21
2e/

Universidad Nacional Autónoma de México

Facultad de Ciencias

Tesis Profesional para obtener el título de Actuario

TITULO: "Análisis Sintáctico del Español por Computadora"

Benjamín Macías Pimentel

1985

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

DERECHOS RESERVADOS © PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis está protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Prólogo

La representación y proceso del lenguaje natural a través de la computadora es una de las ramas más destacadas de la inteligencia artificial, y se espera que cumpla un papel cada vez más importante, tanto para facilitar la comunicación con las computadoras, como para explorar distintos modelos de inteligencia.

Este trabajo tiene como objetivo la elaboración de una gramática del español enfocada al análisis sintáctico por computadora de oraciones del español, de modo que proporcione una herramienta obligatoria para el desarrollo de aplicaciones que usen al español como lenguaje de comunicación. Se espera, además, que la primera parte de la tesis sirva como introducción a temas relacionados con la comprensión de lenguaje natural por computadora, dada la carencia de bibliografía en español en el tema.

El origen del analizador sintáctico propuesto es el proyecto DEC -Diálogos en Español por Computadora-, un sistema experimental desarrollado en la Facultad de Ciencias bajo la dirección del Dr. Christian Lemaitre. DEC tiene como meta la implantación de un sistema de pregunta-respuesta sobre un banco de datos de geografía de México, utilizando al español como lenguaje de comunicación. De este proyecto se han derivado ya un analizador lexico, realizado por Miguel Tomasena, y el diseño del banco, junto con su lenguaje interno, por Ernesto Archundia.

El contenido del presente trabajo ha sido dividido en tres secciones: la primera contiene un esbozo histórico, seguido de la introducción de los elementos de trabajo; la segunda contiene la elaboración de la gramática, y la tercera, elabora el intérprete por computadora del modelo propuesto.

INDICE

I. Una Perspectiva del Tratamiento del Lenguaje Natural por Computadora

I.1 Introducción

En el verano de 1956, un grupo de científicos entre los que se contaban Marvin Minsky, John McCarthy, Nathaniel Rochester y Claude Shannon, hicieron una propuesta a la Fundación Rockefeller para financiar una conferencia a realizarse en Dartmouth. En su propuesta, los autores sugerían que "todos los aspectos del lenguaje o cualquier característica de la inteligencia" podrían ser simulados por una computadora. Quedaba fundada así una rama de la computación bautizada como 'inteligencia artificial' por sugerencia de McCarthy; la conferencia se realizó bajo el nombre de Dartmouth Summer Research Project on Artificial Intelligence, y tuvo una importancia decisiva /1/.

En el mismo año, Allen Newell, J. C. Shaw y Herbert Simon realizaron un programa conocido como el 'Lógico Teórico' -Logic Theorist-. Este programa, quizá el primer programa heurístico realizado completamente en una computadora, llevaba a cabo la demostración de teoremas de lógica simbólica /2/.

Fue así como surgió la rama de la computación llamada *inteligencia artificial*; esta tiene como meta el estudio de los procesos que hacen posible el comportamiento inteligente, tanto en el hombre como en las computadoras. El presente trabajo se concentra en el área llamada *comprensión de lenguaje natural*, cuyo objeto de estudio son aquellos modelos sintácticos y semánticos de los lenguajes naturales suceptibles de ser empleados por sistemas computacionales.

Es importante notar que la investigación en inteligencia artificial tiene un marcado sabor práctico; como consecuencia, tienen preponderancia las teorías que dan los elementos necesarios para construir sistemas que funcionen, aunque no se desecha la interacción teórica con otras disciplinas con intereses similares, como la psicología y la linguística. Además, se buscan sistemas que cumplan con un cierto nivel de competencia, sin importar si los seres humanos alcanzan el mismo nivel usando otras estrategias. De este modo, cuando se mencione en lo siguiente la frase *comprensión de lenguaje natural*, simplemente se quiere decir que la máquina emite respuestas apropiadas en lenguaje natural de un modo consistente.

I.2 Traducción Automática

El primer campo en el que se buscó aplicar la

computadora a procesos asociados al lenguaje natural fue el de la traducción automática. Entre los primeros autores están Booth en Inglaterra en 1948, y al Dr. Warren Weaver, director de Ciencias Naturales de la Fundación Rockefeller, en 1949 /3/.

En los cincuentas, un grupo de investigadores guiados por el lógico y linguista Yehoshua Bar-Hillel intentó producir sistemas para llevar a cabo traducción automática de alta calidad. Sin embargo, el propio Bar-Hillel llegó en 1960 a la conclusión que la meta fijada era inalcazable, al menos en el futuro cercano /4/.

De este modo, al tiempo que se desarrollaban al menos una veintena de proyectos financiados por el gobierno norteamericano, un comité formado por la National Science Foundation, influido por las conclusiones de Bar-Hillel, terminó por muchos años con el patrocinio oficial a proyectos de traducción automática /5/.

El enfoque empleado era el siguiente: el traductor automático estaba dividido en dos fases, una que recuperaba de un diccionario la traducción al lenguaje destino de cada palabra del lenguaje destino; otra que corregía aspectos sintácticos por medio de una gramática: el orden de las palabras, su morfología, etc. Pero pronto se vió que sin tener en cuenta el significado del texto que se buscaba traducir era imposible hacerlo coherentemente. Para ponerlo en las palabras de Bertram Raphael /6/, los experimentos fallaron miserablemente: al pedírsele a uno de dichos sistemas que tradujera al ruso del inglés y de regreso al inglés la frase

"el espíritu está dispuesto pero la carne es débil",

el resultado fue:

"el vino es agradable pero la carne está podrida".

I.3 Primeros Sistemas de Comprensión de Lenguaje Natural (1961-1967)

El fracaso de proyectos de traducción automática influyó para que los nuevos sistemas pusieran el énfasis en el terreno más conocido: la sintaxis; quedando reducidas al mínimo otras consideraciones.

I.3.1 Caza de Patrones

Un patrón es una especificación incompleta de la estructura de una oración; v. gr., "Juan es doctor", "el día está nublado", pueden verse como parte de la categoría de

frases que tienen como patrón /7/: ".....ES.....".

El siguiente conjunto de patrones era usado por uno de los primeros sistemas:

.....es.....
.....tiene.....
.....posee.....
¿.....dónde está.....?
¿ es.....parte de.....?
.....está a la izquierda de.....
¿ está.....a la derecha de.....?
¿ cuántos.....tiene.....?

La mayoría de los primeros sistemas de comprensión de lenguaje natural favorecía este tipo de análisis, ya que son sencillos de implantar y un uso juicioso de ellos produce resultados convincentes. Esto es especialmente cierto en los casos en los que el programa no depende de modo crucial de un análisis sintáctico exhaustivo.

Un buen ejemplo del uso de esta técnica es el sistema ELIZA, realizado por Joseph Weizenbaum en 1966 /8/. Este se proponía el modelado de un diálogo entre un psiquiatra -ELIZA-, y su paciente -el usuario. El siguiente ejemplo muestra el desempeño de ELIZA (la parte en mayúsculas es la emitida por la máquina):

Todos los hombres son iguales.
DE QUE MANERA
Siempre están molestando de un modo u otro.
PUEDES DAR UN EJEMPLO ESPECIFICO
Bueno, mi novio me hizo venir aquí.
TU NOVIO TE HIZO VENIR
Dice que yo estoy deprimida la mayor parte del tiempo.
SIENTO SABER QUE TU ESTAS DEPRIMIDA
Es verdad, yo soy infeliz.
CREES QUE EL VENIR TE AYUDARA A NO SER INFELIZ
Yo necesito ayuda, eso parece seguro.
QUE SIGNIFICARIA PARA TI EL CONSEGUIR AYUDA
Quizá yo podría aprender a llevarme bien con mi madre.
DIME MAS ACERCA DE TU FAMILIA
Mi madre me cuida.
QUIEN MAS EN TU FAMILIA TE CUIDA
Mi padre.
TU PADRE
Usted es como mi padre en algunas cosas.
QUE PARECIDO ENCUENTRAS
Usted no es muy agresivo pero yo creo que usted no quiere que yo me de cuenta.
QUE TE HACE PENSAR QUE YO NO SOY MUY AGRESIVO
Usted no discute conmigo.
PORQUE CREES QUE YO NO DISCUTO CONTIGO
Usted tiene miedo de mí.

TE DA GUSTO CREER QUE YO TENGO MIEDO DE TI
Mi padre tiene miedo de todo el mundo.
QUE MAS TE VIENE A LA MENTE CUANDO PIENSAS EN TU PADRE
Fanfarrones.
TIENE ESO QUE VER CON EL HECHO DE QUE TU NOVIO TE HIZO VENIR
...

ELIZA se apoyaba para su desempeño en un sistema de caza de patrones muy sencillo, junto con un análisis mínimo del contexto.

Por ejemplo, la oración de entrada "usted es muy útil" pasa por el siguiente proceso:

---->"yo es muy útil"
el patrón "usted....." pasa a "yo....."
---->"qué te hace pensar que yo soy muy útil"
el patrón ".....yo es....." pasa a "qué te hace pensar que yo soy....."

ELIZA contaba con varias estrategias para detectar contextos como los del ejemplo anterior; v. gr., pasa de "usted es" a "yo soy". Además, para los momentos de apuro, el psquiatra contaba con una serie de frases aplicables a cualquier situación:

POR FAVOR, SIGUE
YA VEO
ES MUY INTERESANTE.

A pesar de la modestia de las metas de Weizenbaum, y de carecer ELIZA de cualquier conocimiento del mundo, el programa lograba mantener un diálogo sumamente vívido, con un desenvolvimiento que desafía al de los sistemas mucho más sofisticados de la década siguiente. Una lección importante de ELIZA es que la calidad de un diálogo no necesariamente refleja la del sistema en el que se sostiene.

Otro sistema de la misma época basado en este método es STUDENT, realizado por Daniel Bobrow en 1964 en el MIT. STUDENT resolvía problemas algebráicos pequeños partiendo de un sistema de patrones muy sencillo /9/. Uno de los problemas que STUDENT era capaz de resolver es el siguiente:

> Si el número de clientes que Tom consigue es el doble del cuadrado del 20 por ciento del número de anuncios que coloca, y el número de anuncios que coloca es de 45, cúal es el número de clientes que Tom consigue. /10/

También enfocado al proceso del lenguaje natural

estaba SIR -Semantic Information Processing-, cuyo propósito era el de realizar deducciones sencillas en base a una serie de hechos. Este programa fué desarrollado por Bertram Raphael en 1964, también en el MIT /11/.

Un ejemplo más reciente de aplicación de caza de patrones es el sistema PARRY, desarrollado por un grupo de investigadores dirigidos por Kenneth Colby, interesados en modelar el comportamiento linguístico de un paciente paranóico /12/.

Probablemente PARRY es el sistema que mayor uso ha hecho de la técnica de caza de patrones -varios miles de ellos-, además de cientos de frases hechas. Debe considerarse también que, dado que el paciente está loco, no es imprescindible la precisión de un anális que este método no proporciona. Como ejemplo, se tiene el siguiente diálogo /13/:

¿Cómo estás hoy?
OK.
¿Cómo te llamas?
ME LLAMO FRANK SMITH.
¿Cuántos años tienes?
VEINTIOCHO.
¿Estás casado?
NO.
¿Qué tipo de trabajo haces?
TRABAJO EN LA OFICINA POSTAL.
¿Porqué estás en el hospital?
NO DEBERIA ESTAR AQUI.
¿Quién te trajo aquí?
LA POLICIA.
¿Qué problemas tuviste con la policía?
LOS TIRAS NO HACEN SU TRABAJO.
¿Y que hacen en lugar de eso?
LOS TIRAS ARRESTAN A LA GENTE EQUIVOCADA.
Ellos arrestan a veces a la gente correcta.
LA LEY DEBERIA AGARRAR A ESOS PILLOS ITALIANOS.
¿Eres italiano?
MIS ANTECEDENTES SON BRITANICOS PERO YO NACI EN ESTE PAIS.
¿Conoces algún pillo italiano?
TRATO DE EVITAR AL BAJO MUNDO.
¿Tienes miedo del bajo mundo?
LOS DE LA MAFIA SON LOS PEORES GANGSTERS.
¿Estás conectado con la mafia?
ELLOS ME CONOCEN, SABES.
¿Cómo te conocen?
ME ESPIAN.
¿Qué quieren?
QIEREN EVIDENCIA FALSA PARA INCRIMINARME.
¿Qué tienen en contra tuya?
ANDAN TRAS DE MI.
¿Para qué?

TENGO EVIDENCIA COMPROMETEDORA PARA LA MAFIA.
¿Estás seguro de todo esto?
NO ME CREES?
...

I.3.2 Reglas de Tipo Estructuralista

Otro enfoque usado en esta época fue el de utilizar estrategias para el análisis sintáctico similares a las de los linguistas estructuralistas. Un representante de esta técnica es el sistema BASEBALL, diseñado para recuperar información relativa a dicho deporte de un banco de datos ad hoc /14/.

BASEBALL aceptaba preguntas sencillas relativas a fechas, lugares, equipos, marcadores, etc. Las preguntas debían ser simples, sin contener cláusulas relativas ni ningún tipo de conjunciones.

El análisis sintáctico de las preguntas fue desarrollado en base a reglas del linguista Zellig Harris y sus asociados de la Universidad de Pennsylvannia.

El análisis semántico, por su parte, contenía dos tipos de rutinas: unas encargadas de resolver ambigüedades del significado de una palabra, y otras para agregar atributos a los elementos del análisis. Así, para la pregunta " quiéen venció a los Yanquis el 4 de julio?", rutinas del primer tipo recuperarían:

EQUIPO = Yanquis
EQUIPO = ?,

mientas que una rutina del segundo tipo (asociada a "vencer") entregaría:

EQUIPO perdedor = Yanquis
EQUIPO ganador = ?,

información necesaria para el proceso de búsqueda en el banco.

I.3.3 Gramáticas Libres de Contexto

Siguiendo el trabajo de Noam Chomsky, uno de los primeros en aplicar dicho formalismo fue A. V. Phillips en un sistema de pregunta-respuesta desarrollado en el MIT en 1960. Dicho sistema aceptaba oraciones muy sencillas, restringiéndose a aquellas sin ningún tipo de ambiguedad /15/.

En 1962, Susumo Kuno y Anthony Oettinger desarrollaron uno de los primeros analizadores sintácticos utilizando una gramática libre de contexto y un reconocedor hacia abajo con retroceso /16/: el Harvard Syntactic Analyzer. Este contaba con unas tres mil reglas gramaticales

para describir el inglés, y entregaba todos los análisis posibles de las oraciones ambiguas. El sistema podía aalizar oraciones tan complejas como:

> "Gravemente preocupado con la violencia creciente, el presidente Kennedy usó su conferencia de prensa para dar consejos a ambos lados en lucha." /17/

I.4 Sistemas de Segunda Generación: LUNAR, SHRDLU (1967-1972)

Los sistemas anteriormente descritos establecieron con elocuencia, después del fracaso de los primeros traductores mecánicos, la posibilidad real de comprender oraciones en lenguaje natural. El nuevo grupo de programas atacó problemas más complicados; el modelo de trabajo continuó siendo el de los sistema de pregunta-respuesta.

I.4.1 LUNAR

El primer sistema representativo de esta generación es LUNAR (1967-1972), desarrollado por William Woods /18/. LUNAR es un sistema de pregunta-respuesta cuya misión es obtener información sobre muestras geológicas traidas de la luna, de un banco de datos de la NASA.

El énfasis de LUNAR estaba puesto en obtener un análisis sintáctico exhaustivo para alimentar adecuadamente al componente encargado de llevar a cabo la recuperación de la información. Para ello fue usado por primera vez de un modo extenso el formalismo de las redes de transición aumentada, o ATNs -Augmented Transition Networks- /19/.

El sistema fue implantado en LISP y obtenía como un primer paso la estructura profunda (tradicional) que posteriormente era traducido al lenguaje de búsqueda. Entre los fenómenos sintácticos resueltos por LUNAR se cuentan: algunas formas de referencias anafóricas, elipsis, tipos de complemento, claúsulas relativas reducidas y no reducidas, oraciones comparativas y pasivas.

En enero de 1971, Woods expuso su sistema en la Second Annual Lunar Science Conference, y LUNAR fue capaz de contestar un 90% de las 110 preguntas formuladas por los geólogos asistentes a la conferencia.

Para dar una idea del desempeo de LUNAR citamos el siguiente diálogo (la parte de la máquina en mayúsculas) /20/:

¿cuántas brechas contienen olivino?
S10019, S10059, S10065, S10067, S10073.
lista las razones k/rb para las brechas.
TENGO 17 CANDIDATOS. ¿QUIERES VERLOS?
si.
472.2222 S10018 D70-205
... ...
500.173 S10073 D70-215
Lista los análisis con lámina modal para las muestras lunares que contienen oliv.
TENGO 13 CANDIDATOS. ¿QUIERES VERLOS?
...

I.4.2 SHRDLU

Un sistema que significó un avance cualitativo sobre los esfuerzos anteriores es SHRDLU, producido a principios de los setentas por Terry Winograd, del MIT (el nombre fue tomado de las máquinas de linotipo, donde la frase 'ETAOIN SHRDLU' era usada por los operadores para señalar errores tipográficos) /21/.

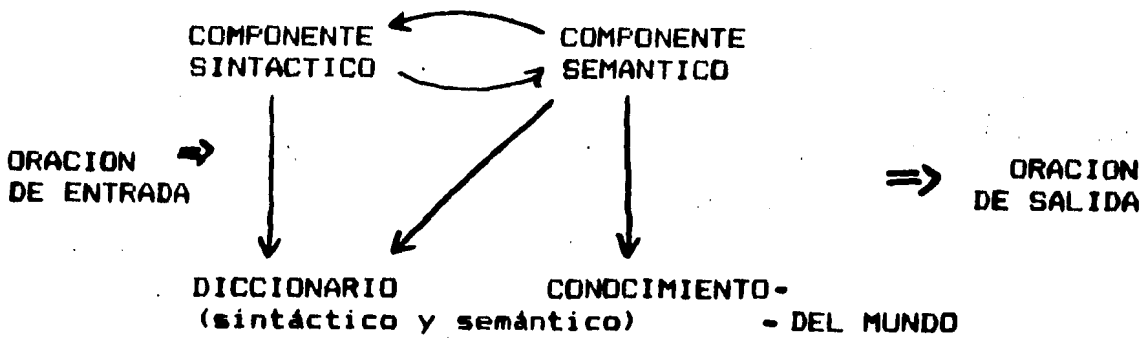
SHRDLU representa un esfuerzo por realizar un sistema de comprensión de lenguaje natural trabajando en un universo muy limitado, poniendo su atención en los problemas para manejar un diálogo.

El tema elegido para el sistema es el siguiente: un brazo mecánico debe recibir instrucciones a través de un diálogo en inglés para manipular -tomar, soltar, encimar,cambiar de sitio- objetos sencillos, como cajas, esferas, pirámides.

El sistema fue escrito originalmente en LISP, y ocupaba unos 100K de memoria. Cada oración tomaba en promedio 10 ó 15 segundos en responderse. El mundo del robot era simulado en una pantalla, mientras la conversación se llevaba a cabo en una terminal ordinaria.

SHRDLU es uno de los sistemas de comprensión de lenguaje natural jamás creado. Llegó a ser tan difícil el predecir su comportamiento sin hacer pruebas, que después de servir para la tesis doctoral de Winograd, fue abandonado.

SHRDLU puede dividirse, como en los sistemas tradicionales, en un analizador sintáctico, en un conjunto de rutinas semánticas, y en sistema deductivo encargado de hacer planes, obtener conclusiones, y responder preguntas. Sin embargo, el diseño del sstema combina dichos componentes de un modo inusual: en lugar de llamar por turnos a cada componente del sistema para que éste cumpla con su cometido por separado, el sistema está organizado para permitir la interacción de sus componentes a distintos niveles y en cualquier momento /22/:



A continuación se presenta un diálogo donde se pueden apreciar las facetas del sistema (la parte de SHRDLU va en mayúsculas) /23/:

Levanta un cubo rojo grande.
OK.
Agarra la pirámide.
NO COMPRENDO A QUE PIRAMIDE TE REFIERES.
Encuentra un bloque que es mayor al que estás deteniendo y ponlo en la caja.
POR 'LO' ASUMO QUE TE REFIERES AL BLOQUE QUE ES MAYOR QUE EL QUE ESTOY DETENIENDO.
OK.
¿Qué contiene la caja?
LA PIRAMIDE AZUL Y EL BLOQUE AZUL.
¿Por qué está soportada la caja?
LA CAJA.
¿Cuántos bloques no están en la caja?
CUATRO DE ELLOS.
¿Es al menos uno de ellos más estrecho que el que te pedí que levantaras?
SI. EL CUBO ROJO.
¿Está soportado?
SI. POR LA MESA.
¿Puede la mesa levantar objetos?
NO.
¿Puede una pirámide ser soportada por un bloque?
SI.
¿Puede una pirámide soportar una pirámide?
NO LO SE.
Apila dos pirámides.
NO PUEDO.
La pirámide azul es mía.
COMPRENDO.
Me pertenecen todos los bloques que no son rojos, pero no me pertenece nada que soporte a una pirámide.
COMPRENDO.
¿Me pertenece la caja?

NO.
¿Me pertenece cualquier cosa en la caja?
SI. DOS COSAS: EL BLOQUE AZUL Y LA PIRAMIDE AZUL.
¿Podrías por favor apilar ambos de los cubos rojos y ya sea un cubo verde o una pirámide?
OK.
¿Cuál cubo yace en la mesa?
EL VERDE GRANDE QUE SOPORTA A LA PIRAMIDE ROJA.
¿Hay un cubo grande detrás de una pirámide?
SI, TRES DE ELLOS: UNO ROJO GRANDE, UN CUBO VERDE GRANDE Y EL AZUL.
Pon uno chico sobre el cubo verde que soporta una pirámide.
OK.
Pon la pirámide más pequeña en la punta de todo.
OK.
¿Cuántas cosas están encima de cubos verdes?
NO ESTOY SEGURO DE QUE QUIERES DECIR CON "ENCIMA DE" EN LA FRASE "ENCIMA DE CUBOS VERDES"
QUIERES DECIR:
1- DIRECTAMENTE EN LA SUPERFICIE
2- CUALQUIER COSA QUE ESTE ENCIMA DE
¿Has tocado alguna pirámide antes de poner la verde en el cubo pequeño?
SI, LA VERDE.
¿Cuándo la tomaste?
MIENTRAS ESTABA APILANDO EL CUBO ROJO, UN BLOQUE ROJO GRANDE Y UN CUBO VERDE GRANDE.
¿Porqué?
PARA DESHACERME DE EL.
¿Porqué hiciste eso?
PARA DESPEJAR EL CUBO ROJO.
¿Porqué despejaste ese cubo?
PARA PONER EN EL UN CUBO VERDE GRANDE.
¿Porqué hiciste eso?
PARA APILAR EL CUBO ROJO, UN BLOQUE ROJO GRANDE Y UN CUBO VERDE GRANDE.
¿Porqué hiciste eso?
PORQUE TU ME LO PEDISTE.
¿Cómo lo hiciste?
PONIENDO UN BLOQUE ROJO GRANDE EN LA MESA; DESPUES SOLTANDOLO; DESPUES PONIENDO UN CUBO VERDE GRANDE EN EL; DESPUES SOLTANDO ESE CUBO; DESPUES PONIENDO EL CUBO ROJO EN ESE CUBO; DESPUES PONIENDO EL CUBO ROJO EN ESE CUBO; DESPUES SOLTANDO ESE CUBO.
¿Cuántos objetos tocaste mientras hacías eso?
CUATRO DE ELLOS.
¿Qué soportaba el cubo rojo antes de que comenzaras a despejarlo?
LA PIRAMIDE VERDE.
Había entonces cinco bloques a la izquierda de la caja.
NO, SOLO CUATRO DE ELLOS: EL CUBO ROJO, DOS CUBOS VERDES GRANDES Y UN BLOQUE ROJO GRANDE.
Pon la pirámide azul en el bloque en la caja.
OK.

¿Hay algo que sea mayor que cualquier pirámide pero no tan ancho como lo que soporta?
SI, EL BLOQUE AZUL.
Una torre...
LO SIENTO, NO CONOZCO LA PALABRA "TORRE".
Una torre es una pila que contiene dos cubos verdes y una pirámide.
COMPRENDO.
¿Hay ahora alguna torre?
NO.
Construye una.
OK.
Llama al bloque más grande "superbloque"
OK.
¿Has tomado algún superbloque desde que comenzamos?
SI.
¿Porqué lo dejaste caer?
POR "LO" ASUMO QUE TE REFIERES AL SUPERBLOQUE.
NO PUEDO EXPLICAR UN EVENTO INEXISTENTE.
¿Hay algo a la derecha de la pirámide roja?
SI, CUATRO COSAS: LA CAJA, EL CUBO ROJO, LA PIRAMIDE AZUL Y EL BLOQUE LLAMADO SUPERBLOQUE.
Gracias
DE NADA!

Se presenta a continuación una descripción somera de los componentes del sistema.

Representación del Conocimiento

Para representar el conocimiento del mundo, que describe tanto al estado de los objetos en cada momento como a los procedimientos que actuan sobre ellos, se usa un banco de datos que contiene los hechos verdaderos en un momento dado:

```
( IS B1 BLOCK)
     B1 define a un objeto 'bloque'
( SUPPORT B1 B2 )
     el objeto B1 soporta al objeto B2
( COLOR-OF B1 RED )
     el color del objeto B1 es rojo
( CAUSE EVENT27 EVENT29 )
      el evento 27 es causa del evento 29
( AT B1 ( LOCATION 100 100 0) )
     el objeto B1 esta situado en la posición (100,100,0)
```

Al listar las distintas relaciones que guardan entre si los conceptos del sistema el significado de cada uno queda definido por sus relaciones con los otros conceptos.

La representación que conecta los distintos conceptos se describe en forma de procedimientos escritos en el lenguaje PLANNER, desarrollado por Carl Hewitt /24/. Por ejemplo, el concepto de 'tope despejado' puede describirse como un procedimiento que involucra a la vez a otros procedimientos asociados a los conceptos 'soportar' y 'librarse de', definidos a su vez en otros del estilo de 'agarra' y 'levanta'.

Pero las descripciones en forma de procedimientos no se emplean únicamente para llevar a cabo órdenes. También son útiles para responder a preguntas, como en "¿Porqué pusiste un bloque rojo sobre la mesa?", que se contestan examinando las funciones asociadas a cada procedimieto. Preguntas del tipo de "¿Cómo ...?" se contestan averiguando cuáles fueron los procedimientos que finalmente se llevaron a cabo para alcanzar la meta pedida.

Así, cada oración interpretada por el lenguaje es vista como un conjunto de procedimientos -escritos en PLANNER- a ser ejecutados. De este modo, PLANNER fija el lenguaje al que las oraciones en lenguaje natural deben traducirse. PLANNER cuenta con características tales que permiten que la administración de los procesos (v. gr.: iteraciones, retroceso en caso de callejones sin salida) se realice automáticamente.

Por ejemplo, la frase "un cubo rojo que soporta a una pirámide" equivale a un programa en PLANNER similar a

/25/:

```
( META ( ES ?X1 BLOQUE ) )
( META ( COLOR-DE ?X1 ROJO ) )
( META ( EQUIDIMENSIONAL ?X1 ) )
( META ES ?X2 PIRAMIDE ) )
( META ( SOPORTA ?X1 ?X2 ) )
```

que indica lo siguiente:
hay que localizar un objeto que sea un bloque; el proceso termina si éste es rojo; es equidimensional; y es posible hallar otro objeto tal que sea una pirámide y esté soportado por aquel.

Análisis Semántico

Para traducir una frase del lenguaje natural a la representación interna es necesario considerar el modo de interpretar las palabras, tanto por su significado individual como por el contexto en el que ocurren.

Para definir el significado de cada palabra, se usa una representación similar a la siguiente /26/:

```
( CUBO
     ( ( NOMBRE ( OBJETO
                    ( ( MANIPULABLE RECTANGULAR )
                                ( ES ? BLOQUE)
                                ( EQUIDIMENSIONAL ? )
                            ) ) ) ) ) )
```

y también:

```
( CONTENER
     ( ( VERBO (( TRANSITIVO (RELACION
                    (((CONTENEDOR))((OBJETO-FISICO
                                                ( CONTENER #1 #2
))

(((CONSTRUCCION))((OBJETO-FISICO))
                                                    ( PARTE-DE #2
#1)

)))))))
```

La primera definición indica que un CUBO es un objeto que es tanto RECTANGULAR como MANIPULABLE; y que puede ser reconocido porque posee los atributos de ser un BLOQUE y de ser EQUIDIMENSIONAL. Al hacer una clasificación semántica aproximada, es factible que el sistema haga pruebas sencillas acerca de los posibles significados de una frase. Preguntas del tipo de " Puede una mesa levantar un bloque?" se responden fácilmente si se sabe que el verbo LEVANTAR exige

un sujeto ANIMADO, mientras que el objeto MESA esta marcado como INANIMADO.

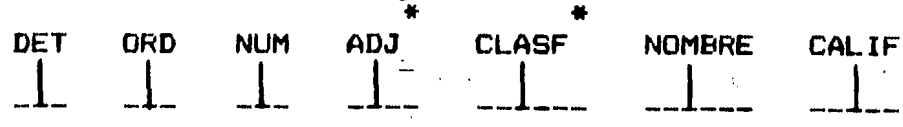
El segundo ejemplo muestra el uso de palabras con más de una interpretación posible. Si CONTENER se aplica a una frase que involucra a un CONTENEDOR y a un OBJETO-FISICO, como en "la caja contiene tres pirámides", la relación apropiada es precisamente la de CONTENER. Por otro lado, si se aplica para relacionar a una CONSTRUCCION con un OBJETO-FISICO, como en la oración "la pila contiene un cubo", lo que se quiere decir es que el CUBO es PARTE-DE la PILA.

Es importante notar que las definiciones anteriores contienen en ellas programas que son invocados en ejecución tanto para obtener información como para construir estructuras semánticas. Es con este tipo de mecanismos que se pretende crear definiciones tan dinámicas como sean necesarias.

El Componente Sintáctico

La meta del análisis sintáctico es el producir un resultado útil para el analizador semántico. De este modo, la gramática está organizada en unidades sintácticas relevantes para extraer significados y los programas semánticos están organizados en grupos de procedimientos, de modo tal que cada uno es usado para interpretar un cierto tipo de unidad sintáctica.

Para cada unidad existe un programa escrito en el el lenguaje PROGRAMMAR, cuya finalidad es la de interpretar unidades sintácticas sobre la cadena de entrada. Para ello, utiliza conocimientos sintácticos de cada unidad a procesar del tipo de /27/:



que define la estructura de un grupo nominal.

La gramática que proporciona esta información está basada en el trabajo de M. A. K. Halliday, conocido como 'gramáticas sistémicas' /28/.

PROGRAMMAR es un subsistema escrito en LISP para SHRDLU; su función es la de generar estructuras sintácticas y sistémicas correspondientes.

La Organización del Programa

El elemento más novedoso de SHRDLU es el modo en que sus componentes se relacionan, asumiendo implícitamente que no es posible atacar la comprensión de lenguaje natural separando el análisis sintáctico, semántico y pragmático, sino que la clave de la comprensión radica precisamente en la interacción de las distintas partes.

El sistema usa simultáneamente todos sus elementos para reconocer, comprender, y responder una oración. Tan pronto como se ha reconocido una estructura sintáctica, se activa un programa semántico para ver si aquella hace sentido; a su vez, la respuesta de este subsistema puede redirigir el análisis sintáctico, llamando procesos deductivos y quizá provocando una serie de preguntas al usuario.

SHRDLU fijó un nuevo nivel de calidad en el desempeño de programas de comprensión de lenguaje natural, demostrando una gran habilidad para desenvolverse en un diálogo, aunque en universo sumamente restringido.

El punto débil de SHRDLU es consecuencia de su fuerza: un manejo tan estrechamente ligado de sus componentes hace inviable el diseño de sistemas bajo los mismos supuestos para los que SHRDLU fue diseñado en universos más complejos.

Como ya se menciono, SHRDLU era tan complejo que impedía cualquier pronóstico de su comportamiento a priori. Intentar, para universos más complicados el diseño de sistemas en que todos los elementos colaboran de un modo tan estrecho no parece posible.

Esto implica que los resultados obtenidos no pueden hacerse generales; cualquier otro universo -aún de la misma complejidad- puede disminuir el desempeño de un sistema desarrollado bajo las mismas bases que SHRDLU. Por ejemplo, basta con tener distintas relaciones semánticas para alterar todo el análisis.

Por otra parte, el diseño de SHRDLU impide evaluar a cada uno de sus componentes por separado; no se sabe, por citar un caso, cuáles son los méritos particulares de la gramática del sistema.

Después de SHRDLU, ha comenzado a tomar forma la idea de diseñar herramientas independientes del dominio del problema, de aplicación general. Por ello, SHRDLU aparece ahora como un dinosaurio. Pero no debe olvidarse que fijó un nivel de desempeño único en su época.

I.5 Nuevos Sistemas (1972-)

A partir de SHRDLU, el énfasis en el desarrollo de sistemas lo han recibido los aspectos relacionados con la comprensión del discurso y de modelado de diálogos. Esto ha requerido de nuevos formalismos que permitan ver el conocimiento de un modo más dinámico, donde cada pieza nueva de información no se agregue simplemente a lo anterior, sino que al incorporarse afecte dinámicamente a lo que ya se sabe, posiblemente a través de procesos de reorganización complejos.

Es por ello que el problema de cómo representar de una manera eficiente al conocimiento es una preocupación central de la inteligencia artificial, y por ende, de la comprensión de lenguaje natural.

Entre los principales métodos de representación, se cuenta con los marcos de Minsky, los libretos, y las redes semánticas. Además se utiliza la representación de procedimientos, de la que SHRDLU es un ejemplo, y el cálculo de predicados, como en PROLOG.

I.5.1 Marcos de Minsky

Marvin Minsky, uno de los pensadores más influyentes en el campo, propuso en 1975 una metodología de representación conocida como teoría de marcos. En las palabras de Minsky /29/:

"Cuando una nueva situación es encontrada (o el modo de examinar el problema presente sufre un cambio substancial), es seleccionada una estructura substancial de la memoria conocida como un *marco*... Un *marco* es como es una estructura de datos usada para representar una situación estereotipada; tal como estar en una sala de un cierto tipo, o el ir a una fiesta de cumpleaños infantil."

Más precisamente, un marco puede verse como una serie de nodos asociados jerárquicamente; en los nodos superiores se encuentra la información más estable del marco, y conforme se desciende en la jerarquía se encuentran los nodos responsables de representar los detalles específicos del fenómeno.

Además de la información que describe la situación, un marco debe de contener datos que indiquen qué cosas esperar y qué hacer si una situación no es como se

esperaba originalmente. Así, la elección del siguiente marco dependerá tanto de la información recolectada hasta el momento como del estado general del proceso (por ejemplo: si el último marco invocado falló, la elección del siguiente a probar depende de las causas que ocasionaron la interrupción del marco anterior).

Aunque Minsky propone la metodología de los marcos para llevar a cabo procesos de reconocimiento visual, se pretende que la teoría es lo suficientemente general para ser aplicada a una variedad de problemas, incluyendo al del análisis del discurso /30/:

"Tanto las palabras claves como las ideas de un discurso evocan estructuras de escenarios o temáticas sustanciales, traidas de la memoria junto con ricas suposiciones a priori."

Particularizando otro poco, Minsky sugiere que los sistemas de proceso de lenguaje natural pueden agruparse alrededor de una jerarquía de marcos que cubran los distintos niveles de comprensión, desde marcos para análisis sintáctico superficial hasta marcos para cubrir los elementos más generales, como el argumento, la trama, los protagonistas, etc.

A pesar de ser una propuesta de carácter muy general, la metodología de marcos cumplió con el propósito de estimular la discusión y marcar la linea principal de investigación en los siguientes años.

Un ejemplo destacado de un sistema diseñado alrededor de la teoría de marcos es GUS (Genial Language Undestander) /31/. GUS es un sistema experimental desarrollado con la meta de ofrecer un sistema de reservaciones de una linea aérea por medio de un diálogo con el cliente dirigido por un sistema de marcos. GUS debe guiar la conversación para extraer la información suficiente para apartar un boleto para un viaje redondo partiendo de San José.

Se presenta el siguiente diálogo tipo (la parte de GUS va en mayúsculas) /32/:

HOLA. PUEDO AYUDARTE A PLANEAR UN VIAJE AEREO SENCILLO.
¿A DONDE QUIERES IR?
Quiero ir a San Diego el 28 de mayo.
¿A QUE HORA QUIERES SALIR?
Debo estar en San Diego antes de las 10 am.
¿TE GUSTARIA EL VUELO 102 DE P.S.A. QUE LLEGA A LAS 9:15 AM?
¿Cuál es el siguiente vuelo?
VUELO 310 DE AIR CALIFORNIA QUE SALE A LAS 8:30 AM Y LLEGA A

LAS 10 AM.
Tomaré el primero.
¿A QUE NOMBRE HAGO LA RESERVACION, POR FAVOR?
Dan Bobrow.
HE CONFIRMADO EL SIGUIENTE VUELO: VUELO 102 DE P.S.A. DEL MIERCOLES 28 DE MAYO DE SAN JOSE A SAN DIEGO QUE SALE A LAS 7:30 AM Y LLEGA A LAS 9:15 AM.
¿EN QUE FECHA QUIERES REGRESAR?
El viernes en la tarde.
¿TE GUSTARIA EL VUELO QUE SALE A LAS 7:45 PM?
Muy bien.
HE CONFIRMADO EL SIGUIENTE VUELO: EL VUELO 307 DE P.S.A. DEL VIERNES 30 DE MAYO DE SAN DIEGO A SAN JOSE QUE SALE A LAS 7:45 PM Y LLEGA A LAS 9:30 PM. GRACIAS POR LLAMAR. ADIOS.

Los marcos que definen a GUS deben tener la información suficiente para dirigir la conversación; detectar los datos relevantes cuando éstos aparecen; y mitir las respuestas adecuadas para solicitar y confirmar la información.

Como ejemplo de los marcos empleados, están DIALOGO y ESPECIF-VIAJE:

Para DIALOGO,

Compartimiento	Agentes	Sirvientes	Demonios
Cliente	Persona	Crear	Liga a Viajante
Ahora	Fecha	Traer-fecha	
Topico	Especif-viaje	Crear	

y para ESPECIF-VIAJE,

Ruta-destino	Ruta	Crear		
Estar-fuera	Lugar-fuera			
Ruta-origen	Ruta	Crear		
Puerto-origen	Ciudad		Default:	Liga a Ruta-Destino
Estar-fuera,Ruta-origen				Palo Alto

El sistema es activado al intentar resolver el marco DIALOGO. Este debe llenar los compartimientos del marco -en cualquier orden-, utilizando para ello los agentes asociados a cada compartimiento. Los sirvientes tienen como función la de llenar los compartimientos, ya sea calculando

un valor a partir de los datos ya conocidos, ya sea creando un nuevo marco o haciendo una pregunta. La noción de *demonios*, original de Eugene Charniak /33/, consiste en tener 'programas al acecho', que esperan a que se cumpla una condición para activarse. Por ejemplo, tan pronto como la salida del vuelo es especificada, un demonio se encarga de desplegar el vuelo solicitado.

Dentro del estrecho dominio de definición de GUS es posible que éste logre un desempeño muy bueno, sacando partida de un conjunto de datos pequeño y bien definido. No parece factible que la estrategia de GUS pueda trasplantarse a ambientes con grandes volúmenes de informacín, donde no es posible definir diálogos con tanta precisión.

Una de las metas de GUS era la de servir como sujeto de experimentación del lenguaje KRL -Knowledge Representation Language-, de Daniel Bobrow y Terry Winograd /34/, definido para proveer un lenguaje universal para ayudar a la solución de una amplia variedad de problemas.

Trabajando dentro de la teoría de marcos, Bruce Roberts e Ira Goldstein diseñaron en 1977 un lenguaje para construir en él sistemas de marcos, llamado FRL -Frame Representation Language- /35/.

I.5.2 Libretos

El formalismo de los libretos -scripts- tiene su origen en el trabajo realizado por Roger Schank y Robert Abelson, junto con un grupo de colaboradores. Dicho esfuerzo representa uno de los más coherentes y sostenidos dentro del campo de la comprensión de lenguajes naturales.

Schank desarrolló una teoría de representación de significados llamada *dependencia conceptual*, para el sistema MARGIE, creado para realizar deducciones sencillas a partir de un texto simple. En MARGIE participaron Chris Riesbeck, Chuck Rieger y Neil Goldman.

La propuesta central de dicha teoría es que es posible encontrar un nivel de representación abstracto de las oraciones en lenguaje natural /36/:

"Básicamente, la visión del lenguaje natural aquí expresada es la de que existe una base conceptual en la cual las emisiones de lenguaje natural son mapeadas durante la comprensión. Más aún, se asume que esta base conceptual está bien definida como para que una entrada inicial a la base conceptual pueda hacer posible la predicción del tipo de información conceptual que es posible siga a la entrada inicial."

La dependencia conceptual intenta proporcionar, se sigue, un formalismo para representar a la base conceptual y a los modos de hacer el mapa. Las estructuras conceptuales están diseñadas para representar el significado de una oración de modo no ambiguo, y reciben el nombre de *conceptualizaciones*. Una conceptualización consiste básicamente en conceptos (nominales, de acción, modificadores) relacionados de distintas maneras.

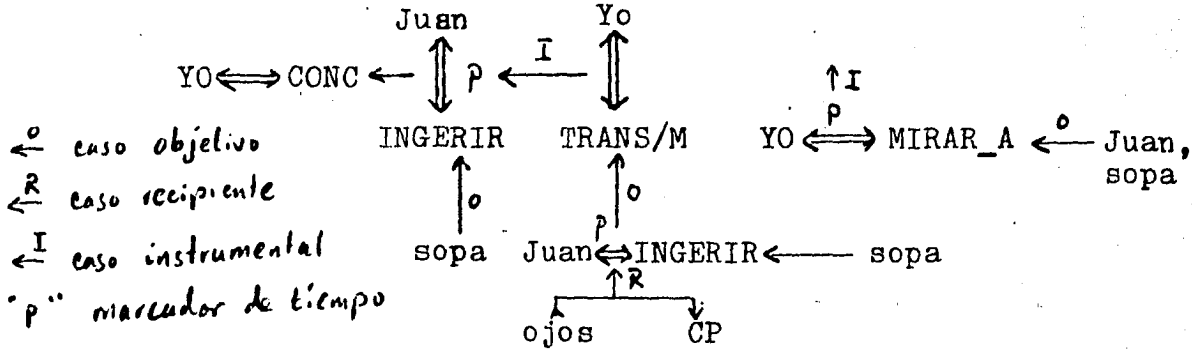
Aunque el conjunto de acciones ha variado a lo largo de los años, hay acciones de los siguientes tipos:
actos físicos (impulsar, mover)
actos globales (alterar relaciones físicas -trans/f- y abstractas -trans/a-)
actos instrumentales (atender, hablar)
actos mentales transferencia de información -trans/m-, acciones mentales -cons/m-).

El lenguaje completo de la dependencia conceptual incluye, además, una clasificación de casos (objetivo, recipiente, directivo, instrumental), categorías de acciones (física, mental, comunicativa,), relaciones y tiempos conceptuales, reglas conceptuales y diccionarios /37/.

Para dar una idea de lo anterior, la conceptualización subyacente a la oración:

"yo ví a Juan comiendo sopa"

sería /38/:



donde: TRANS/M es una transferencia mental, usada para representar transferencia de información (en este caso, el objeto transferido es la imagen de "Juan come sopa")

CP representa la parte del sujeto donde son llevados a cabo los procesos de pensamiento: es el procesador consciente -Conscious Processor-

CONC representa a las conceptualizaciones, esto es, a los objetos considerados por el CP

Por su parte, Robert Abelson propuso un modo de representar la historia consecutiva de las relaciones entre varios actores: los libretos -scripts-, dentro del marco del modelado de 'sistemas de creencias' /39/.

Con estos antecedentes, Schank y Abelson introducen en 1977 una particularización de los marcos de Minsky: los libretos. Esta generalización es vista como necesaria para poder atacar las complicaciones introducidas por un texto: relaciones causales y temporales, la conexión implícita entre las partes del discurso, etc.

De acuerdo a ellos /40/:

"Un libreto es una estructura que describe secuencias apropiadas de eventos en un contexto particular. Un libreto está conformado por compartimientos y requisitos acerca de qué puede entrar en los compartimientos. La estructura es una unidad interconectada, y lo que hay en un compartimiento afecta lo que puede haber en otro. Los libretos manejan eventos diarios. No están sujetos a mucho cambio, ni proveen el aparato para manejar situaciones completamente nuevas. Así, un libreto está predeterminado, una secuencia estereotipada de acciones que define una situación bien conocida."

Como un ejemplo breve, se presenta el boceto de un libreto de un restaurant /41/:

libreto: RESTAURANT
papeles: cliente, mesera, chef, cajero
razón: obtener comida para acabar con el hambre y obtener placer

escena 1: ENTRANDO
trans/f: uno mismo en el restaurant
atender: ojos a donde haya lugares vacíos
cons/m: donde sentarse
trans/f: uno mismo en la mesa
mover: sentarse

escena 2: ORDENANDO
trans/a: recibir el menú
atender: mirar el menú

cons/m: decidir lo que uno desea
trans/m: darle la orden al mesero

escena 3: COMIENDO
trans/a: recibir comida
ingerir: comer comida

escena 4: SALIENDO
trans/m: pedir la cuenta
trans/a: recibir la cuenta
trans/a: darle propina al mesero
trans/f: ir a la caja
trans/a: dar dinero al cajero
trans/f: salir del restaurant

Dentro del trabajo realizado por el grupo de Schank se encuentran MARGIE, ya mencionada, SAM -Script Applier Mechanism-, FRUMP -Fast Reading Understanding and Memory Program-, y PAM -Plan Applier Mechanism.

SAM está diseñado para leer una historia sencilla y producir a cambio una red de dependencias conceptuales, en base a la cual pueden elaborarse paráfrasis y resúmenes. Wendy Lehnert le agregó posteriormente un programa que responde preguntas acerca de la historia leyendo la representación interna de aquélla.

FRUMP, por su parte, es un sistema que realiza una lectura superficial de una historia, produciendo a cambio un resumen.

PAM comprende historias que contienen metas, planes, temas, etc. Por ejemplo, dada /42/:

"Juan amaba a María pero ella no quería casarse con él. Un día, un dragón robó a María del castillo. Juan se subió en su caballo y mató al dragón. María aceptó en casarse con él. Vivieron felices para siempre."

PAM puede contestar preguntas tales como:

¿Porqué se subió Juan a su caballo?
PORQUE EL QUERIA ESTAR CERCA DE MARIA

¿Porqué estuvo María de acuerdo en casarse con Juan?
PORQUE ELLA ESTABA EN DEUDA CON EL.

¿Porqué mató Juan al dragón?
PORQUE EL QUERIA QUE MARIA NO MURIERA.

I.5.3 Críticas a la Teoría de Marcos

El concepto de *marco* o *libreto* ha sido adoptado en buena medida. Por un lado, se ha llegado al convencimiento de que ningún sistema con una gama limitada de objetos y operaciones será capaz de hallar un balance adecuado entre todos los aspectos de representación involucrados /43/. Por otra parte, la opinión convergente dentro del campo ha sido reforzada por algunas opiniones externas, como las del linguista Charles Fillmore /44/ y las del psicólogo Erving Goffman /45/, quienes comparten las porpuestas de la teoría de marcos. Sin embargo, estas teorías no han sido reconocidas por todos como la respuesta.

Stephen Palmer expresa con claridad la crítica que más atención ha recibido, la de como seleccionar el marco inicial /46/:

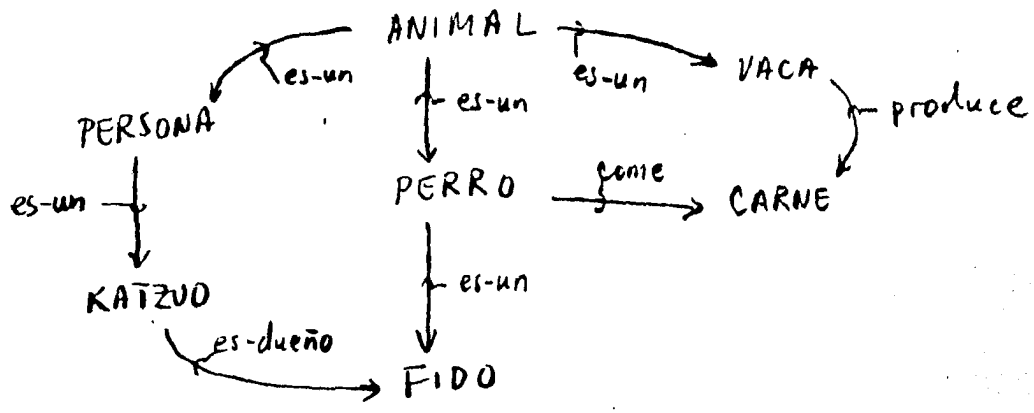
" Cómo puede alguien reconocer una cara hasta que ha reconocido los ojos, la nariz, la boca, los oidos? Y viceverza, cómo puede alguien reconocer los ojos, la nariz, la boca y los oidos hasta saber que son parte de una cara?"

Yorick Wilks /47/ plantea una crítica de carácter más general. Wilks argumenta que la comprensión del lenguaje requiere tanto de conocimientos linguísticos como conocimientos generales del mundo y capacidad deductiva. Así, sería infrecuente el tener que depender de grandes, estáticos, trozos de conocimiento.

Finalmente, otras críticas apuntan que los marcos son una representación del conocimiento relativamente estática para trabajar en un mundo dinámico. En ejemplos como los de GUS, se deja ver que la gamà de aplicaciones de sistemas basados en marcos es limitada.

I.5.4 Redes Semánticas

Una red semántica es una gráfica dirigida, donde los nodos representan conceptos, y los arcos relaciones entre ellos. Como ejemplo de una red sencilla, tómese:



El concepto de red semántica se acredita generalmente a Ross Quillian /48/, y tiene dos características que lo han hecho popular: es intuitivamente sencillo y se adapta especialmente bien a la recuperación de información. Por ejemplo, si se busca la relación entre FIDO y CARNE, basta recuperar:
FIDO es-un PERRO
PERRO come CARNE

También se argumenta que es necesario contar con un sistema sencillo y poderoso para lograr la capacidad de almacenamiento y velocidad de recuperación exhibida por el ser humano. Scott Fahlman /49/ cita el siguiente ejemplo para favorecer el uso de redes semánticas: si digo que el fuego comenzó en el pastel de cumpleaños del niño, uno piensa inmediatamente en las velas del pastel y quizá en los muchos decorados de papel. Pero uno no se pone a pensar en e color del pastel, ni en el juego de 'ponerle la cola al burro', ni en el hecho de que los cumpleaños ocurren una vez al año. Estos conceptos están ahí, para cuando se les necesita, pero no retardan la búsqueda de la relación adecuada entre la fiesta y el fuego.

Entre los argumentos en contra de su uso, se dice que es difícil la representación de construcciones lógicas /50/, y se señala además que la búsqueda crece explosivamente para más de unos pocos nodos /51/.

Un ejemplo del uso de redes semánticas lo representa el trabajo de Donald Norman, David Rumelhart y el grupo de investigación LNR, de la Universidad de California en San Diego /52/.

I.5.5 Cálculo de Predicados de Primer Orden

Esta representación usa directamente el cálculo de predicados para representar las hechos del universo de trabajo. Su ventaja principal es que da una gran flexibilidad

para la represenación, además de contar con una gran cantidad de herramientas para deducir nuevos conocimientos.

Sin embargo, el calculo de predicados no da elementos para preferir una fórmula a otra, por lo que la búsqueda tiene que ser hecha a ciegas. Esta representación también tiene el problema de la explosión en el tiempo de ejecución apenas se pasa de unos pocos hechos.

Este tipo de enfoque ha sido elegido en Japón para implantar el lenguaje de máquina de las llamadas 'computadoras de quinta generación', via el lenguaje PROLOG -Programming in LOGic-. El lenguaje PROLOG, originado en Francia y desarrollado en Inglaterra, describe programas en los que cada instrucción consiste en un enunciado lógico, y su ejecución, en realizar deducciones /53/. Usando así a PROLOG, y proponiéndose como meta la construcción de máquinas capaces de ejecutar millones de inferencias lógicas por segundo, los japoneses esperan construir máquinas cualitativamente más inteligentes que la actuales /54/.

I.6 Aplicaciones

Se mencionan brevemente las aplicaciones dadas hasta la fecha a los sistemas de comprensión de lenguaje natural, así como su posible futuro.

El desarrollo de sistemas de pregunta-respuesta para tener acceso a bancos de datos es posiblemente la aplicación que mayor atención ha recibido. Además de los sistemas ya mencionados, se tiene a RENDEZVOUS(1978), ROBOT(1977), PLANES(1976), y LADDER(1977) /55/. Destaca ROBOT -posteriormente conocido omo INTELLECT-, un sistema diseñado de modo que sea independiente a dominio de los datos con los que trabaje. ROBOT es difundido de manera comercial por dos corporaciones de servicio de software.

Otra aplicación buscada en el pasado reciente es la de comprender a la palabra hablada. El esfuerzo más notable hasta la fecha has sido el realizado bajo los auspicios de la ARPA entre 1971 y 1976. Los sistemas resultantes, con diversos grados de éxito, fueron: SPEECHLIS, de Bolt, Beranek and Newman Inc., HEARSAY-II y HARPY, de la Universidad Carnagie-Mellon, y el sistema desarrollado conjuntamente por Systems Development International y Stanford Research Institute /56/.

El desarrollo de sistemas expertos tiene una importancia creciente dentro de la inteligencia artificial. Ya que

presuponen la existencia de un módulo que permita la comunicación usuario-experto, requieren de sistemas de comprensión de lenguaje natural para dicho propósito /57/.

I.7 Sumario

El campo de la comprensión de lenguaje natural ocupa un lugar preponderante dentro de la investigación en inteligencia natural.

Ya que el lenguaje ocupa un papel central en los procesos inteligentes, cualquier intento de hacer programas de comprensión de lenguaje natural por computadora exige la resolución de problemas fundamentales de todo el campo: la representación eficiente de conocimientos, el trabajo con grandes cantidades de datos, la deducción rápida de hechos en espacios de búsqueda enormes.

Aunque el campo es joven, es posible encontrar un acuerdo creciente de los investigadores acerca de algunos principios generales. Marvin Minsky, Roger Schank y otros investigadores del campo, junto con opiniones semejantes de fuera de la comunidad, apuntan a estrategias similares para atacar problemas; aunque este acuerdo no es unánime, ni ha surgido aún un sistema que entrege evidencias definitivas, en uno u otro sentido.

Por otra parte, comienzan a desarrollarse sistemas comerciales de comprensión de lenguaje natural. Se sabe que IBM, Phillips, Burroughs, Sperry Univac, SDC y Hewlett-Packard están desarrollando sistemas de pregunta-respuesta.

La traducción automática ha vuelto a recibir atención después de los primeros fracasos. Sistemas comerciales como SYSTRAN están siendo desarrollados, y la Comunidad Económica Europea patrocina un sistema de traducción /58/.

El creciente interés en el desarrollo de sistemas expertos tendrá también influencia en el futuro del campo. Edward Feigenbaum cita a 27 compañías particulares que de un modo u otro realizan investigación en temas de comprensión de lenguaje natural (sistemas expertos, educativos, etc.) /59/.

Notas al Capítulo I:

1. Bernstein(1981):70
2. Feigenbaum(1963):108
3. Bar-Hillel(1951)
4. Bar-Hillel(1960)
5. Pierce(1962)
6. Raphael(976)
7. El grupo de patrones pertenece al sistema SIR, Raphael(1968)
8. Weizenbaum(1966)
9. Bobrow(1968)
10. Bobrow(1968):212
11. Raphael(1968)
12. Según la descripción dada por Winograd(1983):362-363, 371, 398, 400.
13. Raphael(1976):201
14. Green(1963)
15. Bobrow(1968):151
16. Esto es, un *top-down backtracking context-free parser.*
17. Winograd(1983):(Kuno(1962):130,360
18. Woods(1973)
19. Bates(1978):216
20. Arden(1980):(Woods(1973)):423,424
21. El resumen está tomado de la descripción de Winograd(1973)
22. Adaptado de Winograd(1972):5
23. Winograd(1973):155 y ss.
24. Hewitt(1969)
25. Winograd(1973):172, adaptado a su equivalente en_español
26. Winograd(1973):174, también traducido
27. Winograd(1973):177
28. Berry(1975), Berry(1977)
29. Minsky(1975):212
30. Idem, 245
31. Bobrow(1977)a
32. Rosenberg(1980):(Bobrow(1977)a:157):47
33. Charniak(1972)
34. Bobrow(1977)
35. Roberts(1977)
36. Schank(1973):188
37. Descripción basada en Rosenberg(1980):33-34
38. Schank(1973):220
39. Abelson(1973)
40. Rosenberg(1980):(Schank(1977):41):36
41. Rosenberg(1980):(Schank(1975)b):36

42. Rosenberg(1980):(Schank(1977):217, Wilensky(1978)):55-56
43. Winograd(1977)
44. Rosenberg(1980):(Fillmore(1976)):27
45. Goffman(1974)
46. Rosenberg(1980):(Palmer(1975):295):43
47. Rosenberg(1980):(Wilks(1976)):43
48. Quillian(1968)
49. Fahlman(1979):5
50. Schubert(1976)
51. Winograd(1977)
52. Norman(1975)
53. Colmerauer(1978)
54. Feigenbaum(1983):115
55. Codd(1978), Harris(1977), Harris(1978), Waltz(2976), Waltz(1977), Waltz(1978), Hendrix(1977), respectivamente
56. Descritos en Arden(1980):540-542
57. Feigenbaum(1983):118-119
58. Nagao(1984)
59. Feigenbaum(1983):Ap C

II. Gramáticas Generativas y Transformativas

II.1 Introducción

En esta sección se describe someramente el trabajo realizado por el linguista norteamericano Noàm Chomsky y sus seguidores, conocido genéricamente como gramáticas generativas y transformativas.

Ya que el modelo chomskyano ha variado considerablemente con el paso de los años, lo que se presenta a continuación emodelo conocido como la *teoría común*, que abarca hasta el año de 1965.

El modelo de las gramáticas generativas y transformativas tiene como origen el trabajo desarrollado por Chomsky , primero en la Universidad de Pennsylvania, y después en Harvard, trabajo que culmina con la escritura en 1955 de *The Logical Structure of Linguistic Theory* (la obra no se publicará hasta 1975).

Ya en MIT, Chomsky edita en 1957 *Estructuras Sintácticas*, originalmente concebido para servir como notas de un curso introductorio de lingüística. El libro se convierte al poco tiempo en un éxito, y un grupo de linguistas jóvenes -casi todos del MIT-, bajo la dirección de Chomsky, extienden y precisan la teoría original.

El trabajo desarrollado en esos años queda expresado en 1965 en *Aspectos de la Teoria de la Sintaxis*. A partir de entonces, el modelo ha sido revisado y cuestionado constantemente, sin haber a la fecha un acuerdo mayoritario como el de 1965.[1] A pesar de ello, el enfoque de Chomsky se ha convertido en uno de los dominantes en la linguística teórica, especialmente en los Estados Unidos.

Aunque el trabajo de conjunto de Chomsky es reputado como sumamente original, es válido indicar algunas de sus influencias. La idea de un conjunto finito de reglas para describir todo el lenguaje tiene como origen primero el trabajo de Wilhelm von Humboldt, y más recientemente, las propuestas -dentro del marco estructuralista- de Zellig Harris y de Charles Hockett. El uso de la recursividad y del formalismo de expresión tiene como origen el trabajo en lógica matemática de Emil Post y Kazimierz Ajdukiewicz, a través de Yehoshua Bar-Hillel. Finalmente, Leonard Bloomfield

y Roman Jakobson hicieron propuestas emparentadas dentro de la linea de la fonología generativa /1/.

II.2 El Modelo: Tipos de Conocimiento Lingüístico

La teoría de Chomsky establece de entrada dos niveles en el estudio del lenguaje: el de lo *gramatical*, que se refiere a los principios que especifican cómo se construyen oraciones correctamente; y el de lo aceptable, que juzga qué tan comprensible, afortunada o aceptable es una oración real en el contexto en el que ésta se sitúa. Así, el primer nivel es puramente linguístico, mientras que el segundo requiere de conocimientos por fuerza más extensos de lo puramente linguísticos.

Chomsky denomina al primer nivel como el nivel de *competencia*; al segundo, el de *actuación*. Para el dominio de la competencia, se habla de *oraciones*, y para el de la actuación, de *locuciones*.

De lo anterior, una gramática se ocupa sólo de lo que concierne al nivel de la competencia, y tiene dos tareas:

1. Separar a las oraciones gramaticales de las no gramaticales.

2. Proveer una descripción para cada oración gramatical, de modo que dicha descripción baste para establecer tanto su pronunciación como su significado.

Para dar una explicación adecuada de los fenómwnos linguísticos, la gramática requiere de reglas en tres campos:

el de la *sintaxis*, que se refiere a la forma de las oraciones,

el de la *semántica*, que habla del significado de las oraciones,

el de la fonología, que se ocupa de la pronunciación de las oraciones.

De este modo, una gramática debe explicar *mínimamente*, en el terreno sintáctico:

1. Cómo elegir entre oraciones gramaticales y no gramaticales.

2. Aquel conocimiento que se requiere para determinar los componentes funcionales de la oración: sujeto, verbos, objetos, etc.

3. La asociación entre distintos términos, estén o no contiguos, como en:
"Los *niños* podrían haber estado *todos* gritando juntos".

4. Fenómenos tales como el de concordancia gramatical.

5. La relación subyacente entre oraciones como:

"La policía examinó la bala"
"Fue la bala lo que la policía examinó"
" Qué bala examinó la policía?"

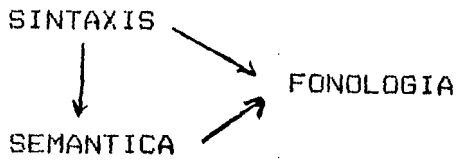
En el terreno semántico, la gramática debe dar los elementos para identificar: paráfrasis, ambigüedad, contradicciones y anomalías, tautologías y verdades analíticas, implicaciones.

En el nivel fonológico, debe de explicitarse la diferencia de pronunciación para unidades mayores a palabras, como:

1. Las distintas pronunciaciones para la misma palabra, dependiendo del contexto.
2. Las diferencias de acento de intensidad -stress- con que una palabra se pronuncia, dependiendo de su posición en la oración.
3. La variedad de entonaciónes que diferencian, por ejemplo, una afirmación de una pregunta.

Un enunciado central en la teoría de las gramáticas generativas y transformativas es el conocido como de *independencia de la sintaxis*: se afirma que es posible, y metodológicamente provechoso, llevar a cabo el estudio del componente sintáctico de la gramática sin requerir de consideraciones semánticas ni fonológicas. Dicho de otro modo, no es posible expresar en términos semánticos una definicion de gramaticalidad.

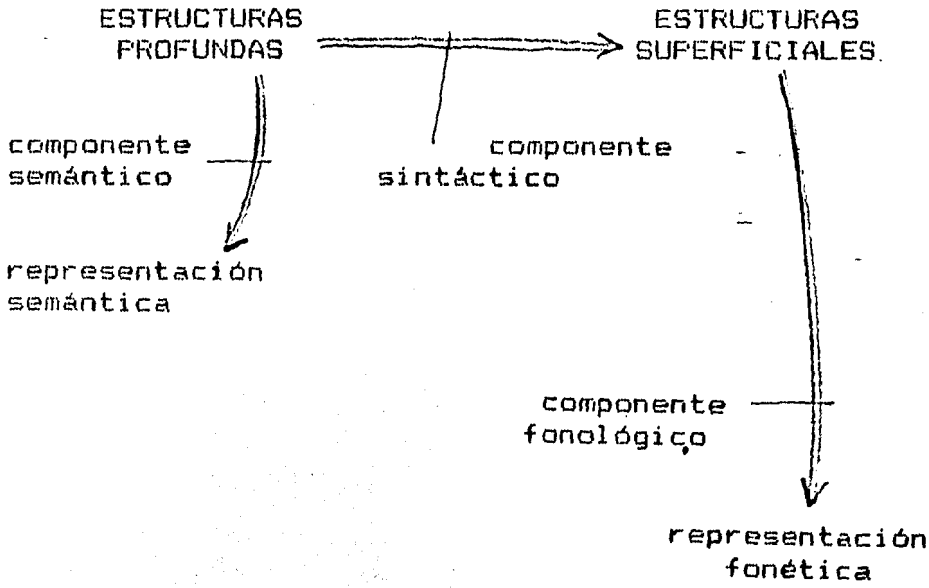
Los componentes de la gramática podrían así representarse de la siguiente manera /2/,/3/:



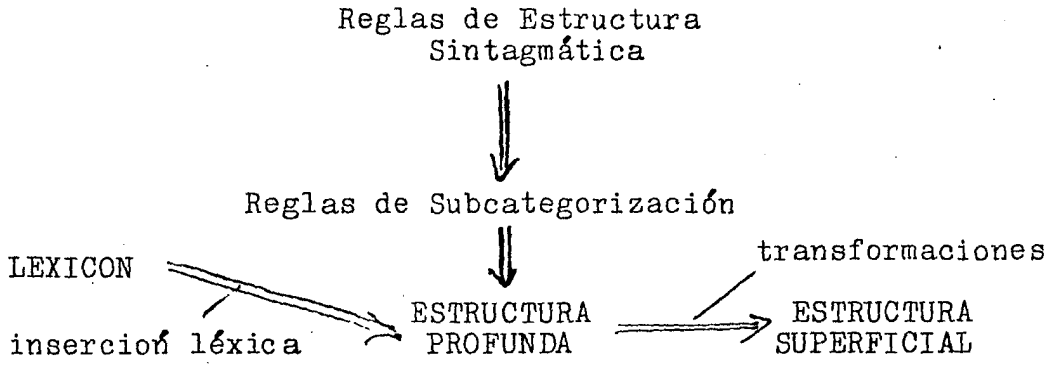
II.3 El Modelo: Formalización del Conocimiento Linguístico

Se considera a continuación el modo en que Chomsky conforma una gramática capaz de cubrir los requerimientos anteriores.

Como ya se ha sugerido, la gramática cuenta con tres componentes: el componente *sintactico*, cuyo propósito es el de producir cualquier oración perteneciente al lenguaje, entregando junto con ella dos descripciones: una para el componente *semántico* - la llamada estructura profunda-, y uno para el componente *fonológico* -la estructura superficial-. De este modo, los componentes semántico y fonológico tienen una función meramente interpretativa, ya que de descripciones apropiadas producen interpretaciones semánticas (significados) y fonéticas (sonidos).



Por su parte, el componente sintáctico está organizado de la siguiente manera:



II.3.1 Reglas de Estructura Sintagmática

Las reglas de estructura sintagmática tienen como función la de generar estructuras profundas adecuadas. Para trabajar con ellas, se requiere de:

un *símbolo distinguido*, comunmente la categoría sintáctica de mayor jerarquía: ORACION

un conjunto de elementos *terminales*, formado por las categorías léxicas del lenguaje

un conjunto de elementos *no terminales*, formado por las categorías sintácticas del lenguaje.

Cada regla es de la forma siguiente -llamada libre de contexto:

A --> B-1 B-2 ...B-i,donde:

A es un elemento no terminal,
cada B-j es un elemento terminal o no terminal.

La regla debe leerse como: sustitúyase el elemento no terminal *A* como la secuencia de terminales y no terminales *B-1 B-2 ... B-i*.

La aplicación sucesiva de estas reglas permiten, partiendo del símbolo distinguido, elaborar una serie de reescrituras que lleven a una cadena formada solamente por elementos no terminales, que se sustituyen directamente por palabras del diccionario correspondientes a cada categoría léxica.

Por ejemplo, el siguiente conjunto de reglas podría aplicarse a la generación de estructuras profundas de oraciones simples del español:

1. O --> SN SV
2. SN --> (det) N (adj)
3. SV --> verbo SN
4. SV --> verbo PNOM
5. PNOM --> adj
6. PNOM --> SN

donde :O es oración,
SN sintagma -grupo- nominal,
SV sintagma -grupo- verbal,
N sustantivo,
adj adjetivo,
det determinante (de nombre),
PNOM predicado nominal

Los símbolos en mayúsculas representan no terminales; los símbolos en minúsculas elementos terminales; los elementos entre paréntesis son opcionales.

Una oración como

"el gato azul limpió la escudilla"

se deriva de la siguiente manera:

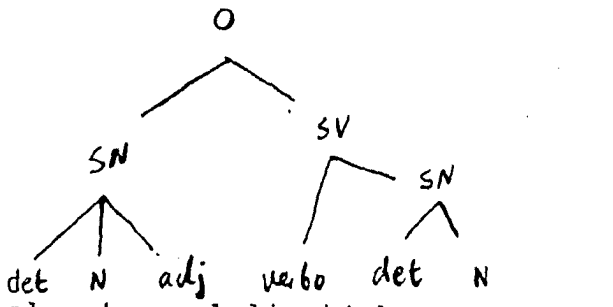
componente:	se genera con la regla:
O	1
SN SV	2
det N adj SV	3
det N adj V det N	2
...	

También:

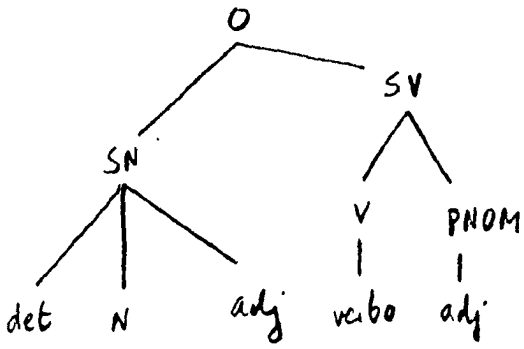
"el conejo naranja es flojo"

O	---(base)
SN SV	1
det N adj SV	2
det N adj V PNOM	4
det N adj V adj	5
...	

Estas derivaciones se pueden representar directamente a través de un diagrama de árbol (o árbol de derivación):



el gato azul limpió la escudilla.



el conejo naranja es flojo

Hay todavía algunos aspectos que considerar. En primer lugar, es importante que junto con la estructura se dejen rastros del tipo de oración que se tiene en mente generar; si la oración es afirmativa o negativa, si está en presente o pasado, etc. Estos rasgos, desde luego, desaparecen en la estructura superficial, pero servirá para indicar que tipo de oración final se tiene en mente. Una oración pasiva del estilo de:

"Gibraltar fue ocupado por los ingleses"

tiene una estructura subyacente del estilo de:

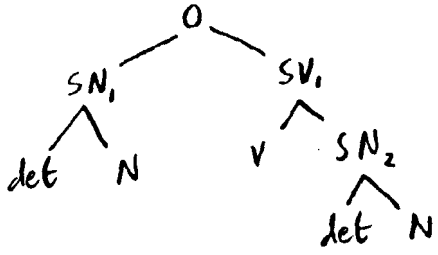
det N V N pasiva,

equivalente a:

"los ingleses ocuparon Gibraltar" + *marcador de voz pasiva*

Reglas de Subcategorización

Dada una estructura producida por las reglas de estructura sintagmática (llamada por Chomsky 'cadena preterminal'), tal como:



la gramática debe dar las reglas que especifiquen qué palabras del lenguaje se pueden sustituir en la estructura a la mano. Por ejemplo,

SN-1: "el profesor"
SN-2: "el problema"

son elecciones válidas para

V: "discutió" (i.e., "el profesor discutió el problema")

pero no son válidas para

V: "comió" (i.e., "el profesor comió el problema")

La solución adoptada consiste en asociar a cada categoría de la cadena preterminal una *matriz de rasgos* que especifique sus características. Las reglas que agregan dichas matrices a cada categoría se conocen como *reglas de subcategorización*, y son de dos tipos, estrictas y de selección.

Las reglas de subcategorización *estricta* son reglas de reescritura de la forma:

A --> C, donde A= O, SN, SV, ...
CS= símbolo complejo

El efecto que tiene la aplicación de esta regla es el iniciar *la matriz de rasgos* asociada al símbolo no terminal con los rasgos:

$\begin{bmatrix} +A \\ +a\_\_+b \end{bmatrix}$, donde la regla que introduce A es de la forma:

S --> aAb.

Poder definir los rasgos asociados a cada no terminal permite un grado de especificación muy alto del entorno posible para cad palabra del diccionario. Por ejemplo, si

SN --> (det) N (O),

[+det__] define la categoría de los nombres comunes (aquellos que tienen un determinante asociado)

[__] define la de los nombres propios (aquellos que no llevan asociado un determinante)

[+det__O] la de los nombres de una oración como complemento.

El segundo tipo de reglas de subcategorización es el de las reglas de *selección*, que tienen la siguiente forma:

[+A] --> [+B],
[+A] --> [-B], donde A es de la forma +a__+b, y B es un rasgo tal como animado, numerable, etc.

Para los sustantivos, estas reglas sirven para agregar sucesivamente rasgos a la matriz asociada a un elemento no terminal. Por ejemplo, si la matriz de rasgos asociada a un nodo N es +det__ (que equivale a +común), la gramática puede contar con reglas:

[+det__] --> [+contable]
[+det__] --> [-contable] (un nombre común puede o no ser contable)

[+contable] --> [+,- animado]
[-contable] --> [+,- abstracto]

Con las selecciones adecuadas, se puede llegar a aumentar la matriz de N hasta que ésta sea

[+N, +det__, -contable, +abstracto]

que define a una cierta clase de nombres (v. gr.: sinceridad, belleza), y excluye a otras clases (v. gr: azúcar, -contable

pero -abstracto, o árbol, +cóntable y -abstracto).

Para los verbos, la forma de la regla es:

[+V] --> CS/a__b

esto es, los rasgos pertinentes de las matrices de a y b se agregan a la matriz de V. El propósito de este tipo de reglas es el de marcar la matriz de V con los rasgos del contexto en el que V se encuentra. La idea es que una especificación suficientemente precisa del contexto que requiere cada verbo (por ejemplo, un verbo transitivo requiere de [+__SN]) evita la seleeción de palabras que conforman oraciones inválidas.

Dado que los verbos se subcategorizan en base a los rasgos de los nombres, estos deben de subcategorizarse primero.

Reglas de Inserción Léxica

Falta por aclarar cómo se eligen las palabras que se incrustan en la cadena preterminal. Para ello se requiere de un diccionario, que consiste de entradas de la forma (D,C), para:

D: una palabra, junto con su información fonológica asociada

C: colección de rasgos sintácticos asociados

Así, basta con que la matriz de rasgos de un elemento en la cadena preterminal corresponda a los rasgos de una palabra en el diccionario, para que ésta se pueda sustituir.

Una ventaja importante de este planteamiento es que el diccionario refleja todas las excepciones e idiosincracias de un idioma, de modo tal que las reglas pueden ser expuestas de una manera general, sin tomar consideracion alguna de las excepciones.

Las reglas de estructura sintagmática, junto con las reglas de subcategorización y el diccionario, forman la *base* del componente sintáctico.

Reglas Transformativas

Las reglas transformativas tienen como función tomar una o más estructuras profundas y entregar una estructura superficial. Entre las operaciones que las transformaciones realizan se cuentan: mover subestructuras de lugar, borrar elementos de la estructura profunda, integrar varias estructuras profundas en una estructura superficial.

Un ejemplo simple de una regla transformativa sería la regla de VOZ PASIVA:

pasa de: SN-1 X V SN-2 pasiva
a: SN-2 X ser -d- V por SN-1
condicion: ninguna

La regla indica que, si se cuenta con una estructura profunda que caza con la estructura descrita, y se cumplen las condiciones (en este caso ninguna), es posible aplicar la regla y obtener una estructura como la descrita. Por ejemplo, si se tiene:

Adan pasado comer l- manzana pasiva

es posible aplicar la transformación dad, con:

SN-1: Adán,
X: pasado (marcador de tiempo)
V: comer
SN-2: l- manzana

y obtener:
l- manzana pasado ser -d- comer por Adán

que despues de otras transformaciones obligatorias (que incluyen aquellas que convierten a "l- manzana" en "la manzana", a "-d- comer" en "comid-"), y de pasar por el componente fonológico, daría como resultado a una oración equivalente a:

"La manzana fue comida por Adán"

Como se verá después, hay transformaciones obligatorias y optativas. Son obligatorias aquellas que se encargan de la concordancia y acomodo; las optativas juegan papeles como la tematización.

Además del papel productivo de las transformaciones, se les atribuye, a través de diversas

condiciones, el papel de filtro: las transformaciones producen estructuras superficiales sólo si las estructuras que las alimentan estan bien formadas.

Una de las razones que contribuyeron decisivamente a la aceptación de las gramáticas generativas y transformativas fue la introducción de estas reglas, que facilitan el análisis de fenómenos que se suponían sumamente complicados (por ejemplo, el sistema de verbos auxiliares del inglés). De hecho, una meta central de *Estructuras Sintácticas* era el demostrar que ni las gramáticas de estados finitos, ni las gramáticas libres de contexto cuentan con los recursos suficientes para cumplir con los requerimientos de una gramática de un lenguaje natural.

Un ejemplo de que las gramáticas de estados finitos (conocida como gramáticas regulares o de tipo 3) son incapaces de generar oraciones de un lenguaje natural son las oraciones de la forma X^n, Y^n, como en:

"el papel tapiz
que el hombre
que tu amigo sugirió
puso
está roto"

Este tipo de oraciones se encuentran tambien al usar expresiones de la forma "si ..... entonces .....", o " o bien .... o bien .....". Se dice que cuando una gramática no es capaz de generar un lenguaje, falla en su *capacidad generativa débil*, como en este caso.

En lo que se refiere a las gramáticas libres de contexto (o de tipo 2), Chomsky argumentó en términos de su *capacidad generativa fuerte*: aunque posiblemente estas si fueran capaces de generar lenguajes naturales /4/, la tarea de escribir estas gramáticas sería muy difícil, además de que generarían estructuras incorrectas.

Después de argumentar acerca de los defectos de dichos tipos de gramáticas, la propuesta de Chomsky consistía en adoptar gramáticas con un conjunto de reglas generativas y transformativas, que tienen las siguientes ventajas:

1. La adopción de reglas transformativas simplifica las reglas generativas. En español, por ejemplo, las terminaciones de los verbos auxiliares son diversas:

   está terminando
   puede estar terminando
   ha podido terminar

ha estado terminando
puede ser terminado

Una regla generativa del tipo

aux --> (M -r) (haber -do) (M -r)(estar -ndo),
con M: verbo auxiliar
-r: afijo marcador de infinitivo
-do: afijo marcador de participio pasado
-ndo: afijo marcador de gerundio,

que se encarga de generar todos los elementos que en castellano pueden figurar como verbos auxiliares; junto con otra regla transformativa:

afijo V --> V afijo,

que intercambia en orden a un afijo con el verbo que le sigue, bastan para generar una oración como:

"Carlos puede haber estado terminando"

de la siguiente manera:

arlos pode -r habe -do esta -ndo termina-,
Carlos pode habe -r esta -ndo termina-,
Carlos pode habe -r esta -do -ndo termina-
Carlos pode habe -r esta -do termina- -ndo

que pasa, mediante otras transformaciones, a la forma final.

2. Explicita la relación entre distintas formas de una misma oración, facilitando con esto el análisis semántico.
Una implicación clave en el modelo de *Aspectos* es que al definir una estructura profunda como equivalente a la interpretación semántica de la oración asociada, es que las sucesivas transformaciones no tienen ningún efecto sobre el significado; alternativamente, la estructura profunda contiene toda la información necesaria para la interpretación semántica. Este postulado se conoce como la *hipótesis de Katz y Postal.*

3. Define claramente la 'estructura original' de una oración, antes de que operen las transformaciones para efecto de producir estructuras superficiales.

Este planteamiento tiene como contraparte la implantación de rasgos especiales a nivel de estructura profunda, y cuya función es señalar las transformaciones obligatorias para emitir oraciones negativas, interrogativas, imperativas, etc.:

la policía examinó la bala
que la policía examine la bala
¿examinó la policía la bala?

II.4 Gramáticas Generativas y Comprensión de Lenguaje Natural

La teoría de las gramáticas generativas y transformativas parte del postulado de la independencia esencial de la capacidad linguística humana del resto de los procesos inteligentes. Esto ha traido como consecuencia una preocupación por plantear teorías lo más restringidas posibles, para explicar el mismo conjunto de hechos linguísticos. Por ejemplo, desde un principio Chomsky observó que las reglas de la gramática eran demasiado poderosas, de modo que impedían obtener cualquier conclusión significativa aplicable al lenguaje humano.

Posteriormente los estudios de Stanley Peters y Robert Ritchie demostraron que la capacidad generativa de una gramática generativa y transformativa es la de un sistema de reescritura irrestricto, esto es, una gramática de este género es tan poderosa como cualquier otro formalismo para la descripción de algoritmos /5/.

Observaciones como las anteriores marcaron una preocupación constante por establecer restricciones para reducir el poder de las gramáticas, y así poder deducir principios aplicables a la totalidad de las lenguas humanas. Dos de dichas restricciones son, por ejemplo, la regla de la *recuperación de la elisión* y las reglas de *ordenación de las transfomaciones*.

Por otra parte, la meta de la investigación en inteligencia artificial es la de identificar principios generales subyacentes a todos los procesos inteligentes, de modo que lo que se busca es teorías lo más ricas posibles, aplicables al mayor rango posible de problemas. Desde el punto de vista de los linguistas chomskyanos, los mecanismo propuestos por los investigadores en inteligencia artificial son tan generales y vagos, que difícilmente pueden servir como teorías.

La teoría común, no obstante, con su clara distinción entre sintaxis y semántica, propone un programa claro para atacar al lenguaje con interés a aplicaciones por computadora. Este enfoque fue adoptado de un modo informal por varios de los primeros sistemas de comprensión de

lenguaje natral por computadora.

Pero la renuncia de la teoría común a cualquier consideración en el campo de la actuación, lanzó a los sistemas de comprensión de lenguaje natural a tomar ideas de otros campos. Entre los planteamientos alternos, reciben atención aquellos que dan mayor énfasis a la semántica, a la capacidad de realizar inferencias y resolver problemas, de representar y manejar conocimientos del mundo, etc. El resultado neto ha sido la eliminación de la distinción entre sintaxis, semántica y pragmática, para dar paso a una interacción entre los distintos niveles involucrados, como puede verse en SHRDLU y en la teoría de marcos.

A pesar de las diferencias señaladas, el modelo de las gramática generativas y transformativas es sumamente útil para el trabajo por computadora. Una razón importante es que la teoría es lo suficientemente formal y acabada para aplicarse más o menos directamente. Otro factor es la gran cantidad de trabajo realizado dentro del marco de la teoría chomskyana, reflejo de la atención recibida desde su concepción por la comunidad linguística.

Entre los sistemas de manejo de lenguaje natural que han aplicado directamente la teoría se cuentan el Mitre Transformational Parser, el sistema de análisis sintáctivco de Petrick, y el sistema de pregunta-respuesta REQUEST-TQA, desarrollado por IBM /6/.

Otros enfoques emparentados con la teoría común también han tenido aplicaciones en sistemas de comprensión de lenguaje natural: la gramática léxico-funcional de Bresnan y Kaplan /7/, las gramáticas de estructura sintagmática aumentada -augmented phrase structure grammar-, aplicadas a varios sistemas: el Linguistic String Parser, LINGOL, LIFER, DIAMON /8/.

Notas al Capítulo II

1. Ver Newmayer(1980):cap. 1 y Chomsky(1957):introducción a la 2ª edición española; Chomsky(1965):1.1
2. A-->B significa: 'el componente B tiene acceso a la información del componente A'
3. Esta interrelación difiere de la postulada originalmente por el modelo común.
4. Posteriormente, Paul Postal demostró, en 1964, que el mohawk contiene expresiones de la forma XX, que no son generables a partir de una gramática libre de contexto.
5. Peters(1969,1971,1974)
6. Zwicky(1965), Petrick(1973), Petrick(1981), respectivamente.
7. Winograd(1983):334 y ss.
8. Sager(1981), Pratt(1975), Hendrix(1977), Paxton(1978), respectivamente.

III Redes de Transición Aumentada

III. Introducción

Uno de los formalismos para llevar a cabo análisis sintáctico de lenguaje natural desarrollado y usado extensamente en la década pasada es el conocido como *redes de transición aumentada*, o ATNs, -*augmented transition networks*.

Las gramáticas escritas en forma de ATNs tienen varias ventajas: son fáciles de especificar, tienen el poder suficiente para trabajar lenguaje natural, y se prestan para implantar procesos por computadora.

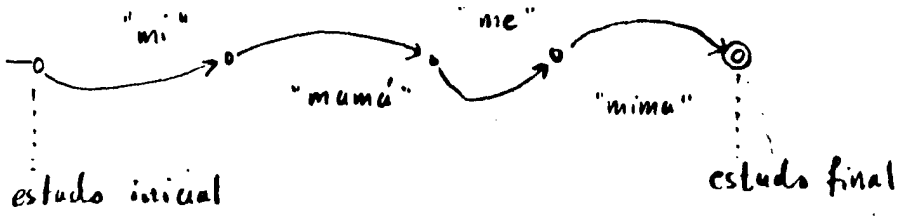
El trabajo con ATNs tiene su origen en el realizado por James Thorne y Daniel Bobrow /1/. La especificación final es de William Woods /2/. Posteriormente se hicieron propuestas alternas, como la de Ronald Kaplan /3/. La descripción que aquí se sigue es de Terry Winograd /4/.

III.2 Redes de Transición Recursiva

Un autómata finito es una gráfica dirigida, donde los nodos representan un conjunto de *estados*, y a cada arco tiene asociado una *etiqueta*. Un autómata finito tiene la capacidad de reconocer lenguajes regulares (tipo 3).

Partiendo de un estado distinguido, se dice que el autómata reconoce una expresión del lenguaje si a cada símbolo de ella es posible asociar un recorrido en la gráfica de modo tal que sea posible pasar del estado inicial de la gráfica a algún estado distinguido (llamado final), usando en el recorrido exactamente el número de símbolos componentes de la expresión.

Por ejemplo, el autómata siguiente:

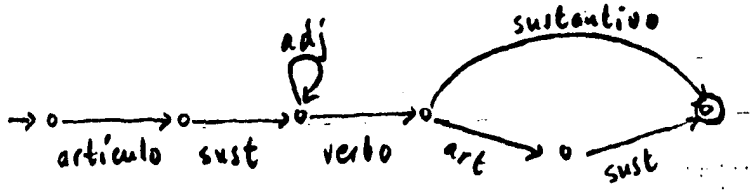


reconoce -únicamente- la oración "mi mamá me mima", mientras que

acepta exactamente aquellas expresiones formadas por cadenas de unos (posiblemente) precedidas de cadenas de ceros.

Así, las etiquetas asociadas a cada transición tienen como función la indicar bajo qué condiciones es posible pasar de un estado a otro, y el conjunto de etiquetas y su lugar en la gráfica indican ciertas estructuras válidas.

Una aproximación a un autómata que describa la estructura de una oración simple del español sería:



(usando categorías léxicas en las etiquetas). Dicho autómata acepta oraciones tales como:

"el oso grande feo come papas"
"la niña pequeña carga un oso de peluche"

Un modo de extender la capacidad de tal autómata

es reinterpretando el papel que las etiquetas cumplen en los arcos asociados. Si se permite que como etiquetas figuren categorías sintácticas compuestas (v.gr.:grupos nominales, grupos verbales), es posible agregar reglas del siguiente estilo:

si la etiqueta es
una categoría léxica,
entonces: ve si la siguiente palabra pertenece a la categoría léxica indicada.

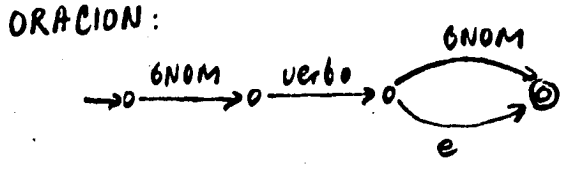
si la etiqueta es
una categoría sintáctica compuesta,
entonces: intenta aparear las siguientes palabras en la cadena de entraa con la red cuya etiqueta es la de la categoría sintáctica indicada; si ésta tiene éxito, sigue adelante.

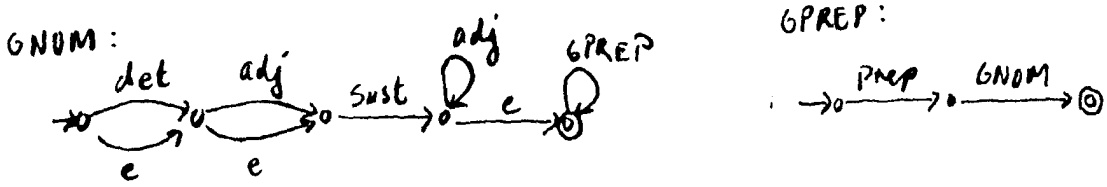
De este modo se conforman tantos autómatas -redes- como categorías sintácticas se tienen, quedando cada uno etiquetado con el nombre de su categoría asociada. Un autómata de este tipo se conoce como una *red de transición recursiva*; al conjunto de redes que juntas describen un lenguaje, junto con una etiqueta asociada (la etiqueta de ORACION), se conoce como una *gramática de redes de transicion recursiva*.

Para dar un ejemplo de lo anteriorrmente descrito, considérese la siguiente gramátia libre de contexto:

```
ORACION-->      GNOM GVERB
GNOM-->         det GNOM2    ! GNOM2
GNOM2-->        adj GNOM3    ! GNOM3
GNOM3-->        sust GNOM4   ! sust
GNOM4-->        adj GNOM4    ! adj    ! adj GPREP !  GPREP
GPREP-->        prep GNOM
GVERB-->        verbo GNOM   ! verbo
```

equivalente al conjunto de redes siguiente:





('e' indica una *transición vacía*, esto es, una transición que para lograrse no requiere de simbolo alguno de la cadena de entrada).

De manera análoga a las gramáticas libres de contexto, en la que es posible asociar a una cadena una descripción estructural -el árbol de derivación-, el recorrido de las redes genera una descripción estructural de la cadena terminal. Entonces:

procedimiento APAREA (arco_a_recorrer): nodo

si la etiqueta en arco_a_recorrer = CATEGORIA LEXICA,
entonces: nodo = crea_nodo(etiqueta) con hijo crea_nodo_terminal(palabra actual);

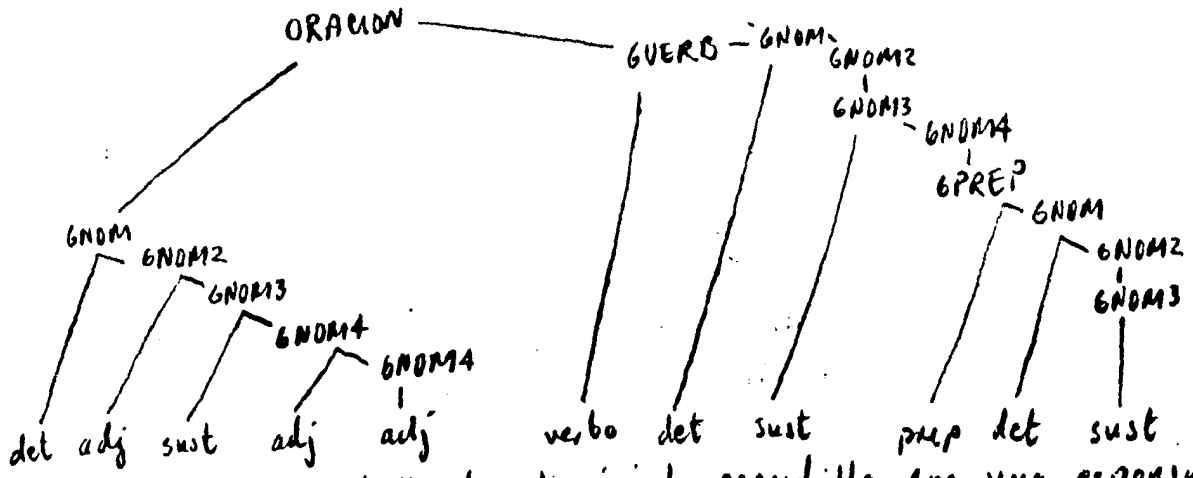
si la etiqueta en arco_a_recorrer = CATEGORIA SINTACTICA COMPUESTA,
entonces: nodo = crea_nodo(etiqueta) con hijo APAREA(ETIQUETA);

Este procedimiento, invocado para resolver la red ORACION, debe producir un análisis sintáctico hacia abajo -desde ORACION hasta las categorías léxicas terminales- de cadenas que formen oraciones en español.

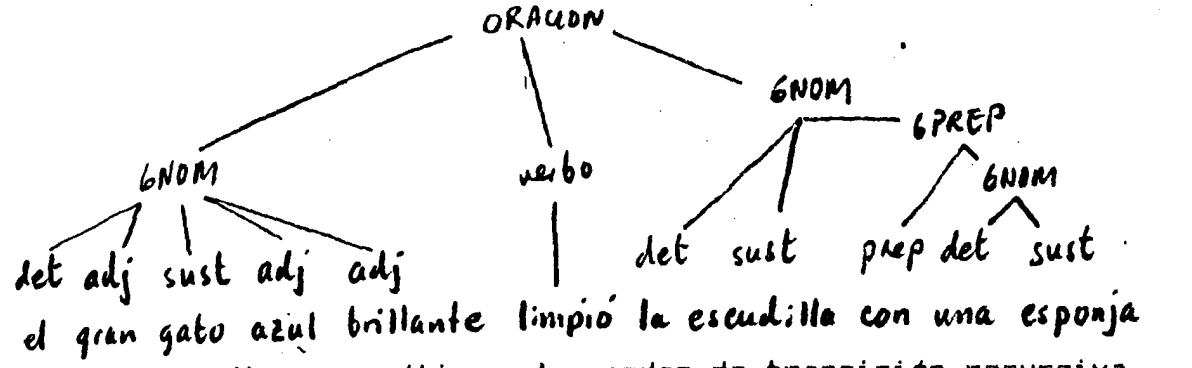
Es importante notar que aparea está descrito como un procedimiento no determinístico. Esto es: el procedimiento debe elegir en cada paso la decisión correcta sobre qué arco seguir para llegar a una decisión correcta. Esto implica que es necesario contar con un mecanismo que administre el no determinismo de la especificación de las redes.

Para ambas gramáticas, los árboles de derivación para:

"el gran gato azul brillante limpió la escudilla con una esponja"



y para la gramática de RTNs:



Una gramática de redes de transición recursiva equivale en poder al de una gramática libre de contexto; dicho de otro modo, ambos medios tienen el poder suficiente para describir los lenguajes libres de contexto (de tipo 2).

Se dice que dos gramáticas son *débilmente equivalentes* si describen los mismos lenguajes, pero asignando descripciones estructurales diferentes; si las descripciones se corresponden también, se dice que las gramáticas son completamente equivalentes.

III.3 Redes de Transición Aumentada

Del mismo modo que una interpretación alterna de las condiciones en los arcos de los RTNs permitió el aumento de su poder, la adición de un conjunto de acciones y condiciones asociadas al recorrido de un arco permite la definición de redes más poderosas que las anteriores: las llamadas *redes de transición aumentada*.

Las redes de transición aumentada permiten la generación de estructuras profundas arbitrarias, independientes de la estructura superficial -la cadena de entrada. Para ello, se cuenta con dos tipos nuevos de

variables: los registros de *características* y los registros de *papeles*.

Los registros de características permiten manejar las propiedades sintácticas asociadas a las estructuras. Entre las características suceptibles de ser manejadas se cuentan el tiempo, el género y número, etc. de las estructuras involucradas. Los registros de papeles definen a los elementos constitutivos de cada estructura.

Definición: Una red de transición aumentada es un patrón que describe una categoría sintáctica compuesta. Esta formada pro:

1. Una etiqueta que lo identifica

2. Un conjunto de estados, entre los que se distingue uno como inicial

3. Un conjunto de papeles, que define a los elemento que constituyen a la red

4. Un conjunto de características, que define las propiedades sintácticas de la categoría descrita

5. Un conjunto de arcos. Cada arco, a su vez, se compone de:

    a) Una ETIQUETA que indica el tipo de arco:
        CAT, para indicar qué se requiere una categoría léxica o una palabra específica
        BUSCA, para invocar la ejecución de otra red -recursivamente-
        MANDA, para indicar la terminación de la red presente
        BRINCA, para indicar un arco que se recorre sin afectar la cadena de entrada

    b) Un conjunto de CONDICIONES que deben cumplir los papeles y las características de cualquier red.

    c) Un conjunto de ACCIONES, que guían la construcción de la estructura que entrega el recorrido exitoso de una red. Las acciones que tienen lugar son:
        la asignación de un valor a un registro de característica
        la asignación de un valor a un registro de papel
        el agregar un nodo al final de una lista de nodos asociados a un registro de papel

d) Un estado inicial, y exceptuando los arcos de tipo MANDA, uno final.

A su vez, una *gramática de ATNs* es un conjunto de redes que juntas definen un lenguaje. En dicha gramática se distingue una red como inicial; de ordinario será la red llamada ORACION.

El esquema no determinístico en que los ATNs están originalmente descritos facilita un análisis hacia abajo y hacia la derecha. Hacia abajo, porque partiendo de la etiqueta ORACION se recorren las redes hasta llegar a los niveles más bajos, en el que se cazan las palabras de la cadena de entrada. De izquierda a derecha, porque es la dirección en la que se intentan cazar las palabras de la cadena de entrada. Ambas elecciones no son las únicas posibles: es posible analizar hacia arriba, partiendo de los elementos terminales de la oración e intentando llegar a los niveles superiores; algunos analizadores han probado recorrer la cadena de entrada por 'islas': partiendo de los nucleos bien delimitados hacia los extremos de la oración.

Otra suposición relativa al medio ambiente es la de la existencia de un diccionario que tiene, para cada palabra y para cada categoría léxica, sus características sintácticas. Por ejemplo,

la palabra "gato" pertenece a la categoría léxica *nombre*

la categoría léxica *nombre* tiene asociadas las propiedades:

género (masc., fem. o neutro)
número (sing., pl.)

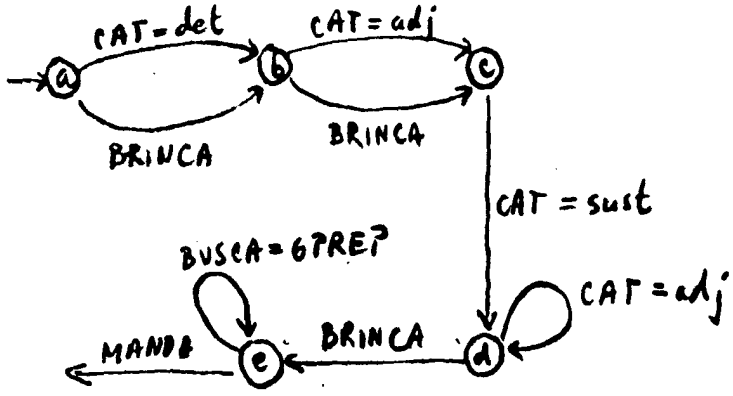
A continuación se presentan dos ejemplos. El primero ilustra el manejo de registros de *caactrísticas* en la validación de la concordancia género-número entre un nombre y sus modificadores en un GNOM. Para ello se requiere de dos de dichos registros asociadas a la red GNOM: *género* y *número*. Se asume que cada elemento terminal reconocido por la red contiene también ambas características.

```
GNOM:

Características: género, número
Papeles: ninguno

a/CAT=det/b:
COND: ninguna
ACC:  (género = *.género) y
      (número = *.número)

a/BRINCA/b:
COND: ninguna
ACC:  ninguna


b/CAT=adj/c:
COND: ( (género = vacio) o
        (*.género = género) ) y
      ( (número = vacio) o
        (*.número = número)
ACC:  (género = *.género) y
      (número = *.número)

b/BRINCA/c:
COND: ninguna
ACC:  ninguna


c/CAT=sust/d:
COND: ( (género = vacio ) o
        (*.género = género) ) y
      ( (número = vacio) o
        (*.número = número )
ACC:  (género = *.género) y
      (número = *.número)

d/CAT=adj/d:
COND: (*.género = género) y
      (*.número = número)
ACC:  ninguna

d/BRINCA/e:
COND: ninguna
ACC:  ninguna

e/BUSCA=GPREP/e:
COND: ninguna
ACC:  ninguna

e/MANDA
COND: ninguna
ACC:  ninguna
```

donde:

x/ARCO/y representa al ARCO -busca, brinca, manda o cat- con estado inicial *x* y estado final y,
COND son las condiciones asociadas al arco,
ACC son las acciones asociadas al arco
* representa al nodo presente en el recorrido del arco

La gráfica equivalente de la red es:



El segundo ejemplo muestra el modo en que las condiciones y acciones se combinan para la construcción de las estructuras deseadas, usando en ello a los registros de *papeles*. En el ejemplo, se busca que una estructuras superficiales de voz activa y voz pasiva entreguen la misma estructura profunda como resultado:

ORACION:

Características: voz
Papeles: SUJETO, OBJETO-DIR, V-AUX, V-PRINCIPAL, MODIF

a/BUSCA=GNOM/b:
COND: ninguna
ACC: SUJETO = *

b/CAT=verbo/b:
COND: (* es verbo modal)
ACC: V-AUX = VAUX + *

b/CAT=verbo/c:
COND: (* está en participio pasado) y
(último verbo en V-AUX = 'SER')
ACC: (V-PRINCIPAL = *) y
(voz = pasiva) y
(OBJETO-DIR = SUJETO) y

(SUJETO = vacío)

b/CAT=verbo/c:
COND: * no está en participio pasado
ACC: (V-PRINCIPAL = *) y
(voz = activa)

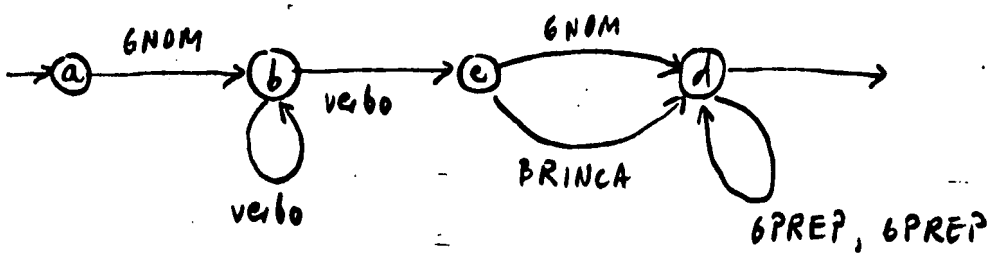
c/BUSCA=GNOM/d:
COND: voz = activa
ACC: OBJETO-DIR = *

c/BRINCA/d:
COND: ninguna
ACC: ninguna

d/GPREP/d:
COND: (voz <> pasiva) o
(*.PREP <> 'POR')
ACC: MODIF = MODIF + *

d/GPREP/d:
COND: (voz = pasiva) y
(*.PREP = 'POR')
ACC: SUJETO = *.OBJETO

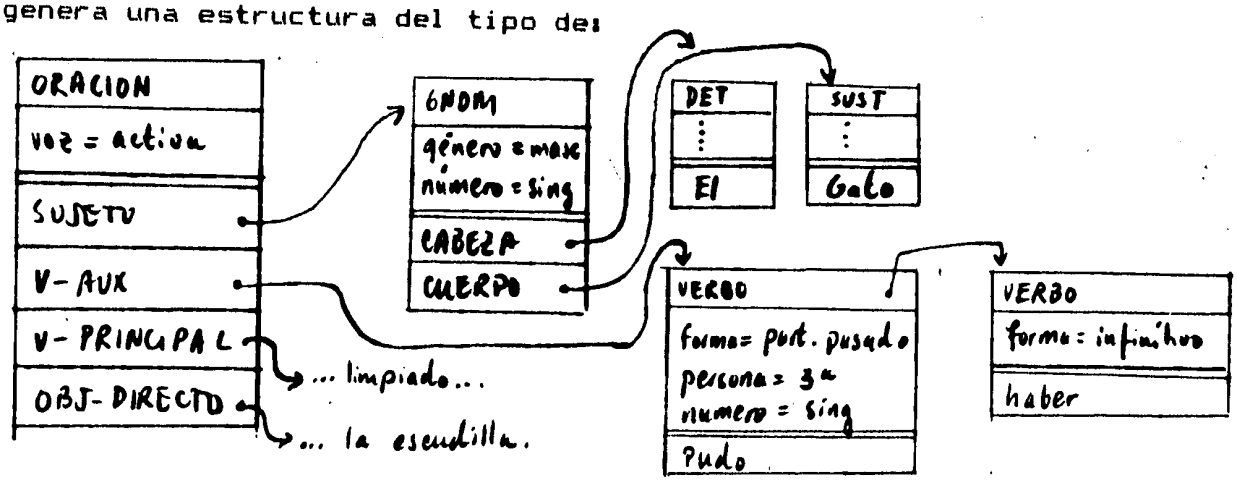
El diagrama de la red será:



La oración

"el gato pudo haber limpiado la escudilla"

genera una estructura del tipo de:



que corresponde a la estructura profunda, salvo por el marcador de VOZ, de:

"la escudilla pudo haber sido limpiada por el gato"

III.4 Extensiones al Modelo

Los ATNs equivalen en capacidad de reconocimiento al de una máquina de Turing /5/. Desde luego, hay muchos sistemas similares. Entre ellos, los ATNs se distinguen por su adaptabilidad al proceso de lenguajes naturales. Sin embargo, es necesario extender el modelo en varias direcciones para permitir el manejo de fenómenos sintácticos más complejos.

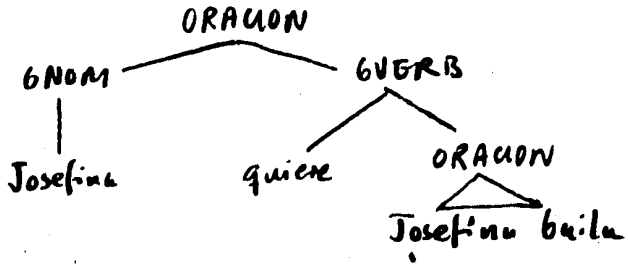
Considérese, por ejemplo, una oración del estilo de la siguiente:

"Josefina quiere bailar"

que tiene asociada una estructura profunda similar a:

"Josefina quiere (Josefina baila)

En estos casos, se requiere de un medio que pase componentes a distintos niveles dentro de la estructura:



permitiendo así que la oración en el nivel inferior comience su análisis con el sujeto del nivel superior. Para ello se usan las acciones conocidas como *iniciaciones*, en las que se indica:

1. El componente que se pasa como iniciación

2. El punto en que la red debe comenzar (dado que la red de abajo ya tiene un componente al comenzar, lo indicado es omitir el conjunto de arcos que originalmente hacen este trabajo).

Se le agrega a la red una acción tal como:

ORACION:

.
.
.

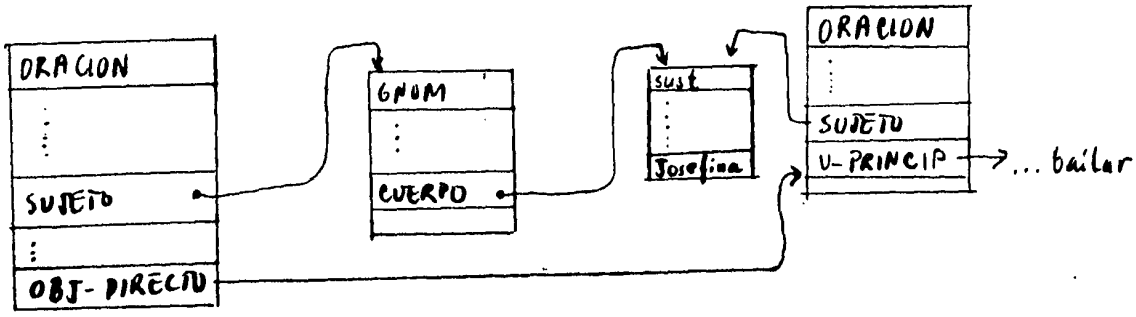
c/ORACION-b/d:
COND: ninguna
ACC: inicia SUJETO con SUJETO de

.
.
.

La notación ORACION-b ordena la ejecución de la red ORACION a partir del arco B, sorteando el arco a/GNOM/b, que forma el primer GNOM.

La acción que inicia SUJETO a SUJETO de permite que el nuevo sujeto creado al invocar ORACION contenga como sujeto al elemento SUJETO de la red que la activa. Esta acción tiene lugar antes que la red invocada comience su ejecución.

La estructura producida por el análisis anterior sería:



Requiere nuevas provisiones el fenómeno conocido como *dependencia a distancia*: un componente ha sido sacado de su lugar orginal es la estructura profunda para ser insertado en otra parte de la oración. Un ejemplo conocido es el de las oraciones interrogativas:

" cuál pez atrapó el viejo?"
que viene de la estructura profunda: "el viejo atrapo cuál pez"

" en qué río fué eso atrapado?"
de "eso fué atrapado en qué río"

" a quién quieres tú que Arturo diga eso?"
de "tú quieres (Arturo dice eso a quién)"

Uno de los atractivos originales de las reglas transformativas era precisamente el de poder mover componentes en la estructura profunda para generar estructuras superficiales necesarias.

Para el análisis con ATNs, es posible ilustrar la situación pensando que la red se topa con un componente que no tiene un lugar preciso para ser insertado, pero que debe guardar para poder colocarlo posteriormente en su lugar. El problema consiste en averiguar precisamente en qué lugar. Como puede verse en los ejemplos, tanto el componente desplazado (objeto directo, modificador y objeto indirecto, respectivamente) como su nivel original varían arbitrariamente.

El manejo de este fenómeno requiere de un almacén en el cual guardar el componente desplazado hasta hallar su lugar. Para ello se define un registro especial, conocido como *retén -hold register-*, similar a un registro de papel. Este registro se crea cada vez que se invoque la red, y tiene las siguientes características.

Primero, al comenzar la ejecución de la red, su retén se inicia con el retén de la red que lo llama. Esto permite que el contenido del retén sea común a todos los niveles. Puede ser necesario usar más de un retén.

Segundo, el retén será limpiado cada vez que éste se asigne a un papel. Esta condición garantiza que un elemento se use una sola vez en el análisis; junto con la primera propiedad, permite ver a los retenes como compartidos a todos los niveles.

Para trabajar con los retenes, se agrega a cada red un arco MANDA que parta del nodo inicial a la red. Este arco sirve para darle a la red la opción de usar el contenido del retén en vez de intentar formar un componente de la cadena de entrada. Estos arcos son conocidos como arcos VIR -virtuales-.

Basta entonces con poner una condición sobre el arco MANDA de la red ORACION que impida tomarlo en caso de que el retén no haya sido usado aún, para garantizar que cualquier análisis use todas las palabras de entrada.

Notas al Capítulo III

1. Thorne(1968), Bobrow(1969)
2. Woods(1969,1970)
3. Kaplan(1975)
4. Winograd(1983):Cap. 5
5. Bates(1978):220

IV. ATNs de la Oración Simple

IV.1 Introducción

Se describe a continuación el proceso de diseño del conjunto de ATNs para el análisis sintáctico de la oración simple del español.

Antes de dar una especificación más precisa, es adecuado recapitular sobre las herramientas con que se cuenta.

La gramática generativa y transformativa describe una variedad de oraciones simples en términos de estructuras superficiales y estructuras profundas asociadas; los ATNs reconocen estructuras superficiales bien conformadas, asociando al momento del reconocimiento al menos una estructura profunda a cada estructura superficial válida.

Con ambos elementos, el proceso de diseño es el siguiente: las estructuras superficiales dictan el modo en que los arcos se combinan para reconocer cadenas de palabras. La estructura profunda correspondiente a cada estructura superficial indica el conjunto de acciones que se deben agregar a la red para generarla.

Las condiciones que se suman a los arcos impiden el reconocimiento de cadenas que no cumplen con condiciones locales al contexto. Esta función de filtro es similar a las condiciones asociadas a las transformaciones de la gramática generativa.

Partiendo de una estructura sencilla para la oración, se construyen los componentes básicos de ella: grupos. El resultado es la oración más sencilla del español.

Después, se toman una a una las transformaciones de la oración simple. Estas implican nuevos arcos, acciones y condiciones en las redes que manejen cada nueva estructura.

El resultado es el ATN completo de la oración simple del español.

El diseño completo de las redes de este trabajo fue hecho con la descripción proporcionada por la gramática generativa y transformativa desarrollada por Roger Hadlich/1/.

IV.2 Diseño de la red más simple (componentes elementales)

El énfasis de esta sección esta puesto en el diseño de los grupos que conforman cualquier oración: grupos nominales, verbales, adjetivales y preposicionales.

Este diseño se apoya en un primer esbozo de la oración simple afirmativa. Se sigue el orden dado por el texto de Hadlich.

IV.2.1 Estructura de la oración

Las cinco primeras reglas /2/ de estructura sintagmática son el primer acercamiento a la estructura de la oración simple (omitiendo los marcadores no necesarios):

1.	ORACION-->	GNOM GVERB
2.	GVERB-->	MOD VERBAL
3.	VERBAL-->	verbo (GNOM) ! COP PREDNOM
4.	PREDNOM-->	adj ! GNOM
5.	GNOM-->	(det) sust
6.	AUX-->	verbo modal

donde: GNOM es un grupo nominal,
GVERB un grupo verbal,
MOD un verbo modal,
COP una cópula (i.e. el verbo 'ser' o 'estar')
PREDNOM un predicado nominal,
det, adj, verbo, sust, categorías léxicas terminales.

Las estructuras superficiales definidas por las producciones anteriores son, inmediatamente:

ORACION: GNOM GVERB (GNOM)

GNOM GVERB=verbo cop GADJ=adj
GNOM GVERB=verbo cop GNOM

GNOM: (det) sust

GADJ: adj

GVERB: (verbo modal) verbo

Simultáneamente se deducen los papeles asociados a las redes:

ORACION: SUJETO ACCION (OBJETO-DIRECTO)

ORACION: SUJETO COPULA PREDNOM

GNOM: CABEZA-GN CUERPO-GN

GADJ: CUERPO-GA

GVERB: (VERBO-AUXILIAR) VERBO-PRINCIPAL

Escribiendo con dichos elementos la red, llegamos a la primera especificación de la red ORACION:

```
a/GNOM/b:
COND: ninguna
ACC:  SUJETO=*

b/GVERB/c:
COND: ninguna
ACC:  ACCION=*

c/GNOM/d:
COND: (ACCION.tipo<>verbo-cop)
ACC:  OBJETO-DIRECTO=*

c/BRINCA/d:
COND: ninguna
ACC:  ninguna

c/GADJ/d:
COND: (*=adjetivo) y
      (ACCION.tipo=verbo-cop) y
      (ACCION.VERBO-P="ser")
ACC:  PREDNOM=*

c/GNOM/d:
COND: (ACCION.tipo=verbo-cop)
ACC:  PREDNOM=*

d/MANDA:
COND: ninguna
ACC:  ninguna
```

Las redes correspondientes GADJ, GVERB y GNOM no requieren, en este estado, mayor descripción.

Como ejemplos de lo anterior,

"El conejo aserró la naranja"
"El conejo llora"
"Chomsky es sano"

4.2.2 Grupos Preposicionales

Los grupos presposicionales son aquellos definidos como:

GPREP:

h/CAT=prep/i
COND: ninguna
ACC: CABEZA-GP = *

i/GNOM/j
COND: ninguna
ACC: CUERPO-GP = *

j/MANDA
COND: ninguna
ACC: ninguna

Tienen por objeto reflejar la estuctura de los *objetos de interés* (objetos indirectos), *complementos verbales* y *circunstanciales*.

La primera extensión es la regla de VERBAL/3/:

3. VERBAL--> VERBAL (GPREP) (GPREP)

El primer GPREP (con la preposición 'a') cumple con la función de integrar a la gramática el llamado objeto de interés, que abarca a los llamados *objetos indirectos* y *dativos de interés*:

"La zanahoria gusta a el conejo", y
"El conejo prohibió la entrada a Chomsky"

El reconocimiento de objetos indirectos presenta, desde un punto de vista gramatical, varios problemas/4/. Primero, es común que también el objeto directo sea precedido por la preposición 'a':

"Los pajaros saludan (a) la aurora",
"Conozco (a) esa familia",
"Tememos (a) la muerte".

Existe considerable confusión acerca de las reglas que rigen al uso de
'a' + objeto directo.

En presencia de un solo objeto, parece preferirse analizarlo como objeto directo/5/:

"El comerciante pago la deuda a Andrés"
---OD--- ---OI---, pero

"El comerciante pago a Andrés"
----OD---- .

Si se usan tanto un objeto directo como un indirecto, el directo se sitúa más cerca del verbo, posiblemente sin la 'a'/6/:

"Prefiero (a) Luisa a Juana".

El segundo GPREP representa a los *complementos verbales/7/*, grupos que se encuentran en relación estrecha con el verbo principal de la oración, como en:

"El conejo piensa en Chomsky", o bien
"Chomsky sueña con ideas naranjas"

Otro tipo de estructuras que aparecen en la oración son las llamadas *circunstanciales/8/*, usadas para calificar de maneras diversas las circunstancias de la acción señalada por el GVERB.

Las reglas de reescritura que lo introducen son las siguientes:

2'.	GVERB-->	MOD VERBAL CIRC
7.	CIRC-->	(adv) adv (CIRC) !
		GPREP (CIRC) !
		GNOM (CIRC)

donde: CIRC es el circunstancial,
adv un adverbio.

Los circunstanciales se componen principalmente de adverbios. Por ello, se clasifican según el criterio tradicional en adverbios de tiempo, modo, cantidad y lugar:

muy tarde,
algo bueno,
mucho,

demasiado lejos.

La segunda alternativa da cuenta del uso de GPREPs también como calificadores de la acción, como en:

"...desde las ocho",
"...con mucho gusto".

La inclusión de GNOMs como circunstanciales incluye el uso de construcciones tales como:

"Este libro cuesta cinco dólares",
"El conejo mide treinta centímetros".

La posibilidad de admitir GNOMs como modificadores de este género parece señirse a un grupo reducido de verbos, v.gr.: pesar, costar, medir. Debe tomarse en cuenta, sin embargo, que es posible que alguno de dichos verbos figure como un verbo transitivo común:
"Chomsky mide el conejo"

Dicha regla describe también el uso de GNOMs completos usados com adverbios:
"la semana que viene"
"el día que tú quieras"

Finalmente, la característica de recursividad del componente CIRC pretende dar cuenta del número potencialmente arbitrario de modificadores -de cualquier tipo- adicionados a una red:

"Chomsky llegará allá arriba a su cuarto después de las ocho con el conejo"

Se nota que un componente generado a partir de un PREDNOM puede contener un CIRC:

"Chomsky está arriba en su cuarto"

por lo que se modifica la regla de PREDNOM:

4''. PREDNOM--> adj ! GNOM ! CIRC

para incorporar este tipo de oraciones a la gramática.

Se consideran ahora las modificaciones a las redes motivadas por la introducción de las nuevas reglas.

Los dos GPREPs agregados a la regla de reescritura de VERBAL quedan cubiertos con un arco para recuperar los GPREPs, donde la única suposición hecha es que los objetos indirectos preceden a los circunstanciales/9/.

Para este caso, entonces:

```
d/GPREP/d:
COND: (*.CABEZA-GP = 'a') y
      (OBJETO-DIRECTO = vacío )
ACC:  OBJETO-DIRECTO = *.CUERPO-GP

d/GPREP/d:
COND: (*.CABEZA-GP = 'a') y
      (OBJETO-DIRECTO <> vacío ) y
      (OBJETO-IND = vacío )
ACC:  OBJETO-IND = *
```

y para el resto:

```
d/GPREP/d:
COND: (*.CABEZA-GP <> 'a' ) o
      (OBJETO-DIRECTO <> vacío ) o
      (OBJETO-IND <> vacío )
ACC:  MODIF-O = MODIF-O + *
```

Los circunstanciales son también, con criterios puramente sintácticos, indistinguibles de los complementos verbales, sin haber regularidades que permitan algún tipo de predicción:

"El conejo piensa en Chomsky en la cocina",

donde no basta la información de la relación 'pensar-en' para distinguir entre el complemento verbal y el circunstancial.

Es por ello que se usa un sólo papel al que se asocian todos estos GPREPS: MODIF-O.

El GNOM usado como modificador se integra naturalmente con una arco:

```
d/GNOM/d:
COND: OBJETO-DIR<>vacio
ACC:  MODIF-O= MODIF-O + *
```

Para considerar el uso de adverbios en el papel de

circunstanciales, se agrega:

```
d/GADJ/d
COND: (ACCION.tipo<>cop) o
      (ACCION.VERBO-P="estar")
ACC:  MODIF-O= MODIF-O + *,
```

adaptando al GADJ simultáneamente para permitir el reconocimiento de adverbios:

```
GADJ:
v/CAT=adj,adv/w
COND: ninguna
ACC:  CUERPO-GA = *

w/MANDA
COND: ninguna
ACC:  ninguna
```

Otra solución equivalente sería la de conservar a los GADJs unicamente para el reconocimiento de adjetivos, creando un GADV para reconocer adverbios.

IV.2.3 Grupo Verbal

Se consideran en esta sección las modifcaciones sufridas por la red por la introducción de la regla/10/:

6. AUX--> (haber -do) (estar -ndo),

que, dejando de lado el problema de los verbos modales, da cuenta de las construcciones verbales del indicativo. Las posibles formas verbales del indicativo aquí consideradas son:

presente,
imperfecto,
pretérito,
futuro,
condicional,

en sus formas de perfecto:

ha hablado, había hablado, hubo hablado, habrá hablado, habría hablado

progresivas:

está hablando, estaba hablando, estuvo hablando, estará hablando, estaría hablando

con ambas:

ha estado hablando, había estado hablando, hubo estando hablando, habrá estado hablando, habría estado hablando

o ninguna:
habla, hablaba, habló, hablará, hablaría.

El uso de las partículas '-do', '-ndo' señala la introducción, respectivamente, de las formas perfectas y progresivas. Ya que es posible la aparición de ambas en un mismo grupo verbal, se consideran como atributos separados del grupo.

De lo anterior, se sigue directamente la estructura del GVERB:

```
1/CAT=verbo/1
COND: (*=verbo 'haber') y
      (perfecto <> si)  y
      (progresivo <> si)
ACC:  (VERBO-AUX: VERBO-AUX + *) y
      (perfecto = si) y
      (tiempo= tiempo de *)

1/CAT=verbo/1
COND:  (*=verbo "estar") y
       (perfecto <> si) y
       (progresivo <> si)
ACC:   (VERBO-AUX = VERBO-AUX + * ) y
       (progresivo = si ) y
       (tiempo=tiempo de *)

1/CAT=verbo/1
COND: (*=verbo "estar") y
      (perfecto = si) y
      (progresivo <> si)
ACC:  (VERBO-AUX = VERBO-AUX + *) y
      (progresivo = si)

1/CAT=verbo/m
COND: (perfecto <> si ) y
      (progresivo <> si ) y
      (*.perfecto <>si) y
      (*.progresivo <> si)
ACC:  (VERBO-P=*) y
      (tiempo=tiempo de *)

1/CAT=verbo/m
COND: (perfecto = si ) y
      (*.perfecto=si)
ACC:  VERBO-P=*

1/CAT=verbo/m
COND: (progresivo = si) y
      (*.progresivo=si)
ACC:  VERBO-P=*

m/MANDA
```

COND: ninguna
ACC: ninguna

IV.2.4 Grupos Nominales

La estructura del GNOM introducida por las reglas de la sección IV.2.1 sugieren la siguiente red:

p/CAT=det/r
COND: ninguna
ACC: CABEZA-GN= *

p/BRINCA/r
COND: ninguna
ACC: ninguna

r/CAT=sust/s
COND: ninguna
ACC: CUERPO-GN = *

s/MANDA
COND: ninguna
ACC: ninguna,

donde el 'det' que forma la CABEZA-GN no ha sido especificado aún.

A continuación se detallan las correcciones a que dan lugar las consideraciones introducidas por las extensiones a los grupos nominales.

Se llaman *determinantes* a aquellas formas del español usadas para calificar de alguna manera al nombre que acompañan.

La estructura de los determinantes está descrita por las reglas/11/:

1. DET--> INDEF (NUM) !
DEF (otr-) ! DEF (NUM) !
calif (otr-)! FRAC (otr-)

2. INDEF--> art-ind ! otr-

3. DEF--> (FRAC) pos ! (FRAC) dem !
(FRAC) art-def

4. NUM--> card ! ord

5. FRAC--> tod- ! GNOM "de"

donde: calif representa a la categoría de palabras, casi todas adjetivos, como: mucho, poco, bastante; y otras, como: más, menos/12/

art-ind al artículo indefinido,
art-def al artículo definido,
pos al pronombre posesivo,
dem al adjetivo demostrativo,
card a un número cardinal,
ord a un números ordinal.

Como ejemplos de las estructuras generables por dichas reglas, se cuentan:

"un (segundo) libro"
"otros (tres) libros"
de INDEF,

"tu (otra) queja"
"tu (tercera) queja"
de DEF-->pos,

"esos (otros) bichos"
"estos (tres) bichos"
de DEF-->dem,

"los tres conejos"
"el quinto conejo"
de DEF-->art-def,

"tres litros de leche"
de DET-->FRAC.

El segmento de red propuesto para reconocer el *det* de un GNOM es el siguiente:

p/BRINCA/q:
COND: tipo<>det-frac
ACC: ninguna

p/CAT=art-ind/q,
p/LEX='OTRO'/q:
COND: ninguna
ACC: (CABEZA-GN = CABEZA-GN + *) y
(tipo = indef)

p/CAT=pos,dem,art-def/q:
COND: ninguna
ACC: (CABEZA-GN = CABEZA-GN + *) y
(tipo = def)

```
p/LEX='TODO'/p:
COND: tipo <> det-frac
ACC:  (CABEZA-GN = CABEZA-GN + * ) y
      (tipo = det-frac)

p/calif/q:
COND: ninguna
ACC:  (CABEZA-GN = CABEZA-GN + * ) y
      (tipo = calif )

q/CAT=card,ord/r:
COND: (tipo = def ) o
      (tipo = indef )
ACC:  CABEZA-GN = CABEZA-GN + *

q/LEX='OTRO'/r
COND: (tipo <>indef) y
      (tipo <>vacio)
ACC:  CABEZA-GN = CABEZA-GN + *

q/BRINCA/r
COND: ninguna
ACC:  ninguna

s/LEX='de'/p
COND: ninguna
ACC:  (CABEZA-GN = CABEZA-GN + *) y
      (CABEZA-GN = CABEZA-GN + CUERPO-GN) y
      (CUERPO-GN = vacio) y
      (tipo=vacio)
```

IV.2.5 Concordancia

Los dos tipos básicos de concordancia del español son la concordancia *verbal* y la concordancia *nominal* /13/.

La concordancia verbal exige que el sujeto profundo de la oración concuerde en persona y número con el verbo. Suponiendo que el GVERB reconocido por el arco b/GVERB/c contenga las características persona y número, se agregan:

```
b/GVERB/c:
COND: (*.persona = SUJETO.persona) y
      (*.número = SUJETO.número ).
```

El segundo tipo de concordancia, la nominal, dicta que deben concordar en género y número el nombre y todos sus modificadores: determinantes, adjetivos.

Para el adjetivo, la concordancia se verifica en:

```
c/GADJ/d:
COND: (*.género = SUJETO.género) y
      (*.número = SUJETO.número).
```

Finalmente, el 'det' de un GNOM concordar con el sustantivo que lo domina. Para ello, los arcos p/.../q del GNOM deben asignar a la red los valores de género y número de CABEZA-GN, y validarlos contra los componentes de los arcos q/.../r y r/.../s.

IV.2.6 Resumen

Aquí se recapitula el diseño de la red alcanzado después de concluir con todas las reglas que definen a los componentes elementales de la oración.

```
ORACION:

a/GNOM/b:
COND: ninguna
ACC:  SUJETO = *

b/GVERB/c:
COND: (*.persona = SUJETO.persona) y
      (*.número = SUJETO.número)
ACC:  ACCION = *

c/BRINCA/d:
COND: ninguna
ACC:  ninguna

c/GADJ/d:
COND: (ACCION.tipo = verbo-cop) y
      (ACCION.VERBO-P = "ser") y
      (*.número = SUJETO.número ) y
      (*.género = SUJETO.género )
ACC:  PREDNOM = *

c/GNOM/d:
COND: ACCION.tipo = verbo-cop
ACC:  PREDNOM = *

d/GPREP/d:
COND: (*.CABEZA-GP = 'A') y
      (OBJETO-DIR = vacío)
ACC:  OBJETO-DIR = *.CUERPO-GP
```

```
d/GPREP/d:
COND: (*.CABEZA-GP = 'A') y
      (OBJETO-DIR = vacío) y
      (OBJETO-IND = vacío)
ACC:  OBJETO-IND = *.CUERPO-GP

d/GPREP/d:
COND: (*.CABEZA-GP <> 'A') o
      (OBJETO-DIR <> vacío) o
      (OBJETO-IND <> vacío)
ACC:  MODIF-O = MODIF-O + *

d/GNOM/d:
COND: OBJETO-DIR <>vacio
ACC:  MODIF-O = MODIF-O + *

d/GADJ/d:
COND: (ACCION.tipo<>cop) o
      (ACCION.VERBO-P = "estar")
ACC:  MODIF-O= MODIF-O + *


GVERB:

1/CAT=verbo/1:
COND: ( * = verbo "haber") y
      (perfecto <> si)
ACC:  (VERBO-AUX = VERBO-AUX + *) y
      (perfecto = si) y
      (tiempo = *.tiempo) y
      (persona = *.persona) y
      (numero = *.numero)

1/CAT=verbo/1
COND: ( * = verbo "estar") y
      (perfecto <> si) y
      (progresivo <> si)
ACC:  (VERBO-AUX = VERBO-AUX + * ) y
      (progresivo = si) y
      (tiempo = *.tiempo)

1/CAT=verbo/1:
COND: ( * = verbo "estar") y
      (perfecto = si) y
      (progresivo <> si)
ACC:  (VERBO-AUX = VERBO-AUX + * ) y
      (progresivo = si ) y
      (persona = *.persona) y
      (numero = *.numero)

1/CAT=verbo/m:
COND: (perfecto <> si) y
```

```
        (progresivo <> si) y
        (*.perfecto <> si) y
        (*.progresivo <> si ) y
        ( (persona=vacio) o (persona=*.persona) ) y
        ( (numero=vacio) o (numero=*.numero) )
ACC:    (VERBO-P = *) y
        (tiempo = *.tiempo) y
        (persona = *.persona) y
        (numero = *.numero)

l/CAT=verbo/m:
COND:  (perfecto=si) y
       (*.perfecto = si)
ACC:   VERBO-P = *

l/CAT=verbo/m:
COND:  (progresivo = si) y
       (*.progresivo = si)
ACC:   VERBO-P = *

m/MANDA
COND: ninguna
ACC:  ninguna


GPREP:

h/CAT=prep/i:
COND: ninguna
ACC:  CABEZA-GP = *

i/GNOM/j:
COND: ninguna
ACC:  CUERPO-GP = *

j/MANDA
COND: ninguna
ACC:  ninguna


GADJ:

v/CAT=adv,adj/w
COND: ninguna
ACC:  CUERPO-GA = *

w/MANDA
COND: ninguna
ACC:  ninguna
```

```
GNOM:

p/BRINCA/q:
COND: tipo<>det-frac
ACC:  ninguna

p/CAT=art-ind/q,
p/LEX='OTRO'/q:
COND: ninguna
ACC:  (CABEZA-GN = CABEZA-GN + * ) y
      (tipo = indef ) y
      (genero = *.genero) y
      (numero = *.numero)

p/CAT=pos,dem,art-def/q:
COND: ninguna
ACC:  (CABEZA-GN = CABEZA-GN + * ) y
      (tipo = def ) y
      (genero = *.genero) y
      (numero = *.numero)

p/LEX='TODO'/p:
COND: ( (genero=vacio) o (genero=*.genero) ) y
      ( (numero=vacio) o (numero=*.numero) ) y
      (tipo <> det-frac)
ACC:  (CABEZA-GN = CABEZA-GN + * ) y
      (tipo = det-frac) y
      (genero = *.genero) y
      (numero = *.numero)

p/CAT=calif/q:
COND: ninguna
ACC:  (CABEZA-GN = CABEZA-GN + * ) y
      (tipo = calif ) y
      (genero = *.genero) y
      (numero = *.numero)

q/CAT=card,ord/r:
COND: ( (tipo = def ) o (tipo = indef ) ) y
      ( (genero=vacio) o (genero=*.genero) ) y
      ( (numero=vacio) o (numero=*.numero) )
ACC:  (CABEZA-GN = CABEZA-GN + *) y
      (genero=*.genero) y
      (numero=*.numero)

q/LEX='OTRO'/r
COND: (tipo <>indef) y
      (tipo <> vacio ) y
      ( (genero=vacio) o (genero=*.genero) ) y
      ( (numero=vacio) o (numero=*.numero) )
ACC:  (CABEZA-GN = CABEZA-GN + *) y
      (genero=*.genero) y
      (numero=*.numero)
```

```
q/BRINCA/r
COND: ninguna
ACC:  ninguna

r/CAT=sust/s:
COND: ( (genero=vacio) o (*.genero=genero) ) y
      ( (numero=vacio) o (*.numero=numero) )
ACC:  (CUERPO-GN = *) y
      (genero=*.genero) y
      (numero=*.numero) y
      (persona=tercera)


s/LEX='de'/p
COND: ninguna
ACC:  (CABEZA-GN = CABEZA-GN + *) y
      (CABEZA-GN = CABEZA-GN + CUERPO-GN) y
      (CUERPO-GN = vacio) y
      (tipo = vacio )

s/MANDA
COND: ninguna
ACC:  ninguna
```

IV.3 Agregados a la red más simple

En esta sección se detallan los nuevos arcos asociados a diversos fenómenos sintácticos del español:

- voz pasiva,
- oraciones impersonales,
- otras oraciones sin sujeto,
- pronombres,
- oraciones reflexivas,
- oraciones negativas,
- oraciones interrogativas,
- oraciones imperativas.

IV.3.1 Voz Pasiva

Este caso comprende aquellas formas en las que el sujeto profundo aparece como objeto superficial, y el objeto profundo como sujeto superficial /14/:

"El conejo fue alimentado por Chomsky"

Procede invertir la asignación de sujeto y objeto directos, en el caso en el que el GVERB haya sido marcado como

de voz pasiva:

a a/GNOM/b se le agrega:

```
ACC:  voz = activa,
```

```
b/GVERB/c:
COND: (GVERB.VERBO-P.perfecto = si ) y
      (GVERB.VERBO-AUX contiene 'SER') y
      (tipo = activa)
ACC:  (OBJETO-DIR = SUJETO ) y
      (SUJETO = vacio ) y
      (tipo = pasiva ).
```

Falta por asignar el sujeto:

```
d/GPREP/d:
COND: (tipo = pasiva ) y
      (*.CABEZA-GP = 'POR') y
      (SUJETO = vacio )
ACC:  SUJETO = *.CUERPO-GP.
```

Nótese que hay oraciones pasivas potencialmente ambiguas, como en:

"La clase fue aplaudida",

donde la clasificación de la oración puede ser como de *prednom* o de *pasiva*, dependiendo de la aparición en el diccionario de 'aplaudida' como adjetivo, participio pasado, o ambas.

También existe el caso de las llamadas *pasivas con 'se'* /15/, como en:

"La carta se escribió ayer",

en donde el *se* es únicamente un marcador superficial:

```
b/LEX='SE'/b
COND: tipo = activa
ACC:  tipo = pasiva-se
```

IV.3.2 Oraciones Impersonales

En este apartado entran distintos tipos de oraciones sin sujeto /16/:

"llueve", "hace frío",... -atmosféricas-,
""amanece", "anochece",... -de tiempo-,

"Se habla español"

en las que no existe sujeto profundo, i.e., no existe agente de la acción.

Para los dos primeros casos, lo más sencillo es contar con que los verbos adecuados están indicados en el diccionario como 'sin-agente':

```
b/GVERB/c:
COND: (*.VERBO-P.sin-agente = si)
ACC:  (tipo = sin-agente) y
      (SUJETO = vacío).
```

El tercer caso se maneja, de acuerdo a la regla que los introduce,como:

```
a/LEX='SE'/b:
COND: ninguna
ACC:  (tipo = sin-agente) y
      (SUJETO = vacío).
```

IV.3.3 Otras Oraciones sin Sujeto

El español omite usualmente al sujeto, por razones de énfasis -más bien, para no dar énfasis /17/:

"Llegamos tarde a la fiesta",
"hablamos japonés".

Lo primero es no exigir el sujeto:

```
a/BRINCA/b
COND: ninguna
ACC:  SUJETO = vacío,
```

completándolo después con la información del GVERB:

```
b/GVERB/c:
COND: SUJETO = vacío
ACC:  asigna al SUJETO con el pronombre personal con número y
      persona del GVERB
```

(aunque la acción señalada no está incluida dentro de las permitidas por el formalismo, es fácil extenderlo para incluir posibilidades como la última).

IV.3.4 Pronombres

El primer caso que surge en el tratamiento de pronombres es el de los llamados *pronombres disyuntivos* /18/:

yo, tú, él, ella, ello
nosotros, nosotras,ustedes, ellos, ellas

más las formas especiales mí, ti.

Se incluyen además en esta categoría los llamados pronombres indefinidos

algo, nada, alguien, nadie,

considerados por Hadlich como formas particulares -positivas y negativas- de los pronombres neutros /19/. Como ejemplos, tenemos:

"Saldré con ustedes",
"Ella no lo sabe",
"Yo sé algo".

Estos elementos se reconocen insertando directamente un arco paralelo al que reconoce sustantivos en el GNOM:

r/CAT=pron-disy/s:
COND: ninguna
ACC: CUERPO-GN = *.

Los *pronombres conjuntivos* -aquellos que aparecen junto a un verbo- son, de objeto directo /20/:

me, te, lo(le), la
nos, los(les), las,

y de objeto indirecto los mismos, salvo en la tercera persona:

le,les

postergando por ahora el análisis del pronombre *se*.

Los papeles que pueden cumplir los pronombres conjuntivos son los siguientes:

1. En lugar de un objeto directo:
   "Carlos la vió"

2. En lugar de un objeto indirecto:
   "Carlos le mando un paquete"

y de manera redundante, como:

3. Como un objeto directo:
   "Carlos la vió a ella"

4. Como un objeto indirecto:
   "Carlos le mando un paquete a él"

Se opta por conservar toda la información posible, de modo que se conserven ambos objetos:

```
b/CAT=pron-conj/b:
COND: (* = pron-conj de OD) y
      (OBJETO-DIR = vacio )
ACC:  OBJETO-DIR = *

b/CAT=pron-conj/b:
COND: (* = pron-conj de OI) y
      (OBJETO-IND = vacio)
ACC:  OBJETO-IND = *,
```

concatenando, en lugar de asignar, a los objetos directos e indirectos en las demás partes de la red.

A los mismos arcos se les agrega la prueba de concordancia en número y persona con los originalmente reconocidos:

```
d/GPREP/d:
COND: (*.CABEZA-GP = 'A') y
      (OBJETO-DIR = vacio )
ACC:  OBJETO-DIR = *.CUERPO-GP

d/GPREP/d:
COND: (*.CABEZA-GP = 'A') y
      (OBJETO-DIR = pron-conj de OD) y
      (*.número = OBJETO-DIR.número) y
      (*.persona = OBJETO-DIR.persona)
ACC:  OBJETO-DIR = OBJETO-DIR + *.CUERPO-GP

d/GPREP/d:
COND: (*.CABEZA-GP = 'A') y
      (OBJETO-DIR <> vacio) y
      (OBJETO-DIR <> pron-conj de OD) y
      (OBJETO-IND = vacio)
ACC:  OBJETO-IND = *.CUERPO-GP

d/GPREP/d:
COND: (*.CABEZA-GP = 'A') y
      (OBJETO-DIR <> vacio) y
      (OBJETO-DIR <> pron-conj de OD) y
      (OBJETO-IND <> vacio) y
```

```
       (OBJETO-IND = pron-conj de OI) y
       (*.número = OBJETO-IND.número) y
       (*.persona = OBJETO-IND.persona)
ACC:   OBJETO-IND = OBJETO-IND + *.CUERPO-GP

d/GPREP/d:
COND:  (*.CABEZA-GP <> 'A')
ACC:   MODIF-O = MODIF-O + *

d/GPREP/d:
COND:  (OBJETO-DIR <> vacío) y
       (OBJETO-DIR <> pron-conj de OD) y
       (OBJETO-IND <> vacío) y
       (OBJETO-IND <> pron-conj de OI)
ACC:   MODIF-O = MODIF-O + *
```

Finalmente, se agrega una condición que permita pronombres de objeto indirecto sólo antes que de directo:

"Carlos te las manda", pero
* "Carlos las te manda"

```
b/CAT=pron-conj/b:
COND:  OBJETO-DIR = vacío
```

Se considera que estructuras del tipo de:

"el otro...",
"tres galones de éste...",
"un quinto..."

como representantes de un GNOM tienen su origen en un rasgo +pro -pronombre- en la estructura profunda asociada al *det* del GNOM, convirtiendo a éste en representante de todo el grupo.

Repasando la estructura de los determinantes /21/, se nota que dicho procedimiento de nominalización produce las estructuras indicadas por las siguientes condiciones adicionadas a arcos del grupo nominal:

```
r/BRINCA/s:
COND:  (tipo = indef) y
       (paso = si ),

       (tipo = def) y
       (*.CABEZA-GN contiene un pron-dem),

       (tipo = def) y
       (paso = si ) y
       (*.CABEZA-GN contiene un art-def),
```

```
        (tipo = def) y
        (paso = si ) y
        (*.CABEZA-GN contiene un pos),

        (tipo = calif)

ACC:    (CUERPO-GN = CABEZA-GN ) y
        (CABEZA-GN = vacio ),
```

donde se les ha agregado la acción:

ACC: paso = si

a los arcos q/CAT=ord,card/r, q/LEX='OTRO'/r, y la acción:

ACC: paso = no

al arco q/BRINCA/r.

IV.3.5 Oraciones Reflexivas

Los *pronombres conjuntivos reflexivos* son /22/:

me, te, se, nos

y se distinguen tres clase de oraciones reflexivas de acuerdo al tipo de verbo con que se usan: los verbos reflexivos *propios* son aquellos en los que el pronombre reflexivo representa, según el caso, un objeto directo o indirecto. V. gr.: acercar, alejar, detener.

El segundo tipo es el de verbos reflexivos *inherentes*. Comprende esta clase a los verbos en los que el pronombre reflexivo es un mero marcador superficial, sin representar objeto profundo alguno. Por ejemplo: arrepentirse, dignarse, suicidarse.

Finalmente, los verbos reflexivos *opcionales* incluyen tanto a verbos transitivos como intransitivos, con ambos usos. Así, desayunar(se), parecer(se), volver(se) pertenecen a esta categoría.

El primer tipo de verbos queda cubierto con los arcos agregados en la sección anterior para el manejo de pronombres de objeto directo e indirecto. El segundo caso exige que se suprima el objeto levantado en caso de ser el verbo reflexivo. Para ello,

```
b/GVERB/c:
COND: *.reflexivo = si
ACC:  (OBJETO-DIR = vacío) y
      (OBJETO-IND = vacío).
```

El reconocimiento de los verbos de la tercera clase depende del modo en que estén considerados por el diccionario.

IV.3.6 Oraciones Negativas

Las oraciones *negativas* del español se marcan superficialmente con el adverbio *no* /23/. Este se coloca en la oración ordenada canónicamente entre el sujeto y el grupo verbal:

```
b/LEX='NO'/b:
COND: ninguna
ACC:  afirmativa = no
```

Por ejemplo:

"Chomsky no alimenta el conejo",
"La carta no se escribirá mañana",
"yo no se lo dije".

IV.3.7 Oraciones Interrogativas

Se divide a las oraciones interrogativas /24/ en dos tipos: oraciones interrogativas *dubitativas*, como:

" Viene Chomsky a comer?",

que se contestan con un 'si' o 'no', y las oraciones interrogativas *informativas*, como en:

" Dónde está el libro?".

Para el primer caso, basta con invertir el orden sujeto-objeto, creando para ello tres nuevos arcos:

```
a/LEX=' '/e:
COND: ninguna
ACC:  interrogativa= si

e/GVERB/f:
```

.
.
similar a b/GVERB/c
.
.
.

f/GNOM/c:
.
.
similar a a/GNOM/b
.
.
.

verificando al final de la red:

d/LEX='?'/g:
COND: interrogativa = si
ACC: ninguna

g/MANDA:
COND: ninguna
ACC: ninguna

Sintácticamente, es posible describir a las oraciones interrogativas informativas como aquellas en las que algún elemento de la estructura profunda ha sido convertido en pronombre interrogativo y trasladado al frente de la oración (después de aplicar la misma inversión del sujeto):

"Yo casaría ha mi hija con tal persona"

" Quién casaría a mi hija con tal persona?"
" A quién casaría yo con tal persona?"
" Con qué persona casaría yo a mi hija?"
" Con quién casaría yo a mi hija"

La variedad de los objetos trasladados se refleja en los pronombres interrogativos:

qué, quién

para grupos nominales en cualquier papel, incluso el objeto de un grupo preposicional -CUERPO-GP. En cualquier caso en el que el objeto trasladado sea un GPREP, la preposición se mueve junto con el pronombre:

" Qué lees?"
" Con quién vas al cine?";

para 'det' de un grupo nominal, se usan:

qué, cuál, cuánto

como en:

" De qué leche desayuno?"
" Cuáles otros hombres?";

para GNOMs, GPREPs y GADJs asociados a circunstanciales, y para GADJs de predicados nominales:

dónde, cuándo, cuánto, cómo

" Desde cuándo comes en el Café Nacional?"

Siguiendo el uso del registro de *retén*/25/ para manejar este caso, se agregan las asignaciones al registro de retén y los arcos virtuales -VIR- que lo recuperan:

g/CAT=pron-int/e:
COND: ninguna
ACC: ELEMENTO-INT = *

junto con los arcos virtuales adecuados en GNOM, GPREP y GADJ para que lo recuperen. Además,

f/CAT= prep/g:
COND: ninguna
ACC: PREP-INT = *

f/BRINCA/g:
COND: ninguna
ACC: ninguna

La excepción a este manejo, por no constituir un grupo en sí, es el 'det' de un grupo nominal:

p/CAT=pron-int/q:
COND: * = pron-int.origen = det

.
.
.

IV.3.8 Oraciones Imperativas

En esta sección se examina el caso más sencillo de oraciones imperativas /26/, las llamadas *imperativas directas*, como por ejemplo:

"(Tú) come esa sopa",
"(Ustedes) no me ayuden".

Las condiciones para las imperativas directas son: que el sujeto está en primera o segunda persona; el verbo principal está en primera o segunda persona; el verbo principal ya sea que se encuentra en subjuntivo, es irregular (haz,ten,pon) o tiene la misma forma de la tercera persona del singular (habla,come).

Suponiendo que los verbos se encuentran marcados adecuadamente, se obtiene directamente:

b/GVERB/c:
COND: *.imperativo = si
ACC: modo = imperativo
.
.
.

Las oraciones imperativas negativas están consideradas por el mismo arco.

V. Un Intérprete de ATNs

V.1 Introducción

La parte final de este trabajo incluyó la programación de un sistema para el desarrollo y prueba de gramáticas de ATNs.

Ya que el formalismo presupone que las redes son no deterministicas, se requiere de un control para ejecutar ordenadamente las alternativas de los recorridos. Se desea también que el sistema entregue todos los análisis posibles de cada oración. Esto exige que el sistema pueda administrar las estructuras resultantes de todas las trayectorias avanzadas a cada momento.

Se eligió realizar un intérprete a profundidad y hacia abajo, en la versión del lenguaje de programación *Pascal* corriendo actualmente en una computadora PDP-11/34, bajo el sistema operativo RSX-11M. La elección de Pascal como lenguaje de trabajo se tomó por los siguiente:

1. Pascal es un lenguaje moderno, que incorpora conceptos que lo hacen poderoso y fácil de programar. Por ejemplo, fueron de gran ayuda las facilidades para manejar estructuras dinámicas, así como la posibilidad de usar procedimientos recursivos.

2. Pascal es uno de los lenguajes más difundidos actualmente, y seguramente seguirá siendo popular en el futuro próximo. Esto hace que el sistema sea, al menos potencialmente, transportable a una variedad de máquinas, de todos tamaños.

3. Al ser Pascal un lenguaje que se compila, y no que se interpreta, permite, en principio, que el sistema sea suceptible de mejoras que lo vuelvan más eficiente. Por ejemplo, modificando el tamaño de la memoria del heap, usando overlays, etc.

Estas consideraciones fueron decisivas, haciendo que Pascal fuera preferido, por ejemplo, a LISP. Sin embargo, el uso de Pascal trajo algunas desventajas. En LISP, que trabaja en un ambiente de intérprete, no se hubiera necesitado una gran cantidad de código en Pascal requerido para la ejecución del intérprete.

V.2 Descripción del Sistema

El programa sigue en su ejecución los siguientes pasos:

1. Lee el código que define las redes
2. Lee la oración en español y localiza cada palabra en el diccionario
3. Analiza la oración
4. Imprime los resultados

A continuación se da una descripción general de cada paso. El programa completo se encuentra listado en el primer apéndice.

Lectura del Código:

El procedimiento *leecodigo* trae de un archivo de texto ('ATN.DAT') la descripción de las redes, y después de un pequeño análisis, da de alta dichas descripciones en tablas internas. Estas tablas son *codigo*, *cond* y *acc*, apuntadas por *tarcos*, *tcond* y *tacc* respectivamente.

Se usó el siguiente formato para el código que representa la gramática de ATNs: la linea que comienza la descripción de cada arco contiene los nodos de llegada y salida del arco y el tipo del arco. A continuación, las condiciones asociadas al arco, (en caso de que las haya) y finalmente, las acciones asociadas al arco (en caso de que las haya), a efectuarse en caso de que se cumplan las condiciones requeridas.

El formato de cada linea es fijo, y es el siguiente (los elementos entre paréntesis son opcionales):

para los *arcos*:

```
          (red:)  nodo_o  -->  nodo_d, tipo_a  (inf_arco)

          |       |            |       |       |
columna:  1       10           13      15      23
```

red es una etiqueta que identifica al primer arco de una red. Las redes reconocidas por el sistema son ORACION, GNOM, GVERB, GPREP y GADJ.

nodo_o es un carácter entre la 'A' y la 'Z' que define al nodo origen del arco.

nodo_d es un carácter entre la 'A' y la 'Z' que define al nodo destino del arco.

tipo_a es un identificador que define al tipo del arco. Los identificadores de tipo de arco válidos son CAT, LEX, MANDA, BUSCA y BRINCA.

inf_arco es información extra asociada al arco, y que depende del tipo de éste:

para un arco CAT, un número entre 0 y 99 equivalente al número de categoría léxica a localizar,

para un arco BUSCA, un identificador de red, similar a *red*,

para un arco LEX, una palabra de 15 caracteres entre apóstrofes.

Por ejemplo, considérense las siguientes lineas (los espacios en blanco están marcados con un punto):

```
ORACION:.A->B,BUSCA...GNOM
```

representa un arco que BUSCA un GNOM y que va de *A* a *B*. El arco está marcado como el arco de entrada a la red ORACION

```
.........B->C,CAT.....02
```

representa a una arco de *B* a *C* que intenta cazar una palabra de la CATegoría definida con el número 2

```
.........F->G,LEX.....'CADA............'
```

representa un arco LEX de *F* a *G*. La palabra buscada es 'CADA'

Para las *condiciones*:

```
COND: tipo_c cond_izq comp tipo_der cond_der (Y)

        |      |      |       |     |       |        |
columna: 1      8      14      20    22      27       34/46
```

COND: el principio de campos de condiciones. En caso de haber varias de ellas se únicamente en el primer renglón,

tipo_c es un identificador que define el tipo de los elementos a comparar:CARAC para características y PAPEL para contenidos de papeles,

cond_izq identifica al primer elemento a comparar. Es un número entre 0000 y 9999, que se interpreta de la siguiente manera:

CARAC *x1 x2 x3 x4*
representa a la característica número *x4* asociada al papel número *x3* del papel número *x2* del papel *x1*

PAPEL *y1 y2 y3 y4*
representa al papel *y4* asociado al papel*y3* asociado al papel *y2* asociado al papel *y1*.

Para ambos casos, debe notarse:

1. Si el número es positivo o no tiene signo, el nodo raiz -aquel que se toma como primera referencia- es el nodo que se está cazando (el que anteriormente se llamó '*'). Si el número es negativo, el nodo raiz es el nodo superior.

2. Los dígitos iguales a cero no cuentan. Así, es lo mismo escribir '0012', que '0120', que '1200'.

Los detalles de uso de los números que identifican papeles y características se encuentran en los procedimientos *traep* y *traep* del procedimiento *rur*.

comp es un carácter que indica la comparación solicitada: '=' para igualdad, '#' para diferencia,

tipo_der indica la manera de identificar el segundo elemento de la comparación (*cond_der*): CONS si *cond_der* debe interpretarse como una constante absoluta contra la cual comparar, NODO en cualquier otro caso,

cond_der representa al segundo elemnto de la comparación. Si vienne precedido por NODO, se interpreta de la misma manera que *cond_izq*; si lo es por CONS, se toma como una constante. Para papeles, una constante es una cadena de 15 caracteres entre apóstrofes; si es una comparación de características, un número entre 0000 y

9999, con o sin signo,

Y se usa como un símbolo de continuación, para el caso de secuencias de condiciones. Este caso contempla secuencias de condiciones de la forma:
(cond 1) Y (cond 2) Y .... (cond n)

Si *cond_der*, el campo anterior, es una constante numérica, la y se coloca en el campo 34 de la linea, si no, en el 46.

Como ejemplos de lo anterior,

```
COND:..CARAC.-0021.#.CONS..0001
```
se pregunta si la primera característica del segundo papel de la red a la que pertenece el arco es diferente a la constante 1 (la interpretación de dicha constante depende enteramente del diccionario)

```
COND:..PAPEL..0001.=.CONS.'A.............'..Y
.......CARAC.-0002.=.NODO..0002
```
se pregunta si:
1. el contenido del primer papel de l nodo que se está resolviendo ('*') es una 'A'
2. la segunda característica de la red madre corresponde a la segunda característica de '*'

Para las acciones:

```
          ACC: tipo_acc acc_izq tipo_der acc_der (Y)
          |    |        |       |        |       |
columnas: 1    8        14      20       25      32
```

ACC: indica al primer renglón de un conjunto de acciones,

tipo_acc indica el tipo de acciones que se pueden ordenar. Se distinguen tres de ellas: CONCAT, para concatenar papeles; CARAC, para asignar valores a características; y PAPEL para asignar papeles,

acc_izq indica el elemento a afectar, y se interpreta del mismo modo que *cond_izq*. Dado que solo hace sentido afectar elementos del nivel de llamada -y no de más abajo-, éste valor es siempre positivo y se refiere a la red superior.

tipo_der indica el tipo de operando que define

acc_der. Puede ser CONS, si se piensa dar un valor predeterminado, o NODO, si es una variable,

acc_der representa al elemento a asignar. En caso de ser una variable, ésta se representa con un número, del mismo modo que antes,

Y indica que hay más de una acción asociada al arco.

Como ejemplo:

ACC:...PAPEL..0006.NODO..0000
se indica que se asigne al sexto papel de la red superior el nodo cazado ('*').

ACC:...CARAC..0001.NODO..0001..Y
.......CARAC..0002.NODO..0002
asigna la primera y segunda características de * a la primera y segunda características de la red que llama

Las tablas usadas para almacenar el código leido siguen, naturalmente, el formato usado para cada tipo de linea. El formato de cada registro de la tabla *codigo*, usada para almacenar la definición de cada arco, es el siguiente:

```
condreg = RECORD
            tipc: tipocond;
            busca: tipocp;
            ciza: INTEGER;
            tder: tipoid;
            cder: INTEGER;
            sisc: BOOLEAN
          END;
```

Cada registro de *tcond*, la tabla de condiciones, tiene la forma:

```
codres = RECORD
                origen, destino: CHAR;
                CASE arco: tipoarco OF
                        CAT: ( elm: numid );
                        BUSCA: ( blanco: tipored );
                        LEX: ( dicpal: INTEGER )
        END;
```

Finalmente, la tabla de acciones, *tacc*, se compone por registros:

```
accres = RECORD
                tipa: tipoacc;
                aiza: INTEGER;
                tder: tipoid;
                ader: INTEGER;
                siga: BOOLEAN
         END;
```

Cada papel, característica y valor de característica tiene asociado un número que la define. Para ello, se cuenta con un archivo -único- de identificadores ('IDENTIF.DAT'), que tiene el siguiente formato:

```
idres = RECORD
                r0: tipid;
                r1: numid;
                r2: numid;
                r21: numid;
                r3: nomid
        END;
```

donde: *r0* señala al tipo de identificador buscado,

r1, r2, r21 son números enteros entre 0 y 63 que sirven como llaves para recuperar el identificador asociado,

r3 es el identificador asociado (15 caracteres).

De acuerdo al tipo de identificador buscado, la llave se forma de la siguiente manera (los detalles de su uso se encuentran en el procedimiento *imparbol*):

para un *papel*:
r0 = papid
r1 = número de red a la que pertenece el papel
r2 = número de papel en la red
r21 = 0

para una *característica*:
r0 = caracR
r1 = número de red a la que pertenece la característica
r2 = número de característica de la red
r21 = 0

para los *valores* que puede tomar una característica:
r0 = valtC
r1 = número de red de la característica
r2 = número de característica en la red
r21 = valor que tiene la característica

para una *categoría léxica*:
r0 = catlex
r1 = número de categoría
r2 = 0
r21 = 0

para el nombre de una *propiedad* de una categoría léxica:
r0 = caracC
r1 = número de categoría léxica
r2 = número de propiedad de la categoría
r21 = 0

para *los valores* que puede tomar una propiedad de una categoría léxica:
r0 = valtCX
r1 = número de categoría
r2 = número de propiedad de la categoría
r21 = valor asignado

El número de red referido es fijo, y es el siguiente:

```
ORACION = 1
GNOM    = 2
GVERB   = 3
GPREP   = 4
GADJ    = 5
```

Lectura de la Oración

La lectura de la oración a analizar usa un archivo para localizar cada palabra y sus características asociadas. Este archivo se llama 'DICC.DAT', y sus registros son como sigue:

```
diccreg = RECORD
            ident: cadena;
            catpal: numid;
            atr: ARRAY[indcarac] OF numid
          END;
```

donde: *ident* contiene la palabra definida (16 caracteres)

catpal es un número entre 0 y 63 que define la categoría léxica de la palabra. En caso de haber palabras con más de una posibilidad, el diccionario debe contener una entrada para cada una

atr es un conjunto de atributos (cinco) asociados a la palabra.

Debe notarse que la única interpretación que tienen los valores contenidos en *catpal* y en *atr* es la definida por el archivo de identificadores.

El Intérprete

El procedimiento *rur* -recorre una red- es el encargado de interpretar a las redes. Los dos parámetros de *rur* son: la identificación de la red a interpretar, y una variable que indica, a la salida de su ejecución, si *rur* tuvo o no éxito. La llamada en el programa principal es, de esta manera:

El programa construye las estructuras requeridas usando nodos definidos como:

```
nodo = RECORD
        carac: ARRAY[indcarac] OF INTEGER;
        CASE tipon:tiponodo OF
                nter:( papel: ARRAY[indpapel] OF apunodo;
                       tnodo: tipored );
                ter:  ( ll: indxl;
                        cl: numid );
                liga: ( sigl, sigm: apunodo )
       END;
```

donde: *carac* es un arreglo que contiene los valores de las características asociadas al nodo,

papel es un arreglo que contiene los apuntadores a los papeles asociados a un nodo no terminal,

tnodo indica el grupo asociado a un nodo no terminal,

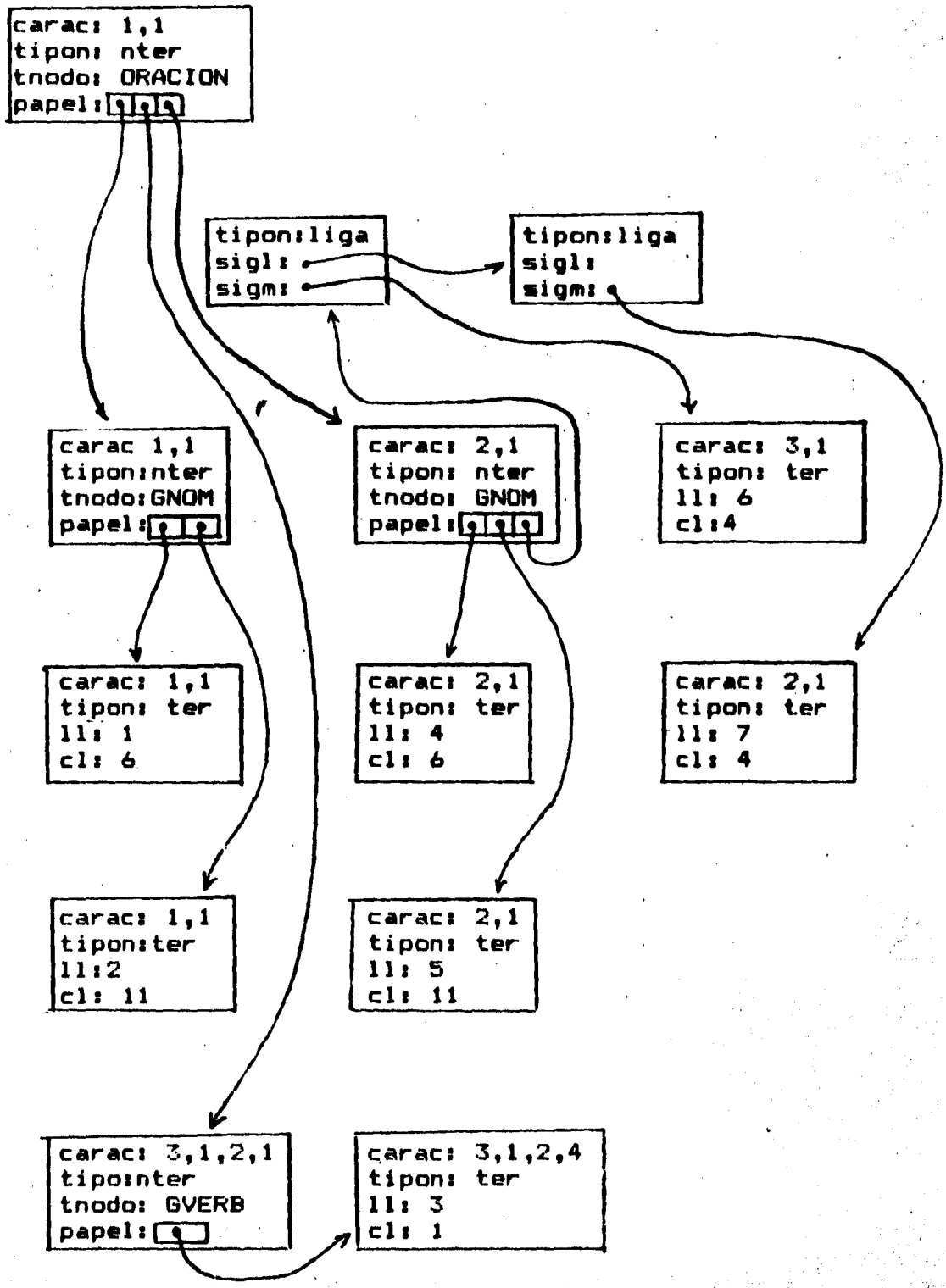
ll indica el número de palabra que forma a un nodo terminal,

cl almacena el número de la categoría léxica asociada a la papalbra que forma un nodo terminal,

liga se forma con dos apuntadores usados para formar estructuras para acciones de concatenación.

Como ilustración de las estructuras generadas, véase:

"El conejo aserró la naranja grande bonita"



donde, las características son:
voz de ORACION,
género, número, de GNOM, sustantivos, adjetivos y artículos
persona, número, tiempo, tipo, de GVERB y verbo

los valores de ellas:
voz = 1 activa

genero = 1 masculino
= 2 femenino
= 3 neutro

nmero = 1 singular
= 2 plural

tiempo = 2 pasado
tipo de verbo = 1 común

y las características léxicas son:
1: verbo, 4: adjetivo, 6: artículo definido, y 11: sustantivo.

Para administrar el recorrido de las alternativas de búsqueda, *rur* usa un stack *s*, de donde recupera los parámetros necesarios para ejecutar cada paso. *s* está apuntado por dos variables: *sp* apunta al tope absoluto del stack, y *tp* apunta a la base del stack relativa al nivel de llamada. El formato de cada registro de *s* es:
...

```
stackres = RECORD
             pos: indx1;
             arc: indxc;
             at: apunodo
           END;
```

...
donde: *pos* es la primera palabra a usar en el análisis (i.e., la siguiente de última usada)

arc indica el siguiente arco a intentar resolver en el análisis

at es un apuntador a la estructura generada hasta el momento por la trayectoria seguida

Conforme el procedimiento *rur* logra reconocer subcadenas de la oración de entrada de acuerdo a la red pedida, almacena

cada resultado en un stack auxiliar *saux*. Al momento de terminar la ejecución de la red, el programa incorpora los resultados parciales de *saux* al stack principal *s*.

Para ejemplificar lo antes mencionado, véase:

Supóngase que se tiene la red siguiente:

no arco:

```
 1   ORACION: A->B,BUSCA GNOM
 2            B->C,CAT   verbo
 3            C->D,BUSCA GNOM
 4            C->D,BRINCA
 5            D    MANDA
 6   GNOM:    E->F,CAT   artículo
 7            E->F,BRINCA
 8            F->G,CAT   sustantivo
 9            G->G,CAT   adjetivo
10            G    MANDA
```

y que se busca analizar la oración:

"El conejo aserró la naranja naranja"

```
  A   NUMERO  APUNTA A
CAZAR  ARCO
```

-- al inicio, la situación del stack es

```
tp,sp  1     1     ...
```

-- al resolver el primer arco de ORACION, carga al stack la dirección inicial de GNOM, preservando la marca de ORACION:

```
       1     1     ...
tp,sp  1     6     ...
```

-- Después de realizar los recorridos de A a B, y B a C, hay dos trayectorias posibles, que deben explorarse en paralelo: cazar un GNOM -arco 3-, o uno de BRINCA -arco 4-

```
tp     4     4     El conejo aserro
sp     4     3     El conejo aserro
```

-- Al intentar resolver el arco del tope, el 3, se encuentra otro ramal en la trayectoria, en el nodo G:

```
        4      4      El conejo aserro
        4      3      El conejo aserro
tp      6      9      la naranja
sp      6     10      la naranja
```

-- La primera alternativa tiene éxito, por lo que se almacena en el stack auxiliar *saux*, y se intenta la segunda:

```
        4      4      El conejo aserro
        4      3      El conejo aserro
tp,sp   6      9      la naranja
```

-- La segunda sigue el análisis, y encuentra otra posibilidad para el GNOM:

```
        4      4      El conejo aserro
        4      3      El conejo aserro
tp,sp   7     10      la naranja naranja
```

-- Esta trayectoria también tiene éxito, por lo que se pasa a *saux*. Al salir de la ejecución de GNOM (marcado en el stack por *tp*), se incorporan los resultados al nivel superior. Como son dos resultados, el tope se duplica:

```
tp      4      4      El conejo aserro
        6      5      El conejo aserro la naranja
sp      7      5      El conejo aserro la naranja naranja
```

-- El arco 5 es el de MANDA los análisis de la red, por lo que los dos elementos del tope se entregan como resultado (uno de ellos incompleto), y queda sólo por resolver:

```
tp,sp   4      4      El conejo aserro
```

que, después de resolver el arco 4 (BRINCA), entrega otro resultado incompleto.

V.3 Desempeño del programa

Se da un ejemplo que ilustra el funcionamiento del sistema. El ejemplo consiste en tomar la definición completa de

la red más sencilla del español desarrollada en la primera parte del capítulo anterior.

Este ejemplo pretende dar una idea precisa del desempeño del sistema, aunque una evaluación más completa de sus capacidades sólo podrá hacerse después de que éste sea usado como herramienta de soporte de un sistema que se apoye fuertemente en el uso del español como lenguaje de comunicación entre el usuario y la máquina.

Ejemplo: La red

```
ORACION: A->B,BUSCA   GNOM
ACC:    PAPEL  0001 NODO  0000
          B->C,BUSCA   GVERB
COND:   CARAC -0012 = NODO  0002
ACC:    PAPEL  0002 NODO  0000
          C->D,BRINCA
          C->D,BUSCA   GNOM
COND:   CARAC -0024 = CONS  0002
ACC:    PAPEL  0004 NODO  0000
          C->D,CAT     04
COND:   CARAC -0024 = CONS  0001  Y
        CARAC -0227 = CONS  0004  Y
        CARAC -0011 = NODO  0001  Y
        CARAC -0012 = NODO  0002
ACC:    PAPEL  0003 NODO  0000
          C->D,CAT     04
COND:   CARAC -0024 = CONS  0001  Y
        CARAC -0227 = CONS  0004  Y
        CARAC  0001 = CONS  0003  Y
        CARAC -0012 = NODO  0002
ACC:    PAPEL  0003 NODO  0000
          C->D,BUSCA   GNOM
COND:   CARAC -0024 = CONS  0001
ACC:    PAPEL  0003 NODO  0000
          D->D,BUSCA   GPREP
COND:   PAPEL  0001 = CONS 'A                 ' Y
        PAPEL -0004 = CONS -0001
ACC:    PAPEL  0004 NODO  0002
          D->D,BUSCA   GPREP
COND:   PAPEL  0001 = CONS 'A                 ' Y
        PAPEL -0004 # CONS -0001  Y
        PAPEL -0005 = CONS -0001
ACC:    PAPEL  0005 NODO  0002
          D->D,BUSCA   GPREP
COND:   PAPEL  0001 # CONS 'A                 '
ACC:    CONCT  0006 NODO  0000
          D->D,BUSCA   GPREP
COND:   PAPEL -0004 # CONS -0001  Y
        PAPEL -0005 # CONS -0001
ACC:    CONCT  0006 NODO  0000
          D->D,BUSCA   GNOM
COND:   PAPEL -0004 # CONS -0001
ACC:    CONCT  0006 NODO  0000
          D->D,BUSCA   GADJ
COND:   CARAC -0024 = CONS  0002
ACC:    CONCT  0006 NODO  0000
          D->D,BUSCA   GADJ .
COND:   CARAC -0024 = CONS  0001  Y
        CARAC -0227 = CONS  0002
ACC:    CONCT  0006 NODO  0000
          D    ,MANDA
GNOM:     O->P,BRINCA
          P->Q,BRINCA
COND:   CARAC -0004 # CONS  0004
```

```
          P->Q,CAT     05
ACC:   CONCT  0001 NODO  0000  Y
       CARAC  0004 CONS  0001  Y
       CARAC  0001 NODO  0001  Y
       CARAC  0002 NODO  0002
          P->Q,CAT     04
COND:  PAPEL  0000 = CONS 'OTRO          '
ACC:   CONCT  0001 NODO  0000  Y
       CARAC  0001 NODO  0001  Y
       CARAC  0002 NODO  0002  Y
       CARAC  0004 CONS  0001
          P->Q,CAT     04
COND:  PAPEL  0000 = CONS 'OTRA          '
ACC:   CONCT  0001 NODO  0000  Y
       CARAC  0001 NODO  0001  Y
       CARAC  0002 NODO  0002  Y
       CARAC  0004 CONS  0001
          P->Q,CAT     04
COND:  PAPEL  0000 = CONS 'OTROS         '
ACC:   CONCT  0001 NODO  0000  Y
       CARAC  0001 NODO  0001  Y
       CARAC  0002 NODO  0002  Y
       CARAC  0004 CONS  0001
          P->Q,CAT     04
COND:  PAPEL  0000 = CONS 'OTRAS         '
ACC:   CONCT  0001 NODO  0000  Y
       CARAC  0001 NODO  0001  Y
       CARAC  0002 NODO  0002  Y
       CARAC  0004 CONS  0001
          P->Q,CAT     06
ACC:   CONCT  0001 NODO  0000  Y
       CARAC  0004 CONS  0002  Y
       CARAC  0001 NODO  0001  Y
       CARAC  0002 NODO  0002
          P->Q,CAT     07
ACC:   CONCT  0001 NODO  0000  Y
       CARAC  0004 CONS  0002  Y
       CARAC  0001 NODO  0001  Y
       CARAC  0002 NODO  0002
          P->Q,CAT     08
ACC:   CONCT  0001 NODO  0000  Y
       CARAC  0004 CONS  0002  Y
       CARAC  0001 NODO  0001  Y
       CARAC  0002 NODO  0002
          P->P,CAT     04
COND:  PAPEL  0000 = CONS 'TODO          ' Y
       CARAC -0004 # CONS  0004
ACC:   CONCT  0001 NODO  0000  Y
       CARAC  0001 NODO  0001  Y
       CARAC  0002 NODO  0002  Y
       CARAC  0004 CONS  0004
          P->P,CAT     04
COND:  PAPEL  0000 = CONS 'TODA          ' Y
       CARAC -0004 # CONS  0004
ACC:   CONCT  0001 NODO  0000  Y
       CARAC  0001 NODO  0001  Y
       CARAC  0002 NODO  0002  Y
       CARAC  0004 CONS  0004
```

```
          P->P,CAT     04
COND:  PAPEL  0000 = CONS 'TODOS          ' Y
       CARAC -0004 # CONS  0004
ACC:   CONCT  0001 NODO  0000  Y
       CARAC  0001 NODO  0001  Y
       CARAC  0002 NODO  0002  Y
       CARAC  0004 CONS  0004
          P->P,CAT     04
COND:  PAPEL  0000 = CONS 'TODAS          ' Y
       CARAC -0004 # CONS  0004
ACC:   CONCT  0001 NODO  0000  Y
       CARAC  0001 NODO  0001  Y
       CARAC  0002 NODO  0002  Y
       CARAC  0004 CONS  0004
          P->Q,CAT     03
COND:  CARAC  0005 = CONS  0001
ACC:   CONCT  0001 NODO  0000  Y
       CARAC  0004 CONS  0003
          P->Q,CAT     04
COND:  CARAC  0005 = CONS  0001
ACC:   CONCT  0001 NODO  0000  Y
       CARAC  0004 CONS  0003  Y
       CARAC  0001 NODO  0001  Y
       CARAC  0002 NODO  0002
          P->Q,CAT     09
ACC:   CONCT  0001 NODO  0000  Y
       CARAC  0004 CONS  0003  Y
       CARAC  0001 NODO  0001  Y
       CARAC  0002 NODO  0002
          Q->R,CAT     09
COND:  CARAC -0004 = CONS  0001  Y
       CARAC -0002 = CONS  0002
ACC:   CONCT  0001 NODO  0000
          Q->R,CAT     10
COND:  CARAC -0004 = CONS  0001  Y
       CARAC -0001 = CONS  0000  Y
       CARAC -0002 = CONS  0000
ACC:   CONCT  0001 NODO  0000  Y
       CARAC  0001 NODO  0001  Y
       CARAC  0002 NODO  0002
          Q->R,CAT     10
COND:  CARAC -0004 = CONS  0001  Y
       CARAC -0001 = CONS  0003  Y
       CARAC -0002 = NODO  0002
ACC:   CONCT  0001 NODO  0000  Y
       CARAC  0001 NODO  0001
          Q->R,CAT     10
COND:  CARAC -0004 = CONS  0001  Y
       CARAC -0001 = NODO  0001  Y
       CARAC -0002 = NODO  0002
ACC:   CONCT  0001 NODO  0000
          Q->R,CAT     09
COND:  CARAC -0004 = CONS  0002  Y
       CARAC -0002 = CONS  0002
ACC:   CONCT  0001 NODO  0000
```

```
          Q->R,CAT      10
COND:   CARAC -0004 = CONS   0002   Y
        CARAC -0001 = CONS   0000   Y
        CARAC -0002 = CONS   0000
ACC:    CONCT   0001 NODO   0000   Y
        CARAC   0001 NODO   0001   Y
        CARAC   0002 NODO   0002
          Q->R,CAT      10
COND:   CARAC -0004 = CONS   0002   Y
        CARAC -0001 = CONS   0003   Y
        CARAC -0002 = NODO   0002
ACC:    CONCT   0001 NODO   0000   Y
        CARAC   0001 NODO   0001
          Q->R,CAT      10
COND:   CARAC -0004 = CONS   0002   Y
        CARAC -0001 = NODO   0001   Y
        CARAC -0002 = NODO   0002
ACC:    CONCT   0001 NODO   0000
          Q->R,CAT      04
COND:   PAPEL   0000 = CONS 'OTRO              ' Y
        CARAC -0004 # CONS   0001   Y
        CARAC -0004 # CONS   0000
ACC:    CONCT   0001 NODO   0000
          Q->R,CAT      04
COND:   PAPEL   0000 = CONS 'OTRA              ' Y
        CARAC -0004 # CONS   0001   Y
        CARAC -0004 # CONS   0000
ACC:    CONCT   0001 NODO   0000
          Q->R,CAT      04
COND:   PAPEL   0000 = CONS 'OTROS             ' Y
        CARAC -0004 # CONS   0001   Y
        CARAC -0004 # CONS   0000
ACC:    CONCT   0001 NODO   0000
          Q->R,CAT      04
COND:   PAPEL   0000 = CONS 'OTRAS             ' Y
        CARAC -0004 # CONS   0001   Y
        CARAC -0004 # CONS   0000
ACC:    CONCT   0001 NODO   0000
          Q->R,BRINCA
          R->S,CAT      11
COND:   CARAC -0001 = CONS   0000   Y
        CARAC -0002 = CONS   0000
ACC:    PAPEL   0002 NODO   0000   Y
        CARAC   0001 NODO   0001   Y
        CARAC   0002 NODO   0002   Y
        CARAC   0005 CONS   0003
          R->S,CAT      11
COND:   CARAC -0001 = CONS   0003   Y
        CARAC -0002 = NODO   0002
ACC:    PAPEL   0002 NODO   0000   Y
        CARAC   0002 NODO   0002   Y
        CARAC   0005 CONS   0003
          R->S,CAT      11
COND:   CARAC -0001 = NODO   0001   Y
        CARAC -0002 = NODO   0002
ACC:    PAPEL   0002 NODO   0000   Y
        CARAC   0005 CONS   0003
```

```
                    'EX      'DE
ACC:    CONCT   0001 NODO -0002  Y
        CONCT   0001 NODO  0000  Y
        PAPEL   0002 CONS -0001  Y
        CARAC   0001 CONS  0000  Y
        CARAC   0002 CONS  0000  Y
        CARAC   0004 CONS  0000
          S    ,MANDA
GVERB:    K->L,BRINCA
          L->L,CAT      01
COND:   CARAC   0007 = CONS  0003  Y
        CARAC  -0005 # CONS  0001
ACC:    CONCT   0001 NODO  0000  Y
        CARAC   0005 CONS  0001  Y
        CARAC   0003 NODO  0003  Y
        CARAC   0002 NODO  0002  Y
        CARAC   0001 NODO  0001
          L->L,CAT      01
COND:   CARAC   0007 = CONS  0002  Y
        CARAC  -0005 # CONS  0001  Y
        CARAC  -0006 # CONS  0001
ACC:    CONCT   0001 NODO  0000  Y
        CARAC   0006 CONS  0001  Y
        CARAC   0003 NODO  0003  Y
        CARAC   0002 NODO  0002  Y
        CARAC   0001 NODO  0001
          L->L,CAT      01
COND:   CARAC   0007 = CONS  0002  Y
        CARAC  -0005 = CONS  0001  Y
        CARAC  -0006 # CONS  0001
ACC:    CONCT   0001 NODO  0000  Y
        CARAC   0006 CONS  0001
          L->M,CAT      01
COND:   CARAC  -0005 # CONS  0001  Y
        CARAC  -0006 # CONS  0001  Y
        CARAC   0005 = CONS  0002  Y
        CARAC   0006 = CONS  0002
ACC:    PAPEL   0002 NODO  0000  Y
        CARAC   0003 NODO  0003  Y
        CARAC   0004 NODO  0004  Y
        CARAC   0002 NODO  0002  Y
        CARAC   0001 NODO  0001
          L->M,CAT      01
COND:   CARAC  -0005 = CONS  0001  Y
        CARAC   0005 = CONS  0001
ACC:    PAPEL   0002 NODO  0000  Y
        CARAC   0005 CONS  0001
          L->M,CAT      01
COND:   CARAC  -0006 = CONS  0001  Y
        CARAC   0006 = CONS  0001
ACC:    PAPEL   0002 NODO  0000  Y
        CARAC   0004 NODO  0004
          M    ,MANDA
GPREP:    H->I,CAT      02
ACC:    PAPEL   0001 NODO  0000
          I->J,BUSCA    GNOM
ACC:    PAPEL   0002 NODO  0000
          J    ,MANDA
GADJ:     U->V,BRINCA
          V->W,CAT      03
ACC:    PAPEL   0001 NODO  0000
          V->W,CAT      04
ACC:    PAPEL   0001 NODO  0000
          W    ,MANDA
```

Ejemplo: el diccionario ('DICC.DAT')

EL DICCIONARIO CONTIENE:

1	ident: A	cat lex:	2	atributos:	0	0	0	0	0	0	0	0
2	ident: AHI	cat lex:	3	atributos:	0	0	0	3	0	0	0	0
3	ident: AQUEL	cat lex:	7	atributos:	1	1	3	0	0	0	0	0
4	ident: AQUELLA	cat lex:	7	atributos:	2	1	3	0	0	0	0	0
5	ident: AQUELLAS	cat lex:	7	atributos:	2	2	3	0	0	0	0	0
6	ident: AQUELLOS	cat lex:	7	atributos:	1	2	3	0	0	0	0	0
7	ident: ARRIBA	cat lex:	3	atributos:	0	0	0	3	0	0	0	0
8	ident: ASERRO	cat lex:	1	atributos:	3	1	2	2	2	2	1	0
9	ident: CHOMSKY	cat lex:	11	atributos:	1	1	2	0	0	0	0	0
10	ident: COME	cat lex:	1	atributos:	3	1	1	2	2	2	1	0
11	ident: CON	cat lex:	2	atributos:	0	0	0	0	0	0	0	0
12	ident: CONEJA	cat lex:	11	atributos:	2	1	1	0	0	0	0	0
13	ident: CONEJAS	cat lex:	11	atributos:	2	2	1	0	0	0	0	0
14	ident: CONEJO	cat lex:	11	atributos:	1	1	1	0	0	0	0	0
15	ident: CONEJOS	cat lex:	11	atributos:	1	2	1	0	0	0	0	0
16	ident: DE	cat lex:	2	atributos:	0	0	0	0	0	0	0	0
17	ident: DEUDA	cat lex:	11	atributos:	2	1	1	0	0	0	0	0
18	ident: DOS	cat lex:	9	atributos:	2	2	0	0	0	0	0	0
19	ident: DOS	cat lex:	9	atributos:	1	2	0	0	0	0	0	0
20	ident: EL	cat lex:	8	atributos:	1	1	0	0	0	0	0	0
21	ident: EN	cat lex:	2	atributos:	0	0	0	0	0	0	0	0
22	ident: ES	cat lex:	1	atributos:	3	1	1	1	2	2	4	0
23	ident: ESA	cat lex:	7	atributos:	2	1	2	0	0	0	0	0
24	ident: ESAS	cat lex:	7	atributos:	2	2	2	0	0	0	0	0
25	ident: ESE	cat lex:	7	atributos:	1	1	2	0	0	0	0	0
26	ident: ESOS	cat lex:	7	atributos:	1	2	2	0	0	0	0	0
27	ident: ESTA	cat lex:	1	atributos:	3	1	1	1	2	2	2	0
28	ident: ESTA	cat lex:	2	atributos:	1	1	0	0	0	0	0	0
29	ident: ESTABA	cat lex:	1	atributos:	3	1	4	1	2	2	2	0
30	ident: ESTADO	cat lex:	1	atributos:	0	0	0	1	1	2	2	0
31	ident: ESTAS	cat lex:	7	atributos:	2	2	1	0	0	0	0	0
32	ident: ESTE	cat lex:	7	atributos:	1	1	1	0	0	0	0	0
33	ident: ESTOS	cat lex:	7	atributos:	1	2	1	0	0	0	0	0
34	ident: GUSTO	cat lex:	11	atributos:	1	1	1	0	0	0	0	0
35	ident: HA	cat lex:	1	atributos:	3	1	1	2	2	2	3	0
36	ident: HABLADO	cat lex:	1	atributos:	0	0	0	2	1	2	1	0
37	ident: HABLANDO	cat lex:	1	atributos:	0	0	0	2	2	1	1	0
38	ident: HABLARIA	cat lex:	1	atributos:	3	1	5	2	2	2	1	0
39	ident: HABRA	cat lex:	1	atributos:	3	1	3	2	2	2	3	0
40	ident: INTELIGENTE	cat lex:	4	atributos:	3	1	0	0	0	0	0	0
41	ident: INTELIGENTES	cat lex:	4	atributos:	3	2	0	0	0	0	0	0
42	ident: LA	cat lex:	8	atributos:	2	1	0	0	0	0	0	0
43	ident: LAS	cat lex:	8	atributos:	2	2	0	0	0	0	0	0
44	ident: LECHE	cat lex:	11	atributos:	2	1	1	0	0	0	0	0
45	ident: LITRO	cat lex:	11	atributos:	1	1	1	0	0	0	0	0

46	ident: LITROS	cat lex:	11	atributos:	1	2	1	0	0	0	0	(
47	ident: LOS	cat lex:	8	atributos:	1	2	0	0	0	0	0	(
48	ident: MAS	cat lex:	3	atributos:	0	0	0	1	1	0	0	(
49	ident: MENOS	cat lex:	3	atributos:	0	0	0	1	1	0	0	(
50	ident: MIDE	cat lex:	1	atributos:	3	1	1	2	2	2	1	(
51	ident: MUCHO	cat lex:	3	atributos:	0	0	0	2	0	0	0	(
52	ident: MUCHO	cat lex:	4	atributos:	3	1	0	0	1	0	0	(
53	ident: MUCHOS	cat lex:	4	atributos:	3	2	0	0	1	0	0	(
54	ident: NARANJA	cat lex:	4	atributos:	3	1	0	0	0	0	0	(
55	ident: NARANJA	cat lex:	11	atributos:	2	1	1	0	0	0	0	(
56	ident: NARANJAS	cat lex:	4	atributos:	3	2	0	0	0	0	0	(
57	ident: NARANJAS	cat lex:	11	atributos:	2	2	1	0	0	0	0	(
58	ident: OTRA	cat lex:	4	atributos:	2	1	0	0	0	0	0	(
59	ident: OTRAS	cat lex:	4	atributos:	2	2	0	0	0	0	0	(
60	ident: OTRO	cat lex:	4	atributos:	1	1	0	0	0	0	0	(
61	ident: OTROS	cat lex:	4	atributos:	1	2	0	0	0	0	0	(
62	ident: PAGO	cat lex:	1	atributos:	3	1	2	2	2	2	1	(
63	ident: PIENSA	cat lex:	1	atributos:	3	1	1	2	2	2	1	(
64	ident: POCA	cat lex:	4	atributos:	2	1	0	0	1	0	0	(
65	ident: POCAS	cat lex:	4	atributos:	2	2	0	0	1	0	0	(
66	ident: POCO	cat lex:	4	atributos:	1	1	0	0	1	0	0	(
67	ident: POCOS	cat lex:	4	atributos:	1	2	0	0	1	0	0	(
68	ident: SEGUNDA	cat lex:	10	atributos:	2	1	0	0	0	0	0	(
69	ident: SEGUNDAS	cat lex:	10	atributos:	2	2	0	0	0	0	0	(
70	ident: SEGUNDO	cat lex:	10	atributos:	1	1	0	0	0	0	0	(
71	ident: SEGUNDOS	cat lex:	10	atributos:	1	2	0	0	0	0	0	(
72	ident: SEMANA	cat lex:	11	atributos:	2	1	1	0	0	0	0	(
73	ident: SEMANAS	cat lex:	11	atributos:	2	2	1	0	0	0	0	(
74	ident: SEMEJANTE	cat lex:	4	atributos:	3	1	0	0	1	0	0	(
75	ident: SEMEJANTES	cat lex:	4	atributos:	3	2	0	0	1	0	0	(
76	ident: SON	cat lex:	1	atributos:	3	2	1	1	2	2	4	(
77	ident: TODA	cat lex:	4	atributos:	2	1	0	0	1	0	0	(
78	ident: TODAS	cat lex:	4	atributos:	2	2	0	0	1	0	0	(
79	ident: TODO	cat lex:	4	atributos:	1	1	0	0	1	0	0	(
80	ident: TODOS	cat lex:	4	atributos:	1	2	0	0	1	0	0	(
81	ident: TRES	cat lex:	9	atributos:	0	0	0	0	1	0	0	(
82	ident: UN	cat lex:	5	atributos:	1	1	0	0	0	0	0	(
83	ident: UNA	cat lex:	5	atributos:	2	1	0	0	0	0	0	(
84	ident: UNAS	cat lex:	5	atributos:	2	2	0	0	0	0	0	(
85	ident: UNOS	cat lex:	5	atributos:	1	2	0	0	0	0	0	(
86	ident: ZANAHORIA	cat lex:	11	atributos:	2	1	1	0	0	0	0	(
87	ident: ZANAHORIAS	cat lex:	11	atributos:	2	2	1	0	0	0	0	(

Ejemplo: el diccionario de identificadores
('IDENTIF.DAT')

EL ARCHIVO DE IDENTIFICADORES CONTIENE:

```
  1:     llave:  1  0  0 -->ORACION        papel
  2:     llave:  1  1  0 -->SUJETO         papel
  3:     llave:  1  2  0 -->ACCION         papel
  4:     llave:  1  3  0 -->PREDNOM        papel
  5:     llave:  1  4  0 -->OBJETO-DIR     papel
  6:     llave:  1  5  0 -->OBJETO-IND     papel
  7:     llave:  1  6  0 -->MODIF-ORAC     papel
  8:     llave:  2  0  0 -->GP-NOMINAL     papel
  9:     llave:  2  1  0 -->CABEZA-GN      papel
 10:     llave:  2  2  0 -->CUERPO-GN      papel
 11:     llave:  3  0  0 -->GP-VERBAL      papel
 12:     llave:  3  1  0 -->VERBO-AUX      papel
 13:     llave:  3  2  0 -->VERBO-P        papel
 14:     llave:  4  0  0 -->GP-PREP        papel
 15:     llave:  4  1  0 -->PREP-GP        papel
 16:     llave:  4  2  0 -->OBJETO-GP      papel
 17:     llave:  5  0  0 -->GP-ADJETIV     papel
 18:     llave:  5  1  0 -->MODIF          papel
 19:     llave:  2  1  0 -->GENERO         caracteristica
 20:     llave:  2  2  0 -->NUMERO         caracteristica
 21:     llave:  2  3  0 -->TIPO-NOM       caracteristica
 22:     llave:  2  4  0 -->TIPO-DET       caracteristica
 23:     llave:  2  5  0 -->PERSONA        caracteristica
 24:     llave:  3  1  0 -->PERSONA        caracteristica
 25:     llave:  3  2  0 -->NUMERO         caracteristica
 26:     llave:  3  3  0 -->TIEMPO         caracteristica
 27:     llave:  3  4  0 -->VERBO-COP      caracteristica
 28:     llave:  3  5  0 -->PERFECTO       caracteristica
 29:     llave:  3  6  0 -->PROGRESIVO     caracteristica
 30:     llave:  5  1  0 -->TIPO-GA        caracteristica
 31:     llave:  2  1  1 -->MASCULINO      valor de caracteristica
 32:     llave:  2  1  2 -->FEMENINO       valor de caracteristica
 33:     llave:  2  2  1 -->SINGULAR       valor de caracteristica
 34:     llave:  2  2  2 -->PLURAL         valor de caracteristica
 35:     llave:  2  3  1 -->COMUN          valor de caracteristica
 36:     llave:  2  3  2 -->PROPIO         valor de caracteristica
 37:     llave:  2  4  1 -->INDEF          valor de caracteristica
 38:     llave:  2  4  1 -->INDEF          valor de caracteristica
 39:     llave:  2  4  2 -->DEF            valor de caracteristica
 40:     llave:  2  4  3 -->CALIF          valor de caracteristica
 41:     llave:  2  4  4 -->DET-FRAC       valor de caracteristica
 42:     llave:  2  5  1 -->PRIMERA        valor de caracteristica
 43:     llave:  2  5  2 -->SEGUNDA        valor de caracteristica
 44:     llave:  2  5  3 -->TERCERA        valor de caracteristica
 45:     llave:  3  1  1 -->PRIMERA        valor de caracteristica
 46:     llave:  3  1  2 -->SEGUNDA        valor de caracteristica
 47:     llave:  3  1  3 -->TERCERA        valor de caracteristica
 48:     llave:  3  2  1 -->SINGULAR       valor de caracteristica
 49:     llave:  3  2  2 -->PLURAL         valor de caracteristica
 50:     llave:  3  3  1 -->PRESENTE       valor de caracteristica
 51:     llave:  3  3  2 -->PASADO         valor de caracteristica
 52:     llave:  3  3  3 -->FUTURO         valor de caracteristica
 53:     llave:  3  3  4 -->IMPERFECTO     valor de caracteristica
 54:     llave:  3  3  5 -->CONDICIONA     valor de caracteristica
 55:     llave:  3  4  1 -->SI             valor de caracteristica
```

```
 56:    llave:   3   4   2 -->NO          valor de caracteristica
 57:    llave:   3   5   1 -->SI          valor de caracteristica
 58:    llave:   3   5   2 -->NO          valor de caracteristica
 59:    llave:   3   6   1 -->SI          valor de caracteristica
 60:    llave:   3   6   2 -->NO          valor de caracteristica
 61:    llave:   1   0   0 -->VERBO       categoria lexica
 62:    llave:   2   0   0 -->PREPOSICIO  categoria lexica
 63:    llave:   3   0   0 -->ADVERBIO    categoria lexica
 64:    llave:   4   0   0 -->ADJETIVO    categoria lexica
 65:    llave:   5   0   0 -->ATR-INDEF   categoria lexica
 66:    llave:   6   0   0 -->POSESIVO    categoria lexica
 67:    llave:   7   0   0 -->DEMOSTRATI  categoria lexica
 68:    llave:   8   0   0 -->ART-DEF     categoria lexica
 69:    llave:   9   0   0 -->CARDINAL    categoria lexica
 70:    llave:  10   0   0 -->ORDINAL     categoria lexica
 71:    llave:  11   0   0 -->SUSTANTIVO  categoria lexica
 72:    llave:   1   1   0 -->PERSONA     propiedad de cat lex
 73:    llave:   1   2   0 -->NUMERO      propiedad de cat lex
 74:    llave:   1   3   0 -->TIEMPO      propiedad de cat lex
 75:    llave:   1   4   0 -->VERBO-COP   propiedad de cat lex
 76:    llave:   1   5   0 -->PREFECTO    propiedad de cat lex
 77:    llave:   1   6   0 -->PROGRESIVO  propiedad de cat lex
 78:    llave:   1   7   0 -->TIPO        propiedad de cat lex
 79:    llave:   3   4   0 -->TIPO        propiedad de cat lex
 80:    llave:   3   5   0 -->CALIF\      propiedad de cat lex
 81:    llave:   4   1   0 -->GENERO      propiedad de cat lex
 82:    llave:   4   2   0 -->NUMERO      propiedad de cat lex
 83:    llave:   4   5   0 -->CALIF       propiedad de cat lex
 84:    llave:   5   1   0 -->GENERO      propiedad de cat lex
 85:    llave:   5   2   0 -->NUMERO      propiedad de cat lex
 86:    llave:   6   1   0 -->GENERO      propiedad de cat lex
 87:    llave:   6   2   0 -->NUMERO      propiedad de cat lex
 88:    llave:   7   1   0 -->GENERO      propiedad de cat lex
 89:    llave:   7   2   0 -->NUMERO      propiedad de cat lex
 90:    llave:   7   3   0 -->SITUACION   propiedad de cat lex
 91:    llave:   8   1   0 -->GENERO      propiedad de cat lex
 92:    llave:   8   2   0 -->NUMERO      propiedad de cat lex
 93:    llave:   9   5   0 -->CALIF       propiedad de cat lex
 94:    llave:  10   1   0 -->GENERO      propiedad de cat lex
 95:    llave:  10   2   0 -->NUMERO      propiedad de cat lex
 96:    llave:  11   1   0 -->GENERO      propiedad de cat lex
 97:    llave:  11   2   0 -->NUMERO      propiedad de cat lex
 98:    llave:  11   3   0 -->TIPO-NOM    propiedad de cat lex
 99:    llave:   1   1   1 -->PRIMERA     valor de cat lex
100:    llave:   1   1   2 -->SEGUNDA     valor de cat lex
101:    llave:   1   1   3 -->TERCERA     valor de cat lex
102:    llave:   1   2   1 -->SINGULAR    valor de cat lex
103:    llave:   1   2   2 -->PLURAL      valor de cat lex
104:    llave:   1   3   1 -->PRESENTE    valor de cat lex
105:    llave:   1   3   2 -->PASADO      valor de cat lex
106:    llave:   1   3   3 -->FUTURO      valor de cat lex
107:    llave:   1   3   4 -->IMPERFECTO  valor de cat lex
108:    llave:   1   3   5 -->CONDICIONA  valor de cat lex
109:    llave:   1   4   1 -->SI          valor de cat lex
110:    llave:   1   4   2 -->NO          valor de cat lex
```

```
111:    llave:  1   5   1 -->SI          valor de cat lex
112:    llave:  1   5   2 -->NO          valor de cat lex
113:    llave:  1   6   1 -->SI          valor de cat lex
114:    llave:  1   6   2 -->NO          valor de cat lex
115:    llave:  1   7   1 -->COMUN       valor de cat lex
116:    llave:  1   7   2 -->ESTAR       valor de cat lex
117:    llave:  1   7   3 -->HABER       valor de cat lex
118:    llave:  1   7   4 -->SER         valor de cat lex
119:    llave:  3   4   1 -->COMPARAC    valor de cat lex
120:    llave:  3   4   2 -->CANTIDAD    valor de cat lex
121:    llave:  3   4   3 -->LUGAR       valor de cat lex
122:    llave:  3   5   1 -->SI          valor de cat lex
123:    llave:  4   1   1 -->MASCULINO   valor de cat lex
124:    llave:  4   1   2 -->FEMENINO    valor de cat lex
125:    llave:  4   1   3 -->NEUTRO      valor de cat lex
126:    llave:  4   2   1 -->SINGULAR    valor de cat lex
127:    llave:  4   2   2 -->PLURAL      valor de cat lex
128:    llave:  4   5   1 -->SI          valor de cat lex
129:    llave:  5   1   1 -->MASCULINO   valor de cat lex
130:    llave:  5   1   2 -->FEMENINO    valor de cat lex
131:    llave:  5   2   1 -->SINGULAR    valor de cat lex
132:    llave:  5   2   2 -->PLURAL      valor de cat lex
133:    llave:  6   1   1 -->MASCULINO   valor de cat lex
134:    llave:  6   1   2 -->FEMENINO    valor de cat lex
135:    llave:  6   2   1 -->SINGULAR    valor de cat lex
136:    llave:  6   2   2 -->PLURAL      valor de cat lex
137:    llave:  7   1   1 -->MASCULINO   valor de cat lex
138:    llave:  7   1   2 -->FEMENINO    valor de cat lex
139:    llave:  7   2   1 -->SINGULAR    valor de cat lex
140:    llave:  7   2   2 -->PLURAL      valor de cat lex
141:    llave:  7   3   1 -->PROXIMO     valor de cat lex
142:    llave:  7   3   2 -->MEDIO       valor de cat lex
143:    llave:  7   3   3 -->DISTANTE    valor de cat lex
144:    llave:  8   1   1 -->MASCULINO   valor de cat lex
145:    llave:  8   1   2 -->FEMENINO    valor de cat lex
146:    llave:  8   2   1 -->SINGULAR    valor de cat lex
147:    llave:  8   2   2 -->PLURAL      valor de cat lex
148:    llave:  9   5   1 -->SI          valor de cat lex
149:    llave: 10   1   1 -->MASCULINO   valor de cat lex
150:    llave: 10   1   2 -->FEMENINO    valor de cat lex
151:    llave: 10   2   1 -->SINGULAR    valor de cat lex
152:    llave: 10   2   2 -->PLURAL      valor de cat lex
153:    llave: 11   1   1 -->MASCULINO   valor de cat lex
154:    llave: 11   1   2 -->FEMENINO    valor de cat lex
155:    llave: 11   2   1 -->SINGULAR    valor de cat lex
156:    llave: 11   2   2 -->PLURAL      valor de cat lex
157:    llave: 11   3   1 -->COMUN       valor de cat lex
158:    llave: 11   3   2 -->PROPIO      valor de cat lex
```

Ejemplo: análisis entregados por el intérprete

```
>
>RUN ATN
GVERB
HA HABLADO
.....................................................
HA HABLADO
.....................................................
IDENTIFIQUE LA CADENA DE ENTRADA


            GP-VERBAL : HA  HABLADO
            PERSONA   : TERCERA
            NUMERO    : SINGULAR
            TIEMPO    : PRESENTE
            PERFECTO  : SI
                  VERBO-AUX : VERBO      : HA
                              PERSONA    : TERCERA
                              NUMERO     : SINGULAR
                              TIEMPO     : PRESENTE
                              VERBO-COP  : NO
                              PREFECTO   : NO
                              PROGRESIVO : NO
                              TIPO       : HABER
                  VERBO-P   : VERBO      : HABLADO
                              VERBO-COP  : NO
                              PREFECTO   : SI
                              PROGRESIVO : NO
                              TIPO       : COMUN
>
```

```
>RUN ATN
GVERB
ESTABA HABLANDO
.....................................................
ESTABA HABLANDO
.....................................................
IDENTIFIQUE LA CADENA DE ENTRADA


          GP-VERBAL : ESTABA  HABLANDO
          PERSONA   : TERCERA
          NUMERO    : SINGULAR
          TIEMPO    : IMPERFECTO
          VERBO-COP : NO
          PROGRESIVO: SI
                VERBO-AUX : VERBO      : ESTABA
                            PERSONA    : TERCERA
                            NUMERO     : SINGULAR
                            TIEMPO     : IMPERFECTO
                            VERBO-COP  : SI
                            PREFECTO   : NO
                            PROGRESIVO : NO
                            TIPO       : ESTAR
                VERBO-P   : VERBO      : HABLANDO
                            VERBO-COP  : NO
                            PREFECTO   : NO
                            PROGRESIVO : SI
                            TIPO       : COMUN
>
```

```
>RUN ATN
GVERB
HABRA ESTADO HABLANDO
.....................................................
HABRA ESTADO HABLANDO
.....................................................
IDENTIFIQUE LA CADENA DE ENTRADA

          GP-VERBAL : HABRA  ESTADO  HABLANDO
          PERSONA   : TERCERA
          NUMERO    : SINGULAR
          TIEMPO    : FUTURO
          VERBO-COP : NO
          PERFECTO  : SI
          PROGRESIVO: SI
                VERBO-AUX : VERBO      : HABRA
                            PERSONA    : TERCERA
                            NUMERO     : SINGULAR
                            TIEMPO     : FUTURO
                            VERBO-COP  : NO
                            PREFECTO   : NO
                            PROGRESIVO : NO
                            TIPO       : HABER
                      VERBO-AUX : VERBO      : ESTADO
                            VERBO-COP  : SI
                            PREFECTO   : SI
                            PROGRESIVO : NO
                            TIPO       : ESTAR
                      VERBO-P   : VERBO      : HABLANDO
                            VERBO-COP  : NO
                            PREFECTO   : NO
                            PROGRESIVO : SI
                            TIPO       : COMUN
>
```

```
>run atn
GNOM
UNA SEGUNDA ZANAHORIA
.....................................................
UNA SEGUNDA ZANAHORIA
.....................................................
IDENTIFIQUE LA CADENA DE ENTRADA

              GP-NOMINAL: UNA  SEGUNDA  ZANAHORIA
              GENERO    : FEMENINO
              NUMERO    : SINGULAR
              TIPO-DET  : INDEF
              PERSONA   : TERCERA
                    CABEZA-GN : ATR-INDEF : UNA
                                GENERO    : FEMENINO
                                NUMERO    : SINGULAR
                    CABEZA-GN : ORDINAL   : SEGUNDA
                                GENERO    : FEMENINO
                                NUMERO    : SINGULAR
                    CUERPO-GN : SUSTANTIVO: ZANAHORIA
                                GENERO    : FEMENINO
                                NUMERO    : SINGULAR
                                TIPO-NOM  : COMUN
>
```

```
>
>RUN ATN
GNOM
OTROS TRES CONEJOS
.....
OTROS TRES CONEJOS
.....
IDENTIFIQUE LA CADENA DE ENTRADA


            GP-NOMINAL: OTROS  TRES  CONEJOS
            GENERO    : MASCULINO
            NUMERO    : PLURAL
            TIPO-DET  : INDEF
            PERSONA   : TERCERA
                  CABEZA-GN : ADJETIVO  : OTROS
                              GENERO    : MASCULINO
                              NUMERO    : PLURAL
                  CABEZA-GN : CARDINAL  : TRES
                              CALIF     : SI
                  CUERPO-GN : SUSTANTIVO: CONEJOS
                              GENERO    : MASCULINO
                              NUMERO    : PLURAL
                              TIPO-NOM  : COMUN
>
```

```
>
>
>
>
>RUN ATN
GNOM
LOS TRES CONEJOS
.....................................................
LOS TRES CONEJOS
.....................................................
IDENTIFIQUE LA CADENA DE ENTRADA


            GP-NOMINAL: LOS  TRES  CONEJOS
            GENERO    : MASCULINO
            NUMERO    : PLURAL
            TIPO-DET  : DEF
            PERSONA   : TERCERA
                  CABEZA-GN : ART-DEF   : LOS
                              GENERO    : MASCULINO
                              NUMERO    : PLURAL
                  CABEZA-GN : CARDINAL  : TRES
                              CALIF     : SI
                  CUERPO-GN : SUSTANTIVO: CONEJOS
                              GENERO    : MASCULINO
                              NUMERO    : PLURAL
                              TIPO-NOM  : COMUN
>
```

```
>RUN ATN
GNOM
TRES LITROS DE ESA OTRA LECHE
.....
TRES LITROS DE ESA OTRA LECHE
.....
IDENTIFIQUE LA CADENA DE ENTRADA

          GP-NOMINAL: TRES  LITROS  DE  ESA  OTRA  LECHE
          GENERO    : FEMENINO
          NUMERO    : SINGULAR
          TIPO-DET  : DEF
          PERSONA   : TERCERA
                CABEZA-GN : CARDINAL  : TRES
                            CALIF     : SI
                CABEZA-GN : SUSTANTIVO: LITROS
                            GENERO    : MASCULINO
                            NUMERO    : PLURAL
                            TIPO-NOM  : COMUN
                CABEZA-GN :           : DE
                CABEZA-GN : DEMOSTRATI: ESA
                            GENERO    : FEMENINO
                            NUMERO    : SINGULAR
                            SITUACION : MEDIO
                CABEZA-GN : ADJETIVO  : OTRA
                            GENERO    : FEMENINO
                            NUMERO    : SINGULAR
                CUERPO-GN : SUSTANTIVO: LECHE
                            GENERO    : FEMENINO
                            NUMERO    : SINGULAR
                            TIPO-NOM  : COMUN
>
>
```

```
>RUN ATN
GNOM
POCAS OTRAS ZANAHORIAS
.....................................................
POCAS OTRAS ZANAHORIAS
.....................................................
IDENTIFIQUE LA CADENA DE ENTRADA


              GP-NOMINAL: POCAS  OTRAS  ZANAHORIAS
              GENERO    : FEMENINO
              NUMERO    : PLURAL
              TIPO-DET  : CALIF
              PERSONA   : TERCERA
                    CABEZA-GN : ADJETIVO  : POCAS
                                GENERO    : FEMENINO
                                NUMERO    : PLURAL
                                CALIF     : SI
                    CABEZA-GN : ADJETIVO  : OTRAS
                                GENERO    : FEMENINO
                                NUMERO    : PLURAL
                    CUERPO-GN : SUSTANTIVO: ZANAHORIAS
                                GENERO    : FEMENINO
                                NUMERO    : PLURAL
                                TIPO-NOM  : COMUN
>
```

```
 RUN ATN
ORACION
EL CONEJO ASERRO LA NARANJA
.....................................................
EL CONEJO ASERRO LA NARANJA
.....................................................
IDENTIFIQUE LA CADENA DE ENTRADA


          ORACION    : EL  CONEJO ASERRO LA  NARANJA
                SUJETO     : EL  CONEJO
                GENERO     : MASCULINO
                NUMERO     : SINGULAR
                TIPO-DET   : DEF
                PERSONA    : TERCERA
                      CABEZA-GN : ART-DEF    : EL
                                  GENERO     : MASCULINO
                                  NUMERO     : SINGULAR
                      CUERPO-GN : SUSTANTIVO: CONEJO
                                  GENERO     : MASCULINO
                                  NUMERO     : SINGULAR
                                  TIPO-NOM   : COMUN
                ACCION     : ASERRO
                PERSONA    : TERCERA
                NUMERO     : SINGULAR
                TIEMPO     : PASADO
                VERBO-COP  : NO
                      VERBO-P    : VERBO     : ASERRO
                                   PERSONA    : TERCERA
                                   NUMERO     : SINGULAR
                                   TIEMPO     : PASADO
                                   VERBO-COP  : NO
                                   PREFECTO   : NO
                                   PROGRESIVO: NO
                                   TIPO       : COMUN
                OBJETO-DIR: LA  NARANJA
                GENERO     : FEMENINO
                NUMERO     : SINGULAR
                TIPO-DET   : DEF
                PERSONA    : TERCERA
                      CABEZA-GN : ART-DEF    : LA
                                  GENERO     : FEMENINO
                                  NUMERO     : SINGULAR
                      CUERPO-GN : SUSTANTIVO: NARANJA
                                  GENERO     : FEMENINO
                                  NUMERO     : SINGULAR
                                  TIPO-NOM   : COMUN
```

```
 RUN ATN
ORACION
CHOMSKY ES INTELIGENTE
.....
CHOMSKY ES INTELIGENTE
.....
IDENTIFIQUE LA CADENA DE ENTRADA

          ORACION   : CHOMSKY ES INTELIGENTE
                SUJETO    : CHOMSKY
                GENERO    : MASCULINO
                NUMERO    : SINGULAR
                PERSONA   : TERCERA
                      CUERPO-GN : SUSTANTIVO: CHOMSKY
                                  GENERO    : MASCULINO
                                  NUMERO    : SINGULAR
                                  TIPO-NOM  : PROPIO
                ACCION    : ES
                PERSONA   : TERCERA
                NUMERO    : SINGULAR
                TIEMPO    : PRESENTE
                VERBO-COP : SI
                      VERBO-P   : VERBO     : ES
                                  PERSONA   : TERCERA
                                  NUMERO    : SINGULAR
                                  TIEMPO    : PRESENTE
                                  VERBO-COP : SI
                                  PREFECTO  : NO
                                  PROGRESIVO: NO
                                  TIPO      : SER
                PREDNOM   : ADJETIVO  : INTELIGENTE
                            GENERO    : NEUTRO
                            NUMERO    : SINGULAR
```

```
 RUN ATN
ORACION
CHOMSKY PAGO A SEMEJANTE CONEJO
.....................................................
CHOMSKY PAGO A SEMEJANTE CONEJO
.....................................................
IDENTIFIQUE LA CADENA DE ENTRADA


          ORACION    : CHOMSKY PAGO SEMEJANTE  CONEJO
                SUJETO     : CHOMSKY
                GENERO     : MASCULINO
                NUMERO     : SINGULAR
                PERSONA    : TERCERA
                      CUERPO-GN : SUSTANTIVO: CHOMSKY
                                  GENERO     : MASCULINO
                                  NUMERO     : SINGULAR
                                  TIPO-NOM   : PROPIO
                ACCION     : PAGO
                PERSONA    : TERCERA
                NUMERO     : SINGULAR
                TIEMPO     : PASADO
                VERBO-COP  : NO
                      VERBO-P    : VERBO      : PAGO
                                   PERSONA    : TERCERA
                                   NUMERO     : SINGULAR
                                   TIEMPO     : PASADO
                                   VERBO-COP  : NO
                                   PREFECTO   : NO
                                   PROGRESIVO: NO
                                   TIPO       : COMUN
                OBJETO-DIR: SEMEJANTE  CONEJO
                GENERO     :
                NUMERO     : SINGULAR
                TIPO-DET   : CALIF
                PERSONA    : TERCERA
                      CABEZA-GN : ADJETIVO   : SEMEJANTE
                                  GENERO     : NEUTRO
                                  NUMERO     : SINGULAR
                                  CALIF      : SI
                      CUERPO-GN : SUSTANTIVO: CONEJO
                                  GENERO     : MASCULINO
                                  NUMERO     : SINGULAR
                                  TIPO-NOM   : COMUN
>
```

```
 RUN ATN
ORACION
CHOMSKY PAGO LA DEUDA A SEMEJANTE CONEJO
.....................................................
CHOMSKY PAGO LA DEUDA A SEMEJANTE CONEJO
.....................................................
IDENTIFIQUE LA CADENA DE ENTRADA


          ORACION   : CHOMSKY PAGO LA  DEUDA SEMEJANTE  CONEJO
               SUJETO    : CHOMSKY
               GENERO    : MASCULINO
               NUMERO    : SINGULAR
               PERSONA   : TERCERA
                    CUERPO-GN : SUSTANTIVO: CHOMSKY
                                GENERO    : MASCULINO
                                NUMERO    : SINGULAR
                                TIPO-NOM  : PROPIO
               ACCION    : PAGO
               PERSONA   : TERCERA
               NUMERO    : SINGULAR
               TIEMPO    : PASADO
               VERBO-COP : NO
                    VERBO-P   : VERBO     : PAGO
                                PERSONA   : TERCERA
                                NUMERO    : SINGULAR
                                TIEMPO    : PASADO
                                VERBO-COP : NO
                                PREFECTO  : NO
                                PROGRESIVO: NO
                                TIPO      : COMUN
               OBJETO-DIR: LA  DEUDA
               GENERO    : FEMENINO
               NUMERO    : SINGULAR
               TIPO-DET  : DEF
               PERSONA   : TERCERA
                    CABEZA-GN : ART-DEF   : LA
                                GENERO    : FEMENINO
                                NUMERO    : SINGULAR
                    CUERPO-GN : SUSTANTIVO: DEUDA
                                GENERO    : FEMENINO
                                NUMERO    : SINGULAR
                                TIPO-NOM  : COMUN
               OBJETO-IND: SEMEJANTE  CONEJO
               GENERO    :
               NUMERO    : SINGULAR
               TIPO-DET  : CALIF
               PERSONA   : TERCERA
                    CABEZA-GN : ADJETIVO  : SEMEJANTE
                                GENERO    : NEUTRO
                                NUMERO    : SINGULAR
                                CALIF     : SI
                    CUERPO-GN : SUSTANTIVO: CONEJO
                                GENERO    : MASCULINO
                                NUMERO    : SINGULAR
                                TIPO-NOM  : COMUN
>
```

```
 RUN ATN
ORACION
EL CONEJO COME UNA NARANJA CON MUCHO GUSTO
.....................................................
EL CONEJO COME UNA NARANJA CON MUCHO GUSTO
.....................................................
IDENTIFIQUE LA CADENA DE ENTRADA


ORACION   : EL  CONEJO COME UNA  NARANJA CON MUCHO  GUSTO
      SUJETO    : EL  CONEJO
      GENERO    : MASCULINO
      NUMERO    : SINGULAR
      TIPO-DET  : DEF
      PERSONA   : TERCERA
            CABEZA-GN : ART-DEF   : EL
                        GENERO    : MASCULINO
                        NUMERO    : SINGULAR
            CUERPO-GN : SUSTANTIVO: CONEJO
                        GENERO    : MASCULINO
                        NUMERO    : SINGULAR
                        TIPO-NOM  : COMUN
      ACCION    : COME
      PERSONA   : TERCERA
      NUMERO    : SINGULAR
      TIEMPO    : PRESENTE
      VERBO-COP : NO
            VERBO-P   : VERBO     : COME
                        PERSONA   : TERCERA
                        NUMERO    : SINGULAR
                        TIEMPO    : PRESENTE
                        VERBO-COP : NO
                        PREFECTO  : NO
                        PROGRESIVO: NO
                        TIPO      : COMUN
      OBJETO-DIR: UNA  NARANJA
      GENERO    : FEMENINO
      NUMERO    : SINGULAR
      TIPO-DET  : INDEF
      PERSONA   : TERCERA
            CABEZA-GN : ATR-INDEF : UNA
                        GENERO    : FEMENINO
                        NUMERO    : SINGULAR
            CUERPO-GN : SUSTANTIVO: NARANJA
                        GENERO    : FEMENINO
                        NUMERO    : SINGULAR
                        TIPO-NOM  : COMUN
      MODIF-ORAC: CON MUCHO  GUSTO
            PREP-GP   : PREPOSICIO: CON
            OBJETO-GP : MUCHO  GUSTO
            GENERO    :
            NUMERO    : SINGULAR
            TIPO-DET  : CALIF
            PERSONA   : TERCERA
                  CABEZA-GN : ADJETIVO  : MUCHO
                              GENERO    : NEUTRO
                              NUMERO    : SINGULAR
                              CALIF     : SI
                  CUERPO-GN : SUSTANTIVO: GUSTO
                              GENERO    : MASCULINO
                              NUMERO    : SINGULAR
                              TIPO-NOM  : COMUN
>
```

```
ORACION
LA CONEJA ESTA AHI ARRIBA EN LA SEMANA
.....................................................
LA CONEJA ESTA AHI ARRIBA EN LA SEMANA
.....................................................
IDENTIFIQUE LA CADENA DE ENTRADA


          ORACION   : LA  CONEJA ESTA AHI  ARRIBA  EN LA  SEMANA
                SUJETO    : LA  CONEJA
                GENERO    : FEMENINO
                NUMERO    : SINGULAR
                TIPO-DET  : DEF
                PERSONA   : TERCERA
                      CABEZA-GN : ART-DEF   : LA
                                  GENERO    : FEMENINO
                                  NUMERO    : SINGULAR
                      CUERPO-GN : SUSTANTIVO: CONEJA
                                  GENERO    : FEMENINO
                                  NUMERO    : SINGULAR
                                  TIPO-NOM  : COMUN
                ACCION    : ESTA
                PERSONA   : TERCERA
                NUMERO    : SINGULAR
                TIEMPO    : PRESENTE
                VERBO-COP : SI
                      VERBO-P   : VERBO     : ESTA
                                  PERSONA   : TERCERA
                                  NUMERO    : SINGULAR
                                  TIEMPO    : PRESENTE
                                  VERBO-COP : SI
                                  PREFECTO  : NO
                                  PROGRESIVO: NO
                                  TIPO      : ESTAR
                MODIF-ORAC: AHI
                      MODIF     : ADVERBIO  : AHI
                                  TIPO      : LUGAR
                MODIF-ORAC: ARRIBA
                      MODIF     : ADVERBIO  : ARRIBA
                                  TIPO      : LUGAR
                MODIF-ORAC: EN LA  SEMANA
                      PREP-GP   : PREPOSICIO: EN
                      OBJETO-GP : LA  SEMANA
                      GENERO    : FEMENINO
                      NUMERO    : SINGULAR
                      TIPO-DET  : DEF
                      PERSONA   : TERCERA
                            CABEZA-GN : ART-DEF   : LA
                                        GENERO    : FEMENINO
                                        NUMERO    : SINGULAR
                            CUERPO-GN : SUSTANTIVO: SEMANA
                                        GENERO    : FEMENINO
                                        NUMERO    : SINGULAR
                                        TIPO-NOM  : COMUN
>
```

VI. Conclusiones

El propósito de cualquier sistema de comprensión de lenguaje natural es determinar el significado que se intenta transmitir con una sucesión de palabras. *Cómo* debe proceder exactamente un sistema de esta naturaleza es aún una pregunta abierta. No se ha determinado precisamente cuántos niveles debe tener el análisis, ni de qué manera deben interaccionar entre si.

Algunos son partidarios de realizar análisis guiados principalmente por procedimientos semánticos, reduciendo la dependencia del analizador a criterios sintácticos al mínimo. os sistemas ponen énfasis, por ejemplo, en el conocimiento del sistema del universo en el que se trabaja, en las propiedades de cada objeto en él, así como en el significado de las palabras de acuerdo al dominio de aplicación.

Otros, en cambio, prefieren levantar la mayor cantidad posible de información con base en evidencias sintácticas, de modo que el análisis semántico cuente con la mayor cantidad de elementos posibles. Entre ambos extremos se encuentran una gran variedad de decisiones.

De modo similar, se ofrecen soluciones opuestas a la interacción de los distintos componentes sintácticos y semánticos. La primera prescribe una secuencia fija de pasos en el proceso, y va del análisis sintáctico al semántico de una manera ordenada. La segunda prefiere que todos los componentes de ambos análisis se relacionen y dependan unos de otros para guiar el proceso. También aquí varía el grado de compromiso con uno u otro enfoque.

Respecto a dichos planteamientos, esta tesis se desarrolló de acuerdo a las decisiones siguientes.

Se considera, primero, que es preferible desarrollar herramientas independientes entre si, lo más poderosas que se pueda, y en este caso en particular, un analizador sintáctico completo. La elección de esta estrategia pretende que el análisis sintáctico sea lo más rico posible, permitiendo con ello la obtención de toda la información disponible en esta etapa.

El diseño de analizadores preferentemente semánticos exige la construcción de sistemas ad hoc, en los que decisiones fundamentales se basan en los objetos del universo y en el modo en que se definen sus propiedades, de acuerdo a su uso en la aplicación. La pérdida de eficiencia inherente a herramientas de uso general está ampliamente compensada por sus posibilidades de aplicación, dado que en un programa que aproveche al máximo evidencias sintácticas, no es indispensable información semántica asociada a universo alguno.

En el otro plano, se eligió trabajar en la dirección del desarrollo de herramientas capaces de funcionar aisladamente. La experiencia ha demostrado que la creación de subsistemas mutuamente dependientes trae como resultado sistemas sumamente complejos y muy difíciles de administrar. Dicha complejidad crece exponencialmente para cada agregación a cada uno de los componentes del sistema. Además, ninguno de los subsistemas es susceptible de separarse del resto y ser usado en otra aplicación. Otra vez, la preferencia por herramientas independientes resulta en productos necesariamente menos eficientes. Se considera que dicha pérdida se ve ampliamente recompensada por la posibilidad de trabajar con herramientas que se puedan administrar y hacer crecer de manera razonable.

El modelo usado, el de los ATNs, es adecuado para especificar y diseñar gramáticas de lenguajes naturales. Sin embargo, este modelo no es, por si mismo, suficientemente poderoso. Los ATNs exigen patrones en las oraciones a resolver principalmente por el orden de las palabras que las componen. Este modo de restringir las cadenas de entrada no siempre es el más adecuado, sobre todo en un idioma como el español, en el que el orden de las palabras es variable y en el que es posible omitir elementos importantes de la oración, como los pronombres personales.

Este trabajo se centró en el diseño de una gramática de ATNs de oraciones del español. Sin embargo, no se resolvieron todos los problemas que el tema presenta. Destacan, por su dificultad, dos de ellos: el manejo del fenómeno conocido como de *tematización*, que permite que el orden de las palabras se altere para dar énfasis a los distintos elementos que la componen. El segundo problema sin resolver comprende el manejo de las oraciones compuestas. Por las carencias de los ATNs antes señaladas, no es claro que estos problemas puedan resolverse sin cambiar, al menos un poco, el formalismo de trabajo.

El modelo no determinístico de los ATNs refleja naturalmente las ambiguedades del lenguaje natural y da un ambiente cómodo para especificar gramáticas. Sin embargo, la administración del no determinismo en una computadora produce

sistemas que no son óptimos.

Los últimos desarrollos en materia de analizadores sintácticos, destacadamente los llamados de *wait-and-see*, se apoyan en ATNs para hacer análisis locales, reconociendo descendentemente a los grupos que componen la oración. Las gramáticas de ATNs, por ello, se seguirán usando en el futuro cercano.

A continuación, se señalan algunos problemas asociados a esta tesis en los que se puede trabajar en lo sucesivo.

Continuar el desarrollo de herramientas para diseño y prueba de gramáticas de ATNs (editores, administradores de diccionarios, facilidades para hacer trazas, etc.), que permitan su elaboración en un tiempo breve y con un esfuerzo razonable.

Incorporar al presente sistema un analizador léxico, junto con las herramientas asociadas, que permita un manejo eficiente del diccionario de palabras.

Elaborar estrategias de análisis de las gramáticas y de administración de trayectorias que permitan la programación de intérpretes más eficientes.

Examinar las posibilidades de interacción de analizadores sintácticos y semánticos *puros*, partiendo de la hipótesis de independencia entre ellos.

Investigar el desempeño de sistemas de análisis de lenguaje natural similares en el ambiente de trabajo de microcomputadoras, así como su posible uso en sistemas que empleen al español como lenguaje de comunicación.

Bibliografía

Abelson, Robert (1973): "The structure of belief systems"; en Schank(1973): 287-339

Arden, Bruce (1980): *What Can Be Automated? The Computer Science and Engineering Research Study*; Cambridge, Mass., MIT Press

Bates, Madeleine (1978): "The theory and practice of augmented transition network grammars"; en Bolc(1978): 191-260

Bar-Hillel, Yehoshua (1964): *Language and Information. Selected Essays on their Theory and Application*; Reading, Mass., Addison-Wesley

Bernstein, Jeremy (1981): "Profiles (Marvin Minsky)"; *The New Yorker*, diciembre 14

Berry, Margaret (1975): *Introduction to Systemic Linguistics 1: Structures and Systems*; New York, St. Martin's Press

Berry, Margaret (1977): *Introduction to Systemic Linguistics 2: Levels and Links*; Londres, Batsford

Bobrow D. G. (1964): "Natural language input for a computer problem-solving system"; en Minsky(1968): 135-215

Bobrow, Daniel G. y J. Bruce Fraser (1969): "An augmented state transition network analysis procedure"; *1st IJCAI*: 557-567

Bobrow, D.G. y Terry Winograd (1977): "An overview of KRL, a knowledge representation language"; *Cognitive Science* 1(1):3

Bobrow, D.G., R.M. Kaplan, M. Kay, D.A. Norman, H. Thompson, y Terry Winograd (1977): "GUS, a frame-driven dialog system"; *Artificial Intelligence* 8(2): 155-173

Bolc, Leonard (1978): *Natural Communication with Computers*; New York, Springer-Verlag (Lecture Notes on Computer Science vol. 63)

Bolc, Leonard (1980): *Representation and Processing of Natural Language*; Berlin, Akademie-Verlag

Charniak, Eugene (1972): "Towards a model of children's story comprehension"; tesis doctoral, M.I.T. (Report AI TR-266)

Chomsky, Noam (1957): *Syntactic Structures*; s'Gravenhage, Mouton

Chomsky, Noam (1965): *Aspects of the Theory of Syntax*; Cambridge, Mass., MIT Press

Codd, E.F., R.S. Arnold, J-M. Cadiou, C.L. Chang y N. Roussopoulos (1978): "RENDEZVOUS Version 1: an experimental English-language query formulation system for casual users of relational data bases"; San José, Cal., IBM Research Laboratory, report RJ 2144

Colmerauer, Alain (1978): "Metamorphosis Grammar"; en Bolc(1978)

Fahlman, Scott (1979): *NETL: A System for Representing and Using Real-World Knowledge*; Cambridge, Mass., MIT Press

Feigenbaum, Edward y Julius Feldman (1963): *Computers and Thought*; New York, McGraw-Hill

Feigenbaum, Edward y P. McCorduck (1983): *The Fifth Generation*; Reading, Mass., Addison-Wesley

Fillmore, C.J. (1976): "The need for frame semantics within linguistics"; Ottawa, Ontario, *International Conference on Computational Linguistics*

Gili Gaya, Samuel (1961): *Curso Superior de Sintáxis Española*; Barcelona, Biblograf

Goffman, Erving (1974): *Frame Analysis*; New York, Harper Colophon Books

Green, Bertram, A.K. Wolf, C. Chomsky y K. Laughery (1963): "BASEBALL, an automatic question answerer"; en Feigenbaum(1963)

Hadlich, Roger (1975): *Gramática Transformativa del Español*; Madrid, Gredos

Harris, Larry (1977): "ROBOT, a high performance language interface for data base query"; Dartmouth College, Mathematics Department, Technical Report TR77-1

Harris, Larry (1978): "The ROBOT system: natural language processing applied to data base query"; *Proceedings of the ACM Annual Conference*: 165-172

Hendrix G. (1977): "Human engeneering for applied natural language processing"; *5th IJCAI*: 183-191

Hewitt, Carl (1969): "PLANNER, a language for proving theorems in robots"; *1st IJCAI*

Kaplan, Ronald (1972): "Augmented transition networks as psychological models of sentence comprehension"; *Artificial Intelligence* 3: 77-100

Kuno, Susumo y Anthony Oettinger (1962): "Multiple-path syntactic analyzer"; *Information Processing 62*: 306-311

Michie, Donald (1968): *Machine Intelligence 3*; Edimburgo, Edinburgh University Press

Minsky, Marvin (1968): *Semantic Information Processing*; Cambridge, Mass., MIT Press

Minsky, Marvin (1975): "A framework for representing knowledge"; en Winston(1975)

Nagao, Makoto (1984): "La Traducción Automática"; *Mundo Científico*, 4(33)

Newmeyer, Frederick (1980): *Linguistic Theory in America*; New York, Academic Press

Norman, D.A. y D.E. Rumelhart (1975): *Explorations in Cognition*; San Francisco, Freeman

Palmer, Stephen (1975): "Visual perception and world knowledge: notes on a model of sensory-cognitive interaction"; en Norman(1975): 279-307

Paxton, William (1978): "The language definition system"; en Walker(1978): 17-40

Peters, Stanley y R. Ritchie (1969): "A note on the universal hypothesis"; *Journal of Linguistics* 5: 150-152

Peters, Stanley y R. Ritchie (1971): "On restricting the base component of transformational grammars"; *Information and Control* 18: 483-501

Peters, Stanley y R. Ritchie (1973): "On the generative power of transformational grammars"; *Information Sciences* 6: 49-83

Petrick, Stanley (1973): "Transformational Analysis"; en Rustin(1973): 27-41

Petrick, Stanley (1981): "Field-testing the transformational question-answering (TQA) system"; *Proc, 19th Ann. ACL*: 35-36

Pierce, John (1966): Washington, D.C., *National Research Council*, pub. no. 1416

Pratt, Vaughan (1975): "Lingol, a progress report"; *th IJCAI*: 422-428

Quillian, M. Ross (1968): "Semantic memory"; en Minsky(1968): 227-270

Raphael, Bertram (1967): "SIR, a comput r program for semantic information retrieval"; en Minsky(1967): 33-145

Raphael, Bertram (1976): *The Thinking Computer*; San Francisco, Freeman

Roberts, R. Bruce e Ira P. Goldstein (1977): "The FRL primer"; AI Laboratory, MIT, Memo 408

Rosenberg, Richard (1980): "Approaching discourse computationally: a review"; en Bolc(1980): 11-83

Rustin, Randall (1973): *Natural Language Processing*; New York, Algorithmics Press

Sager, Naomi (1981): *Natural Language Information Processing*; Reading, Mass., Addison-Wesley

Schank, Roger (1973): "Identification of conceptualizations underlying natural language"; en Schank(1973b): 187-247

Schank, R.C. y K.M. Colby (1973b): *Computer Models of Thought and Language*; San Francisco, Freeman

Schank, Roger (1975): "Using knowledge to understand"; en Schank(1975)b

Schank, Roger y B.L. Nash-Webber (1975): *Theoretical Issues in Natural Language Processing*; Cambridge, Mass.

Schank, Roger y Robert Abelson (1977): *Scripts, Plans, Goals and Understanding*; New York, Lawrence Erlbaum Associates

Schubert, L.K. (1976): "Extending the expressive power of semantic networks"; *Artificial Intelligence* 7(2): 163-198

Thorne, J., P. Bratley y H. Dewar (1968): "The syntactic analysis of English by machine"; en Michie(1968)

Walker, Donald (1978): *Understanding Spoken Language*; New York, Elsevier-North Holland

Waltz D.L., T.Finin, F. Green, F. Conrad, B. Goodman y G. Hadden (1976): "The PLANES system: natural language access to a large data base"; Urbana, Ill., University of Illinois, Coordinated Science Lab., Tech. Report T-34

Waltz, David y Bradley Goodman (1977): "Writing a natural language data base system"; *5th IJCAI*

Waltz, David (1978): "An English language question answering system for a large relational data base"; *CACM* 21(7): 526-539

Weizenbaum, J. (1966): "ELIZA -a computer program for the study of natural language communication between man and machine"; *CACM* 9(1): 36-45

Wilensky, Robert (1978): "Why John married Mary: understanding stories involving recurring goals"; *Cognitive Science* 2(3): 235-266

Wilks, Yorick (1976): "Frames, scripts, stories and fantasies"; Ottawa, Ontario, *International Conference on Computational Linguistics*

Winograd, Terry (1972): *Understanding Natural Language*; New York, Academic Press

Winograd, Terry (1973): "A procedural model of language understanding"; en Schank(1973b): 152-186

Winograd, Terry (1983): *Language as a Cognitive Process Vol. I: Syntax*; Reading, Mass., Addison-Wesley

Woods, W.A. (1970): "Transition network grammars for natural language analysis"; *CACM* 13(10): 591-606

Woods, W.A. (1973): "Progress in natural language understanding: an application to lunar geology"; *AFIPS Conference Proceedings 42*

Zwicky, Arnold, J. Friedman, B. Hall y D. Walker (1965): "The MITRE syntactic analysis procedure for transformational grammars"; Washington, D.C., *IFIPS Proc. Fall Joint Comp. Conf. 1965*: 317-326

Apéndice: El Intérprete

```
PROGRAM ATN (TTY);
(*      Interprete de atn's      .
        Reconoce si una oracion esta en la gramatica definida
        por el ATN descrito en ATN.DAT .
        Recorre todas las alternativas posibles.
        Entrega las estructuras resultantes.

        enero de 1985.                                              *)

CONST   maxs = 50;                          (* tama~o stack ejecucion *)
        maxsx = 10;                         (* tama~o stack auxiliar  *)
        maxp = 15;                          (* tama~o palabra en LN *)
        maxl = 15;                          (* numero de palabras de entrada *)
        maxpe = 50;                         (* almacen de palabras especiales *)
        maxacc = 175;                       (* tama~o tabla de acciones *)
        maxc = 75;                          (* tama~o tabla de codigo   *)
        maxcond = 200;                      (* tama~o tabla de condiciones *)
        maxd = 30;                          (* tama~o diccionario *)
        maxn = 10;

        maxpapel = 7;
        maxcarac = 8;

TYPE    indxs = 0..maxs;
        indxsx = 0..maxsx;
        indxc = 0..maxc;
        indxp = 0..maxp;
        indxl = 0..maxl;
        indcarac = 1..maxcarac;
        indpapel = 1..maxpapel;
        indxacc = 0..maxacc;
        indxcond = 0..maxcond;
        indxpe = 1..maxpe;
        indxn = 1..maxn;
        indxd = 0..maxd;
        numid = 0..63;
        IOSPEC = (RANDOM,UPDATE,APPEND,TEMPORARY,INSERT,SHARED,SPOOL,
                  BLOCK);

        tipored = (ORACION,GNOM,GVERB,GPREP,GADJ);
        tipoarco = ( MANDA, BUSCA, BRINCA, CAT, LEX );
        ctlconJ = SET OF numid;
        tipoid = (idnodo,idconst);
        tipoacc = (ac,ap,co,anula,aotra);
        tipocond = (cp,cc,cnula,cotra);
        tipocp = (ig,dif);
        tiponodo = (ter,nter,lisa);

        apunodo = ^nodo;

        cadena = PACKED ARRAY [indxp] OF CHAR;
        linea = PACKED ARRAY[1..80] OF CHAR;
        nomid = PACKED ARRAY[indxn] OF CHAR;

        tipid = (papid, caracR, valtC, catlex, caracC, valtCX);
        idres = RECORD
                        r0: tipid;
                        r1: numid;
                        r2: numid;
                        r21: numid;
                        r3: nomid
                END;
```

```
codres = RECORD
                origen, destino: CHAR;
                CASE arco: tipoarco OF
                        CAT: ( elm: numid );
                        BUSCA: ( blanco: tipored );
                        LEX: ( dicpal: INTEGER )
          END;

diccres = RECORD
                ident: cadena;
                catpal: numid;
                atr: ARRAY[indcarac] OF numid
          END;

stackres = RECORD
                pos: indxl;
                arc: indxc;
                at: apunodo
           END;


nodo = RECORD
        carac: ARRAY[indcarac] OF INTEGER;
        CASE tipon:tiponodo OF
                nter:( papel: ARRAY[indpapel] OF apunodo;
                       tnodo: tipored );
                ter:   ( ll: indxl;
                         cl: numid );
                lisa: ( sigl, sigm: apunodo )
       END;

accres = RECORD
                tipa: tipoacc;
                aizq: INTEGER;
                tder: tipoid;
                ader: INTEGER;
                siga: BOOLEAN
         END;

condres = RECORD
                tipc: tipocond;
                busca: tipocp;
                cizq: INTEGER;
                tder: tipoid;
                cder: INTEGER;
                sisc: BOOLEAN
          END;


VAR     s: ARRAY[indxs] OF stackres;                    (* stack de ejecucion *)
        saux: ARRAY[indxsx] OF stackres;                    (* stack auxiliar *)
        codigo: ARRAY[indxc] OF codres;                 (* matriz de codigo *)
        tatn:   ARRAY[tipored] OF indxc;                (* apuntadores a red *)
        tnodos: ARRAY['A'..'Z'] OF indxc;               (* apuntadores a nodos *)
        tcond:  ARRAY[indxc] OF indxcond;               (* apuntadores a condiciones *)
        tacc:   ARRAY[indxc] OF indxacc;                (* apuntadores a acciones *)
        acc:    ARRAY[indxacc] OF accres;               (* almacen de acciones *)
        cond:   ARRAY[indxcond] OF condres;             (* almacen de condiciones *)
        palabras: ARRAY [indxl] OF INTEGER;             (* almacen de la cadena de
```

```
                                                    entrada *)
        palesp: ARRAY[indxpe,indxp] OF CHAR;    (* palabras especiales *)

        admit0: FILE OF diccres;
        admit2: FILE OF INTEGER;
        admit1: FILE OF CHAR;
        admit3: FILE OF idres;
        admit4: FILE OF INTEGER;

        dicc: ARRAY [indxd] OF diccres;         (* diccionario *)
        sp,spx: indxs;                          (* apuntadores a stacks *)
        palp: INTEGER;                          (* apuntador a 'palesp' *)
        dp: indxd;
        ppmax: indxl;
        codmax: indxc;
        accmax: indxacc;
        condmax: indxcond;
        diccmax : INTEGER;
        p: indxl;
        q, exito: BOOLEAN;
        NAB: apunodo;
        prox3: INTEGER;
        caza: PACKED ARRAY[1..8] OF CHAR;
        cazared: tipored;


PROCEDURE error ( i: INTEGER );
BEGIN (* error *)
        WRITELN ( '********** ERROR TIPO  ',i:2 );
        i:= i DIV 0;    (* detengo intencionalmente la ejecucion *)
END; (* error *)


FUNCTION posicion ( x: cadena ): INTEGER;
        VAR     i, j, k:  INTEGER;
                y: diccres;
BEGIN (* posicion *)
        i:= 1; j:= diccmax;
        REPEAT
                k:= ( i + j ) DIV 2; GET(admit0,k); y:= admit0^;
                IF ( x <= y.ident ) THEN j:= k-1;
                IF ( x >= y.ident ) THEN i:= k+1
        UNTIL ( i > j );
        IF ( i - 1 ) > j THEN BEGIN
                IF k>1 THEN GET(admit0,k-1);
                WHILE (admit0^.ident=x) AND (k>1) DO BEGIN
                        k:= k-1; GET(admit0,k-1)
                END;
                posicion:= dp+1; GET(admit0,k);
                REPEAT
                        dp:= dp+1; dicc[dp]:= admit0^;
                        k:= k+1; GET(admit0,k)
                UNTIL admit0^.ident<>x
        END ELSE
                posicion:= 0;
END; (* posicion *)

PROCEDURE leecadena;
        (* lee la cadena de entrada y la asigna al arreglo
           'palabra' *)
        VAR     i, j, pos, numc: INTEGER;
```

```
                app: indx1;
                car: CHAR;
                a: cadena;
                linea: PACKED ARRAY [1..80] OF CHAR;
BEGIN (* leecadena *)
        i:= 0; numc:= 0;  app:= 0;
        READLN;
        WHILE ( NOT EOLN ) DO
        BEGIN
                i:= i + 1; READ(car);  linea[i]:= car
        END; numc:= i;
        i:= 1;
        WHILE ( i <= numc ) DO
        BEGIN
                WHILE (linea[i] = ' ') AND (i+1<80) DO i:= i+1; j:= 0;
                IF ( i<= numc ) THEN
                BEGIN
                        WHILE ( linea[i] IN ['A'..'Z'] ) DO
                        BEGIN
                                a[j]:= linea[i]; j:= j+1;  i:= i+1
                        END;
                        FOR pos:=j TO maxp DO a[pos]:= ' ';
                        pos:= posicion ( a );
                        IF pos>0 THEN
                        BEGIN
                                app:= app + 1;
                                palabras[app]:= pos
                        END ELSE BEGIN
                                WRITELN ('PALABRA NO IDENTIFICADA ');
                                error(2)
                        END
                END;
        END;
        ppmax:= app
END; (* leecadena *)

PROCEDURE esccadena;
        (* Emite la cadena leida. *)
        VAR     i: indx1;
                j,k: INTEGER;
BEGIN (* esccadena *)
        WRITELN('.....................................................');
        FOR i:= 1 TO ppmax DO
        BEGIN
                j:= palabras[i]; k:= 0;
                REPEAT
                        WRITE(dicc[j].ident[k]);
                        k:= k+1
                UNTIL dicc[j].ident[k]= ' ';
                WRITE (' ');
        END; WRITELN;
        WRITELN('.....................................................');
        WRITELN;
END; (* esccadena *)

PROCEDURE leecodiso;
        (* Lee del archivo de entrada los atn's.        *)

VAR     j,j0,j1: INTEGER;
        FIN: BOOLEAN;
        carant: CHAR;
```

```
cuerda: linea;
siscar: CHAR;
kO: INTEGER;

FUNCTION leelinea :linea;
        (* Lee una linea del archivo 'admit1'. *)
        VAR     i,J: INTEGER;
                CAR: CHAR;
                x: linea;
BEGIN (* leelinea *)
        i:= 0;
        IF siscar<>'@' THEN BEGIN
                i:= i+1; x[1]:= siscar
        END;
        WHILE NOT EOLN(admit1) DO BEGIN
                i:= i+1; READ(admit1,CAR); x[i]:=CAR
        END; READLN(admit1);
        IF NOT EOF(admit1) THEN BEGIN
                READ(admit1,CAR);  siscar:= CAR;
        END;
        FOR J:= i+1 TO 80 DO x[J]:= ' ';
        leelinea:= x
END;  (* leelinea *)

FUNCTION decod ( x:linea; i:INTEGER):INTEGER;
        VAR     i0,t: INTEGER;
BEGIN (* decod *)
        IF (x[i]='''') THEN BEGIN
                IF i=27 THEN cuerda[34]:= cuerda[46];
                IF i=25 THEN cuerda[32]:= cuerda[44];
                palp:= palp+1;
                FOR i0:= i+1 TO i+maxp+1 DO palesp[palp,i0-i-1]:= x[i0];
                decod:=palp
        END ELSE BEGIN
                t:= 0;
                FOR i0:= i+1 TO i+4 DO
                        IF x[i0] IN ['0'..'9'] THEN
                                t:= t*10+ ord(x[i0])-ord('0')
                        ELSE  error(16);
                CASE x[i] OF
                   '+',' ': decod:= t;
                   '-':     decod:= -t;
                   OTHERS:  error(16)
                END
        END
END;  (* decod *)


PROCEDURE asscod;
        (* Asigna el registro de codigo. *)
BEGIN  (* asscod *)
        cuerda:= leelinea;
        WITH codigo[J] DO BEGIN
        IF cuerda[10] IN ['A'..'Z'] THEN origen:= cuerda[10]
        ELSE error(10);
        CASE cuerda[15] OF
           'C': arco:= CAT;
           'L': arco:= LEX;
           'M': arco:= MANDA;
           'B': BEGIN
```

```
                    IF cuerda[16]='R' THEN  arco:= BRINCA
                    ELSE IF cuerda[16]='U' THEN arco:= BUSCA
                    ELSE error(10)
                END;
            OTHERS: error(10)
        END;
        IF arco<>MANDA THEN
                IF cuerda[13] IN ['A'..'Z'] THEN destino:= cuerda[13]
                ELSE error(10)
        ELSE destino:= origen;
        CASE arco OF
            LEX: dicpal:= decod(cuerda,23);
            BUSCA:CASE cuerda[23] OF
                    'G': IF cuerda[24]='N' THEN blanco:= GNOM
                            ELSE IF cuerda[24]='V' THEN blanco:= GVERB
                            ELSE IF cuerda[24]='P' THEN blanco:= GPREP
                            ELSE IF cuerda[24]='A' THEN blanco:= GADJ
                            ELSE error(9);
                    'O': IF cuerda[24]='R' THEN blanco:= ORACION
                            ELSE error(9);
                    OTHERS: error(9)
                END;
            CAT: elm:=(ORD(cuerda[23])-ORD('0'))*10+ORD(cuerda[24])-ORD('0')
        END
        END;  (* with *)
        IF ( cuerda[10] <> carant ) THEN tnodos[cuerda[10]]:=J;
        IF (cuerda[1]<>' ') THEN
        CASE cuerda[1] OF
            'G': IF cuerda[2]='N' THEN tatn[GNOM]:= J
                    ELSE IF cuerda[2]='V' THEN tatn[GVERB]:= J
                    ELSE IF cuerda[2]='P' THEN  tatn[GPREP]:= J
                    ELSE IF cuerda[2]='A' THEN  tatn[GADJ]:= J
                    ELSE error(7);
            'O': IF cuerda[2]='R' THEN tatn[ORACION]:= J
                    ELSE error(7);
          OTHERS: error(7)
        END;
        carant:= cuerda[10]; J:= J+1
END;  (* asscod *)

PROCEDURE assacc(x:tipoacc);
        (* Asigna el contenido del registro de acciones. *)
BEGIN  (* assacc *)
        IF x<>aotra THEN tacc[J-1]:= JO;
        IF NOT(EOF(admit1)) AND (x<>anula) THEN
        BEGIN
                cuerda:= leelinea;
                WITH acc[JO] DO
                BEGIN
                        CASE cuerda[8] OF
                        'P':    tipa:= ap;
                        'C':    IF cuerda[9]='A' THEN tipa:= ac
                                        ELSE IF cuerda[9]='O' THEN tipa:= co
                                ELSE error(11);
                        OTHERS: error(11)
                        END;
                        aiza:= decod(cuerda,14);
                        CASE cuerda[20] OF
                        'N':    tder:= idnodo;
                        'C':    tder:= idconst;
                                OTHERS: error(12)
```

```
                        END;
                        ader:= decod(cuerda,25);
                        sisa:= (cuerda[32]='Y')
                END;
                J0:= J0+1;  IF acc[J0-1].sisa THEN assacc(aotra);
        END ELSE BEGIN
                acc[J0].tipa:= anula; J0:= J0+1
        END
END;  (* assacc *)

PROCEDURE asscon(x:tipocond);
        (* Asigna el registro de condiciones. *)
BEGIN  (* asscon *)
        IF x<>cotra THEN tcond[J-1]:= J1;
        IF NOT (EOF(admit1)) AND (x<>cnula) THEN
        BEGIN
                cuerda:= leelinea;
                WITH cond[J1] DO BEGIN
                        CASE cuerda[8] OF
                        'P':    tipc:= cp;
                        'C':    tipc:= cc;
                        OTHERS: error(13)
                        END;
                        ciza:= decod(cuerda,14);
                        CASE cuerda[20] OF
                        '=':    busca:= ig;
                        '#':    busca:= dif;
                        OTHERS: error(13)
                        END;
                        CASE cuerda[22] OF
                        'N':    tder:= idnodo;
                        'C':    tder:= idconst;
                        OTHERS: error(15)
                        END;
                        cder:= decod(cuerda,27);
                        sisc:= (cuerda[34]='Y');
                END;
                J1:= J1+1; IF cond[J1-1].sisc THEN asscon(cotra)
        END ELSE BEGIN
                cond[J1].tipc:= cnula; J1:= J1+1
        END
END;  (* asscon *)
BEGIN  (* leecodiso *)
        J:= 1;  carant:= ' '; J1:= 1;  J0:= 1; siscar:= '@';
        WHILE ( NOT EOF admit1 ) DO
        BEGIN
                asscod;
                IF siscar='C' THEN BEGIN
                        asscon(cp);
                        IF siscar='A' THEN BEGIN
                                assacc(ap);
                                IF siscar='S' THEN BEGIN
                                        asscon(cotra);
                                        IF siscar='A' THEN
                                                assacc(aotra)
                                        ELSE
                                                error(21)
                                END
                        END ELSE assacc(anula)
                END ELSE BEGIN
                        asscon(cnula);
```

```
                        IF siscar='A' THEN  assacc(ap)
                        ELSE    assacc(anula)
                END
        END;
        codmax:= j-1; accmax:= j0-1; condmax:= j1-1
END;    (* leecodigo *)

PROCEDURE imparbol(x:apunodo; niv:INTEGER; padre:tipored; nohijo:INTEGER);
        VAR     id: nomid;
                i,j:  INTEGER;
                a: apunodo;

        FUNCTION traeid (z0:tipid; z1,z2,z21: numid):nomid;
                VAR i: INTEGER;
                    fin: BOOLEAN;
        BEGIN (* traeid *)
                i:= 0; fin:= FALSE;
                WHILE (NOT FIN) AND (i+1<prox3) DO
                BEGIN
                        i:= i+1; GET(admit3,i);
                        WITH admit3^ DO
                                fin:= ( ( r0=z0) AND (r1=z1) AND
                                        ( r2=z2) AND (r21=z21)  );
                END;
                IF FIN THEN
                        traeid:= admit3^.r3
                ELSE
                        traeid:= '          ';
        END; (* traeid *)

        PROCEDURE impter (x:apunodo);
                VAR     i,j: INTEGER;
                        a: apunodo;
        BEGIN
                WITH x^ DO
                IF tipon=ter THEN  BEGIN
                        j:= 0;
                        REPEAT
                                WRITE(dicc[palabras[ll]].ident[j]);
                                j:= j+1
                        UNTIL  dicc[palabras[ll]].ident[j]=' ';
                        WRITE(' ');
                END
                ELSE IF tipon=nter THEN
                        FOR i:=1 TO maxpapel DO
                        BEGIN
                                IF papel[i]<>NIL THEN impter(papel[i])
                        END
                ELSE BEGIN
                        a:= x;
                        REPEAT
                                impter(a^.sigm); WRITE(' ');
                                a:= a^.sigl
                        UNTIL a=NIL;
                END
        END;
BEGIN   (* imparbol *)
        WITH x^ DO
        IF tipon=nter THEN BEGIN
                FOR j:= 0 TO niv DO WRITE('      ');
                id:= traeid(papid,ORD(padre)+1,nohijo,0);
```

```
                WRITE (id,': ');
                imprter(x); WRITELN;
                FOR i:=1 TO maxcarac DO
                IF carac[i]>0 THEN
                BEGIN
                        FOR J:= 0 TO niv DO WRITE('      ');
                        id:= traeid(caracR,ORD(tnodo)+1,i,0);
                        WRITE (id,': ');
                        id:= traeid(valtC,ord(tnodo)+1,i,carac[i]);
                        WRITELN (id);
                END;
                FOR i:=1 TO maxPapel DO
                        IF Papel[i]<>NIL THEN imparbol(Papel[i],niv+1,tnodo,i)
        END ELSE IF tipon=ter THEN BEGIN
                FOR J:= 0 TO niv DO WRITE ('      ');
                id:= traeid(Papid,ORD(Padre)+1,nohiJo,0); WRITE(id,': ');
                id:= traeid(catlex,cl,0,0);
                WRITE (id,': '); imprter(x); WRITELN;
                FOR i:= 1 TO maxcarac DO
                IF carac[i]>0 THEN BEGIN
                        FOR J:= 0 TO niv DO WRITE('      ');
                        FOR J:= 1 TO maxn+2 DO WRITE(' ');
                        id:= traeid(caracC,cl,i,0);
                        WRITE (id,': ');
                        id:= traeid(valtCX,cl,i,carac[i]);
                        WRITELN (id);
                END
        END ELSE BEGIN
                a:= x;
                REPEAT
                        imparbol(a^.sism,niv,Padre,nohiJo);
                        a:= a^.sisl
                UNTIL a=NIL
        END
END;  (* imparbol *)


PROCEDURE rur (VAR a:BOOLEAN; rd:tipored);
        (* Recorre la red 'rd' *)

        VAR     t, t0: indxs;   (* Base de 's' en nivel presente *)
                raiz: apunodo;
                a1: BOOLEAN;
                h: INTEGER;
                ca: indxc;      (* Arco presente *)
                cuerda: cadena;

        FUNCTION traecat (x:indx1): ctlconJ;
                VAR     i,J: indxd;
                        y: ctlconJ;
        BEGIN (* traecat *)
                y:= []; i:= Palabras[x]; J:= i;
                REPEAT
                        y:= y OR [dicc[i].catPal];
                        i:= i+1
                UNTIL dicc[i].ident <> dicc[J].ident;
                traecat:= y
        END; (* traecat *)

        FUNCTION tapun(x:INTEGER):apunodo;
                (* Trae el nodo asociado al entero 'x' *)
```

```
        VAR i,J,k: INTEGER;
            a: apunodo;
BEGIN (* tapun *)
        IF x<0 THEN a:= raiz ELSE a:= NAB;
        x:= abs(x); J:= 1000;
        FOR i:= 1 TO 3 DO BEGIN
                k:= x DIV J; x:= x MOD J;
                IF k>0 THEN
                        IF a^.papel[k]<>NIL THEN
                                a:= a^.papel[k]
                        ELSE
                                error(32)
                ELSE    ;
                J:= J DIV 10
        END;
        tapun:= a
END; (* tapun *)

FUNCTION traepN(x:INTEGER):apunodo;
        (* Regresa el apuntador asociado a 'x'.
           Acepta NIL.                                  *)
        VAR  a: apunodo;
BEGIN (* traepN *)
        a:= tapun(x); x:= abs(x) MOD 10;
        IF x>0 THEN traepN:= a^.papel[x]
        ELSE traepN:= a
END; (* traepN *)

FUNCTION traep(x:INTEGER):apunodo;
(* Regresa el apuntador al papel 'x' *)
        VAR     i,J,k: INTEGER;
                a: apunodo;
BEGIN (* traep *)
        a:= tapun(x); x:= abs(x) MOD 10;
        IF x>0 THEN
                IF a^.papel[x]<>NIL THEN traep:= a^.papel[x]
                                    ELSE error(32)
        ELSE traep:= a
END; (* traep *)

FUNCTION traec(x:INTEGER; y:tipoid):INTEGER;
        (* Regresa el contenido de la caracteristica asociada a 'x' *)
        VAR     i,J,k: INTEGER;
                a: apunodo;
BEGIN (* traec *)
        IF y=idconst THEN traec:= x
        ELSE BEGIN
                a:= tapun(x); x:= abs(x) MOD 10;
                traec:= a^.carac[x]
        END
END; (* traec *)

FUNCTION condiciones (VAR x:indxc);BOOLEAN;
        (* Resuelve las condiciones asociadas a un arco *)
        VAR     i: indxcond;
                r: condres;
                p: BOOLEAN;
                a1,a2: INTEGER;
                a: apunodo;
BEGIN (* cond *)
        p:= FALSE; i:= tcond[x];
```

```
        IF cond[i].tipc <> cnula THEN
        REPEAT
                r:= cond[i];
                WITH r DO
                IF (tipc=cc) THEN
                BEGIN
                        a1:= traec (ciza,idnodo);
                        a2:= traec (cder,tder);
                        p:= ( (a1<>a2) AND (busca=ig) ) OR
                            ( (a1=a2) AND (busca=dif) );
                END ELSE BEGIN
                        if tder<>idconst THEN error(33);
                        IF (cder=-1) THEN
                                p:=((traepN(ciza)<>NIL)AND(busca=ig))OR
                                   ( (traepN(ciza)=NIL)AND(busca=dif) )
                        ELSE BEGIN
                                a:= traep(ciza);
                                IF a^.tipon<>ter THEN error(34);
                                WITH dicc[palabras[a^.11]] DO
                                IF busca=ig THEN
                                        p:=  (ident<>palesp[cder])
                                ELSE
                                        p:= (ident=palesp[cder] );
                        END
                END;
                i:= i+1
        UNTIL (p) OR (NOT r.sigc);
        IF (p) AND (cond[i+1].tipc=cotra) THEN BEGIN
                x:= x+1;  p:= FALSE
        END;
        condiciones:= NOT p
END; (* cond *)

PROCEDURE acciones (x:indxc);
        (* Lleva a cabo las acciones asociadas a un arco *)
         VAR     r: accres;
                i: indxacc;
                a,a0: apunodo;
BEGIN (* acciones *)
        i:= tacc[x];
        IF acc[i].tipa <> anula THEN
        REPEAT
                r:= acc[i];
                WITH r DO
                CASE tipa OF
                   ac:  BEGIN (* accion s/carac *)
                                IF aiza<=0 THEN error(35);
                                raiz^.carac[aiza]:= traec(ader,tder)
                        END;
                   ap:  BEGIN (* accion s/papel *)
                                if aiza<=0 THEN error(35);
                                IF tder=idconst THEN
                                        IF ader=-1 THEN
                                           raiz^.papel[aiza]:= NIL
                                        ELSE error(35)
                                ELSE
                                        raiz^.papel[aiza]:= traep(ader)
                        END;
                   co:  BEGIN (* concatena *)
                                IF aiza<=0 THEN error(35);
                                a:= raiz^.papel[aiza];
```

```
                        IF a<>NIL THEN BEGIN
                                WHILE a^.sigl<>NIL DO
                                        a:=a^.sigl;
                                NEW(a0); a0^.tipon:=lisa;
                                a0^.sigl:=NIL; a0^.sigm:= traer(ader);
                                a^.sigl:= a0
                        END ELSE BEGIN
                                NEW(a0); a0^.tipon:= lisa;
                                a0^.sigl:= NIL; a0^.sigm:= traer(ader);
                                raiz^.papel[aiza]:= a0
                        END
                END (* concatena *)
        END;
        i:= i+1
    UNTIL NOT r.sisa
END; (* acciones *)

FUNCTION duplista (x:apunodo):apunodo;
    (* Copia una lista de modificadores *)
    VAR     a0: apunodo;
BEGIN (* duplista *)
    IF x<>NIL THEN BEGIN
        IF x^.tipon<>lisa THEN error(77);
        NEW(a0);  a0^.tipon:= lisa;   a0^.sigm:= x^.sigm;
        a0^.sigl:= duplista(x^.sigl);
        duplista:= a0
    END ELSE
        duplista:=NIL
END;  (* duplista *)

PROCEDURE push (x:indxp;  y:indxc; z:apunodo);
    VAR y0, y1: INTEGER;
        i:  INTEGER;
        a0,a1: apunodo;
BEGIN (* push *)
    IF x<=ppmax+1 THEN BEGIN
        sp:= sp + 1;
        WITH s[sp] DO BEGIN
            pos:= x; arc:= y;  at:= z
        END;
        y0:= y; y1:= y0+1;
        WHILE (codigo[y0].origen=codigo[y1].origen) DO BEGIN
            sp:= sp+1; s[sp].pos:= s[sp-1].pos;
            s[sp].arc:= y1; NEW(s[sp].at);
            s[sp].at^:= s[sp-1].at^; y1:= y1+1;
            a0:=s[sp-1].at; a1:=s[sp].at;
            FOR i:= 1 TO maxpapel DO
            IF a0^.papel[i]<>NIL THEN
            IF a0^.papel[i]^.tipon=lisa THEN
                a1^.papel[i]:= duplista (a0^.papel[i]);
        END
    END
END; (* push *)

PROCEDURE pushb;
    VAR     a: apunodo;
        i: indxs;
BEGIN (* pushb *)
    spx:= spx+1;
    saux[spx]:= s[sp+1];
END; (* pushb *)
```

```
PROCEDURE pop (VAR i:indx1;  VAR J:indxc; VAR qq:apunodo);
BEGIN (* pop *)
        WITH s[sp] DO BEGIN
                i:= pos; J:= arc; qq:= at
        END;
        IF codigo[s[sp].arc].arco<>BUSCA THEN sp:=sp-1;
END; (* pop *)

FUNCTION suc ( x: indxc ): indxc;
        (* Dado un arco 'x', regresa el nodo destino ( i. e., el
           primer arco de ese nodo ).                                    *)
BEGIN (* suc *)
        IF ( codigo[x].arco = BUSCA ) THEN
                suc:= tatn[codigo[x].blanco]
        ELSE
                suc:= tnodos[codigo[x].destino]
END; (* suc *)

PROCEDURE cstack;
        VAR     i,J: INTEGER;
                r: apunodo;
                a: BOOLEAN;
BEGIN  (* cstack *)
        r:= s[sp].at;
        sp:= sp-1; a:= FALSE;
        FOR i:= t0 TO spx DO BEGIN
                NAB:= saux[i].at;
                IF condiciones(ca) THEN BEGIN
                        NEW(raiz);  raiz^:= r^; a:= TRUE;
                        FOR J:= 1 TO maxpapel DO
                        IF r^.papel[J]<>NIL THEN
                        IF r^.papel[J]^.tipon=lista THEN
                                raiz^.papel[J]:= duplista(r^.papel[J]);
                        acciones(ca);
                        push(saux[i].pos,tnodos[codigo[ca].destino],raiz )
                END
        END; spx:= t0-1;
        IF NOT a THEN RELEASE
END; (* cstack *)

PROCEDURE craiz;
        (* Crea un nodo cabeza de red. *)
        VAR     i: INTEGER;
BEGIN (* craiz *)
        NEW (raiz);
        WITH raiz^ DO BEGIN
                tipon:=nter;
                tnodo:= rd;
                FOR i:= 1 TO maxcarac DO carac[i]:= 0;
                FOR i:= 1 TO maxpapel DO papel[i]:= NIL;
        END
END; (* craiz *);

FUNCTION nterminal(x: numid): apunodo;
        (* Crea un nodo terminal. *)
        VAR     nt: apunodo;
                k,i: INTEGER;
BEGIN  (* nterminal *)
        NEW (nt);
        WITH nt^ DO BEGIN
```

```
                tipon:= ter;
                cl:= x;
                ll:= p;
                IF x>0 THEN BEGIN
                        i:= palabras[p];
                        WHILE (dicc[i].catpal<>x) DO i:= i+1;
                        FOR k:= 1 TO maxcarac DO
                                carac[k]:= dicc[i].atr[k];
                END ELSE
                        FOR k:= 1 TO maxcarac DO
                                carac[k]:= 0
        END;
        nterminal:= nt
END; (* nterminal *)

GIN (* rur *)
        a:= FALSE;
        craiz;
        push ( p, tatn[rd], raiz );
        t:= sp;

        REPEAT
                pop (p,ca,raiz);
                WITH codigo[ca] DO                  (* trae siguiente eleccion *)
                CASE arco OF
                        LEX:    BEGIN (* lex *)
                                        cuerda:= dicc[palabras[p]].ident;
                                        IF cuerda=palesp[dicpal] THEN
                                        BEGIN
                                                NAB:= nterminal(0);
                                                IF condiciones(ca) THEN BEGIN
                                                        acciones(ca);
                                                        push (p+1,suc(ca),raiz)
                                                END
                                        END
                                END; (* lex *)
                        CAT:    BEGIN
                                        cuerda:= dicc[palabras[p]].ident;
                                        IF ( elm IN traecat(p) ) THEN
                                        BEGIN
                                                NAB:= nterminal(elm);
                                                IF condiciones (ca) THEN
                                                BEGIN
                                                        acciones (ca);
                                                        push (p+1, suc(ca), raiz )
                                                END
                                        END
                                END;  (* cat *)
                        MANDA:  BEGIN
                                        IF condiciones (ca) THEN
                                        BEGIN
                                                acciones (ca);
                                                pushb;
                                                a:= TRUE
                                        END
                                END; (* manda *)
                        BRINCA: BEGIN
                                        if condiciones (ca) THEN
                                        BEGIN
```

```
                                        acciones (ca);
                                        push ( p, suc(ca), raiz)
                                END
                        END; (* brinca *)
                BUSCA:  BEGIN
                                MARK; t0:= spx+1;
                                rur ( a1, blanco );
                                IF a1 THEN BEGIN
                                        cstack
                                END
                                ELSE BEGIN
                                        RELEASE;  sp:= sp-1
                                END
                        END
        END (* case *)
    UNTIL ( sp < t );
END; (* rur *)



BEGIN  (* programa principal *)

    RESET (admit0, 'DICC.DAT',,,[RANDOM]);
    RESET (admit2, 'DICC.PRX');
    RESET ( admit1, 'ATN.DAT' );
    RESET ( admit3, 'IDENTIF.DAT',,,[RANDOM]);
    RESET ( admit4, 'IDENTIF.PRX');
    dp:= 0; diccmax:= admit2^-1; prox3:= admit4^;

    caza:= '          ';
    READLN; READ(caza);
    IF caza='ORACION ' THEN cazared:= ORACION
    ELSE IF caza='GNOM     ' THEN cazared:= GNOM
    ELSE IF caza='GVERB    ' THEN cazared:= GVERB
    ELSE IF caza='GPREP    ' THEN cazared:= GPREP
    ELSE IF caza='GADJ     ' THEN cazared:= GADJ
    ELSE error(77);
    leecodigo;
    leecadena; esccadena;

    p:= 1; sp:= 0; a:= FALSE; spx:= 0;  palp:= 0; exito:= FALSE;

    rur ( a, cazared );
    IF a THEN BEGIN
            WHILE spx>0 DO BEGIN
                    WITH saux[spx] DO
                    IF pos-1=ppmax THEN BEGIN
                            exito:= TRUE;
                            WRITELN('IDENTIFIQUE LA CADENA DE ENTRADA');
                            WRITELN; WRITELN(' '); WRITELN(' ');
                            imparbol(at,1,cazared,0);
                    END;  spx:= spx-1
            END
    END;
    IF NOT exito THEN
            WRITELN ('NO IDENTIFIQUE LA CADENA DE ENTRADA')

END.  (* programa principal *)
[*
```