15

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

ACATLAN

PRINCIPIOS DEL LENGUAJE NATURAL POR COMPUTADORA CON APLICACION A LA GENERACION DE PROGRAMAS EN dBASE

T E S I S

PARA OBTENER EL TITULO DE:

LICENCIADO EN MATEMATICAS APLICADAS Y COMPUTACION

P R E S E N T A N :

CLAUDIO RENE FOUILLOUX GARCIA

RODOLFO CASILLAS VALADEZ

ASESOR DE TESIS

M. en C. SERGIO VICTOR CHAPA VERGARA

NAUCALPAN EDO. DE MEXICO 1996

TESIS CON
FALLA DE ORIGEN

Universidad Nacional Autónoma de México

ENEP ACATLAN

Principios del Lenguaje natural por computadora con aplicación a la generación de programas en dBase

Tesis para obtener el título de:

Lic. en Matemáticas Aplicadas y Computación

Presentan:

Claudio René Fouilloux Garcia

Rodolfo Casillas Valadez

Asesor de Tesis:

M. en C. Sergio Victor Chapa Vergara

Naucalpan Edo. De México 1995

LA CULMINACION DE UN ESFUERZO,

UN ANHELO

UNA META DE TANTAS EN LA VIDA,

ESO Y MUCHO MAS

ES LO QUE REPRESENTA PARA NOSOTROS

LA CONCLUSION DEL PRESENTE TRABAJO,

ADEMAS

PENSAMOS Y CREEMOS

QUE NO ES UN FIN EN LA PREPARACION ACADEMICA

EL LLEGAR A ESTE NIVEL,

SI, NO AL CONTRARIO,

ES EL INICIO

DE UN LARGO CAMINO POR ANDAR,

POR QUE EL CONOCIMIENTO Y EL SABER,

ES LA BASE QUE SUSTENTA

LA GRANDEZA DEL HOMBRE EN LA VIDA

DAMOS GRACIAS AL SEÑOR NUESTRO DIOS
por habernos permitido alcanzar una meta más en nuestras vidas

A NUESTROS PADRES
con profundo agradecimiento

A NUESTRAS ESPOSAS
con todo nuestro amor

A NUESTRO ASESOR M. en C. SERGIO CHAPA VERGARA
por sus enseñanzas y valiosa dirección

AL ING. JOSE ALFREDO LOPEZ RODRIGUEZ
por su valiosa colaboración

A NUESTRA QUERIDA UNAM

INDICE

"Si puedes soñarlo puedes hacerlo"

WALT DISNEY

PRÓLOGO

Una idea, una inquietud, un deseo de saber y la necesidad de hacer, nos llevó a realizar este trabajo del cuál realmente nos sentimos muy orgullosos cual padres con sus hijos.

La idea principal fue sobre la interacción hombre máquina. La manera en la que pensamos tendrá que llegar a ser: En lenguaje natural, ya sea Español, Francés, Ingles, Mandarin o Ruso.

La inquietud de probar que era posible, aterrizar nuestra idea, darle forma y traerla al mundo. Sin embargo teníamos casi nulo conocimiento sobre el tema y nuestro deseo de saber nos llevo a leer y releer una cantidad considerable de libros, revistas y artículos. Conocimos teorías, ideas, sugerencias y trabajos de autores en su mayoría Estadounidenses que enfocaban al idioma ingles su trabajo.

La necesidad de hacer nuestra Tesis y de hacer de ésta un reflejo de todo lo aprendido, de lo hecho por nosotros sobre el tema de procesamiento del lenguaje natural y entregar a nuestra Universidad un trabajo digno de ella, es resultado de años de esfuerzo, estudio y trabajo que nos hace sentirlo como un hijo de esa idea y esa inquietud satisfecha.

Entregamos con este trabajo mucho de nosotros y deseamos despertar en nuestros compañeros universitarios la inquietud de explorar el vasto universo abstracto de la Inteligencia artificial, la Ingeniería de software, los sistemas expertos, las bases de datos y temas relacionados, entrelazados, convergentes y divergentes, dependientes e independientes. Para ser abordados como lo quieran, siempre y cuando sea más allá de la orilla y de la superficie.

Hay que animarse a ir más allá, a buscar las profundidades para así realmente aspirar al conocimiento y aportar algo nuevo. Nosotros lo intentamos y reconocemos que nos llevo tiempo aprender a nadar, que muchas veces la corriente amenazo con ahogarnos y perdernos en ese inmenso mar de la Ciencia de la Computación, pero hemos logrado arribar sanos y salvos para captar lo que vimos y aprendimos de él para que los nuevos exploradores cuenten con, al menos, una guía para hacer su travesía más rápida y segura.


Tema I
Fundamentos
del
lenguaje natural

"Así como el cántaro quebrado se conoce por su sonido, el seso del hombre es conocido por la palabra".

ALFONSO X EL SABIO

I.1.- INTRODUCCION

La habilidad del hombre de usar un lenguaje para comunicar una gran variedad de pensamientos o ideas es quizá, la cualidad más distinguible entre él y los animales. Sabemos que el hombre con el paso del tiempo ha intentado comunicarse a través del lenguaje natural con los animales, sin embargo ha fracasado debido a que éstos no son seres racionales. Por otro lado, tenemos que con la aparición de las primeras computadoras digitales se marcó un gran desarrollo tecnológico en el área de la ciencia computacional, que en un principio vino a solucionar tanto problemas matemáticos aplicados a la vida real como también a resolver juegos de mesa de manera "inteligente" y rápida. Este auge, dio pauta para cuestionarnos una pregunta que hemos estado haciendo desde la aparición de las primeras computadoras (a mediados del siglo XIX), y es referente a si una computadora entra en la categoría de ser racional o no. Independientemente de la respuesta, ahora el hombre con el avance de la tecnología pretende lograr una comunicación real con las computadoras, y para ello se ha dado a la tarea de implementar diversos programas, en donde la máquina de una manera "inteligente" obtenga la respuesta rápida y por si fuera poco, la solución óptima. Esta idea comenzó a principios de los 60's, donde los grandes especialistas en programación estaban más convencidos de que una computadora pudiera realmente llegar a pensar por si misma, donde ella, propusiera sus propias decisiones, aprendiera al igual que el hombre, de la experiencia y percepción del mundo y en general actuara o simulara el accionar diario de un ser humano.

Analizando un poco el concepto de que una computadora sea "inteligente", implica que la computadora está ejecutando un programa que piensa y esto como consecuencia da a conocer un programa como "inteligente". Por otro lado, el hecho de que una computadora llegue a pensar o razonar por si misma, ha provocado desde su origen un debate, el cual hasta la fecha no se ha resuelto de un lado favorable.

Todas estas inquietudes nos han motivado para desarrollar en este trabajo de tesis una comunicación limitada entre la computadóra y el usuario, así tenemos que en este capítulo daremos a conocer algunas definiciones de lo que se conoce como la **Inteligencia Artificial (IA)**, estableciendo una taxonomía de la misma, más adelante informaremos los antecedentes históricos que le dieron auge, asimismo, conoceremos el concepto de una de las ramas de la IA que es el **Procesamiento del Lenguaje Natural (PLN)** y del cual dio origen el desarrollo del tema principal de nuestra tesis. Posteriormente comentaremos algunas de las

reglas sintácticas y semánticas con las cuales nos apoyamos para el desarrollo de nuestro proyecto y finalmente mencionaremos las principales limitantes que tuvimos que enfrentar al momento de llevar a cabo la comunicación real entre el hombre y la computadora a través del PLN

I.2.- ¿ QUE ES LA INTELIGENCIA ARTIFICIAL ?

Desde el principio de la era de las computadoras, algunos autores han comentado que los especialistas en informática, se han dedicado más a desarrollar técnicas que permitan actuar a las computadoras como lo hace el ser humano. Una de las herramientas de apoyo a esta nueva forma de desarrollar programas, es la **Inteligencia Artificial**. Se conoce como IA una manera nueva de solucionar problemas, tales problemas según Ana María Martínez Enriquez, se clasifican en:

- Programación automática
- Deducción
  - demostración de teoremas
  - deducción de respuestas a preguntas
  - métodos de planeación
  - resolución de problemas
- Aprendizaje
- Procesamiento del Lenguaje Natural
  - comprensión de la voz
  - análisis lexicográfico
  - análisis del discurso
  - generación del lenguaje
  - generación del discurso
  - síntesis de voz
- Resolución de problemas
  - Juegos
  - Búsquedas heurísticas
  - Planeación
- Representación del conocimiento
- Robótica
- Visión
- Software
- Sistemas Expertos

Desde los años 60's, mucha gente ha visto el campo de la IA, como el lado obscuro y misterioso de la ciencia computacional. La gente creía, que así como el Doctor Frankenstein intentaba crear la vida, los programadores e investigadores en computación se dedicaban a crear máquinas pensantes. Asimismo, la gente ha considerado a estos especialistas en computación como lo más selecto en el ámbito computacional. Sin embargo, otras personas los consideran como los fanáticos de esta ciencia y por esa razón, cuando se les preguntaba referente a la viabilidad o factibilidad de las máquinas "inteligentes", normalmente cuidaban hablar en público de sus investigaciones debido a que los críticos siempre los han desprestigiado al publicar temas como: "muchos investigadores en la IA todavía necesitan haber nacido" y "en algún tiempo, en un futuro lejano, habrán importantes descubrimientos, pero aún estamos lejos de que el área de la IA produzca algo novedoso". A pesar de esto, sabemos que tanto investigadores como programadores en IA definitivamente han contribuido con diversos trabajos que han ayudado al continuo desarrollo y mejoramiento de la comunicación hombre-máquina de manera inteligente.

En los últimos cinco años, la IA se ha transformado de ser un pequeño tema sin interés y atrasado de la ciencia computacional, a convertirse en la cosa más brillante que haya pasado a las computadoras, quizás desde la aparición del primer transistor. Este rápido cambio está basado por cuatro principales factores:

a) El éxito de los sistemas expertos, los cuales fueron los primeros sucesos de la IA con carácter financiero.

b) La gran publicidad que han establecido los japoneses con su proyecto de Sistemas de Computación de la Quinta Generación.

c) La lenta pero constante integración de las técnicas computacionales referentes a la IA dentro de las aplicaciones existentes.

d) Finalmente, el hecho de que el tiempo de la IA ya ha llegado.

Por otro lado, existe una gran variedad de definiciones de inteligencia, y sería muy difícil intentar dar alguna que abarcara todas las ideas referente a ella. Por lo tanto mencionaremos algunas que pensamos son las más significativas.

Inteligencia: (en latín intelligentia) se forma del prefijo latino "inter" que es "muy adentro" y "légere" que significa "mirar con atención", "examinar", por tanto inteligencia significa "examen profundo de una cosa" "comprensión plena".

Inteligencia: palabra derivada del latín "intellegere" que significa "recolectar de" y abarca los conceptos de percepción, discernimiento, selección y establecimiento de relaciones.

Inteligencia: facultad de conocer y comprender, capacidad de adaptación a situaciones nuevas empleando los recursos del pensamiento.

Para explicar el concepto de IA, tomaremos las principales ideas de la definición de inteligencia que acabamos de mencionar, para luego hablar de la definición del calificativo artificial.

Como ya dijimos inteligencia:

- Es la facultad de conocer y comprender.
- La destreza, habilidad y experiencia de un individuo ante el mundo que lo rodea.

Ahora con respecto a lo artificial:

- Es algo no natural, es decir, que no se da por si solo, que es hecho por la mano o arte del hombre.

En conjunto estas dos definiciones han formado lo que comúnmente conocemos como **Inteligencia Artificial**, ya que se habla de una **computadora capaz de conocer y comprender; dar respuesta o solución a partir de su habilidad y experiencia ante las percepciones obtenidas del mundo que la rodea**. En nuestra tesis, la percepción del mundo que la rodea será a través del lenguaje escrito. Además todas estas cualidades de la computadora se establecen y se llevan a cabo por programas hechos por la mano o arte del programador.

La definición explicada anteriormente de como llegamos a contemplar el uso de IA, como nombre propio principal para identificar el hecho de que una computadora piense, no es del todo incoherente. Sin embargo, a partir de esta definición, se han dado diversos comentarios referentes a si una computadora puede o no ser inteligente. Según Michael Shallis, así como la opinión de otros autores, "La idea de IA resulta absurda, y sólo podría ser replantada como Simulación Mecánica de Racionalidad". La opinión está fundamentada en el marco de un mundo jerárquico, en el cual las máquinas no pueden ser inteligentes, si por inteligencia entendemos la más elevada función de la condición humana que vincula al ser humano con lo absoluto.

De estas dos concepciones acerca de lo que es la IA, podemos agregar dos opiniones personales:

a) En conformidad con Lady Lovelace aceptamos que una computadora pueda ser "inteligente" siempre y cuando intervenga la mano del hombre, ya que sin él, nunca una computadora por sí sola podrá ser "inteligente", razón por la cual se le da el calificativo de artificial por no darse de manera natural. Por lo anterior, estamos en desacuerdo con el señor Shallis y los demás autores que consideran absurdo el nombre de Inteligencia Artificial.

b) El hecho de que una computadora pueda ser "inteligente", es cuestionable. Se dice que a partir de que la inteligencia es multiprocesada por un conjunto de circuitos o redes jerárquicas, las cuales toman en cuenta para cada decisión una multiplicidad de factores con diferente peso, que determinan en un tiempo aceptable (que no necesariamente es el óptimo), y esa decisión pasa a

ser un nuevo factor para las nuevas decisiones, llevándose a cabo un proceso de recuperación, almacenamiento, asociación, decisión, deducción y aprendizaje en forma concurrente, que la mayoría de las computadoras no son capaces de hacer. Una cuestión de lógica que pone en duda la cualidad "inteligente" de una máquina es: si la inteligencia es un proceso concurrente propio de los seres vivos; la creatividad y deductividad lo es también, sin embargo, en la actualidad, la IA no cuenta con un programa creativo y deductivo que realmente funcione. Ahora, si un ser humano inteligente puede cuestionarse así mismo sobre su existencia, también puede tener reacciones imprevisibles ante cualquier situación, y puede incluso perder la razón. Es por eso que nos preguntamos ¿La IA simulará todos estos procesos?, si la respuesta es si, entonces vendrán los tiempos en los que existan manicomios para cerebros electrónicos.

Mencionaremos otras definiciones referentes a la IA, que pensamos son de las más apropiadas dentro de la ciencia computacional.

La IA se encarga del diseño de sistemas informáticos inteligentes, esto es, sistemas que presentan las características asociadas con la inteligencia humana, entendimiento del lenguaje, aprendizaje, razonamiento, resolución de problemas etc..

La IA como una de las ramas de la ciencia computacional se basa en los principios de un sistema experto, en hacer que una computadora tenga los conocimientos igual o mejor que un experto humano en ciertos temas de interés. Para que en cualquier momento pueda ser consultada.

La IA según Elaine Rich [RIC83], es el estudio de como hacer que las computadoras hagan cosas en cualquier instante y mejores que las que realizarían los propios hombres.

La IA estudia como la computadora debe hacer cosas que imiten y algunas veces mejoren algunas cualidades inteligentes de un ser humano.

Por ejemplo:

- La comprensión del lenguaje
- El aprendizaje
- Razonamiento para resolver problemas

El pensar es una cualidad asociada con la inteligencia humana, por lo tanto podemos deducir que un ser humano que piensa posee inteligencia, a partir de este razonamiento, diremos que si se habla de computadoras "inteligentes" podemos entrar en debate al preguntarnos ¿pueden las computadoras pensar?. Antes de dar nuestra opinión, procederemos a mencionar las principales opiniones que contradicen el hecho de que las computadoras puedan algún día llegar a pensar como el ser humano:

1.- **Objeción Teológica.** "El pensamiento es una función del alma inmortal del hombre. Dios ha dado un alma inmortal a todo hombre y mujer, pero no a algún animal, vegetal y mucho menos a una máquina". Por lo tanto los animales, los vegetales y las máquinas no pueden, ni podrán pensar.

2.- **Objeción del Avestruz.** "Las consecuencias de que las máquinas piensen serían horribles, creemos y esperamos que no sea posible". El hombre por naturaleza le gusta creer que es en algún modo superior al resto de la creación. Por lo tanto teme que algún día sea desplazado por estas máquinas pensantes.

3.- **Objeción Matemática.** Pueden citarse una serie de resultados de la lógica matemática para demostrar que hay limitaciones en el poder de las máquinas de estado discreto. El más conocido es el denominado teorema de Gödel, que demuestra que en cualquier sistema lógico lo bastante potente, pueden formularse afirmaciones que no pueden demostrarse ni refutarse dentro del sistema, salvo el caso de que el sistema sea incoherente. Tal es el caso en los mismos seres humanos que a veces se ven limitados a contestar o responden erróneamente, y no por ello dejan de ser humanos pensantes.

4.- **Objeción de la Conciencia.** A partir de un discurso conmemorativo del profesor Jefferson en 1949, se citó: "Hasta que una máquina sea capaz de escribir un soneto o de componer un concierto, porque tenga la facultad de reflexionar y sea capaz de sentir, y no por la combinación aleatoria de símbolos, no podremos admitir que esa máquina sea igual al cerebro, en el sentido de que no sólo los escriba, sino que sepa que los ha escrito".

   A partir de los siguientes impedimentos por parte de las máquinas como son:

   - Regocijo por los halagos.
   - Depresión por los errores.
   - Atracción sexual.
   - Enfado o decepción cuando no consigue lo que quiere.

   Se afirma la negación de la validez de las máquinas pensantes.

5.- **Objeción de Incapacidades Diversas.** Posiblemente hagan máquinas que realicen todo lo mencionado en el punto 4, pero es imposible construir una máquina que haga "X". A continuación se citan diversas X, por ejemplo: ser amable, ingeniosa, hermosa, amistosa, poseer iniciativa, tener sentido del humor, distinguir entre lo bueno y lo malo, cometer faltas, enamorarse, apreciar las fresas y los helados, tener un comportamiento tan versátil como una persona, hacer algo auténticamente nuevo, ser objeto de su propio pensamiento, utilizar bien las palabras etc.

Algunas incapacidades quizá tengan muy poca importancia, por ejemplo: la incapacidad para apreciar las fresas y los helados. El hecho de construir una máquina que aprecie esos manjares, sería una tontería intentarlo.

6.- **Objeción de Lady Lovelace.** Para Lady Lovelace "las máquinas no pretenden crear nada, simplemente realizan lo que nosotros sepamos mandarle". Coincidiendo con Hartree quien citó: "Esto no implica que sea imposible construir equipo electrónico que (piense por si solo), o en el que, en términos biológicos, no se pueda implementar, un reflejo condicionado que sirva de base al "aprendizaje".

Hartree advierte que él no afirma que la máquina en cuestión no posea la propiedad de pensar, sino que a Lady Lovelace no le constaba que la tuviera.

7.- **Objeción de la Continuidad del Sistema Nervioso.** Cierto es que una máquina de estado discreto es distinta a una máquina continua. Si consideramos que el sistema nervioso es la única máquina de estado continuo. Por lo tanto no hay máquinas que realicen las funciones continuas del cerebro.

8.- **Objeción de la Informalidad de Comportamiento.** No se puede elaborar un conjunto de reglas para describir lo que una persona hace o debe de hacer en todas las circunstancias concebibles. Por ejemplo: sabemos que tenemos por regla que si uno va conduciendo y ve al semáforo en luz roja hay que detenerse y si está en luz verde avanzamos, pero ¿qué sucede si por un error, se iluminan las dos o no prende ninguna? tal decisión puede surgir a partir de ciertas reglas de conducta que rigen la vida del ser humano. Pero intentar sentar reglas de conducta que cubran cualquier eventualidad, resulta sumamente difícil si no es que imposible para aplicarlas sobre una máquina.

9.- **Objeción de la Percepción ExtraSensorial.** Dentro de las variantes de la percepción extrasensorial como son telepatía, clarividencia y precognición. Por ejemplo: una persona mediante telepatía o clarividencia en un juego de adivinar al azar daría la respuesta correcta 130 veces de 400. La computadora por estadísticas sólo puede adivinar 104, obviamente es una limitante para las supuestas máquinas pensantes.

Nosotros creemos firmemente en la grandeza del ser humano, y como tal, será capaz de lograr casi cualquier cosa que se proponga. Por el momento nos encontramos en lo que podemos llamar la prehistoria de la IA y pensar que nunca logre crear una máquina "inteligente" sería como colocarnos en la época de las primeras computadoras digitales (hace 50 años) cuando el hombre ni siquiera podía imaginar lo que hoy estamos viviendo gracias al progreso de esta área. Ahora al preguntarnos ¿Qué no podrá pasar durante los próximos 20 años?, podríamos responder cualquier cosa, sin embargo nosotros no nos atrevemos a negar la posible creación de cerebros inteligentes no humanos.

Finalmente, cuando decimos que una computadora puede ser inteligente no implica que ésta tenga que hacer todo lo que atribuye al ser humano como inteligente, es decir, que necesariamente desarrolle todas las habilidades de los dos hemisferios con que cuenta el cerebro humano, sino simplemente emule algunas de las características básicas de un ser inteligente.

I.3. ANTECEDENTES HISTORICOS

Hasta ahora la IA ha sido un tema desconocido en nuestro medio, sin embargo, trataremos de abordar el tema desde un pasado no muy lejano, hasta llegar a un futuro que se dice ser prometedor, sin llegar por ello a lo absurdo o más categóricamente a la ciencia ficción.

En 1834 la IA comenzó con Charles Babbage cuando sugirió que su máquina analítica jugara ajedrez, más tarde a mediados del siglo XX con la aparición de los primeras computadoras digitales modernas, es cuando realmente la IA da un paso más allá de lo teórico a lo práctico; empiezan aparecer los primeros jugadores automáticos de "damas".

Cuando se tuvo el primer programa inteligente en la historia de la computación, elaborado por Samuel (el cual jugaba ajedrez) no habia computadora competente para llevar a cabo su ejecución.

A mediados del siglo XX la lentitud y capacidad de las primitivas computadoras, propició un notable impulso en el desarrollo de la microelectrónica, que dio auge al nacimiento de una nueva generación de computadoras de mayor capacidad, rapidez y menor costo. Esto favoreció directamente a la IA, con el hardware apropiado a sus necesidades, ya que en esos momentos se tenían programas inteligentes, que no podían implementarse en máquinas competentes para su ejecución.

La comunicación entre una computadora digital y el hombre tiene cerca de cincuenta años de llevarse a cabo. Este diálogo, en un principio se dio en lenguaje de máquina, más tarde en lenguaje ensamblador y así sucesivamente hasta nuestros días, donde hoy en día existen una gran cantidad de lenguajes de programación, siendo los más populares aquellos que tienen una sintaxis cercana al del idioma Inglés.

Asimismo, los lenguajes de programación fueron desarrollados intentando acercarlos lo más posible al LN (con algunos casos excepcionales como LISP, PROLOG, etc.). Por otro lado, los investigadores en "Inteligencia Artificial" consideraban dentro de sus linderos el **procesamiento del lenguaje natural (PLN)** y dentro de sus objetivos inmediatos el desarrollar programas que tuvieran la capacidad de comunicarse con los usuarios en LN. Desgraciadamente tales programas se han colocado más del lado de los escritores de ciencia ficción que de los desarrolladores de software. Es sorprendente ver como los escritores se han imaginado a las computadoras, consideran que se llegará a tal grado en que éstas podrán entender el LN con la misma capacidad de un ser humano, además de que no estaráa limitadas a una sola lengua, es decir, serán políglotas. En la vida real, se han desarrollado únicamente algunos programas, de los cuales sólo mencionaremos dos que consideramos son los más significativos y que llevan a cabo alguna interacción con el usuario en LN. En primer lugar tenemos el que fue creado por Joseph Weizembaum del **Instituto Tecnológico de**

Massachusetts (MIT), al cual le dio el nombre de ELIZA y más tarde de DOCTOR [WEI83] entre 1964 y 1966. Eliza o Doctor (son dos versiones del mismo programa) imitan la personalidad de un psiquiatra de la escuela de Carl Rogers. El método consiste en negarse a dirigir la conversación con el paciente, prefiriendo, en vez, reflejar en éstos sus propias observaciones, de tal modo que la conversación siempre depende del paciente. El tipo de diálogo en el que participa ELIZA es un poco parecido a una sesión de psicoterapia, debido a que ELIZA anima a su interlocutor para que hable acerca de si mismo. ELIZA se programó con ciertas frases base y con instrucciones para tomar y seguir algunas categorías de palabras utilizadas por el interlocutor. Asimismo, el programa se utiliza tecleando por nuestro lado una oración a la cual el programa responde armando una respuesta de su memoria de frases y palabras. Por tal motivo el programa en si es muy simple, ya que únicamente se lleva a cabo un análisis superficial de las oraciones y utiliza el método de comparación de patrones que se detalla más adelante en el capitulo número II.2.

El segundo programa fue más elaborado, llamado Mundo de Bloques, también conocido como SHRDLU de Terry Winograd (en 1973). El Mundo de Bloques es un pequeño universo formado por bloques de diferentes tamaños, formas y colores. El objetivo del programa es lograr que la computadora responda a comandos tecleados en idioma inglés, es decir, en LN de dos formas: la primera es mostrar sus respuestas en el mismo LN, y la segunda es demostrar que ha comprendido los comandos provocando que un robot mueva los bloques según las instrucciones que le haya dado el usuario. Los bloques y el robot por supuesto no existen como tales, ya que no son sino invenciones electrónicas creadas en el cerebro de la computadora. No obstante, vemos una muestra de ellos en pantalla y esta representación cambia según la computadora va moviendo los bloques de un lado a otro. El funcionamiento de este programa se basa en un análisis gramatical de las instrucciones de entrada, luego ejecuta el comando adecuado y finalmente despliega los elementos de su universo en pantalla. SHRDLU es un trabajo bastante notable y como tal lo consideran los investigadores en el PLN.

Dentro de la Robótica que es una rama de la IA, los robots sólo tienen apariencia humana, en casos contados. Sin embargo la semejanza con el ser humano es indiscutible, pues al igual que nosotros: su cerebro es un ordenador, sus músculos son motores, su esqueleto es la estructura mecánica, y finalmente los sentidos (vista, gusto, olfato, oído y tacto) son sus sensores.

Hay que diferenciar tres generaciones de robots.

- Los de la primera generación son máquinas programadas, para hacer algo concreto, por ejemplo: lacar puertas de un automóvil o transportar material de un lugar a otro.

- Los de la segunda generación utilizan sensores para tomar la decisión de lacar todo el automóvil o sólo una parte que corresponde por su forma.

- Los de la tercera generación son como un niño, cuentan con inteligencia, sobre todo en el aprendizaje, toma de decisiones y comunicación. Por lo tanto, la IA es el gran reto de la robótica y de ella depende su futuro.

Según Anita Flynn hay 3 niveles de inteligencia que se relacionan con las computadoras, en este caso para los robots:

a)Reconocimiento de los lugares y movilización del espacio a nivel regular.

b)Reconocimiento y desplazamiento de obstáculos fijos o móviles.

c)El decidirá, cual será el recorrido, define el objetivo, reflexiona y determina el mejor trazado como un ser humano.

En una revista especializada en computación, Samuel mencionaba que el contenido de los primeros trabajos en computación, específicamente en el área de IA, se podrían englobar en "problemas de juguete", esto es, problemas que requerían principios de IA. En 1962 Samuel presentó en una conferencia la primera evaluación del estado actual de avance de la IA y algunas predicciones de desarrollo para finales del siglo XX.

<u>En cuanto a su presente (1962), destacó:</u>

El gran desarrollo tecnológico que se ha venido dando en los últimos años, tanto en el hardware como en el software, ha provocado una serie de avances importantes en la rama de la IA. Sin embargo, también nos hemos dado cuenta que se ha frenado un poco el avance, no tanto a nivel del hardware (en las máquinas), sino más bien, nos hemos detenido a nivel de software debido a la capacidad que el hombre tiene para programarlas.

Dentro de sus predicciones a futuro para fines del siglo XX, destacaremos tres importantes:

a) Habrá computadoras que sean quizá cien o mil veces más rápidas que las presentes, dotadas de memorias grandes, que no obstante de su capacidad ocuparán volúmenes inferiores a las actuales.

b) Las computadoras aprenderán de su experiencia.

c) Las computadoras conversarán libremente con los usuarios.

Analizando estas tres predicciones, en la primera sabemos que es correcta, debido a que hoy en día es una realidad, en cuanto a las otras dos tendremos que esperar a las máquinas de la Quinta Generación, proyecto que esta siendo desarrollado por los japoneses (Proyecto PROSPECTOR (SRI) iniciado en 1981 y el cual tiene como objetivo contar con un sistema operativo que interactue en LN tanto escrito como hablado entre la computadora y el usuario). Además en estas

dos últimas predicciones viéndolas con enfoque a la IA, diremos que la primera tiene que ver con el aprendizaje y la segunda con el poder de comunicación hombre-máquina.

Asimismo, la mayoría de los autores que han abordado el tema llegan a la conclusión que el problema central de la IA es el de las "máquinas que aprenden"; en particular nosotros estamos de acuerdo, debido a que la gente estará más vinculada con la máquina cada vez que ésta, facilite la comunicación con todo tipo de usuario y, por tanto, no se necesite ser un experto para trabajar con ellas.

En cuanto al futuro subrayó:

Es mucho más fácil hablar de él, ya que posiblemente no estaremos aquí para ser desmentidos. A pesar de ello, los siguientes puntos buscan una solución real a un plazo relativamente corto.

a) Marcar un teléfono, en cualquier lugar del mundo y poder conversar con cualquier persona que hable en idioma diferente al nuestro, con las únicas limitaciones en el retraso de la transmisión, estructura de las palabras y oraciones que ocurren entre cada uno de los lenguajes.

b) Transmisión de material pictórico, de texto y video telefónico en la computadora, en términos del esperado alto desarrollo de la mencionada computadora de bolsillo.

c) La computadora estará activa las 24 horas vigilando toda la actividad administrativa de una casa, oficina u empresa.

d) Sistemas expertos para no expertos estarán disponibles en todas las bibliotecas, empresas, servicios públicos, etc., desde luego por vía telefónica.

e) Posiblemente desaparecerá el papel como herramienta para conservar escritos y gráficos; en cambio el libro electrónico será una realidad en cualquier biblioteca.

f) La idea que se tiene de una computadora como un objeto pasivo en espera de ser consultada o informada, desaparecerá. En un futuro la computadora misma podrá retroalimentarse a través de sus sentidos y con ayuda de sus motores estará presente ante nosotros.

g) Llegará el momento en que las computadoras controlen, descubran y decidan problemas que al hombre mismo ni siquiera se le hubieran ocurrido plantearlos y mucho menos resolverlos.

Finalmente Samuel comentó que estos dos últimos incisos pertenecen a una segunda fase de la inteligencia, en la cual la computadora ya actúa con

"inteligencia" de manera independiente. De la misma manera mencionó que la primera fase se basa principalmente en la comunicación hombre-máquina.

Posiblemente de estos presagios hacia el futuro, algunos incursionarán en el área de la ciencia ficción. No obstante, consideramos que muchas de ellas son y estarán siendo una realidad en un periodo relativamente corto. Por otro lado, al decir que algunas predicciones se están pasando del mundo real al mundo de la ciencia ficción; no queremos negar que puedan ser una realidad, y así evitaremos equivocar nuestros juicios en un futuro.

Hasta ahora hemos mencionado, según algunos autores que la IA empezó con Charles Babbage; para otros inició a mediados del siglo XX con el auge de las computadoras potentes. Sin embargo, es difícil determinar con precisión una fecha exacta de cuando empieza lo que comúnmente conocemos como IA. Quizá el crédito al nacimiento de la IA, debería ser otorgado a Alan M. Turing por su invención del programa almacenado en computadora; además en 1950 fue el primero en realizar y publicar la pregunta ¿Puede pensar la máquina?. De esta pregunta, propuso que para comprobar si una computadora era o no capaz de pensar, debería de aprobar un juego el cual consistía en un interrogador humano denotado como X el cual utiliza una terminal de computadora en una sala para efectuar una conversación con Y y Z operadores de otras terminales en otra sala diferente. X sabe que uno de los dos operadores es un ser humano y el otro una computadora. Mediante las preguntas X debe decidir si el operador es el ser humano o la computadora. Si X no puede distinguir Y de Z, entonces se puede decir que la máquina pensaba, esto es, que simulaba la inteligencia humana. Desde que Turing propuso este experimento, no ha habido programa con suficiente inteligencia que se desempeñe bien durante la prueba. ¿Es posible que algún día alguna computadora pase esta prueba?. Nadie lo sabe. El mismo Turing pensó que tal máquina se construiría cerca del año 2000.

Gracias a la aportación de Alan Turing de que los programas podían ser almacenados como datos en la memoria de la computadora y ser ejecutados más tarde, se formaron las bases para la realización de toda computadora moderna. El almacenamiento de programas permitió a las computadoras cambiar su función de manera rápida y fácil, por simplemente correr un programa. Esta capacidad implica que una computadora podría cambiar su propia función, de simplemente resolver problemas matemáticos o juegos, a poder aprender o pensar.

En 1958 fue creado por John McCarthy LISP (List Processing Language), el primer lenguaje utilizado específicamente para resolver problemas algebraicos en el área de la IA.

Su autor resumió las siguientes características:

- Trabaja con expresiones simbólicas más que con números, esto es, los bits de la memoria y los registros pueden ser ocupados por símbolos arbitrarios y no sólo por operadores aritméticos.

- Procesamientos de listas; representación de los datos en listas estructuradas dentro de la memoria.

- Estructura de control basada en la composición de funciones básicas para formar otras más complejas.

- Utilización de la Recursividad como una forma de escribir procesos y resolver problemas.

- Representación interna de los programas escritos en LISP como un conjunto estructurado de listas.

- La función EVAL, escrita a su vez en LISP, sirve como un intérprete del mismo y una definición formal del lenguaje.

Literalmente, lo que comúnmente se conoce como IA se puede decir que empezó alrededor de los años 60's, cuando en el MIT John McCarthy creo LISP. De la misma manera el término IA es generalmente acreditado a Marvin Minsky, también de MIT, el cual en 1961 escribió un tratado llamado "Steps Towards Artificial Intelligent".

Los años 60's fue un período de intenso optimismo referente a la posibilidad de hacer que una computadora pensara. Después de todo, en los años 60's se vieron los primeros jugadores de ajedrez por computadora, las primeras pruebas matemáticas computarizadas orientadas a la resolución de problemas reales, y el famoso programa anteriormente comentado "ELIZA" o "DOCTOR".

Timothy Brown (uno de los directores del laboratorio de MIT) dice que dentro de las metas de MIT, lo más importante es lograr fundir la ciencia con el arte y la filosofía, para imitar en la medida de lo posible la inteligencia. En el MIT se realizan actualmente 69 proyectos, donde el más ambicioso es el llamado "Face Reading Proyect" que consiste en una máquina sensible, capaz de leer los labios, comprender las palabras, traducir a varias lenguas, seguir las miradas e interpretar las sensaciones humanas. No se dan avances ni se prometen fechas, sin embargo podemos hacernos una pregunta ¿Será posible a un corto plazo?.

La gente en aquellos tiempos se preguntaba ¿Deberían las computadoras ser usadas de esta forma?, ¿Puede una computadora psiquiatra ser mejor que los hombres?, ¿Debería permitirse este tipo de programas?. Aun Weizenbaum, creador de Eliza, escribió el libro "Computer Power and Human Reason", en donde desacreditaba su propio programa. Recuerdo que los años 60's fue un periodo de intenso miedo a la automatización, esto es, algunas veces era duro comprender las emociones que existían entre la gente al verse en un futuro desplazados, dependientes y hasta dominados por esas máquinas, esto hace ya más de 30 años.

A causa del aparente éxito de la IA, parecía que el hecho de producir un programa que tuviera inteligencia como los seres humanos se iba a dar más pronto de lo que se esperaba. Sin embargo, como sabemos, esto no ha sido así; lo que no estaba claramente entendido en los años 60's era la dificultad de concretizar

estos éxitos, dentro de un programa flexible e inteligente. A pesar de que los programadores trataban de incrementar la generalidad de ciertos programas, estos intentos requerían de mayores recursos computacionales, superiores a los que se disponían en esos momentos. Así la memoria de la computadora no era lo suficientemente grande ni el tiempo de ejecución era lo bastante rápido.

A mediados de los años 70's, las computadoras ya poseían grandes memorias y la velocidad de ejecución había incrementado favorablemente. Sin embargo, aún con estas mejoras, los programas referentes a la IA presentaban bastantes fallas por insuficiencias propias de los programadores.

A finales de los 70's varios éxitos, tales como: el PLN, la representación del conocimiento y la resolución de problemas habían sido registrados en áreas específicas de la IA. Estos éxitos dieron auge a la introducción del primer producto comercial de la IA, el cual fue el sistema experto (SE). Un SE es un programa que contiene información referente a una cierta área específica, y cuando se le pregunta contesta igual que una persona experta. Uno de los primeros SE fue INTERNIST, desarrollado por H. Pople, científico en computación, y J. Myers, doctor especializado en medicina interna; ambos de la Universidad de Pittsburgh. INTERNIST consiste de una base de conocimientos muy grande, de más de 500 enfermedades y 2900 síntomas asociados. El médico que lo utiliza registra una serie de padecimientos expuestos por el enfermo, y el programa busca en su base de conocimientos una enfermedad o enfermedades que correspondan a los síntomas establecidos por el doctor. Otro SE, más complejo que el anterior en su desarrollo, es el llamado MYCIN, el cual fue desarrollado en la Universidad de Stanford para ayudar a los doctores a diagnosticar infecciones bacterianas que requirieran tratamiento de emergencia. MYCIN no sólo es un SE capaz de responder a preguntas que le formulen de una forma directa cuando posee las soluciones almacenadas en su memoria, sino también, es un sistema capaz de responder a preguntas con soluciones que no se encuentran directamente almacenadas en el computador, así tenemos que su capacidad de inferir esta basada en reglas, es decir, almacena su base de conocimientos en forma de reglas de inferencia denominadas reglas de producción. Estas reglas se basan en la siguiente expresión:

SI Observación

ENTONCES Inferencia

En el caso de MYCIN, las observaciones (SI) describen una infección bacteriana, y la inferencia (ENTONCES) es una estimación de la posible causa. Por ejemplo en una traducción al español de MYCIN tendríamos:

SI

1) la infección es bacteriana primaria, y
2) el lugar de cultivo es estéril, y
3) el lugar de sospecha de entrada al organismo es el tracto gastrointestinal

ENTONCES

> Existe un 70% de probabilidad que la identidad del organismo es bacteroides.

Uno de los más importantes sucesos de la IA que ocurrió en los años 70's fue la creación de PROLOG en 1972, a cargo de Alain Colmerauer en Marsella, Francia. PROLOG, abreviatura de (PROgraming language for LOGic) es un lenguaje que implementa una versión simplificada del cálculo de predicados y utiliza un lenguaje lógico. Así como LISP, PROLOG fue un lenguaje que se diseñó para ayudar a resolver problemas relacionados con la IA; de esta manera LISP facilita la creación de programas que manejan listas, mientras PROLOG facilita el trabajo con expresiones lógicas. No obstante, que estos lenguajes facilitan de manera significativa la programación en la IA, podemos también auxiliarnos de los lenguajes de alto nivel como PASCAL y C principalmente. Según Herbert Schildt [SCH87] "PROLOG difiere de LISP, debido a que éste posee varias herramientas especiales, tales como un constructor de bases de datos interna, además de contar con una sintaxis más simple". Esencialmente, hasta 1980, LISP era el lenguaje oficial que utilizaban los Estados Unidos para sus investigaciones orientadas a la IA; mientras PROLOG tenía el mismo estatus en Europa. Sin embargo, en 1981, está situación cambió con el anunciamiento de que los japoneses usarían PROLOG como lenguaje base para su ambicioso proyecto de las computadoras de la Quinta Generación y como consecuencia los programadores norteamericanos, empezaron a conocer y desarrollar en PROLOG.

Schildt afirma que el lenguaje PROLOG ha sido importante en la historia de la IA ya que éste expresa un profundo entendimiento del proceso del pensamiento lo que LISP no hace. Por ejemplo: PROLOG contiene un fácil constructor de bases de datos interna y una rutina **(Back Tracking)**; ambas herramientas son esenciales para la solución de muchos problemas relacionados con la IA. La técnica back tracking es un algoritmo que permite buscar una solución a un problema, pasando por varios niveles o rutas. Si la rutina llega a un nivel terminal, y no encuentra la solución, entonces simplemente se regresa a un nivel anterior y continua analizando por otra ruta. Cabe aclarar que esta técnica se puede implementar usando el método recursivo back tracking.

El reto en el campo de la IA es cambiar, de la simple investigación a la aplicación. Este cambio significa que las técnicas de IA que fueron desarrolladas en el laboratorio usando un lenguaje de investigación, necesitarán ser implementadas usando varios lenguajes de propósito general para resolver problemas del mundo real.

En el siguiente cuadro resumiremos este tema, estableciendo los aspectos claves que han sido esenciales en la historia y desarrollo de la IA.

PERIODO	SUCESOS CLAVES
Principios del siglo XIX	Aparición de la primera máquina analítica. Primeras ideas referentes a la comunicación hombre-máquina.
Fines del siglo XIX	Primeras aplicaciones importantes de una computadora. Por ejemplo el uso de la máquina del censo de Hollerith.
Principios del siglo XX	Creación de programas inteligentes para la solución de problemas y juegos.
Antes de la segunda guerra mundial.	Lógica formal. Psicología del conocimiento.
Década de 1940	Aparición de la primera computadora digital, La Mark I y la aparición de la primera computadora digital electrónica ENIAC.
La postguerra 1945-1954, inicio de la IA	Desarrollo de la computadora. Conferencias sobre cibernética.
Los años de formación, 1955-1960	Aumenta la disponibilidad de computadoras. Aparición de los lenguajes de Tercera Generación. Psicología del proceso de información. Traducción Automática por la U.R.S.S. (1957).
Los años del desarrollo 1961-1970	LISP (M.I.T.) Boston. Heuristica. Robótica. Solución de problemas de ajedrez. DENDRAL (Stanford). Primeras manifestaciones de la comunicación hombre-máquina. Por ejemplo ELIZA ó DOCTOR y MUNDO DE BLOQUES.
La especialización y los sistemas expertos, 1970-1980	INTERNIST (Universidad de Pittsburgh) MYCIN (Stanford). MACSYMA (M.I.T.). EMYCIN (Stanford). PROLOG. (Marsella,Francia).
La comercialización, 1981	PROSPECTOR (SRI).(Octubre de 1981) Proyecto japonés de la Quinta Generación. INTELLECT. Sistemas inteligentes de recuperación de información. Diversas compañías comercializan herramientas para la construcción de sistemas expertos.

PERIODO	SUCESOS CLAVES
Fines de los 80's	Empiezan a darse los primeros programas terminados referentes al procesamiento del lenguaje natural. Q&A SYMANTEX. TIBAQ (Text and Interface Based Answering of Questions).
Ultimas conquistas	Brazo electrónico más preciso que el de un cirujano, (Oussama Khabit, Stanford). El proyecto Face Reading, una máquina capaz de leer los labios, comprender las palabras, traducir a varias lenguas e interpretar las sensaciones humanas. (M.I.T.)

I.4.-PROCESAMIENTO DEL LENGUAJE NATURAL

El propósito del PLN es facilitar una interrelación hombre-máquina a través de una comunicación mucho más fluida y menos estructurada que los sistemas tradicionales de menues.

De acuerdo con Rich [RIC83], "la comprensión del lenguaje natural requiere tanto del conocimiento lingüístico del lenguaje particular en uso como también del conocimiento del mundo relativo al tópico bajo discusión".

Olvidándonos un poco de la supuesta cualidad "inteligente" de las computadoras; diremos que para interactuar la computadora con los humanos de una manera coherente, ésta debe tener conocimiento referente a su propia identidad, capacidades y limitaciones. Al igual que una relación entre humanos, la cual se encuentra limitada por la capacidad, conocimiento y experiencia de los participantes, la computadora no es la excepción.

En una revista especializada en IA, Klaus K. Obermeier dividia las aplicaciones de un PLN en seis grandes áreas, las cuales menciono a continuación:

1)Lenguaje Natural como interfase a bases de datos.
2)Traducción automática.
3)Programas inteligentes en el análisis de textos para resumirlos.
4)Generadores de textos para la producción de documentos estándares.
5)Sistemas del habla para interacción mediante la voz.
6)Herramientas de sistemas de PLN de aplicaciones específicas.

Nuestra aplicación se basa en la primera área, como algunos trabajos que ya estan a nivel comercial, ver anexo I.

En estos momentos se han desarrollado o están por realizarse una gran cantidad de interfases en LN para la interacción hombre-máquina. Todos estos programas recurren a alguna forma de categorización conceptual. Estas clasificaciones se hacen para poder asignar el filtro de información que deberán formar las bases de conocimientos. Tales bases pueden referirse a características físicas de objetos, adjetivos, "scripts", planes, metas, relaciones, etc.

Al hablar de bases de conocimientos nos podríamos preguntar ¿Porque a la base de conocimientos no se le llama base de datos?, la razón es que una base de conocimientos significa algo más que información almacenada, puesto que incluye una cierta "inteligencia" en cómo y cuándo utilizar la información.

Las bases de conocimientos pueden ser fácticas o dinámicas:

- Fácticas. sus relaciones o datos son previamente establecidos e inalterables.

- Dinámicas. son más poderosas debido a que permiten actualizar y modificar sus relaciones o datos; además, integran de manera organizada la nueva información.

Un mecanismo común que forma parte en toda interfase para el PLN es el "analizador" o "parser", cuya función es examinar las oraciones por partes y organizarlas de acuerdo a una u otra estructura gramatical. Según Rich [RIC83] este proceso de análisis realiza dos cosas principalmente:

- Determina que oraciones se aceptan como sintácticamente bien formadas y cuales no.

- Asigna una estructura a las oraciones sintácticamente bien formadas.

El análisis inferencial es un problema al que se tienen que enfrentar los sistemas que realicen el verdadero proceso interactivo entre "el hombre y la máquina". Este análisis se da en el momento en que se requiere información que no está incluida en las bases de conocimientos y por tanto el sistema intenta almacenar esa nueva información a la base de conocimientos.

El PLN como interfase del conocimiento empezó a surgir a finales de los años 50's. En aquellos tiempos tanto los ingenieros en computación como la prensa esperaban que pronto hubieran sistemas que realizaran funciones tales como: traducción automática, generadores de textos y una comunicación hombre-máquina en lenguaje natural. Sin embargo, los lingüistas fueron más atinados en sus predicciones, debido principalmente a la ambigüedad de los idiomas. A pesar de que no se han logrado totalmente los objetivos esperados, quizas exagerados por lo prematuro, se han dado avances con un grado de efectividad aceptable en: sistemas traductores de lenguajes, sistemas de procesamiento de información, que poseen capacidad limitada para comunicarse en LN en ambas direcciones y finalmente todos los procesadores que han contribuido a la generación de textos de manera rápida y sencilla.

Con la aparición de los sistemas expertos (SE), la utilización de una interfase en LN, es decir, un "asistente inteligente" de usuarios, implicaba darle una mayor atención e investigación al desarrollo de esta técnica, ya que de alguna manera estaríamos impulsando la construcción de SE y por consiguiente el avance de la IA.

De acuerdo con Eva Hajicova [RIC83], la interfase en LN como una de las áreas de aplicación del PLN se clasifica en dos de acuerdo a su uso.

a) La primera aplicación permite hacer consultas con base a un programa que analice sintáctica y semánticamente el enunciado del usuario, compare el enunciado con los datos almacenados por el sistema y proporcione una respuesta sintetizada (generador de oraciones y textos).

Podemos aplicar la interfase en LN a una simple base de datos. Con esto, la tarea de construir el analizador es más sencilla, esto se debe a que tales sistemas están limitados en el alcance de los datos, ya que tratan con un universo restringido; además, la estructura de la pregunta a una base de datos en particular es en principio predecible. Por esta razón, no es raro que en nuestra época existan sistemas que ofrezcan al usuario una interfase en LN que lo comuniquen en su lengua, generalmente en el idioma Inglés. Un ejemplo es la interfase Q&A SYMANTEX (disponible desde Septiembre de 1985), trabajando en computadoras IBM PC/XT/AT. El software de este sistema, contiene un conjunto integrado de asistencia para el usuario, un procesador de palabras y un manejador de errores; además, contiene un asistente inteligente, es decir, un programa que el usuario tiene como interfase a una gran variedad de conjuntos de datos: evidencias personales, bibliografía y sugerencias para la corrección. Adicional a ésta, existen otras interfases en LN (ver anexo I).

b) La segunda aplicación es mucho más ambiciosa que la primera, ya que incluye una meta con un mayor grado de complejidad que es la "comprensión del lenguaje natural". Para el acceso a SE y a sus bases de conocimientos para robots inteligentes o "enciclopedias automáticas" es necesario, un procesamiento de traducción de texto fuerte (hasta ahora a nivel escrito, debido a razones que se explicarán más adelante en el tema 7 del capítulo I) en una representación del conocimiento sin ambigüedad, es decir, en un conjunto de redes semánticas.

Este reto, no sólo constituye un obstáculo en las tareas centrales de la ciencia computacional, sino también en la lingüística moderna; una descripción explícita de las principales características del sistema de lenguaje que es necesaria para estos propósitos, tiene que estar basada en una estructura teórica bien organizada y adecuada tanto para la descripción de la gramática, como también para los aspectos de los modelos lingüísticos de la semántica y pragmática. La estrecha relación entre la lógica y la lingüística, es decir, la ciencia de la computación y la ciencia del aprendizaje no puede desvincularse, ya que es necesaria su constante interrelación para lograr las metas que se han planteado con respecto a esta aplicación, como interfase del PLN.

Un ejemplo de un sistema experimental de este tipo es el sistema basado en el método TIBAQ (Text-and-Inference Based Answering of Question). Queremos hacer notar que esta segunda aplicación es experimental por carecer de sistemas totalmente terminados, debido al grado evidente de dificultad.

Los cuatro procedimientos principales que abarcan este método son:

1) Análisis gramático-semántico.
2) Reglas de inferencia.
3) Identificación de respuesta completa (directa) o parcial.
4) Síntesis (respuestas).

Cabe señalar que para los tres primeros pasos del análisis, no es necesario dividirlos dentro del programa de la computadora; no obstante, tienen que ser dis-

tinguidos como partes de la estructura lingüística interna que se necesita desarrollar.

Las principales funciones que realiza un PLN son las siguientes:

a) Permiten al usuario tener un menor conocimiento del sistema, debido a que evita la utilización de estructuras fijas.

b) Corrigen errores de tipo léxico. Al realizar una consulta el usuario, si la pregunta contiene un falta sintáctica, el programa intentará asociarla con alguna de su diccionario. Si alguna de éstas encaja lo suficiente confirma la corrección del fallo y la frase es analizada.

Ejemplo:

el usuario escribe:
¿Hay **tresientos** alumnos en el turno de la tarde?

La interfase marca error en "tresientos" y sugiere
corregir con la palabra: "trescientos"

c) Permite al usuario construir oraciones refiriéndose a frases anteriores. Esta capacidad se denomina anáfora y existen dos tipos:

1.-Sustitución. El PLN puede sustituir el significado de alguna palabra previa.

Ejemplo:
¿Qué automóviles corren a más de 100 Km/H.?
¿Cuáles de ellos entre 100 y 160 Km/H.?

2.-Elipsis. El PLN puede entender una pregunta en la que no aparece ninguna referencia previa.

Ejemplo:
¿Qué automóviles corren a más de 100 Km/H.?
¿Cuáles entre 160 y 200 Km/H.?

d) Responde a preguntas de manera inteligente.

Ejemplo:
¿Está Pedro inscrito en séptimo semestre?

La respuesta puede ser:
No
o
No, no existe el séptimo semestre
(La respuesta es más clara y precisa).

e) Permite la ampliación de las reglas gramaticales. Es de mucha utilidad que los PLN se puedan ampliar fácilmente, asi permiten su sencilla adaptación a todo tipo de usos particulares.

Los problemas más importantes que afectan de manera directa en el desarrollo de PLN son los siguientes:

a) Se requieren grandes computadoras para su implementación, no sólo para contener el PLN; sino también, el sistema al que va asociado debe ser de cierta importancia y capacidad para almacenar toda la gramática de una lengua en particular.

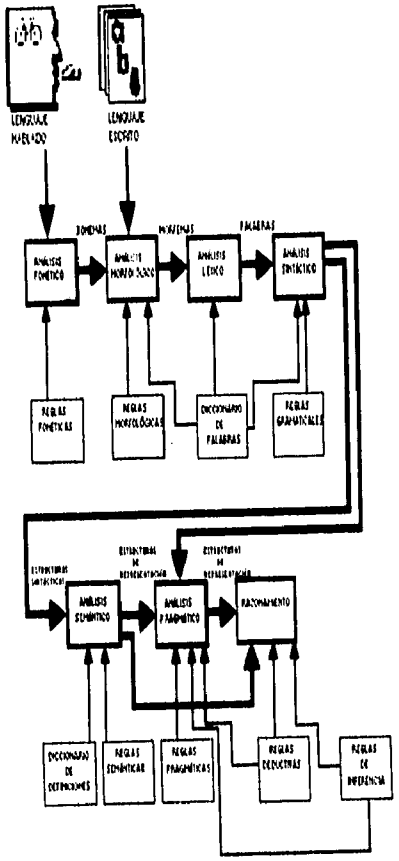
b) Tipificar todo el lenguaje es una tarea ardua, por lo tanto, sólo se utilizarán lenguajes parciales que constituyan un subconjunto de la lengua en particular. Además, unas cuantas reglas sintácticas ayudan a definir la gramática a utilizar en las consideraciones del diseño de tipo semántico.

c) El lenguaje humano es poco apto para transmitir información de manera clara, concisa y precisa.

De acuerdo con Terry Winograd [WIN84], para que un programa lleve a cabo una comprensión automatizada del lenguaje, necesita pasar por varias etapas que constituyen los distintos niveles de análisis del lenguaje.

El siguiente esquema muestra los niveles de análisis por el que pasa un PLN y que permiten realizar una comunicación real entre el usuario y la computadora.

ESQUEMA GENERAL DE UN PROCESADOR DE LENGUAJE NATURAL



La comprensión automatizada del lenguaje requiere que la computadora se sirva de varios tipos de datos almacenados como son:

- Reglas fonéticas
- Reglas morfológicas
- Diccionario de palabras
- Reglas gramaticales
- Diccionario de definiciones
- Reglas semánticas
- Reglas pragmáticas
- Reglas deductivas
- Reglas de inferencia

y que acometa distintos niveles de análisis como son:

1) Análisis fonético

2) Análisis morfológico

3) Análisis léxico

4) Análisis sintáctico

5) Análisis semántico

6) Análisis pragmático

7) Razonamiento

Si el lenguaje es hablado, el primer análisis es fonológico **1)** la computadora analiza las ondas sonoras. Si el lenguaje es escrito, el primer análisis será morfológico **2)** la computadora descompone cada palabra en su radical, o forma básica, y las flexiones (por ejemplo: ando). Sigue después el análisis léxico **3)**, en el cual la computadora pone las palabras en sus categorías léxicas (por ejemplo, verbo) e identifica determinados "rasgos", por ejemplo: género y número. Viene a continuación el análisis sintáctico **4)** la aplicación de las reglas gramaticales para sentar la estructura de la oración. Procede luego el análisis semántico. **5)** La frase se convierte en una forma que la haga susceptible de sacar inferencias. La última etapa es la del análisis pragmático **6)** pone sobre el tapete el contexto de la frase; así como también, la relación que existe entre el momento en que se habla y el momento a que se refiere lo expuesto. La computadora puede ahora extraer inferencias **7)**, quizás a modo de preparación para la generación de respuestas.

Concluiremos diciendo que a pesar de que se ha investigado mucho en lo relativo al diseño de los PLN, aún no existe uno que realice eficazmente todas estas fases. Esto es debido a que no hay un manejo eficiente del significado de las palabras. Por si fuera poco, otro gran problema estriba en la naturaleza del razonamiento humano de sentido común, es decir, la mayor parte de lo que sabe una persona no puede formularse en reglas lógicas de todo o nada; este saber descansa en las "expectativas normales". Si se pregunta ¿Hay polvo en el jardín? la respuesta será afirmativa, casi con toda seguridad. Pero este "si" no será nunca una inferencia lógica; algunos jardines son hidropónicos, donde las plantas crecen en el agua. La gente se guía por las expectativas normales, sin pensar en las excepciones a menos que sean relevantes. Las limitaciones que encuentra la formalización del significado del contexto hacen imposible ahora y quizá por siempre diseñar programas de computadora que realmente lleven a cabo la comprensión humana del lenguaje.

I.5.- COMUNICACION ENTRE EL HOMBRE Y LA MAQUINA

En el inicio de las computadoras, el hombre creía que éstas con cierto nivel de inteligencia, iban a ser capaces de realizar cualquier cosa que ellas se propusieran. Sin embargo, hoy día sabemos que ésto no ha sido así, ya que en la actualidad se esta comenzando a trabajar para considerarla como una máquina lingüística; para la cual su poder deberá residir en la capacidad de manipular símbolos lingüísticos a los que se les ha asignado un significado. Así tenemos por ejemplo: que el hecho de pensar que un robot hable como humano, sin más que un ligero timbre mecánico en su voz, hace que nuevamente el tiempo demuestre a los desmesuradamente optimistas que lo que ellos deseaban, todavía esta lejos de poder realizarse.

La posibilidad de que algún día pueda haber una comunicación abierta entre el hombre y la máquina de manera coherente e hilvanada ha motivado a los especialistas en informática el seguir intentando desarrollar técnicas que permitan a las computadoras actuar como lo hace el ser humano. Algunas técnicas avanzadas en el PLN han dado las herramientas necesarias, pero no suficientes, para llevar a cabo una regular comunicación entre el hombre y la máquina. Asimismo, cuantos intentos se han realizado para fabricar computadoras que hablen y se han topado con dificultades enormes, y a los mejores prototipos de laboratorio todavía les falta mucho para alcanzar la capacidad lingüística media de un niño de tres años.

Por otro lado, las necesidades de información y uso de la computadora a lo que están sometidos la mayoría de los profesionistas en los más diversos campos, está provocando un cambio sustancial en la relación hombre-máquina. En este sentido es necesario crear una computadora que sea lo suficientemente amigable para facilitar su interacción con los usuarios finales y no nada mas con los especialistas en informática. Así tenemos que Japón ha dado el primer paso con el anuncio de su proyecto PROSPECTOR el cual pretende lograr cuatro metas principales que acontinuación mencionamos:

(a) **Rapidez.-** Se pretende conseguir que la computadora funcione más rápido, es decir, que sean capaces de procesar más instrucciones e información que las actuales. Esto se intenta a través de unos CHIPS más rápidos con capacidad de proceso, esto es, de realizar operaciones básicas.

(b) **Trabajo con lógica simbólica.-** Actualmente las computadoras trabajan con estructuras principalmente numéricas. Aunque también manejan símbolos distintos de los números, todavía resulta algo confuso la utilización de letras y caracteres no numéricos en la programación de aplicaciones de todo tipo. Se pretende crear un "software"

(estructuras de programas) capaz de manejar cualquier tipo de información de una manera óptima.

(c) **Utilización del Lenguaje Natural.-** Desde luego, esta ambición no se conseguirá en un 100% hasta dentro de muchos años. Sin embargo, con lenguajes naturales limitados se pueden lograr tales objetivos sin demasiados problemas.

El proyecto propone la consecución del LN en dos direcciones:

1.- Comprensión del usuario.

La computadora sea capaz de comprender lo que el usuario le mande en un lenguaje común.

2.- Generación de mensajes.

La computadora sea capaz de generar la información que suministra el usuario utilizando lenguaje del propio usuario; incluso de viva voz, es decir, que la computadora sea capaz de hablar.

(d) **Arquitectura tipo paralelo.-** Hasta ahora las computadoras trabajan de una forma secuencial, es decir, el microprocesador realiza las tareas asignadas una por una. En este punto se pretende conseguir que existan muchos microprocesadores que realicen varias tareas a la vez, pero sin que dependan siempre de una maestra. Esto resulta muy difícil de conseguir a corto plazo, no obstante se logrará un gran avance en la técnica informática.

En un futuro, cuando comiencen aparecer los primeros frutos de este proyecto se producirá seguramente una revolución dentro de la naciente relación entre el usuario y la computadora. Tal relación ayudará por una lado a la resolución de situaciones que antes, por su complejidad, eran casi imposible de tratar mediante la programación tradicional, y por otro lado resumirá y simplificará la información, destacando aquello que sea realmente importante para las necesidades del usuario.

Si se consigue enseñar a la computadora lo suficiente del LN como para realizar una comunicación real entre el hombre y la máquina, se eliminarán dos grandes inconvenientes que constituyen la necesidad de una interfase en LN para su comunicación:

1.- El intermediario humano. Es difícil que el informático comprenda realmente lo que el usuario quiere preguntar, debido fundamentalmente, al distimo conocimiento del tema que poseen ambas personas.

2.- Tener que enseñar al usuario, profesional en otros campos, los lenguajes propios de la computadora.

Para terminar este tema, diremos que el estudio del PLN por parte de los paises que van a la vanguardia tecnológica, se debe principalmente a sus exitosas investigaciones y avances referentes a la IA. A pesar de la capacidad que sabemos que tienen estas potencias para lograr lo que se proponen, nosotros creemos que difícilmente pueda haber una comunicación hombre-máquina sin límites, debido a que una computadora nunca podrá ser tan compleja como lo es el cerebro humano, ya que éste se considera la entidad más compleja y complicada en el universo. Este impedimento dará pauta a que se lleve una comunicación limitada, que aunque sea bastante aceptable dentro del PLN, no lo es lo suficiente, como si se realizara una comunicación entre seres humanos.

I.6.- REGLAS SINTACTICAS Y SEMANTICAS

Al estudiar nuestra lengua española nos damos cuenta que las palabras tienen diferentes significados, entonces nos podríamos preguntar ¿cómo es posible que nos entendamos?. Bueno, lo que pasa es que escogemos el significado de las palabras siguiendo un conjunto de reglas sintácticas y semánticas o lo que también podemos llamar contexto.

El significado de sintaxis se compone del prefijo griego **sin**, que quiere decir "con", y **taxis** que significa "orden", "arreglo" o "disposición". Por lo tanto, las reglas sintácticas son las que definen el orden en el que pueden aparecer las palabras. El significado de semántico viene del griego **semantikós** que quiere decir significativo. Por lo tanto las reglas semánticas se encargan de identificar el significado de las palabras del LN.

Si juntamos varios sonidos distintivos, es decir, fonemas, y los relacionamos intimamente con un significado, habremos formado un signo lingüístico, o sea una palabra. Las palabras a su vez se unen para formar oraciones o frases. De unir sonidos con sonidos resultan palabras; y de palabras con palabras, los enunciados. Pero es evidente que estas combinaciones están sujetas a reglas específicas. Por lo que el conjunto de sonidos **ilpalliso** carece de sentido, como tampoco podemos reunir palabras a nuestro gusto, pues no tiene sentido decir **casa la tropieza gato**, ni construir enunciados como **la naranja se comió al perro**.

Por lo anterior, se concluye que no es posible comunicarnos de esta manera, y por esa razón, requerimos de reglas gramaticales para poder construir enunciados que hagan posible la comunicación verbal. Así tenemos que esta comunicación no se da con palabras aisladas, ni siquiera con cualquier tipo de combinación de palabras como lo dijimos anteriormente, sino que se da con oraciones. Llamamos oración gramatical a un enunciado compuesto por dos miembros, uno de los cuales dice algo del otro, y que se basta por sí mismo como unidad de comunicación al tener sentido completo. Toda unidad de comunicación, al ser una oración gramatical, está formada por dos miembros íntimamente relacionados a los que llamamos sujeto y predicado.

Si digo Juan lee, expreso una idea concisa, cuyo sentido nadie tiene duda. Pero tal idea es susceptible de ampliarse en diversas formas. A las preguntas inevitables: ¿qué lee Juan?, ¿cuándo lee Juan?, ¿dónde lee Juan?, ¿a quién lee Juan?, etc., contestaremos Juan lee un libro de misterio, todas las noches, en su recámara, a sus hermanos.

En la oración inicial, en su forma más sencilla, Juan lee, como en la más elaborada, Juan lee un libro de misterio, etc., la persona o cosa a quien atribuimos algo "Juan", es el sujeto, y lo que le atribuimos: "lee", "lee un libro de misterio", se denomina predicado.

Las expresiones complementarias: libro de misterio; todas las noches; en su recámara; a sus hermanos, que en nuestro ejemplo sirven para aclarar qué, dónde, cuándo y a quién le lee Juan, se llaman complementos. Una oración, por lo tanto, debe constar necesariamente de sujeto y predicado, aun cuando en ciertas oraciones el sujeto o el predicado vaya sobreentendido.

En la oración Juan lee un libro de misterio, en la que "Juan" es el sujeto y "lee un libro de misterio" el predicado, se hace una sola afirmación del sujeto, que lee, y por esta razón decimos que tal oración consta de una sola proposición; pero hay oraciones que pueden tener dos o más proposiciones. Por ejemplo: Don Miguel de Cervantes y Saavedra, que escribió el Quijote, nació en Alcalá de Henares en 1547, y murió en Madrid en 1616. Por poco que analicemos esta oración encontraremos que tiene tres diferentes proposiciones:

1) Don Miguel de Cervantes Saavedra escribió el Quijote.

2) Don Miguel de Cervantes Saavedra nació en Alcalá de Henares en 1547.

3) Don Miguel de Cervantes Saavedra murió en Madrid en 1616.

A este conjunto de oraciones, que se hallan entre sí estrechamente relacionadas, se le llama cláusula o período, o simplemente oración compuesta.

A continuación daremos una clasificación de las partes de una oración o clases de palabras:

- **Sustantivo** : Designa a los seres vivos y a las cosas que tienen existencia autónoma. Una de sus funciones consiste en ser sujeto de la oración.
- **Verbo** : Señala las acciones y los estados del sujeto situándolos en el eje del tiempo. Normalmente es la parte esencial del predicado.
- **Adjetivo** : Expresa alguna cualidad, circunstancia o condición del nombre. Entre sus funciones está la de modificar al sustantivo.
- **Adverbio** : Señala las circunstancias de lo que expresa el verbo y determina una cualidad. Modifica al adjetivo, a otro adverbio y al verbo sólo con carácter circunstancial.
- **Preposición** : Pone en relación dos elementos e introduce complementos en la oración. El primer elemento se llama núcleo y es subordinante; el segundo, subordinado, se denomina término.

♦**Conjunción :** Esta partícula une en una oración o frase elementos equivalentes, o sea elementos de la misma forma y función.

Cabe señalar que el pronombre no se consideró en esta lista porque realiza las funciones propias del sustantivo, del adjetivo o del adverbio. Tampoco el artículo se consideró porque suele agruparse con el adjetivo, indicando que lo que le sigue es un sustantivo, anunciando su género y número.

Veamos un ejemplo para poder distinguir con más claridad la función de cada una de las palabras dentro de la oración.

Fernando y yo fuimos ayer a la exposición de computadoras modernas.

El sustantivo **Fernando** y el pronombre **yo** (o sea los sujetos de la oración) están unidos por la conjunción **y**; el verbo (núcleo del predicado) es **fuimos**, y está modificado **por ayer**, que es un adverbio de tiempo. Son sustantivos **exposición** y **computadoras**; **modernas** es un adjetivo y **de** es preposición.

Las variaciones que las palabras sufren en su forma se llaman gramemas, y son los siguientes: género, número, tiempo, modo, aumentativo y diminutivo para sustantivos y adjetivos. A la raíz de toda palabra se le llama lexema que representa el significado general.

♦**Género** : Sirve para indicarnos el sexo al que pertenecen las personas o animales y está señalado por la terminación del sustantivo; a está terminación se le llama gramema de género. Los nombres de cosas también tienen gramema de género, pero en la realidad no implican ninguna distinción.

El gramema de género se divide en masculino y femenino. Según su terminación son género masculino las palabras que terminan generalmente en o y son femenino las que terminan en general en a.

♦**Número** : Es el gramema que sirve para indicar si una palabra se refiere a una sola idea, persona o cosa, o a varias.

El gramema de número se divide en singular y plural. Es singular siempre que hace mención de un solo ser u objeto, y plural si son dos o más.
Por ejemplo, en la palabra juegos tenemos:

lexema : **jueg**

gramema de género : **o** (masculino)

gramema de número : **s** (plural)

En los verbos el gramema de número está dado por las personas gramaticales. Para el singular tenemos tres personas gramaticales que son: yo, tu y el; y para el plural tenemos también tres personas gramaticales que son: nosotros, ustedes y ellos.

♦**Tiempo** : El tiempo es un gramema exclusivo de los verbos. Si la acción verbal, en relación con el que habla, ocurrió ya (pasada), decimos que el verbo está en tiempo pretérito; si ocurre ahora (actual), está en tiempo presente, y si es algo que va a ocurrir (venidera), en tiempo futuro.

Pero no basta con estos tres aspectos temporales de la acción verbal; hay tiempos los cuales son anteriores con relación a otro hecho o posteriores a él; es decir, si queremos expresar que una acción sucedió al mismo tiempo que otra, todo en un tiempo pretérito, usaremos el copretérito; si la acción expresada por el verbo es posterior a otra ya pasada, se emplea el pospretérito.

Por ejemplo, en el verbo salir:

gramema de tiempo:

salí	(pretérito)
salgo	(presente)
saldré	(futuro)
yo **salía** cuando tú llegaste	(copretérito)
me aseguró que **saldría** hoy o mañana	(pospretérito)

♦**Modo** : Es la manera de expresar la acción o el hecho contenido en el verbo. Tres son los gramemas de modo que son exclusivos del verbo:

a) Indicativo: Aquí expresamos la acción en una forma objetiva y real sin que el que habla tenga otra actitud que la de indicar el hecho.

Por ejemplo:

yo camino
Carlos vendrá
él salía los domingos

b) Imperativo: Aquí la acción contenida en el verbo se expresa en una forma imperativa, esto es, una orden o un mandato que expresa la voluntad del que habla.

Por ejemplo:

vengan, escuchen los que les digo
paga lo que debes
cállate

c) Subjuntivo: expresa la acción contenida en el verbo a manera de creencia, esperanza, deseo, posibilidad, y esta acción está siempre subordinada a otro verbo que expresa alguna idea de negación, duda, necesidad, probabilidad. Expresa la acción no como real, sino como pensada por el que habla.

Por ejemplo:

no quiero que vayas a
ningún lado. (negación)
quizá yo tenga éxito en mi vida.(duda)
ojalá fuera como tu lo dices. (deseo)

♦**Aumentativo:** Se llaman gramemas aumentativos los que modifican a la palabra primitiva, añadiéndole un rasgo para acrecentar su significación.

♦**Diminutivo :** Se llaman gramemas diminutivos los que modifican a la palabra primitiva, añadiéndole la cualidad de reducir a menos su significación.

La significación tanto para el gramema aumentativo como para el diminutivo se da en tamaño, intensidad, estimación y menosprecio.

Por ejemplo, en la palabra mamá tenemos:

gramema diminutivo : mamacita (estimación)

Y en la palabra silla:

gramema aumentativo : silloia (ramaño)

Todas las reglas que mencionamos con anterioridad sólamente representan una pequeña muestra de lo que realmente es la gramática de la lengua Española. De esta manera podemos concluir que nuestra lengua es sumamente rica en gramática, ya que nos pasaríamos toda la tesis hablando de lo extenso y complicado que es la gramática del español. No obstante que no es la intención de esta tesis, creímos que era conveniente dar un panorama general de las reglas gramaticales que conforman nuestro idioma, para que así, el lector logre entender los grandes problemas que ocasionan el querer establecer una buena comunicación hombre-máquina en L.N.

I.7.- LIMITANTES EN LA COMPRENSION DEL LENGUAJE

El lenguaje ha sido alusivo al análisis de diversos especialistas en informática. Aún cuando el LN es dominantemente hablado, este proyecto se referirá únicamente al lenguaje escrito. Esta limitación es debido al trabajo que existe en la IA concerniente a la comprensión y generación del lenguaje hablado. De esta restricción se derivan dos ventajas a favor del lenguaje escrito:

a) El formato de análisis del texto, conocido como tratamiento de cadenas de caracteres es relativamente manejable a comparación del tratamiento del lenguaje hablado.

b) Las dificultades inherentes al análisis de los sustratos más bajos del lenguaje hablado (entonación, tiempos, énfasis, dicción, etc.) se evitan, simplificándose así la tarea.

Sin embargo el PLN escrito posee también una gran cantidad de dificultades, dentro de las cuales destacaremos a la ambigüedad. Así, la ambigüedad es el peor enemigo para la programación del PLN, es decir, la interpretación y representación del conocimiento. Los programas manipulan símbolos lingüísticos con gran facilidad como en el procesamiento de textos, sin embargo los esfuerzos por conseguir que las computadoras manejen el significado se ven frustrados por la ambigüedad de los lenguajes.

Asimismo, es posible encontrar fuentes de ambigüedad virtualmente en cualquier nivel de análisis del lenguaje [WIN84]. Al momento de implementar programas para el PLN, estos tendrán una gran variedad de errores, que en ocasiones darán resultados desastrosos.

El significado de una frase en LN no depende sólo de la forma de la oración, sino también de su contexto. Según Terry Winograd [WIN84] existen cinco tipos de ambigüedad las cuales son:

a) La ambigüedad léxica

En la ambigüedad léxica una palabra puede tener más de un significado posible, por ejemplo la palabra banco <para sentarse>, banco <Institución financiera>, banco <de datos>, banco <de sangre>, etc. Al intentar solucionar la ambigüedad léxica en el PLN., se ha atendido a la inserción de todas las alternativas, así como el análisis estadístico del texto original, para decidir que interpretación es la correcta.

b) La ambigüedad estructural

En la ambigüedad estructural una misma frase puede tener distintos significados; por ejemplo, en la frase "**La burra de Laura se cayó**" se puede interpretar a simple vista tres estructuras gramaticales posibles que según su contexto pueden ser.

1.- "La burra" (por tonta) de Laura se cayó.

2.- "La burra" (como animal) de Laura (como propiedad) se cayó.

3.- "La burra" (nombre utilizado para referirnos a un instrumento que se utiliza en el juego de billar) de Laura se cayó.

Podemos observar que a pesar de que las palabras que constituyen la frase son iguales, las estructuras gramaticales son diferentes. Por lo tanto, para solucionar esta ambigüedad, es necesario que el expositor (usuario) sea más explícito con la computadora (interprete). También, la computadora debe ser capaz de dar a elegir las diferentes estructuras gramaticales que se establecen de la frase, para que el usuario elija entre ellas y por lo tanto rompa con la ambigüedad .

c) La ambigüedad estructural profunda

En la ambigüedad estructural profunda una frase puede tener las mismas palabras y la misma estructura gramatical, pero con varios significados posibles; por ejemplo, en el idioma inglés la frase "**Los pollos están listos para comer**" implica como estructura gramatical: que algo esta a punto de comer algo; pero, ¿los pollos comerán o serán comidos?. Así tenemos dos posibles significados: que los pollos están listos para comer o los pollos están a punto de comer.

Uno de los avances en la teoría lingüística desde los años cincuenta ha sido el desarrollo de un formalismo que pueda representar la estructura profunda del lenguaje, pero este formalismo ayuda muy poco a determinar la estructura profunda de una frase concreta.

d) La ambigüedad semántica

La ambigüedad semántica se da cuando el receptor le asigna diferentes significados a una frase dentro de una oración; como consecuencia el significado de la oración cambia, por ejemplo en la frase "Javier quiere casarse con una americana", puede tener varios significados dependiendo del contexto. Si Javier es europeo, la palabra "americana" puede referirse a una habitante del continente americano. Si Javier nació en el continente americano, es muy probable que la palabra "americana" signifique una ciudadana estadounidense. Además tenemos los significados que examina Winograd: la oración puede

indicar la intención, más bien abstracta de parte de Javier, de casarse con una mujer cualquiera de cierta nacionalidad. También puede significar que el hablante ha elegido un atributo la nacionalidad de la prometida de Javier para describirla.

e) La ambigüedad pragmática

La ambigüedad pragmática surge del uso de los pronombres y depende del conocimiento del mundo que se tenga, es decir, si se dice la oración "tiró el plato en la mesa y se rompió", casi nadie dudaría que fue el plato el que se rompió. Sin embargo una computadora no tendría la habilidad para hacer esa distinción.

A diferencia de Winograd [LLO87], para otros especialistas en informática existen sólo dos tipos básicos de ambigüedades en el PLN y son:

a) Ambigüedad referencial

Por ejemplo, en la frase "él vino" puede tener varios significados dependiendo del contexto en que se encuentre.

- Primer significado: "Paco ordenó a Héctor que viniera y él vino". En este caso sabemos, por toda la frase, que "él vino" significa que Héctor vino.

- Segundo significado: "Paco le dijo a Héctor: vamos. Sin embargo, Héctor se marchó y sólo él vino". En este caso Paco es el que viene. Observamos como la misma frase puede significar cosas distintas dependiendo del contexto en que se encuentre.

b) Ambigüedad por el sentido de las palabras

Por ejemplo:
- "Gerardo cogió una pelota"
- "Gerardo cogió un resfriado"
- "Gerardo cogió un cogorza"

Se puede observar cómo la palabra es la que crea ambigüedad, ya que el significado de la frase depende de lo que "cogió" Gerardo. Esta ambigüedad es muy difícil eliminarla y tal vez sea, precisamente, la que retrase en muchos años que las computadoras puedan establecer un diálogo con los seres humanos.

Estos problemas se dan en cualquier lenguaje y se complican más en el caso de la traducción. El programa que realice la comunicación entre el hombre y la máquina debe de contener no sólo un amplio léxico, sino también, debe poseer las principales estructuras gramaticales y sus correspondencias. Por lo anterior, solamente se ha dado un ligero avance a nivel de sintaxis y significado en la programación. Sin embargo, en la traducción todavía no tenemos un gran avance, y esto se debe esencialmente a la ambigüedad que existe en los diferentes lenguajes, ya sea a nivel escrito o hablado.

El futuro de la IA se ve prometedor a corto tiempo, sin embargo, a partir del análisis de lo que es posible y de lo que no, hoy día en que la tecnología no ha hecho más que comenzar, las aplicaciones son muy restringidas; los sistemas basados en el PLN, únicamente son capaces de enfrentarse y resolver problemas dentro de una área muy delimitada. No son capaces de razonar a partir de axiomas o teorías generales y sólo aceptan un tipo de hechos y heurística determinados. No tienen la capacidad de desarrollar, la facultad de aprender ni pueden razonar por analogía. La consecuencia de todos estos factores se resume en 2 características:

a) La falta de sentido común.

b) Sus razonamientos se deterioran rápidamente (cuando el problema se sale del área del conocimiento delimitado).

Sin embargo dentro de lo positivo, podemos decir que los sistemas basados en el PLN no realizan juicios incoherentes, no se saltan ninguna posible alternativa ni se empeñan en mantener una postura en contra de los hechos reales. Tampoco tienen malos momentos, esto es, a comparación de los seres humanos donde su capacidad es afectada por diversos factores sentimentales, las máquinas nunca bajan su rendimiento; por otro lado, siempre tienen en cuenta todos los detalles y sistemáticamente consideran todas las posibles alternativas.

Para un ser humano, el entender algo, sea lo que sea, le resultaría imposible si no contara con inteligencia, pero además, es necesaria cierta información para llevar a cabo el proceso de comprensión. Entender el lenguaje que hablamos, nos llevó a la mayoría de los seres humanos cerca de tres años, a partir de los cuales ya podíamos mantener un diálogo hilvanado y coherente, ya que entendíamos lo que se nos decía y de la misma manera, habíamos almacenado en nuestra memoria la información suficiente sobre la estructura del lenguaje y su significado. Aunque quizá dicha información sea mejor llamarle "Saber", ya que si tomamos la siguiente definición del saber, ésta nos facilitará el interpretar un poco mejor lo que estamos hablando.

El "saber" según Feigembuam se define como "... Información recortada, modelada, interpretada, seleccionada y transformada; el artista que hay en cada uno de nosotros recoge diariamente la materia prima y la convierte en un pequeño aparato, y al mismo tiempo en una pequeña gloria humana" .

Un ser humano capta diariamente una fabulosa cantidad de información acerca de su mundo circundante, y a través de los sentidos le toma como ya dijimos aproximadamente tres años en comprender (a un nivel aceptable) su idioma. Y con esto planteamos otra limitante para la comprensión del LN que es el de poder sintetizar el saber humano sobre el LN en alguna estructura que pueda ser implementada, para ser manejada por la computadora en algún lenguaje de programación y por tanto poder llevar a cabo una comunicación en LN

Ahora, al analizar cualquier lengua de nuestro planeta, nos encontramos con que cuenta con una gran cantidad de palabras (decenas de miles) y una asombrosa cantidad de maneras en las que pueden combinarse para formar oraciones, para lo cual cada ser humano cuenta con la capacidad de memorizar un alto número de ellas, además de almacenar su significado, la pronunciación y algunas características propias de cada lengua. Añadiendo unas complicaciones más al problema de la interpretación del LN.

De lo anterior, podemos plantearnos dos importantes preguntas, que son de rigor al inicio del desarrollo de todo trabajo que involucre el PLN.

- ¿Cómo almacenaremos las palabras que vamos a manejar en el computador?
- ¿Cuál estructura facilita el acceso a la base de datos lo más rápido posible?.

Mas adelante se contestarán estas dos preguntas para nuestro caso particular.

Dentro de las limitantes en la IA, hay una que destaca por su importancia, y es el hecho de aseverar que la computadora pueda llegar a ser tan compleja como lo es el cerebro humano. Asi podemos concluir que el cerebro:

a) es compleja porque tiene muchas partes y funciones

b) es complicado porque éstas se interconectan de muchas maneras.

En términos informáticos, el cerebro humano es algo como un ensamble de miles de millones de computadoras, todas comunicándose entre si y procesando información al mismo tiempo.

Regresando a las limitantes del PLN, podemos decir que uno de los principales problemas para enseñar a una computadora a comprender el LN, es el de su contexto. Debido a que el significado de una frase en el LN no depende sólo de la forma de la oración, sino también de su contexto. Asimismo, se presentan otros problemas al momento de interpretar el LN los cuales son: las expresiones idiomáticas, los modismos y la jerga. Un ejemplo que prueba el problema de hacer que las máquinas comprendan el LN, fue el que señaló un pionero en el estudio del LN. El propuso la siguiente prueba, la cual consiste en comprender las siguientes dos frases "la caja está en la pluma" y "la pluma está en la caja". Estas frases no plantean problemas para un ser humano, sin embargo para una computadora las dificultades son enormes.

El siguiente razonamiento habla de una limitante que no sólo es para las computadoras, sino también, para el ser humano, ésta se basa en tomar una decisión totalmente óptima. En este sentido, a los seres humanos nos gusta creer que algunas de nuestras decisiones más importantes, las tomamos por razonamiento lógico. Sin embargo, no podemos actuar de manera completamente lógica, ya que tendríamos que recolectar toda la evidencia relacionada con una decisión específica, posteriormente sopesarla, para al final dar una decisión certera y eficaz. Pero ninguno de estos pasos es posible que pueda realizarlo el ser humano. En primer lugar, rara vez hay suficiente evidencia para garantizar una conclusión como cierta. Luego la evidencia disponible, por lo general es demasiado detallada, elemento por elemento para poderse equiparar con los demás elementos. Así tenemos, que para dar una decisión lógica, ésta no debe carecer de ninguna evidencia, pero por otro lado, si tuvieramos toda esa información, sobrepasaría su capacidad para decidir. Por lo tanto si una persona tuviera que decidir lógicamente si es seguro tomar un avión de la Ciudad de México a la Ciudad de Monterrey; no sólo necesitaría saber el registro de seguridad de la línea aérea, sino también, todo acerca del aeroplano, fabricante, sus pilotos, condiciones meteorológicas en ambas ciudades y durante el recorrido, y mucho más.

La lógica, definida de manera precisa sólo existe en las matemáticas y en el mundo abstracto de los silogismos (todos los hombres respiran; Carlos es un hombre, por lo tanto Carlos respira).

A manera de resumir, diremos que en el mundo real o empírico diario de la ciencia y de las percepciones sensoriales, no existe certidumbre. Todas nuestras decisiones y opiniones inteligentes se basan en información que no esta totalmente completa, que es insuficiente para la lógica. Entonces, nos preguntaríamos ¿como decidirá una persona si su viaje es seguro?, la respuesta es, con base a la experiencia y a las probabilidades. Ya ha volado antes y nada sucedió. La estadística anual del gobierno dice que es más probable sufrir un accidente en un automóvil que en un avión. Así es como verdaderamente realizamos nuestras decisiones inteligentes, recolectando parte de la evidencia posible y a través de un acto de fe guiado por la experiencia pasada, por inspiración divina, las probabilidades, y la tranquilizadora y hogareña sabiduría popular: "el que madruga Dios le ayuda"; "la ropa de lana lavada con agua fría no encoge"; "si escribes a máquina tu trabajo final, obtendrás una mejor calificación"; etc.

De manera informal, estas técnicas se conocen como reglas prácticas; de manera formal, se les denomina Heurística, palabra que viene del griego y que significa "servir para descubrir". La heurística es el arte de inventar o descubrir hechos valiéndose de hipótesis o principios que, aún no siendo verdaderos, estimulan la investigación. Además son técnicas que no tienen garantía de éxito, pero todos las utilizamos porque son necesarias.

Finalmente, mencionaremos que dentro del lenguaje hablado:

- La acústica estudia las características de los sonidos individuales.

- La fonética estudia la manera en que los sonidos se combinan para formar sílabas y palabras.

Dos objetivos importantes que trascienden de la investigación de estos dos campos serían:

- Lograr que las computadoras sirvan a los invidentes.
- Comunicarse con la computadora a través del habla y no escribiendo en el teclado.

Asimismo, dentro del lenguaje escrito:

- La gramática es el sistema de reglas que rige las formas en que se utiliza un lenguaje. Las reglas gramaticales están bien establecidas en la mayor parte de los lenguajes naturales.
- La sintaxis es la forma en que las palabras se juntan para formar oraciones correctas. La computadora maneja bien la simaxis. Puede hacerse que comprenda la estructura de una oración y que produzca oraciones correctas desde el punto de vista sintáctico. Por ejemplo, la computadora sería capaz de decidir que la frase "nosotros ellos jugamos con" es incorrecta sintácticamente hablando.
- La semántica se ocupa del significado de las palabras y de la forma en que éste se relaciona con los posibles significados de la oración y con su sintaxis. Para aplicaciones prácticas, la semántica se define como el estudio del contexto, ambiente de palabras y frases. En esta área es donde las computadoras han tenido tantas dificultades para la comprensión del lenguaje.

Para terminar este capítulo, podríamos concluir diciendo que hoy día en nuestros tiempos, con todos los avances tecnológicos en el hardware y más de medio siglo investigando herramientas que faciliten la programación de PLN; programar un eficiente PLN implica un reto bastante difícil de alcanzar. Hasta ahora la barrera no ha sido franqueada y aunque no pretendemos llegar a tocarla, si buscamos al menos dar un paso hacia el pleno desarrollo del PLN.

Tema II

Técnicas más usadas en el procesamiento del lenguaje natural

"Hay que estudiar mucho para
saber poco"

BARÓN DE MONTESQUIEU.

II.1.- INTRODUCCION.

Los grandes sueños del hombre por lograr implementar una técnica eficiente para el procesamiento del lenguaje natural (PLN) no se ha visto satisfecho, ya que la complejidad de la gramática del idioma sea cual sea, es aún una gran barrera para lograr el diálogo hombre-máquina completo y satisfactorio.

Sin embargo por el esfuerzo no ha quedado, se han hecho estudios bastante interesantes para el PLN y los frutos aunque contados, son muestra de lo dicho. En el capítulo anterior se mencionan algunos de ellos y ahora toca discutir las bases con las que fueron diseñados, obviamente siguen un método o técnica y es ahí donde se centra el contenido de este capítulo.

Durante ya más de 20 años se han dado a conocer tan solo un puñado de técnicas sobre el PLN, y como ya se dijo no se encuentra aún la ideal. Pues bien dentro de este campo queda por realizar mucha investigación y más aún, si consideramos que la mayor parte de trabajos está en base al idioma Inglés, lo que nos deja en una posición de pioneros en lo que respecta a nuestro idioma, y aunque los antecedentes existen en nuestro país, se han seguido casi al pie de la letra las técnicas previamente desarrolladas en el extranjero.

En este capítulo describimos las técnicas más conocidas a un nivel introductorio o explicativo, debido a que consideramos que sólo nos sirven para enfocar las diferentes corrientes existentes y apreciar en su real dimensión el camino que nosotros hemos elegido para basar nuestra investigación.

No esperamos descubrir el hilo negro describiendo una técnica nueva, esto es algo demasiado complejo como para intentarlo, pero si, el adaptar a nuestras propias necesidades alguna de las ya existentes.

Los temas a tratar están ordenados de tal manera que van de lo más simple a lo más complejo, así como también de lo "viejo" a lo "nuevo". Con esto intentamos marcar las caracteristicas de cada técnica y además que se pueda observar la evolución existente en el complejo problema del PLN.

II.2.- COMPARACION DE PATRONES

Un componente importante en muchos programas de I.A. es la técnica de comparación de patrones, ya que les permite ejecutar tareas complejas con un método sencillo y flexible. La comparación de patrones es una de las más antiguas técnicas de I.A. utilizada en la década de los 60's en los primeros trabajos sobre el PLN. En ELIZA y PERRY encontramos dos ejemplos representativos de utilización de ésta.

Esta técnica es de fácil comprensión y la idea técnica consiste en tener un dato y un patrón que serán comparados para determinar si el dato empata con el patrón. El dato contiene al patrón si se cumple una de las siguientes dos reglas :

a) El dato es idéntico al patrón.
b) El dato puede ser un caso del patrón.

Este proceso de comparación que involucra o no, casos en el patrón es lo que llamamos comparación de patrones.

La regla a) es de fácil comprensión y como ejemplo de programas que utilizan sólo esta regla son los correctores de ortografía, que buscan las palabras (datos) en diccionarios (patrones) y al no encontrarlas nos señala esa palabra como desconocida.

La regla b) requiere de otras consideraciones, como el uso de caracteres comodín que representan uno o más caracteres, ejemplo:

Sea: * comodín para sustituir de 1 a más caracteres.
Con el comodín anterior definamos el siguiente patrón:

te * ayer

Por lo tanto los datos:

te busque ayer
te vine a buscar ayer
te amo desde ayer

Son datos de un caso del patrón, ya que contienen el patrón haciendo uso del comodín. Un ejemplo de un programa que utilice la comparación de patrones con la regla b son los sistemas operativos que para algunas instrucciones permiten la utilización de patrones con comodines para llevar a cabo comparaciones con datos. Ejemplo:

MS-DOS	**Descripción**
Dir *.DAT	Lista todos los archivos con extensión "DAT" en el directorio por omisión
UNIX	
ls *.DAT	Lista todos los archivos con extensión "DAT" en el directorio por omisión

Los sistemas operativos utilizan la comparación de patrones con las 2 reglas, por ejemplo, para los comandos la regla a), el dato es idéntico al patrón, y para algunos parámetros la regla b), el dato puede ser caso del patrón.

Las ventajas que se tienen al utilizar esta técnica son entre otras las siguientes:

- Fácil de comprender
- Fácil de programar en cualquier lenguaje
- Rapidez en la búsqueda de patrones
- Posibilidad de tratamiento de oraciones de gramática compleja.
- Transportabilidad de un dominio a otro
- Fácil extender el dominio

Las desventajas desafortunadamente superan en peso a las ventajas. Listaremos sólo algunas:

- Realiza un análisis superficial
- Carece de Análisis sintáctico, semántico, etc.
- No detecta ambigüedades
- Es posible introducir incoherencias que cumplan con un patrón y serán aceptadas como válidas
- Limitan al usuario a los patrones establecidos
- etc.

Por lo anterior es que la gran parte de programas que utilizan esta técnica la combinan con otra para obtener mejores resultados.

A pesar de las desventajas que esta técnica presenta, se utiliza en los modernos programas de PLN, ya que la comparación de patrones no es solamente una forma de accesar datos, es también una técnica para el direccionamiento de control del flujo en un programa. Por ejemplo, un sistema experto típico, selecciona las reglas que debe aplicar, de acuerdo al esquema de comparación de patrones.

Otra aplicación se observa en los sistemas que comparan un nuevo conocimiento con el ya almacenado, para decidir si es agregado como un hecho a la base de conocimientos, o es desechado por su redundante o contradictorio.

En resumen, podemos considerar a la técnica de comparación de patrones, como básica, pero por sus limitantes, es necesario combinarla con alguna otra técnica de PLN para fortalecer los aspectos que esta técnica por sí misma no cubre satisfactoriamente.

La comparación de patrones fue de las primeras técnicas en PLN, como ya se mencionó, en aparecer en el mundo de la IA así mismo consideramos que será por sus características una de las que sobreviva por mucho tiempo más en combinación con alguna otra.

II.3.- REDES DE TRANSICION AUMENTADAS

Las redes de transición aumentadas (Aumented transition networks: RTA) proporcionan un método fácil de reconocimiento de frases en forma eficiente. Son un concepto más refinado de las máquinas de estados finitos.

"Una máquina de estados finitos, es una estructura en donde la computación se modela como la transición de un estado a otro, de un número finito de ellos" [RIC83]

Calificamos las RTA's como un producto de la evolución de las máquinas de estados fiuitos, porque los ATN son en realidad una ampliación a éstas. Las ampliaciones consisten básicamente en:

- Los arcos de transición de un estado a otro pueden ser llamadas a otras redes que reconozcan componentes importantes de una frase. Incluso estas llamadas pueden ser recursivas a la red que realiza la llamada.

- Los arcos pueden estar etiquetados con palabras específicas.

- La transición de estados puede ser mediante la realización de algún(os) procedimiento(s) para la comprobación de la entrada actual.

- Los procedimientos pueden ser además constructores de estructuras para un análisis posterior.

Esto hace que una RTA por la variedad de condiciones que es posible asociar a un arco, tenga la potencia formal de la máquina de TURING. Las RTA's fueron desarrolladas en los 70's. Es también una técnica muy usada en el PLN para el análisis de gramáticas. El funcionamiento de una RTA es en pocas palabras el siguiente:

- Consideremos que una RTA esta constituida por una serie de estados conectados por arcos.

- Cada arco está etiquetado por una categoría gramatical o una palabra específica.

- Se comienza en un estado definido como estado inicial, que compara la cadena de entrada con los arcos asociados a este estado. Si la comparación es satisfactoria la RTA procede al estado que nos conduce al arco que cumplió con la condición, así sucesivamente hasta atravesar la red.

- Si en algún estado, al llevar a cabo las comparaciones con los arcos, no se satisface ninguno de ellos, la RTA simplemente se define y marca el error.

La técnica de RTA's, por su naturaleza, puede utilizarse en cualquier gramática, ya que el formalismo de las RTA's no contienen en si mismas la gramática. Las RTA's son un mecanismo mediante el cual puede definirse y usarse una gramática.

Las RTA's pueden realizar el análisis de frases con la consideración de las siguientes tres alternativas:

1)Regreso (Backtraking), se realiza almacenando en cada arco la información relacionada con la comparación entre la cadena de entrada y el contenido de todos los registros del sistema, así como el puntero de la cadena de estado actual. Si aparece una vuelta atrás en ese punto, puede restaurar los valores apropiados de la cadena de entrada y de los registros.

2)Remiendos, Se adjuntan segmentos de código a cada uno de los arcos para mover los contenidos de un registro de un sistema a otro.

3)Esperar y ver, Se establece un registro del sistema para que sirva de vector en el cual se colocan aquellos componentes cuya función aún no se ha determinado. Los contenidos de ese sector pueden moverse a otros registros mediante el código asociado a los arcos apropiados. La adecuada combinación de las anteriores alternativas es la que facilita que una gramática completa sea fácil de construir y sea interpretada eficientemente.

Al analizar a la técnica de RTA podemos observar que es una herramienta útil para el análisis de la sintaxis de una gramática dada. Las gramáticas y las RTA's son independientes en su diseño, pero son pasos secuenciales, de las cuales, el primero es, definir la gramática en alguna de las formas comúnmente utilizadas, como son las de producciones en forma normal de Backus. Una vez definida la gramática, es relativamente fácil el diseño de la red completa de RTA's. El lenguaje de programación puede ser cualquiera de propósito general que permita la recursión y la programación estructurada, pero recomendamos PASCAL o C, dado que facilitan enormemente el desarrollo.

Los RTA's han demostrado su utilidad en diversos sistemas de PLN, sin embargo tienen algunos puntos a considerar para aumentar su eficiencia y no se conviertan en desventajas.

- Generalmente para una gramática amplia se tienen sentencias localmente ambiguas haciéndose imprescindible la vuelta atrás. Si se abusa de la vuelta atrás, puede resultar costoso, ya que se requerirá más memoria y tiempo para el análisis.

- Es posible la utilización de conocimiento semántico para elegir la ruta más probable de ser explorada en primer lugar. Sin embargo debido principalmente a su complejidad no se ha encontrado una forma de utilizar métodos heurísticos.

- La RTA fracasará si no logra identificar todas las palabras de la frase y la estructura de la misma no empata exactamente con alguna de las rutas de la red. No es posible realizar un empate parcial. Para hacerlo se requieren técnicas aún más flexibles.

II.4.-PRODUCCIONES

Las producciones en si mismas no son una técnica para el PLN, sino herramientas de las cuales el lingüista-programador se vale para representar la estructura del lenguaje o sea la gramática, en la cual se desea llevar a cabo el análisis.

Aunque existen varias formas de representar la gramática, hemos elegido sólo dos de ellas por considerar las de mayor divulgación y fundamentos teóricos. Estas son: la Forma Normal de Backus (BNF Backus-Naur-Form) y gramática por definición de cláusulas (DCG Definite Clause Gramar).

Las dos son analizadas en los siguientes subtemas y se puede apreciar que el objetivo que se persigue en ambos casos es el de representar el conocimiento sobre la gramática con reglas que faciliten su traslado a alguno de los lenguajes de programación, de las cuales, las producciones de BNF están enfocadas a los lenguajes como Pascal, C, Fortran, etc. y la DCG esta desarrollada con la filosofía del PROLOG, por lo que con las dos técnicas complementamos en lo posible las diversas maneras de representar la gramática para los más importantes lenguajes de programación.

II.4.1.-PRODUCCIONES BNF

La notación de producciones de BNF (Backus-Naur-Form) se utilizó por primera vez en la definición de ALGOL 60 dado que permitía estructurar de una manera sencilla una gramática rígida como la de un lenguaje de programación, pero con una diversidad de alternativas. Hasta hoy en día es una notación muy utilizada para definirle al usuario de lenguajes de programación la correcta sintaxis de estos.

Para que sea posible definir las producciones en BNF se deben definir primero los símbolos básicos, los cuales son palabras y la sintaxis a la que estas palabras estarán regidas.

Tenemos por ejemplo la oración: "El mundo cambia". Las palabras "El mundo" determinan el sujeto y "cambia" es el predicado por lo que se puede definir como:

<oración> ::= <sujeto> <predicado>
<sujeto> ::= El mundo | la gente
<predicado> ::= cambia | mueve

Las construcciones <oración> ::= <sujeto> <predicado> son llamados símbolos no terminales; las palabras "El mundo", "La gente", "cambia" y "mueve" son llamados símbolos terminales; y las reglas son llamadas producciones. El símbolo "::=" y "|" son llamados meta símbolos de la notación BNF.

II.4.2.-PRODUCCIONES DCG

El concepto de DCG (definite clause grammar) está íntimamente relacionado con la programación lógica y por tanto hereda sus propiedades. Con las DCG es posible la detección de contextos dependientes, lo cual significa relacionar los elementos de un texto con su entorno. Además, por la semejanza que existe entre una gramática libre y una DCG, se obtiene mayor claridad en su uso que con otras especificaciones.

Considérense oraciones de la forma:

"José platica",
"el reloj camina",
"Juan come rápido",
"el búho duerme poco", etc.

cuya gramática podría ser:

<oración> --> <sujeto> <predicado>
<sujeto> --> <artículo> <nombre> | <nombre>
<predicado> --> <verbo> <adverbio> | <verbo>
<artículo> --> el | la | ...
<nombre> --> reloj | José | ...
<verbo> --> duerme | ...
<adverbio> --> rápido | ...

Podemos especificar una DGC que reconozca el tipo de oraciones que genera la gramática, de manera directa:

```
oración (X Y) --> sujeto (X X1) predicado (X1 Y)
sujeto (X X1) --> nombre (X X1)
sujeto (X X1) --> artículo (X X2) nombre (X2 X1)
predicado (X1 Y) --> verbo (X1 X2) adverbio (X2 Y)
predicado (X1 Y) --> verbo (X1 X1)
artículo ([el | X] X)
artículo ([la | X] X) ...
nombre ([reloj | X] X)
nombre ([José | X] X) ...
verbo ([duerme | X] X) ...
adverbio ([rápido | X] X) ...
```

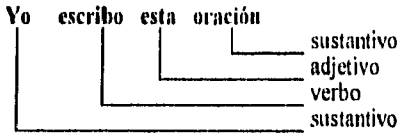
Cada cláusula de la DGC indica "lo que se debe cumplir". Su argumento, es la oración de entrada. Asimismo [] denota una lista. En [X | Y] X e Y son variables: el primer elemento de la lista y la lista residual, respectivamente.

II.5.- ANALISIS POR CASOS

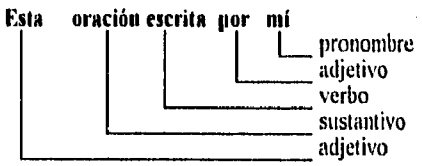
Esta técnica es substancialmente diferente de las que se han descrito hasta el momento, ya que basan su diseño en la definición de una gramática por casos.

Las gramáticas por casos proporcionan un enfoque diferente al problema de combinar las interpretaciones sintácticas y semánticas. Pero las estructuras que producen las reglas, corresponden a relaciones semánticas más que las simplemente sintácticas. Como ejemplo consideremos las siguientes 2 frases.

1) Sujeto Predicado



2) Sujeto Predicado

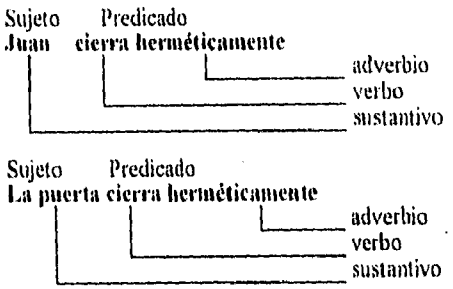


Aunque los papeles semánticos de "yo" y "oración" son idénticos en ambas frases, sus papeles sintácticos están invertidos. Cada uno es el sujeto de una frase y el predicado en otra.

Utilizada la gramática de casos, la interpretación de las dos frases sería:

(Escribo (Agente : Yo)
(Objeto : Esta oración)

Para las estructuras sintácticas de dos frases casi idénticas, puede tener representación gramática diferente, por ejemplo:



Podemos observar que la estructura sintáctica es la misma y sin embargo su representación en gramática de casos para cada una es:

Juan cierra herméticamente

(**Cierra** (Agente : Juan)
(Modo : herméticamente))

La puerta cierra herméticamente

(**Cierra** (Objeto : La puerta)
(Modo : herméticamente))

En esas representaciones se hacen explícitos los papeles semánticos de "Juan" y "La puerta". La conjunción de dos frases paralelas es posible si los dos sintagmas conjuntados están en la misma relación de caso con respecto al verbo. Ejemplo:

La puerta cierra herméticamente

(**Cierra** (Objeto : La puerta)
(Modo : herméticamente))

La ventana cierra herméticamente

(**Cierra** (Objeto : La ventana)
(Modo : herméticamente))

Se convierte al conjugarlas en:

La puerta y la ventana cierran herméticamente.

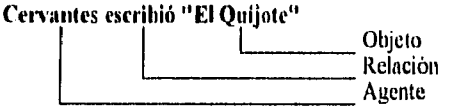
(Cierra (Objeto: La puerta)
(Objeto: La ventana)
(Modo : herméticamente))

Podemos observar que los verbos describen las relaciones entre los verbos y sus argumentos en una gramática de casos. No existe un acuerdo claro sobre cuáles serían los casos semánticos correctos que debe haber, pero podemos definir las siguientes:

(A)	**Agente:**	Investigador de la acción típicamente animado.
(I)	**Instrumento:**	Causa del acontecimiento u objeto usado para causarlo, típicamente inanimado
(D)	**Dativo:**	Entidad afectada por la acción.
(F)	**Factivo:**	Objeto o estado resultante de la acción.
(L)	**Locativo:**	Lugar del acontecimiento
(S)	**Fuente:**	Lugar u origen a partir del cual se mueve algo.
(M)	**Meta:**	Lugar hacia el cual se mueve algo
(B)	**Benefactor:**	Ser en cuyo beneficio ocurre el acontecimiento, típicamente animada
(T)	**Tiempo:**	Momento en el que ocurrió el acontecimiento.
(O)	**Objeto:**	Entidad que cambia o sobre el cual se actúa, el caso más general

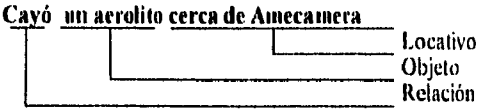
Con los anteriores casos semánticos se pueden analizar las siguientes oraciones.

Cervantes escribió "El Quijote"

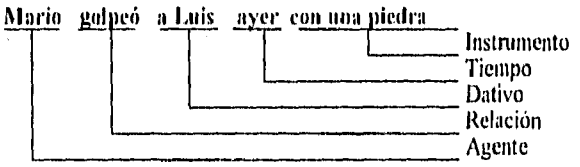


(Escribió (Agente :Cervantes)
(Objeto :"El Quijote"))

Cayó un aerolito cerca de Amecamera



(Cayó (Objeto : un aerolito)
(Locativo : cerca de Amecameca))



(**Golpeó** (Agente : Mario)
(Dativo : Luis)
(Tiempo : Ayer)
(Instrumento: Piedra))

Para ilustrar la versatilidad de este análisis, observemos las oraciones siguientes:

Ayer Mario con una piedra golpeó a Luis.
Con una piedra, Mario golpeó a Luis ayer.
A Luis, con una piedra, Mario golpeó ayer.
Mario golpeó a Luis con una piedra ayer.
Ayer con una piedra, Mario golpeó a Luis.
Con una piedra ayer, Mario golpeó a Luis.
Golpeó a Luis con una piedra Mario ayer.

A pesar de tener todas, una estructura sintáctica diferente, su representación en gramática de casos es exactamente la misma.

Esta técnica ofrece además la facilidad de análisis para oraciones coherentes de sujeto explícito, eligiéndose mediante la siguiente regla:

> **Si esta presente el Agente, es el sujeto. En caso contrario, si está presente Instrumento, es sujeto. Si no, el sujeto es Objeto.**

Un analizador puede aplicar estas reglas al revés. Para determinar la estructura de casos subyacentes a partir de la sintaxis superficial.

Un análisis que use una gramática de casos está conducido por expectativas. Una vez que se ha localizado el verbo de la frase, puede usársele para predecir los sintagmas nominales que aparecerán y para determinar cual; es la relación entre dichos sintagmas y el resto de la frase.

Con esa gramática por casos, es factible la elaboración de redes semánticas para hacer posible el responder preguntas sobre dichas frases.

En conclusión, el análisis del LN mediante una gramática de casos, nos presenta un enfoque diferente y permite el realizar en combinación con RTA's, por ejemplo, un análisis más profundo y completo. Las ventajas que a simple vista podemos observar de esta técnica, son:

- Se determina el papel sintáctico y semántico de cada componente de la frase.

- El resultado del análisis nos conduce en representaciones del conocimiento, como son las redes semánticas y las producciones.

- Los resultados están libres de ambigüedades.

El precio de esta técnica radica en la complejidad de implantación, que una vez superada ofrece una amplia gama de aplicabilidad.

II.6.-DEPENDENCIA CONCEPTUAL

La forma en la que un ser humano aprende a hablar, es mediante asociaciones. Asociamos sonidos a hechos, cosas, acciones, personas, en fin al mundo real, así mismo símbolos escritos.

La idea de la Dependencia Conceptual (DC) se basa en el hecho de que los significados de la palabras en cualquier lenguaje, pueden ser expresadas con primitivas conceptuales combinadas. Esto es, a partir de ciertos conceptos primitivos, es posible incrementar el conocimiento asociándolo con estos conocimientos a conceptos primitivos. Exactamente con el ser humano.

La DC es una teoría de como representar el significado de frases del lenguaje natural de forma que facilite sacar conclusiones y sea independiente del lenguaje de la frase.

Como un ejemplo sencillo de la forma en que se representa el conocimiento en DC, el suceso representado por la frase.

Yo di un libro al hombre

Se representaría como:

```
      p                  o            R -> hombre
Yo ====>ATRANS -------> libro ---|-> yo
```

Donde los símbolos tienen los siguientes significados:

- Las flechas indican dirección de las dependencias
- La doble flecha indica enlace bidireccional entre la acción y el actor
- p indica tiempo pasado
- **ATRANS** es una primitiva de acción permitida por la teoría, indica transparencia de posesión.
- o indica relación causal con objeto.
- **R** indica relación causas con recipiente libro de hombre en yo.

Las acciones primitivas típicas son las siguientes:

ATRANS	Transferencia de una relación abstracta (ej. regalar)
FTRANS	Transferencia de localización física (ej. ir)
IMPELER	Aplicación de fuerza física a un objeto (ej. empujar)
MOVER	Movimiento de una parte del cuerpo por parte de su propietario (ej. patear)
ASIR	Asir un objeto un actuador (ej. lanzar)
INGER	Ingestión de un objeto por parte de un animal (ej. comer)
EXPEL	Expulsión de algo del cuerpo de un animal (ej. llorar)
MTRANS	Transferencia de información mental (ej. contar)
MCONST	Construir nueva información a partir de otra (ej. decidir)
HABLAR	Producir sonidos (ej. decir)
ATENT	Centrar un órgano sensorial sobre un estímulo (ej. oír)

II.7.- REDES NEURONALES

Un modelo que intenta reproducir el funcionamiento del cerebro humano, en específico de las neuronas, mediante computadoras recibe el nombre de redes neuronales artificiales.

Cada neurona tiene una cantidad de sinapsis ("Entradas") y un axon ("Salida"). Entre la entrada y la salida de una neurona, se encuentra el cuerpo de la célula; donde se toman las "decisiones".

La decisión se adopta de una manera muy simple, si la suma de todas las entradas sobrepasa un determinado umbral, si; en caso contrario, no: Algunas entradas pueden ser entradas negativas, las cuales suprimen las entradas positivas de otras procedencias. En cualquier caso, la suma es lo que rige los niveles mas bajos de la mente.

Al igual que las neuronas del cerebro humano, en las redes neuronales se tienen pequeñas unidades inteligentes con capacidad para almacenar y procesar señales, llamadas "nodos", los cuales reciben señales provenientes del exterior de la red o los otros nodos, las procesa, dando distintos pesos a cada una, y genera una única señal de salida que se transmite a otros nodos.

Desde el punto de vista práctico, un nodo debe poseer los medios para almacenar señales y procesarlas según pesos o funciones. Por ello, los nodos deben ser circuitos electrónicos, ordenadores o fragmentos de programas.

En los inicios de la revolución tecnológica, cuando se querían obtener diferentes señales de salida de un sistema dependiendo de los tipos y niveles de estímulo presentes en su entrada, era necesario dotarle de una relación matemática entre dichas entradas y salidas. Con la aparición de las redes neuronales, el sistema se autorregula, deduciendo los pesos para asignar a las distintas señales y la forma de conectar los nodos. Tras esta regulación, puede trabajar en situaciones para las que aún no se hayan establecido reglas de funcionamiento.

Una red neuronal actual establece las conexiones de cada nodo de forma convencional. Su proceso de aprendizaje empieza al darle una pareja de datos de entrada y salida. La red va haciendo pruebas mediante la determinación de pesos más convenientes y de las conexiones de nodos más adecuadas. Al cabo de varios intentos, repetidos con diferentes parejas de datos de entrada y salida conocidos, el sistema esta ya educado, es decir, en condiciones de trabajar; la información que almacena una red se halla dispersa en todos sus nodos lo que confiere características distintas a las de un ordenador convencional y la hace menos propensa a los fallos.

En el campo tecnológico las aplicaciones se refieren, sobre todo, a situaciones donde hay que tratar datos difusos a desarrollar modelos o pronosticar situaciones complejas. Destacan cinco: el PLN manuscrito, el PLN hablado, simulación de centrales de producción de energía, detención de explosivos e identificación de blancos para los radares.

Para la tarea de reconocimiento de textos manuscritos se han desarrollado redes neuronales de casi 200 nodos, que funcionan con tasas de reconocimiento muy elevadas. Los sistemas que incorporan estas redes trabajan en varias etapas. En la primera, se identifican los símbolos más fáciles de distinguir de los demás. Después los caracteres aún sin determinar se combinan con los ya descifrados, buscando las palabras o frases que la red considera más lógicas. Estos sistemas son un buen ejemplo de autoaprendizaje, ya que es fácil enseñarles a distinguir unas letras de otras.

Los dispositivos para el reconocimiento del habla aún no son capaces de dar resultados prácticos. Sin embargo, las técnicas de redes neuronales han conseguido sistemas capaces de reconocer 3000 palabras, con una precisión del 95%. Para ello se han diseñado sistemas con 3 niveles de redes. El primero reconoce de forma global las consonantes, para tratar de identificar las posibles interrogaciones de cada palabra. En el segundo, las redes neuronales aprovechan la entonación para añadir nuevos datos. Finalmente, el tercer nivel realiza una comprobación conceptual del significado de la palabra que, si resulta positiva, considera a dicha palabra reconocida.

En resumen las redes neuronales son una buena alternativa para resolver satisfactoriamente los problemas del procesamiento del lenguaje natural, ya que como están inspiradas en los estudios del cerebro y del sistema nervioso su enfoque es para resolver problemas de reconocimiento.

Tema III

Analizador para el consultador en lenguaje natural

"Quien no conoce bien la fuerza de las palabras no puede conocer bien a los hombres".

CONFUCIO

III.1.- INTRODUCCION

El uso del consultador o interfase en LN en la mayoria de las aplicaciones desarrolladas hasta el momento, esta dirigida a la recuperación de información de bases de datos, y por consiguiente forma parte en la implementación de los sistemas expertos, cuya finalidad es la de procesar datos para satisfacer las necesidades de información y soportar la toma de decisiones de la gran mayoría de organizaciones. Con lo anterior, se logra que la comunicación entre la computadora y el hombre sea más eficiente.

Este capítulo hace énfasis principalmente a las herramientas necesarias para el desarrollo del consultador; se describen los procedimientos y técnicas necesarias para implementar una interfase en LN que sea realmente eficiente y eficaz para las organizaciones. Asimismo, trataremos de cumplir los siguientes propósitos:

1.- Establecer las consideraciones técnicas que hay que tomar en cuenta para el diseño de una interfase en LN.

2.- Presentar la gramática del consultador, a la cual vamos a aplicar todas nuestras consultas.

3.- Mostrar la estructura de la base de datos, así como, los diferentes tipos de datos que serán almacenados dentro de ella.

4.- Presentar las fases a través de las cuales, toda consulta hecha por el usuario, fluirá hasta la obtención de su respuesta.

- Análisis morfológico.
- Análisis léxico.
- Análisis sintáctico.
- Análisis semántico.
- Generador de respuestas.

5.- Mencionar las características propias que distinguen a este consultador.

6.- Exponer las diferentes limitantes que este consultador posee.

7.- Explicar las técnicas a utilizar para el PLN, como son:

- Parser de máquina de estados.
- Parser de libre contexto con descenso recursivo.

8.- Definir los conceptos fundamentales de la teoría de los autómatas.

9.- Describir los principales algoritmos que definen al consultador.

III.2.- CONSIDERACIONES TECNICAS PARA EL DISEÑO DE UNA INTERFASE EN LENGUAJE NATURAL

Tomando en cuenta que, por lo general, las interfases del LN están enfocadas para los usuarios finales, la tecnología para éstas ha tenido considerables progresos en los últimos años y se puede constatar al observar la creciente cantidad de programas comerciales que incluyen una interfase en LN (Ver anexo 1).

En la mayoría de los casos las interfases en LN actúan sobre bases de datos y retoman una serie de consideraciones en su diseño, nosotros describiremos las que han parecido de mayor importancia para nuestro caso. Estas consideraciones son:

- Acceso
- Naturalidad
- Verificación
- Flexibilidad
- Ejecución
- Adaptabilidad

Aunque cada consideración nombrada abarca diferente ámbito, entre ellas existe una estrecha relación como a continuación describimos.

Acceso: para esta consideración técnica, el desarrollo de una interfase en LN, trata de mantener el acceso tradicional a la información de la base de datos, como son:

- Recuperación de información (consultas, reportes)
- Creación de nuevos registros (altas)
- Actualización (modificaciones)
- Eliminación (bajas)
- Procesos *

* Los procesos son una serie de operaciones sobre los datos para obtener un resultado, es la caja negra de todo sistema. Por ejemplo, en una nómina la generación de los recibos no es simplemente un reporte, es la conjunción de un proceso y la impresión de los resultados en un formato especial. Por lo que debido a su naturaleza, los procesos son la parte más compleja en el acceso a la información ya que un proceso puede ser de uno (raramente) o varios (generalmente) pasos para generar el resultado. Cuando son de un solo paso, pueden considerarse como recuperación de información, pero al ser de dos o más, cada paso genera resultados intermedios y el de almacenarlos y conjugarlos con los resultados subsecuentes es una tarea sumamente difícil.

Por lo mencionado anteriormente se ha optado por no abarcar como consideración de acceso a los procesos de más de un paso, considerando únicamente para las consultas los procesos que son de un solo paso.

La generación de reportes representa otro problema, ya que no se considera tomar en cuenta características especiales como tamaño de letra, remarcados, márgenes, encabezados no estándares, etc.. Y aunque restringe al usuario, a la vez da una mayor simplicidad para que los obtenga en un formato llamémosle rústico, pero fácil de obtener. Es digno señalar que los reportes por lo general no son los estándares, ya que se persigue generar reportes especiales para la toma de decisiones.

Por lo que respecta a la creación de nuevos registros y la actualización, se tendría que considerar no dañar y actualizar los archivos de índices que estén asociados a la base de datos. Este tipo de acceso incluyendo también el de eliminación se recomienda llevarlo a cabo mediante un método tradicional como el de menues y pantallas de captura.

Naturalidad : Aquí se considera el tener al consultador con la facilidad de poder procesar consultas en una gran diversidad de formas o estructuras gramaticales. Esto es porque el usuario, puede plantear una consulta con las palabras y la forma que se le viene a la mente o sea de una forma natural.

Es por eso que para no sacrificar la naturalidad del lenguaje y caer en una sintaxis única y rígida que se torne en un lenguaje más por aprender y se aleje de una efectiva interfase en LN, se tienen que considerar las técnicas del PLN que describimos en el capítulo anterior, como por ejemplo una combinación de análisis por comparación de patrones y redes de transición aumentada. La comparación de patrones podría determinar satisfactoriamente consultas simples de pocas palabras por ejemplo :

Consulta	Palabra Clave
Lista Salario	"Lista" "Salario"
Lista salario de	"Lista" "Salario"
las mujeres	"Mujeres"

Pero fracasaría en solicitudes como "De las mujeres solteras lista el nombre y teléfono", donde el análisis tiene que ser más profundo.

La naturalidad es esencial en una interfase en LN para que realmente pueda considerarse como tal, por lo que la selección de la técnica o combinación de técnicas del PLN juegan un papel primordial para la satisfactoria tarea de reconocimiento y obtención de resultados.

Verificación : Consiste en llevar a cabo una interacción con el usuario para determinar si la solicitud de consulta hecha por el usuario esta siendo correctamente interpretada por la interfaz o para romper posibles ambigüedades.

La verificación es recomendable en una interfase en LN porque dado que acepta oraciones con diversas estructuras gramaticales se presenta el gran problema de las ambigüedades (vistas en el capítulo I). Y quizás la forma más fácil de resolverlas es el cuestionar al usuario sobre la interpretación de su consulta, presentándole las diversas opciones que puede significar su solicitud y, que el mismo sea el que en interacción con la interfase en LN rompa la ambigüedad.

Flexibilidad : En el diseño debe contemplarse que los errores en la escritura de las consultas estarán presentes, por lo que el tener un analizador que señale los errores tanto sintácticos como gramaticales y además sea capaz de darnos una serie de alternativas para su corrección, nos sería de gran utilidad. Dichas alternativas pudieran ser :

- Elegir de una lista la palabra correcta
- Agregar la palabra al diccionario con un significado en términos de otras.
- Definir sinónimos
- Etc.

Como en esta parte se debe llevar una interacción con el usuario, esta difiere con la verificación, ya que su fin es el de aclarar una palabra desconocida y el de enriquecer el vocabulario de la interfase en LN, además de determinar errores ortográficos

Ejecución :Consiste en adaptar los algoritmos adecuadamente para que funcionen óptimamente en microcomputadoras con memoria RAM no superior a 640Kb y velocidades de 4.77MHz en promedio. El tiempo de proceso y la cantidad de memoria a utilizar se debe minimizar, para que un usuario pueda llevar a cabo una consulta normal en un tiempo razonable, dado que de nada serviría una interfase en LN que ocupara bastante tiempo en analizar y procesar la consulta para generar resultados en horas o días después. Así, tanto el tiempo de análisis como el tiempo de inicio de resultados deberán reducirse a segundos, esto sin sobrepasar los requerimientos de memoria disponible.

Adaptabilidad: Generar una interfaz adaptable a una variada gama de aplicaciones, es la más problemática consideración en el diseño de la interfase en LN, ya que cada aplicación tendrá un dominio muy distinto, por el simple hecho que los nombres de los archivos de datos y los campos serán distintos, quedando parcialmente inutilizable el diccionario. Se deben generar los nuevos conceptos que definan a la base de datos y todas aquellas palabras que deseamos utilizar antes de poder llevar a cabo nuestra primer consulta, por lo que debe poseer de un mecanismo apropiado para generar los nuevos conceptos de una forma automática (como los campos) y llevar con el usuario un proceso de alimentación o "enseñanza" del nuevo vocabulario.

Cabe señalar que la estructura de la base de datos que envuelve toda la información del diccionario contendrá al menos las palabras básicas de las gramáticas

definidas, como son: comandos, artículos, operadores, conjunciones, preposiciones, etc..

Concluiremos este tema enfatizando un punto muy importante: A pesar de que toda interfase en LN debería de contemplar en su desarrollo todas las características que acabamos de mencionar, nosotros excluimos la gran mayoría de ellas por considerarlas más como herramientas de un producto comercial, que de un trabajo de investigación. Por lo tanto, podemos argumentar que lo más sustantivo de este trabajo fue el PLN.

III.3.- DESCRIPCION GENERAL DEL CONSULTADOR

GRAMATICA DEL CONSULTADOR

El consultador a pesar de ser un intento por acercarse a el lenguaje español, tiene una serie de limitantes que parten de la definición de la gramática que hemos considerado. Se utiliza una simbología para definir las diferentes estructuras gramaticales, que son :

<palabra> Define una estructura gramatical llamada palabra, compuesta por otras estructuras similares o por símbolos terminales.

palabra Es un símbolo terminal el cual esta definido por si mismo como una constante.

{palabra} Es un símbolo terminal que se define por un significado localizado en el diccionario.

La gramática expresada en la forma normal de Bakus (BNF) es la siguiente :

<Consulta> := <Comando>
<Consulta> := <Comando> <Campo bd>
<Consulta> := <Comando> <Campo bd> <Condición>
<Consulta> := <Comando> <Condición>
<Consulta> := <Comando> <Condición> <Campo bd>
<Consulta> := <Condición> <Comando>
<Consulta> := <Condición> <Comando> <Campo bd>
<Consulta> := <Campo bd> <Comando>
<Consulta> := <Campo bd> <Comando> <Condición>

<Comando> := {Comando}
<Comando> := {Comando} <Cuantificador>

<Cuantificador> := {Cuantificador}
<Cuantificador> := Número
<Cuantificador> := Número_letra

<Campo bd> := <Campo>
<Campo bd> := {Artículo} <Campo>
<Campo bd> := <Campo bd> {Conjunción} <Campo bd>
<Campo> := {Campo}

<Condición> := {Op. relacional} Constante
<Condición> := {Op. relacional} <Campo bd> Constante
<Condición> := <Campo bd> {Op. relacional} Constante

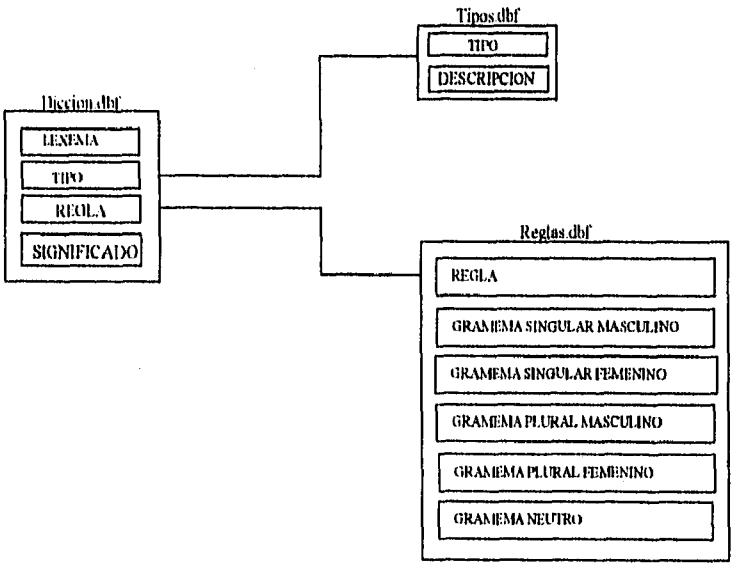
<Condición> := <Sinónimo condicional>
<Condición> := (<Condición>)
<Condición> := <Condición> {Op. lógico} <Condición>

<Sinónimo condicional> := {Artículo} {Sinónimo}
<Sinónimo condicional> := {Sinónimo}

El siguiente esquema muestra la estructura de los archivos de la base de datos que almacenan los tipos de datos, indispensables para el PLN de nuestro consultador. Tales datos son:

- Reglas morfológicas
- Diccionario de palabras
- Reglas gramaticales
- Reglas semánticas
- Diccionario de definiciones

Cabe aclarar que no existe un archivo para cada tipo de dato, sino más bien, existen tres archivos en donde almacenamos toda la información



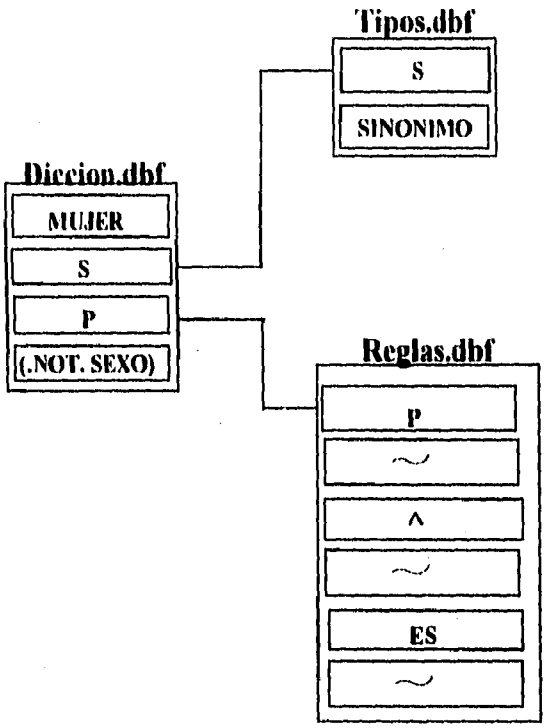
Características del esquema anterior:

- La liga (•----•) indica los campos a través de los cuales se encuentran relacionados los archivos.

- El nombre que va por encima de cada uno de los archivos, es el nombre físico con el cual opera en el sistema. La extensión dbf es la que reconoce el Dbase III Plus.

- Cada uno de los rubros que aparecen en la estructura del archivo, corresponden al nombre del contenido de cada uno de los campos que conforman el registro físico.

El siguiente ejemplo muestra al lexema o radical "**mujer**", junto con toda la información que tiene que ser almacenada en los tres archivos de la base de datos.



S y P representan las llaves que relacionan al archivo de radicales con su correspondiente archivo de tipos y reglas gramaticales.

A continuación daremos la descripción detallada de los registros que conforman los archivos de la base de datos; así como algunos ejemplos de los posibles valores que pueden tomar cada uno de ellos:

ARCHIVO: DICCION

DESCRIPCION DEL REGISTRO

CAMPO	POSICION		TAMAÑO Y TIPO DE CAMPO	SIGNIFICADO
	DEL	AL		
LEXEMA	1	15	15C	RADICAL O FORMA BASICA DE CADA PALABRA.
TIPO	16	16	1C	LLAVE QUE INDICA LA CATEGORIA LEXICA A LA QUE PERTENECE LA PALABRA.
REGLA	17	17	1C	LLAVE QUE IDENTIFICA LOS GRAMEMAS QUE PUEDE TOMAR EL LEXEMA.
SIGNIFICADO	18	92	75C	CONTIENE EL SIGNIFICADO LOGICO Y DEFINICION DE CADA PALABRA.

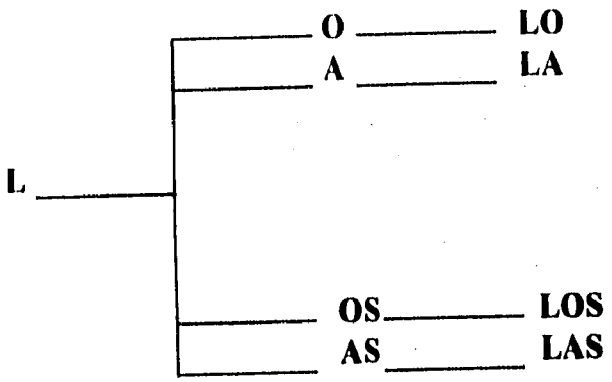
El tipo de valores que pueden tomar los campos de este archivo son:

- Para el campo "lexema" y "significad" es variable.
- El campo "tipo" toma valores de la ("A" a la "Z")
- El campo "regla" toma valores de la ("A" a la "Z", "a" a la "z" y "1" al "9").

Es necesario aclarar que el lexema, es decir, el radical de una palabra en su forma básica, no lo usamos propiamente como marca su definición, sino más bien, el lexema lo usamos de acuerdo a nuestras necesidades. Por lo tanto algunas veces trabajamos con el lexema y otras con un radical definido por nosotros. Ejemplo:

Para las palabras : LO, LA, LOS , LAS

El lexema en comun es : L


L
O
A
OS
AS
LO
LA
LOS
LAS

Estos son algunos registros que conforman el archivo DICCION

LEXEMA	TIPO	REGLA	SIGNIFICADO
AGREGAL	L	b	APPEND BLANK
ALGUN	M	M	NEXT 5
BUSCA	L	a	LIST OFF
CASADA	S	G	(EDO_CIVIL="C".AND..NOT.SEXO)
CASADO	S	D	(EDO_CIVIL="C".AND.SEXO)
COMIEN	R	3	@
DIRECCION	K	P	DIRECCION
DIVORCIADA	S	G	(EDO_CIVIL="D".AND..NOT.SEXO)
DIVORCIADO	S	D	(EDO_CIVIL="D".AND.SEXO)
EDITA	L	e	EDIT
EL	H	N	ARTICULO
EMPIE	R	3	@
HOMBRE	S	D	SEXO
IGUAL	R	F	=
L	H	C	ARTICULO DETERMINADO
LISTA	L	a	LIST
MACHO	S	D	SEXO
MAYOR	R	F	>
MENOR	R	F	<
MI	I	D	PRONOMBRE POSESIVO
MUJER	S	P	(.NOT.SEXO)
NINGUN	Mv	M	NEXT 0
NO	F	O	CONJUNCION PARA NEGAR
NOMBRE	K	D	NOMBRE
O	Fv	O	LETRA 16 DEL ABECEDARIO USADA COMO CONJUNCION
SOLTERA	S	G	(EDO_CIVIL="S".AND..NOT.SEXO)
SOLTERO	S	D	(EDO_CIVIL="S".AND.SEXO)
SUELDO	K	D	SUELDO
TELEFONO		D	TELEFONO
TENG	R	2	^
TERMIN	R	4	%
TOD	M	C	ALL
VIUDA	S	G	(EDO_CIVIL="V".AND..NOT.SEXO)
VIUDO	S	D	(EDO_CIVIL="V".AND.SEXO)
Y	F	O	LETRA 27 DEL ABECEDARIO USADA COMO CONJUNCION
(	U	L	PARENTESIS ABIERTO, SIRVE PARA AGRUPAR CONDICIONES
)	U	L	PARENTESIS CERRADO, SIRVE PARA AGRUPAR CONDICIONES
<	R	O	<
>	R	O	>
=	R	O	=
.	F	O	SIGNO DE PUNTUACION, PARA SEPARAR FRASES O PALABRAS

ARCHIVO: TIPOS

DESCRIPCION DEL REGISTRO				
CAMPO	POSICION		TAMAÑO Y TIPO DE CAMPO	SIGNIFICADO
	DEL	AL		
TIPO	1	1	1C	LLAVE QUE INDICA LA CATEGORIA LEXICA A LA QUE PERTENECE LA PALABRA
DESCRIPCION	21	21	20C	CONTIENE EL NOMBRE DE CADA UNA DE LAS CATEGORIAS LEXICAS.

Estos son los registros que conforman el archivo TIPOS:

TIPO	DESCRIP
A	SUSTANTIVO
B	VERBO REGULAR
C	ADJETIVO
D	ADVERBIO
E	PREPOSICION
F	CONJUNCION
G	VERBO IRREGULAR
H	ARTICULO
I	PRONOMBRE POSESIVO
J	PRONOMBRE DEMOSTRATI
K	CAMPO DE B.D.
L	COMANDO
M	CUANTIFICADOR
O	CONSTANTE
P	NUMERO EN LETRAS
Q	NUMERO EN DIGITO
R	OPERADOR RELACIONAL
S	SINONIMO

Antes de dar la descripción y contenido del archivo REGLAS, definiremos los posibles valores que como ya habíamos comentado, podría tener el campo "regla":

- A..Z Palabras con terminación normal.
- a..z Palabras con terminación especial.
- 1..9 Conjugación de verbos.

También los simbolos ("^" y "~") definen:

- ^ La terminación es tal como se encuentra en su radical.
- ~ No existe terminación para ese gramema.

ARCHIVO: REGLAS

DESCRIPCION DEL REGISTRO

CAMPO	POSICION		TAMAÑO Y TIPO DE CAMPO	SIGNIFICADO
	DEL	AL		
REGLA	1	1	1C	LLAVE QUE IDENTIFICA GRAMEMAS QUE PUEDE TOMAR EL LEXEMA.
GRA_SM	2	4	3C	TERMINACION DEL GRAMEMA EN SINGULAR MASCULINO
GRA_SF	5	7	3C	TERMINACION DEL GRAMEMA EN SINGULAR FEMENINO
GRA_PM	8	10	3C	TERMINACION DEL GRAMEMA EN PLURAL MASCULINO
GRA_PF	11	13	3C	TERMINACION DEL GRAMEMA EN PLURAL FEMENINO
GRA_NEUTRO	14	16	3C	TERMINACION DEL GRAMEMA QUE NO TIENE GENERO NI NUMERO

Estos son los registros que conforman el archivo REGLAS:

REGLA	GRA_SM	GRA_SF	GRA_PM	GRA_PF	GRA_NEUTRO
1	O	ES	E	EN	~
2	O	AS	A	AN	~
3	ZA	ZAS	CEN	ZAN	~
4	E	A	EN	AN	~
A	O	~	OS	~	~
B	~	A	~	AS	~
C	O	A	OS	AS	~
D	^	~	S	~	~
E	^	A	ES	AS	~
F	^	~	ES	~	~
G	~	^	~	S	~
H	Z	~	CES	~	~
I	^	~	^	~	~
J	~	Z	~	CES	~
K	~	~	~	^	~
L	~	^	~	~	~
M	^	A	OS	AS	^
N	^	~	~	~	~
O	~	~	~	~	^
P	^	~	~	ES	~
a	^	ME	NOS	TE	LE
b	^	NOS	TE	LE	ME
c	^	ME	TE	NOS	LE

Por último, falta describir el archivo al cual vamos a llevar a cabo todas nuestras consultas.

DESCRIPCION DEL REGISTRO				
CAMPO	POSICION		TAMAÑO Y TIPO DE CAMPO	SIGNIFICADO
	DEL	AL		
NOMBRE	1	30	30C	NOMBRE DE LA PERSONA
F_NACIM	31	38	8F	FECHA DE NACIMIENTO
SEXO	39	39	1I	SEXO DE LA PERSONA
EDO_CIVIL	40	40	1C	ESTADO CIVIL
TELEFONO	41	52	12C	TELEFONO
DIRECCION	53	112	60C	DOMICILIO
PASATIEMPO	113	142	30C	SUS PASATIEMPOS
RELIGION	143	190	12C	RELIGION DE LA PERSONA
SIGNIFICADO	191	198	8N	INGRESOS DE LA PERSONA

Estos son algunos registros que conforman el archivo AGENDA:

Debido a la dimensión del registro, los campos serán mostrados en tres cuadros.

1)

NOMBRE	FECHA DE NACIMIENTO	SEXO	ESTADO CIVIL	TELEFON
FOUILLOUX GARCIA CLAUDIO RENE	05/08/65	M	C	3-73-61-76
CASILLAS VALADEZ RODOLFO	21/08/65	M	S	
SANTILLAN SANCHEZ FEDERICO	05/06/65	M	C	6-57-76-88
TELLEZ SANTOS MARIA EUGENIA	18/11/64	F	V	3-73-76-75
GONZALEZ CASANOVA MIGUEL ANG	05/04/64	M	S	5-60-11-92
VILLICAÑA MASTACHE CECILIA	27/07/68	F	U	8-56-05-00
HERNANDEZ PEREZ MARIA	01/01/60	F	C	8-22-45-77
GAY PATIÑO PEDRO PABLO	08/12/64	M	C	3-93-45-79

Los siguientes valores denotan las posibles opciones que pueden tomar los campos:
Sexo : .T. para masculino y .F. para femenino
Estado Civil: S para soltero, C casado, D divorciado y U unión libre.
También es importante mencionar que el formato para la fecha es: dd/mm/aa.

2)

DIRECCION	RELIGION
MAR DE LAS ONDAS #61, CD. BRISA NAUCALPAN EDO. DE ME	MORMON
MIGUEL ALEMAN 20, COL. EL POTRERO ATIZAPAN EDO DE ME	CRISTIANA
ARAGON-NEZA DOMICILIO DESCONOCIDO	BUDISTA
CALZADA DE LA VIRGEN 64, TASQUEÑA MEXICO D.F	CATOLICA
CANADA 12, LAS AMERICAS, NAUCALPAN EDO. DE MEX	ATEO
LOS PIRULES 786, CD. BRISA, NAUCALPAN EDO. DE MEXICO	JUDIA
EL HORIZONTE 1590, CD. NEZA, EDO. DE MEXICO	CATOLICA
AV. SAN LORENZO 252, DELEGACION XOCHIMICO, MEXICO D.	PROTESTANT

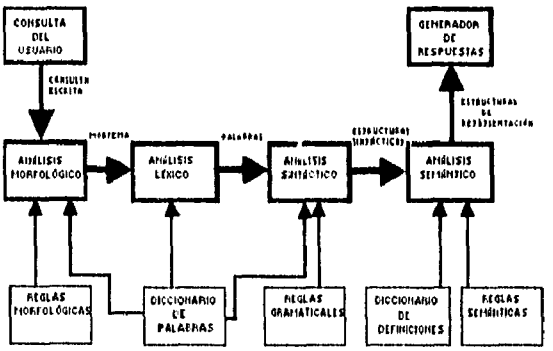
3)

PASATIEMPO	SUELDO
BOLICHE DEPORTES	1650000
LEER NADAR	1500000
ESCRIBIR	950000
CORRER TRABAJAR	750000
MUSICA	500000
TEATRO	800000
TELEVISION RADIO	1500000
CINE	3500000

Concluiremos este tema señalando que el consultador no sólo se aplica al archivo de datos agenda. Sino por lo contrario, la flexibilidad del sistema permite que el consultador pueda ser adaptado a cualquier archivo de datos. Simplemente definiendo los radicales y rasgos característicos en los tres archivos de la base de datos, así como también, hay que definir al archivo de datos a consultar.

III.3.1.- ESQUEMA GENERAL

Basándonos en el esquema general de Terry Winograd [WIN84], el siguiente diagrama muestra el flujo por el cual es procesada toda consulta hecha por el usuario. Desde el registro de su consulta hasta la generación de su respuesta.



Para llevar a cabo la comprensión automatizada del subconjunto del lenguaje español que vamos a manejar, requerimos que la computadora se sirva de varios tipos de datos almacenados como son:

- Reglas morfológicas
- Diccionario de palabras
- Reglas gramaticales
- Diccionario de definiciones
- Reglas semánticas

y que acometa distintos niveles de análisis como son:

- Análisis morfológico
- Análisis léxico
- Análisis sintáctico
- Análisis semántico
- Generador de respuestas

Las fases que omitimos del esquema de Terry Winograd por cuestiones técnicas son:

- **Análisis fonético (reglas fonéticas)**. Esta fase de análisis se utiliza en la comprensión del lenguaje hablado. Por lo tanto como el objetivo es el procesamiento del lenguaje escrito, nos evitamos el difícil análisis de las ondas sonoras.

- **Análisis pragmático (reglas pragmáticas, reglas deductivas y reglas de inferencia)**. Este análisis alimenta a la fase del razonamiento con los posibles significados que pueda tener de cada una de las palabras. Aquí se llevan a cabo deducciones e inferencias a partir del contexto que se tenga del sentido de las palabras. Debido a su alto grado de complejidad, nuestro PLN no deducirá respuestas a preguntas que no se obtengan de manera directa de la base de datos. Tampoco inferirá respuestas a partir de preguntas hechas con anterioridad.

- **Razonamiento**. En ésta fase la computadora extrae las deducciones e inferencias más adecuadas, para preparar las estructuras de representación que responderán a la consulta solicitada. Resulta obvió la omisión de ésta fase por no contar con una etapa que obtenga tanto deducciones como inferencias.

Estas son las fases que contempla nuestro PLN. Así como, una breve explicación de las mismas:

1) **Análisis morfológico**. El programa utiliza unas reglas que descomponen cada palabra en su raíz, hoy llamada lexema, y que representa el significado general; y en su gramema que constituye la variación que cada palabra sufre en su forma. Tales reglas son las que los niños aprenden en la escuela, aprenden, por ejemplo en la palabra "brincando" que el lexema es "brinc" y el gramema es "ando".

   Por otra parte el gramema nos ayuda a identificar el tiempo, modo, persona, género, número, gerundio, participio e infinitivo de las palabras, siendo esto sustancial para los análisis subsecuentes.

Este análisis tiene la ventaja de hacer que el diccionario no sea demasiado grande. Así, el diccionario solamente requiere almacenar las palabras en su más simple expresión; por ejemplo: en el caso de los verbos solamente se necesitan las terminaciones de cada una de las personas. Por lo tanto, en lugar de tener almacenados cinco palabras, sólo almacenamos una que es la raíz y que se relaciona con una regla que involucra su gramema.

2) **Análisis Léxico**. Para cada radical que resulta del análisis morfológico, el programa le asigna un conjunto de características, auxiliándose de un diccionario que ofrece un conjunto de categorías léxicas a las que pertenece dicha raíz. Esto se hace para saber si dicha raíz junto con su gramema corresponde a un artículo, sustantivo, verbo, adjetivo, adverbio, preposición, conjunción, comando, campo de la base de datos, operador relacional, sinónimo, etc. Este análisis nos ayuda también, para conocer el género y número de la palabra; indispensable para la siguiente fase.

   El resultado del análisis morfológico y léxico es, por tanto, una secuencia de las palabras en la oración, llevando consigo cada palabra una cantidad de información sobre sus rasgos e información del diccionario.

3) **Análisis Sintáctico**. El programa aplica las reglas gramaticales para determinar la estructura de la oración. El diseño de un buen analizador (o parser) plantea dos problemas distintos.

   Primero: Especificar un conjunto preciso de reglas; es decir, una gramática que determina el conjunto de estructuras de oración posibles en un idioma en común.

   Segundo: El propio análisis sintáctico, al toparnos con una parte de una frase, no siempre es posible decir con precisión que papel desempeña en la oración, ni si las palabras que la componen deben ir juntas.

4) **Análisis Semántico**. El fruto o salida de la fase del análisis sintáctico se convierte en la entrada de un análisis que constituye en sí, el programa de comprensión del lenguaje. El analizador semántico, se auxilia de un diccionario de definiciones y de unas reglas semánticas para traducir la forma sintáctica de una oración en su forma lógica. Es decir, se pretende colocar expresiones lingüísticas con cierto significado que permita al computador aplicar el procedimiento de generación de respuestas.

5) **Generador de Respuestas**. En esta última fase la estructura generada por el análisis semántico, proporciona los elementos necesarios para que el generador de código en Dbase III Plus, elabore el programa que al ser ejecutado por el interprete Dbase, proporcione de manera sencilla y eficaz la respuesta a la consulta que solicitó el usuario.

Concluiremos este tema con un ejemplo que abarque las distintas fases de nuestro PLN.

CONSULTA:

LISTA EL NOMBRE Y LA DIRECCION DE LAS MUJERES CON SUELDO MAYOR A 1000000

ANALISIS MORFOLOGICO Y LEXICO:

Los primeros análisis (morfológico y léxico) aportan la lista de las palabras con sus radicales, sus categorías léxicas y sus rasgos. "Lista", por ejemplo, es un comando sin género ni número. Estos datos sirven de entrada para el análisis sintáctico de la oración.

PALABRAS	RADICAL	CATEGORIAS LEXICAS	RASGOS
Lista	Lista	Comando	Neutro
el	el	Articulo	Singular y Masculino
nombre	nombre	Campo de la Base de D.	Singular Masculino
Y	Y	Conjunción	Neutro
la	la	Articulo	Singular Femenino
direccion	direccion	Campo dela Base de D.	Singular Masculino
de	*	Preposición	Neutro
las	la	Articulo	Plural Femenino
mujeres	mujer	Sinónimo	Plural Femenino
con	*	Preposición	Neutro
sueldo	sueldo	Campo de la Base de D.	Singular Masculino
mayor	mayor	Operador relacional	Neutro
a	*	Preposición	Neutro
1000000	**	Número en dígito	

* No esta dado de alta en ninguna de las estructuras de la base de datos, sin embargo, son consideradas dentro del subconjunto de las palabras de tipo ruido.

** Poseen un tratamiento diferente.

ANALISIS SINTACTICO

Aplicamos nuestra gramática y vemos que es válida para la siguiente estructura representada en (BNF).

<Consulta> := <Comando> <Campo bd> <Condición>

Donde:
<Comando> = Lista
<Campo bd> = el nombre y la direccion
<Condición> = las mujeres sueldo mayor 1000000

<Comando> := {Comando}

Donde:
<Comando> = Lista

<Campo bd> := {Artículo} <Campo>
<Campo bd> := <Campo bd> {Conjunción} <Campo bd>

Donde:
{Artículo} = el
<Campo> = nombre

{Artículo} = la
<Campo> = dirección

<Campo bd> := <Campo bd> {Conjunción} <Campo bd>

Donde:
<Campo bd> = el nombre
{Conjunción} = y
<Campo bd> = la dirección

<Condición> := <Campo bd> {Op. relacional} Constante

Donde:
<Campo bd> = sueldo
{Op. relacional} = mayor
Constante = 1000000

<Condición> := {Sinónimo condicional}

Donde:
{Sinónimo condicional } = las mujeres

<Condición> :=<Condición> {Op. lógico} <Condición>

Donde:

<Condición	= >las mujeres
{Op. lógico}	= y (por omisión)
<Condición>	= sueldo mayor 1000000

{Sinónimo condicional}	:= {Artículo} {Sinónimo}

Donde:

{Artículo}	= las
{Sinónimo}	= mujeres

Cabe señalar que las palabras ("de", "con" y "a") se eliminan del análisis sintáctico, debido al uso de la técnica (Noise Disposal Parser) que será explicada en el siguiente tema de este capítulo.

ANALISIS SEMANTICO

En esta fase asignamos a las palabras clave de la oración, un conjunto de reglas semánticas y definiciones, para armar las estructuras de representación que serán codificadas en Dbase III Plus.

Lista	List
nombre	nombre
direccion	direccion
mujeres	(.not. sexo)
sueldo	sueldo
mayor	>
1000000	1000000

GENERADOR DE RESPUESTAS

En esta fase se arman las instrucciones tomando el significado lógico de las palabras clave y distinguiendo que forma parte de que en la sintaxis principal:

<Consulta> := <Comando> <Campo bd> <Condición>

Asi, la instrucción en Dbase III Plus que responde a la consulta del ejemplo, queda de la siguiente manera:

LIST NOMBRE,DIRECCION FOR (.NOT. SEXO).AND.SUELDO>100000

III.3.2.- CARACTERISTICAS PRINCIPALES

En este tema explicaremos el porque del uso de un protocolo para la edición de consultas; así como, todas las características que fueron consideradas para el diseño e implementación de este consultador.

La razón principal del uso de un protocolo para llevar a cabo una consulta, es precisamente evitar la ambigüedad de los idiomas; ganando con esto, que la computadora maneje el significado de las palabras de una manera rápida y fácil. También la tarea del análisis semántico se realiza con mayor eficiencia, evitando la incertidumbre del ¿que responder? al momento de entrar a la fase de generación de respuestas.

La siguiente relación establece las principales características que singularizan este proyecto.

- El adjetivo calificativo es considerado dentro de las categorías léxicas como un sinónimo condicional. Ejemplo:

  LISTA LAS MUJERES **SOLTERAS**
  (Sinónimo
  Condicional)

- Los verbos que están contemplados en nuestro diccionario, son considerados dentro de las categorías léxicas como: Comando y Operador relacional. Ejemplo:

  LISTA LOS NOMBRES QUE EMPIECEN CON "B"
  (Comando) (Operador
  Relacional)

- Por consideraciones técnicas los números escritos en letra irán delimitados por diagonales '/' , al principio y final del número. Ejemplo:

  LISTA LOS NOMBRES CON SUELDO MAYOR A /UN **MILLON/**

- Para editar las constantes tipo alfanumérico, éstas tendrán que ir delimitadas por "", al principio y final de la palabra. Ejemplo:

  LISTA LOS TELEFONOS QUE TERMINEN CON "**63**"

- Las constantes numéricas no van entre comillas, unicamente es necesario dar la cifra. Ejemplo:

  BUSCA **5** HOMBRES

En conclusión diremos que este trabajo se ha diseñado al igual que los **"programas limítrofes"**. Por lo tanto, cumple con las siguientes características que definen a un programa como limítrofe:

a) Son elaborados con una finalidad práctica.

b) Las frases en LN que el usuario proporciona a la computadora son interpretadas por ésta como preguntas.

c) El alcance de las preguntas está limitado por la gama de datos a partir de los cuales se emiten las respuestas.

d) El constructor del programa y el usuario se condicionan para beneficio mutuo; esto es, el programador invita al usuario a elaborar preguntas con un cierto protocolo que con el tiempo el usuario no tendrá inconveniente en seguir para tener una eficaz interacción entre el sistema y el usuario.

Es bueno aclarar que nosotros no nos basamos únicamente en una técnica de PLN en especial. Sino más bien, nos avocamos a la tarea de llevar a cabo una combinación de cada una de ellas, utilizando algunas veces sólo su parte más importante. Todo con el fin de realizar un consultador que satisficiera de manera general la aplicación para la cual fue construida cada una de las técnicas de PLN.

Acabaremos, dando una explicación de una técnica de PLN, que consideramos es importante hablar de ella dentro de este tema. La técnica de PLN **Noise Disposal Parser (NDP)** que en español se podría traducir como "analizador eliminador de ruido"; es aquella que gracias a su uso, todas las aplicaciones que sólo se interesan en la información que la oración contiene, analizan a un lenguaje tomando únicamente algunas palabras que forman un subconjunto del LN y que son contempladas en su gramática. El análisis al no contener todas las palabras que constituyen un lenguaje, las considera a todas ellas como ruido, es decir, las elimina de la oración. Normalmente todas las oraciones deben seguir un formato rígido que se asemeje al LN. Es por ello que este tipo de analizador, comunmente es utilizado en aplicaciones como: un procesador de comandos. Por ejemplo, al considerar una base de datos que contiene nombres de compañías y sus respectivos costos de almacenamiento. Asume que la base de datos aceptara consultas como:

Lista todas las compañías con costos > 100.
Lista todo.
Listame XYZ.
Listate uno con costo de almacenamiento < 100.

Como pueden ver, estos tipos de consultas cumplen con la siguiente sintaxis:

comando <modificador> <campo_bd> <operador> <constante>

Nosotros utilizamos esta técnica ya que en cierta forma el consultador que elaboramos es una aplicación que se basa en un procesador de comandos. Sin embargo, como ya dijimos, no sólo nos basamos en esta técnica. Además, no la usamos totalmente, ya que no eliminamos todas las palabras que no se encuentran en el diccionario como el NDP lo establece. Sino más bien, sólo ignoramos algunas de ellas, principalmente preposiciones y algunos pronombres.

Cabe aclarar que esta técnica tiene como principal desventaja el eliminar palabras que en alguna forma pueden ser importantes. Como ustedes saben, en una conversación normal, la mayoría de las palabras son importantes en alguna u otra forma. Otra desventaja es que en muchas situaciones, el consultador puede aceptar oraciones extrañas, tales como:

Lista los hombres **que en que** son casados

Lista mujeres casadas

No obstante, el consultador da la respuesta correcta.

La principal ventaja del NDP es que es sumamente fácil de implementar, y la obtención de la información de una oración es muy rápida. Además, cuando la usamos con otra técnica, ésta puede reducir el número de variaciones que deben ser aceptadas. De hecho, no hay duda que no se haya desarrollado algún PLN con éxito que no utilice al menos en parte esta técnica.

III.3.3.- LIMITACIONES

Este trabajo lo hemos tenido que limitar por diversos factores, los cuales hacen que, la mayoría de programas para el PLN, hoy en día y en un futuro remoto no tengan una comunicación plena entre el hombre y la máquina en LN.

De estos factores, algunas son restricciones que aún nadie ha podido resolver, otros no son muy difíciles, pero si laboriosos, por lo tanto no consideramos prudente su solución para los fines que perseguimos.

A continuación daremos las distintas carencias que limitan a nuestro proyecto y una breve explicación de cada una de ellas.

A) De los tres modos verbales (Indicativo, Subjuntivo e Imperativo), esto es, las maneras de expresar la acción o el hecho contenido en el infinitivo. Se consideran consultas válidas sólo las oraciones que utilicen el modo subjuntivo e imperativo.

- **Modo Subjuntivo**
  Expresa la acción contenida en el verbo a manera de deseo, posibilidad, esperanza, y esta acción está siempre subordinada a otro verbo que expresa alguna idea de negación, duda, deseo, necesidad, probabilidad. En resumen expresa la acción no como real, sino como pensada por el que habla.
  Ejemplo:
  Lista los nombres que empiecen con "A"

- **Modo Imperativo**
  La acción contenida en el verbo se expresa en una forma imperativa, esto es, que significa una orden, un mandato, que desde luego puede ir hasta la súplica o el ruego. Así se expresa la voluntad del que habla para que el que escucha ejecute alguna acción.
  Ejemplo:
  Busca los teléfonos y direcciones de las mujeres

El modo Indicativo no fue contemplado debido a que expresa la acción en forma objetiva y real sin que el que habla tenga otra actitud que la de indicar el hecho
Ejemplo:
Ricardo escribe un libro
Pedro habló por teléfono
Daniel jugará mañana

B) De los dos tiempos (Simple y Compuesto), sólo se considera al tiempo **simple**, aquel que no se forma con la ayuda de un verbo auxiliar: he tenido, había corrido, habré partido; para llevar a cabo una consulta válida.

Ejemplo:

Laura tiene dos teléfonos
Juan corrió toda la mañana
El presidente partirá a Japón en la noche

C) De los tiempos simples sólo ocuparemos al tiempo **presente**, esto es, aquel que indica una acción no terminada en el tiempo en que de ella se habla. Naturalmente tal acción pudo haber empezado mucho antes y podrá terminar mucho después de que hayamos acabado de hablar, ya que el presente no debe considerarse como un instante fugaz, sino como un período de mayor o menor duración en el cual está comprendido en el momento en que hablamos

Ejemplo:

tengo
corro
parto

D) De las tres personas gramaticales (yo, tú y él) y sus plurales (nosotros, ustedes y ellos) utilizamos en la conjugación de los verbos, sólo aquellos que hagan referencia a otra u otras personas. Por lo tanto en el modo subjuntivo usamos la tercera persona de singular y plural, y en el modo imperativo la segunda persona de singular.

Ejemplo:

Lista la religión de las mujeres (modo imperativo)
(tiempo presente)
(segunda persona
de singular)

Lista las mujeres que tengan el nombre de "MARY" (modo subjuntivo)
(tiempo presente)
(tercera persona
de plural)

E) No se aceptan los verbos en la forma infinitiva, gerundio y participio por carecer de tiempo, persona y número. Puesto que estos elementos son indispensables para el análisis de nuestro parser.

F) De todos los tipos de pronombres que hay (personales, posesivos, demostrativos, relativos, indefinidos, interrogativos y numerales) utilizaremos dentro de las consultas sólo aquellos que se reportan como válidos en el siguiente cuadro.

Persona.- Indica	primera **(1a)** persona
	segunda **(2a)** persona
	tercera **(3a)** persona

Número .- Indica singular **(S)**
plural **(P)**
neutro **(N)**

Pronombre	Válido	Persona	Número	Ejemplo
Personales	si	3a	S.P.N	se,la,lo,las,los
Posesivos	no			mio,suyo,tuyo
Demostrativos	si	3a	S.P.N	aquél,aquéllos
Relativos	si	3a	S.P.N	que,cuyo,quien
Indefinidos	no			alguien,muchos
Interrogativos	no			qué,quién,cuál
Numerales	no			uno,primero,mitad

G) De todos los tipos de adjetivos que hay (calificativos, posesivos, demostrativos, indefinidos, cuantitativos y distributivos) utilizaremos dentro de las consultas sólo aquellos que se reportan como válidos en el siguiente cuadro.

Adjetivo	Válidos	No Válidos
Calificativos	mujeres,solteras	
Posesivos	su,sus	mi,tu
Demostrativos		este,ese,aquel
Indefinidos	algún,algunos	cualquier,ningún
Cuantitativos	(No.cardinal),todos	pocos,muchos,mas
Distributivos		sendos,sendo

Cabe señalar que no sólo el pronombre se diferencia del adjetivo por su acentuación, sino también porque el pronombre sustituye al sustantivo o lo determina. En cambio, el adjetivo no sustituye al sustantivo, sino solamente le designa una cualidad o determina o limita su extensión.

H) No se analizan los acentos, por lo tanto carecemos de llevar a cabo un buen análisis gramatical por el énfasis que el acento pueda aplicar a alguna palabra de acuerdo a su contexto y forma de expresarlo.

Ejemplo:

este, ese, aquel (adjetivos)
éste, ése, aquél (pronombres)

I) De los cuatro tipos de enunciados que hay (declarativo, exclamativo, interrogativo e imperativo) sólo son válidos los enunciados imperativos.
Ejemplo:

Declarativo	El cielo es azul
Exclamativo	¡Qué hermoso día!
Interrogativo	¿Qué día es hoy?
Imperativo	Lista los nombres de los hombres cuyo sueldo sea mayor a 900000

J) No se manejan consultas compuestas. A pesar de que al efectuar una consulta podemos estar repitiendo campos de la base de datos, sinónimos y condiciones con los conectores ("," , "y" , "o" y "espacios en blanco"), no podemos repetir nuestra estructura gramatical principal más de una vez. Por lo tanto a partir de esta gramática:

Comando ===>> Campo_bd ===>> Condición

No podemos dar como consulta buena:

Lista los nombres de las mujeres y búscame los hombres con sueldo mayor a 500000

Como podemos observar "lista" y "búscame" son comandos. Y de acuerdo a nuestro parser no es posible repetir más de un comando en una misma consulta.

K) Dentro del análisis de las condiciones no llevamos una validación de éstas por cuestiones técnicas. No obstante hablaremos un poco sobre ellas.
Las condiciones pueden ser simples o compuestas:

- Simples - una condición
- Compuestas - dos o más condiciones unidas por medio de operadores de relación o conectores

Para las condiciones compuestas existe una serie de circunstancias que las hace subdividir en:

- Factibles
- Contradictorias

Terminaremos diciendo que las condiciones factibles pueden darse de dos maneras:

- Perfectas
- Redundantes

Estos son unos ejemplos de los tipos de condiciones:

Simple:

edad < 20
edad de 20 años
salario mayor a 500000

Compuesta, Factible y Perfecta:

edad < 20 y salario mayor a 500000
nombre sea "Juan" o "Pedro"

Compuesta, Factible y Redundante:	A	Simple:
edad < 20 y edad < 10	==>>	edad < 10
edad < 20 o edad < 10	==>>	edad < 20

Aqui podemos observar que es posible reducir condiciones redundantes en compuestas perfectas o simples.

Contradictorias:

edad = 10 y edad < 9
edad < 10 y edad > 20

L) Las cifras numéricas expresadas en letras no incluyen las palabras que representan quebrados ni decimales. También las expresadas en dígitos no incluyen ni quebrados ni decimales. Sin embargo en la expresión en letra los decimales son contemplados hasta centésimas utilizando las palabras que expresen números hasta 99.
Ejemplo:
No Válidos:

un medio
dos cuartos
tres décimas

Válidos:

dos punto noventa y nueve

M) No se puede expresar un número como una combinación de cifras y letras
Ejemplo:

treinta y 2
ciento punto 12
8 millones

Concluiremos diciendo que en estos momentos tanto la ambigüedad de los idiomas como la falta de formalidad en el significado de las palabras, constituyen las principales limitantes, que en muchos años no podremos deshacernos de ellas.

III.4.- REDES DE TRANSICION

Empezaremos este tema hablando un poco de la teoría de **Autómatas**, para luego dar algunos ejemplos. También daremos algunos ejemplos que utilizan la técnica de "Máquina de Estados" a través de una función de mapeo y la de "Libre Contexto con Descenso Recursivo" que analizan la oración a través de reglas de producción de manera recursiva.

El corazón de cualquier sistema PLN es el parser. El parser es la sección de código que lee cada oración, palabra por palabra, para decidir que es que, y que acción tomar a la hora de analizar el significado. A estas palabras también se les conoce como **tokens**. Así un token se define como una secuencia de caracteres que no contienen espacios en blanco.

Una **Expresión Regular (ER)** es una regla algebraica, en síntesis, que me representa un lenguaje. El mecanismo que reconoce las ER es el autómata.

Un **autómata finito no determinístico (AFND)**= M es definido como una quintupla siguiente (S,Σ,δ,So,F):

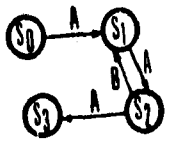
- S es un conjunto finito de estados de control.

Σ es el conjunto del alfabeto.

δ es una función de transición de estados, la cual mapea S X (Σ U {ε}) a un subconjunto de S.

- So ε S es el estado inicial del control finito
- F ≤ S es el conjunto de estados finales (estados aceptores).

Ejemplo:



Donde:

$S=\{S_0,S_1,S_2,S_3\}$

$\Sigma=\{a,b\}$

$\delta(Si,\alpha)=Sj$ es la función de transición de un estado a otro.

S_0 es el estado inicial y S_3 es el estado aceptor.

La siguiente tabla muestra las posibles transiciones que pueden tomar los estados al efectuar la función de mapeo.

	SIMBOLOS DEL ALFABETO		
ESTADOS	a	b	ε
S_0	$\{S_1\}$	ø	ø
S_1	$\{S_2\}$	ø	ø
S_2	$\{S_3\}$	$\{S_1\}$	ø
S_3	ø	ø	ø

ø denota que no existe transición posible.

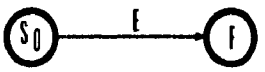
La construcción de un autómata finito no determinístico se efectúa a través de los primitivos obtenidos de la definición de ER:

a) Sea

Es un autómata finito para la transición del conjunto vació.

b) Si ε es la cadena vacía y a ε Σ entonces:

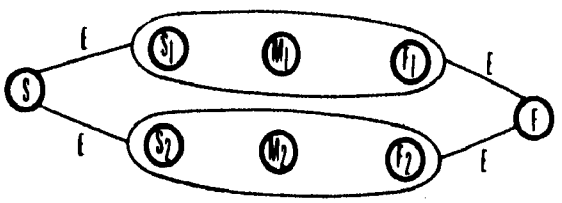
b.1)



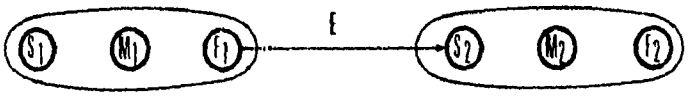
b.2)

En el caso b.1) el arco definido por ε es llamada una transición espontánea

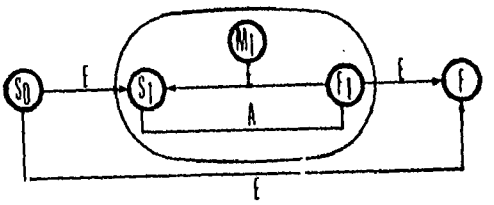
c) Si α y ß son ER y son reconocidas por el autómata M_1 y M_2 respectivamente entonces la U (Unión) se define como:



d) Si α y β son ER y M_1 y M_2 son Autómatas para α y β entonces la concatenación se define como:



e)Si $\alpha \in$ ER entonces α^* $\in$ ER.



Con estos puntos, tenemos los elementos necesarios para la construccion del AFND.

Dada una gráfica denominada AFND la cual está constituida por nodos y arcos etiquetados por símbolos de un alfabeto. La transformación a una "minimización", o bien, transformación a un **autómata finito determinístico (AFD)**. Equivale a encontrar las clases de equivalencia de estados que son apuntados por arcos de un mismo simbolo.

Algoritmo de Cerradura Vacía.

La ε-cerradura (S) es el conjunto de estados que se construyen a partir de las siguientes reglas:

a) Se añade S a la ε-cerradura (S)

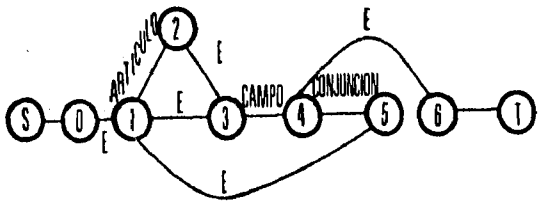
b) Si t pertenece a la ε-cerradura (S) y existe una transición ε de t a u, entonces se añade u a la ε-cerradura (S).

ε-cerradura (s) = {S,t,u}

El siguiente ejemplo muestra la forma en que construimos un AFD minimizado a partir de un AFND que representa la gramática de un campo de la base de datos..

Ejemplo:

AUTOMATA FINITO NO DETERMINISTICO



Donde:

S={0,1,2,3,4,5,6}

Σ={articulo,campo,conjunción}

La siguiente tabla muestra las posibles transiciones que pueden tomar los estados al efectuar la función de mapeo.

	SIMBOLOS DEL ALFABETO			
ESTADOS	artículo	campo	conjunción	ε
0	ø	ø	ø	1
1	2	ø	ø	3
2	ø	ø	ø	3
3	ø	4	ø	ø
4	ø	ø	5	6
5	ø	ø	ø	1
6	ø	ø	ø	ø

Aplicando el Algoritmo de minimización tenemos:

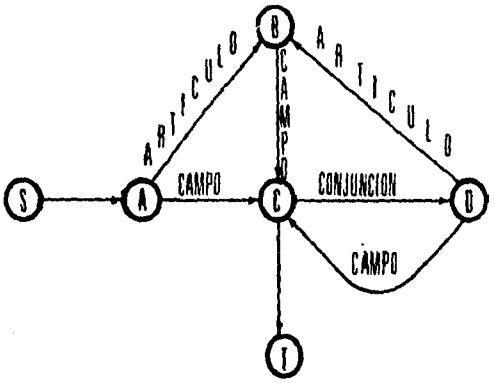
ε-cerradura {0}={0,1,3}=A

Desde A con artículo	={2}	ε-cerradura {2} = {2,3}	**B**
Desde A con campo	={4}	-cerradura {4} = {4,6}	**C**
Desde A con conjunción	= ø		
Desde B con artículo	= ø		
Desde B con campo	= {4}	-cerradura {4} = {4,6}	**C**
Desde B con conjunción	= ø		
Desde C con artículo	= ø		

Desde C con campo	= ø		
Desde C con conjunción	= {5}	-cerradura {5} = {1,3,5}	**D**
Desde D con artículo	= {2}	-cerradura {2} = {2,3}	**B**
Desde D con campo	= {4}	-cerradura {4} = {4,6}	**C**
Desde D con conjunción	= ø		

Este procedimiento nos genera :

AUTOMATA FINITO DETERMINISTICO

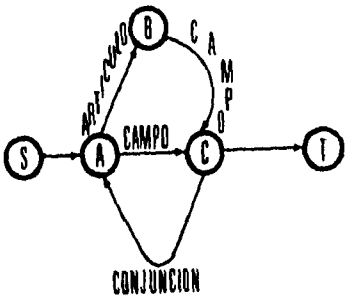


Así, las posibles transiciones que pueden tomar los estados al efectuar la función de mapeo en el AFD son:

	SIMBOLOS DEL ALFABETO		
ESTADOS	artículo	campo	conjunción
A	B	C	ø
B	ø	C	ø
C	ø	ø	A
D	B	C	ø
E	ø	ø	ø

Podemos observar que en la tabla se encuentran dos estados A y D con transiciones equivalentes. Por lo tanto, nos disponemos a eliminar el estado D para obtener el AFD minimizado.

AUTOMATA FINITO DETERMINISTICO (MINIMIZADO)



Este autómata es el que utilizamos para llevar a cabo el análisis de la gramática de un campo de la base de datos.

El parser de máquina de estados utiliza el estado actual en que se encuentra en cada token de la oración, para analizar que tipo de palabra puede legalmente seguir. Por lo tanto una máquina de estados es una gráfica dirigida que muestra las transiciones válidas de un estado a otro. La principal desventaja del uso de esta técnica es su complejidad. No obstante que en la gramática del campo de la base de datos, el número de estados es relativamente pequeño, podemos decir que si quisiéramos aplicar esta técnica a toda la gramática del Español, sería una locura adaptarla por la gran cantidad de estados que manejaríamos. Por esta razón, el parser de máquina de estados sólo se usará en situaciones que puedan utilizar un subconjunto estricto de una gramática en particular.

El siguiente ejemplo utiliza, al igual que la gramática de un campo de la base de datos, la técnica de máquina de estados para analizar sintáctica y semánticamente el programa de conversión de números cardinales a su correspondiente valor en cifra. Cabe señalar que el AFND no fue minimizado a un AFD, ya que el número de estados y elementos que conforman el alfabeto era demasiado grande. Por lo tanto utilizamos únicamente el AFND.

Asimismo, por complejidad en el diseño del diagrama, sólo daremos la tabla que representa todas las transiciones que pueden tomar los estados al efectuar su correspondiente función de mapeo.

Utilizando la siguiente simbología

A= Treinta, Cuarenta, Cincuenta, Sesenta, Setenta, Ochenta y Noventa.

B= Doscientos, Trescientos, Cuatrocientos, Quinientos, Seiscientos, Setecientos, Ochocientos y Novecientos.

C= Uno, Dos, Tres, Cuatro, Cinco, Seis, Siete, Ocho y Nueve.

D= Diez, Once, Doce, Trece, Catorce, Quince, Dieciséis, Diecisiete, Dieciocho, Diecinueve, Veinte, Veintiuno, Veintidós, Veintitrés, Veinticuatro, Veinticinco, Veintiséis, Veintisiete, Veintiocho y Veintinueve.

E= Mil.

F= Millón.

G= Cero.

H Millones.

I= Punto.

J= Cien.

K= Un.

L= Ciento.

M= Y.

Tenemos la siguiente tabla:

	SIMBOLOS DEL ALFABETO													
ESTADOS	A	B	C	D	E	F	G	H	I	J	K	L	M	E
0	1	9	2	2	11	ø	4	ø	5	2	13	10	ø	ø
1	ø	ø	ø	ø	11	ø	ø	19	5	ø	ø	ø	12	31
2	ø	ø	ø	ø	11	ø	ø	19	5	ø	ø	ø	ø	31
3	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	31
4	ø	ø	ø	ø	ø	ø	ø	ø	5	ø	ø	ø	ø	31
5	6	ø	3	3	ø	ø	8	ø	ø	ø	ø	ø	ø	ø
6	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	7	31
7	ø	ø	3	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø
8	ø	ø	3	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	31
9	1	ø	2	2	11	ø	ø	19	5	ø	ø	ø	ø	31
10	1	ø	2	2	ø	ø	ø	ø	5	ø	ø	ø	ø	ø
11	14	16	15	15	ø	ø	ø	19	5	15	ø	17	ø	31
12	ø	ø	2	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø
13	ø	ø	ø	ø	ø	ø	ø	19	ø	ø	ø	ø	ø	ø
14	ø	ø	ø	ø	ø	ø	ø	19	5	ø	ø	ø	18	31
15	ø	ø	ø	ø	ø	ø	ø	19	5	ø	ø	ø	ø	31
16	14	ø	15	15	ø	ø	ø	19	5	ø	ø	ø	ø	31
17	14	ø	15	15	ø	ø	ø	ø	5	ø	ø	ø	ø	ø
18	ø	ø	15	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø
19	20	22	21	21	25	ø	ø	ø	5	21	ø	23	ø	31
20	ø	ø	ø	ø	25	ø	ø	ø	5	ø	ø	ø	24	31
21	ø	ø	ø	ø	25	ø	ø	ø	5	ø	ø	ø	ø	31
22	20	ø	21	21	25	ø	ø	ø	5	ø	ø	ø	ø	31
23	20	ø	21	21	ø	ø	ø	ø	5	ø	ø	ø	ø	31
24	ø	ø	21	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø
25	26	28	27	27	ø	ø	ø	ø	5	27	ø	29	ø	31
26	ø	ø	ø	ø	ø	ø	ø	ø	5	ø	ø	ø	30	31
27	ø	ø	ø	ø	ø	ø	ø	ø	5	ø	ø	ø	ø	31
28	26	ø	27	27	ø	ø	ø	ø	5	ø	ø	ø	ø	31
29	26	ø	27	27	ø	ø	ø	ø	5	ø	ø	ø	ø	ø
30	ø	ø	27	27	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø
31	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø

Como ya habiamos dicho ø representa una transición vacía

Asimismo, el estado 31 representa el estado terminal. Por lo tanto en los estados donde no haya transición para ε no puede haber un token terminal.

De tal forma para la expresión:

"DOS MIL MILLONES SETECIENTOS CUARENTA Y CINCO PUNTO DIEZ"

Tenemos que:

Estado Inicial	Evaluación del Token	Clave del Token	Estado Final
0	DOS	C	2
2	MIL	E	11
11	MILLONES	H	19
19	SETECIENTOS	B	22
22	CUARENTA	A	20
20	Y	M	24
24	CINCO	C	21
21	PUNTO	I	5
5	DIEZ	D	31

Representa el flujo de transiciones que satisfacen sintáctica y semánticamente la expresión dada.

Es necesario aclarar que no todas las gramáticas del consultador son analizadas a través de la técnica de máquina de estados. La otra técnica a utilizar es la llamada "Parser de Libre Contexto con Descenso Recursivo". Para comprenderla se debe mirar la construcción de la oración de manera diferente de como la vimos en el parser de máquina de estados. Para el parser de libre contexto, hay que pensar que la oración se compone de varios elementos, los cuales a su vez están compuestos de otros elementos, y así sucesivamente hasta llegar a sus elementos atómicos (adjetivos, artículos, conjunciones, etc.). Las reglas que definen cuantos elementos conforman a cada uno de los elementos padre son llamadas reglas de producción de la gramática. Así, un parser de libre contexto utiliza estas reglas de producción para analizar la oración.

La siguiente gramática, que en nuestro caso es la de una consulta, es analizada a través de esta técnica.

Donde las reglas de producción son las siguientes:

Consulta --> Comando
Consulta --> Comando + Campo bd
Consulta --> Comando + Campo bd + Condición
Consulta --> Comando + Condición
Consulta --> Comando + Condición + Campo bd

Consulta --> Condición + Comando
Consulta --> Condición + Comando + Campo bd

Consulta --> Campo bd + Comando
Consulta --> Campo bd + Comando + Condición

Es importante mencionar que los elementos Comando, Campo Bd y Condición no son elementos atómicos. Así, este subconjunto de nuestra gramática es utilizada únicamente para ejemplificar el uso de esta técnica.

A partir de la siguiente consulta:

DE LAS MUJERES LISTA **LOS NOMBRES**
Condición Comando Campo bd

El análisis comienza al examinar recursivamente cada regla de producción que corresponde a una consulta. Como podemos ver la regla de producción remarcada corresponde precisamente a la estructura de la consulta. Por lo tanto, el parser regresa una bandera que indica que la consulta fue analizada semánticamente bien en la séptima regla de producción.

Cuando usamos un subconjunto de la gramática del español es fácil definir un conjunto de reglas que describan esa gramática. Sin embargo, si quisiéramos aplicar esta técnica en toda la gramática del español, las reglas serían muy complejas, lo cual podría conducir a una explosión combinatoria de reglas que harían imposible el uso de esta técnica.

Concluiremos este tema diciendo que el parser que analiza nuestra gramática se define a partir de una combinación de estas dos técnicas, haciendo la aclaración que a veces utilizamos el AFND y otras el AFD minimizado según sea la dificultad de la gramática. Además se auxilia de la técnica NDP vista en el tema anterior.

III.5.- ALGORITMOS Y PSEUDO-CODIGO

En el contenido de este tema presentaremos los principales algoritmos que hacen posible alcanzar el objetivo de este proyecto. Es decir, estableceremos un conjunto de acciones que deben ejecutarse en un orden específico. Esto a través de un pseudo-código que explique en un lenguaje convencional cada uno de los pasos a realizar.

La siguiente lista presenta los principales algoritmos a tratar:

A) Tratamiento de cantidades numéricas.

B) Separación y contabilidad de token's, asignándoles a cada uno de ellos sus rasgos lexicográficos.

C) Subautómata que reconoce la gramática de un comando:
<Comando> : = {Comando} [<Cuantificador>]

D) Subautómata que reconoce la gramática de un campo de base de datos:
<Campo bd> : = <Campo>
<Campo bd> : = {Artículo}<Campo>
<Campo bd> : = <Campo bd> {Conjunción}<Campo bd>

E) Subautómata que reconoce la gramática de una condición:
<Condición> : = {Op. relacional} Constante
<Condición> : = {Op. relacional}<Campo bd> Constante
<Condición> : = <Campo bd> {Op. relacional} Constante
<Condición> : = <Sinónimo condicional>
<Condición> : = (<Condición>)
<Condición> : = <Condición> {Op. lógico}<Condición>

F) Subautómata que reconoce la gramática de la consulta del tipo:
consulta = <comando> [<campo_od> [<condición>]]
consulta = <comando> [<condición> [<campo_bd>]].

G) Subautómata que reconoce la gramática de la consulta del tipo:
consulta = <condición><comando> [<campo_bd>].

H) Subautómata que reconoce la gramática de la consulta del tipo:
consulta =<campo_bd><comando> [<condición>].

A)

NOMBRE : coletnum(letra,numero)

FUNCION :Recibe la expresión numérica en la variable "letra". Posteriormente, se realiza un análisis sintáctico y semántico antes de evaluar la cantidad en letra. Si la expresión cumplió la sintaxis y semántica, se lleva a cabo la conversión de letra a número, regresando la cantidad en la variable de paso "numero".

EJEMPLO :Si la variable "letra" es igual a "DOS MIL MILLONES", entonces coletnum(letra,numero) nos da: verifica que la sintaxis y semántica sea correcta y retorna la cantidad numérica 2000000000 en la variable "numero".

PROCEDIMIENTOS UTILIZADOS :

cambia(token,num) = Al enviar el token, por ejemplo "DIEZ", se retorna en la variable de paso "num" la cifra 10.

MACROS UTILIZADAS :

Toknum&n = Es un arreglo dinámico que genera variables desde toknum1 hasta toknumn dependiendo del número de token's que tenga la variable letra.

VARIABLES UTILIZADAS:

letra	=Cadena que contiene la expresión numérica en letra y es de paso.
numero	=Valor entero que retorna la cantidad numérica que se convirtió.
cadnum	=Cadena con la checamos la sintaxis y semántica de la variable letra.
fl	=Bandera que indica si la semántica paso correctamente.
ga	=Cadena que contiene los token's (TREINTA, CUARENTA, CINCUENTA.NOVENTA).
gb	=Cadena que contiene los token's (DOSCIENTOS, TRES CIENTOS, CUATROCIENTOS ... NOVECIENTOS).
gc	=Cadena que contiene los token's (UNO, DOS, TRES ... NUEVE).
gd	=Cadena que contiene los token's (DIEZ, ONCE, DOCE ...VEINTINUEVE).
ge	=Cadena que contiene al token (MIL).
gf	=Cadena que contiene al token (MILLON).
gg	=Cadena que contiene al token (CERO).
gh	=Cadena que contiene al token (MILLONES).
gi	=Cadena que contiene al token (PUNTO).
gj	=Cadena que contiene al token (CIEN).
gk	=Cadena que contiene al token (UN).

gl	=Cadena que contiene al token (CIENTO).
gm	=Cadena que contiene al token (Y).
gz	=Cadena que contiene al token (;).
lg	=Indica la longitud de la cadena "cadnum".
n_t	=Valor que contiene el número de token's en lacadena "cadnum".
n	=Carácter utilizado en la macro toknum&n para identificar el número de token.
ncar	=Apuntador al token en la cadena "cadnum".
estado	=Valor entero que indica el estado en que se encuentra dentro del parser.
car	=Carácter que identifica en donde se encuentra el token dentro del subconjunto de cadenas de token's.
al	=Posición donde un espacio en blanco es encontrado dentro de la cadena "cadnum" .
le	=Longitud del token
estados	=Cadena que indica los posibles estados que puede transitar de un estado a otro.
simbolo	=Cadena que contiene todos los posibles conjuntos de token's donde puede ubicarse un token,
serror	=Cadena que guarda el significado del error en caso de haberlo.
aerror	=Cadena que guarda el token donde se encontro un error.
error_	=Valor lógico que indica si hubo o no error.
word_	=Cadena temporal que guarda al token "MILLON"O"MILLONES".
llag	=Valor lógico de control.
ban	=Valor lógico de control.
ban1	=Valor lógico de control.
ban3	=Valor lógico de control.
paso	=Valor lógico de control.
cantida	=Cantidad entera donde se va almacenando la conversión de token's.
cantcom	=Cantidad entera que auxilia a la variable "cantida".
token	=Cadena a la cual le asignamos cada uno de los token's que contiene la variable "letra".
ajuste	=Contador que indica el número de posiciones a la derecha del punto decimal.
j	=Indice utilizado en un ciclo.
dig	=Número de dígitos que contiene la fracción.
fraccio	=Potencia de 10 con la cual dividimos a la cantidad que va a ser representada como fracción.
aux	=Cantidad que auxilia a la variable "cantida".
entera	=Cantidad que resulta de sumar las variables "cantida" y "cantcom".
product	=Cantidad que multiplica a la variable "cantida" para la obtención de miles o millones.
num	=Cantidad que retorna la cifra en número del token que es enviado al procedimiento CAMBIA.

PSEUDO-CODIGO

```
INICIO<Convertidor letra a número>
    cadnum      <-- letra + ';'
    fl          <-- 0
    lg          <-- longitud (cadnum)
    n           <-- '|'
    ga          <-- '*TREINTA*CUARENTA*CINCUENTA*SESENTA
                    *SETENTA *OCHENTA*NOVENTA*'
    gb          <-- '*DOSCIENTOS*TRESCIENTOS*CUATROCIENTOS
                    *QUINIENTOS*SEISCIENTOS*SETECIENTOS
                    *OCHOCIENTOS*NOVECIENTOS*'
    gc          <-- '*UNO*DOS*TRES*CUATRO*CINCO*SEIS*SIETE
                    *OCHO*NUEVE*'
    gd          <-- '*DIEZ*ONCE*DOCE*TRECE*CATORCE*QUINCE
                    *DIECISEIS*DIECISIETE*DIECIOCHO*DIECINUEVE
                    *VEINTE*VEINTIUNO*VEINTIDOS*VEINTITRES
                    *VEINTICUATRO*VEINTICINCO*VEINTISEIS
                    *VEINTISIETE*VEINTIOCHO*VEINTINUEVE*'
    ge          <-- '*MIL*'
    gf          <-- '*MILLON*'
    gc          <-- '*CERO*'
    gh          <-- '*MILLONES*'
    gi          <-- '*PUNTO*'
    gj          <-- '*CIEN*'
    gk          <-- '*UN*'
    gl          <-- '*CIENTO*'
    gm          <-- '*Y*'
    gz          <-- '*.*'
    MIENTRAS lg > 1
       al       <-- Indica la posición donde un espacio en blanco se
                    encuentra en la cadena "cadnum"
       toknum&n <-- Subcadena desde 1 hasta al-1 de la cadena "cadnum"
       cadnum   <-- cadnum - toknum&n
       lg       <-- Longitud(cadnum)
       n        <-- Cadena (VAL(n) + 1 )
    FIN DEL DO
    n_t             <-- VAL(n) - 1
    ncar            <-- 0
    estado          <-- 0
    MIENTRAS ncar<= n_t
        n           <-- cadena(ncar +1)
        car         <-- ' '
        toknum&n    <-- '*'+toknum&n+'*'
        CONDICION toknum&n
            Se encuentra en ga :  car     <-- 'A'
            Se encuentra en gb :  car     <-- 'B'
            Se encuentra en gc :  car     <-- 'C'
            Se encuentra en gd :  car     <-- 'D'
```

```
        Se encuentra en ge :   car     <-- 'E'
        Se encuentra en gf :   car     <-- 'F'
        Se encuentra en gg :   car     <-- 'G'
        Se encuentra en gh :   car     <-- 'H'
        Se encuentra en gi :   car     <-- 'I'
        Se encuentra en gj :   car     <-- 'J'
        Se encuentra en gk :   car     <-- 'K'
        Se encuentra en gl :   car     <-- 'L'
        Se encuentra en gm :   car     <-- 'M'
        Se encuentra en gz :   car     <-- 'Z'
        EN OTRO CASO           Desplegamos el error, indicando su posición
                               Leemos (toknum&n)
                               Continuamos saltando al FIN DEL DO
FIN DE LA CONDICION
le              <-- Longitud(toknum&n)
toknum&n  <-- Subcadena desde 2 hasta le-2 de la cadena
                  "toknum&n"
simbolo <-- 'A B C D E F G H I J K L M '
CONDICION       estado
     0: estados    <-- '1 9 2 2 11  4   5 2 1410  '
        estado     <-- VAL(subcadena desde la posición de"car" en
                       "simbolo" hasta tomar dos caracteres de la
                       cadena "estados")
        Sl estado  = 0 ENTONCES      fl<-- 1
     1: Sl car     = 'Z' ENTONCES Salimos del FIN DEL DO
        estados    <--'          11  405         12'
        estado     <-- VAL(subcadena desde la posición de "car" en
                       "simbolo" hasta tomar dos caracteres de la
                       cadena "estados")
        Sl estado  = 0 ENTONCES      fl<-- 1
     2: Sl car     = 'Z' ENTONCES Salimos del FIN DEL DO
        estados    <--'          11  405          '
        estado     <-- VAL(subcadena desde la posición de "car" en
                       "simbolo" hasta tomar dos caracteres de la
                       cadena "estados")
        Sl estado  = 0 ENTONCES      fl<-- 1
     3: Sl car     = 'Z' ENTONCES Salimos del FIN DEL DO
                       SINO fl <-- 1
     4: CONDICION car
           'Y'                     : estado       <-- 5
           'Z'                     : Salimos del FIN DEL DO
           EN OTRO CASO            : fl     <-- 1
        FIN DE LA CONDICION
     5: estados    <--'          11  405         12'
        estado     <-- VAL(subcadena desde la posición de "car" en
                       "simbolo" hasta tomar dos caracteres de
                       la cadena "estados")
        Sl estado  = 0 ENTONCES fl <-- 1
     6: CONDICION car
```

```
        'M'                 : estado      <-- 7
        'Z'                 : Salimos del FIN DEL DO
        EN OTRO CASO        : fl      <-- 1
    FIN DE LA CONDICION
7: SI car = 'C' ENTONCES estado   <-- 3
              SINO fl<-- 1
8: CONDICION car
        'C'                 : estado      <-- 3
        'Z'                 : Salimos del FIN DEL DO
        EN OTRO CASO        : fl          <-- 1
    FIN DE LA CONDICION
9: SI car = 'Z' ENTONCES Salimos del FIN DEL DO
    estados     <-- '1  2 2 1 1   405      '
    estado      <-- VAL(subcadena desde la posición de "car" en
                    "simbolo" hasta tomar dos caracteres de
                    la cadena "estados")
    SI estado   = 0 ENTONCES fl <--
10: estados     <-- '1  2 2       5        '
    estado      <-- VAL(subcadena desde la posición de "car" en
                    "simbolo" hasta tomar dos caracteres de
                    la cadena "estados")
    SI estado   = 0 ENTONCES fl <-- 1
11: SI car = 'Z' ENTONCES Salimos del FIN DEL DO
    estados     <-- '21312222    405 22  33  '
    estado      <-- VAL(subcadena desde la posición de "car" en
                    "simbolo" hasta tomar dos caracteres de
                    la cadena "estados")
    SI estado   = 0 ENTONCES fl <-- 1
12: SI car = 'C' ENTONCES estado  <-- 2
              SINO fl       <-- 1
14: SI car = 'F' ENTONCES estado  <-- 40
              SINO fl       <-- 1
21: CONDICION car
        'H'                 : estado      <-- 40
        'I'                 : estado      <-- 5
        'M'                 : estado      <-- 37
        'Z'                 : Salimos del FIN DEL DO
        EN OTRO CASO        : fl          <-- 1
    FIN DE LA CONDICION
22: CONDICION car
        'H'                 : estado      <-- 40
        'I'                 : estado      <-- 5
        'Z'                 : Salimos del FIN DEL DO
        EN OTRO CASO        : fl      <-- 1
    FIN DE LA CONDICION
31: SI car = 'Z' ENTONCES Salimos del FIN DEL DO
    estados     <-- '21  2222    405       '
    estado      <-- VAL(subcadena desde la posición de "car" en
                    "simbolo" hasta tomar dos caracteres de
```

```
                la cadena "estados")
   SI estado   = 0 ENTONCES fl <-- 1
33: estados    <-- '21  2222      5       '
   estado      <-- VAL(subcadena desde la posición de "car" en
                   "simbolo" hasta tomar dos caracteres de
                   la cadena "estados")
   SI estado   = 0 ENTONCES fl <-- 1
37: SI car = 'C' ENTONCES estado<-- 22
                SINO fl                <-- 1
40: SI car = 'Z' ENTONCES Salimos del FIN DEL DO
   estados     <-- '4151424260     5 42  53  '
   estado      <-- VAL(subcadena desde la posición de "car" en
                   "simbolo" hasta tomar dos caracteres de
                   la cadena "estados")
   SI estado   = 0 ENTONCES fl <-- 1
41: CONDICION car
       'E'                    : estado        <-- 60
       'Y'                    :estado<-- 5
       'M'                    :estado<-- 57
       'Z'                    : Salimos del FIN DEL DO
       EN OTRO CASO           : fl      <-- 1
   FIN DE LA CONDICION
42: CONDICION car
       'E'                    :estado<-- 60
       'Y'                    :estado<-- 5
       'Z'                    :Salimos del FIN DEL DO
       EN OTRO CASO           : fl      <-- 1
   FIN DE LA CONDICION
51: SI car = 'Z' ENTONCES Salimos del FIN DEL DO
   estados     <-- '41  424260     5       '
   estado      <-- VAL(subcadena desde la posición de "car" en
                   "simbolo" hasta tomar dos caracteres de
                   la cadena "estados")
   SI estado   = 0 ENTONCES fl <-- 1
53: SI car = 'Z' ENTONCES Salimos del FIN DEL DO
   estados     <-- '41  4242      5       '
   estado      <-- VAL(subcadena desde la posición de "car" en
                   "simbolo" hasta tomar dos caracteres de
                   la cadena "estados")
   SI estado   = 0 ENTONCES fl <-- 1
57: SI car = 'C' ENTONCES estado<-- 42
                SINO fl        <-- 1
60: SI car = 'Z' ENTONCES Salimos del FIN DEL DO
   estados     <-- '61716262      5 62  73  '
   estado      <-- VAL(subcadena desde la posición de "car" en
                   "simbolo" hasta tomar dos caracteres de
                   la cadena "estados")
   SI estado   = 0 ENTONCES fl <-- 1
61: CONDICION car
```

```
                'Y'                 : estado        <-- 5
                'M'                 : estado        <-- 77
                'Z'                 : Salimos del FIN DEL DO
                EN OTRO CASO        : fl    <-- 1
            FIN DE LA CONDICION
        62: CONDICION car
                'Y'                 :estado<-- 5
                'Z'                 :Salimos del FIN DEL DO
                EN OTRO CASO        :       fl <-- 1
            FIN DE LA CONDICION
        71: SI car = 'Z' ENTONCES Salimos del FIN DEL DO
            estados     <-- '61 6262    5       '
            estado      <-- VAL(subcadena desde la posición de "car" en
                            "simbolo" hasta tomar dos caracteres de
                            la cadena "estados")
            SI estado   = 0 ENTONCES fl <-- 1
        73: estados     <-- '61 6262    5       '
            estado      <-- VAL(subcadena desde la posición de "car" en
                            "simbolo" hasta tomar dos caracteres de
                            la cadena "estados")
            SI estado   = 0 ENTONCES fl <-- 1
        77: SI car = 'C' ENTONCES estado<-- 62
                        SINO  fl    <-- 1
    FIN DE LA CONDICION
    ncar    <-- ncar + 1
    SI fl = 1
    ENTONCES
        serror  <-- 'ERROR DE SEMANTICA'
        aerror  <-- toknum&n
        error   <-- VERDADERO
        TERMINA EL PROCEDIMIENTO
FIN DEL DO
aux         <-- 0
entera      <-- 0
dig         <-- 0
cantida     <-- 0
cantcom     <-- 0
ajuste      <-- 0
j           <-- 1
ncar        <-- 0
word        <-- "
flag        <-- FALSO
ban1        <-- FALSO
ban3        <-- FALSO
ban         <-- FALSO
paso        <-- VERDADERO
token       <-- "
n           <-- "
MIENTRAS ncar       < n_t
```

```
ncar    <-- ncar + 1
n       <-- Cadena(ncar)
token   <-- toknum&n
SI token = 'Y'
ENTONCES
    Continuamos al FIN DEL DO
SINO
    product <-- 1
    SI (flag) Y (token = 'MIL')
    ENTONCES
        cantida     <-- cantida + 1000
        token       <-- ''
        paso        <-- FALSO
        flag        <-- FALSO
        SI (token = 'MIL') O (token = 'MILLON') O
           (token = 'MILLONES')
        ENTONCES
            SI token = 'MIL'
            ENTONCES
                SI cantida = 0 ENTONCES cantida<-- 1
                product <-- 1000
            SINO
                product <-- 1000000
                word    <-- token
                flag    <-- VERDADERO
            token <-- ''
        SI token = 'PUNTO'
        ENTONCES
            entera      <-- cantcom + cantida
            ban   <-- VERDADERO
            token <-- ''
            num   <-- 0
            ban3  <-- VERDADERO
            cantida     <-- 0
        SI (token = 'CERO') Y (ban)
        ENTONCES
            ajuste <-- ajuste + 1
            token <-- ''
        SI paso
        ENTONCES
            num   <-- 0
            SI Longitud(token) > 0
            ENTONCES
                Llama a CAMBIA con parámetros (token,num)
                token    <-- ''
            SI ban1
            ENTONCES
                cantcom <-- cantida
                cantida  <-- 0
```

```
                    cantida  <-- cantida + num
                    ban1     <-- FALSO
                SINO
                    cantida  <-- (cantida + num) * product
                SI Longitud(word) > 0
                ENTONCES
                    ban1     <-- VERDADERO
                    word     <-- "
            paso             <-- VERDADERO
    FIN DEL DO
    SI la NEGACION(ban3) ENTONCES cantida<-- cantcom + cantida
    SI ban
    ENTONCES
        aux      <-- cantida
        MIENTRAS VERDADERO
            dig <-- dig + 1
            aux<-- aux / 10
            SI Redondeo(aux-.50,0) = 0.0
            ENTONCES Salimos del FIN DEL DO
        FIN DEL DO
        fraccio  <--1
        j        <--1
        MIENTRAS j<= (dig+ajuste)
            fraccio<-- fraccio * 10
            j        <-- j + 1
        FIN DEL DO
        numero   <-- entera + (cantida/fraccio)
    SINO
        numero   <-- cantida
FIN<Convertidor letra a número>
```

B)

NOMBRE : stoken(oracion,nt)

FUNCION : Separar y contar de la variable de paso "oracion", los token's que la conforman. Almacenándolos en el arreglo global dinámico tipo macro **token&n**. Retorna en la variable "nt" el número de token's encontrados. Asimismo, se definen los rasgos lexicográficos de cada uno de ellos como son: género, número, tipo, regla y significado en arreglos bidimensionales, dinámicos y globales.

EJEMPLO : Si la variable "oracion" es igual a "LISTA 3 MUJERES", entonces stoken(oracion,nt) nos da:
nt = 3
Variables que genera:

token1	= 'LISTA'	token2	='3'	token3	='MUJERES'
ttoken1_1	= 'L'	ttoken2_1	='Q'	ttoken3_1	='S'
rtoken1_1	= 'a'	rtoken2_1	='Z'	rtoken3_1	='P'
stoken1_1	= 'LIST'	stoken2_1	='3'	stoken3_1	=',NOT. SEXO'
genume1_1	= 1	genume2_1	="	genume3_1	=4

PROCEDIMIENTOS UTILIZADOS :

buscar(token,no) = Al enviar el token, por ejemplo "VIVO", se retorna en la variable de paso "no" el número de veces en que fue encontrado el token en el archivo DICCIONARIO. En caso de no encontrarlo al menos una vez, se enciende la bandera de error. Esta bandera es una variable global que se opera en todo el sistema.

coletnum(letra,numero) = Al enviar la expresión numérica en la variable "letra". Se realiza un análisis sintáctico y semántico antes de evaluar la cantidad en letra. Si la expresión cumplió la sintaxis y semántica, se lleva a cabo la conversión de letra a número, regresando la cantidad en la variable de paso "numero".

MACROS UTILIZADAS :

pos&j=	Es un arreglo dinámico que genera variables globales desde pos&**1** hasta pos&**5**. En cada una de ellas se guarda la posición del campo que indica el género y número en el registro del archivo REGLAS del correspondiente token.
rh&j=	Es un arreglo dinámico que genera variables globales desde rh&**1** hasta rh&**n**. En cada una de ellas se guarda el apuntador al registro donde el token fue encontrado en el archivo del DICCIONARIO.
Token&n=	Es un arreglo dinámico que genera variables globales desde token**1** hasta token**n** dependiendo del número de token's que tenga la variable "oracion".
Notok&n=	Es un arreglo dinámico que contiene el número de veces en que el token fue encontrado en el archivo DICCIONARIO.
Ttoken&n&j=	Es un arreglo bidimensional dinámico que genera variables globales. Donde **n** representa el n-ésimo token y **j** el j-ésimo tipo de palabra.
Rtoken&n&j=	Es un arreglo bidimensional dinámico que genera variables globales. Donde **n** representa el n-ésimo token y **j** la j-ésima regla gramatical.
Stoken&n&j=	Es un arreglo bidimensional dinámico que genera variables globales. Donde **n** representa el n-ésimo token y **j** el j-ésimo significado.
Genume&n&j=	Es un arreglo bidimensional dinámico que genera variables globales. Donde **n** representa el n-ésimo token y **j** el j-ésimo género y número de la palabra.

NOTA: Las macros pos&j y rh&j son generadas en el procedimiento buscar.

VARIABLES UTILIZADAS :

oracion=	Cadena que contiene la consulta a analizar.
pcaden=	Cadena que contiene el contenido de la variable "oracion" y con la cual vamos a trabajar.
nt=	Número de token's en la variable "pcaden".
lgl=	Longitud de la variable "pcaden"
token=	Cadena que es utilizada para trabajar cada uno de los token's.
limit=	Cadena que contiene los delimitadores para constante y número en letra, que se dan en la edición de la consulta.
tlimi=	Cadena que contiene los tipos de palabra que describen una constante y número en letra.
n=	Carácter utilizado en todas las macros para identificar el índice de los renglones.
j=	Carácter utilizado en todas las macros para identificar el índice de las columnas.
digito=	Cadena que contiene el conjunto de los números arábigos.
noise=	Cadena que contiene todas las palabras a ser eliminadas por el método NOISE DISPOSAL NOISE.
c=	Indice utilizado en un ciclo.
al=	Posición donde un espacio en blanco es encontrado dentro de la cadena "pcaden" .
lpcad=	Posición donde un carácter de "pcaden" es encontrado dentro de la cadena "limit".
climit=	Carácter que identifica el tipo de carácter encontrado en la cadena "limit".
num=	Valor numérico que guarda la cantidad que se convirtió de letra a número.
letras=	Cadena que contiene la expresión numérica expresada en letra.
serror=	Cadena que guarda el significado del error en caso de haberlo.
aerror=	Cadena que guarda el token donde se encontro un error.
error_=	Valor lógico que indica si hubo o no error.

PSEUDO-CODIGO

```
Inicio<Separador, contador e identificador de token's>
    token     <-- Cadena de 60 blancos
    pcaden    <-- oracion + ' '
    lgl       <-- Longitud(pcaden)
    limit     <-- '"/'
    tlimit    <-- 'OP'
    n         <-- '1'
    digito    <-- '0123456789'
```

```
noise          <-- '*CON*QUE*SE*A*DE*DEL*'
MIENTRAS lg1 > 1
    SI la primera posición de "pcaden" se encuentra en "digito"
    ENTONCES
        a   <-- 1
        MIENTRAS  la posición de "a" en "pcaden" se encuentre en
                  "digito"
            a   <-- a + a
        FIN DEL DO
        SI La subcadena desde "a" hasta "a" de la cadena
          "pcaden"    = ' '
        ENTONCES
            token&n          <-- Subcadena desde "1" hasta "a-1" de la
                                 cadena "pcaden"
            pcaden           <-- Subcadena desde "a" hasta "lg1-a+1"
                                 de la cadena "pcaden"
            lg1              <-- Longitud(pcaden)
            notok&n          <-- 1
            ttoken&n_1       <-- 'Q'
            rtoken&n_1       <-- 'Z'
            stoken&n_1       <-- token&n
            n                <-- Cadena(VAL(N)+1)
            Continuamos saltando al FIN DEL DO
        SINO
            serror           <-- 'No se pueden combinar letras y
            números en un número'
            aerror           <-- Subcadena desde "1" hasta la posición
                                 donde se encuentre un blanco en
                                 "pcaden" de la cadena "pcaden"
            error            <-- VERDADERO
            TERMINA EL PROCEDIMIENTO
        lpcad   <-- Posición donde el primer carácter de "pcaden"
                    se encuentra dentro de la cadena "limit"
        SI lpcad<> 0
        ENTONCES
            climit      <-- Subcadena desde "lpcad" hasta "lpcad"
                            de la cadena "limit"
            a1          <-- Posición donde el valor de "climit" se
                            encuentra en la subcadena desde "2" hasta
                            "lg1-1" de la cadena "pcaden"
            token&n     <-- Subcadena desde "1" hasta "a1+1" de la
                            cadena "pcaden"
            pcaden      <-- Subcadena desde "a1+2" hasta "lg1-a1+1" de
                            la cadena "pcaden"
            lg1         <-- Longitud(pcaden)
            notok&n     <-- 1
            ttoken&n_1  <-- Subcadena desde "lpcad" hasta "lpcad"
                            de la cadena "tlimi"
            rtoken&n_1  <-- 'Z'
```

```
                    SI lpcad = 1
                    ENTONCES
                        stoken&n_1    -- token&n
                    SINO
                        num           -- 0
                        letras        -- Subcadena desde "2" hasta
                                         "Longitud(token&n)-2" de la cadena
                                         token&n
                        Llama a COLETNUM con parámetros (letras,num)
                        stoken&n_1    --Cadena(num)
                    n  <--Cadena(VAL(N)+1)
                    Continuamos saltando al FIN DEL DO
                a1  <--Posición donde un blanco es encontrado en la cadena
                       "pcaden"
                SI La posición donde ("*"+subcadena desde "1" hasta "a1-1" de la
                   cadena "pcaden"+"*" ) es encontrado en la cadena "noise"
                ENTONCES
                    pcaden      <--Subcadena desde "a1+1" hasta "lg1-a1" de la
                                   cadena "pcaden"
                    lg1         <--Longitud(pcaden)
                    Continuamos saltando al FIN DEL DO
                token&n         <--Subcadena desde "1" hasta "a1-1" de la cadena
                                   "pcaden"
                pcaden          <--Subcadena desde "a1+1" hasta "lg1-a1" de la
                                   cadena "pcaden"
                lg1             <--Longitud(pcaden)
                no              <-- 0
                Llama a BUSCAR con parámetros (token&n,no)
                SI error_ ENTONCES  TERMINA EL PROCEDIMIENTO
                notokn&n        <-- no
                c               <-- 0
                MIENTRAS c<  no
                    c                   <-- c + 1
                    j                   <-- Cadena(c)
                    genume&n_&j         <-- Pos&j
                    Abrimos el Archivo(DICCIONARIO)
                    Posicionamos en el registro rh&j
                    stoken&n_&j         <-- Campo(SIGNIFICAD)
                    ttoken&n_&j         <-- Campo(TIPO)
                    rtoken&n_&j         <-- Campo(REGLA)
                FIN DEL DO
                n               <-- Cadena(VAL(n)+1)
        FIN DEL DO
        nt          <--VAL(n) + 1
Fin<Separador, contador e identificador de token's>
```

(')

NOMBRE : cmnd(comando,nt)

FUNCION : Busca un comando en la variable global "consulta"; en caso de encontrarlo, se retorna en la variable de paso "comando". También recibe la variable de paso "nt" que contiene el número de token's.

EJEMPLO : Si la variable global "consulta" es igual a "**LISTA** LAS MUJERES CASADAS", entonces cmnd(comando,nt) nos regresa en la variable "comando" la expresión "LIST".

MACROS UTILIZADAS :

Notok&n=	Es un arreglo dinámico que contiene el número de veces en que el token fue encontrado en el archivo DICCIONARIO.
Ttoken&n&j =	Es un arreglo bidimensional dinámico que genera variables globales. Donde **n** representa el n-ésimo token y **j** el j-ésimo tipo de palabra.
Stoken&n&j =	Es un arreglo bidimensional dinámico que genera variables globales. Donde **n** representa el n-ésimo token y **j** el j-ésimo significado.

VARIABLES UTILIZADAS :

comando=	Cadena a la cual se le asigna la expresión que es valida para un comando.
nt=	Valor entero que contiene el número de token's a evaluar.
ap_token =	Valor entero que indica el token sobre el cual estamos trabajando.
mas=	Cadena que se utiliza para concatenar una sintaxis al momento de generar la cadena "comando".
n =	Carácter utilizado en todas las macros para identificar el índice de los renglones.
j=	Carácter utilizado en todas las macros para identificar el índice de las columnas.
c=	Indice utilizado en un ciclo.

PSEUDO-CODIGO

```
Inicio<Reconocedor de la gramática de un comando>
    n               <-- Cadena(ap_token)
    SI notok&n<> 0
    ENTONCES
        SI ttoken&n._1 = 'L'
        ENTONCES
            comando         <-- stoken&n._1
```

```
        ap_token        -- ap_token + 1
        SI ap_token > nt ENTONCES TERMINA EL
                                    PROCEDIMIENTO
        n               <-- Cadena(ap_token)
        c               <-- 0
        MIENTRAS c < notok&n
            c           <-- c + 1
            j           <-- Cadena(c)
            SI ttoken&n._&j se encuentra en 'MPQ'
            ENTONCES
                SI ttoken&n._&j = 'M'
                ENTONCES
                    mas             <-- ' '
                SINO
                    mas             <-- 'NEXT'
                comando     <-- comando + mas + stoken&n._&j
                p_token     -- ap_token + 1
                salimos del FIN DEL DO
        FIN DEL DO
Fin <Reconocedor de la gramática de un comando>
```

D)

NOMBRE : cmndbd(campo_bd,nt)

FUNCION : Busca un campo de la base de datos en la variable global "consulta"; en caso de encontrarlo, se retorna en la variable de paso "campo_bd". También recibe la variable de paso "nt" que contiene el número de token's.

EJEMPLO :Si la variable global "consulta" es igual a "LISTA **LAS DIRECCIONES Y LOS TELEFONOS** DE LAS MUJERES CASADAS", entonces cmndbd(campo_bd,nt) nos regresa en la variable "campo_bd" la expresión "DIRECCION, TELEFONO".

MACROS UTILIZADAS :

Notok&n=	Es un arreglo dinámico que contiene el número de veces en que el token fue encontrado en el archivo DICCIONARIO.
Ttoken&n&j =	Es un arreglo bidimensional dinámico que genera variables globales. Donde **n** representa el n-ésimo token y **j** el j-ésimo tipo de palabra.
Stoken&n&j =	Es un arreglo bidimensional dinámico que genera variables globales. Donde **n** representa el n-ésimo token y **j** el j-ésimo significado.

Genume&n&j =	Es un arreglo bidimensional dinámico que genera variables globales. Donde n representa el n-ésimo token y j el j-ésimo género y número de la palabra.
Token&n=	Es un arreglo dinámico que genera variables globales desde token1 hasta tokenn dependiendo del número de token's que tenga la variable "oracion".

VARIABLES UTILIZADAS :

campo_bd =	Cadena a la cual se le asigna la expresión que es valida para un campo de base de datos.
nt=	Valor entero que contiene el número de token's a evaluar.
ap_token =	Valor entero que indica el token sobre el cual estamos trabajando.
mas=	Cadena que se utiliza para concatenar una sintaxis al momento de generar la cadena "comando".
n=	Carácter utilizado en todas las macros para identificar el índice de los renglones.
d=	Carácter utilizado en algunas macros para identificar el índice de las columnas.
c=	Indice utilizado en un ciclo y además utilizado en algunas macros para identificar el índice de las columnas.
ban=	Valor lógico para manejar el control de análisis.
serror=	Cadena que guarda el significado del error en caso de haberlo.
aerror=	Cadena que guarda el token donde se encontro un error.
error_=	Valor lógico que indica si hubo o no error.
ultcmp=	Cadena que contiene el último campo de base de datos encontrado.
genum=	Valor entero que indica el género y número de un token que este contenido en la subcadena que representa un campo de base de datos.

PSEUDO-CODIGO

```
Inicio <Reconocedor de la gramática de un campo de la base de datos>
    n           <-- Cadena(ap_token)
    SI error_ ENTONCES TERMINA EL PROCEDIMIENTO
    ban         <-- FALSO
    SINOtok&n<> 0
    ENTONCES
        c       <-- '|'
        MIENTRAS VAL(c)       <= notok&n
            SI ttoken&n._&c = 'K'
            ENTONCES
                campo_bd  <-- campo_bd + stoken&n._&c + ' '
                ultcmp    <-- stoken&n._&c
                ap_token  <-- ap_token + 1
```

```
                SI ap_token > nt
                ENTONCES
                    error    -- FALSO
                    TERMINA EL PROCEDIMIENTO
                n           <-- Cadena(ap_token)
                ban         <-- VERDADERO
                Salimos del FIN DEL DO
            c   <-- Cadena(VAL(c) + 1)
        FIN DEL DO
    SI notok&n<> 0  Y  NEGACION(ban)
    ENTONCES
        c       <-- '1'
        MIENTRAS VAL(c)<= notok&n
            SI ttoken&n._&c = 'H'
            ENTONCES
                ap_token  <-- ap_token + 1
                SI ap_token > nt
                ENTONCES
                    serror <-- 'La consulta esta inconclusa'
                    aerror <-- token&n
                    error_      <-- VERDADERO
                    TERMINA EL PROCEDIMIENTO
                genum     <-- genume&n._&c
                n         <-- Cadena(ap_token)
                SI notok&n<> 0
                ENTONCES
                    d     <-- '1'
                    MIENTRAS VAL(d)<= notok&n
                        SI ttoken&n._&d = 'K'
                        ENTONCES
                            campo_bd  <--campo_bd + stoken&n._&d + ''
                            dtemp     <--stoken&n._&d
                            ap_token  <--ap_token + 1
                            SI genum<> genum&n._&d
                            ENTONCES
                                serror <-- 'Género o número
                                           incompatibles'
                                d          <-- Cadena(VAL(n) -1)
                                aerror <-- token&d + '' + token&n
                                error_ <-- VERDADERO
                                TERMINA EL PROCEDIMIENTO
                            SI ap_token > nt
                            ENTONCES
                                error      <-- FALSO
                                TERMINA EL PROCEDIMIENTO
                            n          <-- Cadena(ap_token)
                            ban        <-- VERDADERO
                            Salimos del FIN DEL DO
                        SINO
```

```
                        ap_token   <-- ap_token - 1
                        n          <-- Cadena(ap_token)
                        TERMINA EL PROCEDIMIENTO
                    d       <-- Cadena(VAL(d) + 1)
                FIN DEL DO
            SI ban ENTONCES Salimos del FIN DEL DO
            c   <-- Cadena(VAL(c) + 1)
    FIN DEL DO
SI notok&n<> 0 Y ban
ENTONCES
    c       <-- '|'
    MIENTRAS VAL(c)<= notok&n
        SI ttoken&n._&c = 'F'
        ENTONCES
            campo_bd <-- campo_bd + ' '
            ap_token  <-- ap_token + 1
            SI ap_token > nt
            ENTONCES
                serror      <-- 'La conjunción quedo incompleta
                aerror      <-- token&n
                error_      <-- VERDADERO
                TERMINA EL PROCEDIMIENTO
            n   <-- Cadena(ap_token)
            Llama a CMNDBD con parámetros (campo_bd,nt)
            TERMINA EL PROCEDIMIENTO
        SINO
            TERMINA EL PROCEDIMIENTO
        c   <-- Cadena(VAL(c) + 1)
    FIN DEL DO
Fin<Reconocedor de la gramática de un campo de la base de datos>
```

E)

NOMBRE : cndcn(condicion,nt)

FUNCION : Busca una condición en la variable global "consulta"; en caso de encontrarla, se relorna en la variable de paso "condicion". También recibe la variable de paso "nt" que contiene el número de token's.

EJEMPLO: Si la variable global "consulta" es igual a "LISTA LAS DIRECCIONES Y LOS TELEFONOS **DE LAS MUJERES CASADAS**", entonces cndcn(condicion,nt) nos regresa en la variable "condicion" la expresión "(.NOT.SEXO) .AND. (EDO_CIVIL="C" .AND. .NOT.SEXO)".

PROCEDIMIENTOS UTILIZADOS :

cmndbd(campo_bd,nt)=Al invocar este procedimiento, la variable "campo_bd" retorna un campo de la base de datos, si se encuentra en la variable global "consulta".

MACROS UTILIZADAS :

Natok&n =	Es un arreglo dinámico que contiene el número de veces en que el token fue encontrado en el archivo DICCIONARIO.
Ttoken&n&j =	Es un arreglo bidimensional dinámico que genera variables globales. Donde **n** representa el n-ésimo token y **j** el j-ésimo tipo de palabra.
Stoken&n&j =	Es un arreglo bidimensional dinámico que genera variables globales. Donde **n** representa el n-ésimo token y **j** el j-ésimo significado.
Genume&n&j =	Es un arreglo bidimensional dinámico que genera variables globales. Donde **n** representa el n-ésimo token y **j** el j-ésimo género y número de la palabra.
Token&n=	Es un arreglo dinámico que genera variables globales desde token**1** hasta token**n** dependiendo del número de token's que tenga la variable "oracion".

VARIABLES UTILIZADAS :

condicion =	Cadena a la cual se le asigna la expresión que es válida para una condición.
nt=	Valor entero que contiene el número de token's a evaluar.
ap_token=	Valor entero que indica el token sobre el cual estamos trabajando.
apto=	Valor entero que nos sirve de auxiliar para guardar el apuntador del token sobre el cual estamos trabajando.
n=	Carácter utilizado en todas las macros para identificar el índice de los renglones.
j=	Carácter utilizado en algunas macros para identificar el índice de las columnas.
d=	Carácter utilizado en algunas macros para identificar el índice de las columnas.
c=	Índice utilizado para los ciclos.
ban=	Valor lógico para manejar el control de análisis.
seanid=	Valor lógico para manejar el control de análisis.
serror=	Cadena que guarda el significado del error en caso de haberlo.
aerror=	Cadena que guarda el token donde se encontro un error.
error_=	Valor lógico que indica si hubo o no error.
ultcmp=	Cadena que contiene el último campo de base de datos encontrado.
ultor=	Cadena que contiene el último operador relacional encontrado.

genum=	Valor entero que indica el género y numero de un token que este contenido en la subcadena que representa un campo de base de datos.
cmp_base=	Cadena que contiene la expresión de un campo de la base de datos al llamar al procedimiento cmndbd.
cndaux=	Cadena auxiliar que se utiliza para guardar una condición en caso de que la consulta tenga otra condición.
cond=	Cadena auxiliar que se utiliza para guardar una condición en caso de que la consulta tenga otra condición.

PSEUDO-CODIGO

```
Inicio<Reconocedor de la gramática de una condición>
    n           <-- Cadena(ap_token)
    cmp_base    <-- ' '
    Llama a CMNDBD con parámetros (cmp_base,nt)
    n           <-- Cadena(ap_token)
    ban         <-- VERDADERO
    SI notok&n <>  0
    ENTONCES
        SI ttoken&n._1 = 'R'
        ENTONCES
            ap_token        <-- ap_token + 1
            ultor           <-- stoken&n._1
            ban             <-- FALSO
            Llama a CMNDBD con parámetros (cmp_base,nt)
            n               <-- Cadena(ap_token)
            SI Longitud(ultcmp) = 0
            ENTONCES
                error_      <-- VERDADERO
                serror      <-- 'No se definió campo para la condición'
                aerror      <-- token&n
                TERMINA EL PROCEDIMIENTO
            CONDICION  <-- condicion + ultcmp + ultor
            SI ap_token > nt
            ENTONCES
                serror      <-- 'Se esperaba una constante'
                aerror      <-- token&n
                error_      <-- VERDADERO
                TERMINA EL PROCEDIMIENTO
            n   <-- Cadena(ap_token)
            c   <-- 0
            MIENTRAS c< notok&n
                c           <-- c + 1
                j           <-- Cadena(c)
                ban         <-- VERDADERO
                SI ttoken&n._&j se encuentra en 'OPQ'
                ENTONCES
```

```
                condicion       -- condicion = stoken&n._&j
                ap_token        -- ap_token = 1
                n               <-- Cadena(ap_token)
                ban             <-- FALSO
                seanid          <-- FALSO
                Salimos del FIN DEL DO
        FIN DEL DO
        SI ban
        ENTONCES
            serror      <-- 'Se esperaba una constante'
            aerror      <-- token&n
            error_      <-- VERDADERO
SI ap_token >           nt ENTONCES TERMINA EL
                                    PROCEDIMIENTO
SI ttoken&n._1 Se encuentra en 'OPQ' Y ban
ENTONCES
    SI Longitud(ultcmp)<> 0  Y  Longitud(ultor)<> 0
    ENTONCES
        condicion   <-- condicion+ultcmp+ultor+stoken&n._1
        ap_token    <-- ap_token + 1
        n           <-- Cadena(ap_token)
        seanid      <-- FALSO
        ban         <-- FALSO
SI ap_token >           nt ENTONCES TERMINA EL
                                    PROCEDIMIENTO
SI ttoken&n._1 ='H'
ENTONCES
    ban             <-- FALSO
    ap_token        <-- ap_token + 1
    SI ap_token >  nt
    ENTONCES
        serror      <-- 'Consulta inconclusa'
        aerror      <-- token&n
        error_      <-- VERDADERO
        TERMINA EL PROCEDIMIENTO
    genum           <-- genume&n._1
    n               <-- Cadena(ap_token)
    SI notok&n<>  0
    ENTONCES
        d           <-- '1'
        MIENTRAS VAL(d)<= notok&n
            SI ttoken&n._&d = 'S'
            ENTONCES
                condicion       <-- condicion + stoken&n._1
                SI genum<> genume&n._&d
                ENTONCES
                    d           <-- Cadena(VAL(n)-1)
                    serror      <-- 'Genero o número incompatible'
                    aerror      <-- token&d+' '+token&n
```

```
                    error       -- VERDADERO
                    TERMINA EL PROCEDIMIENTO
                ap_token        -- ap_token + 1
                n               -- Cadena(ap_token)
                seanid          -- FALSO
                Salimos del FIN DEL DO
            d   <-- Cadena(VAL(d)+1)
        FIN DEL DO
Si ap_token     = nt  Y  ban  Y  ttoken&n._1 = 'S'
ENTONCES
    condicion   <-- condicion + stoken&n._1
    ap_token    <-- ap_token + 1
    n           <-- Cadena(ap_token)
    ban         <-- FALSO
    seanid      <-- FALSO
n       <-- Cadena(ap_token)
Si ap_token <= nt  Y  ban  Y  token&n ='('
ENTONCES
    ap_token    <-- ap_token + 1
    n           <-- Cadena(ap_token)
    cndaux      <-- ""
    Llama a CNDCN con parámetros (cndaux,nt)
    n           <-- Cadena(ap_token)
    Si NEGACION(error_)
    ENTONCES
        Si ap_token > nt
        ENTONCES
            serror      <-- 'Falta cerrar paréntesis'
            aerror      <-- ' '
            error       <-- VERDADERO
            TERMINA EL PROCEDIMIENTO
        Si token&n<> ')'
        ENTONCES
            Llama a CNDCN con parámetros (cndaux,nt)
            Si error_ ENTONCES  TERMINA EL
                                PROCEDIMIENTO
            n           <-- Cadena(ap_token)
            Si token&n<> ')'
            ENTONCES
                serror  <-- 'Falta cerrar paréntesis'
                aerror  <-- token&n
                error_  <-- VERDADERO
                TERMINA EL PROCEDIMIENTO
        Si Longitud(cndaux) = 0
        ENTONCES
            serror      <-- 'Falta la condición entre los paréntesis'
            aerror      <-- '( )'
            error       <-- VERDADERO
            TERMINA EL PROCEDIMIENTO
```

```
        condicion   -- condicion + '( '+endaux+' )'
        ap_token    -- ap_token + 1
        n           -- Cadena(ap_token)
        ban         -- FALSO
SI ap_token<= nt
ENTONCES
    SI token&n Se encuentra en '*Y*O*NO*U*'
    ENTONCES
        CONDICION token&n
            'Y'         condicion   <-- condicion + '.AND.'
            Se encuentra
            en '*O*U*'  condicion   <-- condicion + '.OR.'
            'NO'        condicion   <-- condicion + '.NOT.'
        FIN DE LA CONDICION
        ap_token  <-- ap_token + 1
        SI ap_token > nt
        ENTONCES
            serror      <-- 'La condición quedo incompleta '
            aerror      <-- token&n
            error_      <-- VERDADERO
            TERMINA EL PROCEDIMIENTO
        n               <-- Cadena(ap_token)
        cond            <-- ''
        apto            <-- ap_token
        Llama a CNDCN con parámetros (cond,nt)
        SI Longitud(cond) = 0
        ENTONCES
            ap_token    <-- apto
            n           <-- Cadena(ap_token)
            d           <-- Cadena(ap_token - 1)
            serror      <-- 'La condición es erronea '
            aerror      <-- token&d + ' ' + token&n
            error_      <-- VERDADERO
            TERMINA EL PROCEDIMIENTO
        condicion       <-- condicion + cond
    SINO
        SI token&n = ')'  O  seanid
        ENTONCES
            seanid      <-- FALSO
            TERMINA EL PROCEDIMIENTO
        SINO
            seanid      <-- VERDADERO
        cond            <-- ''
        apto            <-- ap_token
        n               <-- Cadena(ap_token)
        error_          <-- FALSO
        Llama a CNDCN con parámetros (cond,nt)
        SI Longitud(cond)<> 0  Y .NEGACION(error_)
        ENTONCES
```

```
                    seanid          -- FALSO
                    SI Longitud(condicion) <> 0
                    ENTONCES
                        condicion       -- condicion + 'AND ' + cond
                    SINO
                        condicion       <-- cond
                SINO
                    ap_token    <-- apto
                    seanid      <-- VERDADERO
Fin<Reconocedor de la gramática de una condición>
```

F)

NOMBRE : cocmcn

FUNCION : Dentro de la variable global "consulta", se analiza la gramática:
consulta =<comando> [<campo_bd> [<condición>]]
consulta =<comando> [<condición> [<campo_bd>]].
La variable global "error_" se apaga o se enciende, dependiendo si la gramática es o no válida.

EJEMPLO :Si la variable global "consulta" es igual a "LISTA LAS DIRECCIONES Y LOS TELEFONOS DE LAS MUJERES CASADAS", entonces cocmcn nos regresa la bandera apagada en la variable global "error_".

PROCEDIMIENTOS UTILIZADOS :

cmnd(comando,num_token)= Al invocar este procedimiento, la variable "comando" retorna una expresión que identifica al comando, si se encuentra en la variable global "consulta".

cmndbd(campo_bd,num_token)= Al invocar este procedimiento, la variable "campo_bd" retorna un campo de la base de datos, si se encuentra en la variable global "consulta".

cndcn(condicion,num_token)= Al invocar este procedimiento, la variable "condicion" retorna una expresión que identifica a la condición, si se encuentra en la variable global "con-sulta".

VARIABLES UTILIZADAS :

condicion_ = Cadena a la cual se le asigna la expresión que es válida para una condición.

num_token	=	Valor entero que contiene el número de token's a evaluar.
ap_token	=	Valor entero que indica el token sobre el cual estamos trabajando.
auxapt	=	Valor entero que nos sirve de auxiliar para guardar el apuntador del token sobre el cual estamos trabajando.
serror	=	Cadena que guarda el significado del error en caso de haberlo.
aerror	=	Cadena que guarda el token donde se encontro un error.
error_	=	Valor lógico que indica si hubo o no error.
serr	=	Variable auxiliar para guardar el valor de la variable "serror".
aerr	=	Variable auxiliar para guardar el valor de la variable "aerror".
xerr	=	Variable auxiliar para guardar el valor de la variable "error_".
campo_bd	=	Cadena que contiene la expresión de un campo de la base de datos al llamar al procedimiento cmndbd.
comando	=	Cadena que contiene la expresión de un comando de la base de datos al llamar al procedimiento cmndbd.
auxcmp	=	Cadena auxiliar que contiene la expresión de un campo de la base de datos al llamar al procedimiento cmndbd.

PSEUDO-CODIGO

```
Inicio<Reconocedor de la gramática de una consulta del tipo:
      consulta = <comando> [<campo_bd> [<condición>]]
      consulta = <comando> [<condición> [<campo_bd>]].>
    ap_token    <-- 1
    comando     <-- "
    condicion_  <-- "
    campo_bd    <-- "
    xerr        <-- FALSO
    Llama a CMND con parámetros (comando,num_token)
    SI Longitud(comando) = 0
    ENTONCES
        serror  <-- 'Se esperaba un comando'
        aerror  <-- consulta
        error_  <-- VERDADERO
        TERMINA EL PROCEDIMIENTO
    Si num_token >= ap_token
    ENTONCES
        auxapt  <-- ap_token
        Llama a CMNDBD con parámetros (campo_bd,num_token)
        SI error_
        ENTONCES
            serr        <-- serror
            aerr        <-- aerror
            xerr        <-- error_
            error_      <-- FALSO
            ap_token    <-- auxapt
        Si num_token >= ap_token
        ENTONCES
            error_      <-- FALSO
```

```
            auxapt          <-- ap_token
            Llama a CNDCN con parámetros (condicion_,num_token)
            Si error_ Y xerr
            ENTONCES
                serror      <-- serr
                aerror      <-- aerr
        Si NEGACION(error_) Y ap_token<= num_token
        ENTONCES
            auxcmp          <-- ''
            Llama a CMNDBD con parámetros (auxcmp,num_token)
            Si NEGACION(error_) Y Longitud(auxcmp)<> 0
            ENTONCES
                Si Longitud(campo_bd) = 0
                ENTONCES
                    campo_bd    <-- auxcmp
                SINO
                    campo_bd    <-- campo_bd + ',' + auxcmp
Fin<Reconocedor de la gramática de una consulta del tipo:
    consulta =<comando> [<campo_bd> [<condición>]]
    consulta =<comando> [<condición> [<campo_bd>]].>
```

G)

NOMBRE : cncocm

FUNCION :Dentro de la variable global "consulta", se analiza la gramática:
consulta =<condición><comando> [<campo_bd>].
La variable global "error_" se apaga o se enciende, dependiendo si la gramática es o no válida.

EJEMPLO :Si la variable global "consulta" es igual a "DE LAS MUJERES CASADAS LISTA LAS DIRECCIONES Y LOS TELEFONOS", entonces cncocm nos regresa la bandera apagada en la variable global "error_".

PROCEDIMIENTOS UTILIZADOS :

cmnd(comando,num_token)= Al invocar este procedimiento, la variable "comando" retorna una expresión que identifica al comando, si se encuentra en la variable global "consulta".

cmndbd(campo_bd,num_token)= Al invocar este procedimiento, la variable "campo_bd" retorna un campo de la base de datos, si se encuentra en la variable global "consulta".

cndcn(condicion,num_token)= Al invocar este procedimiento, la variable "condicion" retorna una expresión que identifica a la condición, si se encuentra en la variable global "consulta".

VARIABLES UTILIZADAS :

condicion_	=	Cadena a la cual se le asigna la expresión que es válida para una condición.
num_token	=	Valor entero que contiene el número de token's a evaluar.
ap_token	=	Valor entero que indica el token sobre el cual estamos trabajando.
serror	=	Cadena que guarda el significado del error en caso de haberlo.
aerro	=	Cadena que guarda el token donde se encontro un error.
error_	=	Valor lógico que indica si hubo o no error.
campo_bd	=	Cadena que contiene la expresión de un campo de la base de datos al llamar al procedimiento cmndbd.

PSEUDO-CODIGO

```
Inicio<Reconocedor de la gramática de una consulta del tipo:
       consulta =      <condición><comando> [<campo_bd>].>
    ap_token    <-- 1
    comando     <-- ''
    condicion_  <-- ''
    campo_bd    <-- ''
    Si num_token >= ap_token
    ENTONCES
        Llama a CNDCN con parámetros (condicion_,num_token)
        SI error_ O Longitud(condicion_) = 0
        ENTONCES
            TERMINA EL PROCEDIMIENTO
    Si num_token >= ap_token
    ENTONCES
        Llama a CMND con parámetros (comando,num_token)
    Si Longitud(comando) = 0
    ENTONCES
        serror  <-- 'Se esperaba un comando'
        aerror  <-- consulta
        error_  <-- VERDADERO
        TERMINA EL PROCEDIMIENTO
    Si num_token >= ap_token
    ENTONCES
        Llama a CMNDBD con parámetros (campo_bd,num_token)
Fin<Reconocedor de la gramática de una consulta del tipo:
    consulta =<condición><comando> [<campo_bd>].>
```

II)

NOMBRE : cmcocn

FUNCION : Dentro de la variable global "consulta", se analiza la gramática:
consulta =<campo_bd><comando> [<condición>].
La variable global "error_" se apaga o se enciende, dependiendo si la gramática es o no válida.

EJEMPLO :Si la variable global "consulta" es igual a "DE LOS TELEFONOS LISTA EL DE LAS MUJERES", entonces cncocm nos regresa la bandera apagada en la variable global "error_".

PROCEDIMIENTOS UTILIZADOS :

cmnd(comando,num_token)= Al invocar este procedimiento, la variable "comando" retorna una expresión que identifica al comando, si se encuentra en la variable global "consulta".

cmndbd(campo_bd,num_token)= Al invocar este procedimiento, la variable "campo_bd" retorna un campo de la base de datos, si se encuentra en la variable global "consulta".

cndcn(condicion,num_token)= Al invocar este procedimiento, la variable "condicion" retorna una expresión que identifica a la condición, si se encuentra en la variable global "consulta".

VARIABLES UTILIZADAS :

condicion_	=	Cadena a la cual se le asigna la expresión que es válida para una condición.
num_token	=	Valor entero que contiene el número de token's a evaluar.
ap_token	=	Valor entero que indica el token sobre el cual estamos trabajando.
serror	=	Cadena que guarda el significado del error en caso de haberlo.
aerror	=	Cadena que guarda el token donde se encontro un error.
error_	=	Valor lógico que indica si hubo o no error.
campo_bd	=	Cadena que contiene la expresión de un campo de la base de datos al llamar al procedimiento cmndbd.

PSEUDO-CODIGO

Inicio<Reconocedor de la gramática de una consulta del tipo:
consulta = <campo_bd><comando> [<condición>].>
error_ <-- FALSO

```
ap_token     <-- 1
comando      <-- ''
condicion    <-- ''
campo_bd     <-- ''
Si num_token >= ap_token
ENTONCES
    Llama a CMNDBD con parámetros (campo_bd,num_token)
    Si error_ O Longitud(campo_bd) = 0
    ENTONCES
        TERMINA EL PROCEDIMIENTO
Si num_token >= ap_token
ENTONCES
    Llama a CMND con parámetros (comando,num_token)
Si Longitud(comando) = 0
ENTONCES
    serror  <-- 'Se esperaba un comando'
    aerror  <-- consulta
    error_  <-- VERDADERO
    TERMINA EL PROCEDIMIENTO
Si num_token >= ap_token
ENTONCES
    Llama a CNDCN con parámetros (condicion_,num_token)
Fin-Reconocedor de la gramática de una consulta del tipo:
    consulta =<campo_bd><comando> [<condición>],>
```


Tema IV
Generación de
código en dBase

"A partir del momento en que nos decidamos a actuar, los obstáculos irán cayendo y nuestros sueños se irán realizando"

GUSTAVO MENDOZA SILLER

IV.1.- INTRODUCCION.

En el siguiente capitulo intentamos explicar, lo más simple y conciso posible, el concepto de "Generadores de Programas" o también llamados "Generadores de Código", incluyendo sus componentes típicos como son: el módulo de diálogo, el módulo intermedio y el de generación de código.

Es bastante simple entender el concepto de generadores de programas, sin embargo la parte más compleja es su implementación, ya que requiere que todos los componentes de nuestro consultador tengan como finalidad la generación de código, algo que no es simple. La adaptación de un módulo de diálogo con un análisis compuesto por varias técnicas, requiere de enfocar las consultas a ciertas instrucciones del lenguaje de programación en el que se busca generar la solución, de la cual resulta, dependiendo de la riqueza o variedad de éste, el código con la complejidad que permita el módulo de diálogo. Aunque el módulo de diálogo establece el contacto con el usuario, y se le piden los requerimientos y especificaciones del problema, su labor quedaría truncada sin un módulo encargado de descomponer en sus partes los requerimientos del usuario y enviar en un orden establecido al módulo generador de código, que concluye un trabajo rutinario y evite un producto único que es: "Un programa que soluciona los requerimientos del usuario según sus especificaciones".

Como el código a generar esta en lenguaje dBase, se facilita por el hecho que es un lenguaje con pocas instrucciones y es posible hacer corresponder la mayoría de ellas a ciertos verbos e incluirlas en el diccionario directamente como un significado u acción, lo cual permite que los programas no sean grandes. Sin embargo también se generan desventajas y es que los programas generados son muy específicos y estáticos y si el usuario requiere modificar el programa para variar los resultados de acuerdo a sus necesidades, el sistema no puede alterar el código establecido.

Este último capítulo establece ciertas reglas o consejos para los generadores de programas y describe rápidamente el funcionamiento de nuestro consultador generador de código.

IV.2.- GENERADORES DE CODIGO

"Los generadores de código, también llamados generadores de programas, son piezas de software que producen programas en algún lenguaje objeto, los cuales son hechos a la medida para resolver un problema específico".

La descripción anterior es bastante imprecisa porque es factible aplicarla como definición de compiladores o ensambladores.

Al analizar más detalladamente las características entre los generadores de código y los compiladores, podemos encontrar diferencias básicamente en los elementos que componen sus entradas y sus productos de salida. Para resumir observemos la siguiente tabla.

TIPO	ENTRADAS	SALIDAS
Compilador o Ensamblador	Programa Fuente	Programa Ejecutable
Generadores de Programas	Requerimientos y Especificaciones	Programas Fuente

En los compiladores o ensambladores la entrada la proporciona el programador siguiendo las reglas sintácticas del lenguaje de programación específico y en el orden lógico coloca los comandos para la obtención del resultado deseado.

Por otra parte los generadores de programas reciben la entrada de un usuario a través de un diálogo de cierta libertad o flexibilidad donde se definen los requerimientos y especificaciones con las cuales el generador arma una respuesta para obtener la solución.

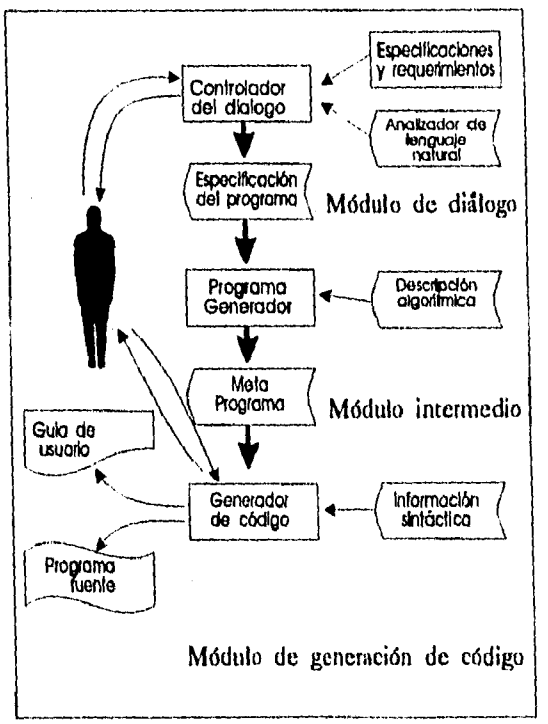
Con lo anterior podemos concretar con una definición más completa :

> **"Un generador de programas es un programa que acepta como entradas diálogos estructurados con humanos y produce, en un lenguaje de alto nivel, un programa bien estructurado y comentado que responde a los requerimientos y especificaciones del usuario."**

Un generador de programas generalmente esta integrado por tres módulos :

1) El módulo de diálogo
2) Un módulo intermedio
3) Módulo generador de código

El siguiente diagrama ilustra los componentes de cada módulo y las relaciones entre si.



El módulo de diálogo obtiene del usuario las especificaciones de la estructura del problema a resolver. Esta definición estructural es usada para que el generador de código produzca el programa fuente correspondiente a la solución del problema. Esta transformación hace uso de un módulo intermedio (una explicación más detallada se da más adelante). Al programa fuente generado se le aplica el interprete o compilador correspondiente y junto con los datos requeridos por el programa se llega a la obtención de un resultado.

Las principales ventajas que se tienen con este tipo de programas son :

- No es necesario que los usuarios tengan conocimientos profundos sobre programación, es más, el objetivo es abarcar a todo tipo de usuario sin importar su nivel de conocimientos sobre computadoras.
- Se producen programas sintácticamente correctos desde la primera vez y esto disminuye costos.
- La redundancia de código no existe, por lo cual los programas generados de esta manera tienen un tamaño que varia de unas pocas líneas a alrededor de un ciento de ellas.
- Al generar programas en un lenguaje de alto nivel, todas sus caracteristicas ventajosas pueden ser utilizadas.
- El tiempo de desarrollo de sistemas por este medio se ve decrementado dramáticamente.

Sin embargo los generadores de código no sólo son miel y dulzura, dado que están diseñados para cierta clase de aplicaciones, por lo cual se deben tomar en cuenta las siguientes recomendaciones para aprovecharlos óptimamente.

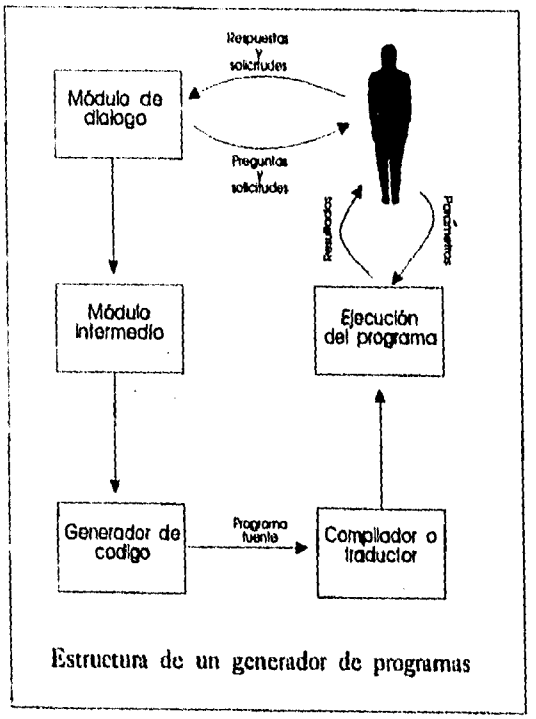
i) Para gente con experiencia insuficiente en programación.

ii) Para programadores de mayor experiencia con el fin de generar programas sintácticamente correctos y ser usados como un esqueleto de otros programas.

iii) Donde los usuarios requieren programas que puedan ser ejecutados en diferentes computadoras que tengan diferentes parámetros, como son la longitud de palabra y la capacidad de memoria, con esto el problema se define sólo una vez.

iv) Cuando el usuario requiera más de una versión de un programa en diferentes lenguajes o dialectos para que como en (iii), el usuario pueda obtener las diferentes codificaciones de un programa de acuerdo a las especificaciones de su problema.

En este diagrama está ilustrada la estructura típica de un generador de programas.



Estructura de un generador de programas

El módulo de diálogo interactúa con el usuario para obtener una completa y consistente especificación del problema con la cual se formará el programa fuente. En este módulo todas las respuestas del usuario son analizadas por el controlador de diálogo, la complejidad podrá variar y muchos diálogos por

ejemplo requerirán del usuario respuestas concretas o la elección de una opción en un menú o la simple respuesta de un si o un no, algunos requieren entradas de expresiones complejas como son las ecuaciones aritméticas o el lenguaje natural. El grado en el cual un generador de programas es usado dependerá del éxito del diseño del módulo de diálogo.

Cuando el diálogo ha sido terminado, las especificaciones del problema también están completas, por lo tanto se requiere almacenar estas especificaciones del programa en un archivo.

El módulo intermedio es una macro-expansión donde las especificaciones del programa se expanden al incluirse cualquier algoritmo al programa fuente. El metaprograma resultante de esta unidad estará en algún metalenguaje el cual en realidad será en algún lenguaje existente de alto nivel.

El metaprograma deberá ser fácil de leer por humanos y deberá de escribir consistentemente las operaciones a ser ejecutadas por el programa. Otra función del módulo intermedio es el de simplificar y optimizar el metaprograma.

El módulo generador de código toma las especificaciones del metaprograma y produce dos archivos de salida, el primero es el programa fuente requerido para solucionar el problema y el segundo es la documentación para asistir el usuario a entender y ejecutar el programa.

Una deficiencia de muchos generadores de programas es la imposibilidad de almacenar las especificaciones de los problemas y además son especialmente limitados a problemas que envuelven cierto tipo de especificaciones por lo tanto alguna clase de edición del programa es requerido, la cual deberá ser hecha por un programador con conocimientos en el lenguaje donde fue generado el programa fuente.

IV.3.- OBSERVACIONES SOBRE EL CODIGO GENERADO EN dBASE.

Finalmente estamos en el último tema de esta tesis y realmente nos es muy grato llegar aquí, y explicar que el código, producto final de un generador de programas, es tan sólo un puñado de instrucciones. Esto es natural, dado que la consulta del usuario consta de una o dos lineas de lenguaje natural y al ser analizada y sintetizada, produce una consulta que generalmente se resuelve con una sola linea de código por corresponder cada verbo a una instrucción en dbase, el cual es complementada con la(s) condición(es) y campos que desea el usuario establecer.

En la siguiente tabla listamos cuales son las instrucciones de dBase que hemos asociado con un verbo en lenguaje natural.

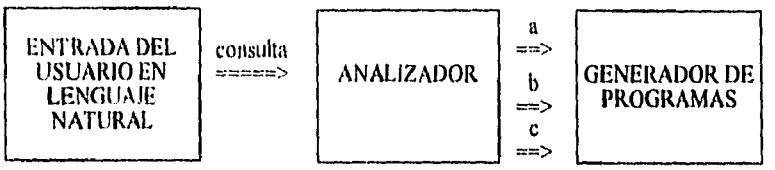
COMANDO EN dBASE	COMANDO EN LENGUAJE NATURAL
AGREGA	APPEND BLANK
ALGUN	NEXT 5
BUSCA	LIST OFF
EDITA	EDIT
TODO	ALL
LISTA	LIST

Además toda consulta debe seguir alguna de las 9 diferentes reglas sintácticas descritas en el tema III.3 (Descripción general de consultador) dentro de las cuales en todas ellas es necesario expresar la consulta por medio de un comando

en lenguaje natural que nos lleva a su comando de dbase correspondiente de acuerdo a la tabla anterior.

El consultador esta dividido en tres partes principales, la primera es donde el usuario proporciona al consultador la frase u oración con la cual realiza su consulta en lenguaje natural, la segunda es donde se descompone y analiza cada palabra de la frase u oración para determinar el comando que le corresponde en dBase y las condiciones y campos que son afectados, finalmente como tercera parte es generar el código en dBase que satisface la consulta del usuario.

Lo anterior se ilustra a través s del siguiente esquema:



Componentes

a: comando en dBase
b: condición (es) (puede ser nula)
c: campo (s) de la base de datos (puede ser nulo)

En la "Entrada del usuario en lenguaje natural" la computadora permite al usuario expresar la consulta en las formas que se desee, este módulo pasa al analizador la consulta tal cual el usuario la proporcionó, el "Analizador" podrá tener 2 estados, estado de error o estado de éxito.

En el estado de error el analizador señala al usuario la sección que no es reconocida satisfactoriamente y devuelve al usuario el control y le permite editar y corregir su consulta.

En el estado de éxito el analizador genera como salida 3 componentes, el comando, la condición o condiciones y los campos de la base de datos a manipular. Estos componentes son pasados al generador de programas.

El "Generador de programas" recibe los componentes y de acuerdo al comando enviado, retomando información sintáctica del comando, genera como salida el programa fuente que satisface los requerimientos del usuario.

Por lo tanto los programas generados son, en esencia un pequeño grupo de instrucciones en lenguaje dBase, básicamente las instrucciones append, list y edit, con variantes de acuerdo a los campos requeridos por el usuario y la condición especificada.

Un ejemplo de este proceso lo podemos detallar con la siguiente consulta en lenguaje natural.

"Lista todos los nombres y teléfonos de los hombres solteros"

El usuario puede expresar su consulta de esa manera, sin embargo sabemos que no siempre pensamos en un orden sintáctico rígido, si no más bien en un orden variable pero que expresa la misma idea, por lo que la anterior consulta puede tener las siguientes variables.

"De los hombres solteros listame su nombre y teléfono".

Otra más:

"Lista los solteros con nombre y teléfono".

O quizá:

"Lista nombre y teléfono de los solteros".

Ahora bien, con cualquiera de las 4 anteriores expresiones el analizador debe obtener:

a) Comando en dBase : List
b) Condiciones : Sexo=.M. .AND. Edo. Civil="S"
c) Campos : nombre, dirección.

Estos 3 componentes son pasados al generador de programas para que armen la estructura adecuada del programa.

El código generado es independiente del consultador y puede ser ejecutado por el usuario las veces que éste desee y en el momento que lo requiera, además si el usuario cuenta con un poco de entusiasmo y deseos de aprender quizás intente modificar el código para obtener alguna variante personal del programa, en fin, el generador de código no es la solución para todo problema pero es una herramienta de Ingeniería de Software que puede y será más utilizada en el futuro próximo.

"Una conclusión es el lugar donde llegaste cansado de pensar"

ANÓNIMO

CONCLUSIONES:

Intentar concluir en tan solo unas páginas el resultado de varios años de estudio y trabajo nos resulta verdaderamente difícil, la razón es que creemos existen varios puntos fundamentales a considerar. Sin embargo haremos mención de nuestros principales resultados.

Primeramente, como un área de la computación se estudió la Inteligencia Artificial (IA), conociendo los orígenes e intentando establecer una definición de la misma; pero con la explicación del nacimiento del concepto de "Inteligencia Artificial" acuñado por Marvin Minsky en la década de los 40's y las posteriores definiciones hechas por Alan Turing, Samuel, Elaine Rich, etc. y la oposición con aseveraciones contrarias a su existencia al catalogarla como una "Simulación mecánica de la racionalidad" nos lleva a concluir que :

- **Definir la Inteligencia Artificial en términos totalmente aceptados es prácticamente imposible.**

El Procesamiento del Lenguaje Natural (PLN) representa un gran reto en lenguaje inglés, ahora en lenguaje español resulta más complicado ya que nuestra lengua es sumamente más rica en gramática. Por tal razón la interfase se aplica únicamente a la interpretación sintáctica y semántica de consultas que se hagan a una base de datos particularmente restringida a un tema dado, sin adentrarnos a lo más ambicioso que es la comprensión del lenguaje natural.

- **La complejidad en el PLN esta en función de la riqueza de la lengua a procesar y el español representa un mayor reto en este sentido que el inglés.**

No obstante de la incredulidad de algunos estudiosos en la rama de la IA referente a la verdadera comunicación hombre-máquina en LN, nosotros afirmamos que:

- **La comunicación restringida entre el hombre y la máquina es algo que ya se ha logrado, sin embargo aún falta mucho por hacer para establecer una comunicación coherente e hilvanada totalmente real como la propuesta por Turing[RIC83].**

Las reglas sintácticas y semanticas están restringidas por razones obvias que se fundamentan con la riqueza en gramatica de nuestra lengua española. Sin embargo se presento las principales reglas que dan orden y significado al consultador en LN. En cuanto a las limitantes para el PLN, evitamos el análisis de ondas sonoras como son principalmente:

a) Entonación
b) Tiempos
c) Dicción
d) Enfasis

Por lo tanto aseveramos que :

- **El PLN hablado es aún más complejo que el escrito y representa, en mucho, una barrera práctica para su instrumentación en la actualidad.**

El PLN únicamente con análisis en el lenguaje escrito también posee una gran cantidad de dificultades, destacando la ambigüedad que representa la interpretación y representación del conocimiento. Derivado de esto, se presentaron de acuerdo con Terry Winograd[WIN84] los 5 tipos de ambigüedades existentes los cuales son:

a) Ambigüedad Léxica
b) Ambigüedad Estructural
c) Ambigüedad Estructural Profunda
d) Ambigüedad Semántica
e) Ambigüedad Pragmática

Así mismo al analizar con mayor detalle los trabajos sobre PLN y las técnicas más usuales desarrolladas desde los 60's hasta la fecha creemos que :

- **Por las características principales de cada una de las diferentes técnicas conocidas para el PLN, la solución más apropiada se encontrará en una combinación adecuada de éstas, aprovechando sus cualidades y complementando sus puntos débiles con la mezcla de otras.**

La inteligencia artificial es sin duda el campo que dará la sensación de humanizar las frías computadoras, y en este sentido podemos afirmar que :

- **El uso de la Inteligencia artificial continuará presentándose cada día con mayor solidez en las aplicaciones informáticas y serán imprescindibles en un futuro cercano las interfases en un lenguaje mas común para los seres humanos con el fin de comunicarnos con los futuros equipos hardware-software más complejos pero más fáciles de usar.**

En resumen este trabajo aporta una investigacion seria con una visión optimista de las conquistas del ser humano por crear la herramienta más poderosa y más parecida a si mismo. Sin embargo se aportan tambien los grandes problemas a enfrentar, que de hecho nosotros nos enfrentamos solo a algunos de ellos, y las posibles alternativas de solución para proveer a nuestras aplicaciones de una interfaz en lenguaje natural con una aplicacion práctica de generación de programas.

Con lo anterior buscamos colaborar en un futuro en el cual se cumpla nuestra ultima conclusión :

- **Los usuarios de equipos de cómputo muy pronto olvidarán los lenguajes de programación tal como los conocemos hoy para pasar a un estado de mayor evolución donde su forma de "programar" será a través de ordenes en lenguaje natural restringido e interfases cada vez más inteligentes, flexibles y poderosas.**

BIBLIOGRAFIA

|COR85| Corro León, Javier.
EL HORIZONTE DE LA INTELIGENCIA ARTIFICIAL.
Información científica y tecnológica.
Vol. 7 No. 109. pp. 15-17.
México, 1985.

|FL085| Flores Romero, Juan José.
INTRODUCCION AL LENGUAJE PROLOG Y ALGUNAS APLICACIONES.
Informe técnico. CINVESTAV.
No. 27. PP. 59.
México 1985.

|HAJ87| Hajicova, Eva.
Curso Internacional de Sistemas Expertos
Departamento de Ingeniería Eléctrica.
Universidad Técnica Checa - U.N.E.S.C.O.
México 1987

|HAR84| Hartnell, Tim.
INTELIGENCIA ARTIFICIAL : Conceptos Programas y Aplicaciones.
Ed. Anaya y Multimedia.
México 1984

|JIM87| Jiménez Salazar. Hector.
CLAUSULAS PARA EL ANALISIS DE ORACIONES.
CINVESTAV IPN. Sección de computación.
México 1987. 2o pp.

|LAW84| Larence, G.Tesler.
LENGUAJES DE PROGRAMACION.
Investigación y Ciencia.
No. 98 pp. 36-48.
España 1984.

|LEN84| Lenat, Douglas B.
PROGRAMACION DE SISTEMAS INTELIGENTES.
Investigación y Ciencia.
No. 98.
España 1984.

|LLO87| Lloréns Murillo Juan.
ENCICLOPEDIA DE LA INFORMATICA.
Inteligencia Artificial y Sistemas Expertos.
Tomo 28
España 1987.

|LUK86| Luker, P.A. & Burns, A.
PROGRAM GENERATORS AND GENERATION SOFTWARE.
The computer journal.
Vol. 29. No. 4 . pp. 315-321.
Inglaterra 1986.

|RIC83| Rich, Elain.
ARTIFICIAL INTELIGENCE.
Mc. Graw Hill International student edition.
U.S.A. 1983. 486 pp

|SCH87| Schildt, Herbert.
ARTIFICIAL INTELLIGENT USING C
Ed. Mc Graw Hill.
U.S.A. 1987.

|WEI83| Weizembaum, Joseph.
ELIZA : a computer program for the study of natural
lenguage.
Comunications of the ACM.
Vol. 26. No. 1. pp. 23-28.
U.S.A. 1983.

|WIN84| Winograd, Terry.
PROGRAMACION Y TRATAMIENTO DE LENGUAJES.
Investigación y Ciencia.
No.98. pp. 70-85.
España 1984.

ANEXO I

La siguiente lista muestra las principales interfases en LN para microcomputadoras IBM PC y compatibles:

X; PLUS American Expertech Inc.
IBM PC,XT,AT,PS/2 y compatibles
512 K en RAM, $1250.
Interfaz para dBase III, Lotus 123, Supercalc y más.

NLQ (Natural Language Query) Battelle
IBM PC,XT,AT y compatibles
512 K en RAM, $995
Interfaz en LN para oracle, DB2 y Manejadores de bases de datos relacionales.

EQL (English Query Language) Information Builders INC.
IBM PC,XT,AT y PS/2 con PC/FOCUS 3.0
640 K en RAM, $249
Interfaz para PC/FOCUS (con acceso a dBase III).

HAL (Human Access Langunge) Lotus Development Corp.
IBM PC,XT,AT compac 386 AT&T 6300 y compatibles con Lotus 123
512 K en RAM, $150
Interfaz en LN para Lotus 1-2-3.

GURU Micro Data Base Systems Inc.
IBM PC,XT,AT,RT y compatibles
640 K, $650
Sistema experto con base de datos, hoja de cálculo, lenguaje de programación, procesador de palabras, gráficas, telecomunicaciones e interfaz en LN.

K-Chat Micro Data Base Systems Inc.
IBM PC,XT,AT y compatibles con Knowledge Man/2
512 K, $295
Interfaz en LN para Knowledge Man/2.

IA (Intelligent Asistan) Symntec
IBM PC,XT,AT y compatibles con Q&A
512 K, $349
Interfaz en LN para Q&A

Metamorph Thunderstone Corp.
IBM PC,XT,AT y compatibles
256 K, $5000
Análisis de texto en LN para recuperación y correlación, usado para investigación y estrategia en el análisis de información.