UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Unidad Académica de los Ciclos Profesional y de Posgrado del Colegio de Ciencias y Humanidades

DESARROLLO DE UN SISTEMA PARA LA COMPRENSION DE LENGUAJE NATURAL, UTILIZANDO PROLOG COMO LENGUAJE DE SOPORTE

T E S I S

Que presenta

ALFONSO SAN MIGUEL AGUIRRE

para obtener el título de

MAESTRO EN CIENCIAS DE LA COMPUTACION

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

C O N T E N I D O

CAPITULO 1.

INTRODUCCION

1.1. Presentación del problema.

Hoy día las computadoras son nuestros interlocutores en grandes áreas de actividad profesional, industrial y social. Lo serán aún más en el futuro si somos capaces de dotarlas de facilidad de comunicación en lenguaje natural. Este es uno de los temas que estudia la Inteligencia Artificial. Es un problema difícil y complejo.

Durante un cierto tiempo se pensó que el tratamiento del lenguaje, y en particular la traducción automática, era un problema de diccionarios y de reglas de correspondencia de una lengua a otra. Es conocido que durante los años cincuenta se invirtieron grandes cantidades de dinero y tiempo de investigación en el tema, sin llegar a resultados razonables y produciéndose casos que son ejemplos célebres de lo que se puede llegar a obtener cuando se hace traducción sin comprender.

Así, es famosa la frase: "The spirit is willing but the flesh is weak", que traducida al ruso, y después otra vez al inglés, se convirtió en: "The wine is agreeable but the meat is rotten".

Aquellos esfuerzos sirvieron para situar el problema y su dificultad. Para procesar el lenguaje es necesario entender, y entender es poseer conocimiento y ser capaz de aplicarlo adecuadamente en cada caso.

Imaginemos dos situaciones: en la primera, dos amigos A y B, pasean juntos y se produce el siguiente diálogo:

I. A - Voy a llamar por teléfono.

B - ¿Sabes dónde hay una cabina?

A - Sí.

En la segunda, una persona aborda a otra en la calle y le pregunta:

II. A - ¿Sabe dónde hay una cabina?

B - En la esquina siguiente.

En la situación uno, A expone su intención de llamar por teléfono y B le responde con una pregunta cuya respuesta es una confirmación. La misma pregunta en la situación dos es una petición de información que requiere una respuesta elaborada; para interpretarla correctamente, B ha deducido que A posiblemente pregunta por un teléfono, y teniendo en cuenta la calle en donde se encuentran, lo indica el que está más próximo.

Para que la comunicación se produzca, B debe comprender la intención de la pregunta expresada por A, y generar una respuesta adecuada; ambos procesos dependen de la situación concreta en la que se encuentran los interlocutores.

En el proceso de comprensión, el oyente realiza inferencias utilizando conocimientos sobre el lenguaje, el tema del que se habla, la situación física en donde se produce el diálogo, etc., que le permiten dar una interpretación a la frase pronunciada por su interlocutor. La comunicación pone en juego grandes cantidades de conocimiento de naturaleza muy variada, y es necesario utilizarlo tanto para interpretar el significado como para producir la respuesta adecuada.

El uso del lenguaje es múltiple como lo demuestran los siguientes ejemplos:

+ Este brazo siempre te echará una mano.
+ 2000: una odisea de Olivetti.
+ No deje que los demás paguen por usted. Puede costarle caro.

En cada uno de estos casos, el significado de la frase no se obtiene haciendo una "interpretación literal" de la misma, por el contrario, la ambigüedad, es decir, el doble sentido se explota a fondo y en algunos casos se resuelve con la imagen que complementa el mensaje. Así por ejemplo, en la primera frase se utiliza un sentido figurado y viene acompañada con un emblema de la Cruz Roja, indicando que esta institución está dispuesta a brindar ayuda. La segunda hace referencia al título de una conocida película futurista, mientras que la última viene subtitulada con un "Haciendo cada vez más cerca" adquiriendo el tono de amenaza sobre el contribuyente.

¿Qué procesos mentales se ponen en juego para comprender el lenguaje? ¿Qué conocimiento es necesario para ser capaz de entender su significado?

Estas preguntas sitúan el enfoque de la Inteligencia Artificial en el tratamiento del lenguaje natural: el estudio del lenguaje como un proceso de comunicación con el objetivo de hacerse entender al interlocutor.

Los ejemplos que hemos visto no son extremos, aún en dominios simples se presentan numerosos casos de interpretación del

sentido literal.

Por ejemplo, una pregunta normal a una base de datos sería:

"Quiero saber cuántos vendedores que trabajen en la planta de zapatería ganan más de 100 000 pesos".

Para responder correctamente, un sistema "inteligente" debe ser capaz de:

1. Interpretar el deseo "quiero saber" como una pregunta.

2. Detectar la presuposición "hay vendedores en la planta de zapatería" con la que se ha realizado la pregunta. Si el sistema ignora esta presuposición (como ocurre en muchas bases de datos) la respuesta "ninguno" no sería muy informativa en el caso de no existir vendedores en dicha planta, o no existir la planta.

De la misma forma, si un usuario de un sistema operativo pudiera preguntar a un menú de ayuda "¿sabes cómo borrar un archivo?", esperaría algo más que un "sí".

Si reflexionamos acerca de los ejemplos anteriores podemos puntualizar algunos de los problemas que hemos resuelto de forma consciente o inconsciente:

- Determinar el significado de las palabras.
- Agrupación correcta de las palabras que forman una unidad, por ejemplo, "echar una mano".
- Interpretación del sentido literal utilizando:
  - Conocimiento sobre instituciones existentes y su función, por ejemplo, la "Cruz Roja".
  - Conocimiento sobre el "carácter mandatorio" de ciertas instituciones.

Además podemos enumerar otros problemas que caracterizan al lenguaje natural:

+ Ambigüedad en las construcciones sintácticas.

+ Agrupación correcta en el caso de grupos preposicionales y conjunciones.

Estas construcciones son muy frecuentes en el lenguaje escrito; nada mejor para ilustrarlo que el siguiente párrafo de "Alicia en el país de las Maravillas":

- "... Incluso Stigard, el patriótico Arzobispo de Canterbury lo encontró razonable.

- ¿Encontró qué?, preguntó el pato.

- Lo encontró, contestó el ratón, por supuesto que sabes lo que lo significa.

- Sé lo que lo significa cuando yo encuentro algo, dijo el pato, normalmente es un sapo o un gusano, pero la pregunta es ¿qué es lo que encontró el Arzobispo? ...".

En el diálogo anterior podemos percatarnos de las distintas funciones que puede desempeñar una palabra, en esta caso la palabra lo.

Estamos lejos aún de la máquina capaz de entender cualquier diálogo, pero en algunos dominios existen resultados concretos que repasaremos brevemente:

+ Aceptar y responder preguntas que impliquen procesos de búsqueda referentes a información contenida en una estructura de datos de tipo arborecente. Aquí se puede citar a los sistemas BASEBALL, SAD-SAM, etc.

+ Aceptar y responder preguntas sobre información contenida en una base de datos. El sistema de comprensión se utiliza como

interfase para facilitar el acceso a usuarios no especialistas, como por ejemplo el sistema LUNAR, [WOOD72].

+ Un robot que acepta órdenes y las ejecuta en un mundo simple de objetos (cubos, pirámides, bloques). Además comprende y responde a preguntas referentes a su propia actividad, situación actual de los objetos, etc., como el sistema SHRDLU, [WINO72].

+ Dotar a un sistema experto de capacidad de comunicación en lenguaje natural para introducir información o realizar preguntas.

+ Un sistema que resuelve problemas elementales de Álgebra, [BOBR64].

+ Atender a un cliente que desea hacer una reservación de boletos para realizar un viaje, [BOBR77a].

Un sistema automático que sea un interlocutor aceptable ha de contar con conocimiento y mecanismos de análisis e interpretación. El problema reside en que no existe una teoría que explique el proceso de comprensión. La Inteligencia Artificial basada en las investigaciones en lingüística, psicología, lógica, etc. ha aportado modelos computacionales de representación y tratamiento automático del conocimiento que ofrecen soluciones a algunos aspectos parciales de este complejo fenómeno.

1.1.1. Componentes de un sistema de comprensión de lenguaje natural.

De los primeros sistemas a los más recientes hay una escala ascendente de calidad en cuanto a los resultados concretos y extensibilidad de las aplicaciones. El factor determinante en el proceso logrado ha sido la mejora en las formas de expresión y

utilización de tipos específicos de conocimiento.

Hoy día es ampliamente admitido que la inteligencia de un sistema se basa en el conocimiento que éste tiene y que sus resultados dependen en gran parte de la representación y uso que se haga del mismo.

Actualmente se aborda el estudio del lenguaje escrito por niveles de conocimiento, en cada nivel se han desarrollado y se investigan técnicas de representación y utilización. No hay una diferencia clara entre ellos, pero es una buena forma de estructurar el problema.

Distinguiremos tres planos: el sintáctico, el semántico y el pragmático.

El nivel sintáctico expresa la forma de combinar palabras en frases de acuerdo a las reglas del lenguaje. Es el nivel más estudiado y formalizado. Los analizadores sintácticos que se han construido se basan en el desarrollo de técnicas aplicadas en lenguajes formales, con extensiones para capturar los aspectos no expresables mediante gramáticas libres de contexto. En general, los analizadores producen árboles de constituyentes que reflejan la estructura sintáctica de la frase.

Hay dos formas básicas de expresar la sintaxis:

A) declarativa: dando el conjunto de reglas de la gramática. En este tratamiento podemos pensar que definimos ciertos hechos y reglas entre ellos, y entonces el problema se reduce a deducir conclusiones legítimas a partir de esos hechos.

B) procedimental: expresada por el propio analizador. Aquí, la mayoría de los conocimientos son representados en forma de procedimientos; es decir, el conocimiento queda inmerso en el programa.

Las representaciones declarativas tienen como ventaja la modularidad, claridad y extensibilidad de las reglas. Las representaciones procedimentales son más eficaces. La elección se centra, pues, en encontrar un compromiso teniendo en cuenta los requisitos de la aplicación. En todo caso, la sintaxis del lenguaje natural es muy compleja y es necesario contar con herramientas que permitan expresarla de forma incremental.

El nivel semántico es más problemático, ya que empieza por no haber consenso en cuanto a definir el significado. La mayoría de los autores lo hacen en términos de verdad sobre el "contenido" literal de la oración.

La representación de la semántica se hace con dos tipos de formalismos:

A) Basados en la lógica simbólica, por ejemplo,

cálculo de predicados de primer orden,

en donde los hechos se representan como fórmulas lógicas y, a su vez, una colección de fórmulas lógicas proporciona una descripción del universo de discurso. Una de las ventajas de emplear fórmulas lógicas para representar eventos es que la lógica posee un conjunto de reglas de inferencia que pueden utilizarse para derivar, a partir de los hechos que conocemos son válidos, otros eventos que también deben ser válidos.

B) Otros modelos de representación ponen el énfasis en los aspectos de organización y estructura del conocimiento, centrándose en el diseño de mecanismos de inferencia que capturen el tipo de razonamiento "posible" o de "sentido común" que se produce en el procesamiento del lenguaje. Entre éstos encontramos:

- las redes semánticas,
- los libretos (scripts) y los marcos (frames),
- las reglas de producción.

Basados en estos esquemas se han diseñado lenguajes de programación que incorporan el nivel de abstracción adecuado para construir sistemas de comprensión de lenguaje natural. Entre estos lenguajes podemos mencionar a KRL [BOBR77b], basado sobre la teoría de marcos, así como FRL.

El nivel pragmático recoge los aspectos de uso del lenguaje en un contexto. Para entender un texto, un programa debe saber eventos conocidos por hombres "cultos", qué acontecimientos son factibles de suceder y también las leyes del mundo.

Por una parte es necesario relacionar las frases con el texto o diálogo donde se produce, para resolver pronombres, elipsis, frases incompletas, etc. Así por ejemplo, en los diálogos I y II :

DIALOGO I

- ¿Dónde está el jamón?
- En el armario.
- Sácalo.

DIALOGO II

- ¿Dónde está el jamón?

- En el armario.

- Ciérralo.

tenemos que :

1) "el jamón" designa a un objeto concreto al igual que "el armario".

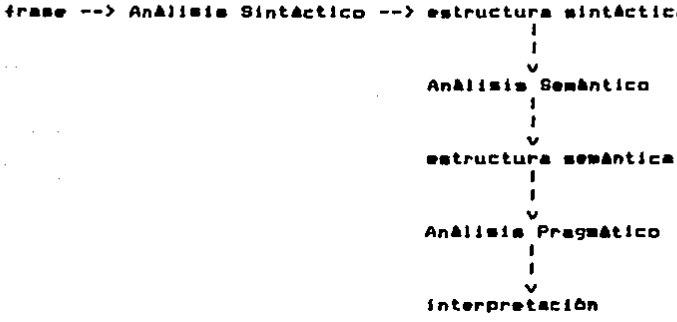
2) "en el armario" es una frase incompleta ligada a la anterior, "el jamón está en el armario".

Sin embargo, el pronombre "lo" se refiere a objetos distintos: en el diálogo I, "lo" se refiere a "jamón", mientras que en el II se refiere a "armario".

De esta forma hemos visto que se ha presentado un problema que es: ¿cómo encontrar el referente del pronombre?

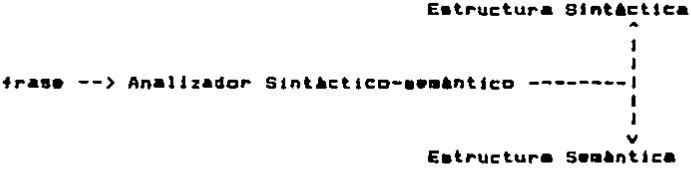
Es interesante darse cuenta del por qué la información pragmática es tan importante. El lenguaje natural es un medio eficiente para transmitir rápidamente una gran cantidad de información. La razón de esta eficiencia es que el escritor supone que el lector es inteligente, y por ende ya sabe varias cosas sobre la materia que se trata. Es innecesario que le dé información que ya sabe: es suficiente que le dé el tema de la comunicación, los hechos concretos y las anomalías a la sucesión normal de acciones. Todos los eventos que son factibles de suceder no se mencionan y aceptamos ambigüedad si sabemos que el lector no tendrá problemas en encontrar el significado correcto.

El proceso de análisis puede estructurarse en etapas secuenciales, separando los tratamientos. Esta es la opción que tienen numerosos sistemas :



En la primera etapa, a partir de la frase, contando con un diccionario y una gramática, se genera una estructura sintáctica (uno o más árboles de derivación). En la segunda etapa, a partir del árbol de derivación, con el diccionario semántico y un conjunto de reglas se genera una representación del significado "literal" de la frase de entrada, y en la tercera etapa se produce una interpretación con las reglas pragmáticas correspondientes.

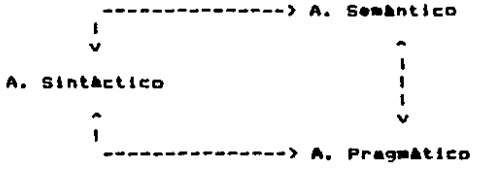
Una segunda posibilidad es realizar el análisis sintáctico y semántico integrados en una sola etapa, aunque el proceso se realice dirigido por la sintaxis.



Una tercera alternativa es la que ofrece la técnica "análi-

sis semántico" en donde la única estructura que se produce es la correspondiente al significado, en términos de la representación del conocimiento. La sintaxis juega un papel secundario. Basados en este concepto se pueden construir, para dominios restringidos, "gramáticas semánticas", desde muy simples hasta más sofisticadas.

Finalmente se puede optar por una estructura modular :



en la que en cada momento actúe sobre los datos el componente que aporte más información para continuar el proceso.

Se escoja una u otra solución, en cada caso hay que plantearse:

1. Conocimiento:

1.a. ¿Qué fuentes de conocimiento son necesarias?

- diccionarios
- gramática
- reglas semánticas
- reglas pragmáticas
- memoria

1.b. ¿Cómo se utilizan? (búsqueda, "caza de patrones", inferencia,...)

1.c. ¿En qué momento intervienen? (estrategia de control)

1.d. ¿Qué interacción debe haber entre ellas?

2. ¿Qué representación de la información se va a utilizar? : uniforme o especializada para cada tratamiento (se plantea el problema de mayor adecuación a costo de creciente complejidad en el sistema).

3. ¿Qué debe generar el analizador? : una(s) estructura(s) sintáctica(s) y/o una estructura semántica.

4. ¿Qué estrategia de análisis se va a seguir?:

4.a. Determinista o no-determinista, y, en este último caso, que es el más actual, ¿se empleará una técnica de exploración en profundidad o en paralelo?

4.b. ¿Se hará un análisis descendente, ascendente o una mezcla de ambos?

4.c. ¿Se analizará un texto de izquierda a derecha o comenzando por palabras clave?

Las decisiones que se tomen en este punto incidirán fundamentalmente en la eficacia del sistema (memoria y tiempo de análisis).

5. ¿Qué flexibilidad se va a tener?, ¿se aceptarán únicamente frases correctas o se admitirán también frases coloquiales?

6. ¿Qué extensibilidad/modificabilidad tendrá la solución adoptada (a otra aplicación dentro del mismo campo, a otros dominios,...) ?

1.2. Descripción global.

1.2.1. Objetivos y relación con otros trabajos.

Este trabajo es una aproximación a la comprensión del EspaÑol por computadora. La razón fundamental para escribir el programa es experimentar técnicas para poder construir sistemas prácticos que entiendan lenguaje natural, y, de esta manera, incrementar la capacidad de comunicación entre las computadoras y los humanos.

El proyecto está "inspirado" en el trabajo de Winograd, SHRDLU, [WINO72] en el sentido de que utilizamos el universo de aplicación propuesto por él. Sin embargo, las ideas y el método que tratamos en la realización del sistema son originales. El dominio particular en el que se trabaja consiste en un robot simulado, con una mano y un ojo, capaz de manipular objetos de juguete (bloques y pirámides) en una mesa. El robot responde realizando la sucesión de acciones necesarias para llevar a efecto una orden dictada por el usuario. También tiene la habilidad de contestar preguntas acerca de la situación de los objetos.

Este sistema, que fue su tesis doctoral en MIT, estaba dividido en varios componentes : sintáctico, semántico, deductivo y generador de respuestas. Para el componente sintáctico diseÑó e implantó un lenguaje: Programar, por medio del cual realizaba dicho análisis basado en una gramática sistémica, [HALL67], en donde se especificaban las características de cada unidad sintáctica.

El razonamiento era en base a una serie de procedimientos escritos en el lenguaje Micro-planner, [SUSS70]. Es decir, la

representación era procedimental. El conocimiento sobre el mundo estaba contenido en procedimientos; estos procedimientos estaban concebidos como pequeños programas que "sabían cómo hacer" tareas específicas. Su modelo estaba basado en la suposición de que las ideas básicas de programación tales como procedimientos, iteraciones, recursividad, etc., son fundamentales para el proceso cognocitivo, y, en particular, para la teoría del lenguaje, [WIND73]. En su sistema uno de los más útiles principios de organización fue la representación de la mayoría del conocimiento en base a procedimientos escritos en micro-planner. En la sección 5.3. haremos algunas comparaciones entre SHRDLU y nuestro sistema.

Posteriormente se hicieron nuevas implantaciones de este sistema utilizando técnicas alternativas. La primera fue una traducción a Conniver, otra se hizo en Lisp, [WINS77]. En general se está de acuerdo en que ambos casos aumentaron la transparencia. Curiosamente, el trabajo original se hizo en micro-planner para demostrar la utilidad de este lenguaje; lo útil no fue el lenguaje "per se", pero en cambio sí lo fue la sintaxis de micro-planner, por su estilo heterárquico, en el cual el control ejecutivo se distribuye a través del sistema [WINS77].

También se ha escrito una versión de SHRDLU en Español, elaborada por M.F. Verdejo [VERD75], como su proyecto de investigación doctoral en la Universidad de Paris VI. El programa lo elaboró en un micro-planner escrito en PL/I que era lo estrictamente necesario para el funcionamiento del sistema. El componente de razonamiento era en base a cálculo de predicados.

El propósito de este trabajo es implantar el sistema SHRDLU en Español, pero utilizando nuevas técnicas. El lenguaje utilizado para la implantación del sistema es el lenguaje de programación PROLOG (en particular, PROLOG86 [MICR85] es el dialecto que utilizamos). Otro de los objetivos de este trabajo es verificar si realmente PROLOG es un lenguaje que facilita la escritura de programas para la comprensión del lenguaje natural.

El programa se ejecuta en una microcomputadora PC-compatible.

1.2.2. Descripción del Sistema

Podemos decir que el sistema está dividido en tres partes principales que son las siguientes :

- un analizador gramatical,
- un componente semántico, y
- un sistema de resolución de problemas.

A grandes rasgos el sistema se puede esquematizar en la forma mostrada en la figura 1.1. Las funciones de cada parte son las siguientes:

1.- El analizador gramatical es el encargado del análisis sintáctico de las oraciones. El analizador es dirigido por metas (top-down) no - determinista, por lo cual utiliza una búsqueda en profundidad y vuelta atrás automática (backtracking), ayudándose fuertemente de las facilidades que brinda PROLOG para ello.

Se cuenta con una serie de reglas que nos especifican el conjunto de oraciones " bien-formadas". Estas reglas tienen asociadas a ellas ciertos predicados semánticos utilizados para la

representación del conocimiento.

2.- La siguiente parte es la encargada de tratar con la representación del conocimiento y con los aspectos semánticos. Este conocimiento se representa en base a predicados PROLOG. Utilizando la " base de datos " de PROLOG podemos dar de alta, o eliminar ciertos hechos, según sean nuestras necesidades.

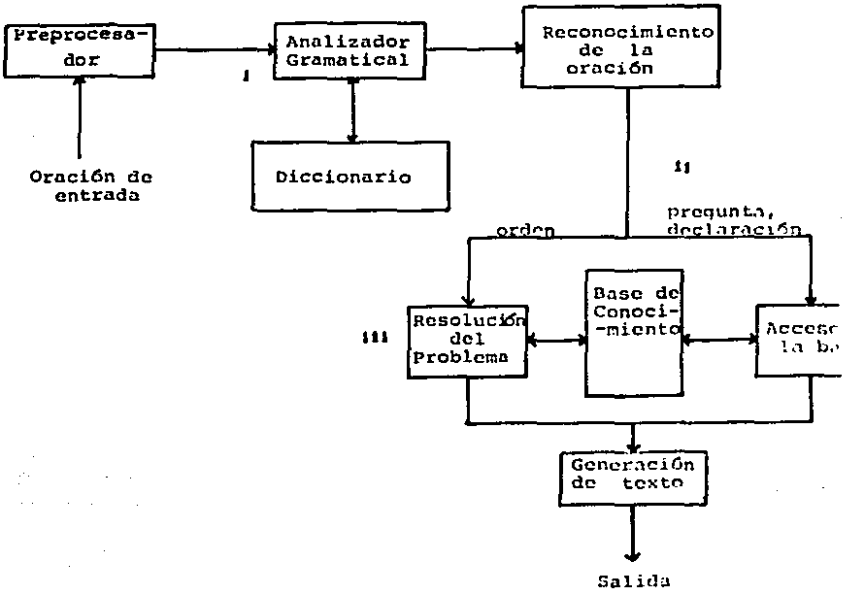


Figura 1.1. Esquematización del sistema.

En esta sección están incluidos predicados encargados de guardar parte de la historia de los objetos que han intervenido en el diálogo y predicados que utilizan esta información para resolver conflictos acerca de cuál es el objeto al que se está refiriendo un pronombre, o para saber a qué objeto se refiere una oración.

3.- Un método de resolución de problemas. El robot, para poder llevar a cabo la orden que el usuario desea, debe realizar una serie de movimientos. El problema a resolver consiste entonces en encontrar esta sucesión de movimientos. El modelo usado para encontrar dicha sucesión es " planeación no-lineal hacia atrás ", (STRIPS, sección 4.2.2.1), [NILS80]. Esta última parte está baseda en el sistema WARPLAN, [KLUS85].

1.2.3. Operación

Inicialmente no hay objetos, de tal forma que el usuario puede decir cuáles figuras desea que existan en el universo diciendo su forma, su color y su posición (ya sea en la mesa o sobre algún otro objeto). El tamaño de los objetos así como su posición en la mesa los genera el sistema aleatoriamente.

El usuario también puede dictar órdenes para modificar la situación del universo. El sistema responde realizando la sucesión de acciones necesarias para pasar de la situación actual a la final. Para obtener esta sucesión de acciones utiliza la parte encargada de la resolución de problemas. Asimismo, el usuario puede definir cuáles objetos le pertenecen.

Cuando se modifica la situación del universo aparece un

diagrama ilustrando esta situación para que el usuario pueda fijarse en ella.

La mesa de nuestro universo la hemos dividido en nueve cuadrados iguales como se muestra en la figura 1.2.; en cada una de estas partes solamente puede estar un objeto como máximo. El diagrama que aparece ejemplificando al universo está dividido en tres planos, uno por cada corte longitudinal de la mesa; el primero es el que corresponde a la parte trasera de la mesa, el segundo el que corresponde a la parte media y el último a la parte frontal. Este diagrama se ilustra en la figura 1.3.

Cada plano contiene cuatro "alturas", pues a lo más pueden estar cuatro figuras colocadas una sobre otra (esta restricción se debe al diagrama y no al manejo interno).

En el lugar que ocupa cada objeto aparece una letra codificando la forma y el color del mismo. Esta codificación también aparece en el diagrama. En los lugares vacios aparece una marca "_". En la tabla 1.4 aparece esta codificación.

El siguiente esquema explica la división de la mesa :

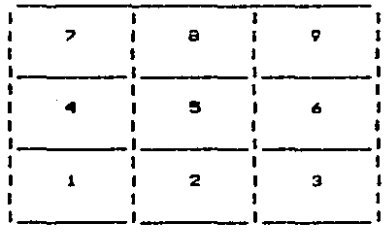


Figura 1.2. Mesa dividida en nueve partes.

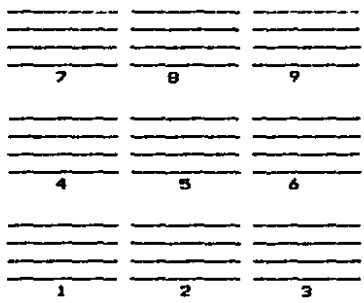


Figura 1.3. Diagramas que aparece en la ejecución del programa. Los números corresponden a las partes de la mesa especificadas en la figura 1.2. Hay cuatro niveles de altura.

color	figura		
	cubo	bloque	pirámide
azul	ca	ba	pa
blanco	cb	bb	pb
rojo	cr	br	pr
verde	cv	bv	pv

Tabla 1.4. Codificación de los objetos en base a su forma y su color.

De igual manera, el usuario puede efectuar preguntas. Estas preguntas podemos catalogarlas en dos tipos : preguntas sobre la situación actual del universo y preguntas acerca del movimiento de objetos.

Entre las preguntas del primer tipo están las relacionadas

con la existencia de objetos, color, posición, cantidad y posesión de ellos. Como preguntas del segundo tipo, el sistema puede responder si ha movido alguna figura, cuándo y por qué lo hizo.

Una observación importante es que el usuario puede utilizar pronombres en sus oraciones, pues el sistema tiene una historia de los últimos objetos nombrados y, al encontrarse un pronombre, utiliza esta información para decidir a cuál figura se refiere.

1.3. Ejemplo de diálogo.

Esta sección contiene un diálogo de muestra con el sistema. En él se realizan acciones y se contestan preguntas sobre un escenario simple que consiste en una mesa, una mano y varios bloques y pirámides.

Las líneas en negrita fueron dadas por una persona, y las líneas en mayúscula son la respuesta de la computadora. Las otras líneas son comentarios que explican algunas de las características importantes que están ocurriendo.

El sistema no cuenta con rutinas de graficación, pues PROLOG86 no brinda facilidades de este tipo, de aquí que se tenga el diagrama que explicamos anteriormente para ejemplificar la situación del universo. Cuando el robot ejecuta alguna acción, realiza y dice las acciones que va efectuando. Estas acciones se explican posteriormente, pero para una mejor comprensión, en este punto omitiremos estas acciones y en su lugar pondremos las figuras correspondientes.

1. existen objetos?

NO, NO HAY

Al principio del diálogo no existen objetos en el universo.

2. quiero una piramide.

EN DONDE QUIERES QUE ESTE?

El usuario dá de alta un objeto diciendo su forma.

3. en la mesa.

Es indispensable conocer su posición en el universo. La preposición en se utiliza como sinónimo de sobre.

NECESITO QUE ME DES SU COLOR

4. azul.

Y también su color.

DE ACUERDO SU TAMANO ES [3, 3, 4].

YA

El sistema genera un tamaño y una posición en la mesa para la figura y dice este tamaño. Responde "YA" después de efectuar alguna acción.

5. quiero un bloque verde sobre la piramide.

?? IMPOSIBLE !!

Sintácticamente esta bien formada esta oración, sin embargo el robot trata de ponerlo y no puede.

6. quiero un bloque verde en la mesa.

DE ACUERDO, SU TAMANO ES [4, 4, 6]

YA

En este caso si se dá toda la información requerida y el sistema no pide más información.

7. quiero un cubo rojo sobre el bloque.

DE ACUERDO, SU TAMANO ES [4, 4, 4]

YA

No hay ambiguedad para saber a qué bloque se refiere el usuario, puesto que sólo hay uno.

8. quiero una piramide blanca sobre el cubo.

DE ACUERDO, SU TAMANO ES [4, 4, 4]

YA

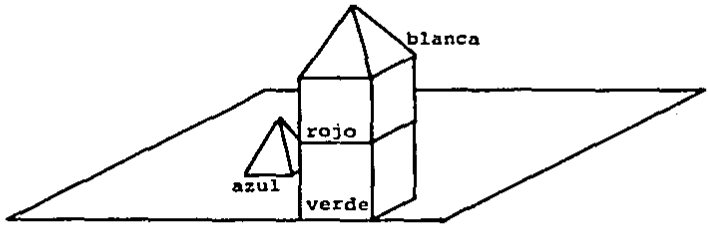


Figura 1.5. Situación inicial.

9. existen piramides?

SI, 2 :

UNA PIRAMIDE BLANCA

UNA PIRAMIDE AZUL

10. levanta la blanca

YA

En este caso se está utilizando un pronombre; el sistema se

ayuda de su memoria para determinar a què objeto se refiere la persona.

11. ponla en la mesa.

YA

Nuevamente utiliza su memoria para resolver conflictos.

12. levanta el bloque.

?? NO SE A CUAL BLOQUE TE REFIERES !

Hay ambigüedad, puesto que existen dos bloques y en el diàlogo no se ha hablado de algùn bloque en particular.

13. levanta el bloque mas grande.

YA

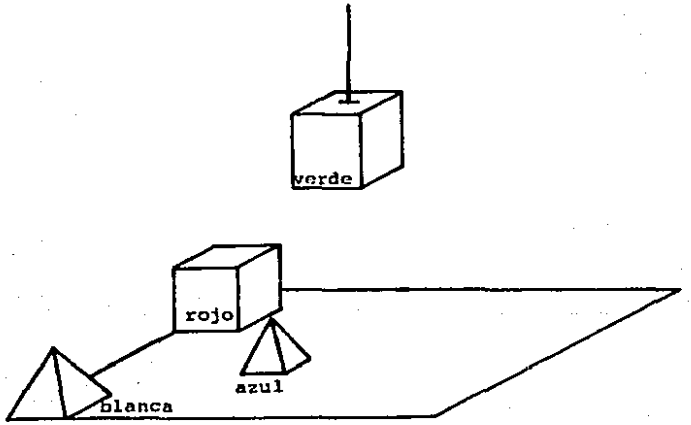


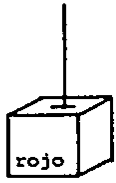
Figura 1.6. "levanta el bloque màs grande."

Usando adjetivos superlativos se puede distinguir a los

diferentes objetos. Para poder realizar la orden, primero debe quitar el cubo que está sobre el bloque.

14. encuentra un bloque menor al que sostienes y levantalo.

YA



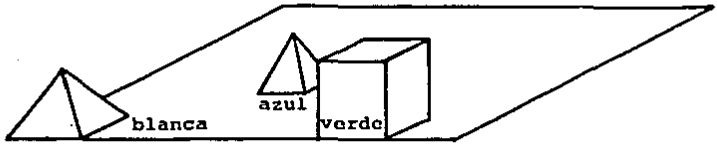


Figura 1.7. "encuentra un bloque menor al que sostienes y levántalo."

En esta orden se utilizan dos pronombres que se refieren a objetos distintos. El sistema se da cuenta que primero debe dejar el objeto que está sosteniendo para posteriormente tomar el bloque que se le ordena que tome.

15. de que color es?

ES DE COLOR ROJO

Nuevamente utiliza su memoria para saber a qué objeto nos estamos refiriendo.

16. ponlo en la mesa.

YA

17. quiero un bloque azul sobre el bloque mas grande.

DE ACUERDO, SU TAMANO ES [4, 4, 5]

YA

18. los objetos azules son mios.

YA

Este es un ejemplo de oración declarativa relacionada con la posesión de objetos.

19. sostienes algun objeto?

NO, A NINGUNO

20. que sostiene a la piramide blanca?

LA MESA

21. esta libre el bloque verde?

NO, SOSTIENE A UN BLOQUE AZUL

22. que objetos estan en la mesa?

UN BLOQUE VERDE

UN CUBO ROJO

UNA PIRAMIDE BLANCA

UNA PIRAMIDE AZUL

23. existen objetos abajo del bloque azul?

SI, 1 :

UN BLOQUE VERDE

24. posee algun objeto?

SI, 2 :

UN BLOQUE AZUL

UNA PIRAMIDE AZUL

25. apilalos en la mesa.

YA

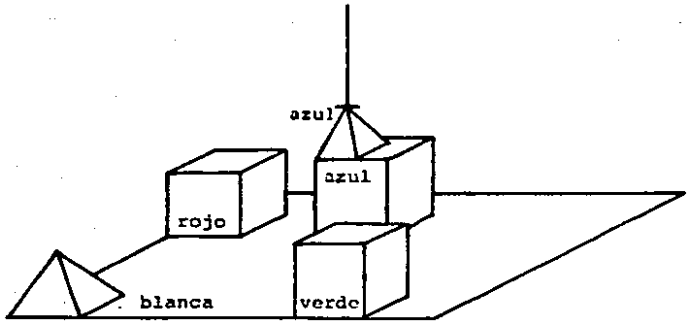


Figura 1.8. "apilalos en la mesa."

El sistema sabe que la pirámide debe ir arriba del bloque.

26. cual es el objeto mas pequeno?

UNA PIRAMIDE AZUL

27. pon el bloque azul sobre el cubo.

YA

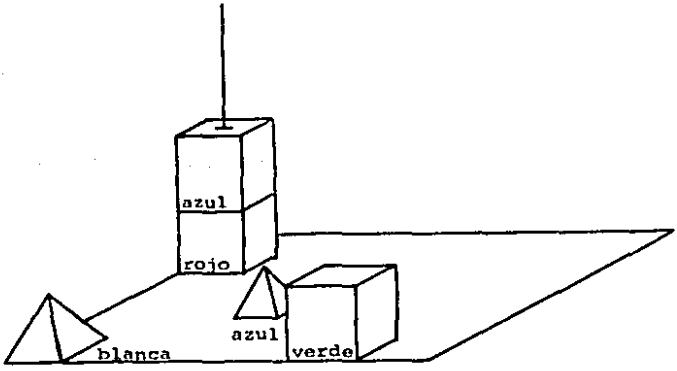


Figura 1.9. "Pon el bloque azul sobre el cubo."

28. apila el cubo y el bloque verde sobre el bloque azul.

YA

Al resolver el problema que presenta esta orden el sistema tarda algunos instantes, puesto que es un problema "difícil".

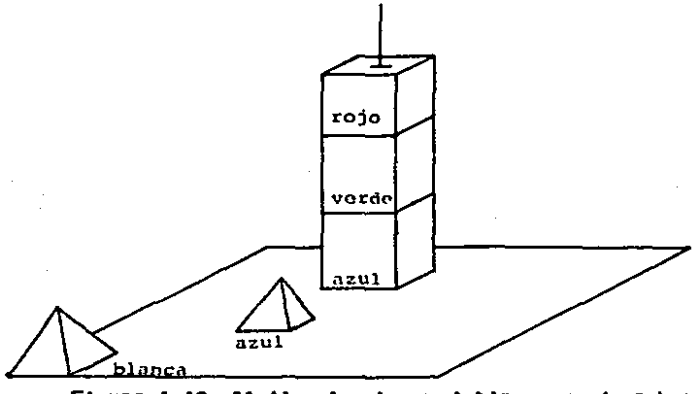


Figura 1.10. "Apila el cubo y el bloque verde sobre el bloque azul."

29. has movido el cubo?

SI

Efectivamente lo he movido! se acuerda de este hecho pues guarda una historia de los movimientos que ha realizado.

30. cuando?

CUANDO IBA A PONER A UN CUBO ROJO

SOBRE UN BLOQUE VERDE

Y A PONER A UN BLOQUE VERDE

SOBRE UN BLOQUE AZUL

El sistema explica la orden que estaba obedeciendo cuando efectuó dicho movimiento.

31. por qué lo hiciste?

PARA DESPEJAR A UN CUBO ROJO

El sistema recuerda las "submetas" que debió resolver.

32. por qué?

PARA PONER A UN BLOQUE VERDE

SOBRE UN BLOQUE AZUL

Dice la siguiente submeta que resolvió.

33. por qué?

PARA LEVANTAR A UN CUBO ROJO

34. por que?

PARA PONER A UN CUBO ROJO

SOBRE UN BLOQUE VERDE

35. por que?

PORQUE TU ME LO ORDENASTE

El último movimiento lo hizo para efectuar la orden del usuario.

36. son mias las pirámides?

SI, I :

UNA PIRAMIDE AZUL

37. puedes levantar un bloque?

SI PUEDO

Efectivamente el robot es capaz de levantar bloques, por eso responde afirmativamente. En este caso no realiza ningún movimiento.

38. adios.

HASTA LUEGO !

CAPITULO 2.

GRAMATICA Y ANALISIS SINTACTICO

2.1. Introducción.

Es importante determinar cuándo decimos que un programa entiende lenguaje natural, pueden darse varias definiciones. Desde un punto de vista, un programa entiende un texto si lo puede traducir a una representación interna y si además es capaz de resolver, usando esta representación, problemas que alguna persona pudiera resolver después de leer el texto, [PITR77].

Para entender un texto (de acuerdo a esta definición), un programa necesita varios tipos de información. Como hemos mencionado con anterioridad, la primera etapa de este proceso consiste en un análisis sintáctico de la oración; este proceso, a su vez, requiere de una gramática que describa la forma de las oraciones "bien formadas" del lenguaje.

Una gramática está compuesta por una serie de reglas que especifican cuáles son las oraciones válidas. Estas reglas pueden ser muy complejas para poder abarcar una gran cantidad de casos. Para procesar una oración es necesario tener un analizador; debido a la complejidad que pueden tener las reglas no es posible contar con un método para construir un analizador dada una gramática arbitraria.

Un tipo de gramáticas para las cuales existen algoritmos eficientes de análisis son las gramáticas libres de contexto. Con estas gramáticas podemos describir la mayoría de los lenguajes de programación y de hecho se utilizan en los métodos de construc-

ción de compiladores. Desafortunadamente, los lenguajes a que dan origen este tipo de gramáticas son muy restringidos y por sí solos no son adecuados para tratar con el gran conjunto de oraciones que pueden formarse en un (subconjunto de un) idioma.

Es así como surge la necesidad de contar con mecanismos más poderosos para reconocer oraciones de un idioma, y, si además queremos reconocer oraciones eficientemente, debemos tener la capacidad de incorporar un mayor conocimiento al analizador. Entre los mecanismos para dotar de conocimiento a los analizadores tenemos los siguientes :

i) Redes de transición aumentadas (ATN por sus siglas en inglés), estas redes son similares a una máquina de estados finitos, en donde se ha añadido un conjunto de acciones a los arcos que definen las transiciones entre los estados; estas acciones se realizan al pasar de un estado a otro. Más aún, una subred puede llamarse recursivamente; Esta recursividad es necesaria para manejar frases tales como:

la pirámide que está sobre el cubo que está sobre el bloque

Un sistema en el que se han usado es el sistema LUNAR [WOOD72].

ii) Gramáticas semánticas, que son gramáticas libres de contexto en donde las reglas se elaboran de acuerdo a funciones sintácticas y semánticas. Con ello se logra que una vez realizado el proceso de análisis, el resultado puede analizarse de inmediato, sin necesidad de procesarlo nuevamente. Como ejemplo podemos mencionar el sistema SOPHIE [BROW74]. A continuación presentamos

una gramática que es una versión simplificada de la usada en el sistema LADDER [HEND78], este sistema es una interfase a una base de datos que contiene información acerca de navios de la marina :

```
S              -> what is SHIP-PROPERTY of SHIP ?
SHIP-PROPERTY  -> the SHIP-PROP | SHIP-PROP
SHIP-PROP      -> speed | length | draft | beam | type
SHIP           -> SHIP-NAME | SHIP2 | the fastest SHIP2 |
                  the biggest SHIP2
SHIP-NAME      -> Kennedy | Kitty Hawk | Constellation | ...
SHIP2          -> COUNTRYS SHIP3 | SHIP3
SHIP3          -> SHIPTYPE LOC | SHIPTYPE
SHIPTYPE       -> carrier | submarine | rowboat
COUNTRYS       -> American | French | British | Russian | ...
LOC            -> in the Mediterranean | in the Pacific | ...
```

Notemos como esta gramática no contiene categorias sintácticas no-terminales tales como SUSTANTIVO o PREDICADO, si no que usa categorias semánticas tales como SHIP y LOC.

En nuestro caso, debido a lo restringido que es el universo de acción en que se trabaja, se puede hacer una combinación de estos conceptos. Asi, existe una gran similitud entre una gramática libre de contexto y una gramática semántica, pues las categorias semánticas corresponden a las sintácticas. Esto es debido a que el universo de discurso está completamente delimitado, evitando con ésto las distintas interpretaciones que las palabras pueden tener. Por ejemplo, el verbo tomar pudiera referirse a la acción de ingerir un liquido o a la acción de sujetar algún

objeto. En nuestro universo no hay confusión puesto que este verbo solamente puede corresponder a la última interpretación.

Por lo tanto elaboramos una gramática libre de contexto para describir el conjunto de oraciones que pueden formarse en nuestro universo de objetos.

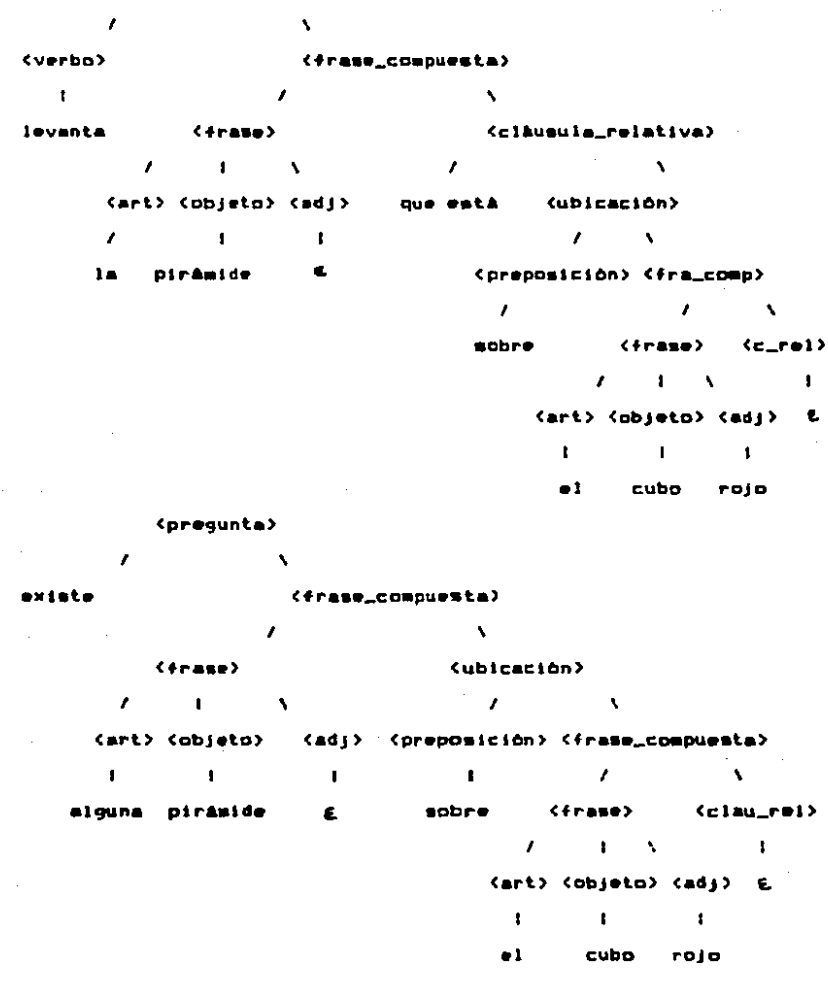
2.2. Una gramática para el español.

Para poder determinar cuáles son las oraciones aceptadas por el sistema, hemos escrito una gramática para un subconjunto del Español. La gramática que se utilizó fue escrita de manera empírica, se hizo un conjunto de oraciones típicas y en base a estas oraciones se elaboraron las producciones de la gramática. Los tres tipos de oraciones que tenemos (declarativas, imperativas e interrogativas) se basan en el mismo tipo de análisis , solamente cambia el sentido que se le da a éste.

Un subconjunto de la versión final de la gramática utilizada se presenta en el apéndice A.

En las oraciones relacionadas con objetos, podemos utilizar pronombres para referirnos a una figura citada anteriormente, o, en su defecto, decir la forma de la misma (cubo, pirámide, etc.) seguida de adjetivos. Al final se da, opcionalmente, la ubicación del objeto utilizando una cláusula relativa.

A continuación presentamos dos ejemplos de análisis de oraciones, la primera imperativa : "levanta la pirámide que está sobre el cubo rojo", y la segunda interrogativa : "¿existe alguna pirámide sobre el cubo rojo?". Notemos la semejanza de los análisis.


<oración>
<verbo>
<frase_compuesta>
levanta
<frase>
<cláusula_relativa>
<art>
<objeto>
<adj>
que está
<ubicación>
la
pirámide
ε
<preposición>
<fra_comp>
sobre
<frase>
<c_rel>
<art>
<objeto>
<adj>
ε
el
cubo
rojo
<pregunta>
existe
<frase_compuesta>
<frase>
<ubicación>
<art>
<objeto>
<adj>
<preposición>
<frase_compuesta>
alguna
pirámide
ε
sobre
<frase>
<clau_rel>
<art>
<objeto>
<adj>
ε
el
cubo
rojo

2.3. Método empleado.

Una gramática nos sirve para especificar el conjunto de oraciones bien formadas que pertenecen a un lenguaje. Es así como un sistema capaz de reconocer oraciones de este lenguaje debe contar con una gramática para poder determinar cuáles son las oraciones válidas. El analizador sintáctico es la parte del sistema encargada de dicho análisis.

PROLOG ofrece una gran ayuda para escribir analizadores sintácticos puesto que, como veremos a continuación, primero podemos escribir una gramática arbitraria y, posteriormente, escribir el analizador sintáctico correspondiente basándonos en esta gramática.

2.3.1. Representación del problema en Prolog.

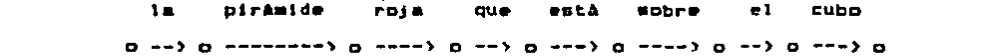
La oración de entrada que hace el usuario se almacena en forma de lista, y, a partir de esta lista, se realiza el análisis sintáctico.

Cabe mencionar que en PROLOG una lista se representa entre corchetes [ ], con sus elementos separados por comas. Además, una lista no-vacía L puede identificarse con el patrón [H | T], en donde H es el primer elemento de L y T es "todo lo demás". Por ejemplo, si L es la lista [a, b, c] entonces H = a, y T = [b, c].

La lista de entrada se somete primero a un análisis morfológico para descomponer, o eliminar, algunas palabras: por ejemplo, los verbos con pronombres se dividen en verbo y en pronombre, los infinitivos se convierten a imperativos, etc.

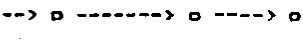
Existen varias formas para representar la estructura de una lista, nosotros usaremos una que nos conducirá a las reglas de la gramática.

Una oración en forma de lista puede representarse por medio de una gráfica, en donde cada nodo de la gráfica corresponde a la frontera entre dos simbolos terminales consecutivos, y la arista que une a dos nodos corresponde a un simbolo terminal. Un ejemplo seria la gráfica siguiente:



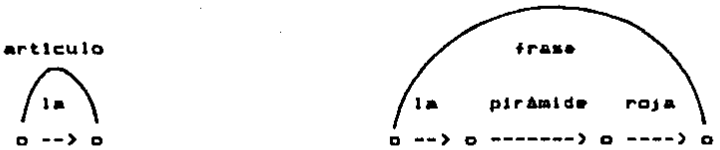
Puede darse el caso de que algunas aristas consecutivas tengan un significado, dando sentido al lado derecho de una producción, como por ejemplo

que es una sucesión de una arista, constituye un articulo, mientras que las aristas

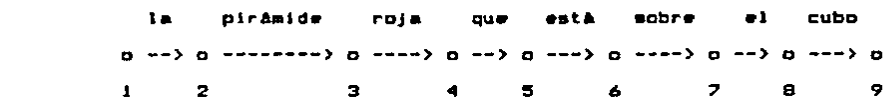


forman una frase.

Para describir estas sucesiones de aristas con significado usaremos una arista que una los nodos inicial y final de la sucesión; a esta arista la etiquetaremos con el nombre del simbolo no-terminal asociado a la producción. Ejemplo :



Para tratar gráficas en un programa, les daremos a los nodos un nombre único, por ejemplo, numerándolos :



De tal forma que podemos representar una gráfica como un conjunto de aristas, estas aristas se representarán por medio de cláusulas. Una cláusula es una expresión de la forma :

a :-- b1, b2, ... , bm .

donde el símbolo ":--" es el símbolo de implicación y expresa que la meta a se satisface si todas las metas bi se satisfacen. En caso de que m sea igual a cero, tendremos que a es una meta verdadera, es decir, un hecho; en esta situación omitiremos el símbolo ":--".

Las cláusulas que usaremos especifican el nombre de la arista y los nombres de los nodos que une, e.g.

```
la(1,2).
pirámide(2,3).
cubo(8,9).
```

Notemos cómo cláusulas etiquetadas con símbolos no-terminales se pueden derivar a partir de cláusulas correspondientes a símbolos terminales. Por ejemplo la palabra la representa un articulo, de aquí que podemos decir que existe un articulo

entre el nodo 1 y el nodo 2. Por tanto, podemos tener la cláusula articulo(1,2) indicando este hecho. En general, tendremos un articulo entre el nodo Y y el Z siempre que tengamos la palabra la entre los nodos Y y Z. Este hecho se representa por la cláusula general :

articulo(Y,Z) <-- la(Y,Z).

que se leeria como : hay un articulo del nodo Y al nodo Z, si hay una arista etiquetada como la del nodo Y al Z.

De la misma forma, la regla :

frase(Y,Z) <-- articulo(Y,T), objeto(T,U), adjetivo(U,Z).

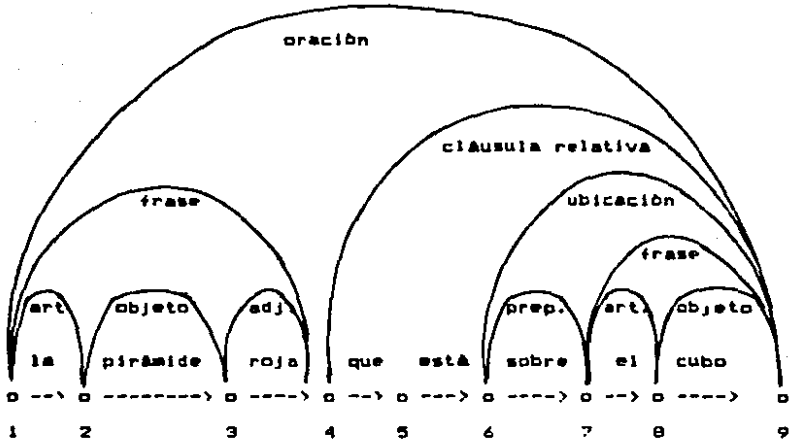
expresa que hay una arista etiquetada como frase del nodo Y al Z, si existen nodos T, U tales que hay una arista de Y a T etiquetada como articulo, y si hay una arista de Y a U etiquetada como objeto y si hay una arista de U a Z etiquetada como adjetivo.

De este modo, una gramática puede darse en forma de cláusulas de PROLOG, cada cláusula corresponde a alguna relación estructural entre una unidad y sus componentes inmediatos, e.g.

frase(Y,Z) :- articulo(Y,T), objeto(T,U), adjetivo(U,Z).

Con esta óptica, reconocer la oración "la pirámide roja que está sobre el cubo" es demostrar que existe una arista del nodo 1 al nodo 9 etiquetada como oración.

El proceso completo produciria la siguiente gráfica :



La importancia de este enfoque es que PROLOG, a partir del conjunto de reglas (gramática de cláusulas) y del teorema a demostrar (cadena del lenguaje) es capaz de producir la derivación/demostración correspondiente.

De hecho la regla :

frase(Y,Z) :- articulo(Y,T), objeto(T,U), adjetivo(U,Z).

se puede interpretar como la definición de un procedimiento de nombre frase con parámetros Y, Z, cuyo cuerpo de instrucciones está formado por las llamadas a tres procedimientos : articulo, objeto y adjetivo.

Es así como una gramática se puede traducir a una gramática en forma de cláusulas. Estas cláusulas forman un programa PROLOG que, a su vez, constituye un analizador sintáctico. La ejecución del programa con una frase concreta como dato produce el análisis

de la misma.

El analizador sintáctico así construido es dirigido por metas ("top-down"), no-determinista con búsqueda en profundidad y vuelta atrás automática.

A manera de ejemplo, supongamos que tenemos la siguiente gramática :

```
% simbolos no-terminales
frase(Y,Z)     :- articulo(Y,T), objeto(T,U), adjetivo(U,Z).
articulo(X,Y)  :- el(X,Y).
objeto(X,Y)    :- cubo(X,Y).
adjetivo(X,Y)  :- rojo(X,Y).
% simbolos terminales
el(1,2).
cubo(2,3).
rojo(3,4).
```

Ahora podemos preguntar de qué nodo a qué nodo existe una estructura frase utilizando la pregunta :

```
frase(X,Z)?
```

a la cual el sistema respondería :

```
X = 1
Z = 4
```

pues efectivamente existe una frase entre el nodo 1 y el 4.

La enumeración de los nodos está implicando un orden fijo (lineal) en la conformación de los constituyentes, lo cual significa una limitación. Sin embargo, los nombres de los nodos no necesitan ser enteros consecutivos; por el contrario, es mejor

derivar nombres únicos a partir de la sucesión de entrada. Así, como el nombre de un nodo tomaremos la lista de aristas que le siguen.

En el ejemplo anterior, el nombre del nodo de la izquierda (nodo 1) será [el, cubo, rojo] y el nombre del último nodo - que corresponde a la sucesión vacía de nodos - será [ ].

De esta manera la gramática toma la siguiente forma :

```
% simbolos no-terminales
frase(Y,Z)     :- articulo(Y,T), objeto(T,U), adjetivo(U,Z).
articulo(X,Y) :- el(X,Y).
objeto(X,Y)    :- cubo(X,Y).
adjetivo(X,Y) :- rojo(X,Y).
% simbolos terminales
el( [el, cubo, rojo], [cubo, rojo] ).
cubo( [cubo, rojo], [rojo] ).
rojo([rojo], []).
```

La cláusula que representa articulo pudiera quedarse sin modificar, pero para poder definir más artículos la expresaremos como :

```
articulo( el, X, Y).
```

Más aún, en lugar de escribir

```
articulo( el, [el, cubo, rojo], [cubo, rojo] ).
```

en donde lo importante para reconocer al artículo el es tenerlo como primer elemento de la primera lista, podemos utilizar una cláusula general :

```
articulo( [el | Resto], [Resto] ).
```

De esta manera una gramática más general toma la siguiente forma :

```
frase(Y,Z)      :- articulo(Y,T), objeto(T,U), adjetivo(U,Z).
articulo( [el | R], R).
articulo( [la | R], R).
objeto( [cubo | R], R).
objeto( [pirámide | R], R).
adjetivo( [azul | R], R).
adjetivo( [rojo | R], R).
```

Una frase reconocida por esta gramática es " el cubo azul ", pues cumple la regla de estar formada por articulo, objeto y adjetivo. Su análisis sería así :

Primero, al analizador le preguntamos si la lista [el, cubo, azul] representa una frase; si este es el caso, después de reconocer esta frase no deben quedar elementos en la lista. Por tanto, debemos efectuar la pregunta siguiente

```
frase( [el, cubo, azul], [])?
```

Con ésto tenemos que Y se instanciaría a la lista [el, cubo, azul] y Z se instanciaría a la lista vacía. La regla que define a frase nos dice que primero debe reconocer un artículo en la lista Y y que después de reconocer a este artículo queda la lista T. La regla que define artículo nos indica que una posibilidad para reconocer un artículo es que la primera palabra de la lista sea el; como efectivamente este es el caso, ha reconocido un artículo y la misma regla nos indica también que después de reconocer a

este artículo la lista que nos queda es la lista original sin el primer elemento, es decir, T se instanciaría a la lista [cubo, azul].

La primera submeta de la cláusula frase se satisface y ahora trata de satisfacer la submeta objeto, por lo cual debe reconocer un objeto en la lista [cubo, azul]. Esta condición sí se cumple y ahora debe reconocer un adjetivo a partir de la lista [azul]. Esta última condición también se satisface, quedando la lista vacía, y, puesto que efectivamente la lista que debería quedar al final era la vacía entonces el análisis termina exitosamente.

Notemos como toda la información necesaria acerca de la gráfica inicial fue proporcionada en la primera llamada. Además la gráfica está ahora en forma implícita: solamente necesitamos - y manipulamos - las dos sucesiones de símbolos terminales.

Otra frase que genera la gramática es "la pirámide rojo", pues también está formada por un artículo, un objeto y un adjetivo, no obstante, existe una incongruencia entre el género del adjetivo y el género del objeto, pero la gramática no detecta esta incongruencia pues no verifica géneros.

2.3.2. Parámetros de símbolos no-terminales

La clase de gramática descrita anteriormente es de poca utilidad práctica, puesto que muy pocas veces analizamos una oración simplemente para aceptarla o rechazarla. Por lo general deseamos obtener una representación de la misma, esta representación debemos hacerla al momento de ir analizando la oración.

Para efectuar este análisis, podemos agregar argumentos a las reglas para que nos permitan expresar aspectos contextuales y generar estructuras en el proceso de análisis. Por ejemplo, la regla :

```
frase(Género, Número, Y, Z) :-
        artículo(Género, Número, Y, T),
        objeto(Género, Número, T, U),
        adjetivo(Género, Número, U, Z).
```

expresa la concordancia en género y número que tiene que haber entre el artículo, objeto y adjetivo que forman una frase. En el ejemplo tendríamos que extender las definiciones en la forma :

```
artículo(femenino, singular, [la | R], R).
objeto(femenino, singular, [pirámide | R], R).
color(femenino, singular, [roja | R], R).
```

Así que la frase "la pirámide roja" tomaría como valor en la llamada, femenino para Género, y singular para Número.

En base a listas, la gramática con argumentos para género y número es la siguiente :

```
frase(Género,Número,Y,Z) :-
        artículo(Género,Número,Y,T),
        objeto(Género,Número,T,U),
        adjetivo(Género,Número,U,Z).

artículo(masculino,singular, [el | R], R).
artículo(masculino,plural,  [los | R], R).
```

```
articulo(femenino,singular,  [la | R], R).
articulo(femenino,plural   ,[las | R], R).
objeto(masculino,singular, [cubo | R], R).
objeto(masculino,plural,   [cubos | R], R).
objeto(femenino, singular, [pirÁmide | R], R).
objeto(femenino, plural,   [pirÁmides | R], R).
adjetivo(_,singular, [azul | R], R).
adjetivo(_,plural,   [azules | R], R).
adjetivo(masculino,singular, [rojo | R], R).
adjetivo(femenino, singular, [roja | R], R).
```

Con esta gramática podemos reconocer frases de la siguiente forma :

```
frase(Género, Número,[las, pirÁmides, azules],[])?
```

y el sistema responderia :

```
Género = femenino
Número = plural
```

Notemos que la regla correspondiente al adjetivo azul no tiene definido un género, puesto que el adjetivo azul puede pertenecer a ambos géneros.

Si ahora deseamos reconocer la frase :

```
frase(Género,Número,[la, pirÁmide,rojo],[])?
```

el sistema responderia

```
##no
```

puesto que al reconocer el articulo, la variable Género se ha instanciado a femenino y posteriormente no encuentra como satisfacer que la palabra rojo sea un adjetivo con género femenino.

2.3.3. Gramáticas de metamorfosis.

La información esencial de la regla anterior es que una frase de un determinado género y número está formada por un artículo, un objeto y un adjetivo, todos del mismo género y número; la otra información, ésto es, la lista dada como entrada a la regla y la lista que resulta después de reconocer la frase (lista vacía), puede ser añadida sin ningún problema a este hecho fundamental.

Para facilitar el uso de esta técnica, se ha desarrollado una notación para escribir reglas de una gramática en PROLOG de una manera legible y fácil.

Esta notación hace más fácil de entender las reglas, pues suprime información que no es interesante. De esta manera podemos escribir reglas en donde no especificamos cuáles son las listas de entrada y de salida.

La notación de estas reglas se basa en la notación de las gramáticas libres de contexto. El lado izquierdo de una regla de gramática PROLOG es el nombre del símbolo no-terminal asociado a ella, y el lado derecho son las partes que lo constituyen. Por ejemplo :

```
frase(Género, Número) -->
        artículo(Género, Número),
        objeto(Género, Número),
        adjetivo(Género, Número).
```

Es importante tener en cuenta que esta notación para las reglas solamente es una forma de simplificar cláusulas ordinarias

de PROLOG. PROLOG86 no maneja este tipo de reglas de gramática, sin embargo, cuenta con un pre-procesador [CLOC84] para traducirlas a cláusulas PROLOG.

La traducción consiste, primero, en que cada símbolo que corresponda al lado derecho de una regla debe traducirse a una cláusula con dos argumentos adicionales, uno para la sucesión dada a la regla y otro para la sucesión resultado (como en las cláusulas de la sección anterior). Segundo, siempre que el lado derecho de una regla se componga de varios símbolos, debe arreglarse de tal forma que los argumentos adicionales reflejen el hecho de que la sucesión dejada por un símbolo forma la entrada del siguiente. Finalmente, los argumentos también deben expresar el hecho de que la cantidad de palabras ocupadas por toda la regla sea la misma cantidad de palabras que ocupan todos los símbolos a la derecha de la "-->".

Estos criterios nos aseguran, por ejemplo, que la regla

```
frase --> articulo, objeto, adjetivo.
```

se traduce a

```
frase(Y,Z) --> articulo(Y,T), objeto(T,U), adjetivo(U,Z).
```

Por último, el sistema debe saber cómo traducir aquellas reglas que introducen palabras, es decir, símbolos terminales. Esto involucra insertar palabras a las listas que forman los argumentos de las cláusulas; de tal forma que los símbolos terminales se denotan entre " [] ", por ejemplo, la regla

```
articulo --> [la].
```

se traduce en

```
articulo( [la | X], X ).
```

A un analizador sintàctico, escrito como una colección de reglas de gramàtica PROLOG, lo llamaremos gramàtica de metamorfosis, [COLM75]. (También se le suele llamar gramàtica de cláusulas definidas, [STERB6]).

La gramàtica con la que hemos trabajado tomaria la forma :

```
frase(Género,Número) -->
     articulo(Género,Número),
     objeto(Género,Número),
     adjetivo(Género,Número).
articulo(masculino,singular) --> [el].
articulo(masculino,plural)   --> [los].
articulo(femenino,singular)  --> [la].
articulo(femenino,plural)    --> [las].
objeto(masculino,singular) --> [cubo].
objeto(masculino,plural)   --> [cubos].
objeto(femenino, singular) --> [pirámide].
objeto(femenino, plural)   --> [pirámides].
adjetivo(_,singular)          --> [azul].
adjetivo(_,plural)            --> [azules].
adjetivo(masculino,singular) --> [rojo].
adjetivo(femenino, singular) --> [roja].
```

Como podemos observar, esta notación es más clara de entender. Las gramàticas de metamorfosis bien merecen su nombre puesto que sus reglas deben cambiarse a cláusulas PROLOG. Debe recordarse que la gramàtica es también un programa "disfrazado", y es ejecutable inmediatamente sin ningún esfuerzo adicional por parte del programador.

También es posible tener una representación de la oración que se está analizando. Por ejemplo, si una frase se refiere a un objeto X (pueden ser varios), entonces, de este objeto necesitamos conocer dos cosas: su forma y algún adjetivo que lo califique. La forma y el adjetivo los representamos por medio de cláusulas PROLOG y de esta manera la representación de la frase puede ser una lista con estas cláusulas. Por ejemplo, si estamos hablando acerca de una pirámide roja, entonces podemos asociar una variable X al objeto pirámide; la cláusula que indica que este objeto es una pirámide tendría la forma pirámide(X), y la que indica su color sería color(X,rojo). Por lo cual, la representación de la frase "la pirámide roja" sería la lista [pirámide(X), color(X,rojo)]. Una gramática para reconocer la oración y formar la lista es la siguiente :

```
frase(X, Género, Número, [P | P1]) -->
        artículo(X, Género, Número),
        objeto(X, Género, Número, P),
        adjetivo(X, Género, Número, P1).
```

en este ejemplo, X es la variable asociada al objeto, P es la cláusula que representa la forma del mismo, y P1 la que representa el adjetivo que lo califica. Las definiciones hay que extenderlas en la siguiente forma :

```
objeto(X, femenino, singular, pirámide(X))
        --> [pirámide].
adjetivo(X, femenino, singular, color(X,rojo))
        --> [roja].
```

de esta forma, la representación de la frase "la pirámide roja" sería la lista [pirámide(X), color(X,rojo)], pues la frase se está refiriendo a un objeto X que debe cumplir las cláusulas pirámide(X) y color(X,rojo).

El uso que le demos a esta representación depende de la situación en que estemos utilizando la frase, como veremos en el capítulo 4.

2.3.4. Condiciones.

Hasta ahora, todo lo mencionado en las reglas de la gramática ha ocasionado que se consuma parte de la sucesión (lista) de entrada. Algunas veces es necesario especificar cláusulas de PROLOG que no sean de este tipo, es decir, deseamos intercalar cláusulas que se realicen sin que consuman símbolos terminales. Estas cláusulas se conocen con el nombre de condiciones, aunque no necesariamente lo sean. Las condiciones en las gramáticas de metamorfosis se enmarcan entre llaves " {} " para no confundirlas con los símbolos terminales o no-terminales.

Un ejemplo en donde es útil usar condiciones es en la parte del diccionario del analizador. Supongamos que deseamos agregar un nuevo objeto, bloque, entonces tendríamos que añadir la regla

objeto(Y,masculino,singular,bloque(X)) --> [bloque].

que, traducida a cláusula PROLOG resulta en

objeto(X,masculino,singular,bloque(X),[bloque | S],S).

que es mucha información a especificar por cada objeto.

Una forma de evitar esto es tener una regla de gramática para expresar información general acerca de los objetos, y tener

la información de las palabras que son objetos en cláusulas PROLOG, por ejemplo :

```
objeto(X,Género,singular,Forma) --> [Objeto],
                    { es_objeto(X,Objeto,Género,Forma) }.
```

donde el predicado es_objeto expresa las palabras que son objetos

```
es_objeto(X,bloque,masculino,bloque(X)).
es_objeto(X,cubo,masculino,cubo(X)).
es_objeto(X,pirámide,femenino,pirámide(X)).
```

La regla de la gramática anterior expresa el hecho de que una estructura objeto está formada por una sola palabra : Objeto, sujeta a la restricción de que esta palabra pertenezca a la colección es_objeto. Este predicado es necesario ponerlo entre llaves puesto que no consume ningún símbolo de entrada. Si no estuvieran las llaves, el pre-procesador de reglas le pondría dos argumentos adicionales al final.

El tratamiento que hemos expuesto hasta ahora todavía no es muy elegante, puesto que al predicado es_objeto deberíamos de agregarle la forma plural de cada objeto. Sin embargo, podemos tener una regla para relacionar las formas singular y plural de los objetos :

"Si N es la forma singular de un objeto, entonces la palabra obtenida al agregar una "s" al final de N, es la forma plural de ese objeto".

Esta regla no es válida en general, por ejemplo, el plural de azul es azules! de cualquier manera, las excepciones existen-

tes se tratan de manera separada.

Utilizando esta regla podemos tener una regla de gramática para reconocer objetos en plural:

```
objeto(X,Género,plural,Forma) --> [Objetos],
                ( plural_a_singular(Objetos,Objeto),
                  es_objeto(X,Objeto,Género,Forma) ).
```

En este ejemplo utilizamos un predicado auxiliar, plural_a_singular, para obtener en el segundo argumento de éste la forma singular del primero, y, una vez obtenida esta forma singular, verificamos que esté en la colección de objetos.

Este tipo de operaciones son las que realizamos en el análisis morfológico de las oraciones.

2.3.5. Consideraciones de Eficiencia

Con las gramáticas de metamorfosis podemos implantar una estrategia de reconocimiento sintáctico muy general con la ventaja adicional de tener gran legibilidad en las reglas, lo cual facilita el uso de ellas para un no-especialista en computación, pues éste puede escribir reglas para una gramática y posteriormente traducirlas rápida y directamente a gramática de metamorfosis.

Los analizadores sintácticos que se pueden generar son dirigidos por metas ("top-down"), no-deterministas con búsqueda en profundidad y vuelta atrás automática. Esta estrategia tiene una complejidad de orden exponencial en tiempo. Esta es la desventaja

de la generalidad y facilidad de la programación con gramáticas de metamorfosis.

Parámetros y condiciones/acciones hacen posible construir una gramática de metamorfosis concisa y legible para subconjuntos razonables de lenguajes naturales, tomando en cuenta sintaxis y semántica.

Una fuente de ineficiencia es la repetición, puesto que dos reglas diferentes pueden empezar con el mismo subconjunto de cláusulas, y, en algunas ocasiones, este subconjunto tendrá que procesarse dos veces. Es posible remediar esta situación modificando las reglas de la gramática, sin embargo, al hacer ésto se pierde claridad, por lo cual es aconsejable hacerlo sólo en la última etapa del trabajo.

Asimismo, una situación que debe eliminarse, al igual que en todos los métodos "top-down", es el tener reglas " con recursividad a la izquierda ", i.e. reglas de la forma : a --> a, a, b; también es posible eliminar este tipo de reglas, (ver [AHO77]).

CAPITULO 3.

REPRESENTACION Y MANIPULACION DEL CONOCIMIENTO

3.1. Introducción.

Al efectuar una comunicación en lenguaje natural con la computadora se presentan dos problemas: uno, escoger una representación interna adecuada a la información, y otro, cómo tener una forma de utilizarla.

Hay varias soluciones posibles, y una vez que se escoge alguna de ellas, el problema consiste en expresar el significado de una oración en la representación interna correspondiente.

La solución por la que hemos optado es expresar el conocimiento por medio de cláusulas PROLOG. Estas cláusulas se almacenan en la base de datos de PROLOG, de tal forma que se pueden agregar o eliminar dinámicamente, según sean las necesidades.

De esta forma el conocimiento lo expresamos en base a predicados. PROLOG facilita la escritura de un sistema de comprensión de lenguaje, pues, como veremos posteriormente, se pueden expresar declaraciones, acciones y preguntas de una manera directa.

Con ayuda de las cláusulas podemos representar objetos, propiedades y relaciones entre ellos en forma de aseveraciones y efectuar preguntas sobre esta información.

En la base de datos de PROLOG almacenamos, entre otras cosas, las definiciones de los objetos que existen, la situación actual del universo, los eventos que han ocurrido y la historia de los objetos nombrados.

Una ventaja de representar el conocimiento en forma imperativa es al tratar con una sucesión de eventos. Un sistema para

demostrar teoremas declarativo no puede aceptar una aseveración de la forma: sobre(b1,b2) (para expresar que el objeto b1 está sobre el objeto b2) como valor. No es un axioma, pues su validez puede cambiar al continuar el proceso. Para volverse un axioma debe expresarse en la forma sobre(b1,b2,estado1), donde estado1 es un símbolo para un estado determinado del mundo. En un sistema de demostración de teoremas declarativo es necesaria una función cuyo valor sea el estado que resulte de poner el objeto b1 sobre el objeto b2.

Este método es el que utiliza el sistema WARPLAN, [KLUS85]. Este sistema dice los movimientos necesarios para pasar de un estado a otro, pero no los realiza, lo cual significa que no transforma la configuración del universo, es decir, siempre se queda en la misma situación.

Una forma de solucionar el problema es el que abordamos en el trabajo, viendo al predicado sobre(b1,b2) como un "estado del mundo", y si este estado se modifica simplemente lo eliminamos de la base de datos de PROLOG. Por tanto, podemos decir solamente sobre(b1,b2) sin necesidad de mencionar estados explícitamente.

3.2. Dominio de aplicación.

Como mencionamos en la sección 1.2.1., el universo en el que se trabajó consiste en un robot simulado, capaz de manipular bloques y pirámides en una mesa, así como también es capaz de contestar preguntas.

3.2.1. El mundo del robot.

En el mundo del robot hay objetos, estos objetos tienen algunas propiedades fisicas y además guardan ciertas relaciones entre ellos. Estas caracteristicas y relaciones se describirán a continuación.

3.2.1.1. Objetos.

Si vemos el cálculo de predicados como un lenguaje formal, podemos pensar que una aseveración es un teorema, y una pregunta una fórmula a probar. Por ejemplo, para especificar que el objeto cubo1 es un cubo, tenemos un predicado de la forma cubo(cubo1).

De esta manera representamos a los objetos y a sus propiedades fisicas, tales como tamaño, color y localización, e.g.

tamaño(cubo1, [6,6,6]).
color(cubo1, rojo).
en(cubo1, [0,10,0]).

Para especificar el tamaño y localización de los objetos usamos un sistema de coordenadas de tres dimensiones, con coordenadas enteras que van del 0 al 30 en las tres dimensiones. El origen [0,0,0] es la esquina izquierda frontal de la mesa. Suponemos que los objetos no pueden rotar, de tal forma que su orientación es la de los ejes coordenados. El tamaño de los objetos se expresa con una lista de tres elementos: [Frente,Ancho,Alto], y estas dimensiones no pueden ser mayores a diez unidades. El tamaño de una pirámide se expresa en base al tamaño del bloque que la circunscribe.

La mesa la hemos dividido en nueve cuadrados de dimensiones iguales, de tal forma que en cada cuadrado puede haber, a lo más, un objeto; es decir, pueden estar nueve objetos en contacto con la mesa como máximo. La ubicación de los objetos se hace en base a una lista de la forma [X,Y,Z] que son las coordenadas de la esquina frontal izquierda inferior del objeto.

3.2.1.2. Relaciones.

Hay algunas relaciones de lugar entre los objetos. Estas relaciones pueden cambiar al efectuarse algún movimiento. Hemos hecho la suposición de que un bloque puede soportar, a lo más, a otro objeto directamente encima de él; por supuesto que una pirámide no puede sostener ninguna figura.

Las relaciones que tenemos son las siguientes :

sosteniendo(X)	- si el robot está sosteniendo al objeto X.
mano_vacia	- si el robot no sostiene ningún objeto.
sobre(X,Y)	- si el objeto X está sobre el objeto Y.
libre(Y)	- si Y no sostiene ningún objeto.
en_la_mesa(X)	- si X está en (sobre) la mesa.
arriba(X,Y)	- si X está arriba de Y (no necesariamente en forma directa).
abajo(X,Y)	
enfrente(X,Y).	
izquierda(X,Y).	

No tenemos relación atrás(X,Y) puesto que un objeto X está

atrás de otro objeto Y si Y está enfrente de X, y ya tenemos el predicado enfrente, de aquí que el predicado atrás se define por la cláusula :

atrás(X,Y) :- enfrente(Y,X).

Sin embargo, si tenemos las dos relaciones arriba y abajo pues se tratan de manera distinta.

3.2.1.3. Acciones.

Los eventos en nuestro mundo son acciones realizadas por el robot. Tenemos cuatro acciones fundamentales :

levanta(X)	- levantar un objeto X de la mesa.
deja(X)	- poner el objeto que se está sosteniendo en la mesa.
apila(X,Y)	- poner el objeto que se está sosteniendo sobre el objeto Y.
quita(X,Y)	- quitar el objeto X de Y (lo levanta).

Como veremos en la parte de resolución de planes, estas acciones necesitan ciertas condiciones para poder efectuarse, y, asimismo, ocasionan que se modifique el estado del mundo.

Las acciones anteriores están formadas, a su vez, por tres acciones primitivas! estas acciones primitivas son las que se realizan "al más bajo nivel" y son :

mueve_mano(Posición)	- mueve la mano (y lo que tenga) a una determinada Posición del sistema cartesiano.

toma(X)	- hace la indicación de que el objeto X se moverá junto con la mano.
deja_obj(X)	- quita esta indicación.

3.2.2. Eventos y memoria.

Para contestar preguntas acerca de eventos pasados, el programa "recuerda" partes "selectas" del árbol de submetas (ver sección 4.2.2.1.). Hace ésto creando predicados llamados eventos y guardándolos en la base de datos de PROLOG.

En el evento guarda la meta final, que es la que ocasiona el evento, y la serie de pasos realizada para lograrla; en realidad no guarda toda la lista de pasos (e.g. mueve_mano), solamente las submetas mayores (apila y quita).

Esta es la memoria utilizada en las preguntas relacionadas con el movimiento de objetos, cuándo se realiza y las razones que lo causaron.

Para ir diciendo las acciones se fija en los pasos que tuvo que realizar; cuando ya ha dicho el último paso, si se vuelve a preguntar por otra razón, contesta diciendo que el movimiento fue debido a que el usuario así lo deseó.

Una segunda clase de memoria corresponde a los "últimos" objetos que han intervenido en el diálogo

Existe un contexto de discurso local, que cubre el discurso

inmediatamente anterior a una oración; es importante para los mecanismos semánticos tales como la referencia de pronombres. Por ejemplo en las situaciones : "por qué?", "apílalos", "levanta el verde", etc.

Para manejar el discurso local tenemos dos predicados: última_frase(Lista) y último_objeto(Lista), en donde Lista es una lista de objetos, ya sea los nombrados en la última frase, o los últimos objetos que intervinieron en el diálogo; la diferencia es que el predicado última_frase solamente se actualiza cuando se da una oración sin pronombres, a diferencia del predicado último_objeto, que se modifica con toda oración referente a objetos.

También existe un discurso global. Si decimos "levanta la pirámide" y hay tres pirámides, no se puede distinguir a cuál de ellas nos referimos, pero si antes habíamos nombrado a "una pirámide sobre el bloque más grande", entonces la referencia es a esa pirámide. Esto es claro aún si hubiera varias oraciones entre estas dos; por tanto, es un problema diferente al del discurso local y los pronombres. Para guardar el discurso global se tienen tres predicados, almacenados en la base de datos de PROLOG, que se "acuerdan" de los últimos objetos a los que se refirió el discurso; son tres predicados, pues los objetos se guardan en base a su forma. Estos predicados se van actualizando cada vez que sea necesario, es decir, cada vez que en una oración intervienen objetos explícitamente (sin pronombres).

Los predicados que tenemos para guardar el discurso global son los siguientes:

último_bloque(Lista),

último_cubo(Lista) y

última_pirámide(Lista)

en donde Lista es una lista con los nombres de los objetos referidos según su forma.

CAPITULO 4.

ANALISIS SEMANTICO.

4.1. Estructuras Semànticas.

En el anàlisis semàntico empleamos la noción de gramàticas de casos de Fillmore [BRUC75], en el sentido de que el proceso de anàlisis està fuertemente dirigido por las entradas lèxicas asociadas con cada verbo. Es decir, el anàlisis es "expectation-driven", una vez que se localiza el verbo de la oración, puede ser usado para predecir lo que sigue y determinar cuàl es la relación de las frases siguientes con el verbo de la oración. Por ejemplo, si tenemos una oración de la forma siguiente :

" toma .... "

con el sólo hecho de conocer el verbo ya sabemos que debemos buscar (y encontrar) cuàl es el objeto que debemos levantar.

Aquèllas palabras, tales como verbos y preposiciones, que pueden aparecer como indicadores de la estructura, tienen asignadas predicados que realizan el trabajo que serà considerado como el significado de la oración. Estas palabras son la fuerza que dirige al analizador. Ellas establecen las condiciones que dicen "en dònde se coloca" lo que se ha reconocido, y lo que se predice que vendrà; mientras que los nombres, adjetivos y el contexto anterior sirven como base de datos para que los predicados puedan operar en ellos.

El analizador ha sido escrito con especial ènfasis en lo que las palabras individuales pueden comunicar. La siguiente descripción del anàlisis de una oración puede ayudar a aclarar el proceso. La oración es :

"levanta la pirámide que está sobre el bloque más grande."

La primera palabra, "levanta", indica que debe efectuarse una acción sobre algún objeto. Este objeto es el sujeto de la oración, que es imperativa. También sabemos que para realizar la acción debemos formar un plan con la ayuda del sistema de planeación STRIPS (sección 4.2.2.1.).

La siguiente palabra: "la", nos indica que el objeto al que se refiere la oración es uno "distinguido", es decir, un objeto que puede diferenciarse de los demás de alguna manera.

La siguiente palabra: "pirámide", nos dice la forma del sujeto de la oración, que es al que se refiere la acción a realizar.

La siguiente palabra, "que", tiene un efecto funcional en el análisis de la oración e indica que el analizador debe estar preparado para reconocer una cláusula relativa, es decir, alguna relación que satisfaga el objeto al que se refiere la oración.

A continuación viene la palabra "está", e indica que el sujeto satisface una relación de lugar; con ésto, el analizador sabe que debe recibir una preposición de lugar y posteriormente la descripción del lugar en donde se encuentra el objeto.

La siguiente palabra, "sobre", satisface las suposiciones hechas anteriormente. Esta palabra tiene asignada a ella un predicado: sobre(X,Y), en donde X se instancia al sujeto objeto de la oración y Y es el objeto cuya descripción se espera que vendrá a continuación.

La siguiente palabra es "el" y nuevamente nos dice que el segundo objeto que debemos reconocer es un objeto distinguido.

La siguiente palabra: "cubo" indica la forma de este objeto.

La siguiente palabra, "más", es el comienzo de un adjetivo superlativo e indica que el segundo objeto de la oración es distinguido debido a su tamaño, y debe ser el más pequeño o el más grande.

Por último, la palabra "grande" nos dice cuál es el objeto que debemos buscar y tiene asignada a ella un predicado para buscar este objeto.

Con ésto hemos terminado el análisis de la oración y al mismo tiempo hemos extraído de ella su significado.

Este tipo de análisis, aunque obteniendo otra representación, es el que realiza el sistema MARGIE [SCHA73].

Podemos darnos cuenta de la forma en que las palabras han guiado la construcción de este analizador. El objetivo del mismo es encontrar el significado y no la estructura de una oración. El proceso está basado en un análisis de izquierda a derecha, en donde las palabras contribuyen a dicho análisis y el objetivo es formar una lista con estas contribuciones.

Esta estrategia no resuelve los problemas clásicos de ambigüedad y relevancia del contexto, pero da una forma de representar las descripciones que las personas hacen del significado de una oración.

Como ejemplos de las palabras indicadoras de la estructura tenemos las subrayadas en las oraciones siguientes :

toma la pirámide que está sobre el cubo rojo.

ponla en la mesa.

levanta el cubo que está enfrente de la pirámide más grande.

Como mencionamos en la sección 2.3.2., a las reglas de una gramática de metamorfosis podemos agregarles argumentos y condiciones. Con la ayuda de estos argumentos y condiciones podemos dar un significado a la oración que se está procesando y, de esta forma, tener una "traducción dirigida por sintaxis".

Ya vimos que el significado de una frase se puede representar en forma de lista. Por ejemplo, la frase "el cubo rojo" estaría representada por la lista [cubo(X), color(X,rojo)].

Otros argumentos de la correspondiente regla de gramática nos indicarían que la frase se refiere a un sólo objeto, definido, de género masculino.

4.1.1. Cláusulas relativas.

Examinemos una frase más complicada:

"el cubo rojo que sostiene una pirámide".

Al analizar la primera parte de la frase, "el cubo rojo", el analizador formaría la lista vista en el ejemplo anterior. Como todavía no se ha consumido toda la lista de entrada el proceso de análisis continúa, buscando alguna relación que deba cumplir el objeto; en este punto se encuentra las palabras "que sostiene" y entonces sabe que debe encontrar un objeto, puesto que el verbo sostiene es transitivo. Para reconocer el objeto se utiliza recursivamente la regla de gramática correspondiente. Esta vez la regla reconocería la frase "una pirámide" y formaría la lista [pirámide(Y)], de tal forma que la frase completa " el cubo rojo que sostiene una pirámide" quedaría representada por la lista [cubo(X), color(X,rojo), sobre(Y,X), pirámide(Y)].

Notemos como el verbo sostiene tiene asignado el predicado sobre(Y,X), donde Y es el objeto al que se refiere la frase que sigue a sostiene y X es el objeto del que se habla en la primera parte de la frase.

4.1.2. Grupos preposicionales.

Al comparar la frase " el cubo rojo que sostiene una pirámide" con la frase "el cubo rojo al lado de la pirámide" vemos que las cláusulas relativas son muy semejantes a las frases preposicionales en estructura y significado. De hecho su análisis semántico es casi idéntico. La función de una preposición como "al lado" es igual a la función del verbo "sostiene".

Los adjetivos de comparación también se tratan de la misma forma, e.g. "un cubo rojo mayor que la pirámide verde".

4.1.3. Tipos de descripciones de objetos.

Los objetos a los que se refieren las oraciones pueden ser indefinidos o definidos: "un cubo rojo" o "la pirámide". Si la oración se refiere a un objeto indeterminado, cualquiera que cumpla las características mencionadas en la frase es un objeto válido; de estos objetos se toma el "mejor", es decir, aquél que tenga el menor número de objetos encima de él.

Sin embargo, si se utiliza un artículo definido, e.g. "la pirámide" entonces la persona que habla se debe estar refiriendo a un objeto en particular, a una pirámide que espera el sistema conozca. Si sólo hay un objeto que cumpla esta descripción no ha-

problema, el problema se presenta cuando hay varios objetos que lo cumplan y, en esta situación, el sistema debe buscar en su memoria la pirámide más reciente a la que se ha referido el usuario (si es que se ha referido a alguna), si este es el caso, esa debe ser la pirámide a la que se refiere; de otra forma escribe un mensaje de error, diciendo que no sabe a qué pirámide se está refiriendo.

Cuando en una frase se habla de adjetivos superlativos: "el bloque más grande que está en la mesa", no es posible asignar un predicado para expresar el significado de "más grande", puesto que el bloque es más grande con respecto a un grupo de objetos, y no se sabe qué grupo de objetos será hasta que no se haya terminado de analizar la frase completa. De tal forma que los adjetivos superlativos tienen asociados predicados encargados de encontrar el objeto más grande, o más pequeño de un grupo de objetos.

4.2. Representación de las oraciones.

Ya mencionamos que todas las oraciones que se refieren a objetos se representan de igual manera, en forma de lista. El uso que le damos a esta lista depende del tipo de oración que estamos analizando. El sistema actúa en base a esta oración, ya sea respondiendo, realizando alguna acción o guardando cierto conocimiento.

Si la oración termina con un signo de interrogación "?" se trata de una pregunta, para los otros tipos de oraciones se debe dar al final un punto "." El proceso de análisis consiste en

reconocer una estructura sintáctica, oración o pregunta, según sea el caso, a partir de la lista de entrada y consumir todos los elementos de ésta. La primera palabra nos indica el tipo de oración que se está analizando y en base a ella se utiliza la regla correspondiente. si en algún momento no se puede continuar el análisis se busca otra regla alternativa utilizando las facilidades de búsqueda de PROLOG.

Las oraciones declarativas de ambos tipos, ya sea que se defina alguna figura o que se indique que se posee algún objeto, realizan "ellas mismas" las acciones necesarias para llevar a efecto el sentido de la oración. Igual sucede con las preguntas, dentro de la misma regla se dice la respuesta apropiada.

No pasa lo mismo con las oraciones imperativas. Estas oraciones forman un predicado con una lista que contiene el estado del universo al que el usuario desea pasar. Este predicado tiene la forma:

plans([sobre(pirámide2,cubo1)])

para posteriormente resolver este plan realizando las acciones necesarias obtenidas por medio del sistema de planeación STRIPS.

4.2.1. Oraciones declarativas.

4.2.1.1 Creación de objetos.

Inicialmente no hay objetos en el universo, de tal forma que el usuario puede decir los objetos que desea que existan. Es decir, puede definirlos dinámicamente. Para poder hacer ésto debe dar su forma, su color y su ubicación (ya sea en la mesa o sobre

algún otro objeto). Si su posición es en la mesa busca una posición disponible en ella; si es sobre algún otro objeto debe estar libre (y no ser pirámide). Si en la oración dada al sistema, el usuario no menciona el color o la situación geográfica del objeto que está definiendo, el sistema se percata de este hecho y pide la información correspondiente, puesto que es indispensable conocerla. El tamaño de los objetos los genera aleatoriamente.

Al saber la posición del objeto podemos conocer las coordenadas que tendrá en el sistema y la posición que le corresponde en el diagrama que aparece dibujado en la pantalla.

Una vez conocida toda la información, lo único que debemos hacer es "dar de alta" los predicados correspondientes que están en la lista que representa la oración junto con otros predicados, como son los que se refieren al tamaño del objeto, su ubicación y también especificar que está libre (i.e. no sostiene a ningún objeto).

Es importante hacer notar el hecho de que los objetos que se crean no intervienen en la historia del diálogo, es decir, no se guarda ninguna historia de ellos y, por ende, no puede el usuario referirse a ellos por medio de pronombres o por artículos definidos.

4.2.1.2. Posesión de objetos.

El sistema también acepta oraciones que se refieren a la relación de pertenencia de objetos; se podían haber tratado de igual forma otras relaciones tales como gustar, etc.

De esta relación el sistema no sabe nada, excepto lo que se

dice en el diàlogo.

El usuario puede decir cuàles son los objetos que posee, refiriéndose a ellos de una manera definida. Las oraciones pueden ser de la forma :

"yo poseo los cubos rojos",

"la piràmide azul es mia".

Para tratar este caso, primero se forma una lista con los predicados que describen al objeto (s los objetos) que involucra la oración; posteriormente encuentra cuàles son los objetos que cumplen esta descripción, formando una lista con los nombres de ellos, y, por último, esta lista se agrega a la lista que nos indica cuàles son los objetos que posee el usuario. Esta lista de objetos se guarda en la base de datos de PROLOG y al principio del diàlogo es vacia.

En éste, y en todos los casos en que necesitamos encontrar cuàles son todos los objetos que cumplen una descripción (dada como lista de predicados) utilizamos una cláusula PROLOG: findall2, que tiene tres argumentos :

findall2(X,Predicados,Lista).

Este predicado forma una Lista con los nombres de todos los objetos X tales que satisfacen los predicados que están en la lista de Predicados. Estos objetos X se buscan en la base de datos de PROLOG.

4.2.2. Oraciones imperativas.

El sistema acepta órdenes en forma de oraciones imperativas. Estas oraciones pueden referirse a un objeto definido, indefinido o a un pronombre.

Si el objeto al que nos estamos refiriendo es un objeto indefinido, como en la oración "levanta un cubo rojo", entonces, a partir de la lista [cubo(X), color(X,rojo)] formamos una lista con todos los objetos que cumplen los dos predicados. Si la lista es vacía significa que hay algún error y el sistema lo dice; si la lista consiste de un sólo elemento no hay problema, pero si hay más de un objeto el sistema busca cuál es el mejor objeto que conviene tomar, es decir, aquél que facilite la realización de la acción, este objeto es alguno que tenga el menor número de objetos arriba de él.

Si la acción se refiere a un objeto definido, entonces, como se mencionó en la sección 3.2.2., el sistema utiliza su memoria referente al discurso global para determinar cuál es el objeto al que se está refiriendo la oración. Si hay más de un objeto que cumpla la descripción dada y no se ha hecho mención a alguno de estos objetos entonces hay ambigüedad y el sistema la indica.

Por último en las oraciones se pueden utilizar pronombres; para poder tratarlos el sistema guarda una historia de los últimos objetos y, al encontrarse un pronombre, utiliza esta información para decidir a qué figura se refiere.

Un ejemplo del uso de los pronombres es el siguiente :

"¿cuáles objetos están en la mesa?"

- la pirámide azul

"levántala".

Aquí el pronombre la se refiere a la pirámide de la que se habló en la pregunta anterior.

Otros ejemplos:

"toma un bloque mayor al que sostienes y ponlo en la mesa."

En este caso los dos pronombres se refieren a objetos distintos.

"levanta uno mayor que él."

"¿sostienes algún objeto?"

- sí.

- ¿de qué color es?"

En este último ejemplo, aunque no se menciona ningún objeto, es claro que la pregunta se está refiriendo al objeto mencionado en la frase anterior.

Los pronombres se manejan en base a un discurso local. Este discurso local consiste en guardar una lista con los objetos que se nombraron en la última oración, pues los pronombres, a diferencia de los artículos definidos, siempre se refieren a los últimos objetos que se trataron. Cuando se encuentra un pronombre, el sistema revisa en su "memoria a corto plazo" cuál es el objeto al que se está refiriendo el usuario.

Independientemente de la forma en que el usuario se haya referido a un objeto (pueden ser más de uno, en el caso en que la orden sea apilar algunos objetos), el sistema encuentra el nombre del objeto al que se refiere la orden, y entonces el significado de la oración será llevar a efecto una sucesión de acciones para pasar de la situación actual del universo a la situación que desea el usuario.

El sistema responde entonces realizando la sucesión de acciones necesarias para llegar a esta última situación. Para saber cuáles son las acciones requeridas utiliza el sistema de planeación STRIPS [NILS80], [RICH83].

Posteriormente dibuja un diagrama en donde se esquematiza la nueva situación del universo.

4.2.2.1. Strips - Sistema de resolución de planes.

Al ir realizando las acciones, el robot cambia un estado o configuración del mundo en otro. Para determinar cuáles son las acciones a seguir hemos implantado una versión del sistema STRIPS [NILS80]. El mecanismo de resolución de PROLOG se adapta perfectamente a este sistema.

Como dijimos anteriormente tenemos cuatro acciones fundamentales, que son : levanta(X), deja(X), apila(X,Y) y quita(X,Y). Estas acciones ocasionan que se modifique la base de datos del universo.

Cada acción tiene asociados tres componentes:

El primero es una fórmula de precondiciones, que es una

lista de los predicados (hechos) que deben cumplirse para poder llevar a efecto dicha acción. En estos predicados pueden estar variables sin instanciar, el proceso seguido para instanciar variables lo discutiremos posteriormente.

El segundo componente de una acción es una lista de predicados llamada la lista de eliminación. Los predicados (hechos) que estén en esta lista deben eliminarse al llevar a cabo la acción.

El tercer componente es una lista de adición, que es una lista con los predicados que se "vuelven verdaderos" al aplicar la acción. Estos predicados se agregan a la base de datos de PROLOG.

Las cuatro acciones, junto con sus componentes, son las siguientes:

1.- levanta(X)

Precondiciones:	en_la_mesa(X), libre(X), mano_vacia
L. de Eliminación:	en_la_mesa(X), mano_vacia
L. de Adición:	sosteniendo(X)

2.- deja(X)

Precondiciones :	sosteniendo(X)
L. de Eliminación:	sosteniendo(X)
L. de Adición:	en_la_mesa(X), mano_vacia

3.- apila(X,Y)

Precondiciones:	sosteniendo(X), libre(Y)
L. de Eliminación:	sosteniendo(X), libre(Y)
L. de Adición:	mano_vacia, sobre(X,Y)

4.- quita(X,Y)

Precondiciones:	sobre(X,Y),	mano_vacia,
	libre(X)	
L. de Eliminación:	mano_vacia, sobre(X,Y)	
L. de Adición:	sosteniendo(X), libre(Y)	

Para elaborar planes en forma eficiente (no realizando acciones innecesarias), STRIPS trabaja "hacia atrás", es decir, de la meta final hacia la descripción del estado actual. Para llegar a la descripción del estado actual el sistema va generando submetas, y el proceso termina cuando produce una submeta que coincida con los hechos que se dan en el estado actual.

El sistema entonces realiza un razonamiento "hacia atrás" tratando de llegar a la situación actual. STRIPS primero busca cuál debe ser la última acción para llegar a la meta final, y si en el estado actual ya se dan las condiciones que necesita esta acción, simplemente la realiza y ha terminado el plan; en caso contrario, si no se da alguna condición C, busca una acción que tenga en su lista de adición a C, es decir, busca una acción que ocasione que C "se vuelva verdadera", y entonces trata de realizar esta acción, así sucesivamente. El sistema utiliza técnicas (que se explican posteriormente) para no realizar una búsqueda exhaustiva.

Si en algún momento no puede lograr que se dé una condición, entonces efectúa vuelta atrás ("backtracking") regresando al punto anterior y busca aplicar otra acción. Si ya no hay más acciones por aplicar ésto significa que la acción que había

tomado no es la última en realizarse, y entonces busca otra acción.

Cuando aplicamos alguna acción hacia atrás, debemos fijarnos en cuáles predicados (hechos) se siguen dando y cuáles deben eliminarse. Este proceso se conoce con el nombre de regresión.

El proceso de regresión consiste en determinar què sucede con un hecho al aplicar una acción. Pueden suceder tres cosas: que se agregue, que se elimine o que no se modifique. A manera de ejemplo supongamos que queremos aplicar la acción

quita(pirámide2, cubo1).

entonces :

- regresión(libre(cubo1), quita(pirámide2,cubo1)) = verdadero, puesto que el hecho libre(cubo1) se vuelve verdadero al aplicar la acción quita(pirámide2,cubo1). El hecho está en la lista de adición de la acción.

- regresión(mano_vacia, quita(pirámide2,cubo1)) = falso, ya que el hecho mano_vacia desaparece al efectuar la acción quita(pirámide2,cubo1). El hecho está en la lista de eliminación de la acción.

- regresión(libre(bloque1), quita(pirámide2, cubo1)) = libre(bloque1), es decir, el hecho libre(bloque1) no se ve afectado al realizar la acción quita(pirámide2,cubo1).

Es importante destacar que cuando se efectúa la regresión con una acción, los objetos involucrados ya deben estar instanciados, si no lo están, entonces antes de hacer el proceso se

instancian con los nombres de los objetos que están directamente involucrados en el plan. Para saber cuáles son estos objetos lo primero que se hace al entrar al plan es formar una lista con los nombres de los objetos que:

i) menciona el plan,

ii) los objetos que están arriba o abajo de los anteriores, y

iii) el objeto que está sosteniendo.

Todos los demás objetos no intervendrán de ninguna manera en el plan. Si la acción instanciada con un objeto no conduce a la resolución del plan, entonces se efectúa vuelta atrás y se instancia otro objeto.

Al ir realizando acciones hacia atrás e ir agregando nuevos hechos que se van dando, puede suceder que lleguemos a un estado que es imposible de lograrse, por ejemplo, un estado en que tengamos los hechos: mano_vacia y sosteniendo(cúbo1). Para detectar que no se llegue a ningún "estado de contradicción" tenemos el predicado imposs(L), donde L es una lista con hechos que forman un estado imposible de darse. De tal forma que al generar una nueva situación, un requisito para seguir adelante es que esta situación no contenga a ninguna lista L que se encuentre en el predicado imposs(L)! si este es el caso se elimina este camino y se busca otro.

El sistema trabaja de una manera no-lineal, en el sentido de que si un plan consta de más de un hecho a lograrse, no trata primero de satisfacer alguno de ellos y después otro, pues eso

generaría acciones innecesarias, si no que hace parte del trabajo de un plan, parte del trabajo de otro, etc. Para ilustrar este caso, supongamos que tenemos la siguiente situación inicial :

Figura 4.1.

y que la meta es :

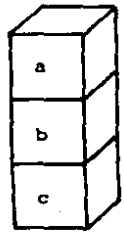


Figura 4.2.

es decir, los hechos a lograrse son : sobre(a,b) y sobre(b,c).

Si fuera lineal trataría de resolver alguno de ellos, lo cual nos llevaría a los siguientes casos:

a) si satisface primero sobre(a,b) :

Figura 4.3.

que ocasiona trabajo extra (quitar otra vez a) para lograr el hecho sobre(b,c));

b) si satisface sobre(b,c) :

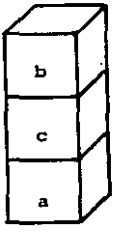


Figura 4.4.

que vuelve más difícil de lograr el hecho sobre(a,b).

La dificultad radica en que los componentes de la meta interactúan entre ellos: al resolver una meta se "destruye" la solución lograda independientemente por la otra. Esto significa

que no debemos buscar soluciones independientes para cada componente del plan.

Nuestro sistema trabaja en forma no-lineal, en el sentido de que primero trata de realizar planes que hagan parte del trabajo de una meta, después parte del trabajo de otra, etc., y luego regresa a trabajar sobre metas anteriores aún no completas.

Al probar una acción la guarda en una lista de acciones, de tal manera que al llegar al estado actual del mundo, y, por consiguiente, al resolver el plan, en esta lista de acciones tenemos la sucesión de pasos que debemos efectuar para llegar a la situación deseada.

Al ir buscando cuáles son las acciones necesarias para lograr todas las submetas se genera un árbol de búsqueda bastante frondoso, entonces se tienen técnicas para eliminar varios de sus nodos. Estas técnicas son las siguientes :

a) verificar metas inconsistentes,

b) analizar los estados anteriores, para ésto, se tiene una lista con los caminos por los que ha tratado para que en ningún caso el sistema vuelva a buscar por uno de ellos nuevamente y así no caiga en ciclos,

c) unificación de figuras sólo directamente involucradas con la meta.

Asimismo, el sistema cuenta con heurísticas para determinar, cuando se necesita aplicar alguna acción, cuál de ellas es la más viable de realizar.

Por ejemplo, si queremos lograr el hecho de estar sosteniendo un objeto X y no lo estamos sosteniendo ya, entonces podemos ó

levantarlo de la mesa ó intentar quitarlo de otro objeto Y. Una heurística que usamos nos dice que si el objeto X está en la mesa, lo primero que debemos intentar hacer es levantarlo de ella, y, si ésto falla, entonces tratar de quitarlo de otro objeto.

Una segunda heurística nos indica que si un hecho ya se da (es verdadero) en el estado actual del universo, entonces no intentemos satisfacerlo por el momento, es decir, lo "retardamos", solamente regresaremos a satisfacerlo en caso de que no podamos aplicar ninguna otra acción.

Una última heurística consiste en que, cuando hay hechos "retardados" y queremos satisfacer alguno de ellos, el primero que debemos lograr es el hecho de que la mano del robot está vacía; esta heurística se basa en la suposición de que el hecho de que la mano del robot esté vacía es un estado que cambia continuamente y entonces será fácil de satisfacer.

4.2.2.2. Un ejemplo de solución.

Supongamos que el estado inicial es el que se muestra en la figura 4.1, y que deseamos llegar al estado de la figura 4.2.

Este último estado se representa por las dos metas: sobre(a,b) y sobre(b,c). Un diagrama con las submetas que va generando se representa en la figura 4.5.

De estas metas busca cuál debe ser la última en lograrse; si esta última meta fuera poner b sobre c (nodo 3) entonces las precondiciones de esta operación, que son: sobre(a,b), sostenien-

do(b) y libre(c) deberían ser verdaderas antes de ejecutarla, sin embargo, este estado contiene una situación imposible de darse, puesto que no se puede estar sosteniendo a b si b sostiene a a.

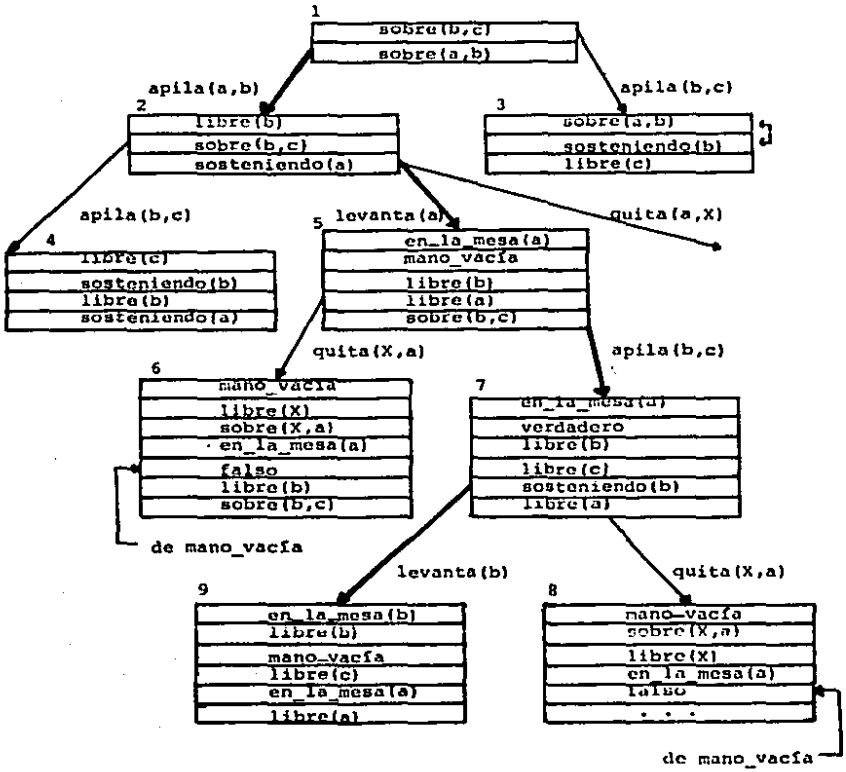


Figura 4.5.

De aquí que esta trayectoria puede eliminarse y el plan continúa por el nodo 2.

Este nodo, a su vez, tiene tres submetas que se pueden satisfacer antes de realizar la acción apila(a,b). Supongamos que escoge libre(b) como la última meta a lograr, puesto que este hecho es verdadero en el estado actual, lo retarda, ésto es, no lo intenta satisfacer por el momento, solamente regresará a él si todo lo demás falla.

El nodo 4 también se elimina, puesto que si deseamos lograr al último la meta sobre(b,c) entonces entre las condiciones que deben darse tenemos los dos hechos: sosteniendo(b) y sosteniendo(a) que es una situación inconsistente. De tal forma que la última submeta a lograrse debe ser sosteniendo(a) para lo cual podemos efectuar dos acciones para lograrla: quita(a,X), para algún objeto X, y levanta(a). Una heurística nos dice que si un objeto está en la mesa, como es el caso de a, y lo queremos sostener entonces lo primero que debemos intentar hacer es levantarlo de la mesa, por este motivo tomamos la acción levanta(a) que produce el nodo 5.

En el nodo 5 tenemos cinco hechos que deben darse. El primero es en_la_mesa(a), y puesto que es verdadero en el estado inicial, lo retarda; lo mismo sucede con las submetas mano_vacia y libre(b).

El siguiente hecho es libre(a) que para lograrlo debemos efectuar la acción quita(X,a); sin embargo, al hacer regresión de los hechos con esta acción obtenemos que regresión(mano_vacia, quita(X,a)) = falso. Puesto que la meta falso no se puede satis-

facer, esta rama puede eliminarse.

El único hecho que nos queda es sobre(b,c) que genera un estado consistente, por tal motivo la trayectoria seguirá por allí. La acción para lograr este hecho es apila(b,c) que genera el nodo 7.

Las primeras tres submetas de este nodo : en_la_mesa(a), libre(c) y libre(b) ya se dan en el estado actual y, por ende, las retarda. La siguiente submeta es el hecho libre(a) que lo satisface efectuando la acción quita(X,a). Sin embargo, al regresar los hechos de este nodo obtenemos la situación mostrada en el nodo 8 que corresponde a un estado inconsistente, pues contiene a la meta falso, que se obtiene al regresar el hecho mano_vacia.

La última meta a lograrse es sosteniendo(b), que la puede lograr quitando b de algún objeto o levantándolo de la mesa. Puesto que b está en la mesa lo intenta levantar y obtiene el nodo nueve.

En este nodo ya se dan varios hechos, como son en_la_mesa(b), libre(b), mano_vacia, libre(c) y en_la_mesa(a) . El único que no se dá es el hecho libre(a) y para lograrlo debemos efectuar la acción quita(X,a), obteniendo el nodo 10.

Este nodo corresponde a un estado inconsistente puesto que al regresar el hecho mano_vacia con la acción quita(X,a) obtenemos falso.

La figura 4.6. ilustra la última parte del proceso.

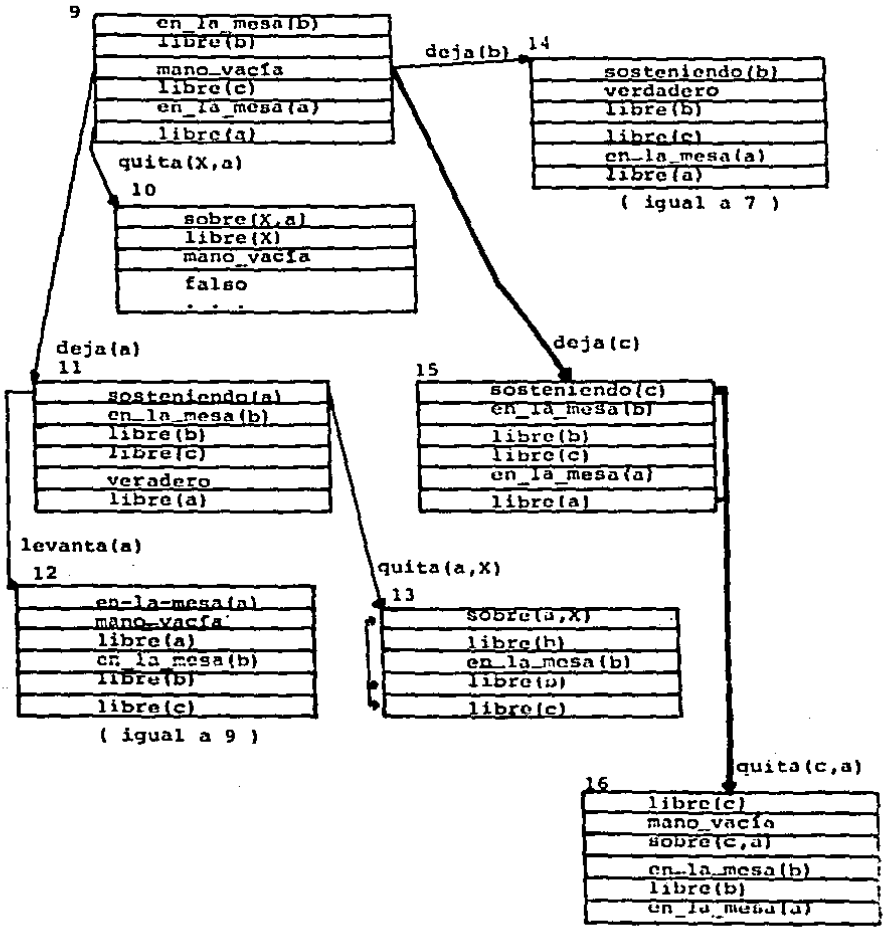

9
en_la_mesa(b)
libre(b)
mano_vacía
libre(c)
en_la_mesa(a)
libre(a)
deja(b)
14
sosteniendo(b)
verdadero
libre(b)
libre(c)
en_la_mesa(a)
libre(a)
(igual a 7)
quita(X,a)
10
sobre(X,a)
libre(X)
mano_vacía
falso
. . .
deja(a)
11
sosteniendo(a)
en_la_mesa(b)
libre(b)
libre(c)
verdadero
libre(a)
deja(c)
15
sosteniendo(c)
en_la_mesa(b)
libre(b)
libre(c)
en_la_mesa(a)
libre(a)
levanta(a)
12
en-la-mesa(a)
mano_vacía
libre(a)
en_la_mesa(b)
libre(b)
libre(c)
(igual a 9)
quita(a,X)
13
sobre(a,X)
libre(b)
en_la_mesa(b)
libre(b)
libre(c)
quita(c,a)
16
libre(c)
mano_vacía
sobre(c,a)
en_la_mesa(b)
libre(b)
en_la_mesa(a)

Figura 4.6.

Como ya se terminaron los hechos intentamos re-satisfacer alguno de los que habiamos retardado. Al efectuar acciones hay que levantar y dejar objetos, de aqui que un estado que cambia continuamente es que la mano del robot esté vacia; en vista a ello una heuristica que utilizamos nos dice que el primero de los hechos retardados que debemos tratar de re-satisfacer es mano_vacia. Para lograr este hecho tenemos dos acciones: apila(X,Y) y deja(X). Otra heuristica nos dice que lo primero que debemos intentar es dejar el objeto X sobre la mesa. Para efectuar esta acción hay un objeto sin instanciar. Si instanciamos X con a obtenemos el nodo 11; de este último el 12, que es igual al 9, y el nodo 13 que es inconsistente. Por tanto se efectúa vuelta-atrás e instancia X con b, obteniendo el nodo 14, que contiene al 4. Nuevamente realiza vuelta-atrás instanciando X con c y obtiene el nodo consistente 15.

En este nodo solamente faltan por darse los hechos libre(a) y sosteniendo(c). Para lograr el hecho libre(a) debe efectuar la acción quita(X,a). Para instanciar X verifica primero si a sostiene algún objeto, y, puesto que a sostiene a c entonces instancia X con c. Al realizar esta acción obtenemos el nodo 16 que ya coincide con los hechos del estado inicial, lo cual significa que ya encontramos la solución. Esta solución la obtenemos siguiendo las lineas sombreadas en orden inverso y es la siguiente : [quita(c,a), deja(c), levanta(b), apila(b,c), levanta(a), apila(a,b)].

4.2.3. Preguntas.

Las preguntas las podemos clasificar en tres categorías: preguntas relacionadas con la capacidad del robot para efectuar movimientos, preguntas relacionadas con la situación actual del universo y preguntas relacionadas con el movimiento pasado de objetos.

Las preguntas del primer tipo son de la forma :

"puedes levantar una pirámide?"

que se manejan como si se tratara de una oración imperativa, solamente es necesario transformar la forma infinitiva del verbo, levantar, en imperativa, levanta. La única diferencia con las oraciones imperativas radica en que en las preguntas no se realiza el plan, es decir, no se efectúa acción alguna.

Las preguntas relacionadas con la situación actual del universo son similares a las cláusulas relativas. Esta semejanza se ve claramente en la pregunta "¿qué cubo rojo está en la mesa?", que está relacionada con la oración "un cubo rojo que está en la mesa". El sistema puede contestar la pregunta construyendo la descripción de "un cubo rojo que está en la mesa", que es [cubo(X), color(X,rojo), en_la_mesa(X)], y utilizando el predicado PROLOG findall2 forma una lista con los objetos que cumplan esta restricción, estos objetos son el resultado de la pregunta. Si la pregunta es "cuántos" en lugar de "qué", el sistema realiza un proceso idéntico, pero responde diciendo un número.

Existen varios tipos de preguntas de este género, pero todas se trabajan de la misma manera: se forma una lista con la

descripción de los objetos relacionados, se utiliza el predicado PROLOG findall2 para encontrar una lista con los nombres de los objetos que satisfacen esta descripción y se genera una respuesta apropiada.

En las preguntas del último tipo, las relacionadas con el movimiento pasado de objetos, el foco de atención es un evento y no un objeto. Al preguntar al robot si ha movido algún objeto, busca en su memoria si existe algún evento en donde haya intervenido este objeto. Si este es el caso se acuerda de este último evento al que se hizo referencia guardándolo en la base de datos de PROLOG.

Este último evento se utiliza si el usuario pregunta cuándo se movió el objeto, pues en el evento aparece la meta que se quería satisfacer cuando se hizo el desplazamiento y esta meta es la que se describe en la respuesta que se da. También este último evento interviene en las preguntas relacionadas con la razón del por qué se efectuó algún movimiento: en el evento se guarda una lista con las acciones necesarias para lograr el plan, estas acciones son las respuestas a preguntas sucesivas del por qué se efectuó una acción. Al acabarse esta lista de acciones, si se vuelve a preguntar el por qué se efectuó algun movimiento (en este caso es el primero) la razón es debido a que el usuario así lo quizo, y ésto es lo que se responde.

CAPITULO 5

RESULTADOS

En este capitulo presentamos más muestras del diálogo que se puede llevar a cabo con el sistema para que, de esta manera, el lector tenga más elementos de juicio. En la última parte del capítulo hacemos una comparación entre el sistema de Winograd y el nuestro.

El sistema forma una representación interna de la oración que se le proporciona y posteriormente utiliza esta información para demostrar que realmente ha entendido lo que el usuario desea comunicar. Si el usuario ordena alguna acción, el robot obedece demostrando con ésto que realmente ha comprendido la orden; si, por el contrario, el usuario hace alguna pregunta, el robot responde de manera apropiada demostrando nuevamente que es capaz de comprender. Al principio de cada diálogo se forma un universo para trabajar en él.

Puesto que ya explicamos previamente las acciones que puede realizar el robot, ahora si presentaremos las sucesion de movimientos que va efectuando el robot al obedecer una orden dada por un usuario.

También pondremos dibujos para ilustrar el desarrollo del diálogo.

5.1. EJEMPLO 1.

1. quiero un bloque verde en la mesa.
   DE ACUERDO, SU TAMANO ES [4, 4, 5]

2. quiero un bloque blanco sobre el bloque.

   DE ACUERDO, SU TAMANO ES [4, 4, 6]

3. quiero un cubo rojo sobre el bloque blanco.

   DE ACUERDO, SU TAMANO ES [4, 4, 4]

4. quiero un cubo azul sobre el cubo.

   DE ACUERDO, SU TAMANO ES [4, 4, 4]

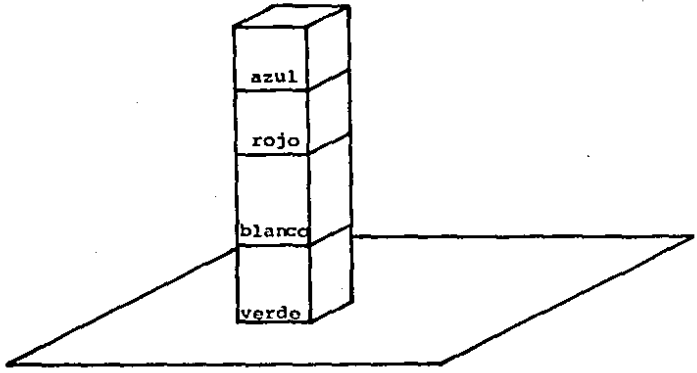


Figura 5.1. Situación inicial del universo.

5. levanta el bloque verde.

   QUITA(CUBO2, CUBO1)

   MUEVE_MANO([12, 12, 19])

   TOMA(CUBO2)

   LEVANTA_MANO([12, 12, 50])

```
DEJA(CUBO2)
        MUEVE_MANO( [2, 22, 4] )
        DEJA_OBJ(CUBO2)

QUITA( CUBO1, BLOQUE2 )
        MUEVE_MANO( [12, 12, 15] )
        TOMA(CUBO1)
        LEVANTA_MANO( [12, 12, 50] )

DEJA(CUBO1)
        MUEVE_MANO( [2, 2, 4] )
        DEJA_OBJ(CUBO1)

QUITA( BLOQUE2, BLOQUE1 )
        MUEVE_MANO( [12, 12, 11] )
        TOMA(BLOQUE2)
        LEVANTA_MANO( [12, 12, 50] )

DEJA(BLOQUE2)
        MUEVE_MANO( [22, 2, 6] )
        DEJA_OBJ(BLOQUE2)

LEVANTA(BLOQUE1)
        MUEVE_MANO( [12, 12, 5] )
        TOMA(BLOQUE1)
        LEVANTA_MANO( [12, 12, 50] )
YA
```

Para levantar el bloque verde, primero debe quitar los objetos que están sobre él. Cuando se crean los diferentes obje-

tos, se les asigna un nombre único, que es el que se utiliza para referirse a ellos dentro del sistema. Este nombre se crea a partir del la forma de la figura y de un número entero que comienza en uno y que se va incrementando a medida que se van generando más objetos. Para levantar una figura, el robot debe mover su mano al centro de ella.

Después de efectuar esta orden la situación del mundo queda como se muestra en la siguiente figura :

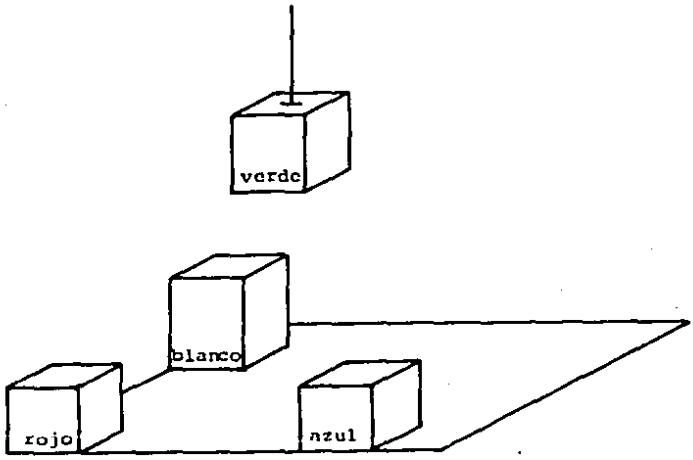


Figura 5.2. "Levanta el bloque verde."

6. ¿Sostienes algún objeto?

SI,

UN BLOQUE VERDE

7. existen objetos enfrente del bloque blanco?

SI, 2 :

UN CUBO AZUL

UN CUBO ROJO

8. has movido el bloque blanco?

SI

9. cuando?

CUANDO IBA A LEVANTAR A UN BLOQUE VERDE

10. cuantos objetos hay en la mesa?

3 :

UN BLOQUE BLANCO

UN CUBO AZUL

UN CUBO ROJO

11. pon el bloque que sostienes en la mesa.

DEJA(BLOQUE1)

MUEVE_MANO([12, 12, 5])

DEJA_OBJ(BLOQUE1)

YA

12. existen cubos?

SI 2 :

UN CUBO AZUL

UN CUBO ROJO

13. apilalos sobre el bloque blanco.

LEVANTA(CUBO1)

MUEVE_MANO([2, 2, 4])

```
        TOMA(CUBO1)
        LEVANTA_MANO( [2, 2, 50] )

APILA( CUBO1, BLOQUE2)
        MUEVE_MANO( [22, 2, 10] )
        DEJA_OBJ(CUBO1)

LEVANTA(CUBO2)
        MUEVE_MANO( [2, 22, 4] )
        TOMA(CUBO2)
        LEVANTA_MANO( [2, 22, 4] )

APILA( CUBO2, CUBO1 )
        MUEVE_MANO( [22, 2, 14] )
        DEJA_OBJ(CUBO2)

YA
```

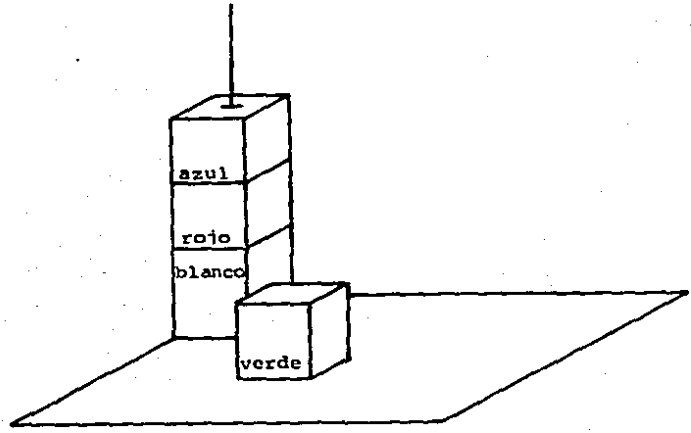


Figura 5.3. "apilalos sobre el bloque blanco."

14. levanta el bloque blanco.

```
QUITA( CUBO2, CUBO1 )
        MUEVE_MANO( [22, 2, 14] )
        TOMA(CUBO2)
        LEVANTA_MANO( [22, 2, 50] )

DEJA(CUBO2)
        MUEVE_MANO( [22, 2, 4] )
        DEJA_OBJ(CUBO2)

QUITA( CUBO1, BLOQUE2 )
        MUEVE_MANO( [22, 2, 10] )
        TOMA(CUBO1)
        LEVANTA_MANO( [22, 2, 50] )

DEJA(CUBO1)
        MUEVE_MANO( [2, 2, 4] )
        DEJA_OBJ(CUBO1)

LEVANTA(BLOQUE2)
        MUEVE_MANO( [22, 2, 6] )
        TOMA(BLOQUE2)
        LEVANTA_MANO( [22, 2, 50] )
YA
```

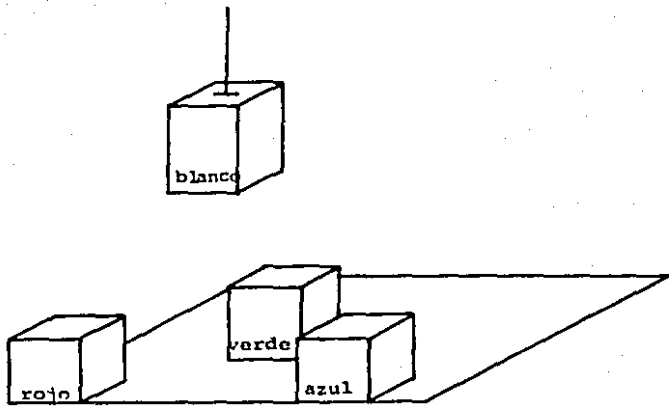


Figura 5.4. "levanta el bloque blanco."

15. existen objetos a la derecha del cubo rojo?

    SI, 2 :

        UN BLOQUE VERDE

        UN CUBO AZUL

16. levanta el cubo.

    DEJA(BLOQUE2)

        MUEVE_MANO([22, 2, 6])

        DEJA_OBJ(BLOQUE2)

    LEVANTA(CUBO2)

        MUEVE_MANO([2, 22, 4])

        TOMA(CUBO2)

        LEVANTA_MANO([2, 22, 50])

    YA

El robot sabe que primero debe dejar el bloque que sostiene. También se da cuenta que el cubo al que se refiere la persona es el cubo nombrado en la oración anterior.

```
17. ponlo sobre el cubo.
    APILA( CUBO2, CUBO1 )
          MUEVE_MANO( [2, 2, 8] )
          DEJA_OBJ(CUBO2)
  . YA
```

El sistema sabe a qué objetos se refieren el pronombre lo y el artículo el.

La situación final aparece en la figura 5.5.

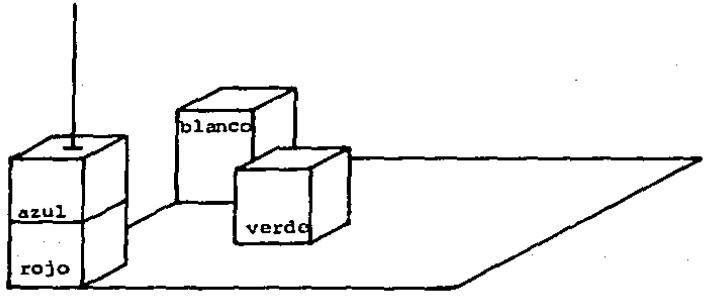


Figura 5.5. Estado final del diálogo del ejemplo 1.

5.2. EJEMPLO 2.

1. quiero un cubo rojo en la mesa.

   DE ACUERDO, SU TAMANO ES [6, 6, 6]

2. quiero una piramide roja en la mesa.

   DE ACUERDO, SU TAMANO ES [4, 4, 4]

2. quiero una piramide blanca en la mesa.

   DE ACUERDO, SU TAMANO ES [4, 4, 6]

4. quiero un bloque verde en la mesa.

   DE ACUERDO, SU TAMANO ES [6, 6, 8]

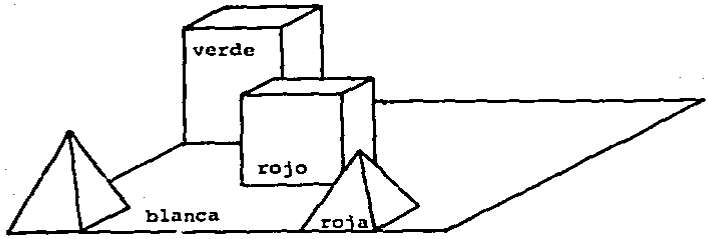


Figura 5.6. Situación inicial del universo.

5. pon la piramide blanca sobre el bloque mas grande.

```
LEVANTA(PIRAMIDE2)
        MUEVE_MANO( [2, 2, 6] )
        TOMA(PIRAMIDE2)
        LEVANTA_MANO( [2, 2, 50] )

APILA( PIRAMIDE2, BLOQUE1 )
        MUEVE_MANO( [23, 3, 12] )
        DEJA_OBJ(PIRAMIDE2)

YA
```

6. que sostiene a la piramide blanca?

UN BLOQUE VERDE.

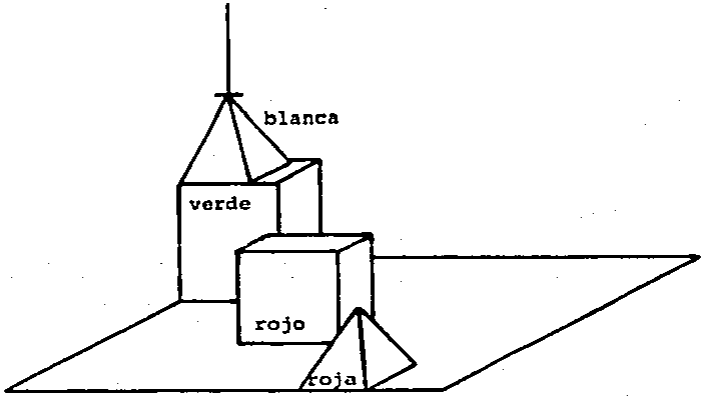


Figura 5.7. "pon la pirámide blanca sobre el bloque más grande."

```
7.  esta libre el bloque verde?
    NO, SOSTIENE A UNA PIRAMIDE BLANCA

8.  existen objetos arriba del cubo?
    NO, NO HAY

9.  pon el bloque verde sobre el cubo.

    QUITA( PIRAMIDE2, BLOQUE1 )
          MUEVE_MANO( [23, 3, 12] )
          TOMA(PIRAMIDE2)
          LEVANTA_MANO( [23, 3, 50] )

    DEJA(PIRAMIDE2)
          MUEVE_MANO( [2, 2, 6] )
          DEJA_OBJ(PIRAMIDE2)

    LEVANTA(BLOQUE1)
          MUEVE_MANO( [23, 3, 8] )
          TOMA(BLOQUE1)
          LEVANTA_MANO( [12, 12, 50] )

    APILA( BLOQUE1, CUBO1 )
          MUEVE_MANO( [13, 13, 14] )
          DEJA_OBJ(BLOQUE1)
    YA

10. has movido el cubo?
    NO

11. has movido alguna piramide?
    SI
```

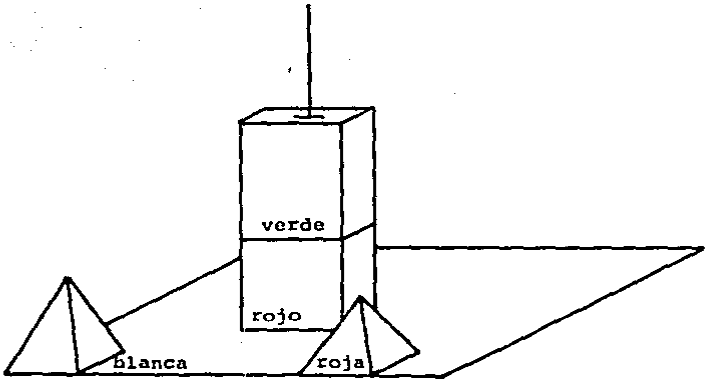


Figura 5.8. "pon el bloque verde sobre el cubo."

12. cuando?

    CUANDO IBA A PONER A UN BLOQUE VERDE

    SOBRE UN CUBO ROJO

13. por que lo hiciste?

    PARA DESPEJAR A UN BLOQUE VERDE

14. despeja el cubo.

    LEVANTA(BLOQUE1)

        MUEVE_MANO([13, 13, 14])

        TOMA(BLOQUE1)

        LEVANTA_MANO([13, 13, 50])

15. apila la piramide roja y el cubo sobre el bloque.

    DEJA(BLOQUE1)

        MUEVE_MANO([23, 3, 8])

```
        DEJA_OBJ(BLOQUE1)

    LEVANTA(CUBO1)
        MUEVE_MANO( [13, 13, 6] )
        TOMA(CUBO1)
        LEVANTA_MANO( [13, 13, 50] )

    APILA( CUBO1, BLOQUE1 )
        MUEVE_MANO( [23, 3, 14] )
        DEJA_OBJ(BLOQUE1)
    LEVANTA(PIRAMIDE1)
        MUEVE_MANO( [2, 22, 4] )
        TOMA(PIRAMIDE1)
        LEVANTA_MANO( [2, 22, 50] )
    APILA( PIRAMIDE1, CUBO1 )
        MUEVE_MANO( [23, 3, 18] )
        DEJA_OBJ(PIRAMIDE1)

    YA

16. cuantos objetos no estan en la mesa?
    2:
        UN CUBO ROJO
        UNA PIRAMIDE ROJA

17. adios.
    HASTA LUEGO '
```

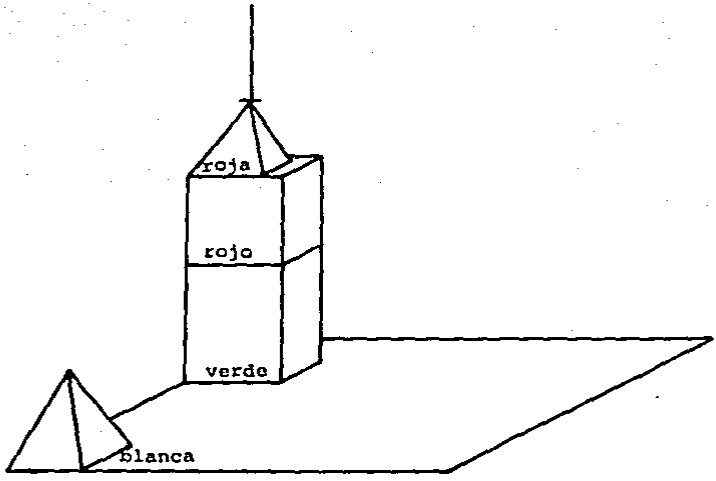


Figura 5.9. "apila la pirámide roja y el cubo sobre el bloque."

5.3. Comentarios.

El tiempo de respuesta de una oración dada por el usuario, no obstante que el analizador es no-determinista y prueba varias opciones, es cuando más de un minuto aproximadamente. La parte que consume más tiempo es la correspondiente a la resolución de planes, cuando el plan a resolver es complicado. Sin embargo, no se pensó que el tiempo de respuesta fuera un factor importante en la evaluación de resultados por tratarse de un sistema experimental.

Una situación que no hemos tratado y que no está claro cómo debiera tratarse es de qué forma el sistema se puede dar cuenta de palabras indefinidas en el diccionario (e.g. adjetivos) dentro de una oración correcta en estructura, para de esta manera preguntar por su significado y darla de alta en el vocabulario. El darla de alta no presenta problema alguno, la parte que no está clara es cómo reconocer la oración, no obstante la palabra desconocida. En el trabajo de Winograd sí se podían realizar definiciones de este tipo.

En su universo además existía una caja en la cual se podían colocar objetos; nosotros no efectuamos el tratamiento de esta caja.

En el trabajo de Winograd se podía preguntar si había sucedido algún evento antes de que hubiera ocurrido otro; ésto se debe a que manejaba el "tiempo" en el cual ocurrían los distintos eventos. En el nuestro no manejamos tal tiempo, solamente almacenamos los eventos que han sucedido en el diálogo .

El sistema de Winograd, si necesitaba colocar un objeto en la mesa y no había espacio libre en ella, era capaz de "formar" un lugar libre, apilando objetos sobre otros. Nuestro sistema no puede formar este espacio.

En el presente trabajo se pueden definir objetos dinámicamente, este hecho no está mencionado en [WIND72].

En lo referente a la generación de planes, STRIPS, creo que

nuestro enfoque es más general que el dado por Winograd, pues su sistema era una gran colección de procedimientos ad hoc, uno para cada caso que pudiera presentarse; a diferencia del nuestro, en donde especificamos las condiciones que deben darse para efectuar una acción y los hechos que ocasiona la ocurrencia de esta acción, con ésto ya no debemos preocuparnos de todos los casos que pudieran suceder en un momento dado (ver sección 4.2.2.1). Esta ventaja se debe a la naturaleza descriptiva de PROLOG.

En nuestro sistema no se puso gran importancia al aspecto de generación de texto que utiliza el sistema cuando responde preguntas debido a que éste no era el objetivo del proyecto.

Por lo que respecta a la complejidad de los sistemas, podemos decir que SHRDLU es uno de los sistemas de comprensión de lenguaje natural más complicado 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.

Como mencionamos en la sección 1.2.1., el elemento más novedoso de SHRDLU fue el modo en que se desarrollaban sus componentes. El sistema usaba simultáneamente todos sus elementos para reconocer, comprender y responder una oración. Tan pronto como había reconocido una estructura sintáctica, activaba un programa semántico para verificar que aquélla tenía sentido; a su vez, la respuesta de este subsistema podía redirigir el análisis sintáctico, llamando procesos deductivos y quizá provocando una serie de preguntas al usuario.

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

Como ya mencionamos, 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 colaboren 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. 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, [WINO73] .

El sistema que hemos desarrollado no tiene todas las capacidades de SHRDLU, sin embargo, es más dúctil, en el sentido de que pudiera modificarse, o extenderse, con relativa facilidad. Esto se debe al carácter declarativo de PROLOG. Como mencionamos con anterioridad, la gramática del sistema la da un conjunto de reglas, escritas como gramática de metamorfosis; esta gramática es factible de modificar.

CAPITULO 6.

CONSIDERACIONES FINALES.

6.1. Perspectivas.

Como se ha podido comprobar, PROLOG es un lenguaje que efectivamente facilita la construcción de sistemas para la comprensión de lenguaje natural. Sin embargo, estos sistemas deben restringirse a un dominio de aplicación particular; dentro de un dominio restringido, es posible construir sistemas que abarquen subconjuntos de lenguaje natural de razonable complejidad, aunque el conjunto de oraciones que comprende está restringido de antemano, es decir, está completamente delimitado. Esta restricción se debe a que, como mencionamos al principio del trabajo, para entender lenguaje natural es necesario poseer una gran cantidad de conocimiento y saber de qué forma utilizarlo.

Elaborar sistemas más generales, ya sea sin restringir el dominio de aplicación, o sin delimitar el conjunto de oraciones bien formadas es un proceso bastante complicado. Podemos decir que los trabajos realizados sobre la comprensión de lenguaje natural no han logrado ser generales en los sentidos mencionados anteriormente. El camino por recorrer es grande, y tal vez deban aplicarse otras técnicas. Por citar un ejemplo, podemos mencionar a J. Pitrat [PITR77], quien afirma que : "...evidentemente, elaborar programas que sean tan capaces como los humanos para comprender lenguaje natural es una tarea sumamente difícil, y muchos años serán necesarios para encontrar un método adecuado. Aún cuando este método se encontrara, solamente se tendría

resuelta una parte de la tarea. Creo que la información pragmática es esencial. Por tanto, probablemente será necesario dar a los programas todo el conocimiento humano en todos los campos, aún para un programa que solamente entienda textos en un dominio limitado. Entonces tendremos que enfrentarnos a otro problema: cómo puede darse esta enorme información a un programa ...".

No obstante, no debemos desanimarnos ante esta situación, puesto que si nos enfrentamos a la necesidad de escribir un sistema de comprensión de lenguaje natural para utilizarlo en una aplicación dada, las posibilidades de éxito son grandes, ya que en realidad solamente nos interesa reconocer cierto tipo de oraciones, y no debemos ocuparnos en reconocer "cualquier" oración.

Debido a ello creo que elaborar interfases en lenguaje natural para una gran cantidad de "paquetes" ya existentes es sumamente fácil y sería de gran ayuda contar con ellas, pues los usuarios no tendrían que preocuparse por aprender comandos, si no en aprender el lenguaje que reconoce la susodicha interfase.

Una crítica posible es que tal vez presente más dificultad aprender el lenguaje que aprender los comandos. Para tener más fundamentos al respecto es necesario efectuar experimentos, realizando este tipo de interfases y dejando que usuarios prueben las dos opciones: lenguaje natural o comandos.

De la discusión de la sección 5.3., se infiere que PROLOG es un lenguaje que ofrece grandes facilidades para la construcción

de sistemas de comprensión de lenguaje natural. Pienso que para contar con más puntos de comparación de las ventajas que ofrece PROLOG sobre otros lenguajes, sería conveniente reproducir otros trabajos clásicos que se hayan realizado, pero ahora utilizando PPOLOG, para así comparar resultados.

Como se mencionó en la sección 2.2, la gramática utilizada en el sistema se escribió de manera empírica. Una línea de investigación puede ser acercarse a personas estudiosas de la materia tales como lingüistas para que de esta forma se puedan utilizar sus tratamientos sobre lenguaje.

Asimismo, la gramática que utilicé es una gramática libre de contexto; sin embargo, también es posible elaborar gramáticas de metamorfosis para gramáticas sensitivas del contexto, enriqueciendo, de este modo, el poder del analizador sintáctico y poder abarcar así una gran variedad de oraciones.

Por otra parte, el sistema no cuenta con un paquete de graficación, pues PROLOG86 no ofrece facilidades para ello. Una opción posible para remediar esta situación es utilizar otra versión de PROLOG, como puede ser TURBOPROLOG, que sí brinda facilidades de graficación. Sin embargo, la traducción del sistema tal vez no sea inmediata; PROLOG86 es bastante semejante al PROLOG descrito en [CLOC84], pero TURBOPROLOG no se adapta completamente a esta versión, pues carece de varios predicados.

6.2. Conclusiones.

El sistema aquí presentado es la culminación de un largo tiempo de trabajo. Tiempo en el cual el autor se informó de bastante literatura existente acerca del tema.

Desafortunadamente la literatura que hay no es concreta, es decir, da pautas a seguir muy generales, pero no es posible inferir como utilizarlas para un caso particular.

El autor espera que las personas interesadas en el tema puedan, a partir del presente trabajo, formarse una idea clara del método utilizado en la construcción del sistema, y que de esta manera pueda servirles de ayuda.

Como se puede inferir del trabajo, PROLOG ofrece amplias facilidades a las personas interesadas en elaborar sistemas para la comprensión de lenguaje natural por computadora. Se puede diseñar una gramática según sean las necesidades, y posteriormente transformarla en gramática de metamorfosis de una manera fácil y rápida; al tener la gramática de metamorfosis se tiene el analizador sintáctico del sistema.

Asimismo, las gramáticas de metamorfosis ofrecen una gran claridad, por esta razón es verdaderamente sencillo para cualquier persona entender la gramática que se está utilizando. Una consecuencia de ésto es que se pueden realizar extensiones de la gramática sin necesidad de modificar las reglas ya existentes.

Con las gramáticas de metamorfosis podemos realizar un análisis sintáctico y semántico dirigido por sintaxis. Con este

método pueden escribirse interfases en lenguaje natural para programas ya existentes.

Por otra parte, el trabajo puede servir de pauta para conocer la forma de utilizar las gramáticas de metamorfosis, que como se puede observar, son una herramienta poderosa.

BIBLIOGRAFIA

[AHO 77] Aho, A. V., Ullman, J. D., : " Principles of compiler design". Addison-Wesley. 1977.

[BOBR64] Bobrow, D.G. : "A question answering system for high school algebra word problems". Fall joint computer conference, 591-614 . 1964.

[BOBP77a] Bobrow, D.G. et. al.: "GUS: a frame-driven dialog system". Artificial Intelligence, v8-2, 155-173. 1977.

[BOBP77b] Bobrow, D. G., Winograd, T. : "An overview of KRL, a knowledge representation language". Cognitive Science, v1, 3-46. 1977.

[BROW74] Brown, J.S. et. al.: "SOPHIE: A sophisticated instructional enviroment for teaching electronic trouble-shooting". BBN. Rep. 2790. 1974.

[BRUC75] Bruce, B.: "Case systems for natural language". Artificial Intelligence 6, 327-360. 1975.

[CLOC84] Clocksin, W.F., Mellish, C.S.: "Programming in Prolog". Springer-Verlag. 1984.

[COHE85] Cohen, J.: "Describing Prolog by its interpretation and compilation". CACM, v28-12, 1311-1324. 1985.

[COLM75] Colmerauer, A.: "Le grammaires de métamorphose". Groupe d'Intelligence Artificielle. Universitè d'Aix-Marseille. 1975.

[HALL67] Halliday, M.A.K. : "Notes on transitity and theme in English". Journal of Linguistics. v3, 37-81, 1977 y v4, 179-215, 1968.

[HEND78] Hendrix, G. G. et. al.: "Developing a natural language interface to complex data".ACM Transactions on database systems, v3. 1978.

[KLU985] Kluzniak, F., Szpakowicz, S.: "Prolog for Programmers". Academic Press, 1985.

[KOWA79] Kowalski, R.: "Predicate logic as programming language" Proc. IFIP North-Holland. 569-574. 1979.

[MICR85] Micro-AI, "Prolog86: user's guide and reference manual". 1985.

[NILS80] Nilsson, N.: "Principles of Artificial Intelligence". Tioga Publishing Co. 1980.

[PITR77] Pitrat, J.: "Formalisms for text analyzes". Séminaire international sur dos systèmes de question-réponse et grandes banques de donnéex. Bonas, France. 1977.

[RICH83] Rich, E.: "Artificial Intelligence". McGraw-Hill. 1983.

[SCHA73] Schank, R. C. et. al.: "MARGIE: memory, analysis, response generation , and inference on English". 3rd IJCAI. 255-261. 1973.

[STER86] Sterling, L., Shapiro, E. : "The art of Prolog". The MIT Press. 1986.

[SUSS70] Sussman, G. et. al.: "Micro-Planner reference manual". A. I. Memo 203, Artificial Intelligence Laboratory, MIT, julio 1970.

[VERD75] Verdejo, M.F.: "Etude du language naturel". Thèse de 3eme cycle, Paris VI. 1975.

[VERD85] Verdejo, M.F.: "Comprensión automática del lenguaje natural". Mundo Electrónico no. 150. 1985.

[WINO72] Winograd, T . : "Understanding Natural Language". Edinburgh University Press. 1972.

[WINO73] Winograd, T. : " A procedural model of language understanding", en "Computer models of thought and language". Schank y Colby, eds. W.H. Freeman and Company. 1973.

[WINS77] Winston, P. H.: "Artificial Intelligence". Addison-Wesley. 1977.

[WOOD72] Woods, W. A.: "The lunar sciences natural language information system: final report". BBN Rep. 2378. 1972.

APENDICE A

SUBCONJUNTO DE LA GRAMATICA UTILIZADA.

A continuación presentaremos un subconjunto de la gramática (en su versión final) que se utilizó para reconocer las oraciones aceptadas por nuestro sistema. Esta gramática es una versión ampliada de la que explicamos en la sección 2.3.3.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% articulos                                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% articulos singulares

art(X,mas,sin,def) --> [el], !.

art(X,mas,sin,ind)    -->    [Art],       % articulos masculinos
        { es_art(Art) },                  % que no terminan en "o"
        !.

art(X,fem,sin,ind) --> [Art],             % articulos femeninos
        { name(Art,Fem),
          append(Raiz, [a], Fem),         % debe terminar en "a"
          name(Masculino,Raiz),           % forma el masculino
          es_art(Masculino)               % eliminando esta "a"
        }, !.

art(X,Gen,sin,DoI) --> [Art],             % articulos masculinos
        { es_art(Art,Gen,DoI) }, !.       % que terminan en "o"

% articulos plurales

art(X,Gen,plu,DoI) --> [Art],
        { plu_a_sin(Art,Articulo),        % convierte a singular
          es_art(Articulo,Gen,DoI) },!.   % busca el singular

% un articulo puede omitirse

art(_, _ , _ , _ ) --> [].

% lista de los diferentes articulos

es_art(un)               :- !.             % articulos masculinos
es_art(algun)            :- !.
es_art(lo,mas,def)       :- !.
es_art(otro,mas,def)     :- !.

% articulos femeninos cuyo masculino termina en "o"

es_art(Art,fem,DoI)      :- fem_a_mas(Art,Articulo),
                              es_art(Articulo,mas,DoI).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% objetos                                                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

objeto(X,Gen,sin,Forma)     -->                      % objeto singular
        [Obj],
        { es_objeto(X,Obj,Gen,Forma) },
        !.

objeto(X,Gen,plu,Forma)     -->                      % objeto plural
        [Obj],
        { plu_a_sin(Obj,Singular),                   % obtiene el sin-
          es_objeto(X,Singular,Gen,Forma) }.         % -gular

es_objeto(X,bloque,mas,bloque(X))        :- !.
es_objeto(X,cubo,mas,cubo(X))            :- !.
es_objeto(X,piramide,fem,piramide(X))    :- !.
es_objeto(X,objeto,_,objeto(X))          :- !.  % cualquier género

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% pronombres                                                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

pronom(fem,sin,def) --> [ella],!.        % caso especial

pronom(mas,sin,Dol) --> [Pronom],        % masculinos
        { es_pronom(Pronom,Dol) },
        !.
pronom(fem,sin,Dol) --> [Pronom],        % femeninos
        { fem_a_mas(Pronom,Masculino),   % obtiene el masculino
          es_pronom(Masculino,Dol)
        }.

es_pronom(uno,ind)       :- !.
es_pronom(alguno,ind)    :- !.
es_pronom(lo,def)        :- !.
es_pronom(el,def)        :- !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% adjetivos                                                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% puede no haber adjetivos
adjs(_,_,_,[])              --> [].

% o haber más de uno
adjs(X,Gen,Num,[P1|P2]) --> adj(X,Gen,Num,P1),adjs(X,Gen,Num,P2).

% regla para reconocer un adjetivo singular

adj(X,Gen,sin,Adjetivo)          -->
        [Adj],
        { es_adj(X,Adj,Gen,Adjetivo) },
        !.
```

```
% plural particular
adj(X, _ ,plu,color(X,azul))     --> [azules], !.

% reglas para reconocer adjetivos en plural
adj(X,Gen,plu,Adjetivo)          -->
        [Adj],
        ( plu_a_sin(Adj,Singular),          % forma singular
          es_adj(X,Singular,Gen,Adjetivo) ).

% lista de adjetivos

es_adj(X,pequeno,mas,pequeno(X))          :- !.
es_adj(X,rojo,mas,color(X,rojo))          :- !.
es_adj(X,azul,_,color(X,azul))            :- !.
es_adj(X,blanco,mas,color(X,blanco))      :- !.
es_adj(X,verde,_,color(X,verde))          :- !.
es_adj(X,grande,_,grande(X))              :- !.
es_adj(X,Adj,fem,Adjetivo)                :-        % femeninos
        fem_a_mas(Adj,Masculino),
        es_adj(X,Masculino,mas,Adjetivo).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% preposiciones de lugar                                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

prep(X,Y,enfrente(Y,X))          --> [atras], !.
prep(X,Y,enfrente(X,Y))          --> [enfrente], !.
prep(X,Y,arriba(X,Y))            --> [encima], !.
prep(X,Y,arriba(X,Y))            --> [arriba], !.
prep(X,Y,abajo(X,Y))             --> [abajo], !.
prep(X,Y,izquierda(Y,X))         --> [a, la, izquierda], !.
prep(X,Y,izquierda(Y,X))         --> [a, la, derecha], !.
prep(X,Y,al_lado(X,Y))           --> [al, lado], !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% frase simple                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% regla  para reconocer  frases simples. Una frase  simple  está
% formada por artículo, objeto y adjetivos; todos ellos del mismo
% género y número.

frase_simple(X,Gen,Num,Sol,[P|P1]) -->
     art((X,Gen,Num,Sol),
     objeto(X,Gen,Num,P),
     adjs(X,Gen,Num,P1).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           predicados auxiliares usados en el análisis        %
%                                                              %
%           morfológico de las oraciones                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% obtiene el Singular de un Plural

plu_a_sin(Plural,Singular) :-
        name(Plural,Lista_P),
        append(Lista_S, [s], Lista_P),
        name(Singular,Lista_S),
        !.

% obtiene el Masculino de un Femenino

fem_a_mas(Femenino,Masculino) :-
        name(Femenino,Fem),
        append(Raiz,[a],Fem),
        append(Raiz,[o],Mas),
        name(Masculino,Mas),
        !.
```