 'a' connective) (not(NP1=NP2)) (TV (-) NP2) */
sp_verb_phrase(NP1,[vp,TV,CONN,NP2,PREP_PH],S0,S):-
    [!
      sp_transitive_verb(NP1,TV,S0,S1),
      not(check_tv(TV)),
      connective(CONN,S1,S2),
      sp_noun_phrase(NP2,S2,S3)
    !],
    not(same(NP1,NP2)),
    match_tv_np(TV,NP2),
    sp_prepositional_phrase(TV,PREP_PH,S3,S).

/* VP --) TV, NP, ADV, PP  (not(TV =) 'a' connective)) (not(NP1=NP2)) (TV (-) NP2) */
sp_verb_phrase(NP1,[vp,TV,NP2,ADV,PREP_PH],S0,S):-
    [!
      sp_transitive_verb(NP1,TV,S0,S1),
      check_tv(TV),
      sp_noun_phrase(NP2,S1,S2)
    !],
```

```
    not(same(NP1,NP2)),
    match_tv_np(TV,NP2),
    sp_adverb(ADV,S2,S3),
    sp_prepositional_phrase(TV,PREP_PH,S3,S).

/* VP --> TV, CONN, NP, ADV, PP  (TV = 'a' connective) (not(NP1=NP2)) (TV (-) NP2) */
sp_verb_phrase(NP1,[vp,TV,CONN,NP2,ADV,PREP_PH],S0,S):-
    [!
      sp_transitive_verb(NP1,TV,S0,S1),
      not(check_tv(TV)),
      connective(CONN,S1,S2),
      sp_noun_phrase(NP2,S2,S3)
    !],
    not(same(NP1,NP2)),
    match_tv_np(TV,NP2),
    sp_adverb(ADV,S3,S4),
    sp_prepositional_phrase(TV,PREP_PH,S4

/* VP --> TV, NP, ADVP, PP  (not(TV = 'a' connective)) (not(NP1=NP2)) (TV (-) NP2) */
sp_verb_phrase(NP1,[vp,TV,NP2,ADVP,PREP_PH],S0,S):-
    [!
      sp_transitive_verb(NP1,TV,S0,S1),
      check_tv(TV),
      sp_noun_phrase(NP2,S1,S2)
    !],
    not(same(NP1,NP2)),
    match_tv_np(TV,NP2),
    sp_adverbial_phrase(ADVP,S2,S3),
    sp_prepositional_phrase(TV,PREP_PH,S3,S).

/* VP --> TV, CONN, NP, ADVP, PP  (TV = 'a' connective) (not(NP1=NP2)) (TV (-) NP2) */
sp_verb_phrase(NP1,[vp,TV,CONN,NP2,ADVP,PREP_PH],S0,S):-
    [!
      sp_transitive_verb(NP1,TV,S0,S1),
      check_tv(TV),
      connective(CONN,S1,S2),
      sp_noun_phrase(NP2,S2,S3)
    !],
    not(same(NP1,NP2)),
    match_tv_np(TV,NP2),
    sp_adverbial_phrase(ADVP,S3,S4),
    sp_prepositional_phrase(TV,PREP_PH,S4,S).



/*                          */
/* Complementary  Routines */
/*                          */

/* PN */
sp_proper_noun([pn,W,PN_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S)
```

```
    !],
    lex_pn(_,PN_SSFEAT,W),
    append([singular,3],PN_SSFEAT,PN_SUBFEAT).


/* PRON */
sp_personal_pronoun([ppron,W,PPRON_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S)
    !],
    lex_pron(_,PPRON_SUBFEAT,W).


/* DET */
sp_determiner([det,W,DET_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S)
    !],
    lex_sp_det(W,DET_SUBFEAT).


/* CN */
sp_common_noun(ELEM,[com_noun,W,CN_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S),
      last(ELEM,ELEM_SUBFEAT)
    !],
    lex_com_noun(_,CN_SUBFEAT,W),
    same(ELEM_SUBFEAT,CN_SUBFEAT).


/* ADJ */
sp_adjective(ELEM,[adj,W,ADJ_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S),
      last(ELEM,ELEM_SUBFEAT)
    !],
    lex_adj(_,ADJ_SUBFEAT,W),
    same(ELEM_SUBFEAT,ADJ_SUBFEAT).


/* Q */
sp_quantifier([quant,W,Q_SUBFEAT],S0,S):-
    [!
      connects(S0,W,S)
    !],
    lex_quant(_,Q_SUBFEAT,W).


/* RC */
sp_relative_clause(ELEM,[rclause,W,VP],[WIS0],S):-
    lex_rel(_,W),
    sp_verb_phrase([_,ELEM],VP,S0,S).
```

```
/* PP */
sp_prepositional_phrase(REL,[prep_ph,[prep,PREP,PREP_SUBFEAT],NP],S0,S):-
    [!
      connects(S0,PREP,S1),
      PREP = 'con'
    !],
    lex_prep(_,PREP_SUBFEAT,PREP),
    [!
      sp_noun_phrase(NP,S1,S),
      connects(S1,PRON,S2),
      sp_compare_pron(PRON,S2),
      get_subfeat(NP,NP_SUBFEAT)
    !],
    same(PREP_SUBFEAT,NP_SUBFEAT).
    compare_prep_np(PREP_SUBFEAT,NP_SUBFEAT),
    sp_compare_rel_np(REL,NP_SUBFEAT).

sp_prepositional_phrase(REL,[prep_ph,NP],S0,S):-
    [!
      connects(S0,PRON,S),
      PRON = 'conmigo'
    !],
    sp_noun_phrase(NP,S0,S),
    get_subfeat(NP,NP_SUBFEAT),
    sp_compare_rel_np(REL,NP_SUBFEAT).

sp_prepositional_phrase(REL,[prep_ph,NP],S0,S):-
    [!
      connects(S0,PRON,S),
      PRON = 'contigo'
    !],
    sp_noun_phrase(NP,S0,S),
    get_subfeat(NP,NP_SUBFEAT),
    sp_compare_rel_np(REL,NP_SUBFEAT).

sp_prepositional_phrase(IV,[prep_ph,[prep,PREP,PREP_SUBFEAT],NP],S0,S):-
    [!
      connects(S0,PREP,S1),
      PREP = 'con'
    !],
    lex_prep(_,PREP_SUBFEAT,PREP),
    [!
      sp_noun_phrase(NP,S1,S),
      connects(S1,PRON,S2),
      sp_compare_pron(PRON,S2),
      get_subfeat(NP,NP_SUBFEAT)
    !],
    same(PREP_SUBFEAT,NP_SUBFEAT),
    compare_prep_np(PREP_SUBFEAT,NP_SUBFEAT),
    sp_compare_iv_np(IV,NP_SUBFEAT).
```

```
sp_prepositional_phrase(IV,[prep_ph,NP],S0,S):-
    [!
      connects(S0,PRON,S),
      PRON = 'conmigo'
    !],
    sp_noun_phrase(NP,S0,S),
    get_subfeat(NP,NP_SUBFEAT),
    sp_compare_iv_np(IV,NP_SUBFEAT).

sp_prepositional_phrase(IV,[prep_ph,NP],S0,S):-
    [!
      connects(S0,PRON,S),
      PRON = 'contigo'
    !],
    sp_noun_phrase(NP,S0,S),
    get_subfeat(NP,NP_SUBFEAT),
    sp_compare_iv_np(IV,NP_SUBFEAT).

sp_prepositional_phrase(TV,[prep_ph,[prep,PREP,PREP_SUBFEAT],NP],S0,S):-
    [!
      connects(S0,PREP,S1),
      PREP = 'con'
    !],
    lex_prep(_,PREP_SUBFEAT,PREP),
    [!
      sp_noun_phrase(NP,S1,S),
      connects(S1,PRON,S2),
      sp_compare_pron(PRON,S2),
      get_subfeat(NP,NP_SUBFEAT)
    !],
    same(PREP_SUBFEAT,NP_SUBFEAT),
    sp_compare_tv_np(TV,NP_SUBFEAT).

sp_prepositional_phrase(TV,[prep_ph,NP],S0,S):-
    [!
      connects(S0,PRON,S),
      PRON = 'conmigo'
    !],
    sp_noun_phrase(NP,S0,S),
    get_subfeat(NP,NP_SUBFEAT),
    sp_compare_tv_np(TV,NP_SUBFEAT).

sp_prepositional_phrase(TV,[prep_ph,NP],S0,S):-
    [!
      connects(S0,PRON,S),
      PRON = 'contigo'
    !],
    sp_noun_phrase(NP,S0,S),
    get_subfeat(NP,NP_SUBFEAT),
    sp_compare_tv_np(TV,NP_SUBFEAT).
```

```
sp_prepositional_phrase(_,[prep_ph,[prep,PREP,PREP_SUBFEAT],NP],S0,S):-
    [!
      connects(S0,PREP,S1),
      PREP = 'en',
      lex_prep(_,PREP_SUBFEAT,PREP)
    !],
    sp_noun_phrase(NP,S1,S),
    [!
      get_subfeat(NP,NP_SUBFEAT),
      same(PREP_SUBFEAT,NP_SUBFEAT),
      connects(S1,LOC,_)
    !],
    sp_check_loc(LOC).

sp_prepositional_phrase(_,[prep_ph,[prep,PREP,PREP_SUBFEAT],NP],S0,S):-
    [!
      connects(S0,PREP,S1),
      PREP = 'en',
      lex_prep(_,PREP_SUBFEAT,PREP)
    !],
    sp_noun_phrase(NP,S1,S),
    [!
      get_subfeat(NP,NP_SUBFEAT),
      same(PREP_SUBFEAT,NP_SUBFEAT),
      connects(S1,_,S2),
      connects(S2,LOC,_)
    !],
    sp_check_loc(LOC).


/* check syntactic concordance between 'with' PREP and PRON */
sp_compare_pron(PRON,S):-                          /* for correct use of */
    [!                                             /* 'with' preposition */
    connects(S,CN,_)
    !],
    case([PRON == 'yo' -> fail,
          PRON == 'tu' -> lex_com_noun(_,_,CN)|true]).

sp_compare_pron(PRON,NEXT):-
    [!
      NEXT == []
    !],
    case([PRON == 'el' -> true,
          PRON == 'ella' -> true,
          PRON == 'nosotros' -> true,
          PRON == 'nosotras' -> true,
          PRON == 'ellos' -> true,
          PRON == 'ellas' -> true| fail]).

sp_compare_pron(PRON,_):-
    lex_pn(_,_,PRON).
```

```
/* check semantic concordance between NP and PP verbs */
sp_compare_rel_np(REL,NP_SUBFEAT):-
    last(REL,_,TEMP1), !,
    tail(TEMP1,[[_,VERB,_,REL_SUBFEAT]]), !,
    case([VERB == desaparecer -) true,
          VERB == volar -) compare_verb1(REL_SUBFEAT,NP_SUBFEAT),
          VERB == vivir -) compare_verb2(REL_SUBFEAT,NP_SUBFEAT),
          VERB == jugar -) compare_verb3(REL_SUBFEAT,NP_SUBFEAT),
          VERB == correr -) compare_verb1(REL_SUBFEAT,NP_SUBFEAT),
          VERB == dormir -) compare_verb3(REL_SUBFEAT,NP_SUBFEAT),
          VERB == fumar -) compare_verb4(REL_SUBFEAT,NP_SUBFEAT),
          VERB == hablar -) compare_verb4(REL_SUBFEAT,NP_SUBFEAT),
          VERB == cantar -) compare_verb4(REL_SUBFEAT,NP_SUBFEAT),
          VERB == estudiar -) compare_verb4(REL_SUBFEAT,NP_SUBFEAT)| fail]).


/* check semantic concordance between IV and PP verbs */
sp_compare_iv_np([IV,NP_SUBFEAT]):-
    tail([IV,[VERB,_,IV_SUBFEAT]], !,
    case([VERB == desaparecer -) true,
          VERB == volar -) compare_verb1(REL_SUBFEAT,NP_SUBFEAT),
          VERB == vivir -) compare_verb2(REL_SUBFEAT,NP_SUBFEAT),
          VERB == jugar -) compare_verb3(REL_SUBFEAT,NP_SUBFEAT),
          VERB == correr -) compare_verb1(REL_SUBFEAT,NP_SUBFEAT),
          VERB == dormir -) compare_verb3(REL_SUBFEAT,NP_SUBFEAT),
          VERB == fumar -) compare_verb4(REL_SUBFEAT,NP_SUBFEAT),
          VERB == hablar -) compare_verb4(REL_SUBFEAT,NP_SUBFEAT),
          VERB == cantar -) compare_verb4(REL_SUBFEAT,NP_SUBFEAT),
          VERB == estudiar -) compare_verb4(REL_SUBFEAT,NP_SUBFEAT)| fail]).


/* check semantic concordance between TV and PP verbs */
sp_compare_tv_np([_,VERB,_,_,[_,_,TV_SSFEAT]],[_,_,NP_SSFEAT]):-
    case([VERB == adoptar -) same(NP_SSFEAT,[anim_y,_,_,_,_,_]),
          VERB == admirar -) same(NP_SSFEAT,_),
          VERB == romper -) same(NP_SSFEAT,[anim_n,_,_,_,wing_n,loc_n,_]),
          VERB == comprar -) same(NP_SSFEAT,TV_SSFEAT),
          VERB == atrapar -) same(NP_SSFEAT,[_,_,_,_,_,loc_n,_]),
          VERB == cazar -) same(NP_SSFEAT,[_,_,_,_,_,loc_n,_]),
          VERB == limpiar -) same(NP_SSFEAT,[anim_n,_,_,_,_,_,_]),
          VERB == cortar -) same(NP_SSFEAT,[anim_n,hum_n,masc_y,wing_n,loc_n,meal_n]),
          VERB == defender -) same(NP_SSFEAT,[anim_y,_,_,_,_,_,_]),
          VERB == comer -) same(NP_SSFEAT,[_,_,_,_,_,meal_y]),
          VERB == encontrar -) same(NP_SSFEAT,_),
          VERB == oir -) same(NP_SSFEAT,[_,_,_,_,_,loc_n,meal_n]),
          VERB == escuchar -) same(NP_SSFEAT,[anim_y,_,_,_,_,_,_]),
          VERB == amar -) same(NP_SSFEAT,[_,_,_,_,_,loc_n,_]),
          VERB == ver -) same(NP_SSFEAT,_),
          VERB == estudiar -) same(NP_SSFEAT,[_,hum_n,_,_,_,loc_n,meal_n])| fail]).
```

```
/* check semantic concordance between verb and 'en' PREP */
sp_check_loc(LOC):-
    lex_pn(_,[_,_,_,_,loc_y,_],LOC).

sp_check_loc(LOC):-
    lex_com_noun(_,[_,_,_,_,_,_,loc_y,_],LOC).


/* IV */
sp_intransitive_verb(NP,[iv,ROOT,TENSE,IV_SUBFEAT],S0,S):-
    [!
      get_subfeat(NP,NP_SUBFEAT),
      connects(S0,W,S)
    !],
    lex_iv(_,_,TENSE,IV_SUBFEAT,ROOT,W),
    same(NP_SUBFEAT,IV_SUBFEAT).


/* ADV */
sp_adverb([adv,W],S0,S):-
    [!
      connects(S0,W,S)
    !],
    lex_adv(_,W).


/* ADVP */
sp_adverbial_phrase([advp,DEG,ADV],S0,S):-
    [!
      sp_degree(DEG,S0,S1)
    !],
    sp_adverb(ADV,S1,S).


/* DEG */
sp_degree([deg,W],S0,S):-
    [!
      connects(S0,W,S)
    !],
    lex_deg(_,W).


/* TV */
sp_transitive_verb(NP,[tv,ROOT,TENSE,TV_SUBFEAT1,TV_SUBFEAT2],S0,S):-
    [!
      get_subfeat(NP,NP_SUBFEAT),
      connects(S0,W,S)
    !],
    lex_tv(_,_,TENSE,TV_SUBFEAT1,TV_SUBFEAT2,ROOT,W),
    same(NP_SUBFEAT,TV_SUBFEAT1).
```

```
/* check if 'a' connective not needed */
check_tv([_,VERB|_]):-
    case([(VERB == romper -) true,
          (VERB == comprar -) true,
          (VERB == limpiar -) true,
          (VERB == cortar -) true,
          (VERB == comer -) true,
          (VERB == estudiar -) true| fail]).


/* CONN */
connective([conn,W],S0,S):-
    [!
      connects(S0,W,S)
    !],
    lex_conn(_,W).


/* AUX */
sp_auxiliary(NP,[aux,ROOT,TENSE],S0,S):-
    [!
      get_subfeat(NP,NP_SUBFEAT),
      connects(S0,W,S)
    !],
    lex_aux(_,_,TENSE,AUX_SUBFEAT,ROOT,W),
    same(NP_SUBFEAT,AUX_SUBFEAT).
```

```
/*********************************************************/
/*                                                       */
/*    Traductor Sintáctico-Semántico Bidireccional       */
/*    Inglés-Español Basado en Gramáticas de Cláusula    */
/*    Definida (DCG) y Gramáticas de Estructura de       */
/*    Frase Generalizada (GPSG).                         */
/*                                                       */
/*    PARTE 5: Lexicón.                                  */
/*                                                       */
/*                          Alfredo Masanao D. Maeda     */
/*                          Ing. en Computación, UNAM    */
/*                          Abril, 1988     Méx. D.F.    */
/*                                                       */
/*********************************************************/


/*           */
/* adjectives */
/*           */

lex_adj(american,[singular,3,_,_,masc_y,_,_,_],americano).
lex_adj(american,[singular,3,_,_,masc_n,_,_,_],americana).
lex_adj(american,[plural,3,_,_,masc_y,_,_,_],americanos).
lex_adj(american,[plural,3,_,_,masc_n,_,_,_],americanas).

lex_adj(bad,[singular,3,_,_,masc_y,_,loc_n,_],malo).
lex_adj(bad,[singular,3,_,_,masc_n,_,loc_n,_],mala).
lex_adj(bad,[plural,3,_,_,masc_y,_,loc_n,_],malos).
lex_adj(bad,[plural,3,_,_,masc_n,_,loc_n,_],malas).

lex_adj(beautiful,[singular,3,_,_,masc_y,_,_,_],hermoso).
lex_adj(beautiful,[singular,3,_,_,masc_n,_,_,_],hermosa).
lex_adj(beautiful,[plural,3,_,_,masc_y,_,_,_],hermosos).
lex_adj(beautiful,[plural,3,_,_,masc_n,_,_,_],hermosas).

lex_adj(big,[singular,3,_,_,_,_,_,_],grande).
lex_adj(big,[plural,3,_,_,_,_,_,_],grandes).

lex_adj(black,[singular,3,_,_,masc_y,_,loc_n,meal_n],negro).
lex_adj(black,[singular,3,_,_,masc_n,_,loc_n,meal_n],negra).
lex_adj(black,[plural,3,_,_,masc_y,_,loc_n,meal_n],negros).
lex_adj(black,[plural,3,_,_,masc_n,_,loc_n,meal_n],negras).

lex_adj(good,[singular,3,_,_,masc_y,_,loc_n,_],bueno).
lex_adj(good,[singular,3,_,_,masc_n,_,loc_n,_],buena).
lex_adj(good,[plural,3,_,_,masc_y,_,loc_n,_],buenos).
lex_adj(good,[plural,3,_,_,masc_n,_,loc_n,_],buenas).

lex_adj(intelligent,[singular,3,anim_y,_,_,_,loc_n,meal_n],inteligente).
lex_adj(intelligent,[plural,3,anim_y,_,_,_,loc_n,meal_n],inteligentes).
```

```
lex_adj(little,[singular,3,anim_y,_,_,_,loc_n,meal_n],pequeno).
lex_adj(little,[singular,3,anim_y,_,_,_,loc_n,meal_n],pequena).
lex_adj(little,[plural,3,anim_y,_,_,_,loc_n,meal_n],pequenos).
lex_adj(little,[plural,3,anim_y,_,_,_,loc_n,meal_n],pequenas).

lex_adj(lovely,[singular,3,_,_,_,_,_,_],adorable).
lex_adj(lovely,[plural,3,_,_,_,_,_,_],adorables).

lex_adj(old,[singular,3,_,_,masc_y,_,_,_],viejo).
lex_adj(old,[singular,3,_,_,masc_n,_,_,_],vieja).
lex_adj(old,[plural,3,_,_,masc_y,_,_,_],viejos).
lex_adj(old,[plural,3,_,_,masc_n,_,_,_],viejas).

lex_adj(pretty,[singular,3,_,_,masc_y,_,_,_],bonito).
lex_adj(pretty,[singular,3,_,_,masc_n,_,_,_],bonita).
lex_adj(pretty,[plural,3,_,_,masc_y,_,_,_],bonitos).
lex_adj(pretty,[plural,3,_,_,masc_n,_,_,_],bonitas).

lex_adj(red,[singular,3,anim_n,_,_,_,loc_n,meal_n],rojo).
lex_adj(red,[singular,3,anim_n,_,_,_,loc_n,meal_n],roja).
lex_adj(red,[plural,3,anim_n,_,_,_,loc_n,meal_n],rojos).
lex_adj(red,[plural,3,anim_n,_,_,_,loc_n,meal_n],rojas).

lex_adj(small,[singular,3,anim_n,_,_,_,_,_],chico).
lex_adj(small,[singular,3,anim_n,_,_,_,_,_],chica).
lex_adj(small,[plural,3,anim_n,_,_,_,_,_],chicos).
lex_adj(small,[plural,3,anim_n,_,_,_,_,_],chicas).

lex_adj(tasty,[singular,3,_,hum_n,masc_y,_,loc_n,meal_y],sabroso).
lex_adj(tasty,[singular,3,_,hum_n,masc_n,_,loc_n,meal_y],sabrosa).
lex_adj(tasty,[plural,3,_,hum_n,masc_y,_,loc_n,meal_y],sabrosos).
lex_adj(tasty,[plural,3,_,hum_n,masc_n,_,loc_n,meal_y],sabrosas).

lex_adj(ugly,[singular,3,_,_,masc_y,_,_,_],feo).
lex_adj(ugly,[singular,3,_,_,masc_n,_,_,_],fea).
lex_adj(ugly,[plural,3,_,_,masc_y,_,_,_],feos).
lex_adj(ugly,[plural,3,_,_,masc_n,_,_,_],feas).



/*         */
/* adverbs */
/*         */

lex_adv(carefully,cuidadosamente).
lex_adv(sincerely,sinceramente).
lex_adv(slowly,lentamente).
lex_adv(well,bien).
lex_adv(gladly,alegremente).
```

```
/*                          */
/* auxiliary verb (gerund) */
/*                          */

lex_aux(be,am,present,[singular,1,_,_,_,_,_,_,_],estar,estoy).
lex_aux(be,are,present,[singular,2,_,_,_,_,_,_,_],estar,estas).
lex_aux(be,is,present,[singular,3,_,_,_,_,_,_,_],estar,esta).
lex_aux(be,are,present,[plural,1,_,_,_,_,_,_,_],estar,estamos).
lex_aux(be,are,present,[plural,2,_,_,_,_,_,_,_],estar,estais).
lex_aux(be,are,present,[plural,3,_,_,_,_,_,_,_],estar,estan).

lex_aux(be,was,past,[singular,1,_,_,_,_,_,_,_],estar,estuve).
lex_aux(be,were,past,[singular,2,_,_,_,_,_,_,_],estar,estuviste).
lex_aux(be,was,past,[singular,3,_,_,_,_,_,_,_],estar,estuvo).
lex_aux(be,were,past,[plural,1,_,_,_,_,_,_,_],estar,estuvimos).
lex_aux(be,were,past,[plural,2,_,_,_,_,_,_,_],estar,estuvisteis).
lex_aux(be,were,past,[plural,3,_,_,_,_,_,_,_],estar,estuvieron).



/*               */
/* common nouns */
/*               */

lex_com_noun(airplane,[singular,3,anim_n,hum_n,masc_y,wing_y,loc_n,meal_n],avion).
lex_com_noun(apple,[singular,3,anim_n,hum_n,masc_n,wing_n,loc_n,meal_y],manzana).
lex_com_noun(apples,[plural,3,anim_n,hum_n,masc_n,wing_n,loc_n,meal_y],manzanas).
lex_com_noun(ball,[singular,3,anim_n,hum_n,masc_n,wing_n,loc_n,meal_n],pelota).
lex_com_noun(balls,[plural,3,anim_n,hum_n,masc_n,wing_n,loc_n,meal_n],pelotas).
lex_com_noun(bed,[singular,3,anim_n,hum_n,masc_n,wing_n,loc_y,meal_n],cama).
lex_com_noun(bird,[singular,3,anim_y,hum_n,masc_y,wing_y,loc_n,meal_n],pajaro).
lex_com_noun(birds,[plural,3,anim_y,hum_n,masc_y,wing_y,loc_n,meal_n],pajaros).
lex_com_noun(bird,[singular,3,anim_y,hum_n,masc_y,wing_y,loc_n,meal_n],ave).
lex_com_noun(birds,[plural,3,anim_y,hum_n,masc_n,wing_y,loc_n,meal_n],aves).
lex_com_noun(boy,[singular,3,anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],nino).
lex_com_noun(boys,[plural,3,anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],ninos).
lex_com_noun(burger,[singular,3,anim_n,hum_n,masc_n,wing_n,loc_n,meal_y],hamburguesa).
lex_com_noun(burgers,[plural,3,anim_n,hum_n,masc_n,wing_n,loc_n,meal_y],hamburguesas).
lex_com_noun(butterfly,[singular,3,anim_y,hum_n,masc_n,wing_y,loc_n,meal_n],mariposa).
lex_com_noun(butterflies,[plural,3,anim_y,hum_n,masc_n,wing_y,loc_n,meal_n],mariposas).
lex_com_noun(car,[singular,3,anim_n,hum_n,masc_y,wing_n,loc_n,meal_n],auto).
lex_com_noun(cars,[plural,3,anim_n,hum_n,masc_y,wing_n,loc_n,meal_n],autos).
lex_com_noun(car,[singular,3,anim_n,hum_n,masc_y,wing_n,loc_n,meal_n],coche).
lex_com_noun(cars,[plural,3,anim_n,hum_n,masc_y,wing_n,loc_n,meal_n],coches).
lex_com_noun(cat,[singular,3,anim_y,hum_n,masc_y,wing_n,loc_n,meal_n],gato).
lex_com_noun(cats,[plural,3,anim_y,hum_n,masc_y,wing_n,loc_n,meal_n],gatos).
lex_com_noun(coke,[singular,3,anim_n,hum_n,masc_n,wing_n,loc_n,meal_y],coca_cola).
lex_com_noun(cream,[_,3,anim_n,hum_n,masc_n,wing_n,loc_n,meal_y],crema).
lex_com_noun(dog,[singular,3,anim_y,hum_n,masc_y,wing_n,loc_n,meal_n],perro).
lex_com_noun(dogs,[plural,3,anim_y,hum_n,masc_y,wing_n,loc_n,meal_n],perros).
lex_com_noun(doll,[singular,3,anim_n,hum_n,masc_n,wing_n,loc_n,meal_n],muneca).
lex_com_noun(garden,[singular,3,anim_n,hum_n,masc_y,wing_n,loc_y,meal_n],jardin).
lex_com_noun(girl,[singular,3,anim_y,hum_y,masc_n,wing_n,loc_n,meal_n],nina).
```

```
lex_com_noun(girls, [plural,3,anim_y,hum_y,masc_n,wing_n,loc_n,meal_n],ninas).
lex_com_noun(glass, [singular,3,anim_n,hum_n,masc_y,wing_n,loc_n,meal_n],vidrio).
lex_com_noun(glasses, [singular,3,anim_n,hum_n,masc_y,wing_n,loc_n,meal_n],vidrios).
lex_com_noun(hammer, [singular,3,anim_n,hum_n,masc_y,wing_n,loc_n,meal_n],martillo).
lex_com_noun(house, [singular,3,anim_n,hum_n,masc_n,wing_n,loc_y,meal_n],casa).
lex_com_noun(knife, [singular,3,anim_n,hum_n,masc_y,wing_n,loc_n,meal_n],cuchillo).
lex_com_noun(library, [singular,3,anim_n,hum_n,masc_n,wing_n,loc_y,meal_n],biblioteca).
lex_com_noun(man, [singular,3,anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],hombre).
lex_com_noun(men, [plural,3,anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],hombres).
lex_com_noun(milk_shake, [singular,3,anim_n,hum_n,masc_n,wing_n,loc_n,meal_y],malteada).
lex_com_noun(park, [singular,3,anim_n,hum_n,masc_y,wing_n,loc_y,meal_n],parque).
lex_com_noun(policeman, [singular,3,anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],policia).
lex_com_noun(policemen, [plural,3,anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],policias).
lex_com_noun(school, [singular,3,anim_n,hum_n,masc_n,wing_n,loc_y,meal_n],escuela).
lex_com_noun(salad, [_,3,anim_n,hum_n,masc_n,wing_n,loc_n,meal_y],ensalada).
lex_com_noun(sofa, [_,3,anim_n,hum_n,masc_y,wing_n,loc_y,meal_n],sofa).
lex_com_noun(stone, [singular,3,anim_n,hum_n,masc_n,wing_n,loc_n,meal_n],piedra).
lex_com_noun(student, [singular,3,anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],estudiante).
lex_com_noun(students, [plural,3,anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],estudiantes).
lex_com_noun(taco, [singular,3,anim_n,hum_n,masc_y,wing_n,loc_n,meal_y],taco).
lex_com_noun(tacos, [plural,3,anim_n,hum_n,masc_y,wing_n,loc_n,meal_y],tacos).
lex_com_noun(thief, [singular,3,anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],ladron).
lex_com_noun(toy, [singular,3,anim_n,hum_n,masc_y,wing_n,loc_n,meal_n],juguete).
lex_com_noun(toys, [plural,3,anim_n,hum_n,masc_y,wing_n,loc_n,meal_n],juguetes).
lex_com_noun(university, [singular,3,anim_n,hum_n,masc_n,wing_n,loc_y,meal_n],universidad).
lex_com_noun(woman, [singular,3,anim_y,hum_y,masc_n,wing_n,loc_n,meal_n],mujer).
lex_com_noun(women, [plural,3,anim_y,hum_y,masc_n,wing_n,loc_n,meal_n],mujeres).
lex_com_noun(yard, [singular,3,anim_n,hum_n,masc_y,wing_n,loc_y,meal_n],patio).


/*            */
/* connective */
/*            */

lex_conn(_,a).


/*              */
/* degree adverb */
/*              */

lex_deg(so,tan).
lex_deg(very,muy).


/*             */
/* determiners */
/*             */

lex_det(a, [singular,3,_,_,masc_y,_,_,_,_],un).
lex_det(a, [singular,3,_,_,masc_n,_,_,_,_],una).
lex_det(an, [singular,3,_,_,masc_y,_,_,_,_],un).
lex_det(an, [singular,3,_,_,masc_n,_,_,_,_],una).
```

```
lex_det(all,[singular,3,_,_,masc_y,_,_,_],todo).
lex_det(all,[singular,3,_,_,masc_n,_,_,_],toda).
lex_det(all,[plural,3,_,_,masc_y,_,_,_],todos).
lex_det(all,[plural,3,_,_,masc_n,_,_,_],todas).

lex_det(every,[singular,3,_,_,_,_,_,_],cada).

lex_det(some,[plural,3,_,_,masc_y,_,_,_],algunos).
lex_det(some,[plural,3,_,_,masc_n,_,_,_],algunas).

lex_det(my,[singular,3,_,_,_,_,_,_],mi).
lex_det(your,[singular,3,_,_,_,_,_,_],tu).
lex_det(his,[singular,3,_,_,_,_,_,_],su).
lex_det(her,[singular,3,_,_,_,_,_,_],su).
lex_det(its,[singular,3,_,_,_,_,_,_],su).
lex_det(our,[singular,3,_,_,masc_y,_,_,_],nuestro).
lex_det(our,[singular,3,_,_,masc_n,_,_,_],nuestra).
lex_det(yours,[singular,3,_,_,masc_y,_,_,_],vuestro).
lex_det(yours,[singular,3,_,_,masc_n,_,_,_],vuestra).
lex_det(theirs,[_,3,_,_,_,_,_,_],sus).

lex_det(my,[plural,3,_,_,_,_,_,_],mis).
lex_det(yours,[plural,3,_,_,_,_,_,_],tus).
lex_det(his,[plural,3,_,_,masc_y,_,_,_],sus).
lex_det(her,[plural,3,_,_,masc_n,_,_,_],sus).
lex_det(its,[plural,3,_,_,_,_,_,_],sus).
lex_det(our,[plural,3,_,_,masc_y,_,_,_],nuestros).
lex_det(our,[plural,3,_,_,masc_n,_,_,_],nuestras).
lex_det(yours,[plural,3,_,_,masc_y,_,_,_],vuestros).
lex_det(yours,[plural,3,_,_,masc_n,_,_,_],vuestras).
lex_det(theirs,[plural,3,_,_,_,_,_,_],sus).

lex_det(that,[singular,3,_,_,masc_y,_,_,_],ese).
lex_det(that,[singular,3,_,_,masc_n,_,_,_],esa).
lex_det(those,[plural,3,_,_,masc_y,_,_,_],esos).
lex_det(those,[plural,3,_,_,masc_n,_,_,_],esas).

lex_det(that,[singular,3,_,_,masc_y,_,_,_],aquel).
lex_det(that,[singular,3,_,_,masc_n,_,_,_],aquella).
lex_det(those,[plural,3,_,_,masc_y,_,_,_],aquellos).
lex_det(those,[plural,3,_,_,masc_n,_,_,_],aquellas).

lex_det(this,[singular,3,_,_,masc_y,_,_,_],este).
lex_det(this,[singular,3,_,_,masc_n,_,_,_],esta).
lex_det(these,[plural,3,_,_,masc_y,_,_,_],estos).
lex_det(these,[plural,3,_,_,masc_n,_,_,_],estas).

lex_det(the,[singular,3,_,_,masc_y,_,_,_],el).
lex_det(the,[singular,3,_,_,masc_n,_,_,_],la).
lex_det(the,[plural,3,_,_,masc_y,_,_,_],los).
lex_det(the,[plural,3,_,_,masc_n,_,_,_],las).
```

```
/* english determiners */

lex_eng_det(a,[singular,3,_,_,_,_,_,_]).
lex_eng_det(an,[singular,3,_,_,_,_,_,_]).
lex_eng_det(all,[_,3,_,_,_,_,_,_]).
lex_eng_det(every,[singular,3,_,_,_,_,_,_]).
lex_eng_det(some,[plural,3,_,_,_,_,_,_]).
lex_eng_det(that,[singular,3,_,_,_,_,_,_]).
lex_eng_det(the,[_,3,_,_,_,_,_,_]).
lex_eng_det(these,[plural,3,_,_,_,_,_,_]).
lex_eng_det(this,[singular,3,_,_,_,_,_,_]).
lex_eng_det(those,[plural,3,_,_,_,_,_,_]).
lex_eng_det(my,[_,3,_,_,_,_,_,_]).
lex_eng_det(your,[singular,3,_,_,_,_,_,_]).
lex_eng_det(his,[_,3,_,_,_,_,_,_]).
lex_eng_det(her,[_,3,_,_,_,_,_,_]).
lex_eng_det(its,[_,3,_,_,_,_,_,_]).
lex_eng_det(our,[_,3,_,_,_,_,_,_]).
lex_eng_det(yours,[plural,3,_,_,_,_,_,_]).
lex_eng_det(theirs,[plural,3,_,_,_,_,_,_]).



/* spanish determiners */

lex_sp_det(un,[singular,3,_,_,masc_y,_,_,_]).
lex_sp_det(una,[singular,3,_,_,masc_n,_,_,_]).
lex_sp_det(todo,[singular,3,_,_,masc_y,_,_,_]).
lex_sp_det(toda,[singular,3,_,_,masc_n,_,_,_]).
lex_sp_det(todos,[plural,3,_,_,masc_y,_,_,_]).
lex_sp_det(todas,[plural,3,_,_,masc_n,_,_,_]).
lex_sp_det(cada,[singular,3,_,_,_,_,_,_]).
lex_sp_det(algunos,[plural,3,_,_,masc_y,_,_,_]).
lex_sp_det(algunas,[plural,3,_,_,masc_n,_,_,_]).
lex_sp_det(ese,[singular,3,_,_,masc_y,_,_,_]).
lex_sp_det(esa,[singular,3,_,_,masc_n,_,_,_]).
lex_sp_det(aquel,[singular,3,_,_,masc_y,_,_,_]).
lex_sp_det(aquella,[singular,3,_,_,masc_n,_,_,_]).
lex_sp_det(el,[singular,3,_,_,masc_y,_,_,_]).
lex_sp_det(la,[singular,3,_,_,masc_n,_,_,_]).
lex_sp_det(los,[plural,3,_,_,masc_y,_,_,_]).
lex_sp_det(las,[plural,3,_,_,masc_n,_,_,_]).
lex_sp_det(estos,[plural,3,_,_,masc_y,_,_,_]).
lex_sp_det(estas,[plural,3,_,_,masc_n,_,_,_]).
lex_sp_det(este,[singular,3,_,_,masc_y,_,_,_]).
lex_sp_det(esta,[singular,3,_,_,masc_n,_,_,_]).
lex_sp_det(esos,[plural,3,_,_,masc_y,_,_,_]).
lex_sp_det(esas,[plural,3,_,_,masc_n,_,_,_]).
lex_sp_det(aquellos,[plural,3,_,_,masc_y,_,_,_]).
lex_sp_det(aquellas,[plural,3,_,_,masc_n,_,_,_]).
lex_sp_det(mi,[singular,3,_,_,_,_,_,_]).
lex_sp_det(mis,[plural,3,_,_,_,_,_,_]).
lex_sp_det(tu,[singular,3,_,_,_,_,_,_]).
```

```
lex_sp_det(tus,[plural,3,_,_,_,_,_,_,_]).
lex_sp_det(su,[singular,3,_,_,_,_,_,_,_]).
lex_sp_det(sus,[plural,3,_,_,masc_y,_,_,_,_]).
lex_sp_det(su,[singular,3,_,_,_,_,_,_,_]).
lex_sp_det(sus,[plural,3,_,_,masc_n,_,_,_,_]).
lex_sp_det(su,[singular,3,_,_,_,_,_,_,_]).
lex_sp_det(sus,[_,3,_,_,_,_,_,_,_]).
lex_sp_det(nuestro,[singular,3,_,_,masc_y,_,_,_,_]).
lex_sp_det(nuestra,[singular,3,_,_,masc_n,_,_,_,_]).
lex_sp_det(vuestro,[singular,3,_,_,masc_y,_,_,_,_]).
lex_sp_det(vuestra,[singular,3,_,_,masc_n,_,_,_,_]).
lex_sp_det(nuestros,[plural,3,_,_,masc_y,_,_,_,_]).
lex_sp_det(nuestras,[plural,3,_,_,masc_n,_,_,_,_]).
lex_sp_det(vuestros,[plural,3,_,_,masc_y,_,_,_,_]).
lex_sp_det(vuestras,[plural,3,_,_,masc_n,_,_,_,_]).



/*                    */
/* Intransitive Verbs */
/*                    */

lex_iv(disappear,disappear,present,[singular,1,_,_,_,_,_,_,_],desaparecer,desaparezco).
lex_iv(disappear,disappear,present,[singular,2,_,_,_,_,_,_,_],desaparecer,desapareces).
lex_iv(disappear,disappears,present,[singular,3,_,_,_,_,_,_,_],desaparecer,desaparece).
lex_iv(disappear,disappear,present,[plural,1,_,_,_,_,_,_,_],desaparecer,desaparecemos).
lex_iv(disappear,disappear,present,[plural,2,_,_,_,_,_,_,_],desaparecer,desaparecéis).
lex_iv(disappear,disappear,present,[plural,3,_,_,_,_,_,_,_],desaparecer,desaparecen).

lex_iv(disappear,disappeared,past,[singular,1,_,_,_,_,_,_,_],desaparecer,desaparecí).
lex_iv(disappear,disappeared,past,[singular,2,_,_,_,_,_,_,_],desaparecer,desapareciste).
lex_iv(disappear,disappeared,past,[singular,3,_,_,_,_,_,_,_],desaparecer,desapareció).
lex_iv(disappear,disappeared,past,[plural,1,_,_,_,_,_,_,_],desaparecer,desaparecimos).
lex_iv(disappear,disappeared,past,[plural,2,_,_,_,_,_,_,_],desaparecer,desaparecisteis).
lex_iv(disappear,disappeared,past,[plural,3,_,_,_,_,_,_,_],desaparecer,desaparecieron).

lex_iv(disappear,will_disappear,future,[singular,1,_,_,_,_,_,_,_],desaparecer,desaparecere).
lex_iv(disappear,will_disappear,future,[singular,2,_,_,_,_,_,_,_],desaparecer,desapareceras).
lex_iv(disappear,will_disappear,future,[singular,3,_,_,_,_,_,_,_],desaparecer,desaparecera).
lex_iv(disappear,will_disappear,future,[plural,1,_,_,_,_,_,_,_],desaparecer,desaparecereмos).
lex_iv(disappear,will_disappear,future,[plural,2,_,_,_,_,_,_,_],desaparecer,desaparecereis).
lex_iv(disappear,will_disappear,future,[plural,3,_,_,_,_,_,_,_],desaparecer,desapareceran).

lex_iv(disappear,disappearing,gerund,[_,_,_,_,_,_,_,_,_],desaparecer,desapareciendo).




lex_iv(fly,fly,present,[singular,1,_,_,_,_,loc_n,_],volar,vuelo).
lex_iv(fly,fly,present,[singular,2,_,_,_,_,loc_n,_],volar,vuelas).
lex_iv(fly,flies,present,[singular,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],volar,vuela).
lex_iv(fly,flies,present,[singular,3,_,hum_n,_,wing_y,loc_n,_],volar,vuela).
lex_iv(fly,fly,present,[plural,1,_,_,_,_,loc_n,_],volar,volamos).
```

```
lex_iv(fly, fly, present, [plural, 2, _, _, _, _, loc_n, _], volar, volais).
lex_iv(fly, fly, present, [plural, 3, anim_y, hum_y, _, wing_n, loc_n, meal_n], volar, vuelan).
lex_iv(fly, fly, present, [plural, 3, _, hum_n, _, wing_y, loc_n, _], volar, vuelan).

lex_iv(fly, flew, past, [singular, 1, _, _, _, _, loc_n, _], volar, vole).
lex_iv(fly, flew, past, [singular, 2, _, _, _, _, loc_n, _], volar, volaste).
lex_iv(fly, flew, past, [singular, 3, anim_y, hum_y, _, wing_n, loc_n, meal_n], volar, volo).
lex_iv(fly, flew, past, [singular, 3, _, hum_n, _, wing_y, loc_n, _], volar, volo).
lex_iv(fly, flew, past, [plural, 1, _, _, _, _, loc_n, _], volamos, volar).
lex_iv(fly, flew, past, [plural, 2, _, _, _, _, loc_n, _], volar, volasteis).
lex_iv(fly, flew, past, [plural, 3, anim_y, hum_y, _, wing_n, loc_n, meal_n], volar, volaron).
lex_iv(fly, flew, past, [plural, 3, _, hum_n, _, wing_y, loc_n, _], volar, volaron).

lex_iv(fly, will_fly, future, [singular, 1, _, _, _, _, loc_n, _], volar, volare).
lex_iv(fly, will_fly, future, [singular, 2, _, _, _, _, loc_n, _], volar, volaras).
lex_iv(fly, will_fly, future, [singular, 3, anim_y, hum_y, _, wing_n, loc_n, meal_n], volar, volara).
lex_iv(fly, will_fly, future, [singular, 3, _, hum_n, _, wing_y, loc_n, _], volar, volara).
lex_iv(fly, will_fly, future, [plural, 1, _, _, _, _, loc_n, _], volar, volaremos).
lex_iv(fly, will_fly, future, [plural, 2, _, _, _, _, loc_n, _], volar, volareis).
lex_iv(fly, will_fly, future, [plural, 3, anim_y, hum_y, _, wing_n, loc_n, meal_n], volar, volaran).
lex_iv(fly, will_fly, future, [plural, 3, _, hum_n, _, wing_y, loc_n, _], volar, volaran).

lex_iv(fly, flying, gerund, [_, _, anim_y, hum_y, _, wing_n, loc_n, meal_n], volar, volando).
lex_iv(fly, flying, gerund, [_, _, _, hum_n, _, wing_y, loc_n, _], volar, volando).




lex_iv(live, live, present, [singular, 1, anim_y, _, _, _, _, loc_n, _], vivir, vivo).
lex_iv(live, live, present, [singular, 2, anim_y, _, _, _, _, loc_n, _], vivir, vives).
lex_iv(live, lives, present, [singular, 3, anim_y, _, _, _, _, loc_n, _], vivir, vive).
lex_iv(live, live, present, [plural, 1, anim_y, _, _, _, _, loc_n, _], vivir, vivimos).
lex_iv(live, live, present, [plural, 2, anim_y, _, _, _, _, loc_n, _], vivir, vivis).
lex_iv(live, live, present, [plural, 3, anim_y, _, _, _, _, loc_n, _], vivir, viven).

lex_iv(live, lived, past, [singular, 1, anim_y, _, _, _, _, loc_n, _], vivir, vivi).
lex_iv(live, lived, past, [singular, 2, anim_y, _, _, _, _, loc_n, _], vivir, viviste).
lex_iv(live, lived, past, [singular, 3, anim_y, _, _, _, _, loc_n, _], vivir, vivio).
lex_iv(live, lived, past, [plural, 1, anim_y, _, _, _, _, loc_n, _], vivir, vivimos).
lex_iv(live, lived, past, [plural, 2, anim_y, _, _, _, _, loc_n, _], vivir, vivisteis).
lex_iv(live, lived, past, [plural, 3, anim_y, _, _, _, _, loc_n, _], vivir, vivieron).

lex_iv(live, will_live, future, [singular, 1, anim_y, _, _, _, _, loc_n, _], vivir, vivire).
lex_iv(live, will_live, future, [singular, 2, anim_y, _, _, _, _, loc_n, _], vivir, viviras).
lex_iv(live, will_live, future, [singular, 3, anim_y, _, _, _, _, loc_n, _], vivir, vivira).
lex_iv(live, will_live, future, [plural, 1, anim_y, _, _, _, _, loc_n, _], vivir, viviremos).
lex_iv(live, will_live, future, [plural, 2, anim_y, _, _, _, _, loc_n, _], vivir, vivireis).
lex_iv(live, will_live, future, [plural, 3, anim_y, _, _, _, _, loc_n, _], vivir, viviran).

lex_iv(live, living, gerund, [_, _, anim_y, _, _, _, _, loc_n, _], vivir, viviendo).
```

```
lex_iv(play,play,present,[singular,1,_,_,_,_,loc_n,meal_n],jugar,juego).
lex_iv(play,play,present,[singular,2,_,_,_,_,loc_n,meal_n],jugar,juegas).
lex_iv(play,plays,present,[singular,3,anim_y,_,_,_,loc_n,meal_n],jugar,juega).
lex_iv(play,play,present,[plural,1,_,_,_,_,loc_n,meal_n],jugar,jugamos).
lex_iv(play,play,present,[plural,2,_,_,_,_,loc_n,meal_n],jugar,jugais).
lex_iv(play,play,present,[plural,3,anim_y,_,_,_,loc_n,meal_n],jugar,juegan).

lex_iv(play,played,past,[singular,1,_,_,_,_,loc_n,meal_n],jugar,jugue).
lex_iv(play,played,past,[singular,2,_,_,_,_,loc_n,meal_n],jugar,jugaste).
lex_iv(play,played,past,[singular,3,anim_y,_,_,_,loc_n,meal_n],jugar,jugo).
lex_iv(play,played,past,[plural,1,_,_,_,_,loc_n,meal_n],jugar,jugamos).
lex_iv(play,played,past,[plural,2,_,_,_,_,loc_n,meal_n],jugar,jugasteis).
lex_iv(play,played,past,[plural,3,anim_y,_,_,_,loc_n,meal_n],jugar,jugaron).

lex_iv(play,will_play,future,[singular,1,_,_,_,_,loc_n,meal_n],jugar,jugare).
lex_iv(play,will_play,future,[singular,2,_,_,_,_,loc_n,meal_n],jugar,jugaras).
lex_iv(play,will_play,future,[singular,3,anim_y,_,_,_,loc_n,meal_n],jugar,jugara).
lex_iv(play,will_play,future,[plural,1,_,_,_,_,loc_n,meal_n],jugar,jugaremos).
lex_iv(play,will_play,future,[plural,2,_,_,_,_,loc_n,meal_n],jugar,jugareis).
lex_iv(play,will_play,future,[plural,3,anim_y,_,_,_,loc_n,meal_n],jugar,jugaran).

lex_iv(play,playing,gerund,[_,_,anim_y,_,_,_,loc_n,meal_n],jugar,jugando).



lex_iv(run,run,present,[singular,1,anim_y,_,_,_,_,_],correr,corro).
lex_iv(run,run,present,[singular,2,anim_y,_,_,_,_,_],correr,corres).
lex_iv(run,runs,present,[singular,3,anim_y,_,_,wing_n,loc_n,meal_n],correr,corre).
lex_iv(run,run,present,[plural,1,anim_y,_,_,_,_,_],correr,corremos).
lex_iv(run,run,present,[plural,2,anim_y,_,_,_,_,_],correr,correis).
lex_iv(run,run,present,[plural,3,anim_y,_,_,wing_n,loc_n,meal_n],correr,corren).

lex_iv(run,ran,past,[singular,1,anim_y,_,_,_,_,_],correr,corri).
lex_iv(run,ran,past,[singular,2,anim_y,_,_,_,_,_],correr,corriste).
lex_iv(run,ran,past,[singular,3,anim_y,_,_,wing_n,loc_n,meal_n],correr,corrio).
lex_iv(run,ran,past,[plural,1,anim_y,_,_,_,_,_],correr,corrimos).
lex_iv(run,ran,past,[plural,2,anim_y,_,_,_,_,_],correr,corristeis).
lex_iv(run,ran,past,[plural,3,anim_y,_,_,wing_n,loc_n,meal_n],correr,corrieron).

lex_iv(run,will_run,future,[singular,1,anim_y,_,_,_,_,_],correr,correre).
lex_iv(run,will_run,future,[singular,2,anim_y,_,_,_,_,_],correr,correras).
lex_iv(run,will_run,future,[singular,3,anim_y,_,_,wing_n,loc_n,meal_n],correr,correra).
lex_iv(run,will_run,future,[plural,1,anim_y,_,_,_,_,_],correr,correremos).
lex_iv(run,will_run,future,[plural,2,anim_y,_,_,_,_,_],correr,correreis).
lex_iv(run,will_run,future,[plural,3,anim_y,_,_,wing_n,loc_n,meal_n],correr,correran).

lex_iv(run,running,gerund,[_,_,anim_y,_,_,wing_n,loc_n,meal_n],correr,corriendo).
```

```
lex_iv(sing,sing,present,[singular,1,_,hum_y,_,_,_,_],cantar,canto).
lex_iv(sing,sing,present,[singular,2,_,hum_y,_,_,_,_],cantar,cantas).
lex_iv(sing,sings,present,[singular,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],cantar,canta).
lex_iv(sing,sings,present,[singular,3,anim_y,hum_n,_,wing_y,loc_n,_],cantar,canta).
lex_iv(sing,sing,present,[plural,1,_,hum_y,_,_,_,_],cantar,cantamos).
lex_iv(sing,sing,present,[plural,2,_,hum_y,_,_,_,_],cantar,cantais).
lex_iv(sing,sing,present,[plural,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],cantar,cantan).
lex_iv(sing,sing,present,[plural,3,anim_y,hum_n,_,wing_y,loc_n,_],cantar,cantan).

lex_iv(sing,sang,past,[singular,1,_,hum_y,_,_,_,_],cantar,cante).
lex_iv(sing,sang,past,[singular,2,_,hum_y,_,_,_,_],cantar,cantaste).
lex_iv(sing,sang,past,[singular,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],cantar,canto).
lex_iv(sing,sang,past,[singular,3,anim_y,hum_n,_,wing_y,loc_n,_],cantar,canto).
lex_iv(sing,sang,past,[plural,1,_,hum_y,_,_,_,_],cantar,cantamos).
lex_iv(sing,sang,past,[plural,2,_,hum_y,_,_,_,_],cantar,cantasteis).
lex_iv(sing,sang,past,[plural,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],cantar,cantaron).
lex_iv(sing,sang,past,[plural,3,anim_y,hum_n,_,wing_y,loc_n,_],cantar,cantaron).

lex_iv(sing,will_sing,future,[singular,1,_,hum_y,_,_,_,_],cantar,cantare).
lex_iv(sing,will_sing,future,[singular,2,_,hum_y,_,_,_,_],cantar,cantaras).
lex_iv(sing,will_sing,future,[singular,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],cantar,cantara).
lex_iv(sing,will_sing,future,[singular,3,anim_y,hum_n,_,wing_y,loc_n,_],cantar,cantara).
lex_iv(sing,will_sing,future,[plural,1,_,hum_y,_,_,_,_],cantar,cantaremos).
lex_iv(sing,will_sing,future,[plural,2,_,hum_y,_,_,_,_],cantar,cantareis).
lex_iv(sing,will_sing,future,[plural,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],cantar,cantaran).
lex_iv(sing,will_sing,future,[plural,3,anim_y,hum_n,_,wing_y,loc_n,_],cantar,cantaran).

lex_iv(sing,singing,gerund,[_,_,anim_y,hum_y,_,wing_n,loc_n,meal_n],cantar,cantando).
lex_iv(sing,singing,gerund,[_,_,anim_y,hum_n,_,wing_y,loc_n,_],cantar,cantando).




lex_iv(sleep,sleep,present,[singular,1,anim_y,_,_,_,_,loc_n,_],dormir,duermo).
lex_iv(sleep,sleep,present,[singular,2,anim_y,_,_,_,_,loc_n,_],dormir,duermes).
lex_iv(sleep,sleeps,present,[singular,3,anim_y,_,_,_,_,loc_n,_],dormir,duerme).
lex_iv(sleep,sleep,present,[plural,1,anim_y,_,_,_,_,loc_n,_],dormir,dormimos).
lex_iv(sleep,sleep,present,[plural,2,anim_y,_,_,_,_,loc_n,_],dormir,dormis).
lex_iv(sleep,sleep,present,[plural,3,anim_y,_,_,_,_,loc_n,_],dormir,duermen).

lex_iv(sleep,slept,past,[singular,1,anim_y,_,_,_,_,loc_n,_],dormir,dormi).
lex_iv(sleep,slept,past,[singular,2,anim_y,_,_,_,_,loc_n,_],dormir,dormiste).
lex_iv(sleep,slept,past,[singular,3,anim_y,_,_,_,_,loc_n,_],dormir,durmio).
lex_iv(sleep,slept,past,[plural,1,anim_y,_,_,_,_,loc_n,_],dormir,dormimos).
lex_iv(sleep,slept,past,[plural,2,anim_y,_,_,_,_,loc_n,_],dormir,dormisteis).
lex_iv(sleep,slept,past,[plural,3,anim_y,_,_,_,_,loc_n,_],dormir,durmieron).

lex_iv(sleep,will_sleep,future,[singular,1,anim_y,_,_,_,_,loc_n,_],dormir,dormire).
lex_iv(sleep,will_sleep,future,[singular,2,anim_y,_,_,_,_,loc_n,_],dormir,dormiras).
lex_iv(sleep,will_sleep,future,[singular,3,anim_y,_,_,_,_,loc_n,_],dormir,dormira).
lex_iv(sleep,will_sleep,future,[plural,1,anim_y,_,_,_,_,loc_n,_],dormir,dormiremos).
lex_iv(sleep,will_sleep,future,[plural,2,anim_y,_,_,_,_,loc_n,_],dormir,dormireis).
lex_iv(sleep,will_sleep,future,[plural,3,anim_y,_,_,_,_,loc_n,_],dormir,dormiran).
```

```
lex_iv(sleep,sleeping,gerund,[_,_,anim_y,_,_,_,loc_n,_],dormir,durmiendo).


lex_iv(smoke,smoke,present,[singular,1,_,hum_y,_,_,_,_],fumar,fumo).
lex_iv(smoke,smoke,present,[singular,2,_,hum_y,_,_,_,_],fumar,fumas).
lex_iv(smoke,smokes,present,[singular,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],fumar,fuma).
lex_iv(smoke,smoke,present,[plural,1,_,hum_y,_,_,_,_],fumar,fumamos).
lex_iv(smoke,smoke,present,[plural,2,_,hum_y,_,_,_,_],fumar,fumais).
lex_iv(smoke,smoke,present,[plural,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],fumar,fuman).

lex_iv(smoke,smoked,past,[singular,1,_,hum_y,_,_,_,_],fumar,fume).
lex_iv(smoke,smoked,past,[singular,2,_,hum_y,_,_,_,_],fumar,fumaste).
lex_iv(smoke,smoked,past,[singular,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],fumar,fumo).
lex_iv(smoke,smoked,past,[plural,1,_,hum_y,_,_,_,_],fumar,fumamos).
lex_iv(smoke,smoked,past,[plural,2,_,hum_y,_,_,_,_],fumar,fumasteis).
lex_iv(smoke,smoked,past,[plural,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],fumar,fumaron).

lex_iv(smoke,will_smoke,future,[singular,1,_,hum_y,_,_,_,_],fumar,fumare).
lex_iv(smoke,will_smoke,future,[singular,2,_,hum_y,_,_,_,_],fumar,fumaras).
lex_iv(smoke,will_smoke,future,[singular,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],fumar,fumara).
lex_iv(smoke,will_smoke,future,[plural,1,_,hum_y,_,_,_,_],fumar,fumaremos).
lex_iv(smoke,will_smoke,future,[plural,2,_,hum_y,_,_,_,_],fumar,fumareis).
lex_iv(smoke,will_smoke,future,[plural,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],fumar,fumaran).

lex_iv(smoke,smoking,gerund,[_,_,anim_y,hum_y,_,wing_n,loc_n,meal_n],fumar,fumando).



lex_iv(speak,speak,present,[singular,1,_,hum_y,_,_,_,_],hablar,hablo).
lex_iv(speak,speak,present,[singular,2,_,hum_y,_,_,_,_],hablar,hablas).
lex_iv(speak,speaks,present,[singular,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],hablar,habla).
lex_iv(speak,speak,present,[plural,1,_,hum_y,_,_,_,_],hablar,hablamos).
lex_iv(speak,speak,present,[plural,2,_,hum_y,_,_,_,_],hablar,hablais).
lex_iv(speak,speak,present,[plural,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],hablar,hablan).

lex_iv(speak,spoke,past,[singular,1,_,hum_y,_,_,_,_],hablar,hable).
lex_iv(speak,spoke,past,[singular,2,_,hum_y,_,_,_,_],hablar,hablaste).
lex_iv(speak,spoke,past,[singular,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],hablar,hablo).
lex_iv(speak,spoke,past,[plural,1,_,hum_y,_,_,_,_],hablar,hablamos).
lex_iv(speak,spoke,past,[plural,2,_,hum_y,_,_,_,_],hablar,hablasteis).
lex_iv(speak,spoke,past,[plural,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],hablar,hablaron).

lex_iv(speak,will_speak,future,[singular,1,_,hum_y,_,_,_,_],hablar,hablare).
lex_iv(speak,will_speak,future,[singular,2,_,hum_y,_,_,_,_],hablar,hablaras).
lex_iv(speak,will_speak,future,[singular,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],hablar,hablara).
lex_iv(speak,will_speak,future,[plural,1,_,hum_y,_,_,_,_],hablar,hablaremos).
lex_iv(speak,will_speak,future,[plural,2,_,hum_y,_,_,_,_],hablar,hablareis).
lex_iv(speak,will_speak,future,[plural,3,anim_y,hum_y,_,wing_n,loc_n,meal_n],hablar,hablaran).

lex_iv(speak,speaking,gerund,[_,_,anim_y,hum_y,_,wing_n,loc_n,meal_n],hablar,hablando).
```

```
lex_iv(study,study,present,[singular,1,_,hum_y,_,_,_,_],estudiar,estudio).
lex_iv(study,study,present,[singular,2,_,hum_y,_,_,_,_],estudiar,estudias).
lex_iv(study,studies,present,[singular,3,_,hum_y,_,_,_,_],estudiar,estudia).
lex_iv(study,study,present,[plural,1,_,hum_y,_,_,_,_],estudiar,estudiamos).
lex_iv(study,study,present,[plural,2,_,hum_y,_,_,_,_],estudiar,estudiais).
lex_iv(study,study,present,[plural,3,_,hum_y,_,_,_,_],estudiar,estudian).

lex_iv(study,studied,past,[singular,1,_,hum_y,_,_,_,_],estudiar,estudie).
lex_iv(study,studied,past,[singular,2,_,hum_y,_,_,_,_],estudiar,estudiaste).
lex_iv(study,studied,past,[singular,3,_,hum_y,_,_,_,_],estudiar,estudio).
lex_iv(study,studied,past,[plural,1,_,hum_y,_,_,_,_],estudiar,estudiasos).
lex_iv(study,studied,past,[plural,2,_,hum_y,_,_,_,_],estudiar,estudiasteis).
lex_iv(study,studied,past,[plural,3,_,hum_y,_,_,_,_],estudiar,estudiaron).

lex_iv(study,will_study,future,[singular,1,_,hum_y,_,_,_,_],estudiar,estudiare).
lex_iv(study,will_study,future,[singular,2,_,hum_y,_,_,_,_],estudiar,estudiaras).
lex_iv(study,will_study,future,[singular,3,_,hum_y,_,_,_,_],estudiar,estudiara).
lex_iv(study,will_study,future,[plural,1,_,hum_y,_,_,_,_],estudiar,estudiaremos).
lex_iv(study,will_study,future,[plural,2,_,hum_y,_,_,_,_],estudiar,estudiareis).
lex_iv(study,will_study,future,[plural,3,_,hum_y,_,_,_,_],estudiar,estudiaran).

lex_iv(study,smoking,gerund,[_,_,_,hum_y,_,_,_,_],estudiar,estudiando).




/*              */
/* Proper nouns */
/*              */

lex_pn(alex,[anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],alejandro).
lex_pn(charles,[anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],carlos).
lex_pn(edward,[anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],eduardo).
lex_pn(edward,[anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],lalo).
lex_pn(eugene,[anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],eugenio).
lex_pn(fred,[anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],federico).
lex_pn(gerard,[anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],gerardo).
lex_pn(jim,[anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],jim).
lex_pn(john,[anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],juan).
lex_pn(joseph,[anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],jose).
lex_pn(billy,[anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],memo).
lex_pn(peter,[anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],pedro).
lex_pn(pete,[anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],pedro).
lex_pn(sam,[anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],samuel).
lex_pn(william,[anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],guillermo).

lex_pn(alice,[anim_y,hum_y,masc_n,wing_n,loc_n,meal_n],alicia).
lex_pn(carmen,[anim_y,hum_y,masc_n,wing_n,loc_n,meal_n],carmen).
lex_pn(chris,[anim_y,hum_y,masc_n,wing_n,loc_n,meal_n],cristina).
lex_pn(guadalupe,[anim_y,hum_y,masc_n,wing_n,loc_n,meal_n],guadalupe).
lex_pn(laura,[anim_y,hum_y,masc_n,wing_n,loc_n,meal_n],laura).
lex_pn(lupe,[anim_y,hum_y,masc_n,wing_n,loc_n,meal_n],lupe).
```

```
lex_pn(lucy, [anim_y,hum_y,masc_n,wing_n,loc_n,meal_n], lucia).
lex_pn(mary, [anim_y,hum_y,masc_n,wing_n,loc_n,meal_n], maria).
lex_pn(norma, [anim_y,hum_y,masc_n,wing_n,loc_n,meal_n], norma).
lex_pn(paty, [anim_y,hum_y,masc_n,wing_n,loc_n,meal_n], patricia).
lex_pn(priss, [anim_y,hum_y,masc_n,wing_n,loc_n,meal_n], priscila).
lex_pn(rose, [anim_y,hum_y,masc_n,wing_n,loc_n,meal_n], rosa).
lex_pn(susan, [anim_y,hum_y,masc_n,wing_n,loc_n,meal_n], susana).

lex_pn(fido, [anim_y,hum_n,masc_y,wing_n,loc_n,meal_n], fido).
lex_pn(lassie, [anim_y,hum_n,masc_n,wing_n,loc_n,meal_n], lassie).
lex_pn(neron, [anim_y,hum_n,masc_y,wing_n,loc_n,meal_n], neron).
lex_pn(piolin, [anim_y,hum_n,masc_y,wing_y,loc_n,meal_n], piolin).
lex_pn(silvester, [anim_y,hum_n,masc_y,wing_n,loc_n,meal_n], silvestre).

lex_pn(california, [anim_n,hum_n,_,_,loc_y,meal_n], california).
lex_pn(cancun, [anim_n,hum_n,_,_,loc_y,meal_n], cancun).
lex_pn(cuernavaca, [anim_n,hum_n,_,_,loc_y,meal_n], cuernavaca).
lex_pn(london, [anim_n,hum_n,_,_,loc_y,meal_n], londres).
lex_pn(madrid, [anim_n,hum_n,_,_,loc_y,meal_n], madrid).
lex_pn(mexico, [anim_n,hum_n,_,_,loc_y,meal_n], mexico).
lex_pn(new_york, [anim_n,hum_n,_,_,loc_y,meal_n], nueva_york).




/*              */
/* Prepositions */
/*              */

lex_prep(in, [_,_,_,_,_,_,loc_y,_], en).
lex_prep(with, [singular,1,_,_,_,_,loc_n,_], conmigo).
lex_prep(with, [singular,2,_,_,_,_,loc_n,_], contigo).
lex_prep(with, [_,_,_,_,_,_,loc_n,_], con).



/*          */
/* Pronouns */
/*          */

lex_pron(i, [singular,1,anim_y,hum_y,_,wing_n,loc_n,meal_n], yo).
lex_pron(me, [singular,1,anim_y,hum_y,_,wing_n,loc_n,meal_n], conmigo).
lex_pron(you, [singular,2,anim_y,hum_y,_,wing_n,loc_n,meal_n], tu).
lex_pron(you, [singular,2,anim_y,hum_y,_,wing_n,loc_n,meal_n], contigo).
lex_pron(he, [singular,3,anim_y,hum_y,masc_y,wing_n,loc_n,meal_n], el).
lex_pron(him, [singular,3,anim_y,hum_y,masc_y,wing_n,loc_n,meal_n], el).
lex_pron(she, [singular,3,anim_y,hum_y,masc_n,wing_n,loc_n,meal_n], ella).
lex_pron(her, [singular,3,anim_y,hum_y,masc_n,wing_n,loc_n,meal_n], ella).
lex_pron(it, [singular,3,_,_,_,_,_,_], ello).
lex_pron(we, [plural,1,anim_y,hum_y,masc_y,wing_n,loc_n,meal_n], nosotros).
lex_pron(we, [plural,1,anim_y,hum_y,masc_n,wing_n,loc_n,meal_n], nosotras).
```

```
lex_pron(us,[plural,1,anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],nosotros).
lex_pron(us,[plural,1,anim_y,hum_y,masc_n,wing_n,loc_n,meal_n],nosotras).
lex_pron(you,[plural,2,anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],vosotros).
lex_pron(you,[plural,2,anim_y,hum_y,masc_n,wing_n,loc_n,meal_n],vosotras).
lex_pron(they,[plural,3,anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],ellos).
lex_pron(they,[plural,3,anim_y,hum_y,masc_n,wing_n,loc_n,meal_n],ellas).
lex_pron(them,[plural,3,anim_y,hum_y,masc_y,wing_n,loc_n,meal_n],ellos).
lex_pron(them,[plural,3,anim_y,hum_y,masc_n,wing_n,loc_n,meal_n],ellas).



/*             */
/* Quantifiers */
/*             */

lex_quant(many,[plural,3,_,_,masc_y,_,_,_,_],muchos).
lex_quant(many,[plural,3,_,_,masc_n,_,_,_,_],muchas).
lex_quant(few,[plural,3,_,_,masc_y,_,_,_,_],algunos).
lex_quant(few,[plural,3,_,_,masc_n,_,_,_,_],algunas).
lex_quant(few,[plural,3,_,_,masc_y,_,_,_,_],pocos).
lex_quant(few,[plural,3,_,_,masc_n,_,_,_,_],pocas).



/*             */
/* relational  */
/*             */

lex_rel(that,que).



/*                  */
/* Transitive Verbs */
/*                  */

lex_tv(admire,admire,present,[singular,1,_,hum_y,_,_,_,_,_],_,admirar,admiro).
lex_tv(admire,admire,present,[singular,2,_,hum_y,_,_,_,_,_],_,admirar,admiras).
lex_tv(admire,admires,present,[singular,3,_,hum_y,_,_,_,_,_],_,admirar,admira).
lex_tv(admire,admire,present,[plural,1,_,hum_y,_,_,_,_,_],_,admirar,admiramos).
lex_tv(admire,admire,present,[plural,2,_,hum_y,_,_,_,_,_],_,admirar,admirais).
lex_tv(admire,admire,present,[plural,3,_,hum_y,_,_,_,_,_],_,admirar,admiran).

lex_tv(admire,admired,past,[singular,1,_,hum_y,_,_,_,_,_],_,admirar,admire).
lex_tv(admire,admired,past,[singular,2,_,hum_y,_,_,_,_,_],_,admirar,admiraste).
lex_tv(admire,admired,past,[singular,3,_,hum_y,_,_,_,_,_],_,admirar,admiro).
lex_tv(admire,admired,past,[plural,1,_,hum_y,_,_,_,_,_],_,admirar,admiramos).
lex_tv(admire,admired,past,[plural,2,_,hum_y,_,_,_,_,_],_,admirar,admirasteis).
lex_tv(admire,admired,past,[plural,3,_,hum_y,_,_,_,_,_],_,admirar,admiraron).
```

```
lex_tv(admire,will_admire,future,[singular,1,_,hum_y,_,_,_,_,_],_,admirar,admirare).
lex_tv(admire,will_admire,future,[singular,2,_,hum_y,_,_,_,_,_],_,admirar,admiraras).
lex_tv(admire,will_admire,future,[singular,3,_,hum_y,_,_,_,_,_],_,admirar,admirara).
lex_tv(admire,will_admire,future,[plural,1,_,hum_y,_,_,_,_,_],_,admirar,admiraremos).
lex_tv(admire,will_admire,future,[plural,2,_,hum_y,_,_,_,_,_],_,admirar,admirareis).
lex_tv(admire,will_admire,future,[plural,3,_,hum_y,_,_,_,_,_],_,admirar,admiraran).

lex_tv(admire,admiring,gerund,[_,_,_,hum_y,_,_,_,_,_],_,admirar,admirando).



lex_tv(adopt,adopt,present,[singular,1,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adopto).
lex_tv(adopt,adopt,present,[singular,2,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adoptas).
lex_tv(adopt,adopts,present,[singular,3,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adopta).
lex_tv(adopt,adopt,present,[plural,1,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adoptamos).
lex_tv(adopt,adopt,present,[plural,2,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adoptais).
lex_tv(adopt,adopt,present,[plural,3,anim_y,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adoptan).

lex_tv(adopt,adopted,past,[singular,1,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adopte).
lex_tv(adopt,adopted,past,[singular,2,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adoptaste).
lex_tv(adopt,adopted,past,[singular,3,anim_y,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adopto).
lex_tv(adopt,adopted,past,[plural,1,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adoptamos).
lex_tv(adopt,adopted,past,[plural,2,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adoptasteis).
lex_tv(adopt,adopted,past,[plural,3,anim_y,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adoptaron).

lex_tv(adopt,will_adopt,future,[singular,1,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adoptare).
lex_tv(adopt,will_adopt,future,[singular,2,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adoptaras).
lex_tv(adopt,will_adopt,future,[singular,3,anim_y,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adoptara).
lex_tv(adopt,will_adopt,future,[plural,1,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adoptaremos).
lex_tv(adopt,will_adopt,future,[plural,2,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adoptareis).
lex_tv(adopt,will_adopt,future,[plural,3,anim_y,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adoptaran).

lex_tv(adopt,adopting,gerund,[_,_,anim_y,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_,_],
       adoptar,adoptando).
```

```
lex_tv(break,break,present,[singular,1,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,rompo).
lex_tv(break,break,present,[singular,2,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,rompes).
lex_tv(break,breaks,present,[singular,3,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,rompe).
lex_tv(break,break,present,[plural,1,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,rompemos).
lex_tv(break,break,present,[plural,2,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,rompeis).
lex_tv(break,break,present,[plural,3,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,rompen).

lex_tv(break,broke,past,[singular,1,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,rompi).
lex_tv(break,broke,past,[singular,2,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,rompiste).
lex_tv(break,broke,past,[singular,3,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,rompio).
lex_tv(break,broke,past,[plural,1,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,rompimos).
lex_tv(break,broke,past,[plural,2,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,rompisteis).
lex_tv(break,broke,past,[plural,3,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,rompieron).

lex_tv(break,will_break,future,[singular,1,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,romperé).
lex_tv(break,will_break,future,[singular,2,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,romperás).
lex_tv(break,will_break,future,[singular,3,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,romperá).
lex_tv(break,will_break,future,[plural,1,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,romperemos).
lex_tv(break,will_break,future,[plural,2,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,romperéis).
lex_tv(break,will_break,future,[plural,3,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,romperán).

lex_tv(break,breaking,gerund,[_,_,anim_y,_,_,_,_,_],[_,_,anim_n,_,_,_,_],loc_n,meal_nl],
       romper,rompiendo).




lex_tv(buy,buy,present,[singular,1,_,hum_y,_,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,comoro).
lex_tv(buy,buy,present,[singular,2,_,hum_y,_,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,compras).
lex_tv(buy,buys,present,[singular,3,_,hum_y,_,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,compra).
lex_tv(buy,buy,present,[plural,1,_,hum_y,_,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,compramos).
lex_tv(buy,buy,present,[plural,2,_,hum_y,_,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,compráis).
lex_tv(buy,buy,present,[plural,3,_,hum_y,_,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,compran).
```

```
lex_tv(buy,bought,past,[singular,1,_,hum_y,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,compré).
lex_tv(buy,bought,past,[singular,2,_,hum_y,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,compraste).
lex_tv(buy,bought,past,[singular,3,_,hum_y,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,compró).
lex_tv(buy,bought,past,[plural,1,_,hum_y,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,compramos).
lex_tv(buy,bought,past,[plural,2,_,hum_y,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,comprasteis).
lex_tv(buy,bought,past,[plural,3,_,hum_y,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,compraron).

lex_tv(buy,will_buy,future,[singular,1,_,hum_y,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,comprare).
lex_tv(buy,will_buy,future,[singular,2,_,hum_y,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,comprarás).
lex_tv(buy,will_buy,future,[singular,3,_,hum_y,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,comprará).
lex_tv(buy,will_buy,future,[plural,1,_,hum_y,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,compraremos).
lex_tv(buy,will_buy,future,[plural,2,_,hum_y,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,compraréis).
lex_tv(buy,will_buy,future,[plural,3,_,hum_y,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,comprarán).

lex_tv(buy,buying,gerund,[_,_,_,hum_y,_,_,_,_],[_,_,_,hum_n,_,_,_,_],comprar,comprando).
```

```
lex_tv(catch,catch,present,[singular,1,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atrapo).
lex_tv(catch,catch,present,[singular,2,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atrapas).
lex_tv(catch,catches,present,[singular,3,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atrapa).
lex_tv(catch,catch,present,[plural,1,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atrapamos).
lex_tv(catch,catch,present,[plural,2,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atrapais).
lex_tv(catch,catch,present,[plural,3,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atrapan).

lex_tv(catch,caught,past,[singular,1,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atrapé).
lex_tv(catch,caught,past,[singular,2,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atrapaste).
lex_tv(catch,caught,past,[singular,3,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atrapó).
lex_tv(catch,caught,past,[plural,1,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atrapamos).
lex_tv(catch,caught,past,[plural,2,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atrapasteis).
lex_tv(catch,caught,past,[plural,3,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atraparon).

lex_tv(catch,will_catch,future,[singular,1,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atraparé).
lex_tv(catch,will_catch,future,[singular,2,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atraparás).
lex_tv(catch,will_catch,future,[singular,3,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atrapará).
lex_tv(catch,will_catch,future,[plural,1,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atraparemos).
```

```
lex_tv(catch,will_catch,future,[plural,2,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atrapareis).
lex_tv(catch,will_catch,future,[plural,3,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atraparan).

lex_tv(catch,catching,gerund,[_,_,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,loc_n,_],
       atrapar,atrapando).



lex_tv(chase,chase,present,[singular,1,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,cazo).
lex_tv(chase,chase,present,[singular,2,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,cazas).
lex_tv(chase,chases,present,[singular,3,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,caza).
lex_tv(chase,chase,present,[plural,1,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,cazamos).
lex_tv(chase,chase,present,[plural,2,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,cazais).
lex_tv(chase,chase,present,[plural,3,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,cazan).

lex_tv(chase,chased,past,[singular,1,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,caze).
lex_tv(chase,chased,past,[singular,2,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,cazaste).
lex_tv(chase,chased,past,[singular,3,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,cazo).
lex_tv(chase,chased,past,[plural,1,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,cazamos).
lex_tv(chase,chased,past,[plural,2,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,cazasteis).
lex_tv(chase,chased,past,[plural,3,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,cazaron).

lex_tv(chase,will_chase,future,[singular,1,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,cazare).
lex_tv(chase,will_chase,future,[singular,2,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,cazaras).
lex_tv(chase,will_chase,future,[singular,3,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,cazara).
lex_tv(chase,will_chase,future,[plural,1,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,cazaremos).
lex_tv(chase,will_chase,future,[plural,2,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,cazareis).
lex_tv(chase,will_chase,future,[plural,3,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,cazaran).

lex_tv(chase,chasing,gerund,[_,_,anim_y,_,_,_,_,_],[_,_,anim_y,hum_n,_,_,_,_],
       cazar,cazando).
```

```
lex_tv(clean,clean,present,[singular,1,_,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpio).
lex_tv(clean,clean,present,[singular,2,_,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpias).
lex_tv(clean,cleans,present,[singular,3,_,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpia).
lex_tv(clean,clean,present,[plural,1,_,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpiamos).
lex_tv(clean,clean,present,[plural,2,_,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpiais).
lex_tv(clean,clean,present,[plural,3,anim_y,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpian).

lex_tv(clean,cleaned,past,[singular,1,_,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpie).
lex_tv(clean,cleaned,past,[singular,2,_,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpiaste).
lex_tv(clean,cleaned,past,[singular,3,anim_y,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpio).
lex_tv(clean,cleaned,past,[plural,1,_,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpiamos).
lex_tv(clean,cleaned,past,[plural,2,_,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpiasteis).
lex_tv(clean,cleaned,past,[plural,3,anim_y,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpiaron).

lex_tv(clean,will_clean,future,[singular,1,_,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpiare).
lex_tv(clean,will_clean,future,[singular,2,_,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpiaras).
lex_tv(clean,will_clean,future,[singular,3,anim_y,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpiara).
lex_tv(clean,will_clean,future,[plural,1,_,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpiaremos).
lex_tv(clean,will_clean,future,[plural,2,_,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpiareis).
lex_tv(clean,will_clean,future,[plural,3,anim_y,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpiaran).

lex_tv(clean,cleanning,gerund,[_,_,anim_y,hum_y,_,_,_,_],[_,_,anim_n,_,_,_,_,_],
       limpiar,limpiando).
```

```
lex_tv(cut,cut,present,[singular,1,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,corto).
lex_tv(cut,cut,present,[singular,2,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,cortas).
lex_tv(cut,cuts,present,[singular,3,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,corta).
lex_tv(cut,cut,present,[plural,1,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,cortamos).
lex_tv(cut,cut,present,[plural,2,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,cortais).
lex_tv(cut,cut,present,[plural,3,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,cortan).
```

```
lex_tv(cut,cut,past,[singular,1,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,corte).
lex_tv(cut,cut,past,[singular,2,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,cortaste).
lex_tv(cut,cut,past,[singular,3,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,cortó).
lex_tv(cut,cut,past,[plural,1,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,cortamos).
lex_tv(cut,cut,past,[plural,2,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,cortasteis).
lex_tv(cut,cut,past,[plural,3,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,cortaron).

lex_tv(cut,will_cut,future,[singular,1,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,cortare).
lex_tv(cut,will_cut,future,[singular,2,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,cortaras).
lex_tv(cut,will_cut,future,[singular,3,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,cortara).
lex_tv(cut,will_cut,future,[plural,1,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,cortaremos).
lex_tv(cut,will_cut,future,[plural,2,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,cortareis).
lex_tv(cut,will_cut,future,[plural,3,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,cortaran).

lex_tv(cut,cuting,gerund,[_,_,_,hum_y,_,_,_,_],[_,_,_,_,_,_,_,meal_y],cortar,cortando).
```

```
lex_tv(defend,defend,present,[singular,1,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
    defender,defiendo).
lex_tv(defend,defend,present,[singular,2,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
    defender,defiendes).
lex_tv(defend,defends,present,[singular,3,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
    defender,defiende).
lex_tv(defend,defend,present,[plural,1,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
    defender,defendemos).
lex_tv(defend,defend,present,[plural,2,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
    defender,defendeis).
lex_tv(defend,defend,present,[plural,3,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
    defender,defienden).

lex_tv(defend,defended,past,[singular,1,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
    defender,defendi).
lex_tv(defend,defended,past,[singular,2,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
    defender,defendiste).
lex_tv(defend,defended,past,[singular,3,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
    defender,defendio).
lex_tv(defend,defended,past,[plural,1,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
    defender,defendimos).
lex_tv(defend,defended,past,[plural,2,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
    defender,defendisteis).
lex_tv(defend,defended,past,[plural,3,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
    defender,defendieron).

lex_tv(defend,will_defend,future,[singular,1,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
    defender,defenderé).
lex_tv(defend,will_defend,future,[singular,2,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
    defender,defenderás).
lex_tv(defend,will_defend,future,[singular,3,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
    defender,defenderá).
lex_tv(defend,will_defend,future,[plural,1,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
    defender,defenderemos).
```

```
lex_tv(defend,will_defend,future,[plural,2,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
        defender,defendereis).
lex_tv(defend,will_defend,future,[plural,3,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
        defender,defenderan).

lex_tv(defend,defending,gerund,[_,_,anim_y,_,_,wing_n,_,_],[_,_,anim_y,_,_,_,_,_],
        defender,defendiendo).



lex_tv(eat,eat,present,[singular,1,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,como).
lex_tv(eat,eat,present,[singular,2,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,comes).
lex_tv(eat,eats,present,[singular,3,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,come).
lex_tv(eat,eat,present,[plural,1,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,comemos).
lex_tv(eat,eat,present,[plural,2,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,comeis).
lex_tv(eat,eat,present,[plural,3,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,comen).

lex_tv(eat,ate,past,[singular,1,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,comi).
lex_tv(eat,ate,past,[singular,2,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,comiste).
lex_tv(eat,ate,past,[singular,3,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,comio).
lex_tv(eat,ate,past,[plural,1,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,comimos).
lex_tv(eat,ate,past,[plural,2,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,comisteis).
lex_tv(eat,ate,past,[plural,3,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,comieron).

lex_tv(eat,will_eat,future,[singular,1,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,comere).
lex_tv(eat,will_eat,future,[singular,2,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,comeras).
lex_tv(eat,will_eat,future,[singular,3,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,comera).
lex_tv(eat,will_eat,future,[plural,1,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,comeremos).
lex_tv(eat,will_eat,future,[plural,2,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,comereis).
lex_tv(eat,will_eat,future,[plural,3,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,comeran).

lex_tv(eat,eating,gerund,[_,_,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,meal_y],comer,comiendo).



lex_tv(find,find,present,[singular,1,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
        encontrar,encuentro).
lex_tv(find,find,present,[singular,2,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
        encontrar,encuentras).
lex_tv(find,finds,present,[singular,3,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
        encontrar,encuentra).
lex_tv(find,find,present,[plural,1,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
        encontrar,encontramos).
lex_tv(find,find,present,[plural,2,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
        encontrar,encontrais).
lex_tv(find,find,present,[plural,3,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
        encontrar,encuentran).

lex_tv(find,found,past,[singular,1,anim_y,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
        encontrar,encontre).
```

```
lex_tv(find,found,past,[singular,2,anim_y,_,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
       encontrar,encontraste).
lex_tv(find,found,past,[singular,3,anim_y,_,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
       encontrar,encontro).
lex_tv(find,found,past,[plural,1,anim_y,_,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
       encontrar,encontramos).
lex_tv(find,found,past,[plural,2,anim_y,_,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
       encontrar,encontrasteis).
lex_tv(find,found,past,[plural,3,anim_y,_,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
       encontrar,encontraron).

lex_tv(find,will_find,future,[singular,1,anim_y,_,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
       encontrar,encontrare).
lex_tv(find,will_find,future,[singular,2,anim_y,_,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
       encontrar,encontraras).
lex_tv(find,will_find,future,[singular,3,anim_y,_,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
       encontrar,encontrara).
lex_tv(find,will_find,future,[plural,1,anim_y,_,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
       encontrar,encontraremos).
lex_tv(find,will_find,future,[plural,2,anim_y,_,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
       encontrar,encontrareis).
lex_tv(find,will_find,future,[plural,3,anim_y,_,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
       encontrar,encontraran).

lex_tv(find,finding,gerund,[_,_,_,anim_y,_,_,_,wing_n,_,_],[_,_,_,_,_,_,_,_],
       encontrar,encontrando).
```

```
lex_tv(hear,hear,present,[singular,1,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       oir,oigo).
lex_tv(hear,hear,present,[singular,2,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       oir,oyes).
lex_tv(hear,hears,present,[singular,3,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       oir,oye).
lex_tv(hear,hear,present,[plural,1,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       oir,oimos).
lex_tv(hear,hear,present,[plural,2,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       oir,ois).
lex_tv(hear,hear,present,[plural,3,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       oir,oyen).

lex_tv(hear,heard,past,[singular,1,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       oir,oi).
lex_tv(hear,heard,past,[singular,2,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       oir,oiste).
lex_tv(hear,heard,past,[singular,3,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       oir,oyo).
lex_tv(hear,heard,past,[plural,1,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       oir,oimos).
lex_tv(hear,heard,past,[plural,2,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       oir,oisteis).
```

```
lex_tv(hear,heard,past,[plural,3,anim_y,_,_,_,_,_],[_,_,_,_,_,_,loc_n,meal_n],
       oir,oyeron).

lex_tv(hear,will_hear,future,[singular,1,anim_y,_,_,_,_,_],[_,_,_,_,_,_,loc_n,meal_n],
       oir,oire).
lex_tv(hear,will_hear,future,[singular,2,anim_y,_,_,_,_,_],[_,_,_,_,_,_,loc_n,meal_n],
       oir,oiras).
lex_tv(hear,will_hear,future,[singular,3,anim_y,_,_,_,_,_],[_,_,_,_,_,_,loc_n,meal_n],
       oir,oira).
lex_tv(hear,will_hear,future,[plural,1,anim_y,_,_,_,_,_],[_,_,_,_,_,_,loc_n,meal_n],
       oir,oiremos).
lex_tv(hear,will_hear,future,[plural,2,anim_y,_,_,_,_,_],[_,_,_,_,_,_,loc_n,meal_n],
       oir,oireis).
lex_tv(hear,will_hear,future,[plural,3,anim_y,_,_,_,_,_],[_,_,_,_,_,_,loc_n,meal_n],
       oir,oiran).

lex_tv(hear,hearing,gerund,[_,_,anim_y,_,_,_,_,_],[_,_,_,_,_,_,loc_n,meal_n],
       oir,oyendo).




lex_tv(listen,listen,present,[singular,1,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_],
       escuchar,escucho).
lex_tv(listen,listen,present,[singular,2,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_],
       escuchar,escuchas).
lex_tv(listen,listens,present,[singular,3,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_],
       escuchar,escucha).
lex_tv(listen,listen,present,[plural,1,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_],
       escuchar,escuchamos).
lex_tv(listen,listen,present,[plural,2,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_],
       escuchar,escuchais).
lex_tv(listen,listen,present,[plural,3,anim_y,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_],
       escuchar,escuchan).

lex_tv(listen,listened,past,[singular,1,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_],
       escuchar,escuche).
lex_tv(listen,listened,past,[singular,2,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_],
       escuchar,escuchaste).
lex_tv(listen,listened,past,[singular,3,anim_y,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_],
       escuchar,escucho).
lex_tv(listen,listened,past,[plural,1,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_],
       escuchar,escuchamos).
lex_tv(listen,listened,past,[plural,2,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_],
       escuchar,escuchasteis).
lex_tv(listen,listened,past,[plural,3,anim_y,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_],
       escuchar,escucharon).

lex_tv(listen,will_listen,future,[singular,1,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_],
       escuchar,escuchare).
lex_tv(listen,will_listen,future,[singular,2,_,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_],
       escuchar,escucharas).
lex_tv(listen,will_listen,future,[singular,3,anim_y,hum_y,_,_,_,_,_],[_,_,anim_y,_,_,_,_,_],
```

```
        escuchar, escuchara).
lex_tv(listen, will_listen, future, [plural, 1, _, hum_y, _, _, _, _, _], [_, _, anim_y, _, _, _, _, _],
        escuchar, escucharemos).
lex_tv(listen, will_listen, future, [plural, 2, _, hum_y, _, _, _, _, _], [_, _, anim_y, _, _, _, _, _],
        escuchar, escucharéis).
lex_tv(listen, will_listen, future, [plural, 3, anim_y, hum_y, _, _, _, _, _], [_, _, anim_y, _, _, _, _, _],
        escuchar, escucharán).

lex_tv(listen, listening, gerund, [_, _, anim_y, hum_y, _, _, _, _, _], [_, _, anim_y, _, _, _, _, _],
        escuchar, escuchando).




lex_tv(love, love, present, [singular, 1, _, hum_y, _, _, _, _, _], [_, _, _, _, _, _, _, loc_n, _],
        amar, amo).
lex_tv(love, love, present, [singular, 2, _, hum_y, _, _, _, _, _], [_, _, _, _, _, _, _, loc_n, _],
        amar, amas).
lex_tv(love, loves, present, [singular, 3, _, hum_y, _, _, _, _, _], [_, _, _, _, _, _, _, loc_n, _],
        amar, ama).
lex_tv(love, love, present, [plural, 1, _, hum_y, _, _, _, _, _], [_, _, _, _, _, _, _, loc_n, _],
        amar, amamos).
lex_tv(love, love, present, [plural, 2, _, hum_y, _, _, _, _, _], [_, _, _, _, _, _, _, loc_n, _],
        amar, amais).
lex_tv(love, love, present, [plural, 3, _, hum_y, _, _, _, _, _], [_, _, _, _, _, _, _, loc_n, _],
        amar, aman).

lex_tv(love, loved, past, [singular, 1, _, hum_y, _, _, _, _, _], [_, _, _, _, _, _, _, loc_n, _],
        amar, amé).
lex_tv(love, loved, past, [singular, 2, _, hum_y, _, _, _, _, _], [_, _, _, _, _, _, _, loc_n, _],
        amar, amaste).
lex_tv(love, loved, past, [singular, 3, _, hum_y, _, _, _, _, _], [_, _, _, _, _, _, _, loc_n, _],
        amar, amó).
lex_tv(love, loved, past, [plural, 1, _, hum_y, _, _, _, _, _], [_, _, _, _, _, _, _, loc_n, _],
        amar, amamos).
lex_tv(love, loved, past, [plural, 2, _, hum_y, _, _, _, _, _], [_, _, _, _, _, _, _, loc_n, _],
        amar, amasteis).
lex_tv(love, loved, past, [plural, 3, _, hum_y, _, _, _, _, _], [_, _, _, _, _, _, _, loc_n, _],
        amar, amaron).

lex_tv(love, will_love, future, [singular, 1, _, hum_y, _, _, _, _, _], [_, _, _, _, _, _, _, loc_n, _],
        amar, amaré).
lex_tv(love, will_love, future, [singular, 2, _, hum_y, _, _, _, _, _], [_, _, _, _, _, _, _, loc_n, _],
        amar, amarás).
lex_tv(love, will_love, future, [singular, 3, _, hum_y, _, _, _, _, _], [_, _, _, _, _, _, _, loc_n, _],
        amar, amará).
lex_tv(love, will_love, future, [plural, 1, _, hum_y, _, _, _, _, _], [_, _, _, _, _, _, _, loc_n, _],
        amar, amaremos).
lex_tv(love, will_love, future, [plural, 2, _, hum_y, _, _, _, _, _], [_, _, _, _, _, _, _, loc_n, _],
        amar, amaréis).
lex_tv(love, will_love, future, [plural, 3, _, hum_y, _, _, _, _, _], [_, _, _, _, _, _, _, loc_n, _],
        amar, amarán).
```

```
lex_tv(love,loving,gerund,[_,_,_,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,_,loc_n,_],
       amar,amando).



lex_tv(see,see,present,[singular,1,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,vi).
lex_tv(see,see,present,[singular,2,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,ves).
lex_tv(see,sees,present,[singular,3,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,ve).
lex_tv(see,see,present,[plural,1,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,vemos).
lex_tv(see,see,present,[plural,2,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,veis).
lex_tv(see,see,present,[plural,3,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,ven).

lex_tv(see,saw,past,[singular,1,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,vi).
lex_tv(see,saw,past,[singular,2,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,viste).
lex_tv(see,saw,past,[singular,3,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,vio).
lex_tv(see,saw,past,[plural,1,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,vimos).
lex_tv(see,saw,past,[plural,2,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,visteis).
lex_tv(see,saw,past,[plural,3,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,vieron).

lex_tv(see,will_see,future,[singular,1,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,vere).
lex_tv(see,will_see,future,[singular,2,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,veras).
lex_tv(see,will_see,future,[singular,3,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,vera).
lex_tv(see,will_see,future,[plural,1,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,veremos).
lex_tv(see,will_see,future,[plural,2,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,vereis).
lex_tv(see,will_see,future,[plural,3,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,veran).

lex_tv(see,seeing,gerund,[_,_,anim_y,_,_,_,_,_,_],[_,_,_,_,_,_,_,_,_,_],
       ver,viendo).



lex_tv(study,study,present,[singular,1,_,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudio).
```

```
lex_tv(study,study,present,[singular,2,_,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudias).
lex_tv(study,studies,present,[singular,3,anim_y,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudia).
lex_tv(study,study,present,[plural,1,_,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudiamos).
lex_tv(study,study,present,[plural,2,_,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudiais).
lex_tv(study,study,present,[plural,3,anim_y,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudian).

lex_tv(study,studied,past,[singular,1,_,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudie).
lex_tv(study,studied,past,[singular,2,_,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudiaste).
lex_tv(study,studied,past,[singular,3,anim_y,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudio).
lex_tv(study,studied,past,[plural,1,_,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudiamos).
lex_tv(study,studied,past,[plural,2,_,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudiasteis).
lex_tv(study,studied,past,[plural,3,anim_y,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudiaron).

lex_tv(study,will_study,future,[singular,1,_,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudiare).
lex_tv(study,will_study,future,[singular,2,_,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudiaras).
lex_tv(study,will_study,future,[singular,3,anim_y,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudiara).
lex_tv(study,will_study,future,[plural,1,_,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudiaremos).
lex_tv(study,will_study,future,[plural,2,_,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudiareis).
lex_tv(study,will_study,future,[plural,3,anim_y,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudiaran).

lex_tv(study,studing,gerund,[_,_,anim_y,hum_y,_,_,_,_,_],[_,_,_,_,_,_,_,loc_n,meal_n],
       estudiar,estudiando).
```

A P E N D I C E B

PRUEBAS HECHAS AL SISTEMA DE TRADUCCION AUTOMATICA

```
                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
Alex spoke

  alex spoke

  alejandro hablo

[ENTER]


                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
fido spoke

  fido spoke

  **** Invalid English Sentence ****

  (*) fido spoke

[ENTER]


                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
Peter that sings smoked

  peter that sings smoked

  pedro que canta fumo

[ENTER]


                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
Susan that played in the yard will_run

  susan that played in the yard will_run

  susana que jugo en el patio correra

[ENTER]
```

```
                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
Chris spoke so sincerely

  chris spoke so sincerely

  cristina hablo tan sinceramente

[ENTER]


                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
the car speaks with the boy

  the car speaks with the boy

  **** Invalid English Sentence ****

  (*) the car speaks with the boy

[ENTER]


                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
Alice fly in California

  alice fly in california

  **** Invalid English Sentence ****

  (*) alice fly in california

[ENTER]

                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
Alice flies in California

  alice flies in california

  alicia vuela en california

[ENTER]
```

```
                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
Sam runs very well

  sam runs very well

  samuel corre muy bien

[ENTER]


                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
Norma lives with John

  norma lives with john

  norma vive con juan

[ENTER]


                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
they that smoke sang

  they that smoke sang

  ellos que fuman cantaron

[ENTER]


                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
some apple disappeared

  some apple disappeared

  **** Invalid English Sentence ****

  (*) some apple disappeared

[ENTER]
```

```
                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
some apples disappeared

  some apples disappeared

  algunas manzanas desaparecieron

[ENTER]


                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
the policeman slept in the park

  the policeman slept in the park

  el policia durmio en el parque

[ENTER]


                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
Lucy admires the birds in the yard

  lucy admires the birds in the yard

  lucia admira a las aves en el patio

[ENTER]


                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
Charles loves Laura

  charles loves laura

  carlos ama a laura

[ENTER]
```

```
                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
Mary lives with Sam that smokes with the thief

  mary lives with sam that smokes with the thief

  maria vive con samuel que fuma con el ladron

[ENTER]


                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
the intelligent man will_study in the university

  the intelligent man will_study in the university

  el hombre inteligente estudiara en la universidad

[ENTER]

                    ENGLISH - SPANISH TRANSLATION

Please type in the sentence:
     ('end' to finish)
the intelligent dog will_study in the university

  the intelligent dog will_study in the university

  **** Invalid English Sentence ****

  (*) the intelligent dog will_study in the university

[ENTER]
```

```
                    TRADUCCION ESPAÑOL - INGLES

Por favor teclee la oraciòn:
     ('fin' para terminar)
Jose fuma contigo

  jose fuma contigo

  joseph smokes with you

[ENTER]


                    TRADUCCION ESPAÑOL - INGLES

Por favor teclee la oraciòn:
     ('fin' para terminar)
Piolin volo lentamente

  piolin volo lentamente

  piolin flew slowly

[ENTER]


                    TRADUCCION ESPAÑOL - INGLES

Por favor teclee la oraciòn:
     ('fin' para terminar)
el gato vuela en el parque

  el gato vuela en el parque

  **** Oraciòn EspaÑola Incorrecta ****

  (*) el gato vuela en el parque

[ENTER]


                    TRADUCCION ESPAÑOL - INGLES

Por favor teclee la oraciòn:
     ('fin' para terminar)
Lassie corre con Fido

  lassie corre con fido

  lassie runs with fido

[ENTER]
```

```
                    TRADUCCION ESPAÑOL - INGLES

Por favor teclee la oraciòn:
     ('fin' para terminar)
nosotros vivimos con ellas

  nosotros vivimos con ellas

  we live with them

[ENTER]


                    TRADUCCION ESPAÑOL - INGLES

Por favor teclee la oraciòn:
     ('fin' para terminar)
juan corta las hamburguesas con un cuchillo

  juan corta las hamburguesas con un cuchillo

  john cuts the burgers with a knife

[ENTER]


                    TRADUCCION ESPAÑOL - INGLES

Por favor teclee la oraciòn:
     ('fin' para terminar)
el auto duerme en el sofa

  el auto duerme en el sofa

  **** Oraciòn Española Incorrecta ****

  (*) el auto duerme en el sofa

[ENTER]


                    TRADUCCION ESPAÑOL - INGLES

Por favor teclee la oraciòn:
     ('fin' para terminar)
pocos estudiantes estudiaran en esa biblioteca

  pocos estudiantes estudiaran en esa biblioteca

  few students will_study in that library

[ENTER]
```

TRADUCCION ESPAÑOL - INGLES

```
Por favor teclee la oraciòn:
     ('fin' para terminar)
memo que compro el auto vive en cancun

  memo que compro el auto vive en cancun

  billy that bought the car lives in cancun

[ENTER]
```

TRADUCCION ESPAÑOL - INGLES

```
Por favor teclee la oraciòn:
     ('fin' para terminar)
el gato cazo a el pajaro

  el gato cazo a el pajaro

  the cat chased the bird

[ENTER]
```

TRADUCCION ESPAÑOL - INGLES

```
Por favor teclee la oraciòn:
     ('fin' para terminar)
la hermosa mujer que canta vive conmigo

  la hermosa mujer que canta vive conmigo

  the beautiful woman that sings lives with me

[ENTER]
```

TRADUCCION ESPAÑOL - INGLES

```
Por favor teclee la oraciòn:
     ('fin' para terminar)
los ninos comen los tacos con una coca_cola

  los ninos comen los tacos con una coca_cola

  the boys eat the tacos with a coke

[ENTER]
```

TRADUCCION ESPAÑOL - INGLES

Por favor teclee la oraciòn:
('fin' para terminar)
las ninas rompen la ensalada

las ninas rompen la ensalada

**** Oraciòn Española Incorrecta ****

(*) las ninas rompen la ensalada

[ENTER]

TRADUCCION ESPAÑOL - INGLES

Por favor teclee la oraciòn:
('fin' para terminar)
las mariposas volaron alegremente en la escuela

las mariposas volaron alegremente en la escuela

the butterflies flew gladly in the school

[ENTER]

TRADUCCION ESPAÑOL - INGLES

Por favor teclee la oraciòn:
('fin' para terminar)
los ninos americanos comen los tacos muy cuidadosamente

los ninos americanos comen los tacos muy cuidadosamente

the american boys eat the tacos very carefully

[ENTER]

TRADUCCION ESPAÑOL - INGLES

Por favor teclee la oraciòn:
('fin' para terminar)
la nina pequena se comio una sabrosa manzana

la nina pequena comio una sabrosa manzana

the little girl ate a tasty apple

[ENTER]

TRADUCCION ESPAÑOL - INGLES

```
Por favor teclee la oraciòn:
     ('fin' para terminar)
los policias buenos atraparon a el ladron feo

  los policias buenos atraparon a el ladron feo

  the good policemen catch the ugly thief

[ENTER]
```

TRADUCCION ESPAÑOL - INGLES

```
Por favor teclee la oraciòn:
     ('fin' para terminar)
Mexico juega alegremente con alejandro

  Mexico juega alegremente con alejandro

  **** Oraciòn Española Incorrecta ****

  (*) Mexico juega alegremente con alejandro

[ENTER]
```

TRADUCCION ESPAÑOL - INGLES

```
Por favor teclee la oraciòn:
     ('fin' para terminar)
Silvestre cazo a Piolin

  silvestre cazo a piolin

  silvester catched piolin

[ENTER]
```

TRADUCCION ESPAÑOL - INGLES

```
Por favor teclee la oraciòn:
     ('fin' para terminar)
Rosa que vive en nueva_york defendio a los ninos

  rosa que vive en nueva_york defendio a los ninos

  rose that lives in new_york defended the boys

[ENTER]
```