2 ej.
1 8

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

# CORRECTOR ORTOGRAFICO POR MEDIO DE LA UTILIZACION DE TECNICAS DE INTELIGENCIA ARTIFICIAL

T E S I S

Que Para Obtener el Título de:
INGENIERO EN COMPUTACION
P r e s e n t a :
MARCO CESAR GLORIA FLORES

Dirigida por el M. en C. Efraín Pardo Ortíz

México, D.F. 1989


**UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso**

# INDICE

CAPITULO III: Representación del Conocimiento

CAPITULO IV: Lenguaje Natural



CAPITULO V: Técnicas de Reconocimiento de Fallas y Sistemas Expertos

## CAPITULO VI: ORTHOS

Cápitulo VII: Conclusiones

# INTRODUCCION.

El auge que en los últimos dos años han tenido las microcomputadoras, ha propiciado que el mercado de software experimente un increíble crecimiento en cuanto a sistemas de todos tipos.

También debido a este gran auge, los usuarios finales, se han vuelto más exigentes con los productos que adquieren, lo cual obliga a la gente de desarrollo de software a utilizar técnicas más sofisticadas en sus productos.

A medida de que los conocimientos en el campo de la computación han ido evolucionando, un campo en especial ha tenido un gran auge, este es el de la Inteligencia Artificial.

Esto se debe a que la mayoría de los diseñadores de computadoras, pretenden que sus equipos sean fácilmente operables. Y no existe máquina más operable que aquella que pueda entender lenguaje natural y que a su vez se pueda expresar en lenguaje natural.

De esta manera la Inteligencia Artificial es un conjunto de técnicas, que pretenden que las máquinas emulen el funcionamiento de la mente humana.

Por otro lado, dentro del medio industrial existe una idea o concepto, que día a día ha ido cobrando fuerza, esto es la Automatización de la Oficina.

La automatización de Oficina se puede entender como un conjunto de ideas, las cuales pretenden ofrecer, por medio de diferentes máquinas, un mejor aprovechamiento de los recursos con los que cuentan, que van desde el uso adecuado de máquinas tan simples como las calculadoras hasta máquinas complejas como las computadoras.

El presente trabajo, es el desarrollo de un sistema de software, el cual queda ubicado dentro de el grupo de software de microcomputadoras para la automatización de la oficina.

Este sistema se integra como un complemento a los sistemas ya desarrollados para proceso de palabras. Y su objetivo es hacer un análisis completo de la ortografía de los documentos desarrollados en otros procesadores de palabras comerciales.

El presente trabajo se encuentra organizado de la siguiente manera:

*Parte uno, Principios Básicos de la Inteligencia Artificial:*

- Que es la Inteligencia Artificial.
- Principales Herramientas de la Inteligencia Artificial.
- Representación del conocimiento.

*Parte dos, Conceptos para un Sistema Experto Ortográfico:*

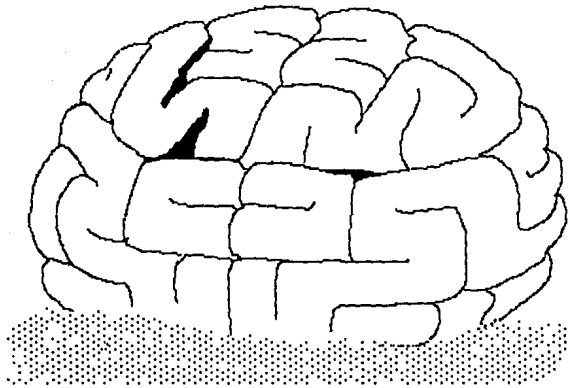
- Lenguaje Natural.
- Sistema Experto para el análisis de problemas gramaticales.
- Análisis de editores comerciales y formas de almacenamiento.
- Análisis de estructuras de datos necesarias para la implementación eficiente de las búsquedas.
- Análisis de las estructuras de datos necesarias para la implementación del diccionarios de datos.

*Parte tres, Implementación de un Sistema Experto Ortográfico:*

- Análisis del sistema por medio de la metodología de Yourdon.
- Diseño del sistema.
- Implementación del sistema.

# PARTE UNO

## INTELIGENCIA ARTIFICIAL



# CAPITULO I

# ¿QUE ES LA INTELIGENCIA ARTIFICIAL?

## 1.0 Introducción.

En el presente capítulo, se dan las bases de lo que es la Inteligencia Artificial, a la cual en lo sucesivo se le denominará a través de las siglas IA. Comenzaremos por dar una definición de lo que este concepto significa, para continuar con una breve historia de su desarrollo. Posteriormente se definirán sus bases para concluir con una explicación del porqué de la IA dentro de la computación.

## 1.1 Definición.

El definir que es exactamenta la IA, no es precisamente una tarea sencilla, comenzando porque la palabra "Inteligencia" por sí sola, abarca un sin número de conceptos que hasta la fecha no han podido ser correctamente delimitados.

Sin embargo, después de consultar a varios autores, elegimos la que a nuestro parecer define a esta área, de la manera más sencilla y práctica, de modo que la IA *es el estudio de las facultades mentales a tráves del uso de modelos computacionales,* (Mc. Dermoth, 1986).

Los principales ámbitos del conocimiento, que han sido estudiados bajo el campo de la IA, son:

- Teoría de juegos,
- Solución de teoremas,
- Percepción:
  - + Visual,
  - + Auditiva,
- Entendimiento de lenguaje natural,
- Sistema expertos:
  - + Matemáticas simbólicas,
  - + Diagnósticos médicos,
  - + Análisis químicos,
  - + Diseño de Ingeniería:
    - * CAD/CAM,
    - * Comunicaciones.

## 1.2 Breve historia.

Las primeras incursiones en la IA, fueron realizadas por Samuel y Newell. El primero desarrollo un sistema para jugar ajedrez, con la particularidad de que este sistema no solo jugaba en base a un algoritmo predefinido, sino que usaba las experiencias de un juego para mejorar su estrategia de las siguientes partidas.

Newell por su parte desarrollo un sistema el cual realizaba las demostraciones del primer capítulo del libro "Principia", (Whitehead, 1950).

Posteriormente Newell, Shaw y Simon, se enfocaron a tratar de solucionar problemas comunes de la vida diaria, como el de elegir el mejor camino para dirigirse al trabajo por las mañanas. De este trabajo surgió GPS, "Sistema de resolución de problemas generales".

Al hacer un análisis sobre el funcionamiento de estos sistemas, se puede observar, que no envuelven una gran cantidad de conocimiento del mundo al cual tratan de modelar.

Esta limitante, la imponían las deficientes técnicas con las que se contaban en esos momentos.

Posteriormente con la evolución en la investigación en IA, las técnicas para el manejo de cantidades masivas de conocimiento y los avances en computación, permitieron el inicio de investigaciones en campos más sofisticados del conocimiento humano como la percepción, visual y auditiva, el entendimiento de lenguaje natural escrito y solución de problemas de dominio específico.

Como ejemplo de las investigaciones en estos campos se tienen:

Percepción auditiva: Reddy, 1976; Walker, 1976; Lea, 1980.
Percepción visual: Winston, 1975; Brady, 1981; Marr, 1982.
Lenguaje natural: Feigenbaum, 1963; Minsky, 1968; Boden, 1977; Schank, 1980; Winograd, 1980.
Sistemas expertos: Feigenbaum, 1977; stefik, 1982; Sridharam, 1978; Shotliffe, 1979; kulikowsky, 1980.

## 1.3 Bases de la IA.

Las bases de la IA fueron dadas por Newell y Simon (Rich, 1983), en lo que ellos llamaron "Hipótesis del Sistema de Símbolos Físicos", a dicho sistemas lo definieron de la siguiente manera:

> *Un sistema de Símbolos Físicos, consiste en un conjunto de entidades llamadas símbolos, los cuales son patrones físicos que pueden ocurrir como componentes de otro tipo de entidad llamado expresión o estructura del símbolo. Entonces una*

*estructura del símbolo esta compuesta por un número de instancias o símbolos relacionados de alguna manera. En cualquier instante del tiempo el sistema puede contener una colección de estas estructuras. Entre estas estructuras, el sistema también contiene una colección de procesos que operan sobre una expresión para producir otra expresión: procesos de creación, modificación, reproducción y destrucción. De tal forma que un sistema de símbolos físicos es una máquina que produce procesos a través del tiempo y envuelve una colección de estructuras de símbolos. De modo que el sistema existe en un mundo de objetos más que sobre las expresiones simbolicas por si mismas.*

Dada la anterior definición, se plantea la siguiente hipótesis por Newell.

Hipótesis del Sistema de Símbolos Físicos:

> *Un sistema de Símbolos Físicos, cuenta con el significado necesario y suficiente para tomar acciones generales inteligentes.*

Esta hipótesis ha sido ampliamente discutida, sin embargo su importancia radica en su aproximación a la naturaleza de la inteligencia humana ya que ha creado la posibilidad de creer que es posible construir sistemas que puedan ejecutar acciones inteligentes.

## 1.4 IA y Computación.

Para finalizar con este capítulo, trataremos de contestar una pregunta básica de la IA, esta es porque tratar de simular Inteligencia en las máquinas.

La respuesta viene dada en dos sentidos.

El primero de ellos radica en el hecho de que el entender e implementar procesos inteligentes en una computadora, es un medio para estudiar y comprender la inteligencia en general.

Los principales promotores de la IA opinan que la experimentación de procesos inteligentes, se convierte en una poderosa herramienta de investigación la cual se agrega a métodos tradicionales de ciencias como la psicología, filosofía y lingüística.

Los fundamentos de este razonamiento, se basan en que la computadora es una herramienta ideal para el modelado e investigación de diversos procesos. Ejemplificando, se puede modelar una parte de algún tipo específico de conocimiento y agregarle y quitarle piezas, en orden de poder determinar la importancia de estas piezas.

Un resultado de este nuevo punto de vista pueden ser nuevas ideas de como ayudar a las personas a ser más "inteligentes". Solo el completo conocimiento a cerca de los procesos de información humana, puede ayudar a "hacer" computadoras inteligentes, y las teorías derivadas puramente de las máquinas, a menudo sugieren posibilidades de como mejorar los procesos de aprendizaje en los humanos. En otras palabras el uso de metodologías para construir máquinas inteligentes, podría ser transferida a los procesos de educación humanos para lograr personas más inteligentes.

Por otro lado, la creciente complejidad del mundo en el que vivimos, ha obligado a los profesionales de sistemas a ir investigando nuevas tecnologías. Dado que el punto esencial en cualquier desarrollo es su capacidad de adaptación a cambios en el medio ambiente, las técnicas de IA, proveen ideas y metodologías, la combinación de estas herramientas con las tradicionales herramientas de desarrollo de sistemas, aparecen como la solución más viable para aquellos sistemas que quieran sobrevivir más allá de los cambios en el medio ambiente en el cual van a operar.

# CAPITULO II

# TECNICAS DE LA INTELIGENCIA ARTIFICIAL

## 2.0 Introducción.

Debido a la amplia variedad de problemas que abarca la IA y su complejidad, resulta difícil imaginar técnicas para su resolución.

Sin embargo existen una serie de técnicas que pueden ser modeladas de distinas maneras, de tal forma que se adapten a la situación a resolver.

En el presente capítulo, se estudiarán las técnicas más difundidas, se analizarán sus bases y metodologías.

Así mismo se estudiará la manera de modelar una situación bajo la metodologías de la IA.

## 2.1 IA y el conocimiento.

Uno de los principales resultados arrojados de las primeras investigaciones en IA, es que *la inteligencia requiere forzozamente conocimiento.*

Y al estudiar las formas de como poder representar conocimiento en una máquina, los investigadores encontrarón que el conocimiento lleva consigo ciertas características inherentes, estas son:

- Es voluminoso,
- Es difícil de caracterizar,
- Sufre de constantes cambios.

De manera que cualquier forma de implementar estructuras sobre el conocimiento debera de cumplir ciertas características, en orden de poder ser usadas por sistemas de IA, estas son:

1) Poder agrupar datos que compartan propiedades importantes.

En términos de poder reducir el espacio necesario para almacenar los datos que conforman el conocimiento, datos que compartan propiedades importantes deben de ser agrupados.

2) Los datos que conformen el conocimiento, deben de estar organizados de tal forma que el usuario final los pueda entender y modificar por sí mismo.

Sencillamente, en los sistemas de IA, la retroalimentación, es la base de la flexibilidad de los sistemas. Y dado que estos dependen de los datos almacenados en el conocimiento y que este se encuentra en constante cambio, el diseño para representarlo, debera de considerar el poder hacer cambios sobre los datos que lo integran.

3) El diseño debe de ser flexible para poder reflejar cambios en su medio ambiente.

La flexibilidad es una de las piedras angulares en cualquier sistema de IA, tanto para corregir errores como para reflejar cambios en su medio ambiente de operación.

4) El conocimiento deberá estar estructurado de tal forma que pueda ser usado en una gran variedad de situaciones.

Las maneras de representación del conocimiento, dada la gran diversidad de situaciones que deben de enfrentar, deben contar forzosamente con una gran flexibilidad, además deberán de estar fundamentadas sobre razonamientos suficientemente claros y generales.

5) Deberán ser diseñadas de tal forma que ayuden por sí mismas a reducir las búsquedas sobre la base de datos.

Las bases de datos sobre conocimiento, invariablemente contienen cantidades masivas de datos de tal forma que los diseños, deben de ayudar a reducir los rangos de posibilidades en las búsquedas.

En base a lo anterior, se puede decir que, aunque cualquier técnica desarrollada debe de tomar en cuenta que las estructuras del conocimiento deben de poseer las características anteriores, el depender del conocimiento, las provee de cierta independencia con respecto a los distintos problemas a enfrentar.

## 2.1.2 Sistemas en base a estados.

Cualquier metodología para la resolución de problemas, contempla generalmente a grandes rasgos tres pasos, estos son:

A) La primera tarea a realizar, consiste en definir de una manera clara y precisa cual es el problema a resolver. Esta definición debe de incluir, cuales son los estados inicial y final. Para este último, se debe de indicar los rangos de soluciones aceptables.

B) El segundo paso consiste en realizar un análisis detallado del problema. Este paso es de vital importacía ya que de las características esenciales del problema, se derivarán las técnicas a ser usadas.

C) El tercer paso será la aplicación de las técnicas elegidas y la resolución del problema.

Cuando se piensa solucionar un problema en base al enfoque de IA, el primer paso consisten en modelar el problema en base a estados de transición.

Esta metodología consiste en que dado un estado inicial, un estado final y un conjunto de estados intermedios, en base al conocimiento que el sistema tenga del problema, trazará una trayectoria del estado inicial al estado final.

Para ejemplificar, usaremos el siguiente caso:

*Se tienen dos jarras, la primera de ellas de 4 lt. y la segunda de 3 lt., ninguna de ellas se encuentra marcada, de manera que no se puede cuantificar su contenido a menos que las jarras esten llenas. Se desea que la jarra de 4 lt. contenga exactamente 2 lt. de agua.*

| *Capacidad total* | *Contenido inicial* | *Contenido final* |
|---|---|---|
| *4 lt.* | *V** | *2 lt.* |
| *3 lt.* | *V* | *V* |

*Para facilitar,el ejemplo, denotaremos a la jarra de 4 lt. como "A" y a la de 3 lt. como "B".*

*Nuestros estados intermedios, estaran definidos de acuerdo a las siguientes reglas de transición entre estados.*

*V Este simbolo implica que no importa el contenido.

| Regla | Jarra | Contenido al inicio de la transición | Contenido al final de la transición | Descripción de la acción |
|---|---|---|---|---|
| 1 | A<br>B | ∀<br>∀ | 4 lt.<br>∀ | Llenar la jarra A |
| 2 | A<br>B | ∀<br>∀ | ∀<br>3 lt. | Llenar la jarra B |
| 3 | A<br>B | ∀<br>∀ | 0 lt.<br>∀ | Tirar toda el agua de la jarra A |
| 4 | A<br>B | ∀<br>∀ | ∀<br>0 lt. | Tirar toda el agua de la jarra B |
| 5 | A<br>B | X<br>Y | 4 lt.<br>Y - X | Llenar la jarra A con la B |
| 6 | A<br>B | X<br>Y | X - Y<br>3 lt. | Llenar la jarra B con la A |
| 7 | A<br>B | X<br>Y | X + Y<br>0 lt. | Vacía la jarra B en la A |
| | A<br>B | X<br>Y | | Vacía la jarra |

*Luego, dado el estado inicial y pasando a través de los distintos estados de transición, deberemos de llegar al estado final que nosotros hemos definido.*

*Una muestra de un trayectoria de solución se da a continuación.*

| Litros en la jarra "A" | Litros en la jarra "B" | Regla aplicada |
|---|---|---|
| 0 | 0 | 2 |
| 0 | 3 | 7 |
| 3 | 0 | 2 |
| 3 | 3 | 5 |
| 4 | 2 | 3 |
| 0 | 2 | 7 |
| 2 | 0 | |

Resumiendo, dentro de la definición de cualquier problema, bajo el esquema de IA las primeras cuatro acciones a realizar son:

- Definir todos los estados relevantes para plantearlos en función de un espacio de estados del sistema.

- Seleccionar uno o más estados de el sistema, los cuales serán el punto inicial para la solución del problema.

- Seleccionar uno o más estados del sistema, los cuales cumplan con las condiciones de aceptabilidad para ser considerados metas a alcanzar.

- Definir las reglas que de acuerdo a las condiciones del problema estipulen la transitividad entre los estados.

### 2.1.3 Estrategias de control.

Una vez que se han realizado los puntos anteriores, queda otro punto importante por definir, este es, dado que puede haber varias reglas que se puedan aplicar a un mismo tiempo, como decidir cual de ellas debe de ser aplicada.

Cualquier estrategia que intente controlar esta situación, debera cubrir satisfactoriamente las siguientes características:

a) Debe de causar movimiento a través de los estados tratando de llegar siempre al estado meta.

b) Debe de ser sistemática.

Para facilitar la conceptualización del problema, a partir de la definición, el problema se puede representar por medio de árboles. De modo que el estado inicial sea la raíz, a partir de la raíz, las transiciones entre estados representaran las ramificaciones del árbol, y el estado meta sea parte del conjunto de hojas del árbol.

De forma que las estrategias de control en su forma más primitiva, pueden ser generadas en base a metodologías de búsquedas y selección sobre un árbol definido.

## 2.2 Técnicas primitivas de busqueda y selección.

Primero describiremos las técnicas de busqueda y selección primitivas, para que sobre estas bases posteriormente se puedan analizar técnicas con un mayor grado de complejidad.

Para ejemplificar las metodologías, nos basaremos en el siguiente ejemplo.

Supongamos que después de análizar algun problema, nosotros llegamos a modelarlo a través de una red de estados, en orden de asignarle prioridades a las posibles transiciones, nosotros le asignamos un costo a cada ruta posible de un estado a otro, tal como la muestra la figura 2-1.

Partiendo de esta red, nosotros construimos el árbol mostrado en la figura 2-2.

Y es necesario llegar del estado "S" al estado "F" al menor costo posible.

## 2.2.1 Metodo "Depth First".

Este método consta de los siguientes pasos:

1) Posicionarse en el estado inicial.
2) Generar los estados de transición (si aplica) y viajar al estado de más a la izquierda que no ha sido investigado.
3) Verificar si es el estado es hoja de el árbol, si no es ir al paso 2.

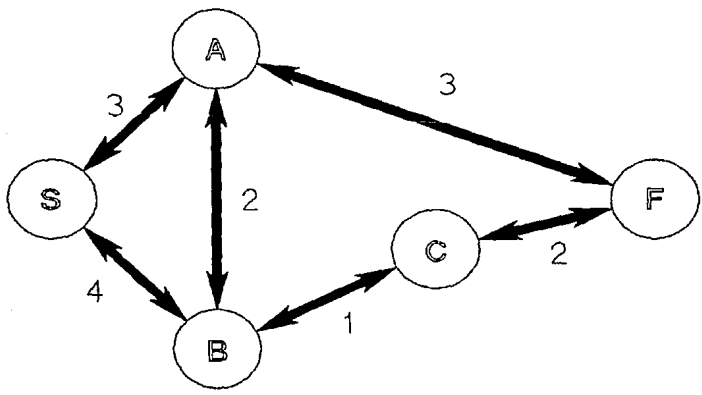



*Figura 2-1.*

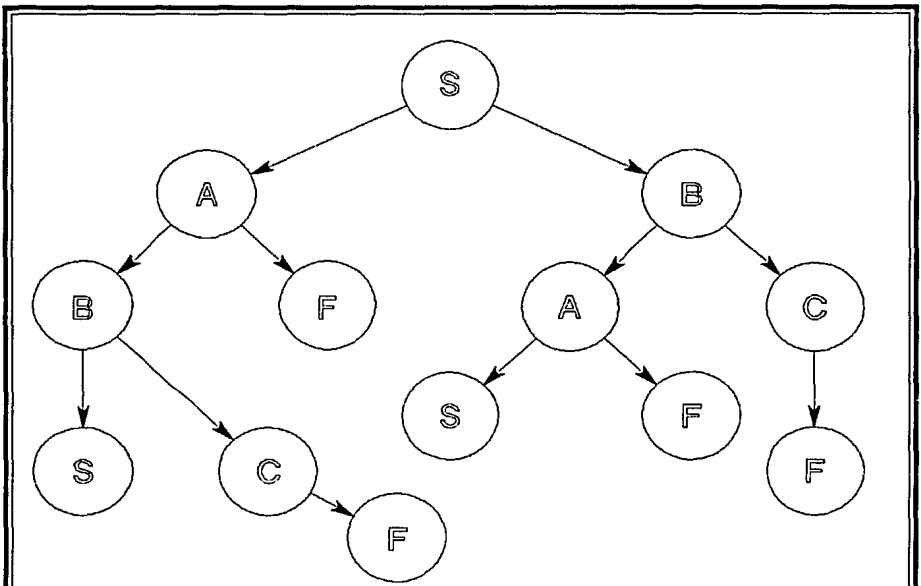



*Figura 2-2.*

4) En caso de que el estado sea hoja del árbol, verificar si el estado en que nos encontramos es el nodo meta, en caso de ser asi terminal la búsqueda.
5) Si no es asi, regresar al estado predecesor.
6) Ir al paso 2.

Tomando el árbol del ejemplo, la trayectoria que seguiría la búsqueda, es mostrada en la figura 2-3.

Notese que para que este método funcione adecuadamente, los estados fueron acomodados de tal forma que las trayectorias van de izquierda a derecha en función de su costo.

Sin embargo, este método no garantiza que la ruta encontrada sea la mejor.

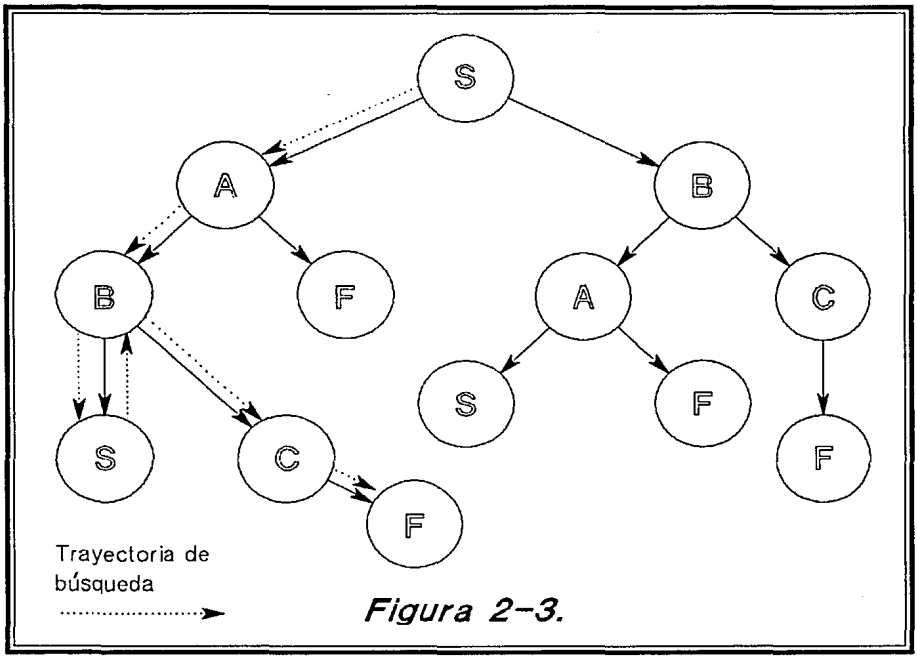



*Figura 2-3.*

## 2.2.2 Metodo "Breadth First".

En orden de tratar de disminuir el número de comparaciones a efectuar para encontrar el estado meta, en este método se encuentra una importante variación sobre el anterior.

El algoritmo para ir del estado inicial al final, es el siguiente:

1) Posicionarse en el estado inicial.
2) General los estados sucesores ( si aplica ).
3) Visitar todos los estados sucesores de izquierda a derecha, si alguno es el estado meta ir al paso 5.
4) Viajar al estado sucesor de más a la izquierda e ir al paso 2.
5) Terminar el proceso.

La ruta que seguiría este método, para el problema planteado se muestra en la figura 2-4.

Este método reduce el número de comparaciones a efectuar para llegar al estado meta, comparándolo con el método anterior, sin embargo, no garantiza tampoco que la ruta elegida sea la mejor.

### 2.2.3 Metodo "Hill Climbing".

Cuando se realizan búsquedas, la eficiencia puede ser mejorada notablemente si los estados más promisorios, son analizados primero.

Esta metodología plantea una forma de lograrlo si se siguen los pasos mostrados a continuación.

1) Posicionarse en el estado inicial.
2) Generar los estados posteriores ( si aplica).
3) Si todos los estados posteriores han sido examinados, viajar al estado predecesor e ir al paso 3.
4) Viajar al mejor de los nodos sucesores no examinado.
5) Verificar si es el estado meta, en caso de no ser asi ir al paso 2.

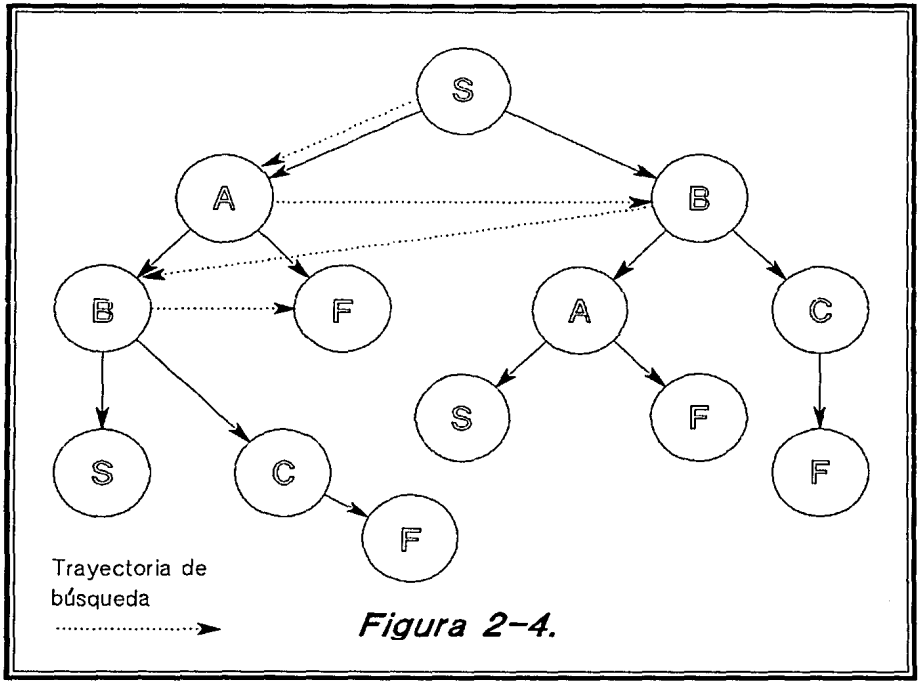



*Figura 2-4.*

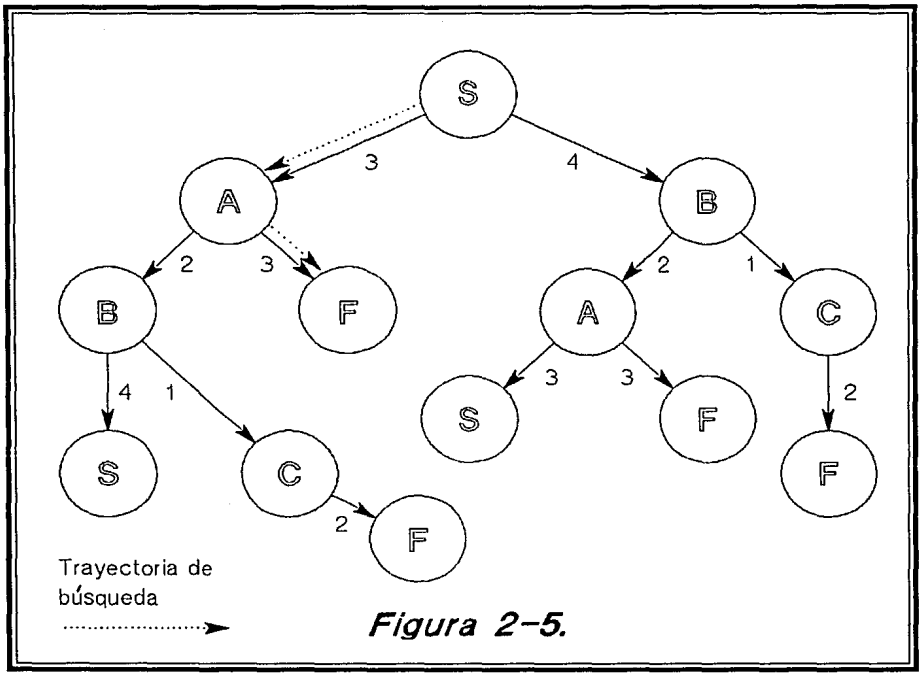



*Figura 2-5.*

6) Terminar el proceso

Para el ejemplo que hemos estado manejando, en la figura 2-5 se muestra la ruta que seguiría este método.

Con esta metodología, se pueden reducir bastante las transiciones necesarias entre estados, en orden de poder arrivar al estado meta.

Pero de nueva cuenta, este método no garantiza, al igual que los dos anteriores, la elección de la mejor ruta.

### 2.2.4 Metodo "Best First".

Este método, es una variación sobre el anterior. La diferencia básica consiste en que mientras el sistema anterior realiza una búsqueda exhaustiva sobre las mejores posibles trayectorias, este elige la mejor y sigue esa ruta hasta encontrar la solución.

El algoritmo para seguir este sistema se muestra a continuación.

1) Posicionarse en el estado inicia.
2) Generar los estados sucesores ( si aplica ).
3) Encontrar cual es la ruta con mayores ventajas y viajar hacia ese estado.
4) Verificar si es el estado meta, en caso de no ser asi ir al paso 2.
5) Terminar el proceso.

El árbol que se generaría para el ejemplo que hemos manejado, se muestra en la figura 2-6.

A primera vista, parecería que este sistema puede generar mejores soluciones que lo primeros dos métodos que se trataron, sin embargo esto no es absolutamente cierto.

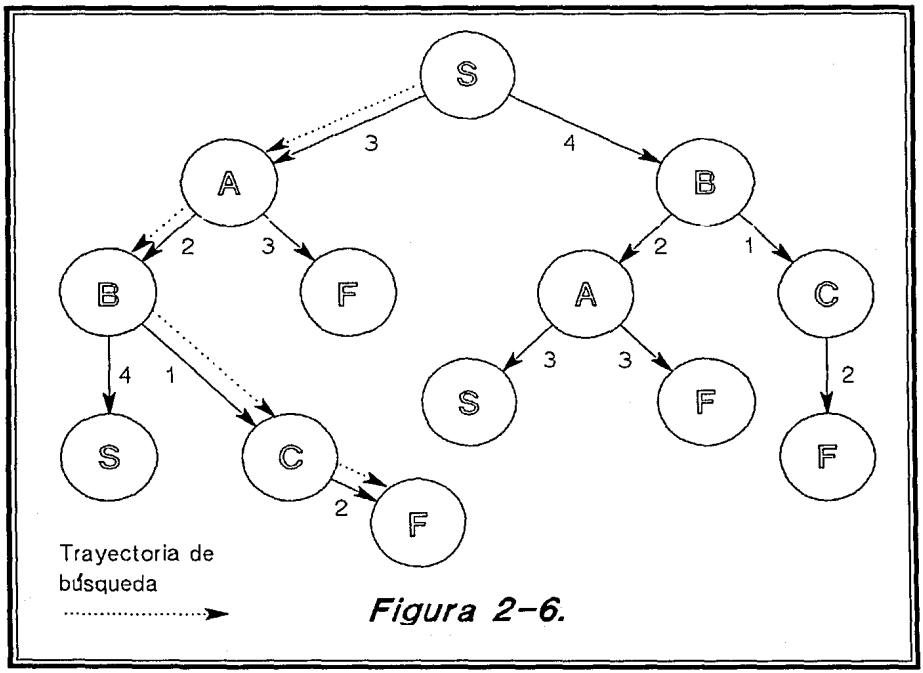



*Figura 2-6.*

## 2.2.5 Metodo "Branch and Bound".

De las metodologías vistas hasta estos momentos, esta es la más eficaz.

La idea en la cual está fundamentada es bastante simple, dado que lo que se desea es encontrar la mejor ruta, este método consiste en irla buscando a través de las mejores trayectorias que se va encontrando.

Los pasos a seguir son los siguientes:

1) Posicionarse en el estado inicial.
2) Generar los estados posteriores ( si aplica ).
3) Evaluar la mejor alternativa de ruta, guardando el estatus de las restantes.
4) Viajar al estado seleccionado y verificar si es el estado final, en caso de no ser así, ir al paso 2.
5) Terminar proceso.

De los métodos analizados, este es el único que garantiza que la ruta seleccionada al final del proceso es la óptima.

Sin embargo este método tiene la desventaja que si por alguna razón, alguna trayectoria corta tiene como hoja un estado intermedio, el algoritmo cae en un ciclo y nunca encuentra el camino deseado.

| # Comparación | Trayectorias a evaluar | Trayectoria elegida | Costo |
|---|---|---|---|
| 1 | S-A<br>S-B | S-A | 3 |
| 2 | S-A-B<br>S-A-F<br>S-B | S-B | 4 |
| 3 | S-A-B<br>S-A-F<br>S-B-A<br>S-B-C | S-B-C | 5 |
| 4 | S-A-B<br>S-A-F<br>S-B-A<br>S-B-C-F | S-A-B | 5 |
| 5 | S-A-B-S<br>S-A-F<br>S-B-A<br>S-B-C-F | S-A-F | 6 |

En la tabla anterior, se muestra el camino que hubiera seguido esta técnica para llegar a la solución del problema que hemos estado siguiendo.

### 2.2.6 Arboles "AND-OR".

Para concluir con esta subsección, nosotros analizaremos como solucionar problemas a traves de busquedas sobre arboles "AND-OR".

La idea de la cual desciende la técnica de "AND-OR", parte de dos conceptos básicos obvios, estos son:

- Tratar de convertir un problema complejo en uno más simple equivalente.

- Tratar de convertir un problema complejo en varios subproblemas simples.

Analizando la primera idea, para la gran mayoría de los problemas, muy probablemente exista más de un problema simple equivalente al complejo bajo estudio. Esto implica que al resolver culaquiera de los problemas simples equivalentes, nosotros habremos encontrado la solución de nuestro problema complejo bajo estudio.

Con lo antes expuesto, nosotros podemos construir un árbol "OR", donde el nodo raíz será el problema complejo inicial y sus nodos hijos serán problemas simples equivalentes. Al solucionar nosotros cualquiera de los problemas simples equivalentes, habremos encontrado la solución al problema original.

Analizando la segunda idea, otro método para la resolución de una situación problemática, es la de partir el problema complejo en varios subproblemas, de tal forma que para llegar a la solución de la situación original, se deben de resolver primero los subproblemas asociados al punto de partida original.

Tomando esto en consideración, nosotros podemos construir un árbol donde la raíz estara dada por la situación original y sus hijos seran los subproblemas a resolver.

En la figura 2-7, se muestra la forma en que esquemáticamente se representan los arboles "OR" y los arboles "AND".

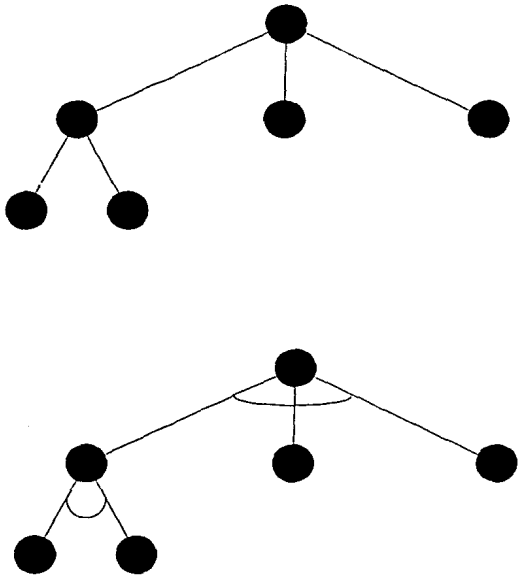


*FIGURA 2-7*

El árbol que se muestra en la parte superior de la figura, corresponde a un árbol "OR", mientras que el de la parte inferior corresponde a un árbol "AND".

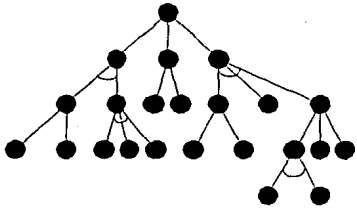
Como se puede observar, las ramas del árbol "AND", se encuentran relacionadas por medio de un semicirculo, indicando con ello que todos los nodos inferiores deben de ser satisfechos en orden de poder arribar a la solución del problema original.

Debido a que cuando se enfrentan situaciones del mundo real, rara vez se pueden encontrar arboles "AND" o "OR" puros, nació el concepto del arboles "AND/OR".

En orden de que un árbol pueda ser catalogado dentro de esta clasificación, debe de cumplir con ciertas características, de las cuales la más importante es:

*- Un árbol "AND/OR", debe de constar de una mezcla pura de nodos del tipo "AND", los cuales deberán de ser totalmente satisfechos en orden de llegar a la solución final del problema original. De la misma manera debera de contar con nodos puros del tipo "OR", de forma que al encontrar la solución de cualquiera de estos nodos, la rama en la cual se encuentren quedará satisfecha.*

Un ejemplo de un árbol "AND/OR", se muestra en la figura 2-8, como se podrá observar el nodo original es un nodo del tipo "OR", lo cual implica que si se encuentra la solución a cualquiera de los nodos hijos, se habrá resuelto el problema. Mientras que los nodos hijos de la derecha e izquierda son nodos del tipo "AND", lo cual implica que todos sus nodos hijos deberán de ser resueltos para encontrar la solución de estos nodos.



Otro problema que se ha encontrado en la práctica, radica en el hecho de que algunas veces es difícil encontrar nodos puros "AND" o "OR". En orden de solucionar este tipo de conflictos, Un nodo mixto puede ser convertido a nodos puros, tal como se muestra en la figura 2-9.

*FIGURA 2-8*

Esta técnica, es útil para representar una gran variedad de problemas, los cuales puedan ser redefinidos en función de varios problemas simples o problemas simplificados alternos.

Sin embargo se debe de tener una gran precaución con esta técnica, debido a que ya en problemas complejos regularmente los arboles "AND/OR" generados son excesivamente grandes. Se debe de poner un especial empeño en aquellos nodos terminales que no representen una aportación significativa a la solución de el problema. Y para eficientar las búsquedas sobre el árbol se puede utilizar una técnica alterna de las ya vistas en los puntos anteriores.

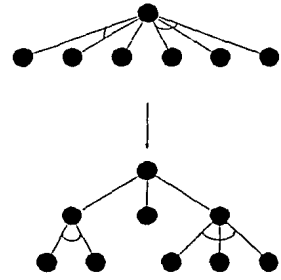


*FIGURA 2-9*

## 2.3 Ayudas Heurísticas*.

En orden de poder resolver problemas de una gran complejidad, a menudo se tiene que llegar a un compromiso, entre los requerimiento de movilidad, sistematicidad y la estructura de control del sistema. De aquí nace la idea de "heurística". Las técnicas heurísticas, son usadas para eficientar la búsqueda de soluciones, a menudo a costa de la exactitud de las mismas. Las ayudas heurísticas funcionan como una especie de guía de turistas, encontrando las trayectorias de mayor interés y guiando la búsqueda hacía estas trayectorias promisorias.

Algunas de estas técnicas guian el proceso de búsqueda sin sacrificar exactitud. Otras (muchas de ellas las mejores), pueden ocasionalmente no tomar en cuenta la mejor trayectoria, sin embargo en promedio eligirán las alternativas de mejor calidad.

*La palabra heurística viene de la palabra griega huriskein, que significa "descubrir"

Usando ayudas heurísticas, se puede esperar que se encuentre una muy buena solución, si bien no lo óptima, en problemas sumamente complicados.

Existen gran cantidad de técnicas heurísticas de propósito general que pueden ser utiles en una gran variedad de situaciones. Además, es posible construir ayudas heurísticas de propósito especial, para solución de situaciones muy específicas.

Las técnicas heurísticas pueden ser usadas junto con las anteriores ya tratadas.

Un ejemplo de este tipo de ayudas, para solucionar el problema planteado en la sección anterior, sería la siguiente:

1) Ir al nodo origen.

2) Seleccionar el nodo más cercano, entre aquellos que no hayan sido visitados, e ir a él.

3) Repetir el paso anterior hasta encontrar la solución.

Otro ejemplo, sería: Dada la tarea de descubrir ideas interesantes de algun tópico específico:

Si hay alguna función interesante de dos aurgumentos $f(x,y)$ observar que pasa si x *es igual a* y.

Si el tópico analizado es matemáticas, esta idea ayuda a descubrir el cuadrado si *f* esta en función de repetición, por otra parte ayuda a descubrir la función identidad si *f* es el conjunto unión, ( AM "Discover mathematics function" Lenat 1983).

Para concluir con esta sección, es necesario advertir que las técnicas heurísticas generales a menudo ofrecen un margen de error de tal manera que por medios probabilísticos se puede determinar el porcentaje de incertidumbre de la solución obtenida.

Sin embargo, la mayoría de los problemas de IA, utilizan técnicas generales en combinación con otras desarrolladas a la medida del problema, es decir particularizadas. De modo que en estas circunstancias, resulta bastante complicado determinar un rango de incertidumbre de la solución obtenida.

De todas formas, abogando por el uso de ayudas heurísticas, podemos argumentar que en la mayoría de problemas de tipo suave*, se cumplen las siguientes características:

En muy pocas ocasiones, se necesita la solución óptima, ya que en la gran mayoría una buena aproximación satisface completamente las necesidades del problema.

El margen mayor de error, se plantea para condiciones extremadamente adversas y estas suelen por sí mismas ser valuadas probabilísticamente y tener con ello una idea de lo que pasaría si estas condiciones se presentaran.

Para un estudio mucho más a fondo de este tipo de herramientas, se puede consultar a "How to solve it", Polya 1957.

## 2.4 Caracterización de Problemas.

Todas las técnicas vistas en las dos secciones anteriores, son por sí mismas demasiado generales, de forma que usadas en su forma pura, sólo son útiles para la resolución de problemas muy específicos, salvo las ayudas heurísticas desarrolladas para la solución de un problema en particular.

De manera que para decidir cual técnica o conjunto de técnicas utilizar es necesario primero analizar las características del problema a resolver, para entonces tener bases suficientes y elegir el método o conjunto de métodos que mejor se puedan adaptar en la búsqueda de la solución.

Las características más relevantes de cualquier problema de IA, se pueden derivar al analizar la situación y enclarecer dudas como:

¿ Se puede descomponer el problema en situaciones independientes de mayor sencillez ?

¿ Pueden los pasos definidos para la solución, ser ignorados o desechados si se prueba que en esos momentos son irrelevantes ?

¿ Es el medio ambiente del problema predecible ?

¿ Se puede tomar por buena una solución sin necesidad de buscar y comparar soluciones aún no investigadas ?

* Se utiliza la terminología de la teoría de sistemas, en donde un problema de tipo suave se conoce como aquel en el cual

¿ La estructura del conocimiento a ser usada en la solución es internamente consistente ?

¿ Se requiere de una gran cantidad de información para resolver el problema o esta sólo es relevante en determinadas condiciones de duda-acción ?

¿ Puede un algoritmo por sí solo solucionar el problema o requiere de interacción humana forzosamente ?

A continuación se analizarán cada una de estas preguntas para conocer cual es su implicación, en el planteamiento de soluciones y metodos.

2.4.1 ¿ Se puede descomponer el problema ?

Supongamos que nosotros queremos resolver el problema de calcular la siguiente expresión:

$$\int (x^2 + 3x + \text{sen}^2 x * \cos^2 x)\,dx$$

Este problema se puede resolver partiendolo en varios problemas más sencillos, los cuales a su vez pueden ser resueltos usando una pequeña colección de reglas bien definidas. En la figura 2-10, se muestra como sería la solución de este problema usando arboles "AND/OR". Es evidente que este problema cuenta con la característica de que es facilmente descompuesto en subproblemas.

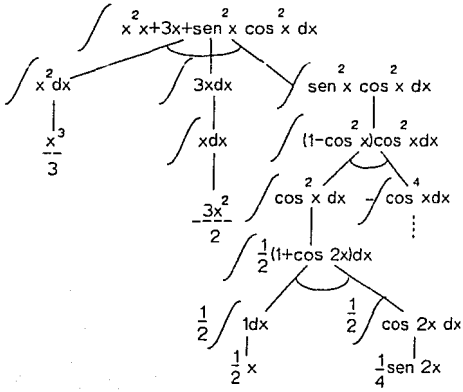



*Figura 2-10*

Un integrador por medio de este tipo de herramientas, seguiría algunos pasos bien definidos, estos son:

Comenzaría por ver si la expresión tiene una solución directa, en caso de que así fuera regresaría la solución. De otra forma vería la manera de partir la expresión en varias subexpresiones más sencillas y se llamaría recursivamente hasta haber resuelto por completo todo el problema.

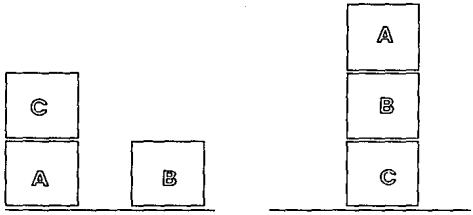



*Figura 2-11*

Sin embargo no todos los problemas presentan este circunstancia en especial.

Dentro de los problemas clásicos de la IA, existe uno llamado el mundo en bloques.

Considerese la figura 2-11, el estado inicial es el del lado izquierdo de la figura, el estado final es el que se muestra en el lado derecho de la misma.

Vamos a asumir que tenemos los siguientes operadores válidos:

Si el bloque "X" no tiene nada, coloca el bloque Y sobre el bloque X, Sobre (X, Y).

Si el bloque "X" tiene algo encima, quitar lo que hay sobre el, Quita de (X).

Suponiendo que el problema se puede descomponer, como se vera más adelante esto es falso, nosotros generamos el árbol de la figura 2-12.

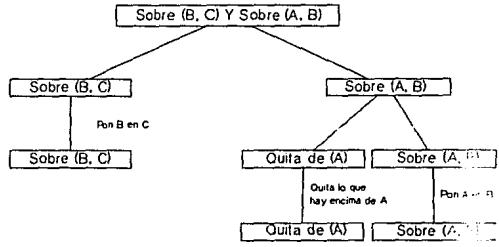



*Figura 2-12*

La rama izquierda del árbol, no tiene mayor complicación, simplemente tenemos que ejecutar la primera opción disponible que tenemos "Sobre (B, C)" y esta rama estará resuelta.

Pero en el caso de la rama de la derecha la solución no es tan simple ya que solo contamos con dos posibles movimientos tenemos que ejecutar el segundo, con esto quitamos lo que hay encima del bloque A y después ponemos A en B. Pero al tratar nosotros de combinar las dos ramas ya solucionadas, el resultado no es el esperado.

La explicación de la falla es obvia, ya que las dos ramas interactuan entre ellas y esta interacción debe de ser tomada en cuenta para poder llegar al estado final deseado.

2.4.2 ¿ Pueden los pasos definidos para la solución ser ignorados ?

Para aclarar este punto, vamos a analizar los siguientes casos:

a) Probar teoremas,
b) Jugar "rompe-cabezas",
c) Jugar ajedrez.

Para el primer caso, nosotros deseamos verificar si un teorema matemático es cierto o no. Lo primero que hacemos, es búscar algún axioma de partida que pensamos que pueda ser útil. Pero que pasa si ese axioma no ayuda en la forma que nosotros habiamos planeado, ¿ Nos encontramos ante un problema grave ?

Claro que no, ya que lo único que hemos perdido es tiempo, de modo que podemos buscar otro axioma y empezar de nuevo.

Este problema tiene una característica muy importante y esta es que: Nosotros podemos ignorar a aquel conjunto de reglas o pasos siempre y cuando demostremos que en un instante especifico no son relevantes para la solución del problema.

Ahora analisemos el problema de jugar "rompe-cabezas". Vamos a suponer que nuestros estados inicial y final son los que se muestran en la figura 2-13. ¿ Que pasa si comenzamos realizando una jugada erronea, tal como mover la ficha 5 al hueco ?

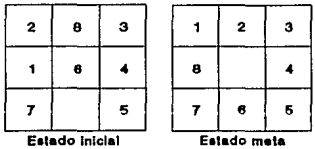



*Figura 2-13*

Nuestro problema no es muy grave ya que nosotros nos podemos recuperar de nuestro error, regresando la ficha 5 a su lugar original y moviendo la ficha 6 al hueco.

Este problema, presenta una situación distinta al caso anterior, y esta es que: en caso de una equivocación, nosotros nos podemos recuperar y proseguir con la búsqueda de la solución.

En el problema de jugar ajedrez, la situación es muy distinta, ya que si realizamos una mala jugada, por ejemplo al mover un peón y ponerlo en peligro, nosotros no podemos regresar el juego a su estado original.

De forma que en este caso, la característica de la situación es la siguiente: El pasar de un estado a otro, es un proceso irreversible.

Con estos tres ejemplos, hemos ilustrado tres situaciones distintas. De la manera que nosotros conceptualisemos el problema, dependera la formulación del sistema de producción a ser usado para su solución.

### 2.4.3 ¿ Es el universo del sistema predecible ?

Usaremos de nuevo el ejemplo del juego de "rompe-cabezas".

Siempre que pensamos el movimiento a realizar, sabemos exactamente lo que va a pasar, de modo que podemos planear nuestros movimientos y asi evitar el cometer errores.

Pero esta situación cambia si consideramos el juego de domino, ya que nunca conocemos con exactitud los movimientos que va a realizar el contrario, de modo que para planear nuestro juego, tenemos que hacer uso de las probabilidades.

Estos dos ejemplos sencillos, nos muestran la diferencia entre dos situaciones distintas, la primera cuenta con un universo completamente predecible, mientras que la segunda se mueve en un universo incierto.

### 2.4.4 ¿ La solución búscada es absoluta o relativa ?

Suponiendo de nueva cuenta los casos del "rompe-cabezas" y el domino, se puede observar lo siguiente.

Para el primero, se ha asignado un estado meta único al cual nosotros debemos llegar, de modo que la solución búscada es absoluta y no existe otra posible.

En cambio en el juego de domino, suponiendo que la meta sea ganar el juego, existen varias formas de lograrlo, como cerrar el juego o ahorcarle piezas al oponente o ayudar al compañero de juego a ganar, de tal forma que existen varias soluciones relativas para este juego.

Esta caracteristica va a ser de vital importancia en la selección del método de búsqueda de soluciones.

2.4.5 ¿ Es la estructura del conocimiento consistente ?

Si nuestro problema fuera el comprobar un teorema, a partir de un conjunto de axiomas, la primera labor seria la de verificar que entre los axiomas no existan contradicciones, o sea que sean consistentes.

Para esto existen ya métodos matemáticos muy provados, de modo que se pueda garantizar la consistencia de los axiomas.

Ahora suponiendo el caso propuesto por Novak, (1980), el cual consiste en lo siguiente:

> Un hombre se encuentra ubicado a 150 ft. de un blanco, su meta consiste en pegarle al blanco por medio de una bala. Suponiendo que la bala viaja a 1500 ft./seg., ¿ cuanto debe de levantar su brazo para realizar su meta ?

Una forma de solucionar este problema, es por medio del siguiente razonamiento. A la bala le toma .1 segundos viajar del arma al blanco, suponiendo que viaja en linea recta los 150 ft.. Durante ese tiempo la bala pierde una altura igual a:

$$1/2\ g\ t^2\ ft = 1/2\ (32)\ (.1)^2\ ft = .16\ ft$$

De modo que si el hombre le quiere pegar al blanco, debera de apuntar 0.16 pies arriba de este.

Ahora es importante notar, que para solucionar este caso, nosotros hicimos dos suposiciones que se contradicen. La primera de ellas es que el proyectil viaja en linea recta, mientras que la segunda dice que describe un arco y pierde altura.

Estos dos simples ejemplos, nos muestran las diferencias entre situaciones con estructuras de información consistentes e incosistentes.

2.4.6 ¿ Cual es en realidad el rol del conocimiento ?

Considerando de nueva cuenta el problema de ajedrez y suponiendo que tuvieramos una computador de poder ilimitado, la pregunta sería, ¿ Que tanto conocimiento se requeriría para realizar un programa perfecto ?

La respuesta a este pregunta, es muy poco. Ya que podemos realizar un excelente programa a partir de un conjunto reducido de reglas y estrategias de control. Obviamente el conocimiento adicional de táctias y estrategias, ayudara a mejorar el tiempo de respuesta del sistema.

Pero consideremos ahora el problema de realizar un sistema el cual pueda buscar noticias en los periodicos de trascendencia politica. En este caso el sistema para realizar su función necesitaría de una gran cantidad de información acerca de partidos politicos, historia de las personalidades envueltas en la politica, probables estrategias electorales, areas geográficas de interes, etc..

Estos dos ejemplos nos muestran las diferencias entre sistemas en los cuales el conocimiento solo es necesario para optimizar la busqueda y aquellos en los cuales el conocimiento es usado para interpretar el medio ambiente del sistema.

2.4.7 ¿ Requiere el sistema de interacción con un humano ?

Para aclarar este último punto, ejemplificaremos con las siguientes situaciones:

a) El problema del agente viajero,
b) El problema de diagnostico medico.

En el primer caso, se desea saber que ruta se debe de seguir para visitar un conjunto de ciudades, las reglas son: sólo se debe de visitar cada ciudad un vez y se debe de elegir la ruta más corta.

Para solucionar este ejemplo se puede usar el método de "Branch and Bound" o alguna ayuda heurística. Simplemente hay que proporcionarle los datos y el algoritmo se encargara de encontrar la solución sin ninguna interacción adicional.

Para el segundo caso, se desea contar con un sistema de diagnostico medico, las reglas son: el paciente debe de llegar y proporcionarle sus datos y molestias al sistema, si el sistema tiene alguna duda, debera indagar más acerca de esa molestia en particular con el paciente, en caso de ser algún padecimiento simple debera recetar dosis, en caso contrario debera encausar al paciente con el especialista.

Para que este sistema pueda cumplir con su función, es obvio que requiere de una gran cantidad de intercambio de información con el usuario del sistema.

A los problemas del primer tipo, se les conoce como sistemas solitarios, mientras que a los del segundo tipo se les denomina como sistemas conversacionales.

## 2.5 Características de los Sistemas de Producción.

Para concluir con este capítulo, analizaremos las características de los Sistemas de Producción y su interrelación en la solución de problemas.

Después de haber analizado en el punto anterior, las características de distintos tipos de problemas, parece razonable hacernos los siguientes cuestionamientos:

¿ Pueden los Sistemas de Producción, al igual que los problemas, ser modelados en base a sus características, de forma que puedan ser facilmente implementados ?

¿ Si es así que relación existe entre los tipos de sistemas y los tipos de problemas ?

La respuesta a la primera pregunta es si. Considerese la siguiente clasificación de Sistemas de Producción.

**Sistemas de Producción Monotonos.**

*Son aquellos, en los cuales la aplicación de un regla en un tiempo dado, nunca previene de la aplicación de otra regla que pudo ser usada en su lugar.*

**Sistemas de Producción Parcialmente Conmutativos.**

*Son aquellos, en los cuales si la aplicación de una secuencia de reglas particulares transformo el estado X en el estado Y, entonces cualquier permutación de estas reglas es permitida y también transformarán el estado X en Y.*

**Sistemas de Producción Conmutativos*.**

*Son aquellos sistemas que son monotonos y parcialmente conmutativos.*

La importancia de estas categorias, radica en la relación que existe entre la estrategia de implementación y el tipos de Sistema de Producción.

*Esta corresponde a la definición de Sistemas de Producción Conmutativos dada en Nilsson, (1980).

Ahora, regresando a la segunda pregunta, existe un número infinito de Sistemas de Producción para cada problema. Algunos se adaptaran de una manera más natural, otros serán más eficientes. Resumiendo, cualquier problema puede ser resuelto por cualquier tipo de Sistemas de Producción y cualquier problema que puede ser resuelto por cualquier Sistema de Producción, puede ser resuelto por un *Sistema Conmutativo*. De manera que formalmente no existe ninguna relación entre los tipos de problemas y los tipos de Sistemas de Producción, aunque en la realidad si existan.

# CAPITULO III

# REPRESENTACION DEL CONOCIMIENTO

## 3.1 Introducción.

Uno de los resultado más evidentes que han dejado las investigaciones de la IA, es sin duda alguna, el papel que desempeña el conocimiento, dentro de cualquier sistema que posea inteligencia.

Uno de los aspectos más relevantes en el análisis y desarrollo de sistemas bajo el enfoque de la IA, es el como se estructura el conocimiento, para que el sistema pueda hacer un buen uso de él.

En el presente capitulo, se explica el problema de representación del conocimiento así como de diversas técnicas propuestas. De la misma manera, se explica el razonamiento y su papel dentro de cada sistema de IA.

## 3.2 Conceptos fundamentales.

Antes de embarcarnos en las técnicas básicas de representación del conocimiento, empezaremos por repasar algunos conceptos básicos, que posteriormente serán usados.

### 3.2.1 Metodos débiles.

Con este nombre se les denomina a aquellos sistemas que tienen las siguientes características:

- Se mueven en un mundo incierto,

- Dependen de un proceso de búsqueda para encontrar una solución viable.

Este tipo de métodos se componen de las siguientes partes:

- Un método o combinación de métodos de búsqueda de los ya

analizados en el capítulo anterior.

- Un procedimiento que guie el proceso de búsqueda.

Reciben el nombre de métodos débiles, ya que siempre se encuentran expuestos a sufrir de explosiones combinatorias en el proceso de búsqueda de soluciones.

Generalmente el procedimiento que guía la búsqueda, será algún procedimiento heurístico, ya sea de propósito general o creado especialmente para la solución del problema en cuestión.

### 3.2.2 Representación de estados.

La forma en que el problema será representado, constituye el esqueleto sobre el cual se va a apoyar el método de solución.

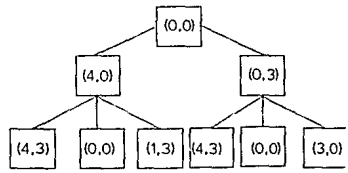



*Figura 3-1*

Los problemas se pueden representar gráficamente de varias formas. Pero para los fines practicos de este trabajo nos concentraremos en solo dos de ellas.

La primera de ellas es la representación en forma de árbol, ejemplificando, usaremos el problema de las dos jarras descrito en la sección 2.1.2, para este caso, el árbol que se generaría se muestra en la figura 3-1.

La ventaja de este método radica en su sencillez para ser implementado, pero cuenta con una gran desventaja, esta es que si por las características propias del problema bajo estudio, los estado se repiten con una gran frecuencia, el sistema acaba con la memoria disponible para su representación. En este ejemplo, los estados (4,3) y (0,0) apenas en el segundo nivel ya se encuentra repetidos.

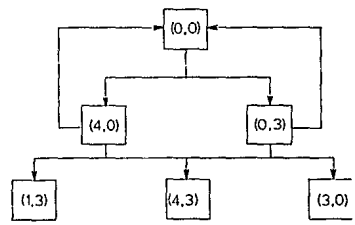



*Figura 3-2*

La segunda forma, es a tráves de la técnica de representación de gráficas. Para el mismo caso, la gráfica que se generaría se muestra en la figura 3-2.

Como se puede observar, con esta forma de representación, la redundancia se elimina a costa de una cantidad importante de proceso adicional.

### 3.2.3 Selección de Reglas.

En el capítulo anterior, se explico que era una regla de producción. También se comentó que este tipo de reglas no ayudan a determinar la dirección que va a seguirse en los algoritmos de búsqueda.

Se recordará que este tipo de reglas son de la forma:

Si (condición) entonces (ejecuta una acción)

Pero, que pasa cuando nuestro conjunto de reglas de producción es demasiado grande ¿ Como vamos a encontrar la regla adecuada ?, (lo cual es una situación bastante común en sistemas complejos).

Existen varias técnicas para solucionar este problema, a continuación se explicarán brevemente las más populares.

#### 3.2.3.1 Indexación.

Este algoritmo consiste en tomar el estado actual como índice para encontrar las reglas aplicables.

Tiene la ventaja de que es bastante sencillo de aplicar, pero con la desventaja de que si por alguna razón el problema requiere de regresar a un estado anterior, el método falla, ya que no comtempla regresiones, tómese como ejemplo el método de búsqueda branch-and-bound, en el cual es necesario regresarse en multiples ocasiones al estado anterior, para poder encontrar la mejor trayectoria.

#### 3.2.3.2 Comparación con Variables.

En algunas ocasiones, el problema de selección de reglas, va más allá de simples comparaciones. Vamos a suponer el siguiente caso:

Nosotros implementamos una base de datos, usando lógica de predicados*, con los siguientes datos:

```
Padre(Antonio, Juan), léase Antonio es padre de
                      Juan.
Padre(Juan, Sergio),
Padre(Juan, Ignacio)
```

También nosotros tenemos implementada una base de datos de reglas de producción, de las cuales dos de ellas son:

*Esta técnica sera explicada más adelante.

Abuelo( X, Z ) si Padre( X, Y ) y Padre( Y, Z )

Si Abuelo ( Antonio, Ignacio ) = Cierto entonces
  Termina proceso

Como se puede observar, la primera regla nos define la relación abuelo en base a dos condiciones de la relación padre. La segunda regla nos dá una condición de terminación, en caso de que la relación abuelo(Antonio, Ignacio) sea cierta.

El proceso, al deteminar la aplicabilidad de alguna de las dos reglas, tendría que hacer substituciones para determinar si son ciertas o falsas.

Este método cuenta con una gran flexibilidad y de hecho en lenguajes como PROLOG, la implementación es de una forma natural.

Sin embargo, la desventaja de este método, estriba en que debe de ir acompañado de algún sistema de resolución de conflictos, ya que en problemas complejos las relaciones pueden ser de muchos a muchos, de modo que las variables pueden tomar más de un valor a un tiempo.

Como en este ejemplo, si nosotros tuvieramos una regla del tipo:

Padre(Antonio, Z) entonces ( acción )

la variable Z tomaría dos valores, Z=Sergio y Z=Ignacio.

### 3.2.3.3 Comparación con Incertidumbre.

En algunos problemas, principalmente en sistemas de reconocimiento de lenguaje natural y visión, es necesario tratar con reglas de producción que manejen inexactitudes y aproximaciones.

Para este tipo de problemas, se tiene que contar con un conjunto primario de reglas de producción. En este conjunto primario se deberán de manejar aproximaciones y hechos clave.

Ejemplificando, vamos a suponer un sistema experto para consulta de una base de datos geográfica, la cual sirva para proporcionar información como:

- Estados de un país,
- Ríos de un estado,
- Población de un estado o país, etc..

Al momento que el usuario diera una instrucción al sistema para obtener información, este deberá de tomar algunas palabras clave, como:

> dame *todos* los *estados* de México
> cual es el *tamaño* del *río bravo*

En la primera pregunta se podrían tomar las palabras clave *"todos, estados, México"* y con esto inferir la búsqueda de la respuesta en la base de datos, eliminando las demás palabras que no son relevantes, en este caso, *"dame los de"*. En la segunda, de la misma manera se tomarían *"tamaño y río Bravo"*, eliminando *"cual es el del"*.

Más adelante, en el capítulo cuatro, se explicará en más detalle este método.

3.2.3.4 Dirección de las Búsquedas.

Para explicar este punto, comenzaremos por definir lo que es el razonamiento. El razonamiento es:

> *El proceso por medio del cual, se explica y justifica la toma de una decisión.*

En la solución de problemas bajo el enfoque de la IA, el razonamiento es usado para guiar los cambios de estado de acuerdo a las reglas de producción.

El razonamiento se produce en tres formas básicas, las cuales se explican a continuación.

*Razonamiento Progresivo.* De esta forma, se razona desde un estado inicial para encontrar una trayectoria que nos conduzca hacía algun estado meta.

*Razonamiento Regresivo.* Utilizando este tipo de razonamiento, desde algún estado meta partimos para encontrar una trayectoria que nos conduzca a algún estado inicial.

*Razonamiento Bidireccional.* Esta es una combinación de los dos anteriores, dadas ciertas condiciones el razonamiento sera progresivo y en otras sera regresivo.

La forma a elegir, dependerá de las características del problema bajo estudio. Sin embargo se pueden dar algunos caminos de elección como los siguientes:

Si el número de estados finales es muy superior al número de estado iniciales, convendrá utilizar razonamiento progresivo.

Si el número de estado iniciales es muy superior al número de estados finales, convendra utilizar razonamiento regresivo.

Si la situación bajo estudio requiere que durante su solución, constantemente revierta sus decisiones, como ejemplo se tiene el juego del "rompe-cabezas" (consultar la sección 2.4.2), convendrá utilizar razonamiento bidireccional.

### 3.2.4 El Problema de Representación.

Antes de tomar una decisión, sobre como vamos a estructurar el conocimiento que nuestro sistema va a utilizar, debemos de analizar la manera en como vamos a representar las ideas envueltas en el conocimiento.

La experiencia en las investigaciones de IA, ha mostrado que un buen diseño de la representación del conocimiento, es a menudo la llave que torna un problema extremadamente complejo en uno mucho más simple, de modo que se justifica una inversión considerable de tiempo el establecimiento de los símbolos a ser usados y en como estos símbolos van a ser arreglados para poder producir descripciones a partir de ellos.

Nos basaremos en la Teoría de Minsky, también conocida como la "Teoría de Representaciones Ricas en Símbolo"*. Para una consulta más en detalle de lo que aquí vamos a tratar, se puede consultar a [Theory of Frames, Minsky 1974].

Cuando nos enfrentamos a una nueva situación o cuando analizamos una situación desde un punto de vista diferente, extraemos de nuestro problema una estructura llamada representación. Esta estructura puede ser comprendida como el esqueleto a ser adaptado para representar la realidad.

Una representación, es una estructura de datos, por medio de la cual vamos a modelar situaciones estereotipadas. Asociada a cada representación, se encuentran diversas clases de información como:

- Información acerca de como usar la representación,
- Información de que es lo que puede suceder en un paso posterior,

*Se escogio la traducción que más se aproxima al significado real en ingles

- Información de pasos a seguir si el punto anterior no se cumple.

Una representación puede ser visualizada como una red de nodos y relaciones. Donde el nivel superior de la representación es fija y en ella se encuentra descritas ideas y situaciones que siempre son verdaderas para la situación supuesta. En el nivel inferior, se encuentra modeladas situaciones especiales y/o datos, también se pueden encontrar referencias o ligas a otras sub-representaciones. Condiciones muy complejas pueden requerir relaciones especificas que involucren a diversos nodos del nivel inferior.

Diversas representaciones pueden ser agrupadas para formar un *Sistema de Representaciones*. Acciones complejas que involucran a más de una representación, son conocidas como *transformaciones*.

Para clarificar esta idea, vamos a suponer que nosotros estamos modelando un sistema para el análisis de escenas visuales. Para este caso, las distintas representaciones del sistema, modelarían la escena desde distintos puntos de vista y las transformaciones entre una representación y otra, representarían escenas desde diferentes puntos visuales. Para otro tipo de sistemas, las diferentes representaciones, modelarían acciones, relaciones causa-efecto o cambios en puntos de vista conceptuales.

La principal idea en sistemas de representaciones radica en: ***Diferentes representaciones del sistema comparten nodos terminales***, y esto es lo que hace posible la integración de puntos de vista diferentes.

Los sistemas de representaciones, se deben de encontrar necesariamente ligados a una *Red de extracción de Información*. El objetivo de esta red es de proveer representaciones de remplazo, lo cual se justifica cuando condiciones adversas causan que una representación terminal no encuentre una salida rentable a la situación a que se enfrenta.

Posteriormente, se vuelve a tomar este tema para asociarlo ya a representaciones del conocimiento en sistemas expertos.

## 3.3 Lógica en Predicados.

Hasta estos momentos, nosotros hemos análizado los conceptos fundamentales de representación del conocimientos. A partir de estos momentos, nos dedicaremos a análizar distintos esquemas de representación.

La lógica de predicados, nos va a servir para representar hechos del mundo real en forma de funciones, (Wff), esta característica nos va a permitir usar instrucciones de lógica para representar conocimiento y además nos va a proveer de herramientas para razonar con este conocimiento.

Para investigar este método, vamos a utilizar el siguiente ejemplo:

Vamos a suponer que nosotros tratamos de representar la siguiente información, en base a lógica de predicados.

1. Julio fue un hombre,
2. Julio fue un pompeyo,
3. Todos los pompeyos eran romanos,
4. César era el gobernante,
5. De todos los romanos, algunos eran leales al César y los demás lo odiaban,
6. Todos son leales a algo o a alguien,
7. Las personas solo tratan de asesinar a gobernantes a quienes no le son leales,
8. Julio trata de asesinar a César.

Los hechos descritos en las oraciones anteriores pueden ser representados en lógica de predicados en forma de funciones, tal como se muestra a continuación.

a) Julio fue un hombre.

Esta aseveración, puede ser representada de la siguiente manera:

hombre ( Julio )

Como se puede observar, esta función capta la escencia de que Julio fué o es un hombre, sin embargo, hay que notar que no representa el hecho de que el verbo de la oración se encuentra en tiempo pretérito, de modo que si esta función la quisieramos regresar a su forma original, podría tener tres formas básicas:

a) Julio sera un hombre
b) Julio es un hombre
c) Julio fue un hombre

El punto de aceptabilidad de este error, dependerá de la estructura completa del conocimiento y del uso que se le intente dar.

b) Julio fue un Pompeyo.

Tomando en cuenta las restricciones que acabamos de mencionar, esta oración se representa:

pompeyo ( Julio )

c) Todos los pompeyos eran romanos.

Esta oración se puede representar de la siguiente forma:

$\forall$X pompeyo ( X ) --> romano ( X )*

Es importante recalcar que en esta representación se hace presente el concepto de"inferenci", ya que si la función pompeyo ( X ) resulta ser cierta, nosotros "inferimo" que la función romano ( X ), también es cierta.

d) César era el gobernante.

La forma de la función es:

gobernante ( César )

Aquí es importante comentar que en esta función, no se esta tomando en cuenta que varias personas pueden tener el mismo nombre y no ser gobernantes.

e) De todos los romanos, todos eran leales al César o lo odiaban.

Este conjunto complejo de oraciones, se puede representar de la siguiente forma:

```
∀X romano ( X ) -> [ leal ( X,César ) V
                     odia ( X,César )  ]
```

El símbolo "V" implica la función lógica "O exclusiva", sin embargo en el español la palabra "O" puede representar la función O inclusiva o exclusiva. De modo que una función más deseable que representa exactamente lo que se desea es la siguiente:

```
∀X romano ( X ) -> (( leal ( X,César ) V
                      odia ( X,César ) )
                   ^
                NO ( leal ( X,César ) ^
                     odia ( X,César ))
```

El símbolo "^" implica la función lógica "Y".

∀= para todo; -> = implica;
∃ = existe

f) Todos son leales a algo o alguien.

La función para esta oración sería:

$$\forall X \; \exists Y \; \text{leal}\,(X,Y)$$

Se puede notar que esta función es especialmente ambigüa, ya que lo mismo puede representar que para cada "Y" existen varios "X" que le son leales o viceversa, que varios "Y" le son leales a un "X".

g) Las personas solo tratan de asesinar a gobernantes a quienes no son leales.

La función sería de la siguiente forma:

$$\forall X \; \forall Y \; \text{persona}\,(X) \wedge \text{gobernante}\,(Y) \wedge \text{trata-de-asesinar}\,(X,Y) \rightarrow \text{NO leal}\,(X,Y)$$

Al igual que la anterior, esta sentencia también es muy ambigüa, ya que lo mismo puede ser que diga que la gente solo trata de matar a aquellos gobernantes a los cuales no son leales, o que lo único en que la gente piensa es en matar a gobernantes a los cuales no le son leales.

Para esta función se tomó la primera interpretación.

h) Julio trató de asesinar al César.

Esta oración es representada por la siguiente función:

$$\text{trata-de-asesinar}\,(\text{Julio},\text{César})$$

Ya que tenemos todas nuestras funciones listas, procederemos a realizar una prueba formal, para esto, hagamos la siguiente pregunta:

¿ Julio era leal al César ?

Nosotros tenemos dos formas de contestar esta pregunta. La primera de ellas, es tratando de probar que Julio fue leal al César, mientras que la segunda es tratando de demostrar lo contrario.

En este caso la primera de ellas no puede ser usada, ya que en nuestro conjunto de funciones no nos dá un camino para realizar la demostración.

Así que trataremos por el segundo método.

Nosotros sabemos por la regla "g" que:

```
∀X ∀Y persona ( X ) ^ gobernante ( Y ) ^
      trata-de-asesinar ( X,Y ) -> NO leal ( X,Y )
```

de modo que realizando substituciones:

```
persona ( Julio ) ^ gobernante ( César ) ^
trata-de-asesinar ( Julio,César )
```

de "d" nosotros sabemos que gobernante ( César ) se cumple, tambien de "h" se satisface trata-de-asesinar ( Julio, César ), sin embargo no tenemos forma de satisfacer la cláusula persona ( Julio ).

De manera que para completar nuestra base del conocimiento, nosotros tenemos que agregar la regla:

```
i) ∀X hombre ( X ) -> persona ( X )
```

Esto es nosotros inferimos que si x es hombre, forzosamente x es persona.

Volviendo a nuestra pregunta y una vez que hemos agregado esta última regla, nosotros podemos contestar que Julio no era leal al César.

### 3.3.1 Algoritmo de Resolución.

El representar hechos en forma de funciones lógicas es un mecanismo muy conveniente, sin embargo, tienen que ir acompañado de mecanismos eficientes para la búsqueda y selección de reglas de inferencia.

Uno de los algoritmos más populares por su eficiencia, es el algoritmo de resolución. Este algoritmo opera en base a obtener pruebas por refutación, esto es, siempre trata de demostrar que las consultas sobre el conocimiento son falsas. Y basa su eficiencia, en la conversión de las funciones de lógica, a una forma estándar.

Así que antes de entrar a explicar como opera el algoritmo de resolución, vamos a explicar cual debe de ser la forma de las funciones.

#### 3.3.1.1 Conversión a FNC (Forma Normal Conjuntiva).

Vamos a suponer, que nosotros queremos representar la siguiente frase, en una función lógica.

> Todos los romanos que conocen a Julio, opinan que odian al César o piensan que una persona que odia a otra esta loca.

entonces la función sería:

```
∀X [ romanos ( X ) ^ conoce ( X,Julio )] ->
     [ odia ( X,César) V ( ∀Y ( EZ odia ( Y,Z )) ->
       piensa-loco ( X,Y )) ]
```

El poder usar esta función como ayuda para la solución de una situación envuelve un complejo método de comparaciones y substituciones.

Este proceso sería mucho más eficiente si todas las funciones cumplieran con dos características, éstas son:

- Que contengan los menos componentes posibles,
- Los cuantificadores ( ∀,E ) fueran separados del resto de la función, de forma que no fueran considerados.

La forma normal conjuntiva, [Davis, 1960], cuenta con ambas propiedades, la función anterior, en su forma normal conjuntiva sería:

```
NO romano( X ) V NO conoce ( X,Julio ) V
   odia ( X,César ) V NO odia ( Y,Z ) V
   piensa-loco ( X,Z )
```

El algoritmo para convertir cualquier función a su equivalente en forma normal conjuntiva se muestra a continuación, ( se asume un previo conocimiento del algebra de Bool ).

1) Eliminar ->, aprovechando la propiedad que *a -> b* es equivalente a NO *a* V *b*. Tomando el ejemplo, la función despues de esta primera transformación quedaría:

```
∀X NO[ romano ( X ) ^ conoce ( X,Julio) ]
   V [ odia ( X,César ) V (∀Y NO (EZ odia( Y,Z )
      V piensa-loco ( X,Y )) ]
```

2) Reducir el uso de NO, utilizando para ello las leyes de Morgan *(NO(NO p))=p, NO(a^b)=NO a V NO b* y *NO( a V b )=NO a ^ NO b*. Para cuantificadores, podemos usar el siguiente estandar [Rich, 1980], *(NO ∀X P(X))=EX NO P(X)* y *NO EX P(X)=∀X NO P(X)*. Aplicando este paso sobre el ejemplo, nos queda:

```
∀X [NO romano ( X ) V NO conoce ( X,Julio ) ]
 V [ odia ( X,César ) V (∀Y ∀Z NO odia ( Y,Z )
    V piensa-loco ( X,Y )) ]
```

3) Estandarizar variables, de modo que cada cuantificador cuente con un nombre de variable único. Es bueno recordar que los nombres de las variables los asignamos nosotros, y el ejecutar este proceso no afecta la naturaleza lógica instrínseca de la función.

Por ejemplo, la formula ∀X P(X) V ∀X Q(X), puede ser convertida a ∀X P(X) V ∀Y Q(Y).

Este paso es necesario y sirve de preparación para el siguiente. Nuestra función de ejemplo no resulta afectada aquí.

4) Mover todos los cuantificadores a la izquierda de la fórmula. Este paso es posible ya que debido al paso anterior, no existen conflicto en los nombres. Después de aplicar este paso a nuestra función de ejemplo, ésta queda:

```
∀X∀Y∀Z [NO romano ( X ) V NO conoce ( X,Julio )] V
       [ odia ( X,César ) V (NO odia ( Y,Z )
        V piensa-loco ( X,Y ))]
```

En este punto, la función se encuentra en una "Forma Normal Prenex". Esta consiste de un prefijo de cuantificadores seguido por una matríz libre de ellos.

5) Eliminar los cuantificadores de existencia E. Nosotros podemos eliminar el cuantificador, subsituyendo el argumento por una función que nos regrese el valor deseado.

Por ejemplo, en la función EY presidente ( Y ), puede ser transformada en la función presidente ( S1 ), donde "S1" es una función sin argumentos, la cual nos retornara un valor que satisfaga a la función presidente.

Nuestra función de ejemplo no sufre alteraciones en este paso.

6) Quitar los prefijos de universalidad (∀), sencillamente en este punto, todas las variables deben de cumplir con su universalidad, de modo que este prefijo sale sobrando.

De forma que nuestra función de ejemplo queda:

```
[NO romano ( X ) V NO conoce ( X,Julio )] V
[ odia ( X,César ) V (NO odia ( Y,Z ) V
  piensa-loco ( X,Y )) ]
```

7) Remover parentesis usando las propiedades asociativas y distributivas. Para nuestra función, despues de apicar este paso queda:

NO romano ( X ) V NO conoce ( X,Julio ) V
odia ( X,César ) V NO odia ( Y,Z ) V
piensa-loco ( X,Y )

8) A partir de la función del punto anterior, se pueden formar conjuntos separados de clausulas. Para que la función sea cierta, todas las clausulas generadas deben de ser ciertas. Si se va a trabajar con diversas funciones, todas las clausulas generadas deben de ser combinadas para representar el mismo conjunto de factores representados por la función original.

9) Repetir el paso 3, tomando en cuenta ahora todas las clausulas generadas en el paso anterior.

Despues de aplicar el procedimiento completo en lugar de tener un conjunto de funciones, tendremos un conjunto de clausulas y estas clausulas ya pueden ser usadas por el algoritmos de resolución.

### 3.3.2 Bases del Algoritmo de Resolución.

El algoritmo de resolución es un simple proceso iterativo, en donde a cada paso se toman dos clausulas, llamadas clausulas padres, las cuales son unificadas formando una nueva clausula, la cual ha sido inferida de las clausulas padre. La nueva clausula representa la forma en que las dos clausulas interactuan.

Ejemplificando, vamos a suponer un sistema que contiene las siguientes dos clausulas:

invierno V verano
NO invierno V frio

Observese que en una clausula contiene "invierno" y la otra "NO invierno" y ambas clausulas deben de cumplirse. Para que esto se logre, si "invierno" se cumple, entonces "frio" se debe de cumplir para que la segunda clausula sea cierta. Si "NO invierno" se cumple, entonces "Verano" se debe de cumplir para que la primera clausula sea cierta.

Analizando ambas clausulas, puede inferirse que una clausula equivalente seria:

verano V frio

Este es el camino por el cual procede el algoritmo de resolución. El algoritmo opera tomando dos clausulas, en ellas debe de existir una misma litera, la cual debe de ocurrir en forma positiva en una y negada en la otra. La resolución opera combinando las literales que no se cancelan de las dos clausulas padre y produciendo con ellas una nueva clausula.

### 3.3.3 Algoritmo de Unificación.

En lógica de predicados, el procedimiento de comparación de literales, es un proceso bastante complejo. Para ejemplificar tomemos las siguientes clausulas:

a) hombre ( Enrique ) V NO hombre ( Enrique )
b) hombre ( Enrique ) V NO hombre ( Fido )

La clausula "a" representa una contradicción sobre la cual se puede aplicar el algoritmo de resolución, mientras que "b" no es una contradicción. De modo que se necesita algun procedimiento para determinar cuando existe una contradicción.

Existe un algoritmo sumamente simple, llamado "algoritmo de unificación", el cual realiza esta función.

La idea del algoritmo, es bastente sencilla. Los elementos de la clausula, pueden ser representados por una lista, en donde el primer elemento es el nombre del predicado y los demás, son los argumentos. Ejemplificando:

odia ( Julio,César ) -> (odia Julio César)

El procedimiento consiste en lo siguiente.

Se va a tratar de unificar las dos clausulas comparando el primer elemento de cada lista, si este checa, la rútina se llamara recursivamente, eliminando de ambas listas el primer elemento que ya ha sido verificado.

La única complicación en este algoritmos, es que nosotros debemos de ser capaces de encontrar una substitución simple y consistente para la literal completa, y no separar las substituciones por cada elemento. Por ejemplo, supongamos que nosotros deseamos unificar las siguiente clausulas:

P(X,X) -> (P X X)
P(Y,Z) -> (P Y Z)

La primera literal no tiene problema, sin embargo en la segunda, nosotros podemos substituir Y/X*, se puede hacer esto ya que hay que recordar que los nombres de las variables son arbitrarios. Pero ahora si nosotros simplemente continuamos, tendriamos que comparar X con Z y producir la substitución Z/X, pero nos encotramos con un problema, ya que no podemos substituir "Y" y "Z" por "X" de modo que no hemos producido una substitución consistente.

Lo que es necesario hacer es, despues de haber realizado la primer substitución, realizar la misma en las literales remanentes:

```
(P X X) -> (X X) -> (Y X) -> (Y Y) (Y) (Z) ()
(P Y Z) -> (Y Z) -> (Y Z) -> (Y Z) (Z) (Z) ()
```

De forma que al unificar estas literales nosotros podamos producir la substitución Y/Z. Con esto el algoritmos de unificación ha concluido.

### 3.3.4 Algoritmo de Resolución.

Ahora que ya contamos con una rútina sencilla para determinar cuando dos literales son contradictorias, esto es si una de ellas puede ser unificada con el negado de la otra, por ejemplo hombre (X) con NO hombre (X). De modo que para usar el algoritmo de resolución en lógica de predicados, debemos de utilizar el algoritmos de unificación para localizar pares de clausulas que se cancelen.

También el algoritmo de unificación es utilizado en la creación de clausulas de resolución, regresando al ejemplo de la sección anterior:

```
invierno V verano
 NO invierno V frio
```

Al unificar invierno y NO invierno, producen la clausula vacia, quedando la clausula de resolución:

```
verano V frio
```

A continuación se dan los pasos para el algoritmo de resolución, dado un conjunto F de funciones y una función S a ser provada:

a) Convertir todas las F a forma de clausulas
b) Negar S y convertir el resultado a forma de clausula y agregarla al conjunto obtenido en el paso anterior.

c) Repetir hasta que:

- Una contradicción sea encontrada, o
- Ningun avance se logre, o
- Se halla invertido una cantidad predeterminada sin resultados.

1. Seleccionar dos clausulas, llamelas clausulas padre.

2. Resolverlas, la resolución debe de ser la disyunción de las literales de las dos clausulas padre con las subsituciones ejecutadas y con la siguiente excepción: Si existe un par de literales T1 y NO T2 de modo que T1 se encuentre en una de las clausulas padre y NO T2 en la otra y si T1 y NO T2 son unificables entonces ni T1 ni T2 deben de aparecer en la resolución. Usar la substitución producida por el algoritmo de unificación para crear la resolución.

3. Si la resolución es la clausula vacía, entonces una contradicción ha sido encontrada. Si no agregar la resolución al conjunto de clausulas disponibles.

Si la elección de las clausulas a resolver, es realizada sistemáticamente, este procedimiento va a encontrar una contradicción, si esta existe. Pero esto puede tomar demasiado tiempo de modo que puede ser muy útil tomar en cuenta alguna de las siguientes estrategias.

- Solo resolver pares de clausulas que contengan literales complementarias, ya que si esto no se cumple, la clausula producida es más dificil de satisfacer. Para facilitar esto, hay que indexar las clausulas por su predicado, combinando con una indicación de si se encuentra o no negado el predicado.

- Eliminar ciertas clausulas de las cuales se tenga la certeza que no van a participar en resoluciones posteriores, en este caso caen: Tautologias, las cuales nunca van a ser satisfechas y clausulas que son subconjuntos de otras.

- Cuando sea posible, resolver clausulas que contengan la instrucción que estamos tratando de refutar. Esta estrategia es llamada "Conjunto de soporte".

- Cuando sea posible, resolver clausulas con menos número de literales. Esto genera resoluciones con menos literales y más cerca de la solución que es aquella con cero terminos. Este método es llamado "Estrategia de preferencia unitaria".

Un ejemplo de como aplicar este método con el conjunto de clausulas generadas en la sección 3.3.2 es dado en la figura 3.3.

## 3.4 Razonamiento Nomonótono.

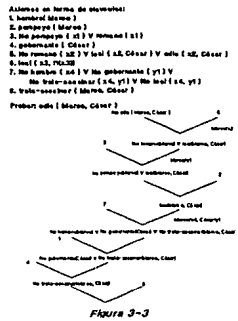


*Figura 3-3*

Sistemas tradicionales basados en lógica de predicados son monótonos en el sentido de que las instrucciones que se toman como ciertas, se incrementan en el tiempo. Nuevas instrucciones son agregadas al sistema y nuevos teoremas pueden ser provados, pero ninguno de estos eventos, puede causar nunca que conocimiento previo o instrucciones anteriores se vuelvan invalidas. Existen diversas ventajas en trabajar con un sistema asi, entre las cuales se encuentran:

- Cuando una nueva instrucción es agregada, no se realiza ningún chequeo para verificar que esta nueva regla no sea inconsistente con las que ya formaban parte del conocimiento.

- No es necesario recordar, para cada regla que ha sido provada, la lista de reglas en las cuales todavia hacen falta pruebas para quedar satisfechas, sencillamente este hecho no importa.

Desafortunadamente, los sistemas monótonos no son muy buenos en la solución de problemas del mundo real con las siguientes caracteristicas:

-Información incompleta,
-Situaciones muy dinámicas,
-Generación de asumpciones.

### 3.4.1 Razonamiento por Ausencia.

Muy pocas veces un sistema cuenta con toda la información que necesita. Sin embargo a menudo esta falta de información puede ser cubierta asumiendo algún valor lógico para esta información la cual va a ser tomada como cierta mientras no se encuentre evidencia de lo contrario.

Este tipo de procedimiento es conocido como *Razonamiento por ausencia*.

Ejemplificando, vamos a suponer la siguiente situación. Un hombre tiene su primer cita con una mujer, y ya que es la primera vez que van a intimar, este desconoce los gustos y preferencias de ella.

El le desea llevar un presente y partiendo del supuesto de que a la gran mayoria de las mujeres les gustan las flores y utilizando este razonamiento por ausencia de información, él puede decidir comprar flores y llevarselas como presente.

Esta clase de razonamiento por ausencia es no monótono, ya que el razonamiento efectuado depende de las creencias asumidas como ciertas.

Pero que pasaria si al momento en que el hombre arriba a la cita y le entrega las flores, ella comienza a toser.

La primera creencia que habíamos supuesto, la tendríamos que invalidar ante la prueba evidente de que en este caso en particular es falsa. Obviamente, si en nuestra base del conocimiento, existen otras reglas asociadas a la que vamos a descartar, estas tendrían también que ser descartadas.

El ejemplo anterior, ilustra una clase muy común de razonamiento por ausencia y es conocido con el nombre de "elección del más probable" y se basa en la idea que en la ausencia de información completa, siempre elegiremos la opción con más probabilidades de exito.

Otra clase común de razonamiento por ausencia es el que se conoce como "circunscripción" y funciona a partir de la idea de que solo pueden satisfacer una propiedad "P", aquellos objetos que han probado con anterioridad que la satisfacen.

### 3.4.2 Dependencia del Conocimiento en Sistemas No Monótonos.

Ya hemos analizado que los sistemas de razonamiento no monótonos son necesarios cuando el problema a atacar cuenta con alguna de las siguientes características:

- Información incompleta, lo cual requiere razonamiento por ausencia.
- Un mundo cambiante, lo cual requiere una base de datos cambiantes.
- La generación de una solución completa, requiere asumir información temporal para soluciones parciales.

Pero al trabajar nosotros con sistemas no monótonos es necesario contar con un sistema de actualización dinámica de nuestra base del conocimiento ya que el borrar una regla o instrucción puede tener un

efecto bastante significativo sobre toda la base de datos ya que existe una dependencia a tráves de los datos ahi almacenados.

Es por esto que al diseñar sistemas no monótonos uno de los aspectos básicos, es el garantizar que el sistema no consuma todos sus recursos en la propagación de cambios de la base de datos.

Esto implica, que dentro de la base de datos se deba de construir una red de dependencias y con esto mantener la consistencia del conocimiento almacenada en nuestra base de datos.

Un ejemplo de esto fue implementado por Doyle, 1979, en el subsistema TMS (Thrue Maintenance System). Este subsistema sirve para mantener la consistencia de la base de datos de un sistema de razonamiento.

En TMS cada regla es llamada un nodo y cada nodo puede estar en dos estados, estos son:

Entrada: Se cree que es cierto
Salida: Actualmente no hay manera de probar su validez.

Cada nodo cuenta con una lista de justificaciones, esto es otros nodos que sirven para establecer su validez.

Cuando el sistema detecta una inconsistencia, ejecuta un mecanismo de razonamiento, el cual realiza una búsqueda regresiva de dependencias, alterando con esto un conjunto minimo de reglas en la base de datos.

Para un estudio más formal de lógica no monótona, se puede consultar a (Mc. Dermott, 1980).

## 3.5 Razonamiento Estadístico y Probabilístico.

En la exposición de métodos que hemos hecho, se ha asumido, que una regla puede ser cierta o falsa. Ahora vamos a considerar la posibilidad de que algo sea *probablemente cierto*.

Existen tres situaciones en las cuales el uso de razonamiento estadistico es una buena alternativa, estas son:

- La situación bajo estudio es realmente aleatoria.

- La situación bajo estudio, no seria aleatoria si se contara con suficiente información, a la cual nuestro sistema, no

siempre tendra acceso.

- La situación bajo estudio, tiende a ser aleatoria, debido a que no podemos modelarla adecuadamente.

Para los primeros dos casos, el razonamiento probabilístico siempre será apropiado. Para el tercer caso, antes de decidir usar razonamiento probabilístico, nosotros debemos de buscar un modelo más adecuado de la situación bajo estudio.

A continuación, describiremos más ampliamente algunas técnicas especificas para representar y manipular información probabilística.

### 3.5.1 Técnicas Para Modelado de Situaciones Aleatorias.

La teoria matemática de probabilidad, provee un medio para describir y manipular conocimiento incierto. Algunas veces técnicas probabilisticas muy simples pueden ser usadas con eficacia en problemas de IA.

En las siguientes secciones, se asume que el lector cuenta con antecedentes en la Teoria de Probabilidad y Estadística.

#### 3.5.1.1 Técnicas Para la Solución de Problemas de Caracteristicas Aleatorias.

Consideremos la situación de decidir como jugar una partida de poker. Nuestro problema radica en que no contamos con información acerca de las cartas con las que cuentan nuestros oponentes. Asumamos que dado el juego con el que contamos, es crítico evaluar las posibilidades de cuales de nuestros oponentes cuentan con un as de corazones asi como cuantos diamentes tiene cada oponente.

Una manera de evaluar esta situación se detalla a continuación.

Para comenzar habría que calcular primero la probabilidad de que cada oponente tenga un as de corazones. En ausencia de información, podemos asumir que la probabilidad sea de 0.5. Despues, tendríamos que evaluar las probabilidades de cada una de las posibles distribuciones de diamantes. A continuación, se tendrían que relacionar las posibilidades de que un jugador cuente con un as de corazones y algún número particular de diamantes, usando para ello el hecho que:

prob( A y B ) = prob( A ) * prob( B )

si a y b son eventos independientes*

El siguiente paso es evaluar todas las posibles jugadas que

nosotros podemos realizar. Para esto primero debemos de asignarles un peso a cada jugada, por cada distribución de cartas. Posteriormente tenemos que multiplicar este peso por la probabilidad de que la distribución de cartas suceda y sumar estos productos, o sea:

$$\text{Perspectiva del Juego} = \sum_{i=1}^{n} \left( \text{prob}( i ) * \text{peso} ( i ) \right)$$

donde n es el número posible de jugadas.

Esto nos da una perspectiva de la posición promedio que nosotros podemos esperar si hacemos un movimiento en particular.

Esta simple manipulación estadística nos permite manipular situaciones efectivamente, cuando estas cuentan con cierto grado de incertidumbre en el conocimiento de el estado del problema.

Los mejores resultados han sido obtenidos con la utilizacion del *Teorema de Bayes*. Este teorema nos provee de un método para calcular la probabilidad de que ocurra un evento particular, dado un conjunto de observaciones.

El teorema establece lo siguiente:

$$P( H_i / E ) = \frac{P( E / H_i ) * P( H_i )}{\sum_{n=1}^{k} P( E / H_n ) * P( H_n )}$$

donde:

$P( H_i / E )$ : Es la probabilidad de que la hipótesis $H_i$ es cierta dada una evidencia E.

$P( E / H_i )$ : Es la probabilidad, de que observemos la evidencia E dado que la hipótesis $H_i$ es cierta.

$P( H_i )$ : Es la probabilidad *a priori* de que la hipótesis $H_i$ es cierta, en ausencia de cualquier evidencia.

k : Es el número de posibles hipótesis.

La utilización del teorema de Bayes ha jugado un papel de extrema importancia en problemas de reconocimiento de patrones, como visión, identificación de letra manuscrita, digitalización de imagenes, etcétera.

### 3.5.1.2 Técnicas Para la Solución de Problemas Deterministicos con Información Deficiente.

A menudo la solución de problemas completamente deterministicos, requieren el uso de técnicas probabilisticas debido a que en los pasos necesarios para la solución, a menudo la información necesaria para la solución, no se encuentra disponible. En estas situaciones, se usan ayudas heuristicas que representan información probabilistica para ayudar en la elección de la o las acciones a elegir.

Algunos sistemas basados en reglas, operan eligiendo la primera regla que cumpla con las condiciones vigentes en ese momento. El orden implicito de las reglas en la base de datos es una priorización, de forma que las reglas más deseadas se encuentran primero. En otras ocasiones, información probabilística es codificada en funciones heuristicas que evaluan cada nodo en forma individual. Otra opción es la de dividir el proceso de hacer una evaluación final, en pequeños pasos, en cada uno de los cuales hay que convinar algunas piezas de evidencia. Cada una de las pequeñas evidencias intermedias, deberan de ser combinadas posteriormente, asignandoles una valor que determina el grado de influencia, al formar parte de la decisión final.

Para ejemplificar este último sistema, supongamos un sistema de selección de medicamentos para un paciente enfermo. Las reglas de este sistema, podrian llevar la siguiente tonica:

*Si* hay evidencia de un organismo con propiedades X1, Y1
*Entonces*
con probabilidad de 0.7 existe una infección causada por el organismo Z.

Notese que para determinar la acción final puede usarse cualquier de los métodos provistos por la teoria de las probabilidades, desde probabilidad de ocurrencia de eventos independientes hasta el Teorema de Bayes.

### 3.5.2 Problemas con el Uso de Metodos Probabilisticos.

A pesar del poder de las técnicas probabilísticas, su uso implica riesgos, los cuales deben de ser considerados ya que pueden convertirse en barreras infranqueables.

La primera desventaja, radica en como convertir terminos humanos

a factores numéricos, ejemplificando, que factor numérico le podríamos asignar a una frase como: *Es deseable*.

Una segunda desventaja, se debe a la imposibilidad de normalizar distintos criterios humanos. Tomando el ejemplo de la regla para el sistema de medicamentos, un Medico podria confirmar que la probabilidad es 0.7, mientra otro podría rechazar esta cifra simplemente argullendo que según su experiencia la probabilidad debiera ser de 0.9, por ejemplo.

Otro problema se encuentra cuando al sistema se le provee con una nueva evidencia, ya que para mantener la coherencia de los factores probabilísticos, cuando una nueva evidencia es proporcionada, los factores deben de ser adaptados. Y si esta situación es frecuente el sistema podría ciclarse en la tarea de propagación de cambios a la base de datos.

Por último, tenemos el problema de la retroalimentación, ya que al ser situaciones dinámicas, el sistema debe de poder mantener una base de datos actualizados y con factores probabilísticos reales.

## 3.6 Representaciones Estructuradas de Conocimientos.

En las secciones anteriores, se han discutido una variedad de métodos para la representación de conocimiento, a tráves del uso de formalismos lógicos.

En esta sección, se van a mostrar algunos métodos de representación estructuradas del conocimiento. Cada una de ellas, cuenta con una estructura de datos, en la cual el conocimiento acerca de un dominio particular puede ser almacenado.

Un buen sistema para la representación del conocimiento en un dominio particular, debe de cumplir con cuatro propiedades, las cuales son:

*Adecuada Representación:*

Habilidad para representar todos los tipos de conocimiento que son necesarios para cualquier situación que abarque el dominio en cuestión.

*Adecuación Estructural:*

Habilidad para manipular la representación estructurada de modo que pueda crear nuevas estructuras, correspondientes a nuevo cono-

cimiento, inferido de hechos pasados.

*Inferencia Eficaz:*

Habilidad para incorporar dentro de las estructuras del conocimiento, información adicional, la cual pueda ser usada para enfocar la atención de los mecanismos de inferencia a las direcciones más promisorias.

*Adquisición Eficaz:*

Habilidad de adquirir nueva información facilmente. El caso más simple envuelve, inserción directa por parte de una persona. Idealmente, el programa por si mismo debe de ser capaz de controlar la adquisición de conocimiento.

### 3.6.1 Estructuras comunes del conocimiento.

Diversos objetos complejos pueden ser descompuestos en objetos más simples. Lo mismo sucede con clases de objetos.

Para poder llevar a cabo esta descomposición, podemos nosotros aprovechar dos caracteristicas muy importantes asociadas a cualquier objetos, estas son:

*Es un(a):*

Esta es una relación taxonómica entre objetos en una jerarquia, por ejemplo:

- Un perro *es un* mamífero,
- Un mamífero *es un animal*,
- Un animal *es un* objeto vivo.

*Es parte:*

Esta es una relacion entre objetos que son componentes de otros objetos en una relación, por ejemplo:

- Una mano *es parte* de un cuerpo,
- Un dedo *es parte* de una mano,
- Una falange *es parte* de un dedo.

Ambas relaciones, *es una* y *es parte*, se encuentran parcialmente ordenadas en sus respectivos dominios. Para diversos dominios, estas relaciones tienen al menos un valor superior, a menudo un concepto muy

general, como una entidad.

Una de las características mas importantes de ambas relaciones, es su transitividad, ya que si:

un dalmata *es un* perro, y
un perro *es un* animal, entonces
un dalmata *es un* animal.

Todas las representaciones que se van a discutir en las subsecuentes secciones, hacen uso de estas relaciones primitivas dentro de sus estructuras.

### 3.6.2 Redes Semánticas.

Este tipo de estructuras, fué desarrollada primeramente por [Quillian, 1968 y Raphael, 1968].

Las redes semánticas fuerón originalmente diseñadas como un medio para representar el significado de palabras en ingles. En una red semántica, la información es representada como un conjunto de nodos conectados a otros mediante un conjunto de arcos etiquetados, los cuales representan las relaciones existentes entre los nodos.

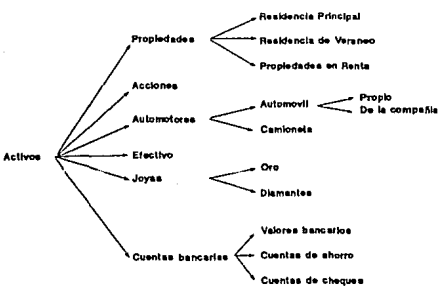



*Figura 3-4*

Las redes semánticas, se han convertido en las herramientas fundamentales para las representaciones del conocimiento, en los sistemas expertos actuales. Cada sistema experto comercial, ha desarrollado sus propios esquemas, la mayoria de ellos se encuentran basados sobre trabajos academicos desarrollados en los 70.

Una red semántica, o simplemente red, define y describe los terminos usados en una aplicación.

Estos terminos son representados como objetos estructurados, a menudo llamados esquematas o esqueletos, en la mayoría de los lenguajes de representación del conocimiento, y se encuentran conectados a tráves de ligas. En la figura 3-4, se muestra una porción muy pequeña de una red semántica.

Cada objeto de en la figura 3-4, es el nombre de un esquema. Algunas aplicaciones contienen cientos o incluso miles de esquematas.

Las relaciones indicadas por las flechas entre objetos en la figura 3-4 son llamadas ligas.

Gran parte del poder de las redes semánticas, se deriva del poder de las ligas. Las ligas en la figura, son ligas a subclases. En el diagrama de la red, ligas a subclases son dibujadas entre dos objetos de la misma clase, y van del objeto mas general a otro más especifico. Por ejemplo, si el objeto *Cuenta de Banco*, es de una clase determinada y, *Cuenta de Cheques*, es de la misma clase, nosotros podemos dibujar una liga de subclase de *Cuenta de Banco* a *Cuenta de Cheques*.

Las ligas, siempre tienen una dirección. Una liga de subclases, por definición siempre apunta de un objeto general a otro más específico. Obviamente, nosotros podemos tener ligas en dirección opuesta, esto es de un objeto específico a uno más general, estas son conocidas como ligas de superclase.

Usualmente, las ligas de subclases, implican que objetos más especificos, comparten propiedades. Por ejemplo, en la clase *Cuenta de Banco,* nosotros podemos almacenar información como balance o taza de interes. Una vez que nosotros asociamos esta información a la clase *Cuenta de Banco*, van a compartir la misma información.

Las redes semánticas, a parte de contener ligas a subclases y a superclases, deben de contar con ligas a instancias.

Una liga de instancia, siempre relaciona un esquema con un objeto en particular, el cual es una instancia de ese esquema. Ejemplificando, el número de cuenta bancaria de un cliente, es una instancia del esquema *Cuenta de Banco*. En general, una instancia es un objeto único, el cual no puede tener subclases.

Dependiendo de los sistemas en los cuales se utilize, las ligas a subclases son llamada ligas *"es_un(a)"*.

### 3.6.2.1 Razonamiento en Redes Semánticas.

Asi como cualquier otro mecanismo para la representación del conocimiento, el poder de las redes semanticas, radica en su habilidad para manipular los datos en la solución de problemas. Uno de los primeros usos que se le dio a las redes semánticas, fue el de encontrar relaciones a tráves de objetos, por medio de la activación de las ligas que existian entre ellos. Este proceso fue llamada búsqueda de intersección, [Quillian, 1968].

Usando este proceso, es posible usar la red para contestar preguntas tales como ¿ Cual es la relación entre un objeto *A* y otro

objeto *B* ?.

Aplicaciones más recientes, han usado las redes semánticas para procesos en los que son necesarias consultas más especificas de los objetos dados de alta en la red. Y esto es posible, ya que no existen restricciones en la forma en que los datos deben de ser representados en la red. La única restricción, radica en el hecho que para que el algoritmo de búsqueda implementado sea efectivo, debe de existir una completa consistencia en el significado de cada nodo y liga dentro de la red.

### 3.6.3 Dependencia Conceptual.

El esquema de dependencia conceptual, al cual en lo sucesivo se le denominada como DC, fue desarrollado como una teoria para representar el significado de oraciones de lenguaje natural, de modo que esta representación contara con las siguiente caracteristicas:

Facilitar los mecanismos de inferencia a tráves de las oraciones,

Independencia entre el lenguaje y el mecanismo de representación.

Una entidad básica en DC, es un evento o conceptualización, de modo que cuando en una instrucción, nosotros nos encontramos con el predicado evento, esto es una afirmación de que ese objeto es un evento dentro de la representación del conocimiento. Por ejemplo:

evento( ev-29 ( tarea Juan-22 ( ptrans libro-3 Alejandro Pepé )

si esta instrucción es cierta, nos indica que ev-29 es un evento, el cual consiste de Juan-22, y el predicado ptrans, este indica movimiento, del libro-3 de Alejandro a Pepé. En la instrucción, se pueden notar dos tipos de objetos. Los objetos ev-29, Juan-22, libro-3, Alejandro y Pepé, son denominados objetos de clase simbólo mientras que evento, tarea y ptrans son objetos de clase tipo. La clase tipo, puede ser conceputalizada como funciones, las cuales van a tomar determinadas acciones de acuerdo a como son definidas en la base del conocimiento. Para este ejemplo, podemos describir la acción de cada una de las funciones de acuerdo a:

evento: Entidad básica en la base de datos.

tarea: Consta de dos parámetros, el primero de ellos es un agente, mientras el segundo es una acción a ser ejecutada por el agente.

ptrans: Indica un movimiento fisico de un objeto entre dos

localidades.

Alternativamente, se puede usar otro medio para la representación de esta intrucción, tal y como se muestra a continuación:

```
instancia( ev-29 ptrans )
agente( ev-29 Juan-22 )
paciente( ev-29 libro-3 )
fuente ( ev-29 Alejandro )
destino( ev-29 Pepé )
```

Esta dos formas de representar la información, se encuentran fuertemente relacionadas, simplemente solo varian en su notación, lo cual hace que el pasar de una a otra representación sea sumamente fácil.

Ahora, como necesitamos la forma de representar el significado de oraciones dentro de un esquema del conocimiento, un punto importante es el poder distinguir si el evento bajo analisis, esta sucediento actualmente o si es un evento pasado o futuro.

De modo que nuestro evento ev-29, permanece hipotetico hasta que agregamos a la base de datos el siguiente hecho:

```
actual( ev-29 )
```

Dentro del esquema de DC, nosotros definiremos eventos primitivos, los cuales deben de definir acciones humanas de ocurrencia diaria. Uno es ptrans, el cual ya hemos definido, los otros serían:

```
mover ( parte-del-cuerpo destino )
```

Este evento se debe de utilizar para definir el movimiento de una parte del cuerpo hacia un destino. Es distinto a ptrans ya que requiere un solo actor.

```
atrans( objeto dueño-1 dueño-2 )
```

El control de un objeto es transferido de un dueño a otro.

```
propel( objeto lugar-1 lugar-2 )
```

Indica el movimiento de un objeto a tráves de una fuerza fisica directa, de un lugar a otro.

```
agarrar( objeto )
```

El objeto en cuestión es tomado.

ingerir( objeto orificio )
expeler( objeto orificio )

Estas dos formas, son usadas para describir las acciones de comer, beber, defecar, etc..

mtrans( info lugar-1 lugar-2 )

Información es transferida entre dos entidades fisicas o "mentales". La información puede ser diversa, formulas o simbolos.

mconst( formula )

El actor formula una creencia. Esta forma es usada para describir oraciones tales como "*Alejandro provo un nuevo teorema*".

hablar( sonido )

Hacer un sonido. A menudo el sonido, va a ser un conjunto de fonemas en algun mensaje. Este es un camino usual de mtrans*ferencia*.

atender( organo-sensor objeto )

Un objeto, recibe atención de parte de algún organo sensor. Por ejemplo atender( ojo libro-23 ) denota la acción de "*leer el libro-23*''.

En adición a estas funciones para describir eventos, existen funciones para describir estados. Estos estados pueden ser ciertos o falsos en un instante de tiempo, a menudo como resultado de algún evento. Cuando nosotros tratamos de expresar un estado como una proposición, por ejemplo feliz( Alberto-30 ), esta puede fallar ya que no especifica un periodo de tiempo durante el cual Alberto-30 es feliz. Una mejor idea es hacer una relación con algún periodo de tiempo, algo como estado( est-40 feliz( Alberto-30 muy )). En esta función, nosotros representamos el hecho que durante el periodo que dura el estado est-40, Alberto se encuentra en un estado feliz, con un calificativo de muy. Este tipo de representación, nos va a permitir representar el mismo estado de animo, con muy distintos grados. En DC, diversos estados pueden ser representados de la forma atributo( objeto valor ).

Ahora que tenemos dos clases de elementos, eventos y estados, nosotros debemos de hacer distinciones entre ellos, ya que ambos poseen propiedades distintas.

Un elemento estado es cierto sobre un espacio de tiempo, esto es, tiene un inicio un medio y un fin. Mientras que un elemento evento es

instantaneo. Nosotros refenciaremos a estos elementos como de tiempo.

Tambien debemos de hacer una distinción entre estados reales y posibles. Usaremos la notación (Actual elemento), para representar que un evento es real en un tiempo dado, para referirnos a estados usaremos la notación (Estado-actual elemento estado). Dos funciones requieren atención especial, estas son *loc* y *mloc*, las cuales se refieren a localidades fisicas y mentales respectivamente. Loc (ob loc) es el estado del objeto *ob* que se encuentra en la localidad *loc*. Mloc (formula, loc), es el estado de *formula*, que se encuentra en la localidad mental *loc*. Por ejemplo Mloc (idea-20 (cabeza Juan-2)), significa que la *idea-20* se encuentra localizada en la cabeza de *Juan-2*. Asi como *loc* es el resultado lógico de *ptrans*, *mloc* es el resultado de *mtrans* y *mbuild*.

Además de las funciones anteriores, nosotros tambien requerimos de eventos que indiquen cambios de estado. Asi que usaremos la función Inicia (estado), esta función indica que el estado va a pasar de un valor de falso a un valor de verdadero. Por ejemplo Inicia (feliz Juan-22 muy), significa que Juan-22 empieza a sentirse muy feliz. Tecnicamente, nosotros no podemos representar el hecho de que Juan-22 ya no se encuentra muy feliz como (No (feliz Juan-22 muy)), ya que *no* es conectivo y transforma una proposición en su negación. Por lo cual nosotros necesitaremos la función *Estado-no*. Esta función toma un estado como argumento y denota que el estado toma un valor de falso si y solo si este estado tenia un valor de verdadero. De aqui se desliga la necesidad de funciones como *Estado-y*, *Estado-o*, etcétera.

En adición a las funciones de tiempo antes descritas, también pueden existir relaciones entre elementos. Por ejemplo un evento puede preceder a otro o por el contrario puede ocurrir al mismo tiempo. De esto se desligan las siguientes funciones.

Precede (evento-1 evento-2)
Mismo-tiempo (evento-1 evento-2)

Más aun, nosotros podemos tener:

Instr (evento-1 evento-2)

Lo cual significa que *evento-2* fue el medio por el cual *evento-1* ocurrio, o sea, que *evento-2* fue el instrumento de *evento-1*.

También, nosotros debemos de contar con una función para indicar subconjuntos de eventos, esta es:

Subevento (evento-1 evento-2)

esta función nos dice que *evento-1* es un subevento de *evento-2*.

Un evento puede originar que un estado cambie su valor de falso a cierto. Luego, un estado que cambia de valor, puede originar que ciertas acciones ocurran, asi que para representar estos hechos, nosotros usaremos las funciones resultado y habilita, las cuales se muestran a continuación.

Resultado (evento estado)
Habilita (estado evento)

Luego, un evento puede prevenir la ocurrencia de otro evento, de manera que la siguiente función representa este hecho:

Previene (evento-1 evento-2)

En esta función *evento-2* no puede ocurrir, ya que la ocurrencia de *evento-1* lo impide.

Del mismo modo, nosotros podemos deducir, que necesitamos alguna función para indicar el inicio de eventos, que originen estado mentales, para ejemplificar usaremos la siguiente secuencia:

```
inicio (ev-3 ev-4)
estado-actual (ev-3 (quemando casa-42 rápido))
estado-actual (ev-4
  (tarea juan-22
   (mtrans(estado-actual ev-3(quemando casa-42 ráp))
      (cabeza juan-22)
          departamento-de-bomberos-5)))
```

Este ejemplo, sale del parrafo "*Juan esta tratando de llamar al departamento de bomberos ya que la casa se esta incendiando*".

Finalmente, tenemos la siguiente función:

razón (estado-mental evento)

lo cual significa que la razón de la ocurrencia de *evento* es el *estado-mental* descrito.

Para ilustrar aun más el uso de DC, estudiese a fondo la representación de una historia simple, que se muestra en la figura 3-5.

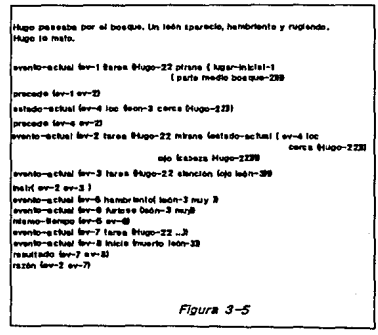



*Figura 3-5*

Se muestra un resumen de las funciones antes descritas en la tabla 3-1.

3.6.3.1 Razonamiento en Dependencia Conceptual.

Son tres los modos en los cuales el modelo de DC, facilita el razonamiento.

1) Son necesarias menos reglas de inferencia, ya que el conocimiento se encuentra particionado en pequeños eventos primitivos.

2) Muchas de las reglas de inferencia son incluidas en la representación en si misma.

3) La estructura inicial que es construida para representar la información contenida en una oración puede contener huecos que necesiten ser llenados. Estos huecos pueden servir como focos de atención, y asegurar con esto el entendimiento de la oración por parte del programa.

Cada uno de estos puntos se amplia a continuación.

El primer argumento en favor de la representación del conocimiento en terminos de primitivos de DC, más que en terminos de estructuras de alto nivel normalmente usadas, es que el uso de primitivos, facilita el describir las reglas de inferencia por medio de las cuales el conocimiento va a ser manipulado. Las reglas necesitan solo ser representadas por cada acto primitivo, en lugar de por cada palabra o acción que describa un acto. Por ejemplo, cada uno de los siguientes verbos describe una transferencia de poseción de un objeto:

- Dar
- Tomar
- Robar
- Donar

Si cualquiera de ellas ocurre, entonces los actos de inferencia acerca de quien tiene ahora el objeto, quien lo perdio y en si información importante de esta transferencia, son ahora relevantes. En una representación de DC, todas las posibles inferencias estarán asociadas con el primitivo de ACT Atrans, (actos de transferencia).

| *Función* | *Tipos de Argumentos* | *Tipo de Valor* |
|---|---|---|
| hacer | (hacer agente acción) | tipo-even |
| inicia | (inicia estado) | tipo-even-cambio-edo |
| mloc | (mloc formula loc) | estado |
| loc | (loc objeto loc) | estado |
| tarea | (tarea agente acción) | estado-mental |

| *Predicado* | *Tipos de Argumentos* |
|---|---|
| evento | (evento elemento-even tipo-even) |
| ocurre | (ocurre elemento-even) |
| evento-actual | (even-act elemen-even tipo-even) |
| estado | (estado elemen-estado estado) |
| actual | (actual elemen-estado) |
| estado-actual | (edo-act elemen-estado estado) |
| precede | (prec elemen-even1 elemen-even2) |
| mismo-tiempo | (m-tm elemen-even1 elemen-even2) |
| instr | (inst elemen-even1 elemen-even2) |
| subevento | (sube elemen-even1 elemen-even2) |
| resultado | (res elemen-evenp elemen-evenp) |
| habilita | (hab elemen-cambio elemen-evenp) |
| previene | (prev elemen-even1 elemen-even2) |
| inicia | (ini elemen-t elemen-edo-mental) |
| razón | (raz elemen-mental elemen-evenp) |

*Tabla 3-1*

El segundo argumento en favor del uso de DC, es que al construir la representación, es que nosotros no solo debemos de usar la información explicita que aparece en una oración, si no que también debemos de asociar las reglas de inferencia, con la información especifica de la oración. Una vez que las reglas de inferencia fuerón aplicadas a la oración, el resultado es almacenado como parte de la representación, lo cual nos va a permitir usar el conocimiento repetidas veces sin necesidad de volver a aplicar las reglas de inferencia de nuevo.

El tercer argumento para el uso de DC, radica en el hecho de que elementos no especificados en la representación de una pieza de información, pueden ser usados por el programa como focos de atención para el entendimiento de eventos posteriores, si son encontrados.

Como en cualquier modelo, también este esquema de representación, cuenta con algunas desventajas, las cuales deben de ser tomadas en cuenta. Al tener que descomponer el conocimiento, en pequeños primitivos de bajo nivel, el manejo de ellos se puede volver muy ineficiente, y en algunos casos imposible.

Otra desventaja, estriba en que el modelo se basa en representación de eventos, y en un mundo tan complejo, esta tarea resulta titánica si se desean modelos generales. A parte de que a menudo se desean representar otro tipo de objetos que no pueden ser considerados como eventos.

Para un estudio más a fondo de este tema, se puede consultar a [Schank, Roger C., Conceptual Information Processing, Amsterdam 1975].

### 3.6.4 Frames'.

Hasta este momento, hemos presentado mecanismos que pueden ser usados para representar eventos especificos o experiencias. Pero siempre ha existido un particular interes, por dado un conocimiento y una nueva situación, analizar ambos y construir con esto un nuevo conocimiento que tome en cuenta la experiencia adquirida de esta nueva situación. Un mecanismo diseñado para la representación en computadora de conocimiento es el frame. Este tipo de estructuras usualmente puede ser visto como complejas redes semánticas, pero estas representaciones tipicamente, tienen una gran cantidad de estructura interna diseñada para ser de utilidad en tareas especificas de solución de problemas.

#### 3.6.4.1 Representación del Conocimiento.

Tipicamente, un frame describe clases de objetos, como podrian ser *SILLA* o *CUARTO*. Consiste en una colección de "tarjetas", las cuales

'Se prefirio el uso de la palabra en ingles al no encontrarse una traducción adecuada o equivalente.

describen caracteristicas del objeto. Estas tarjetas son llenadas por otros frames, los cuales describen otros objetos. Asociados a cada tarjeta, pueden existir un conjunto de condiciones para el llenado de la tarjeta. Adicionalmente cada tarjeta puede ser llenada con un valor de default, o sea, en ausencia de información especifica, la tarjeta puede ser llenada con información que se asuma al momento del llenado.

Además, información de procedimientos puede estar tambien asociada con algunas tarjetas en particular. Por ejemplo, a menudo es útil describir los pasos a realizar cuando una tarjeta va a ser llenada (comunmente llamados procedimientos *if-added*), o como debe de ser calculado el valor de un campo en una tarjeta, (normalmente llamados procedimientos *if-needed* o *to-establish*).

La gran mayoria de los sistemas, explotan no solo uno, sino diversos frames. Frames relacionados pueden ser agrupados para formar sistemas de frames.

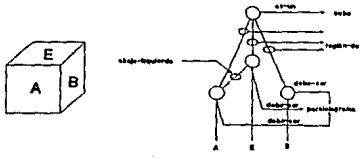



*Figura 3-6*

Un ejemplo sencillo, tomado de [Minsky, 1975], se muestra en la figura 3-6. Este frame representa una vista de un cubo. La liga *es-un* del nivel superior, apunta a "CUBO". Al siguiente nivel, cada nodo describe una cara del cubo. Las ligas *debe-ser*, apuntan a "PARALELOGRAMA" y describen el conjunto de valores que deben ser llenados en la tarjeta que cada nodo describe. Las ligas al siguiente nivel, asocian cada cara de esta vista en particular, con los nodos descritos en el parrafo anterior.

Pero, para la mayoria de las aplicaciones, es de suma importancia el poder considerar un objeto desde diversos puntos de vista. Para poder hacer esto, se necesita un sistema de frames. Un ejemplo muy sencillo de un sistema de este tipo, tambien tomado de [Minsky, 1975], es mostrado en la figura 3-7.

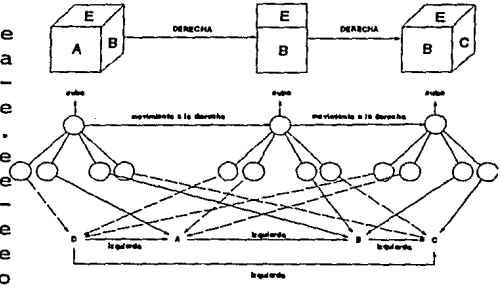



*Figura 3-7*

Este sistema consiste de tres frames, cada uno de los cuales representa, una perspectiva de un cubo. Algunos de los detalles han sido omitidos de la figura con el objeto de conservar la claridad del ejemplo. Las lineas solidas de las tarjetas, representan caras que se pueden observar desde la perspectiva en cuestión. Las lineas punteadas, indican caras ocultas del cubo desde la vista de referencia.

Las ligas entre los frames, indican la relación entre las perspectivas que cada frame representa. En adición a estas ligas explicitas entre los nodos en si mismo, los frames se conectan a tráves de nodos compartidos que representan tarjetas. Estos nodos compartidos describen cada uno una imagen independiente de una cara del cubo que ellos representan. En este ejemplo, estos nodos se representan como caras simples, sin embargo en problemas más complejos, podrian estar representados por estructuras más complejas, incluso podrian ser otros sistemas de frames. Sencillamente cada tarjeta de un frame puede ser compartida por otra estructura de frames, lo cual evita la redundancia de información ya que información común a varias perspectivas es compartida por todo el sistema.

### 3.6.4.2 Razonamiento en Frames.

Los Frames son utiles por el hecho de que hacen fácil la tarea de inferir caracteristicas no observadas en nuevas situaciones. Esta tarea se facilita por una variedad de situaciones, entre las cuales tenemos:

- Los frames contiene información acerca de diversos aspectos de los objetos o situaciones que ellos describen. Esta información puede ser usada aun sin haber sido explicitamente observada. Por ejemplo, un programa que accesa el frame de un habitación, puede inferir la existencia de al menos una puerta, a pesar de que no exista evidencia explicita de su existencia. Esto sucede ya que el frame *habitación* contiene la descripción de una habitación y esta descripción incluye el hecho de que al menos una puerta debe de estar presente.

- Los frames contienen atributos de los objetos que van a ser usados para llenar tarjetas individuales, estos atributos deben de ser necesariamente ciertos. Construir una descripción de una situación particular, requiere construir descripciones de las situaciones que la componen. La información asociada con las tarjetas del frame para la situación puede ser usada como una guía de como construir las situaciones que la componen.

- Los frames describen instancias tipicas de los conceptos que ellos representan. Sin una situación en particular bajo estudio, se parece mucho al frame, pero contiene alguna diferencia sustancial, esta diferencia va a servir como foco de atención al programa para que sea estudiada a fondo, ya que de ella se pueden obtener datos adicionales de vistas no observadas de la situación descrita en el frame.

Los frames, al igual que las redes semánticas, son estructuras de proposito general, dentro de las cuales, conjuntos particulares de conocimiento de dominio especifico, puede ser almacenado. Los detalles de la operación de los sistemas basados en frames, varian con las clases especificas de conocimiento que contienen y con la clase de razonamiento que el sistema debe de ejecutar.

Para un estudio más a fondo de los fundamentos de este tipo de estructuras, se puede consulta a [Minsky, M., "A Framework for Representig Knowledge" en "The Psychology of Computer Vision", P. Winston (Ed.), Mc Graw Hill, N.Y. 1975].

### 3.6.5 Manuscritos.

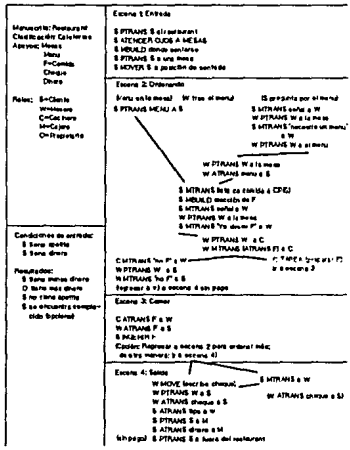


*Figura 3-8*

Un manuscrito es una estructura que describe una secuencia de eventos estereotipados en un contexto particular. Para los fines que en esta obra se persiguen, un manuscrito estara compuesto de un conjunto de tarjetas. Asociada a cada tarjeta, podra existir información acerca de la clase de valores que esta puede contener, asi como, un valor por ausencia, el cual sera usado cuando no exista otra clase de información disponible.

Se puede observar, que esta definición, es muy similar a la definición dada a los frames en la sección anterior, más aun, a este nivel de detalle, ambas estructuras son identicas. La diferencia entre ambas, estriba en el hecho de que un manuscrito, necesariamente debe de ser muy especializado, al contrario de un frame, que es una estructura de proposito general.

3.6.5.1 Representación del Conocimiento en Manuscritos.

En la figura 3-8, se muestra parte de un manuscrito tipico. Este manuscrito representa a un restaurant, y fue tomado de [Schank, 1977].

A continuación se ilustran los principales componentes de un manuscrito.

Condiciones de Entrada:

Conjunto de condiciones que en general deben de ser satisfechas, antes de que los eventos en el manuscrito puedan ocurrir.

Resultado:

Conjunto de condiciones que en general deben de estar satisfechas, despues que los eventos descritos en el manuscrito hayan ocurrido.

Apoyos:

Tarjetas, representando objetos que estan envueltos en la descripción de eventos en el manuscrito. La presencia de estos objetos, puede ser inferida aun si no son mencionados explicitamente.

Papeles, (Roles):

Tarjetas, representando a la gente que se encuentra envuelta en la descripción de eventos del manuscrito. La presencia de estos personajes, también puede ser inferida aunque no sean explicitamente mencionados. Si un individuo especifico, es mencionado, este debera ser insertado en la tarjeta correspondiente.

Clasificación:

La variación especifica sobre un patrón más general del manuscrito en uso. Diferentes clasificaciones del mismo manuscrito, pueden compartir la mayoria, más no todos, los componentes del manuscrito.

Escena:

La secuencia actual de eventos que ocurren. Los eventos son presentados en formalismo de dependencia conceptual.

Los manuscritos son útiles, ya que, en el mundo real, existen patrones de ocurrencia de eventos. Estos patrones surgen, debido a las

relaciones causales que existen entre los eventos. Los agentes pueden ejecutar una acción, lo cual les posibilitará el ejecutar otras acciones posteriores.

Los eventos descritos en un manuscrito, forman una enorme cadena causal. El inicio de la cadena, es el conjunto de condiciones de entrada, las cuales permitiran que el primer evento del manuscrito ocurra. El final de la cadena, es el conjunto de resultados, los cuales permitiran que eventos o secuencias de eventos posteriores, posiblemente descritos por otros manuscritos, ocurran. En medio de la cadena, la ocurrencia de eventos anteriores, habilita la posibilidad de ocurrencia de eventos que le siguen.

3.6.5.2 Razonamiento en Manuscritos.

Si un manuscrito en particular se conoce que es apropiado para una situación especifica, su uso puede ser muy útil para predecir la ocurrencia de eventos que no son explicitamente mencionados. Los manuscritos también son útiles al indicar como los eventos que son mencionados, se relacionan unos con otros. Pero antes de que un manuscrito en particular pueda ser usado, este debe ser activado. Existen dos razones por las cuales puede ser útil activar un manuscrito, estas son:

Manuscritos de breve uso, (aquellos que son mencionados brevemente y pueden ser usados otra vez, pero que no son parte fundamental de la situación central), es suficiente con almacenar un apuntador a estos manuscritos, por si vuelven a ser referenciados posteriormente.

Manuscritos de uso completo, esto es cuando un manuscrito llena completamente los requerimientos para ser usado en una situación especifica. El encabezado del manuscrito ( sus precondiciones, localidades, clasificasión, papeles y eventos) pueden servir como indicadores de que el manuscrito debe de ser activado. En orden de evitar una fragmentación excesiva, a probado ser útil el requerir que al menos dos de los componentes del encabezado del manuscrito sean satisfechos antes de activar el manuscrito.

Una vez que un manuscrito ha sido activado, existen una variedad de caminos por medio de los cuales se puede interpretar una situación en particular. La más importante, es la habilidad de predecir eventos que no han sido explicitamente observados. Ejemplificaremos con la siguiente historia:

Alex fue al restaurant la noche anterior. El ordeno un asado. Cuando el pago, noto que se habia quedado sin dinero. Alex se preocupo ya que comenzaba a llover.

Si nos hicieran la pregunta:

¿ Alex ceno la noche anterior ?

nosotros podemos responder con certeza que el lo hizo. Pero usando un programa de computadora basado en manuscritos, el programa activaria el manuscrito del restaurant y podria inferir que la secuencia entera predecida por el manuscrito ocurrio normalmente, de modo que podria concluir que Alex ceno. En esta habilidad para predecir eventos no observados, los manuscritos son iguales a los Frames, y a otras estructuras que representan situaciones estereotipadas. Una vez que esta estructura es activada, diversas predicciones pueden ser hechas.

Una segunda característica importante de los manuscritos, es que proveen de un modo de construir interpretaciones coherentes a partir de un conjunto de observaciones. Consideremos la siguiente historia:

Alex salio a comer, el se sentó en una mesa y llamo al mesero. El mesero se acerco a el y le mostro el menu, el ordeno una hamburguesa.

Ahora consideremos la pregunta:

¿ Porque el mesero le mostro el menú a Alex ?

El manuscrito provee de dos posibles respuestas.

Porque Alex se lo pidio. (Esta respuesta se obtiene usando búsqueda regresiva en la cadena causal).

Porque Alex decidio que queria comer. (Esta respuesta se obtiene usando búsqueda progresiva en la cadena causal).

Una tercer razón, por la cual los manuscritos son útiles, es porque pueden afocar eventos inusuales. Consideremos la siguiente historia:

Alex fue al restaurant. El se sento en una mesa y ordeno un bisteck. El estuvo sentado por un largo rato, se enojo y se marcho.

La importancia de esta historia, es que el final difiere de la

secuencia de eventos esperada en el restaurant. Alex se enojo porque tuvo que esperar para ser atedido. Una vez que una secuencia de eventos es interrumpida, el manuscrito no puede ser usado para predecir eventos, de modo que en este caso el manuscrito no podría responder a la pregunta ¿ Pago Alex la cuenta ?, pero podría inferir que Alex vio el menú, ya que el evento de lectura de menú sucedio forzozamente antes de la interrupción.

De estos ejemplos, nosotros podemos observar como información a cerca de secuencia típicas de eventos pueden ser representada en manuscritos, y ser interpretada a tráves de este tipo de estructuras.

A pesar de que los manuscritos son estructuras menos generales que los Frames, y por lo tanto no puede ser usada para representar conocimiento general, son muy útiles para representar conocimiento de dominio especifico, para el cual fue diseñado.

Para un estudio más profundo de este tipo de estructuras se puede consultar a [Cullingford, 1981].

## 3.7 El Espectro de Representaciones Sintáctico-Semántico.

Para Concluir este capítulo, es útil revisar los esquemas de representación que nosotros hemos descrito hasta el momento, considerando una importante dimensión, por medio de la cual se puede caracterizar a cada uno de ellos. En un extremo son sistemas puramente sintácticos, en los cuales no importa el significado del conocimiento que se encuentra en el almacenado. Estos sistemas tiene simples reglas de manipulación de la representación. No se debe de tener especial cuidado en la información que la representación contiene. En el otro extremo se encuentran sistemas puramente semánticos, entre los cuales no existe una uniformidad. Cada aspecto de la representación corresponde a diferentes piezas de información y por consiguiente las reglas de inferencia son complicadas.

Hasta el momento, hemos discutido cinco estructuras declarativas, en las cuales el conocimiento puede ser representado, estas son:

- Lógica de Predicados
- Redes Semánticas
- Frames
- Dependencia Conceptual
- Manuscritos

De estos esquemas, lógica de predicados es el más puramente sintáctico. Sus reglas de inferencia son procedimientos estrictamente

sintácticos que operan sobre reglas bien estructuradas, en las cuales no importa demasiado el significado.

Las redes semánticas, como su nombre lo implica, son estructuras semánticas orientadas. Estas estructuras son comunmente usadas con un conjunto de reglas de inferencia que son especialmente diseñadas para manejar correctamente los tipos especificos de arcos presentes en la red. Por ejemplo, las ligas *es un* son tratadas diferente de un esquema a otro, y con respecto a las otras clases de ligas.

Los Frames son estructuras que se encuentran más orientadas al lado semántico que al sintáctico. Mientras que las redes semánticas son una simple colección de nodos y arcos, los frames son estructuras de objetos más complejas, especificamente diseñadas para permitir la representación de características comunes de objetos. Por ejemplo, los Frames pueden contener descripciones de valores por ausencia que pueden ser utilizados cuando conocimiento más especifico se encuentra auscente. Los procedimientos para manipular Frames son similarmente más estructurados para poder usar de un modo eficaz el conocimiento que esta representado en el Frame.

El esquema de Dependencia Conceptual, también es una estructura más semántica que sintáctica. Esta forma no solo provee una estructura abstracta de la representación, si no además, una indicación especifica de los componentes que la representación debe contener ( como actos primitivos y las relaciones de las dependencias). Luego como las representaciones de DC, pueden ser vistas como instancias de redes semánticas, pueden ser usadas por mecanismos más poderosos de inferencia que exploten el conocimiento de dominio especifico almacenado en la estructura.

Similarmente, los manuscritos son muy parecidos a los Frames, de hecho son frames en los cuales las tarjetas han sido minusiosamente elegidas para representar información seleccionada que es útil en razonamiento de situaciones muy especificas. Esto hace posible, que los procedimientos de manipulación de manuscritos que usan el conocimiento, puedan trabajar en la solución de problemas de una manera muy eficiente.

En general las representaciones sintácticas son a la representación del conocimiento, lo que los metodos débiles son a la solución de problemas. Esto es, en principio, son adecuados para cualquier problema. Pero para problemas complicados, generalmente las respuestas no seran obtenidas de una forma rápida. Los metodos semánticos son más poderosos y hacen un uso más eficiente del conocimiento. La decisión de que usar, dependerá del problema en si mismo.

# CAPITULO IV

# LENGUAJE NATURAL

## 4.1 Introducción.

La habilidad para comunicarse en algún tipo de lenguaje natural, como puede ser el Español o el Ingles, a menudo parece ser una cualidad única de la raza humana. Las computadoras aún no son capaces de realizar diversas tareas que el ser humano realiza día tras día, como la habilidad de usar un lenguaje. Por ejemplo, un niño de tres años, no es capaz de llevar a cabo un juego legal de ajedrez y mucho menos ser un experto en algún topico. Sin embargo si es capaz de hablar y entender su lengua nativa.

El entender lenguaje natural es dificil por dos situaciones en especial. La primera de ellas radica en el hecho de que se requiere conocimiento ligüistico del lenguaje a usar y la segunda, estriba en que también se requiere conocimiento del tópico a discutir.

En este capítulo, analizaremos el conocimiento lingüistico que es necesario para el entendimiento de lenguaje natural, asi como la forma en que el conocimiento del mundo y lingüistico pueden ser combinados para producir un programa efectivo que entienda lenguaje natural.

## 4.2 Breve Historia.

Diversos programas han sido desarrollados en las decadas de los 50,60 y 70's, con el fin de procesar lenguaje natural. En sus bases, los programas que trabajan con algún aprovechamiento de lenguaje natural, pueden ser divididos en cuatro categorias historicas.

Los primeros programas, lograrón trabajar con resultados limitados, en dominios especificos y muy restringidos. Estos programas como BASEBALL de Green, SAD-SAM de Lindsay, STUDENT de Bobrow y ELIZA de Weizanbaum, usaban estructuras de datos especialmente diseñadas para almacenar caracteristicas a cerca de su dominio limitado. Las frases de entrada, eran restringidas a formas declarativas e interrogativas, estas frases eran filtradas para encontrar palabras clave o patrones previamente definidos, los cuales servían para buscar objetos conocidos y relaciones. Reglas de dominio especifico, llamadas heuristicas, eran usadas para encontrar la respuesta deseada en base a la palabra clave

en la frase de entrada y el conocimiento almacenado en la base de datos. Debido a lo restringido del dominio con el cual trabajaban, estos primeros sistemas ignoraban muchas de las complejidades del lenguaje.

La segunda categoria, se refiere a un aprovechamiento más reciente, el cual fue intentado en PROTO-SYNTEX-I de Simmons, Burger y Long y en Memoria Semántica de Quillian. Estos sistemas, esencialmente almacenaban un representación de un texto en si mismo en una base de datos, usando esquemas de indexación de claves para extraer información que contuviera una frase o palabra especifica. En este aprovechamiento basado en textos, la construcción no estaba enfocada a un dominio especifico, sencillamente el texto almacenado en la base de datos, podia ser de cualquier tema. Pero estos sistemas cuentan con diversas restricciones, entre las cuales se encuentran que el sistema solo puede responder sobre material previamente almacenado y la principal es que no toman en cuenta el *significado* de los parrafos que almacenan.

En la tercera categoria, se trato de manejar el problema de como caracterizar y usar el significado de las frases que entraban a los sistemas. En estos sistemas de lógica limitada, se incluyen SIR de Raphael, TLC de Quillian, DEACON de Thompson y CONVERSE de Kellogg. La información en la base de datos, era almacenada en algún tipo de notación formal y fuerón provistos mecanismos para trasladar las frases de entrada a la notación formal. La meta global de estos sistemas, era ejecutar inferencias en la base de datos, de modo de encontrar respuestas a preguntas que no se encontraban explicitamente almacenadas. Por ejemplo si un sistema sabia que Fido era un Collie y que todos los Collies son perros, el sistema podia contestar preguntas como ¿ Es Fido un perro ? Los sistemas de este periodo, eran limitados, en el sentido de que las deducciones que podían hacer, eran sobre un subconjunto de las inferencias lógicas usadas en una conversación ordinaria.

El cuarto grupo de sistemas, es llamado *Sistemas Basados en el Conocimiento*, su desarrollo se encuentra fuertemente interrelacionado con las investigaciones de IA sobre la representación del conocimiento, (capítulo III). Estos sistemas usan una gran cantidad de información acerca del dominio en discusión para ayudar al sistema a entender frases. El conocimiento es almacenado usando algún esquema de los ya analizados, como lógica de predicados, redes semánticas o frames.

### 4.2.1 ¿ Que es el "entendimiento" ?

El *entender* algo, es tranformarlo de una representación a otra, donde esta segunda representación ha sido escogida para corresponder a un conjunto de acciones disponibles que deben de ser ejecutadas y donde

el mapeo ha sido diseñado para que por cada evento, una acción apropiada sea ejecutada. Existe muy poco de absoluto en la noción de *entendimiento*. Si a un sistema de base de datos de una aerolinea, le decimos "Necesito ir a Monterrey, lo más rápido posible", el sistema lo interpretará como si deseara encontrar el primer avión disponible. Sin embargo, si lo mismo le decimos a nuestro mejor amigo y el sabe que la familia vive en Monterrey, el amigo *interpretará* que probablemente exista algún problema con la familia y que necesitamos que nos conforte.

Siempre que nos refiramos a sistemas que *entienden*, debemos tener en cuenta, que muy raramente podemos evaluar a estos sistemas en terminos absolutos.

Existe un sentido formal, en el cual el lenguaje puede ser definido simplemente como un conjunto de cadenas de simbolos, sin relación con ningun mundo a describir o una tarea a ejecutar. Aunque algunas de las ideas del estudio formal de los lenguajes pueden ser usadas en partes del proceso de *entendimiento* estas son solo el principio. Para obtener una vista global, necesitamos pensar en el lenguaje como un par (lenguaje fuente, representación objeto), además de un mapeo de un elemento del par al otro. La representación destino debe ser diseñada de acuerdo a un propósito efectivo que tengamos en mente. A menudo, si la tarea a ejecutar esta lo suficientemente clara, y los detalles de la representación destino no son importantes en una discusión en particular, nosotros solo hablaremos del lenguaje en si mismo, pero es importante recordar siempre que la otra parte del par, se encuentra presente.

### 4.2.2 ¿ Que es lo que dificulta el "entendimiento" ?

Existen tres razones principales, por las cuales el problema de *entender*, se dificulta, estas son:

La complejidad de la representación destino contra la cual la comparación va a ser efectuada.

El tipo de mapeo: uno-uno, muchos-uno, uno-muchos, o muchos-muchos.

El nivel de interacción de los componentes de la representación fuente.

A continuación ilustraremos la importancia de cada uno de estos factores.

Suponiendo una frase en español, para comunicarse con un sis-

tema de extracción de datos en base a palabras clave. si tuvieramos la siguiente frase:

Quiero leer todo acerca de la última elección presidencial.

tendría que ser convertida en una representación similar a:

(BUSCA CLAVE=ELECCION ∧ PRESIDENTE)

Pero ahora supongamos que las frases van a ser usadas como entrada a un sistema, cuyo proposito es grabar eventos, de modo que este sistema pueda contestar una gran variedad de preguntas acerca de los eventos y sus relaciones. Por ejemplo, consideremos la siguiente historia:

Guillermo le dijo a Adriana, que el no podía ir al cine con ella.
Ella se sintió herida en sus sentimientos.

Esta historia puede ser representada, usando el modelo de DC discutido en el capito anterior, tal y como se muestra en la figura 4-1. Esta representación, es considerablemente más compleja que la anterior.

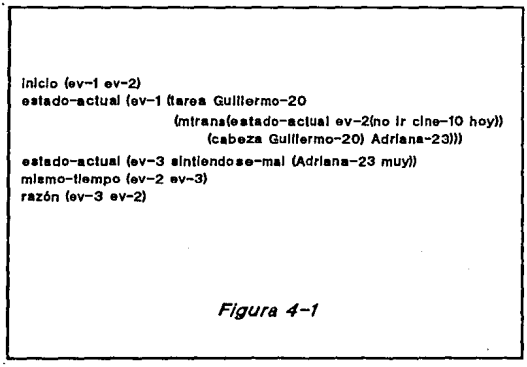


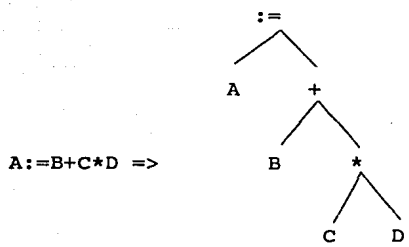

*Figura 4-1*

Conforme se ha ido avanzando en las investigaciones sobre lenguaje natural, las representaciones destino se han vuelto más complejas, ya que se trata de modelar aproximaciones más cercanas al mundo real y con esto lograr un *entendimiento* más profundo del mundo.

Tipo de Mapeo.

Retomando el punto, que para nuestros efectos, el *entendimiento* es un proceso de mapeo de una frase de su forma original a otra más útil, la clase de mapeo más fácil de manejar es uno-uno donde cada elemento de la oración fuente corresponde a uno y solo un elemento de la representación destino.

Muy pocos lenguajes son totalmente uno-uno, sencillamente alguna variabilidad sintáctica es siempre permitida. Pero un ejemplo de mapeo uno-uno, lo podemos hallar en los lenguajes de expresiones aritméticas en la mayoria de los lenguajes de programación. En estos lenguajes un mapeo como el siguiente puede ocurrir:




En general los mapeos uno-uno son los más simples de ejecutar, sin embargo son raros en los lenguajes de interes.

Los mapeos muchos-uno, son más comunes, particularmente cuando se esta mapeando de un lenguaje natural, (ya que son ricos en estructura y vocabulario), a una representación destino más simple y pequeña.

Ejemplificando, podemos tener mapeos tales como:^

Dime todo acerca de la última elección presidencial.

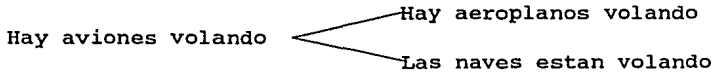
Me gustaria ver todo acerca de la última elección presidencial

Estoy interesado en la última elección presidencial

(BUSCA CLAVE=
ELECCION
PRESIDENTE)

Los mapeos muchos-uno, requieren que el sistema que los interpreta, conozca todas las formas en que la representación destino pueda ser expresada en los lenguajes fuente. Pero a menudo, no requieren de conocimiento adicional.

Por otro lado, los mapeos uno-muchos, requieren de gran cantidad de conocimiento no lingüistico, para poder hacer una correcta elección dentro de todas las representaciones destino disponibles. Un ejemplo de este tipo de mapeos, ( en donde se puede decir que el lenguaje fuente es ambigüo), es el siguiente:



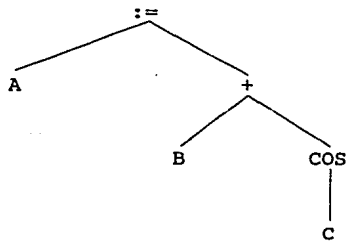

Los lenguajes naturales, tienen ambas de la caracteristicas de los dos últimos ejemplos, esto implica mapeo muchos-muchos, en los cuales existen muchos caminos para decir lo mismo, e igual que una sentencia dada, puede tener diversos significados. El implementar este tipo de mapeos en sistemas, implica que estos manejen una gran cantidad de conocimiento lingüistico y no lingüistico.

Nivel de Interacción entre los Componentes.

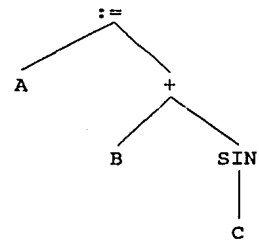
En la mayoria de los lenguajes interesantes, cada oración esta compuesta de diversos componentes (palabras, simbólos o ambos). El proceso de mapeo es simple si cada componente puede ser mapeado sin importar el orden de los componentes. De otro modo, en cuanto mayor sea el número de interacciones, mayor sera la complejidad del mapeo.

Los lenguajes de programación, proveen un buen ejemplo en donde hay muy poca interacción entre los componentes de la instrucción. Por ejemplo:

A:= B + COS( C )




A:= B + SIN( C )




el cambiar una palabra, implica solo un cambio en un nodo.

Por otra parte, en la mayoría de los lenguajes natùrales, el cambiar un palabra no solo puede alterar un nodo de la representación, sino su estructura entera.

Conclusión: Los lenguajes naturales son difíciles.

Como hemos visto en los últimos ejemplos, existen tres factores importantes que influyen en dificultar el proceso de entendimiento para un lenguaje dado. Con estos ejemplos, hemos demostrado, las razones de el porque el *entendimiento* de cualquier lenguaje natural es difícil.

## 4.3 Entendiendo Oraciones Simples.

Como se ha sugerido en diversas ocasiones, el entender oraciones en lenguaje natural es difícil. En esta sección exploraremos técnicas que son útiles en el *entendimiento* de oraciones individuales. Entender grupos de oraciones, como puede ser un texto o un dialogo, es una tarea más difícil ya que requiere de encontrar relaciones entre las oraciones. Por esta razón, la discusión de este problema la diferiremos para más adelante. La construcción de una interpretación coherente de un conjunto de oraciones es difícil, sin embargo, una vez que una parte de la intepretación ha sido construida, esta puede ser usada para asistir en el entendimiento de las oraciones individuales remanentes del texto.

En orden de *entender* (construir representaciones apropiadas) una oración, es necesario realizar dos cosas:

Entender cada una de las palabras de la oración

Ordenar estas palabras en una estructura que represente el significado de la oración completa.

*Entendiendo Palabras.*

Esta primera tarea a realizar, luce a simple vista, muy sencilla. Parecería, que con solo buscar la palabra en un diccionario, ( a menudo llamado *Lexicon* ), que contenga la representación destino que nos indique el significado de la oración, sería suficiente. Desafortunadamente, no es así, ya que una palabra puede tener diversos significados, y no es posible escoger el significado correcto, solo análizando la palabra en si misma. Por ejemplo, la palabra diamante, puede tener el siguiente conjunto de significados:

Figura geométrica

Parte interna de un campo de baseball

Joya extremadamente dura y valiosa

Para elegir el correcto significado de la palabra diamante en la oración:

> Adriana vio resplandecer el diamante de Susana a tráves de la habitación.

es necesario saber que ni las figuras geométricas ni los campos de baseball brillan, mientras que las joyas si.

El proceso de determinar el significado correcto de una palabra individual, es llamado *disambigüedad del sentido de la palabra* o *disambigüedad lexica*. Esta tarea es realizada, asociando a cada palabra en el diccionario, información acerca del contexto en el cual cada una de las interpretaciones de la palabra puede aparecer. Cada una de las palabras en la oración puede servir como parte del contexto con el cual el significado de otras palabras puede ser determinado.

Algunas veces solo información directa acerca del sentido de cada palabra es necesario. Por ejemplo la interpretación de diamante referida a campo de baseball puede ser marcada como *LOCALIDAD*. Entonces el correcto significado de diamante en la oración *Te veo en el diamante* puede ser facilmente determinada si tomamos en cuenta que *en* requiere tiempo o localidad y pueden ser grabadas como entradas lexicas para *en*. Algunas propiedades sencillas de el sentido de las palabras, son denominadas *marcadores semánticos*.

Otros útiles marcadores semánticos son:

Objeto fisico
Objeto animado
Objeto abstracto

Usando estos marcadores, el significado correcto de diamante en la oración *Yo tire mi diamante* puede ser obtenido. Como parte de su entrada lexica, el verbo tirar puede especificar que el objeto puede ser *objeto fisico*. Y además el significado de diamante como gema puede ser marcado como *objeto fisico*. Y tomando en cuenta las dos ligas, se puede obtener el significado apropiado para este contexto.

Desafortunadamente, para resolver completamente el problema de disambigüedad léxica, es necesario introducir más y más marcadores semánticos. Por ejemplo en la oración acerca del diamante de Susana, para su interpretación correcta, un sentido de la palabra diamante debe de ser marcado como *Brilloso* mientras que los otros dos deben ser marcados como *No brillosos*. Y asi conforme el número de marcadores crece, el tamaño del diccionario se convierte en inmanejable además,

cada nueva entrada en el diccionario puede requerir que un nuevo marcador sea agregado a cada una de las entradas existentes.

Otro metodo es el llamado de "preferencias semánticas" [Wilks, 1972]. Este metodo esta basado en la idea que las características de los objetos raramente son demandas rigidas. En realidad, estas pueden ser mejor descritas como preferencias. Por ejemplo nosotros podemos decir que verbos como *odiar* prefieren sujetos que son animados. Por lo tanto no tendriamos dificultad en *entender* la oración:

Pepé odia el frio

Pero ahora considerando la oración:

Mi cesped odia el frio

Al no haber sujeto animado disponible, el uso metáforico de cesped actuando como objeto animado, es aceptado.

*Entendiendo oraciones: Sintácticamente, Semánticamente, Pragmaticamente.*

La segunda parte del proceso de entendimiento, combina palabras para formar estructuras representando el significado de la oración. Esta parte también es difícil, ya que se deben de combinar diversas fuentes de información, entre las cuales se incluye, conocimiento del lenguaje usado, conocimiento del dominio en discusión y conocimiento de las convenciones del lenguaje usadas por las fuentes envueltas en la comunicación. Debido a todas estas cosas que se encuentran envueltas en el proceso de interpretación, a menudo es útil dividirlas en tres componentes:

Análisis sintáctico: Secuencias lineares de palabras son transformadas en estructuras que muestran como las palabras se relacionan unas con otras.

Análisis semántico: A la estructura creada por el análizador sintáctico, se le asigna significado. En otras palabras, un mapeo es hecho entre la estructura sintáctica y objetos en el dominio de las tareas.

Análisis pragmático: A la estructura representando que se dijo, es reinterpretada para determinar que significa realmente.

Las fronteras entre estas tres fases son a menudo muy confusas. Las fases, algunas veces son ejecutadas en secuencia mientras otras, son ejecutadas todas en una pasada. Aun más, aunque las fases sean

ejecutadas en secuencia, siempre interactuan en diversos modos, de forma que una separación completa es imposible.

Más adelante retomaremos este punto para desarrollarlo más a fondo.

### 4.3.1 Comparación en base a claves.

Un modo muy simple de resolver el problema de *entendimiento* del lenguaje, es el combinar todas las operaciones necesarias en un paso sencillo. Esto es hecho, comparando las oraciones de entrada contra un esqueleto de claves. La ventaja mayor de esta técnica, es que permite que oraciones cuya gramática es poco usual, o aun más, oraciones no gramáticales, ser reconocidas. Las palabras clave son reconocidas, mientras que el resto es ignorado. Esto constituye una forma de comparación por aproximación. Esta idea fue explotada en dos de los primeros sistemas que respondian a sentencias de entrada en ingles. El primero fue ELIZA [Weizenbaum, 1966], el cual simulaba el comportamiento de un terapista Rogeriano. El otro ejemplo de esta técnica modelaba un paciente psiquiatrico, PARRY [Colby, 1975] es un modelo de comportamiento paranoico. En ambos sistemas, no es importante que lo que se dice sea interpretado correctamento. De hecho para PARRY, es más importante que no lo sea. De modo que esta técnica de esqueletos de claves producen un espejismo de forma que el usuario del sistema piensa que ellos *entienden*.

Los manuscritos en los cuales esta basado ELIZA, contiene esqueletos que son comparados contra las oraciones de entrada. Asociados a cada esqueleto de entrada, se encuentran uno o más esqueletos de salida, los cuales permiten generar la respuesta correcta. ELIZA en realidad no entiende las oraciones de entrada, en el sentido de que las mapeé a estructuras que representen su significado. En lugar de esto, las sentencias son mapeadas en respuestas inmediatas, y en general, despúes son desechadas. Para simplificar la generación de respuesta, algunas palabras son inmediatamente trasladadas. Por ejemplo, algunas de estas palabras que se trasladan en automático son *I* en *You* y *Me* en *You*. Por ejemplo, si el paciente dice *I cried*, Eliza respondera *Why did you crie ?* Estas tranformaciones inmediatas, son aplicadas antes de tratar de aplicar cualquier esqueleto. Un manuscrito de ELIZA esta organizado por claves primarias. Por ejemplo, supongamos la palabra *remember*. Para esta palabra, nosotros podemos tener los siguientes esqueletos de comparación:

*( 0 YOU REMEMBER 0 )*
*( 0 DO I REMEMBER 0 )*

con el siguiente esqueleto de salida:

( *WHY DO YOU REMEMBER 4 JUST NOW* )

En estos esqueletos, 0 puede ser comparado o substituido con cualquier número de palabras en la oración de entrada ( incluyendo ninguna ). En el esqueleto de salida, 4 debe de ser substituido por la oración de entrada a partir de la cuarta posición. Por ejemplo, si alguien le dice a ELIZA *I remember the first time I went to the beach*, de acuerdo a los esqueletos anteriores, ELIZA respondera *WHY DO YOU REMEMBER THE FIRST TIME YOU WENT TO THE BEACH JUST NOW*.

En un sistema basado en claves, hay dos situaciones que deben de ser tomadas en cuenta. La primera de ellas, es cuando una oración de entrada tiene correspondencia con más de un esqueleto. ELIZA resuelve este problema, asignandole a cada esqueleto una prioridad, de modo que se toma en cuenta el esqueleto que tiene la mayor prioridad.

El segundo problema, es cuando ningun esqueleto aplica a la sentencia de entrada. ELIZA resuelve esto, asignando un esqueleto que siempre aplica, el cual contiene respuestas de proposito general como *CAN YOU ELABORATE ON THAT* o *THAT'S QUITE INTERESTING*.

La mayor desventaja de esta técnica, es que ignora una gran cantidad del significado de las sentencias no gramáticales.

## 4.3.2 Análisis sintáctico.

Para estar seguro de que ninguno de los detalles del significado de una oración son ignorados, es necesario el determinar los detalles de su estructura. Esto es hecho por un proceso llamado análisis (*parse* en ingles). Para análizar una oración, es necesario usar una gramática que describa la estructura de las cadenas de un lenguaje en particular. Dada una gramática, el analizador puede asignar una estructura a cada sentencia gramátical que análiza. Esta estructura es llamada árbol de análisis. La figura 4-2 muestra una gramática simplificada para un subconjunto del español, asi como el árbol que se produce para la oración *Juan le pego al balón*, usando esta gramática. Notesé

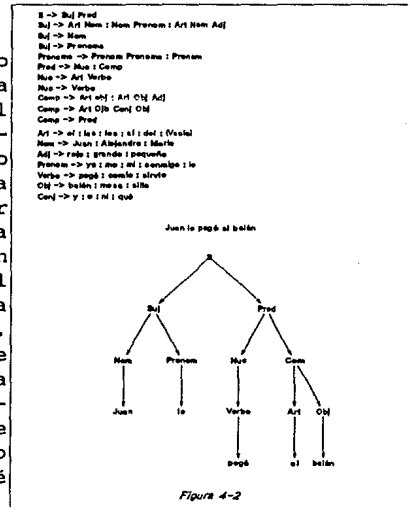



*Figura 4-2*

que la estructura del árbol corresponde al camino en que las reglas gramáticales van siendo aplicadas para transformar el simbolo inicial *S* en la oración final.

El proceso de análisis realiza dos tareas:

Determina cuales oraciones son aceptadas como sintácticamente bien estructuradas y cuales no.

Para oraciones gramáticales bien formadas, les asigna un árbol o estructura.

A continuación daremos la definición formal de gramáticas.

### 4.3.2.1 Definición formal de lenguaje y gramática por las Ciencias de la Computación.

Comenzaremos por definir lo que es un lenguaje.

Se puede decir, en términos prácticos que un lenguaje es un medio que permite que dos entidades distintas se puedan comunicar.

Desde el punto de vista de las ciencias de la computación un lenguaje está definido por un conjunto de cadenas (secuencia de símbolos), dos conjuntos distintos de reglas las cuales determinan, la sintaxis (esto es, si la cadena se encuentra bien construida), y la semánticas (si el significado de la cadena tiene sentido)(1).

Todo lenguaje tiene un alfabeto, el cual consiste de todos los símbolos que una cadena puede contener. También todo lenguaje tiene una gramática, la cual consiste (en parte), de el conjunto de reglas para generar enunciados de el lenguaje.

Formalmente la gramática "G" se puede definir como el conjunto de 4 elementos o subconjuntos de las siguientes características:

$$G= (V_T, V_N, P, S)$$

donde V subíndice T es un subconjunto cuyos símbolos son conocidos como símbolos terminales. V subíndice N es un subconjunto cuyos símbolos son conocidos como no terminales y V subíndice N y V subíndice T no tienen elementos en común, es decir que la intersección de estos dos conjuntos nos dá el conjunto vacío. P es el conjunto de las reglas de producción, cada elemento de los cuales consiste a su vez en un par (alfa, beta). Alfa es normalmente conocido como la parte izquierda de la producción y beta como la parte derecha de la misma. S es un elemento de los símbolos no terminales, y es normalmente

el punto de partida en la producción de cualquier enunciado del lenguaje.

Para tener una panorámica más amplia de estos conceptos, ejemplificaremos de la siguiente manera:

Una gramática que genera el siguiente lenguaje

$$[0^n 1^n / n => 0]$$

es $G_0$ , donde

$$G_0 = ([0,1],[S], P, S)$$

y P = [ S -> 0S1, S -> (vacio) ]

esto es hemos definido una gramática donde los elementos terminales pueden ser los símbolos "0" y "1", los elementos no terminales están compuestos por el símbolo S, se encuentra sujeta a las reglas de producción P y el axioma o punto inicial es S. Esta gramática nos produciría enunciado del siguiente estilo:

S -> 0S1 -> 00S11 -> 000S111 -> 000111

A las gramaticas que cumplen las reglas mencionadas con anterioridad, se les denomina gramaticas de estructura de frase. Chomsky( ) ha dividido a estas gramaticas en cuatro tipos de acuerdo con la forma de sus reglas de producción, estas se explican a continuación.

Gramática de tipo 0 no restringida.

En este tipo de gramática, no hay restricciones en las producciones, puede tener cadenas formadas de distintas maneras, las cadenas pueden tener substituciones a la derecha o a la izquierda. Este tipo de gramaticas, es demasiado general para ser útil y el problema de las gramaticas de tipo cero, consiste en que las cadenas generadas suelen ser impredecibles. Los lenguajes generados por gramaticas de tipo cero, son llamados lenguajes de tipo cero.

Gramática de tipo 1 sensitivas al contexto.

Las producciones de las gramaticas de tipo uno se encuentra restringidas de la siguiente manera:

$phi_1$ A $phi_2$ -> $phi_1$ beta $phi_2$

donde A pertenece a los elementos no terminales, phi subindice1, phi subindice2 y beta son elementos de V, y beta es una cadena no vacia. Esto se puede leer como ''A puede ser remplazado por beta en el contexto $phi_1$, $phi_2$.

Esto implica que en valor absoluto A es menor o igual que beta.

Los lenguajes generados por gramaticas sensitivas al contexto, son llamados lenguajes de tipo uno o lenguajes sensitivos al contexto. Un ejemplo de este tipo de lenguajes es:

$(0^{n_1}n_0^n$ donde n=1,2,3,.....)

Gramaticas tipo 2 libres del contexto.

Las producciones de este tipo de gramaticas son de la forma:

A->beta

donde A es elemento de los no terminales y beta es elemento de V. Hay que notar que las producciones de esta forma permiten al no terminal A ser remplazado por la cadena beta independientemente del contexto en el cual A aparezca. Los lenguajes generados por este tipo de gramaticas son llamado de tipo dos o lenguajes de contexto libre.

Una alternativa para describir cualquier derivación en gramaticas de contexto libre, consiste en usar un árbol de derivación. Un árbol de derivación para una gramática de contexto libre puede ser construido de acuerdo con el siguiente procedimiento:

1.- Cada nodo del árbol tiene una etiqueta, la cual es un simbolo de V.

2.- La raíz del árbol, tiene que ser S.

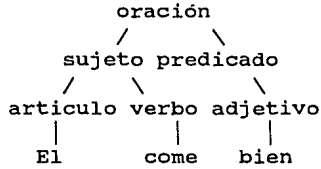
3.- Si un nodo tiene al menos un descendiente, este tiene que estar etiquetado como A.

4.- Si los nodos n1, n2, . . ., nk son descendientes directos de el nodo n, con etiqueta A, entonces

A -> A1,A2,...,Ak

debe de ser una regla de produccion en P.

Ejemplificando, un árbol de derivación para la oración "El come bien", seria:




Gramaticas de tipo 3 regulares o de estado finito.

Las producciones de las gramaticas de tipo tres, son de la forma:

A -> aB o A ->b

donde A y B son elementos de los no terminales, a y b son elementos de los terminales. Hay que notar que A, B, a, b son simbolos simples. Por ejemplo

$(0^{m}1^{n}$ donde m,n=1,2,...,)

es un lenguaje de estado finito. Los lenguajes generados por este tipo de gramática son conocidos como lenguajes de tipo tres o lenguajes regulares o lenguajes de estado finito.

De la definición de los cuatro tipos de gramaticas, es claro que cada gramática de estado finito es de contexto libre; cada gramática de contexto libre es sensitiva al contexto y cada gramática sensitiva al contexto es de tipo cero.

Todos los lenguajes, tanto los naturales como los de programación, se encuentran definidos de la misma manera, la principal diferencia radica en como fueron concebidos, generalmente se utilizan gramáticas de contexto libre para representarlos.

Mientras que los lenguajes naturales fueron evolucionando junto con las necesidades de comunicación de las sociedades humanas, los lenguajes de programación fueron concebidos con el propósito de realizar tareas especificas, como son la de expresar algoritmos.

De aquí que mientras los lenguajes de programación son sencillos

en su análisis, los lenguajes naturales no lo son, ya que las reglas de producción de estos lenguajes son limitadas y no contemplan la mayoría de los casos, además existen una serie de ambigüedades, y esto sin contar que entre los elementos que hablan un mismo lenguaje, los modismos del lenguaje cambian de región a región e incluso varían entre los diferentes estratos sociales de una misma región.

| Tipo | Clase de lenguaje | Restricciones |
|---|---|---|
| 0 | Recursivo enumerable | Ninguna |
| 1 | Sensitivo al contexto | Sin reglas de reducción |
| 2 | Contexto libre | El lado izquierdo debe ser no terminal |
| 3 | Regular | El lado izquierdo debe ser no terminal mientras que el derecho deber ser terminal o un terminal seguido por un no terminal |

Un resúmen de la jerarquía de Chomsky es mostrada en la figura 4-3.

Los lenguajes de la parte de abajo de esta jerarquía, pueden ser reconocidos con procedimientos de análisis muy eficientes. Los lenguajes regulares pueden ser reconocidos usando *máquinas de estado finito*. Una *máquina de estado finito*, es una estructura en donde los calculos son modelados como una transición de un estado a otro dentro de un número finito de estados. Uno o más estados pueden ser marcados como estados finales. Si el fin de una entrada es encontrado y la máquina se encuentra en un estado final, entonces la entrada es aceptada como sentencia gramátical correcta. Los estados de la máquina, estan concectados por arcos, estos se encuentran etiquetados con el caracter que va a permitir la transición de un estado a otro. La figura 4-4, muestra un ejemplo de una máquina de estado finito que acepta cualquier cadena de 0's y 1's siempre y cuando el número de 1's sea inpar.

*Figura 4-3*

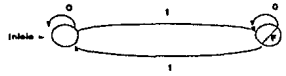



*Figura 4-4*

Lamentablemente ninguno de los lenguajes naturales, puede ser modelado correctamente por una gramática regular.

En lo referente a lenguajes de contexto libre, particularmente en lenguajes deterministicos de contexto libre, han sido desarrollados algoritmos de análisis muy eficientes para ser usados en compiladores de lenguajes de programación. Lamentablemente mientras que las gramáticas de contexto libre pueden definir muy bien a los lenguajes de programación, no son buenas para modelar a los lenguajes naturales.

Redes de Transición Aumentadas.

Por lo visto hasta ahora, se puede deducir, que para poder reconocer setencias de un lenguajes natural, necesitamos el poder de una máquina de Turing, y para que esta sea más eficiente, necesitamos la habilidad de incorporarle una variedad de clases de conocimiento al

sistema de análisis. *Redes de Transición Aumentadas* (ATN's en ingles) [Woods, 1970], nos proveén de un camino para hacer esto. Un ATN es similar a una máquina de estado finito en la cual las etiquetas asociadas a los arcos que definen la transición entre los estados, han sido aumentadas. Los arcos pueden estar etiquetados con un combinación arbitraria de los siguientes elementos:

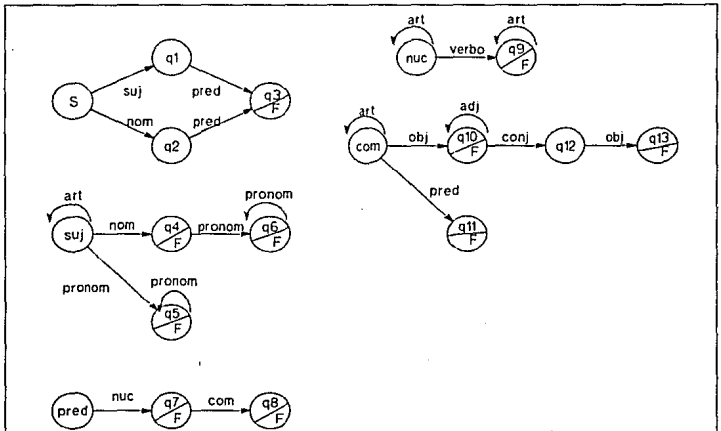



*Figura 4-5*

Palabras especificas.

Llamadas a otras redes que puedan reconocer componentes significativos de las sentencias. Una vez que se ha hecho la llamada a la red del segundo nivel, puede ocurrir una llamada recursiva a la red del primer nivel o a otra de un tercer nivel.

Procedimientos que ejecuten pruebas arbitrarias tanto en la entrada actual como en los componentes que ya han sido identificados.

Procedimientos que contruyan estructuras que formen parte del análisis final.

Debido a toda esta variedad de condiciones que pueden estar asociadas a los arcos, una red ATN tiene el poder formal de una máquina de Turing.

Un ejemplo es mostrado en la figura 4-5, este ejemplo, ilustra diversos puntos interesantes acerca del uso de las redes ATN como son:

1) Una subred, solo se define una sola vez, a pesar de que pueda ser llamada en diversos puntos.

2) Una red puede ser llamada recursivamente.

3) Cualquier número de registros internos puede ser usados para contener el resultado del análisis.

4) Un estado, puede ser a la vez un estado final o un estado de transición.

5) El contenido de un registro puede ser modificado en cualquier momento.

6) El contenido de los registro puede ser intercambiado. Esto permite que el significado de las siguientes dos oraciones sea el mismo:

Alex golpeo a Maria
Maria fue golpeada por Alex

7) Pruebas arbitrarias, posiblemente semánticas, pueden ser agregadas a los arcos.

8) Procedimientos de regresión deben de ser implementados eficientemente, de forma que si un error ocurre, la cadena de entrada y los registros puedan ser regresados a su estado original.

Por el otro lado, las redes ATN cuentan con las siguientes desventajas:

Pueden ser muy ineficientes si una gran cantidad de regresión es requerida.

Aunque puede ser incorporado conocimiento semántico para rechazar posibles trayectorias, este rara vez puede ser usado para elegir la mejor trayectoria, dentro de una amplia gama.

A menos que todas las palabras en la oración de entrada sean conocidas, y además, la estructura de la oración concuerde exactamente el proceso de análisis fallara, ya que no hay manera de hacer un análisis parcial.

Pero a pesar de estas desventajas, las redes ATN permanecen como uno de los mecanismos más eficientes para realizar el proceso de análisis. Para un estudio a profundidad de este tema se puede consultar

a [Winograd, 1983].

### 4.3.3 Análisis Semántico.

Producir el análisis sintáctico es solo el primer paso del entendimiento. En algún punto, una interpretación semántica de la oración debe de ser producida.

Existen una gran variedad de métodos para llevar a cabo esto, los cuales pueden ser divididos en cuatro categorias, estas son:

*Gramáticas Semánticas*, el cual combina conocimiento sintáctico y semántico dentro de un conjunto de reglas en forma de gramática.

*Gramáticas de Caso*, en el cual la estructura que es construida por el análizador contiene información semántica.

*Filtrado Semántico de Análisis Generados Sintácticamente*. Existen dos caminos para llevar a cabo esto:

Filtrando análisis parciales de acuerdo a como son encontrados. Por ejemplo, incluyendo pruebas semánticas en redes ATN.

Esperar hasta que el análizador termine para despúes evaluar su aceptabilidad semántica.

No enfatizar el análisis sintáctico y manejar el proceso de entendimiento por conocimiento semántico en lugar de sintáctico.

#### 4.3.3.1 Gramáticas Semánticas.

Una gramática semántica es una gramática de contexto libre en la cual la elección de no terminales y reglas de producción es gobernanda por funciones sitácticas y semánticas.

Las principales ventajas de este método son las siguientes:

Cuando el análisis es completado, el resultado puede ser usado inmediatamente sin necesidad de el proceso adicional que se requeriría si la interpretación semántica no hubiera sido ejecutada durante el análisis.

Muchas de las ambigüedades que surjen durante un análisis

puramente sintáctico son excluidas ya que no cuentan con sentido semántico.

Características puramente sintácticas que no afectan semánticamente pueden ser ignoradas.

Las desventajas son las siguientes:

El número de reglas requeridas, es muy grande.

Al ser muy grande el número de reglas, el proceso de análisis es muy costoso.

### 4.3.3.2 Gramáticas de Caso.

Las gramáticas de caso, provéen una alternativa al problema de como las interpretaciones sintácticas y semánticas pueden ser combinadas. Las reglas gramáticales son diseñadas para describir regularidades sintácticas más que semánticas. Pero la estructura que las reglas producen corresponde a relaciones semánticas más que sintácticas.

Por ejemplo, vamos a suponer las siguientes dos oraciones:

- Adriana golpe a Alberto
- Alberto fué golpeado por Adriana

Usando gramática de casos, la interpretación para ambas oraciones sería:

```
( Golpear ( Agente Adriana )
          ( Receptor Alberto ))
```

Ahora considerense las siguientes oraciones:

- Mamá cocino por tres horas
- El pastel se cocino por tres horas

La estructura sintáctica de estas oraciones es identica. En un caso *Mamá* es el sujeto de *cocinar*, mientras que en el otro *El pastel* es el sujeto. Pero la relación en *Mamá* y *cocinar* es muy diferente que entre *pastel* y *cocinar*. Un análisis por gramática de casos, refleja esta diferencia.La primera oración sería interpretada asi:

```
( Cocinar ( Agente Mamá )
          ( Periodo-de-tiempo tres horas ))
```

Mientras que la segunda oración sería:

```
( Cocinar ( Objeto El pastel )
          ( Periodo-de-tiempo tres horas ))
```

Notesé que los casos usados por la gramática describen relaciones entre verbos y sus argumentos.

No existe algún acuerdo claro sobre el conjunto ideal de los casos o roles de los componentes, sin embargo, algunos obvios son:

(A) Agente: (tipicamente animado) instigador de la acción.

(I) Instrumento: (tipicamente inanimado) causa del evento u objeto causante del evento.

(R) Receptor: (tipicamente animado) entidad afectada por la acción.

(H) Hecho: Resultado del evento.

(L) Localidad: Lugar del evento.

(F) Fuente: Lugar desde donde algo se mueve.

(M) Meta: Lugar hacía donde algo se mueve.

(B) Beneficiario: (tipicamente animado) beneficiado de que el evento ocurra.

(T) Tiempo: Tiempo en el cual el evento ocurre.

(O) Objeto: Entidad sobre la que se efectuan los cambios.

El proceso de análisis en representación de casos, esta estrechamente relacionado con las entradas léxicas asociadas a cada verbo.

El análisis usando gramática de casos, puede ser concebido como un *manejador de expectativas*. Una vez que el verbo de la oración ha sido localizado, puede ser usado para predecir y determinar la relación entre los demás componentes de la oración.

### 4.3.3.3 Dependencia Conceptual.

Análizar una oración en base a una representación de DC es similar a análizarla usando gramática de casos. En ambos esquemas, el proceso de análisis se encuentra fuertemente relacionado al conjunto de expectativas que se desprenden del verbo principal de la oración. Pero debido a que la representación de un verbo en DC es considerada como primitivo, DC usualmente provée de un grado mayor de flexibilidad en su poder de predicción. El primer paso para mapear oraciones en representaciones de DC, consiste en extraer de la oración el sujeto y verbo principales, a tráves de un proceso sintáctico. Esto va a dar la pauta para determinar la categoria sintáctica del verbo ( transitivo, intransitivo, etc.). En este punto es donde entra en acción el procesador conceptual, el cual haciendo uso del diccionario de acciones por verbo, extraera las entradas que determinan en que medio ambiente el verbo puede aparecer.

Ejemplificando, el verbo *quiero*, puede tener asociados los siguientes medio ambientes:

- *Querer* que algo suceda
- *Querer* un objeto
- *Querer* a una persona

Una vez que la elección se hace del diccionario, el procesador conceptual análizara el resto de la oración, buscando compaginarla con alguno de los esquemas seleccionados del diccionario.

Este método produce muy buenos resultados, sin embargo se debe de tomar en cuenta que, para que el sistema realice un excelente análisis, se requiere de una gran cantidad de información semántica, lo cual puede hacer el proceso muy costoso.

## 4.4 Analisis de Lenguajes por las Ciencias Comunicación.

En las secciones anteriores nos hemos dedicado basicamente a estudiar el problema desde un punto de vista únicamente computacional. A continuación pasaremos a análizar las reglas que rigen el idioma español, con el objeto de tener un alcance más amplio de la problemática.

Para poder comunicarse, el hombre usa diversos medios simbólicos de comunicación; signos no lingüísticos, guiños, señales, gestos, dibujos, etcétera. A esta capacidad humana de comunicarse simbólicamente se le llama lenguaje.

Uno de estos medios de comunicación simbólica, el más útil y

perfecto, es la lengua.

A la lengua la podemos definir de la siguiente manera:

| La lengua es un sistema de signos lingüísticos. |
|---|

La lengua funciona como un código; es decir, como un arsenal de signos lingüísticos. Los signos de este código, son relacionados y combinados ordenadamente, conforme a reglas. Esto es, por ejemplo, si en lugar de escribir "la casa donde vives", yo escribo "vives la donde casa", ya no estoy realizando la lengua, ya que no estoy logrando comunicarme porque los signos no han sido articulados; es decir, no están relacionados en virtud de que no han sido combinados ordenadamente conforme a ciertas reglas, por lo tanto no hay mensaje.

Ahora definiremos lo que es la lingüística. La lingüística es la ciencia que estudia, a través de sus ramas, que son disciplinas científicas, todos los aspectos de los signos lingüísticos.

Entre las ramas de la lingüística, relevantes al desarrollo de este trabajo tenemos:

a) Fonología. Esta disciplina estudia el valor funcional de los fonemas. La fonología, al igual que la fonética y las demás disciplinas, puede ser sincrónica y en este caso describe fenómenos simultáneos considerados como si fueran estáticos, en una etapa dada del desarrollo de la lengua. O bien, puede ser diacrónica, cuando estudia sucesivas fases evolutivas de los mismos fenómenos.

b) Semántica. Estudia el significado de los signos lingüísticos.

c) Gramática. Estudia la función y la forma de los signos lingüísticos.

Empezaremos por analizar la primera de estas disciplinas.

### 4.4.1 Fonología.

Cada fonema representa un sonido, pero no siempre está representado por una sola letra o grafía. Por eso el sistema fonológico, que representa los sonidos, y el alfabeto ortográfico, las grafías o letras, con que se escribe el español, no guardan entre sí una exacta correspondencia.

El español posee fonemas que se realizan en vocales: a, e, i, o, u, y fonemas que se realizan en consonantes:

d, f, l, m, n, ñ, p, t.

Los desajustes entre ambos consisten en que:

El fonema "b", corresponde a dos letras ("b" y "v"), lo mismo que el fonema "y" (letras "ll" y "y" inicial de sílaba), y el fonema "rr" (letras "r" y "rr" intervocálica).

Corresponden a tres letras: el fonema ''k'' (que se escribe con la k, la qu y la c que precede a las vocales a, o, u: ca, co, cu) y el fonema "s" (que se escribe con las letras s, z y c seguida de e ó de i: ce, ci).

El fonema "g" corresponde a la letra g cuando va antes de a, o, u, (ga, go, gu), o bien, corresponde a la letra doble gu, antes de e ó de i (gue, gui).

El fonema "j" corresponde a la letra j y también a la letra g, pero solamente cuando va seguida de las vocales e, i (ge, gi).

Hay letras dobles: ch (fonema "ch"), ll (fonema "y"), qu (fonema "k"), rr (fonema "rr") y gu seguida de e ó de i (fonema "g").

La letra x corresponde a dos sonidos representados por sus respectivos fonemas: "k" y "s".

La letra r a veces representa al fonema "r" (arete) y a veces al fonema "rr" (rata).

La letra h no correponde a ningún sonido ni representa ningún fonema, sólo tiene valor ortográfico.

La letra w no se usa en español más que en palabras procedente del inglés (y en ese caso suena como u: Washington), o procedentes del alemán (y entonces suena como las letras representadas por el fonema "b": Wagner).

El español que hablamos en México tiene 22 fonemas (que representan los sonidos que realmente pronunciamos). En cambio, el alfabeto ortográfico que usamos para escribir los sonidos contiene 31 letras, contado la h que no representa sonido (ni fonema).

La mayoría de los errores ortográficos, se suscitan debido a estos desajustes entre los fonemas o la forma de pronunciarlos y entre

los signos gráficos o la forma de escribirlos.

### 4.4.2 Semántica.

Como se mencionó con anterioridad, la semántica estudia el significado de los signos lingüísticos.

Los signos se asocian conforme a dos modos de actividad mental, estos son:

Relación sintagmática. Formando sintagmas que son cadenas lineales, horizontales, de palabras, es decir, formando frases u oraciones articuladas, o sea, combinadas y relacionadas según ciertas normas de distribución, orden y dependencia.

Dentro de la cadena cada signo adquiere su valor gramatical debido a la presencia de los otros signos y a su relación con ellos, las relaciones sintagmáticas se dan detro del plano sucesivo del discurso.

Un ejemplo de relación sintagmática, serían las frases y las oraciones.

Relación paradigmática. Formando paradigmas que son cadenas verticales en que los signos se asocian mentalmente, ya sea por alguna relación de semejanza entre sus significantes (morfemas), o por alguna relación entre sus significados, relaciones que se dan dentro del plano del sistema.

Ejemplos de relaciones paradigmáticas, serían las conjugaciones como parto, partes, parte, etcétera, o las variantes que expresan los accidentes gramaticales nominales como niño, niña, ninos, niñas, o los derivados dentro de una familia de palabras como niño, niñera, niñito, niñería, etcétera, o las palabras que tienen el mismo morfema derivativo como grandeza, belleza, pereza, vileza o el de los sinónimos como malo, perverso, o los antónimos como fuerte, débil.

La forma en que se jerarquizan los sintagmas, es:

Discurso: Es una sucesión coherente de sintagmas.

Sintagma: Cadena lineal, horizontal, de palabras articuladas.

Palabra: Mínima unidad sintáctico-semántica separada.

Morfema: Mínima forma significativa, unidad morfo-semántica.

Fonema: Mínima unidad fonológica.

Existen basicamente dos clases de sintagmas, estos se explican a continuación.

Sintagmas sintácticamente dependientes. Son frases que carecen de sujeto y predicado u oraciones subordinadas que constan de sujeto y predicado. Su dependencia sintáctica consiste en que cumplen una funcion dentro de otra forma, es decir, dentro de otro sintagma.

Sintagmas sintácticamente independientes. Son oraciones simples, compuestas o complejas. Estas oraciones no se encuentran subordinadas a ninguna otra forma. Las oraciones compuestas pueden constar de dos o más oraciones simples. Las oraciones complejas pueden constar de una o más oraciones subordinadas.

### 4.4.3 Gramática.

Al articularse las palabras en el sintagma, se combinan conforme a ciertas normas de distribución:

a) Cuántos artículos pueden acompañar a un sustantivo.

b) Orden, el necesario para que la cadena resulte coherente.

c) Dependencia, unas palabras quedan subordinadas a otras.

Según la relación que el sintagma establece, cada palabra adquiere un valor gramático, una jerarquía (de mayor o menor importancia), que depende, en cada sintagma, de la función que allí cumple. Las funciones sólo se dan dentro de los sintagmas. El sintagma más importante es la oración, porque en él se dan todas las funciones.

Por su función, las palabras o sintagmas pueden ser de categoría:

1. Primaria, las que son principales y se llaman núcleos;

2. Secundaria, complementaria o subordinada, las que dependen de los núcleos y se llaman modificadores o complementos;

3. Terciaria, las que vinculan entre sí palabras o sintagmas y se llaman nexos o relacionantes.

Los núcleos son independientes, pueden llevar o no, complementos o modificadores.

Los modificadores y complementos sólo tienen valor gramatical referidos a su núcleo.

Los nexos relacionan de dos maneras:

a) Coordinan cuando vinculan entre sí dos palabras o sintagmas del mismo nivel sintáctico (porque ninguno depende del otro).

b) Subordinan cuando vinculan un modificador o complemento con su núcleo.

- Son núcleos:

a) El verbo, núcleo del predicado verbal

b) El sustantivo, núcleo del sujeto, del complemento directo, etcétera.

- Son modificadores o complementos:

a) El adjetivo y el artículo, modifican al sustantivo.

b) El adverbio, modifica o complementa al verbo.

- Son nexos coordinantes o subordinantes:

a) Las preposiciones, generalmente subordinan.

b) Las conjunciones, coordinan o subordinan.

c) Los pronombres relativos, subordinan.

d) Algunos adverbios, coordinan o subordinan.

e) Algunos pares de palabras correlativas, coordinan (aquí..allá..unos..otros).

De todo lo anterior se deduce que sólo donde hay sintagma las categorías cumplen funciones.

Concluyendose que la oración es el sintagma más importante porque en ella se dan todas las funciones anteriormente expuesta. Por eso para estudiar las funciones es necesario conocer la estructura de la oración:

1. Un sintagma es oración si consta de sujeto y predicado:

(Esa gran planicie desértica) (siempre estuvo sola)
Sujeto Predicado

2. No tiene importancia el orden del sujeto y el predicado:

(Siempre estuvo sola) (esa gran planicie desertica)
Predicado Sujeto

3. No tiene importancia que el sujeto sea tácito (el verbo lo indica):

Fuimos al campo (nosotros)
Predicado Sujeto

4. No tiene importancia que el sujeto sea indefinido:

a) Cuando lo ignoramos: Cuentan que viene al anochecer un fantasma.

b) Cuando el verbo es unipersonal: Llueve mucho. ¿Quién llueve?. Llueve la lluvia, pero nunca lo decimos, y usamos la 3a. persona.

5. No tiene importancia que el sujeto esté expresado por el contexto: A veces, cuando el núcleo del predicado es un verboide (infinitivo, gerundio o participio):

(Yo lo hago) (para convercerte)

El sujeto del verboide convencer es yo (para que yo te convenza, es un sintagma equivalente). En estos casos el contexto indica cuál es el sujeto.

El núcleo del sujeto generalmente va acompañado por modificadores que pueden ser:

A) Modificador mediato (mediante nexo):

a) Una frase: Casa del risco.

b) Una oracion subordinada: Casa que me gusta.

El nexo puede ser:

1. Una preposición: aro de plata, mar sin orillas, guiso con sal.

2. Una conjunción: un río, y grande, nos cerró el paso.

3. Un pronombre relativo: ese pueblo (cuyo nombre ignoro) desapareció.

4. Un adverbio: la casa (donde viví) será renovada.

B) Modificador inmediato (sin nexo):

a) Una palabra: la casa, casa blanca (artículo o adjetivo)

b) Una frase: casa muy blanca (de naturaleza adjetiva)

Puesto que el núcleo del sujeto es un sustantivo, todos sus modificadores tienen siempre un valor adjetivo, sean palabras, frases u oraciones subordinadas.

El verboide y predicado son otras partes importantes que se deben analizar:

1. Los verboides (infinitivo, gerundio y participio) son las únicas formas no personales del verbo, pues no podemos conjugarlos (yo partir, tú partir, etcétera) por no poder atribuirles una persona gramatical.

2. Los tres verboides cumplen más de una función. Los tres pueden ser núcleo del predicado verbal, aunque sin expresar el sujeto en su morfema. El sujeto, explícito o tácito, se halla en el contexto.

3. El infinitivo cumple dos funciones:

a) De sustantivo

b) De verbo: solo ó con auxiliar.

Ejemplos:

a) Sustantivo: El piar (núcleo) de las golondrinas (sujeto) me adormecía (predicado)

b) Verbo solo: (Olvidar yo los favores) (que te debo) (no será posible)

Son tres oraciones con tres verbos; se advierte mejor si buscamos un equivalente de la primera oración: (No será posible) (que yo olvide)

(los favores que te debo)

El sujeto: yo, que aquí está explícito, hace evidente la presencia de los dos elementos: el sujeto y el predicado.

c) Verbo en perífrasis (con auxiliar):

Yo voy a leer.
Mañana piensan salir.
Ellos quisieron ser.

En estas tres oraciones el verbo es una frase verbal o perífrasis verbal uno de cuyos elementos (el último) es el infinitivo. El infinitivo es una parte del núcleo del predicado.

Con todos estos elementos, nosotros contamos ya con los antecedentes necesarios para hacer un enfoque adecuado del analisis del problemas ortográfico, desde el punto de vistas de las ciencias de la comunicación.

# CAPITULO V

# TECNICAS DE RECONOCIMIENTO DE FALLAS Y SISTEMAS EXPERTOS.

## 5.1 Introducción.

En los capítulos anteriores, se han mostrado y explicado las distintas herramientas con que se cuenta en el campo de la IA.

En el presente capítulo, tomaremos el problema de reconocimiento de fallas ortográficas desde un punto de vista mucho más practico. Para esto daremos algunas de las principales características de los lenguajes naturales, haremos un análisis de algunas técnicas practicas para reconocimiento de falls, principalmente de origen sintáctico. Daremos la esplicasión de lo que es un sistema experto y para concluir, mostraremos algunas de las principales reglas practicas para el razonamiento de fallas ortográficas.

## 5.2 Características y Análisis de Lenguajes Naturales.

Como se comentaba en la sección anterior, los lenguajes nacen de la necesidad de comunicación entre los elementos de una sociedad.

Debido a esto, los lenguajes naturales fueron evolucionando de una manera desordenada, o sin ninguna lógica aparente, ya que no existe ninguna razón, cuando menos obvia, que nos indique el porqué un pizarrón debiera llamarse así.

De tal manera que cuando se trata el problema del análisis de un lenguaje natural, nos encontramos con la sorpresa de que tenemos que tomar en cuenta una serie de importantes limitantes. Por principio de cuentas, y tomando por ejemplo el lenguaje español, las palabras casa y caza estarán correctas dependiendo de el significado completo de la oración, ya que si nos encontramos con la oración ''la caza de Pedro es chica'', la palabra ''caza'' estaría semánticamente incorrecta, pero si nos encontramos con ''Pedro va de caza'', la palabra estaría correcta. Por otro lado y utilizando las dos oraciones anteriores, vemos que la estructura de ambas

oraciones cambia, pero ambas estructuras son reconocidas por el lenguaje español.

Partiendo de lo anterior, y con desilusión vemos que nuestra primer problema a resolver, al analizar lenguajes naturales, es que tenemos que tomar necesariamente un análisis semántico, ya que un análisis puramente sintáctico no es suficiente para realizar esta tarea.

Tomando esto en cuenta, seguiremos análizando otras limitaciones como las que se muestran a continuación.

Tomaremos por ejemplo la palabra ''mas'', si esta palabra es usada como adjetivo la palabra no va acentuada, pero si es usada como pronombre la palabra debe de ir acentuada, y de nueva cuenta el análisis sintáctico dependerá de el sentido semántico de la oración, como en el primer ejemplo.

De esto nosotros podemos deducir que el problema no es el lenguaje en sí, más bien es una parte del mismo, el punto principal del problema es el conocimiento, ya que el hacer que un sistema pueda analizar un texto, radica en el algo más que proveerlo de un diccionario de palabras o verbos. Simplemente nadie esta capacitado para opinar sobre cualquier asunto a menos que lo conozca. De tal forma que la conclusión lógica al problema sería proveer al sistema de una cantidad de conocimiento acerca del mundo real, para que sobre esta base, todas las complicaciones descritas con anterioridad, pudieran ser solucionadas de una manera satisfactoria(2).

Pero nos encontramos con la limitante de los recursos con los que contamos. Ya que nuestro objetivo es diseñar un sistema el cual pueda detectar errores ortográficos, y que además pueda ser ejecutado en cualquier microcomputador de los popularmente conocidos como ''PC'S'', y el cual sea competitivo en cuanto a ejecución, tiempo de respuesta e interface con el usuario y aprovechamiento de recursos.

Esto nos plantea limitantes en cuanto a la cantidad de memoria con la que contamos, el tiempo que tardaría en realizar el proceso, y las herramientas con las cuales nosotros vamos a proveer al sistema para que pueda ejecutar esta tarea.

En las siguientes secciones, se discuten las medidas adoptadas para contrarestar estas limitantes.

## 5.3 Técnicas de Reconocimiento Existentes.

Existen diversas técnicas para el reconocimiento de fallas ortográficas, durante los siguientes párrafos analizaremos algunas de ellas, que van desde las técnicas formales usadas por los compiladores, hasta las empíricas, usadas en algunos sistemas comerciales.

### 5.3.1 Compiladores e Intérpretes.

Los compiladores y los intérpretes fueron diseñados con el fin de poder analizar lenguajes de programación, pero como veremos más adelante, algunas de las técnicas que utiliza pueden ser adaptadas al problema del análisis de errores ortográficos.

Comenzaremos por definir lo que es un intérprete y lo que es un compilador. Un intérprete es un sistema, el cual toma como entrada un programa escrito, siguiendo las reglas de algún lenguaje de programación (código fuente), y lo va ejecutando instrucción por instrucción, esto es, no produce ningún tipo de código adicional, sino que simplemente va ejecutando las instrucciones del programa fuente. Por el contrario un compilador, tomando como entrada un código fuente, lo traduce en un lenguaj de bajo nivel como puede ser ensamblador o lenguaje de máquina, el cual posteriormente se ejecuta.

La ejecución de un programa escrito en un lenguaje de alto nivel, es básicamente un proceso de dos pasos. El primero de ellos consiste en compilar el programa fuente y trasladarlo a código objeto, y el segundo en cargar este código objeto a memoria y ejecutar el programa. Para poder llevar a cabo el proceso de compilación se requieren generalmente de siete fases*. Estas fases son:

-Análisis lexicográfico.
-Análisis sintáctico.
-Análisis semántico.
-Generación de código intermedio.
-Optimización de código.
-Generación de código.

Cada una de estas fases es explicada a continuación.

* Se tomó como fuente el Ullman y el Sorenson.

Análisis lexicográfico.

Esta primera fase, también a menudo llamada ''scanner'', separa los caracteres del lenguaje fuente en grupos, los cuales tienen una fuerte relación lógica entre ellos, estos grupos son llamados ''tokens''. Los ''tokens'' suelen ser frases claves dentro de los lenguajes de programación como ''DO'' o ''IF''' etc., también los identificadores o variables de un programa son tokens, operadores como ''<'', ''+'', etc. y los símbolos de puntuación como paréntesis o comas. Al finalizar esta fase, conjuntos de ''tokens'' son pasados a la siguiente fase, análisis sintáctico o ''parser''.

Análisis sintáctico.

En esta segunda fase, grupos de tokens son acomodados dentro de estructuras sintácticas. Por ejemplo, los tres tokens que representa A+B, son agrupados para formar estructuras sintácticas llamadas expresiones. Las expresiones a su vez son combinadas para formar instrucciones. A menudo las estrucuturas sintácticas son acomodadas en árboles, cuyos niveles son los tokens. Los nodos del árbol representan cadenas de tokens que tienen una relación lógica.

Análisis semántico.

El árbol sintáctico producido por el paso anterior, es usado en esta fase para determinar *el significado*, (en un sentido restringido de la palabra), de el programa fuente. Aunque conceptualmente es deseable separar el análisis sintáctico y semántido de un programa fuente, en realidad estas dos fases tienen una fuerte interacción. Nunca el análisis semántico es un proceso diferente y único en un compilador. Por ejemplo para una expresión como:

(A+B)*(C+D)

el análisis semántico, debe determinar que acciones son especificadas de acuerdo a los operadores aritméticos de adición y multiplicación. Cuando el proceso de *parser* detecta operadores de este tipo, es llamada una rútina semántica la cual especificara las acciones a tomas. Esta rútina puede checar que los operandos hayn sido declarados, que sean del mismo tipo y que ambos contengan algún valor.

Generación de código intermedio.

Esta fase utiliza las estructuras dejadas en la fase anterior, para construir conjuntos de instrucciones simples como ''ADD2''. La principal diferencia entre el código intermedio y el lenguaje ensamblador, radica en que en el código intermedio no es necesario

especificar los registros a ser usados en cada operación.

Optimización de código.

Esta es una fase adicional la cual tiene por objeto depurar el código generado en la fase anterior, con el propósito de que sea más eficiente en su ejecución o de reducir el tamaño del código producido.

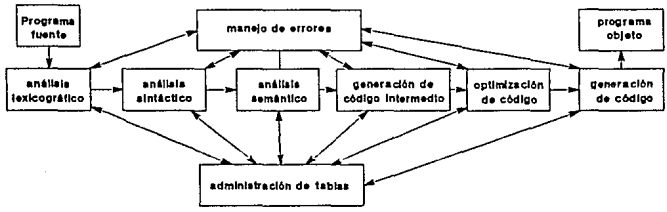



*Figura 5-1*

Generación de código.

Esta es la fase final del proceso de compilación. Durante esta fase se decide en que localidades van a ser almacenados los datos, se selecciona el código para tener acceso a cada dato y se decide sobre que registros van a ser realizadas las operaciones.

Existen además dos proceso paralelos a las cinco fases, estos se explican a continuación.

Administración de tablas.

Es la parte del compilador que guarda los nombres de las variables usados por el programa, y su tipo.

Manejo de errores.

Esta parte es invocada cuando se detecta algún error en cualquiera de las fases. Esta parte es la encargada de pasar la información a las fases posteriores para su adecuado manejo, lo mismo que reportarlo al usuario para su corrección.

En la figura 5-1, se muestra la interrelación que existe entre todas las fases.

### 5.3.2 Técnicas Usadas en Productos Comerciales.

Para poder tener una idea más clara de como las diferentes casas de software habían enfocado el problema, se estudiaron las técnicas que éstas usan para realizar el análisis ortográfico.

Una de las técnicas más populares, consiste en tener un diccionario, el cual puede ir creciendo de acuerdo a las necesidades del usuario. Pero esta técnica tiene la gran deficiencia de no hacer un análisis ni sintáctico ni semántico del texto bajo estudio, además de depender en gran parte de la buena o mala ortografía de la persona que lo esta usando. Otra deficiencia consiste en que estos sistemas son lentos debido a su uso extensivo de accesos a disco. Todos los sistemas estudiados cuentan con este método, y algunos además usan otras técnicas alternativas.

Los diseñadores del sistema CPLUS (MR), además de contar con un diccionario de datos, extrajeron una relación de errores comúnes, y de éstos los que son más fáciles de analizar, de manera que al estar trabajando en el editor de este sistema, tiene una verificación en linea de errores del tipo: terminaciones en ''ción'' y ''sión'' que no estén acentuadas, algunas palabras de uso común y su ortografía, etc..

Por su parte los diseñadores del sistema ESCRIBIEN (MR), además de incluir también el diccionario de datos, introdujeron la característica de que al momento de estar haciendo la verificación, si alguna palabra no la reconoce la reporta, al momento de reportarla le dá la opción al usuario de analizar verbos regulares e irregulares más comunes, junto con sus posibles conjugaciones.

Los demás paquetes analizados se concretaban a atacar el problema de reducir el tiempo de acceso a disco, para conformarse con el diccionario de datos, como spellstar, multimate, bilingüe de selecciones, etc..

## 5.4 Tecnología de Sistema Expertos.

Como se comentó en el capítulo I, el objetivo de la Inteligencia Artificial, es emular algunas de las funciones cerebro humano, a tráves del uso de modelos computacionales. Entre estas funciones se encuentra el análisis y procesamiento de lenguajes naturales.

Pero como también se mencionó en el capítulo IV, el procesamiento de lenguaje natural se encuentra restringido al conocimiento que se le pueda dar al sistema, de el mundo real.

Para poder darle este conocimiento del mundo, o cuando menos un subconjunto de el, nosotros debemos usar un sistema experto.

## 5.4.1 Conceptos básicos de Sistemas Expertos. (SE).

En la figura 5-2, se muestran los principales componentes de un SE, estos se explican en detalle a continuación.

USUARIO.

Es el encargado de proveer al sistema de conocimiento, además de interactuar con él.

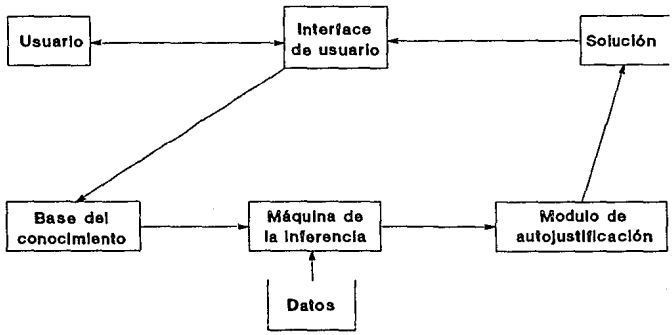



*Figura 5-2*

INTERFACE CON EL USUARIO.

Este modulo se encarga de actuar como interprete entre el usuario y el sistema. Este es uno de los modulos primordiales, ya que constituira la imagen del sistema ante el usuario.

BASE DEL CONOCIMIENTO.

En este modulo, se captura el conocimiento que provee el usuario. Una vez capturado es convertido a un esquema, de los ya análizados en

el capítulo III, tales como:

- Lógica de predicados
- Frames
- Manuscritos
- Reglas de producción.

MAQUINA DE INFERENCIA.

Este es otro de los modulos fundamentales de un SE, ya que es el encargado de proveer soluciones, en base al conocimiento almacenado. Para esto se pueden usar algunos metódos de búsqueda de los ya vistos en el capitulo dos, usando ya sea razonamiento progresivo o regresivo.

JUSTIFICACION.

Este es otro de los modulos que hace distintivo a un SE, ya que este debe de ser capaz de poder justificar sus acciones, en base al conocimiento con el cual cuenta.

RESULTADOS.

En esta parte, el resultado del problema así como su justificación, es propuesto al usuario para su análisis. Haciendo uso para esto del modulo de interface con el usuario.

### 5.4.3 Desarrollo de Sistemas Expertos.

El metódo más distintivo en la programación de SE, es a tráves de la construcción de prototipos rápidos, en contraste con las técnicas tradicionales de desarrollo de sistemas. Estos prototipos, deben de ser conceptualizados como aprovechamientos heurísticos en el desarrollo de SE, y se basan en la premisa de que un sistema rudimentario que satisfaga las necesidades iniciales de un usuario, puede ser construido rápidamente y además a partir de este modelo inicial, se puede ir expandiendo y refinanado iterativamente el sistema.

#### 5.4.3.1 Aprovechamientos del Desarrollo.

El prinicpal foco de atención de un prototipo, es el usario en prospecto. La estrategia para un rápido desarrollo, se centra en dos ideas. La primera de ellas, consiste en obtener una cooperación absoluta por parte del usuario mientras que la segunda estriba en perfeccionar el prototipo para que cumpla con las necesidades del usuario. Para llevar a cabo esta estrategia, se sugieren los siguientes pasos:

- Involucrar al usuario desde el inicio del proyecto.
- Identificar areas funcionales para la construcción del prototipo.
- Establecer requerimientos claros a ser satisfechos por el prototipo.
- Obtener el primer modelo rápido, probarlo e iterar junto con el usuario.
- Enfatizar ante el usuario, que el objetivo del sistema NO ES REMPLAZARLO.
- Mantener un equipo pequeño de desarrollo.

Cada uno de estos puntos será desarrollado a fondo en los siguientes parrafos.

### 5.4.3.1 Involucrar al usuario desde el inicio del proyecto.

Debido a las areas de aplicasión que comunmente son atacadas por los SE, los sistemas a desarrollar normalmente son confusos e indefinidos, de modo que la única forma de construir un prototipo, es invitando al usuario a juntas, en las cuales se determinarán, las areas funcionales de la operación diaria, que más afectan de acuerdo a parámetros tales como costo ó tiempo que se llevan para su consecución. La figura 5-3 ilustra este proceso.

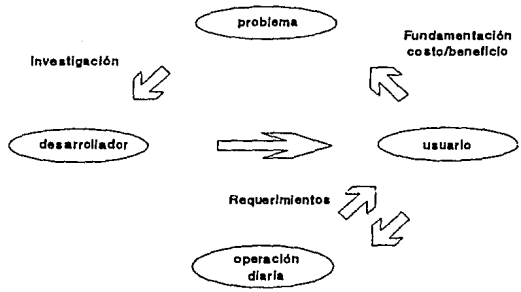



*Figura 5-3*

La idea principal en este punto, y sobre la cual se debe enfatizar es que el usuario es el *EXPERTO* en la operación del trajabo, y su *CONOCIMIENTO* debe de ser obtenido, además de que en la gran mayoria de las ocasiones, es la única fuente de información para el equipo de desarrollo.

### 5.4.3.2 Identificación de Areas Funcionales.

La identificación de areas funcionales criticas, es de vital importancia, ya que una adecuada selección asegura el exito de la implementación. El usuario debe de ser consultado desde los inicios del proyecto, para que nos ayude a identificar aquellas areas donde la ayuda sea más necesaria.

Una vez que estas areas han sido identificadas, el primer prototipo debe de ser construido para que asista al usuario, liberandolo con ello de el peso de estas funciones, volviendo con esto al usuario más creativo en su trabajo.

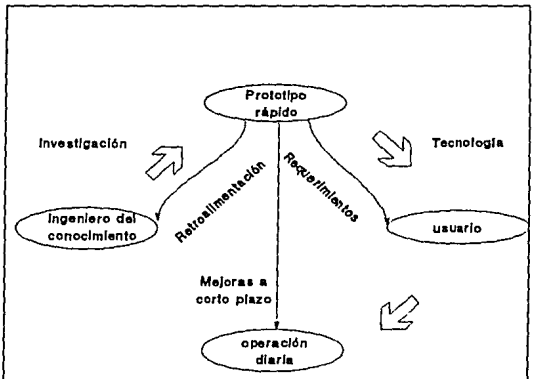



*Figura 5-4*

La figura 5-4, demuestra un ejemplo de un proceso de este tipo.

### 5.4.3.3 Establecimiento de Requerimientos.

En este paso, el usuario y el equipo de desarrollo establecen los lineamientos de los requerimientos que deben de ser satisfechos por el prototipo para que este pueda ser aceptado.

Dependiendo de la aplicación, los requerimientos pueden incluir caracteristicas fisicas tales como:

- Velocidad
- Presición
- Seguridad
- Parámetros de ejecución
- Apariencia
  + Manejo de dimensiones
  + Color
- Interface con el usuario
  + Lenguaje Natural
  + Gráficos
  + Reconocimiento de voz

Diversas iteraciones pueden ser requeridas para refinar los requerimientos.

Es de suma importancia, que al final de esta fase, queden completamente claros y por escrito los requerimientos, ya que una gran cantidad de fracasos en proyectos de software pueden ser atribuidos a falsas expectativas creadas en el usuario.

### 5.4.3.4 Desarrollo del Primer Modelo.

Los pasos anteriores, habran creado una gran expectación en el usuario. En este paso esa expectativa es parcialmente satisfecha al proveerlo de un primer modelo para que el lo pueda provar. Funcionalmente este primer modelo, no debe de estar completo, sin embargo es de vital importancia que ya cuente con una interface de usuario efectiva, con gráficos y comandos en lenguaje semi-natural. Este detalle asegurará que el usuario no se sienta desilusionado inmediatamente.

Un prototipo rápido, sirve como un aprovechamiento formal para proveer al usuario de un desarrollo incremental paso a paso.

### 5.4.3.5 Enfatizar el Objetivo del Sistema.

Para poder obtener una cooperación completa por parte del usuario, es crucial **remarcarle** el hecho que el objetivo del sistema no es de ninguna manera el remplazarlo a el, sino que por el contrario la finalidad del mismo, es ayudarlo a ser más productivo y creativo en sus actividades cotidianas. Este paso crucial en la estrategia, puede fácilmente ser ignorado, resultando en un fracaso del prototipo debido a la falta de cooperación del usuario.

### 5.4.3.6 Mantener un equipo pequeño de desarrollo.

El desarrollo de prototipos rápidos por grupos grandes es una tarea complicada. Un equipo, de desarrollo debe de estar compuesto por no más de tres o cuatro miembros, exluyendo con esto la necesidad de elaborar complejos metódos de administración de proyectos. Un equipo grande ocasiona diversas dificultades como burocracía, disperción de objetivos, necesidad de puntos de control, mala comunicación entre componentes y en general una gran cantidad de trabajo adicional. Todas estas características son contrarias a la filosofía de prototipos la cual busca: gran velocidad de desarrollo, gran comunicación entre los integrantes y un bajo costo de desarrollo.

### 5.4.4 Reglas de Diseño en SE.

Una vez que hemos analizado los seis pasos fundamentales para un sano aprovechamiento de la fase de desarrollo, pasaremos a análizar las reglas de diseño que deben de gobernar el desarrollo de SE.

Existen cinco principios básicos en el diseño de prototipos, estos son:

- Modularidad
- Programación orientada a objetos
- Máquina de Inferencía simple
- Redundancia de Conocimiento
- Interface hombre-máquina eficaz

Estos cinco principios son explicados a continuación.

#### 5.4.4.1 Modularidad.

La modularidad se refiere a construir los componentes en los prototipos como modulos que pueden ser usados una y otra vez. Estos modulos consisten en ''cajas negras'', que tienen ciertas caracteristicas de entrada/salida y pueden ser recombinadas para ser usadas en otros prototipos. Los modulos por definición son portables. Por ejemplo, si el desarrollo de un prototipo no es satisfactorio, entonces el modulo del conocimiento puede ser transferido a un nuevo prototipo. La modularidad implica también, una representación consistente del conocimiento.

#### 5.4.4.2 Programación Orientada a Objetos.

En programación orientada a objetos, las entidades en un programa son vistas como objetos, que se comunican con otros objetos vía mensajes. Cada objeto tiene propiedades distintas y procedimientos o reglas asociadas a él. La programación orientada a objetos puede ser altamente modular, puede ejecutar acciones locales como desplegados de información o auto-modificaciones y puede recibir y enviar información desde y hacía otros objetos. Uno de los propositos de la programación orientada a objetos, es también la reusabilidad del código, ya que objetos representando conocimiento pueden ser usados una y otra vez en el mismo prototipo o en otro prototipo.

#### 5.4.4.3 Máquina de Inferencía Simple.

Como el poder radica en el conocimiento, entonces, la inferencia no es critica en prototipos. Una simplificación de la máquina de

inferencia es útil en dos formas. Primero, las explicaciones son faciles de producir con una máquina simplificada, lo cual asegura que un menor esfuerzo es necesario para generar explicaciones comprensibles que el usuario pueda entender. Segundo, la adquicición del conocimiento es simple, ya que se requiere de menos esfuerzo para determinalr el concimiento exácto a ser agregado para mejorar la ejecución del sistema. El trabajo de construir prototipos se vuelve menos complicado y además, los errores pueden ser más fácilmente detectados cuando la máquina de inferencia es simple.

### 5.4.4.4 Conocimiento Redundante.

Redundancia en el conocimiento, es el cuarto principio, y es un conveninte remedio para conocimiento incompleto o inexácto, situaciones comunes si tomamos en cuenta que aún los expertos humanos, no siempre conocen exactamente todo lo que su area comprende. Para construir redendancia de conocimiento en prototips, el equipo de desarrollo debe de consultar varias fuentes para obtener conocimiento multiple y con traslapes. El uso apropiado de fuentes de conocimiento diversas, enriquecera la representación almacenada en la base del conocimiento.

### 5.4.4.5 Interface Hombre-Máquina Eficaz.

El último principio de diseño, es sin embargo el más importante. Sin este, los prototipos no pueden ser usados y probados con propiedad y eventualmente pueden ser olvidados debido a la dificultad de su uso. Una interface eficaz debe incluir un efectivo desplegado de datos de entrada y salida, y funciones fáciles de usar.

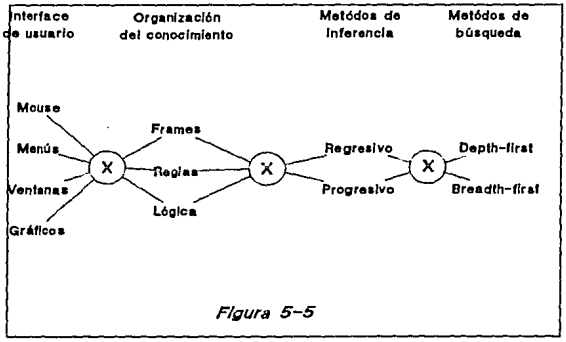



*Figura 5-5*

Estos elementos pueden fácilmente ser ignorados por los diseñadores, quiene a menudo se encuentra tan en contacto con la computadora, que olvidan que el usuario puede ser un completo iletrado en computación. Una excelente interface hombre-máquina, ofrece el único seguro de que el prototipo desarrollado será aceptado por el usuario.

Para terminar con esta sección, en la figura 5-5 se muestra en forma gráfica la estructura de un sistema experto.

## 5.5 Adecuación de Técnicas al Problema Ortográfico.

Después de todo lo anteriormente expuesto, llegamos al punto de decidir, como adecuar las técnicas anteriormente expuestas, a nuestro problema específico, o sea como usarlas para verificar la ortografía de un texto.

### 5.5.1 Analizador Sintáctico.

Partiendo de el punto, de que la mayoría de las reglas de producción del lenguaje español son bastante ambigüas, y a menudo dependen del contexto en el que se encuentre, la totalidad de un texto no se podría analizar simplemente usando las técnicas usadas en la construcción de compiladores. Pero sin embargo un subconjunto de ellas nos son de gran utilidad. En especial la fase del análisis sintáctico, ya que si bien el lenguaje permite una gran variedad de construcciones, no existe ninguna regla que permita construcciones del tipo ''verbo verbo verbo'' o ''artículo artículo artículo'', o cadenas del tipo ''aaaa....'' o ''bbbb....'', por lo que malas construcciones y cadenas sin sentido puedan ser detectadas facilmente.

Y si a esto nosotros le agregamos un SE, el cual basandose en una red ATN, nos pueda análizar sintácticamente, sino un cien por ciento si cuando menos un ochenta por ciento, nosotros le habremos agregado al sistema una capacidad que no se ha encontrado hasta el momento en ninguno de los sistemas comerciales análizados.

### 5.5.2 Análizador Semántico.

Como se vio en el cápitulo IV, un análisis sintáctico, forzozamente debe de hacer uso de una análisis semántico, por lo cual este no puede ser excluido del todo. Sin embargo, dadas las restricciones con las que contamos, aunque podamos proveer a nuestro sistema de esquema como el de DC, no podemos garantizar un análisis a un cien por ciento. Sin embargo esta idea es bastante atractiva, y muy bien puede ser combinada con el análizado sintáctico del punto anterior dentro de un SE.

### 5.5.3 Técnicas Comunes Usadas en Otros Sistemas.

Dentro de las técnicas usadas por los distintos paquetes analizados, existen un subconjunto de ellas, las cuales implican ideas muy interesantes.

La primera de ellas es el uso de un diccionario de datos, ésta es un idea interesante, ya que provee al sistema de un conomiento inicial del mundo que lo rodea, amén de ser algo muy fácil de implementar.

La segunda de ellas es extraer un subconjunto con los errores ortográficos más comunes, ésto nos permite eliminar de entrada un alto porcentaje de los errores que puede contener un documento.

La tercera, es un variación sobre la primera, y presupone dos diccionario, el primero de ellos contiene palabras de uso común como articulos y adjetivos, y reside de manera permanente en memoria, y el segundo de ellos, es una estrutura masiva con el grueso de las palabras que contiene un diccionario. El seguir esta estrategia nos permite optimizar el tiempo de ejecución del sistema ya que reduce en gran medida el acceso a disco.

## RAZONAMIENTO DE PROBLEMAS ORTOGRAFICOS.

Hasta estos momentos, hemos visto algunas técnicas, las cuales van desde esquemas formales hasta esquemas empiricos.

Normalemente los sistemas empiricos son los que le dan a un sistema una gran velocidad de ejecución, ya que necesariamente parte de errores comúnes y fáciles de resolver. Asi que a continuación vamos a pasar a análizar algunas reglas practicas que nos pueden ser útiles para la solución de problemas ortográficos.

### 5.6.1 Reglas Ortográficas.

Existen las suficientes reglas ortográficas como para hacer un tratado de este tema. Pero para los fines prácticos de este trabajo, las dividiremos en cuatro grandes familias, que son:

- Reglas para el uso de diptongos y triptongos.
- Reglas para un correcto uso de sílabas.
- Reglas de acentuación.
- Reglas para palabras de escritura dudosa.

Cada una de estas familias se describen a continuación.

### 5.6.2 Reglas Para el Uso de Diptongos y Triptongos.

Si una sílaba tiene dos vocales, a la reunión de estas dos vocales se le llama diptongo. Como ejemplo podemos dar las siguientes palabras baile, causa.

Para poder entender mejor como funcionan las reglas para los diptongos y triptongos, es necesario distinguir dos tipos de vocales, las vocales fuertes on a, e, o. Las vocales débiles son i, u, y, (cuando ''y'' es utilizada como vocal, esto es al final de una palabra, como ejemplo podemos citar la palabra ley.)

Podemos nosotros distinguir tres reglas básicas en el uso de diptongos, estas se explican a continuación.

Los diptongos se forman de la siguiente manera:

- Vocal fuerte con débil, por ejemplo gaita.
- Vocal débil con fuerte, por ejemplo novia.
- Vocal débil con débil, por ejemplo cuita.

Cuando la vocal débil esta acentuada, entonces no hay diptongo. Por ejemplo en la palabra leí, el diptongo es roto por la acentuación de la vocal débil.

De la primera regla que dimos, se puede inferir la tercera, esta es: Dos vocales fuertes no forman diptongo, para ejemplificar podemos usar la palabra león, la cual no tiene diptongo por estar formada de dos vocales fuertes.

Sí una sílaba tiene tres vocales, a la reunión de estas tres vocales se le llama triptongo. Para ejemplificar podemos citar las palabras buey, apreciáis.

En el uso de triptongos existen dos reglas básicas, las cuales se explican a continuación.

Los triptongos se combinan de la siguiente manera:

Vocales débil, fuerte, débil.

Como ejemplo podemos citar las palabras Cuautitlán, codiciáis.

La segunda regla nos indica que cuando en una reunión de tres vocales la primera vocal débil está acentuada, entonces no hay triptongo. Como ejemplo podemos citar las palabras apreciaríais, habriais.

### 5.6.3 Reglas Para un Correcto Uso de Sílabas.

Las reglas para el uso correcto de sílabas son un poco más subjetivas, según un texto de curso programado de ortografía consultado en la elaboración de este trabajo(6), para dividir una palabra en sílabas se tiene que recordar lo siguiente.

Primero, que las sílabas se pronuncian en una sola emisión de voz, concepto bastante complicado de automatizar en un algoritmo.

Segundo, que las sílabas pueden tener una, dos o tres vocales.

Tercero, que cuando en una sílaba se reunen dos vocales, a la unión de estas se le conoce como diptongo.

Cuarto, cuando la vocal débil en un diptongo está acentuada, entonces forma sílaba aparte.

Quinto, que dos vocales fuertes forman sílaba aparte, o sea no hay diptongo.

Sexto, cuando una sílaba tiene tres vocales, ésta se llama sílaba con triptongo.

Séptimo, que un triptongo está formado por vocales débil, fuerte, débil.

Octavo, que cuando en la combinación mencionada en el punto anterior, la primera vocal débil se encuentra acentuada, entonces forma sílaba aparte, o sea se rompe el diptongo.

### 5.6.4 Reglas de Acentuación.

Existen tres tipos de acentuación, estas son:

- Acentuación prosódica.
- Acentuación ortográfica.
- Acentuación diacrítica.

La acentuación prosódica se utiliza en quellas palabras que se acentúan solo al pronunciarlas y no al escribirlas, como ejemplo podemos citar las palabras libro, Teresa.

La acentuación ortográfica es aquella que se refleja al momento de escribir la palabra, como ejemplo podemos citar las palabras

sílaba, acentuación.

Tanto para la acentuación prosódica u ortográfica, por la sílaba donde se ubica el acento, nosotros podemos distinguir diferentes tipos de palabras:

-Palabras agudas. Todas las palabras que tienen acento ortográfico o prosódico en la última sílaba se llaman palabras agudas. Como ejemplo podemos citar las palabras podré, fusil.

-Palabras graves o llanas. Aquellas palabras que tienen acento ortográfico o prosódico en la penúltima sílaba, son conocidad como palabras graves, como ejemplo tenemos las siguientes palabras: nácar, examen.

-Palabras esdrújulas. Son aquellas palabras que tienen ubicado el acento en la antepenúltima sílaba, podemos ejemplificar con las palabras práctica, ráfaga.

-Palabras sobreesdrújulas. Todas las palabras que van acentuadas en la sílaba anterior a la antepenúltima. Como ejemplo tenemos las palabras pídanmelo, entréguensela.

La acentuación diacrítica se refiere a aquellas palabras que se escriben igual, pero que tienen diferente significado, y se usa el acento diacrítico para distinguir un significado de el otro. Como ejemplo tenemos las palabras dé y de, la primera se acentua por ser del verbo dar, mientras que la segunda no se acentua por ser preposición.

Para los fines prácticos de este trabajo centraremos nuestra atención en las acentuaciones ortográficas y diacríticas.

### 5.6.5 Reglas Para Palabras de Escritura Dudosa.

En este punto, y debido a la gran incidencia de errores de este tipo, centraremos nuestra atención en las reglas para el uso correcto de las letras b, v, c y s.

Se utiliza siempre ''b'' antes de consonante, como en las palabras antebrazo, abnegación.

Se escribe ''b'' siempre después de m, podemos ejemplificar con las palabras combate, embajada.

Se escribe ''v'' siempre después de b y n, ejemplificaremos con las palabras obvio, invariable.

Se escribe con ''c'' las palabras que terminan en ción y que provienen de otras que terminan en to, como ejemplo podemos citar las palabras atención que proviene de atento y corrupción que proviene de corrupto.

Se escribe con ''s'' las palabras que terminan en sión y que provienen de otras que terminan en so, podemos ejemplificar con las palabras tensión que proviene de tenso, profesión que proviene de profeso.

Por supuesto no se incluyeron todas las reglas que regulan el uso de estas letras, sino las que son susceptibles de automatizarse.

### 5.6.6 Razonamiento de Fallas Ortográficas.

Las fallas ortográficas comúnmente detectadas en un texto, se pueden dividir en dos grupos.

El primero de ellos se refiere a errores de ''dedo'', estos errores son muy comúnes, y la manera de resolverlos o de poder crear soluciones a ellos es por la asignación de palabras similares.

El segundo grupo y el más complicado de analizar es el de los errores por fallas ortográficas propias del usuario del sistema. Dentro de estas, un alto porcentaje de ellas, ocurren por confusión entre las letras c y s, o b y v, y por mala acentuación de palabras. De ahi el énfasis que se puso en el inciso anterior a estas reglas.

Como ya se mencionó solo algunas de estas reglas son fácilmente detectables por medios automatizados, pero otras no, ya que dependen del contexto de la frase, y nos referimos en especial a la acentuación diacrítica.

La mejor manera de resolver este problema es el de concientizar al usuario, sobre cuando se debe y cuando no se debe acentuar una palabra. En otras palabras, que aprenda ortografía con el sistema.

### 5.6.7 Reconocimiento de Excepciones a las Reglas.

Todas las reglas ortográficas tienen una o más excepciones. El tratar con estas excepciones es una tarea bastante complicada, ya que antes de verificar si una palabra cumple con la regla correspondiente, primero se tiene que verificar si la palabra en

cuestión no es excepción de la regla.

Debido a ésto sólo se tomaron en cuenta las excepciones de las reglas más comunes. Estas se listan a continuación junto con la regla de la cual son excepción.

Se escribe con ''b'' los verbos que terminan en aber, eber, y bir. Se exceptúan los verbos hervir, servir y vivir.

Se escriben con ''v'' las palabras compuestas que empiezan con pre, pri, pro. Se excluyen las palabras prebenda, preboste, probidad, probar y probabilidad.

Se escriben con ''c'' las palabras que terminan en cia, cie, áce, áceo. Exceptuando los nombres de algunos paises o continentes como Rusia, Asia, Indonesia.

Se escriben con ''c'' las palabras que terminan en cer, cir, y ciar. De esta regla se excluyen algunos verbos como los siguientes: coser, asir, extasiar.

Se escriben con ''z'' las palabras que terminan en azgo, izco y uzco. Exceptuando las siguientes formas verbales: busco, Ofusco.

### 5.6.8 Modismos.

El uso de modismos, es algo lo bastante común, como para no ser tomado en cuenta. Sin embargo con un medio de aprendizaje bien diseñado cualquier sistema puede estar capacitado para incluirlos dentro de su medio ambiente.

Sin embargo dentro de estos existe una clase especial, estas son las abreviaturas. Las abreviaturas son muy usadas debido a la comodidad que su uso implica. Dentro de estas, sin embargo, se pueden extraer un subconjunto de las más comunes que pueden ser incluidas para evitarle al usuario, el tener que darlas de alta dentro del medio ambiente del sistema.

# CAPITULO VI

# ORTHOS

## 6.0 Introducción.

En los capítulos anteriores, nosotros hemos sentado las bases y conocimiento necesarios, para poder desarrollar un sistema de acuerdo a ciertas características y restricciones.

En el presente cápitulo, presentamos la fase de análisis del sistema de acuerdo a la metodología de Yourdon. Basicamente se presentan los siguientes documentos:

- Reporte de Investigación,
- Reporte de Alternativas de Diseño,
- Diagramas de Flujo de Datos,
- Diccionario de Datos.

## 6.1 Reporte de Investigación.

### 6.1.1 Prefacio.

En esta sección, se encontrarán con el reporte de investigación, el cual describe un nuevo desarrollo de software ORTHOS. El propósito de esta sección, es el de presentar un análisis objetivo de los actuales productos en producción, identificar los problemas y oportunidades, definir el alcance del proyecto, explicar el estado actual del proyecto junto con conclusiones y recomendaciones.

Este documento esta intentado para diferentes audiencias: para un equipo técnico que pudiera participar en fases futuras del proyecto, para los usuarios finales, quienes puedan dar valiosas recomendaciones para futuras versiones y para inversionistas que pudieran apoyar al equipo de desarrollo en futuras fases.

### 6.1.2 Identificación del Proyecto.

#### 6.1.2.1 Nombre del Proyecto y Mnemónico.

Este proyecto fué iniciado, en reconocimiento de la necesidad de contar con una herramienta poderosa y automática para la corrección de

documentos. Diversas casas de software, han producido diversos productos, sin embargo la mayor diferencia entre este desarrollo y los demás, radica en la utilización de técnicas de inteligencia artificial.

El mnemónico para el proyecto es ORTHOS, el cual es la primera raiz etimológica de ortografía, y la cual significa recto, fué escogido dada la orientación principal que se le trata de dar al sistema, y esta es la de lograr que los usuarios de él cuentan con una ortografía recta.

### 6.1.2.2 Resumen del Proyecto.

Este proyecto, fué iniciado en reconocimiento de la necesidad de contar con una herramienta de procesamiento de palabra, la cual pudiera contemplar varias características inherentes al lenguaje español.

La mayoria de los desarrollos de software que existen en este momento en el mercado, se limitan únicamente a atacar algunos de los procesos que el proceso de palabra envuelve, sin embargo, dejan otros de ellos completamente de lado. Esto hace que la mayoria de ellos cuenten con algunas ventajas y otras desventajas. Aunado a lo anterior, esta el hecho de que también la gran mayoria de estos desarrollos, nacierón originalmente en paises de habla inglesa, por lo cual no contemplan muchas de las características que son inherentes al lenguaje español. Y por último diversos de estos desarrollos, se encuentran aislados, ya que no proveen la manera de migrar la información que ellos producen de un producto, a otro, lo cual en un ambiente tan altamente competitivo con lo es el de desarrollo de software, constituye una seria desventaja.

Otro elemento que influencio de manera definitiva la creación de este paquete, fué el hecho de poder ofrecer al mercado, un producto altamente versátil, el cual desde su inicio fuera concebido con las técnicas más modernas en el campo de desarrollo de software.

De manera que en resumen, este proyecto nace bajo las siguientes premisas:

- Ofrecer un producto versátil, el cual se pueda adaptar fácilmente a diversas circunstancias.

- Ofrecer un producto innovador, el cual ofresca varios conceptos nuevos al mercado.

- Ofrecer un producto de inmejorable calidad.

- Ofrecer un producto altamente competitivo en su relación precio / desempeño.

## 6.1.3 Análisis del Problema.

### 6.1.3.1 Descripción de los Sistemas Actuales.

El proceso de palabras, fué de los primeros productos de oficina, que empezaron a usarse de una forma generalizada. Más aún, con el surgimiento de los computadoras personales, este tipo de paquetes tuvieron una gran aceptación dentro de las oficinas, ya que a un costo relativamente bajo se podían adquirir tanto una computadora de este tipo, como el software básico para su uso.

Los primeros paquetes que surgieron, eran bastante complejos en su uso, ya que basicamente estaban orientados a personal de formación técnica, ya que la gran mayoria de ellos, antes de ser comercializados, fueron desarrollados por personal especializado como una ayuda para el mejor desarrollo de sus funciones.

Posteriormente, y conforme la computación se fué moviendo más de lo técnico a lo comercial, y con ello, el mercado fué demandando más productos orientados hacía usuarios con poco o nada de experiencia técnica, se fueron sofisticando los productos y las características que ellos ofrecian.

Actualmente, existen un sin número de casas de software, las cuales ofrecen una gran diversidad de productos con muy distintas características, de modo que el usuario final, puede elegir entre una gran variedad de precios y funciones.

Sin embargo, este aparente adelanto, ha significado un problema en la gran mayoria de las compañias, ya que al principio, la computación personal significaba una gran ayuda para ellos ya que les quitaba una cantidad importante de requerimientos que atender. Sin embargo con la explosión de productos que se ha dado, principalmente en los últimos años y debido a la aparición de la computadora personal IBM, ha complicado significativamente la labor de soporte y asesoría que los departamentos de sistemas tienen que brindar a sus usuarios, ya que comúnmente tienen que lidiar con una gran cantidad de software distinto e incompatible en cuanto a su uso y manera de almacenar la información, lo cual ha originado que los departamento de sistemas empiezen a regular tanto la marca de computadoras a usar como el software que debe de contener cada computadora.

Esta misma problemática ha originado en las casas de desarrollo de software el fenómeno de tratar de imponer un estandar en la industria, lo cual hasta estos momentos no se ha dado, sin embargo la gran mayoría

de paquetes ya traen integrada la capacidad de poder exportar datos en formatos estandar como ASCII o DIF (Data Interchange Format).

En lo que se refiere exclusivamente a los paquetes de procesamiento de palabra, estos avances permiten que los usuarios de hoy en día, cuenten con paquetes en una diversa gama de opciones, desde paquetes básicos de edición hasta paquetes de composición sumamente sofisticados.

Sin embargo en algunas cosas todos ellos siguen siendo primitivos, y estas es, en la capacidad de analizar de una manera real el contenido sintáctico y semántico del texto en si mismo.

Esto es, todas las casas de software han trabajado para tratar de que sus paquetes sean faciles de usar, que permitan una presentación elegante de documentos e incluso algunos de ellos ofrecen procesos de verificación de palabras en base a diccionarios, pero ninguno de ellos ha tratado de ir más allá. La razón es simple, ya que el realizar un análisis del contenido de un documento, es una tarea que implica varios años de esfuerzo, y la manera en como generalmente funciona la industria del software, es en espera de que los laboratorios realizen la labor de investigación, y ya que esta rinde frutos, entonces pasa al mercado común y en ese momento todas las principales casas entrar a la labor de desarrollo, en base a las experiencias obtenidas en los laboratorios.

### 6.1.3.2 Deficiencias de los Actuales Sistemas.

Debido a aspectos de índole técnico, no existe practicamente ningún sistema de software que no sea perfectible o universal. Simplemente desde la fase de diseño, la mayoria de los sistemas son encuadrados dentro de una frontera de alcances, ya que de otra manera el sistema estaría perpetuamente en desarrollo.

De modo que todos los sistema cuentan con alguna deficiencia, pero en esta sección, nos enfocaremos a las que se consideran más suceptibles de ser perfeccionadas.

Dado que la gran mayoria de los productos de software que se encuentran actualmente en el mercado, fueron originalmente diseñados en paises de habla sajones, muy pocos de ellos toman en cuenta las caracteristicas inherentes al lenguaje español.

A parte, ninguno de ellos realiza un real análisis del texto, incluso mínimo, de modo que los que cuentan con algún tipo de verificación básica se conforman con incluir un diccionario de palabras, el cual muchas veces se encuentra en ingles.

Esto hace que los usuarios no corrijan en ningún momento sus vicios ortográficos o en el mejor de los casos los corrijan de una forma muy pobre.

Esto aunado al hecho de que contar con una forma de escritura correcta, no se le presta una gran importancia ha redundado a que actualmente muy pocas personas cuenten con una buena ortografía y aún más escasas las que redactan en una forma adecuada.

### 6.1.4 Análisis de Mercadotécnia.

Los productos de procesamiento de texto, fueron de los primeros en aparecer en la escena de productos para el mercado de oficina.

En estos momento existen un sinnúmero de paquetes, cada uno con distintas características tanto de precio como de capacidades.

En el presente documento nos enfocaremos a aquellos que cumplan con funciones similares a las que se pretende ofrecer con este desarrollo de software.

El más sencillo de los analizados, es Executive Memomaker de Hewlett Packard, es básicamente un editor muy simple con algunas características interesantes como justificación, remarcado, subrayado, manejo de bloques de texto, interfase de usuario muy consistente con otros productos de la misma índole ofrecidos por HP, viene de fábrica con un diccionario en ingles para verificación y sugerencia muy primitivo pero bastante sencillo en su uso.

Los siguientes en la escala son Wordstar, Multimate y bilingüe, los cuales son procesadores más sofisticados, pero en el aspecto de verificación, son igual de primitivos que el analizado en el párrafo anterior.

El siguiente sistema es CPLUS, este sistema fué creado en México. Básicamente como procesador de textos cuenta con características similares a los ya analizados, sin embargo en la parte de verificación, cuenta con algunos aspectos interesantes ya que lleva a cabo una verificación en línea al momento de estar capturando datos en el editor. El tipo de errores que reporta, son comúnes como palabras con terminación en ''sión'' y ''ción'', no acentuadas y algunas palabras de uso común.

El último sistema y el más sofisticado en la parte de verificación es Escribien, el cual además de contar con un diccionario, provee algunas características adicionales como conjugación de verbos.

En cuanto a precios y soporte por parte de la casa de software, varían mucho desde los 50.00 dolares hasta los 300.00 dolares, y en cuanto al soporte varía también de acuerdo al tamaño de la casa de software.

Para finalizar esta sección, es importante remarcar que el análisis se enfocó a sus características de verificación y no a su poderío como editores de texto.

### 6.1.5 Propuesta de Alcances para el Sistema.

#### 6.1.5.1 Objetivos del Sistema.

Incrementar la productividad del usuario, reduciendo para ello el número de errores de índole ortográfico antes de la producción de un documento.

Incrementar la satisfacción del usuario, al evitarle repeticiones innecesarias de trabajo debido a error en los documentos.

Corregir paulatinamente los vicios ortográficos que tenga el usuario, a través de módulos de autoenseñanza que puedan ser utilizados a gusto del usuario.

Proteger la inversión de los posibles clientes al proveer interfases con productos que el cliente previamente haya adquirido.

Hacer una sistema con innovación tecnológica, al utilizar técnicas de vanguardia en el campo de computación administrativa.

#### 6.1.5.2 Entorno Propuesto de Interacción.

ORTHOS debe de ser diseñado, para operar en un ambiente clásico de oficina, en conjunto con otro productos de diversos proveedores.

Básicamente nosotros nos enfocaremos a tres tipos de usuarios, cada uno de ellos es descrito a continuación.

***Personal de Apoyo.***

Este personal, dentro de los cuales entran secretarias, oficinistas y personal de apoyo administrativo, regularmente requieren

de un proceso de palabra elemental, el cual les permita crear documentos sencillos sin demasiada sofisticación de edición.

***Profesionistas y Gerentes.***

Este personal, se caracteriza por la necesidad de crear documentos los cuales a menudo incluyen material de apoyo gráfico.

***Profesionistas de Mercadotécnia.***

Este tipo de personal, requiere de la creación de documentos con una alta sofisticación de edición, creado de documentos en extremo profesionales para campañas de mercado, revistas, etc.

ORTHOS les debe de proveer a cada uno, una solución a la medida de sus necesidades, ya sea a través de las diversas funciones del producto en si mismo, o a través de interfases con otros productos del mercado.

### 6.1.5.3 Características del Sistema.

Existen una multitud de características que pueden ser incorporadas dentro del sistema ORTHOS. Pero implementar todas ellas va a tomar muchos años de esfuerzo de desarrollo. De modo que el conjunto mínimo de a ser implementado en el primer modelo, se muestra a continuación:

Editor con las siguientes características:
- Remarcado
- Subrayado
- Configuración de diversas impresoras
- Justificado con:
    + espacios
    + silabeo
- Módulo de conversión desde y hacia:
    + Executive Memomaker
    + Windows
    + Archivos ASCII

Verificación en base a diccionario con:
- Sugerencia de palabras
- Función de Corrección
- Función de Mantenimiento a diccionario

Análisis de Textos
- En base a una Red de Transición Aumentada
- Función de Mantenimiento a la Red

- Módulo de auto justificación

Maestro
- Cinco lecciones

Ayuda
- En forma gráfica

Conjunto adicional de características a ser implementadas en las siguientes versiones:

Integración al entorno de operación de MS-Windows

Sofisticación del módulo de análisis

Funcionamiento en red

Funcionamiento bajo OS/2

### 6.1.5.4 Requerimientos del Sistema.

El sistema debe de ser capaz de funcionar sobre hardware estandar de computadoras personales, desde el modelo XT hasta modelos basados en el procesador INTEL 80386 y posteriores.

En lo referente al sistema operativo, debe funcionar a partir del MS-DOS 3.2 en adelante.

En cuanto a la construcción en si mismo del sistema, este debe de ser cuidadosamente diseñado para facilitar el mantenimiento del sistema en base a liberaciones de solo algún o algunos módulo para mejorar la ejecución del mismo, entre liberaciones mayores.

La arquitectura del sistema debe de ser capaz de segmentar y almacenar desde 1000 hasta varios cientos de miles de entradas en el diccionario sin demeritar por ello el funcionamiento del sistema en general.

El sistema debe de procesar un gran número de transacciones primarias en un tiempo razonable, ya que de no lograr un tiempo de respuesta adecuado, el usuario puede verse desilusionado del sistema muy rápido.

### 6.1.5.5 Limitantes del Sistema.

Cualquier sistema de verificación ortográfica, va a desempeñarse con la misma precisión con la que el usuario lo actualice, y si esto es de vital importancia en los sistemas de verificación tradicional, es aún de mayor importancia en un sistema de análisis más complejo.

De forma que este módulo en especial, tiene que se muy bien planeado principalmente en los subsistemas de mantenimiento al diccionario de datos.

## 6.1.6 Conclusiones.

### 6.1.6.1 Consideraciones de Diseño.

Para que el presente producto pueda destacar con algo distintivo entre los demás productos que se puedierán considerar como competencia, es necesario que nuestro producto, cuente con un sistema experto, el cual le ayuda en la parte de análisis de documentos, en base a cierto conocimiento de estructuras gramaticales previamente definidas.

Por otro lado, y tomando en cuenta lo visto en los capítulos anteriores, se hace patente, que la mejor estrategia disponible, es el uso de redes de transición aumentada para llevar a cabo el análisis ortográfico. Sin embargo esto no va a bastar para realizar un análisis completo inteligente por las limitaciones de índole técnico ya analizadas.

### 6.1.6.2 Resúmen de Funciones.

La función principal del producto, va a ser la de permitir una verificación y análisis ortográfico de textos, en base a un diccionario y a una red de transición aumentada.

Como segunda función, deberá de proveer enseñanza y asesoría al usuario, para que este pueda ir mejorando paulatinamente sus vicios ortográficos.

Como funciones secundarias, deberá de contar con un módulo de procesamiento de palabra básico, el cual le permita al usuario crear, editar, corregir e imprimir textos.

### 6.1.6.3 Interfases.

En el mercado actual, un producto de software que no cuente con medios para interactuar con su medio ambiente, se encuentra en desventaja competitiva con respecto a otros productos que si permiten este tipo de acciones.

De modo que como una medida estratégica de mercadotécnia, el producto deberá de contar con interfases con otros productos de edición de textos, de manera de permitir la interacción con otros productos.

Por razones de fuerza de mercado, se decidió, que el sistema contara con interfase con MS-Windows y pudiera comunicarse con otros paquetes a través de archivos en formato estandar ASCII.

### 6.1.6.4. Consideraciones de Almacenamiento en Disco.

Dado que el sistema debe de ser capaz de almacenar una gran cantidad de datos, sin demeritar en su ejecución, el modo de almacenamiento de consulta, es de vital importancia.

Esto tiene fuertes implicaciones técnicas, en cuanto a su manejo, ya que deberá de contar con algún tipo de estructura de datos que le permita al sistema el acceso a cualquier parte del diccionario de una forma casi inmediata.

### 6.1.6.5 Impacto del Sistema.

Sumario de las características propuestas:

Editor básico de textos.

Interfase con MS-Windows y a través de archivos ASCII

Verificación en base a diccionarios

Análisis en base a una red de transición aumentada

Módulo de Auto aprendizaje

Interfase de usuario en base a menus

Análisis de Beneficios Anticipados.

El primer, y consideramos el que justifica todo el proyecto, es el reducir los vicios ortográficos de los usuarios del sistema.

El segundo, y es justificación de índole económica, se basa en la reducción del reproceso de documentos debido a errores de índole ortográficos.

Análisis de Costos Anticipados.

La primera fase del proyecto se llevará sobre un año de trabajo, desde el diseño hasta la implementación, considerando una persona de medio tiempo.

Adicionalmente, se tendrán los siguientes costos antes de que el producto pueda comenzar a ser promovido.

- Costos de Imprenta, para manuales y guías del sistema.

- Costos de Publicidad, para la promoción del producto

- Costos Fijos, discos magneticos de producción

- Costos Fijos, de índole administrativo

### 6.1.6.6 Factibilidad de Implementación.

Diversos aspectos pueden impactar la implementación y liberación del sistema al público.

El primero de ellos, radica en el hecho de que el sistema esta orientado a satisfacer a un gama de usuarios distintos, lo cual necesariamente implica que la prueba beta del sistema deberá de tomar en cuenta a un usuario tipo de cada uno de ellas.

El segundo de ellos, estriba en la necesidad de creación de un plan de soporte de software para el período inicial de vida del sistema.

El tercero implica la creación de un equipo de trabajo formal para el análisis de necesidades y puesta en marcha de las próximas versiones del sistema.

### 6.1.6.7 Análisis de Contribución Técnica.

Dado que este proyecto, es de índole académico, un aspecto importante del mismo es la contribución que ofrece el producto al medio.

El área de inteligencia artificial, no es un área nueva, de hecho ha estado presente en la mente de muchos científicos desde los años 60, sin embargo recientemente es cuando ha hecho su incursión exitosa en el mercado.

Sin embargo ha pesar de esto, este campo ha sido motivo de un sinnúmero de mitos y discusiones alrededor de él.

Uno de las principales aportaciones planteadas al inicio de este proyecto, fué la de clarificar que es la inteligencia artificial, mostrar su métodos y tratar de orientar al lector acerca de sus alcances presentes y futuros, además de demostrar que puede y de hecho tiene aplicación el procesos de la vida cotidiana, y por lo tanto no se restringe ni se puede restringir a los ambientes puramente académicos o de laboratorio.

## 6.2 Especificaciones Externas.

Comenzaremos por mostrar el diagrama de contexto, el cual se ilustra en la siguiente figura.

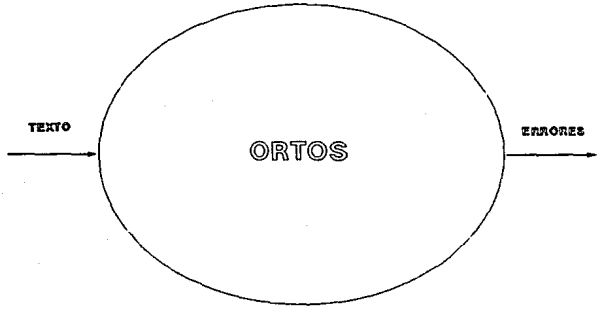



Diagrama de Contexto

Básicamente, la acción del paquete, se va a iniciar a partir de un texto de entrada, el cual va a ser analizado o verificado, teniendo como salida errores si existen, y recomendaciones.

El diagrama de nivel 0, se muestra en la siguiente figura, en el se muestran cada uno de los módulos que componen al producto.

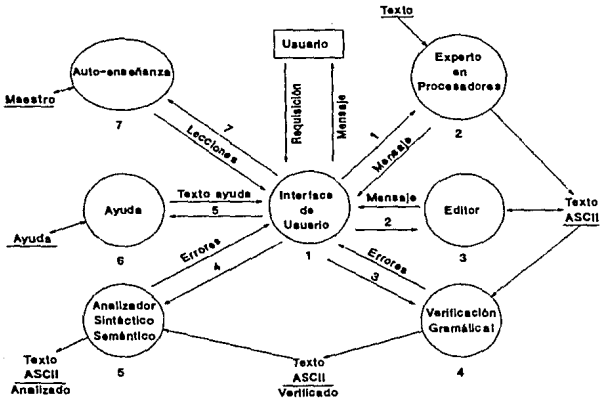



Diagrama 0

Interfase de Usuario.

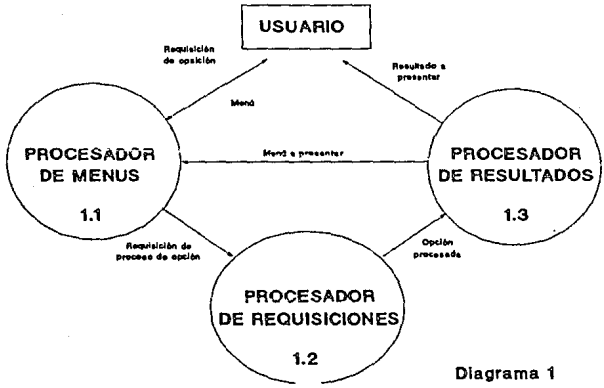



Diagrama 1

Este módulo es el encargado de comunicarse con el usuario desde cualquiera de los módulos del paquete. Su función básica es la de hacer más simple la interacción del sistema con el usuario.

Una de los principales factores de éxito de cualquier producto de orientación hacia usuario final, constituye en contar con una interfase coherente y sencilla de usar, de tal manera que el usuario no tenga que aprender complicados comandos para explotar todas las capacidades del paquete. Otro factor importante, es que en todos los módulos del paquete, las interfases, sean consistentes lo cual disminuira el período de de aprendizaje del usuario.

Experto en Conversión.

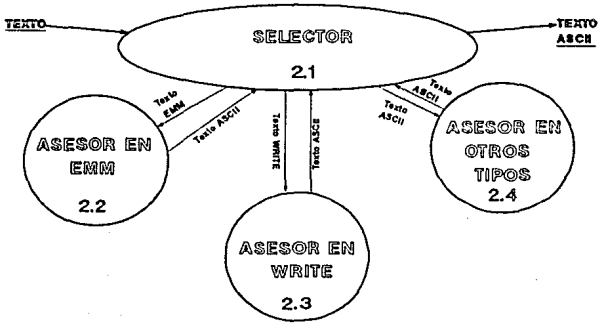



Diagrama 2

Una de las principales armas, para subsistir en un medio ambiente de multiples proveedores, radica en la capacidad de poder importar y exportar datos desde y hacia cualquier producto similar. Por esta razón se decidió incluir la capacidad de poder convertir texto desde y hacia otros paquetes con distintos formatos.

Editor.

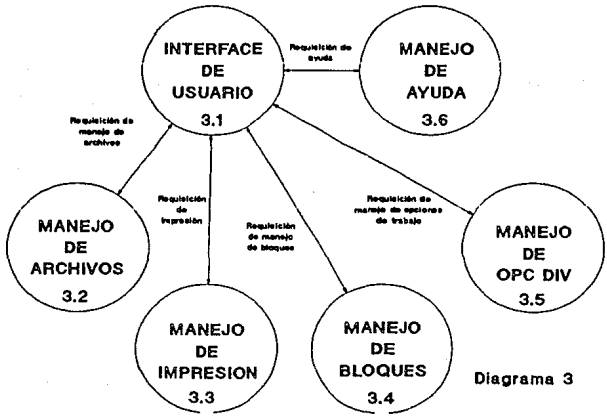
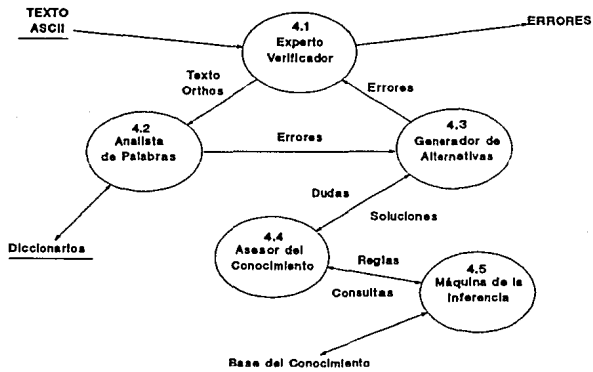



Diagrama 3

Aunque el objetivo principal de el proyecto no es el de proveer de un editor, se decidió incluir uno, el cual le permitiera al usuario manejar las opciones básicas de un editor, tales como remarcado, subrayado, centrado, conversión de y hacia otros paquetes de edición, justificación con espacios o silabeo. El objetivo primordial de este módulo, es el de facilitarle el trabajo a los módulos de verificación y análisis al tratar con textos uniformes y el de enriquecer el producto.

Experto en Verificación.

Diagrama 4

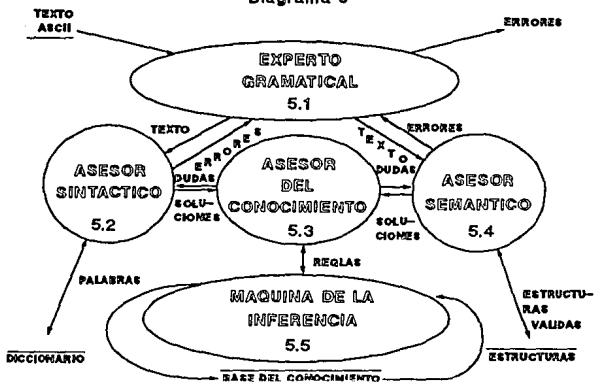



Este módulo es de alta velocidad, y se basa en las técnicas tradicionales de verificación de palabras en base a diccionarios. El objetivo es el proveer al usuario con un módulo de corrección simple de errores basicamente de tipo mecanográfico. Este módulo únicamente se basa en búsquedas de palabras en diccionario, no realiza ningún análisis sintáctico o semántico del documento en cuestión.

Experto Analizador.

Este módulo es el más sofisticado del paquete, su función es la de realizar un análisis sintáctico - semántico en base a una red de transición aumentada modificable por el usuario. Este módulo es en si el que hace uso de todas las herramientas discutidas en las secciones anteriores de este documento.

Diagrama 5




Maestro.

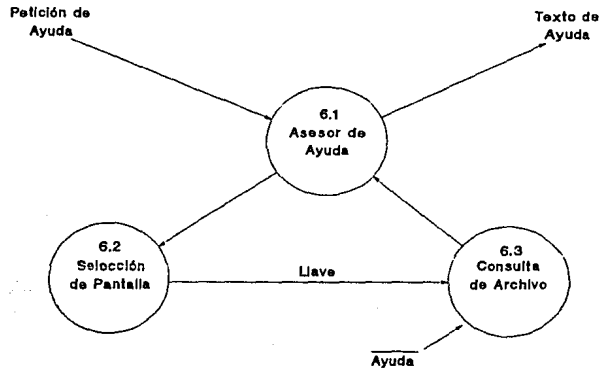



Diagrama 6

Este módulo es basicamente una ayuda para el usuario, esta basado en el libro del maestro Godínez de la Barrera ''Curso Programado de Ortografía''. Su intención, es que en base a reglas sencillas el usuario puede ir mejorando sus defectos ortográficos.

Ayuda.

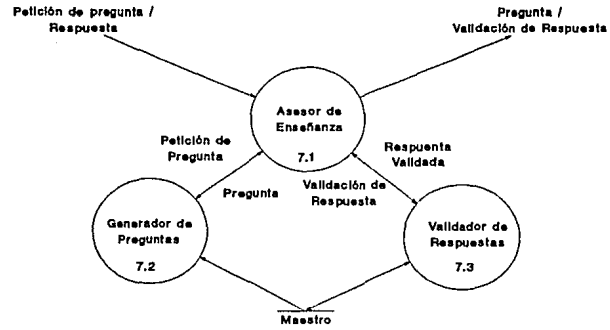



Diagrama 7

Este módulo es el encargado de dar una vista global del paquete al usuario. A través de un sencillo menú, el usuario puede consultar cual es la función especifica de cada uno de las opciones disponibles del sistema.

## 6.3 Conclusiones.

A lo largo de este cápitulo, hemos discutido el análisis y el diseño del paquete, con lo cual este trabajo se puede dar por concluido.

Sin embargo, antes de hacerlo recapitularemos con algunos de los aspecto más importantes de este trabajo.

En nuestro país, y debido fundamentalmente a diversos errores cometidos en el pasado, contamos con un serio atraso tecnológico, el cual en algunas áreas como electrónica, física, etc., va a ser muy difícil de superar.

Sin embargo, particularmente en el área de desarrollo de software, el panorama es completamente distinto en estos momentos. Ya que nos encontramos en un momento en el cual nuestro país se puede situar a la vanguardia en este campo junto con paises industrializados. Será decisión y responsabilidad de los egresados en las carreras de computación y afines, el aceptar y superar este reto que hoy se nos presenta.

Esta situación se debe al hecho de que lo único que se necesita para desarrollar software, es profesionales con talento e inventiva, una máquina, las cuales debido a la reducción de precios en el mercado son cada día más accesibles e inversionistas con visión para sacar provecho de este nuevo mercado.

# Cápitulo VII.

# Conclusiones.

## 7.0 Introducción.

En los cápitulos anteriores, se han analizados diversos puntos relacionados con la inteligencia artificial y su aplicación a un problema cotidiano, como lo es la revisión y análisis de documentos electrónicos.

En este capítulo, resumiremos los puntos más relevantes de este trabajo, obteniendo de ello una visión más clara de los resultados obtenidos con la consecución de este proyecto.

## 7.1 Inteligencia Artificial, Mitos y Realidades.

La inteligencia artificial, realmente es uno de los muchos caminos, en los cuales el futuro de la computación se esta dando en el presente. Curiosamente, cuando alguna de las investigaciones de IA, asume el estado de práctica, o practicable, tiende a no ser llamada nunca más IA, asumiendo con ello una etiqueta propia como ''Programación Orienta a Objetos'' o ''Sistemas Expertos''.

IA, es una etiqueta aplicada a un amplio rango de investigaciones en computación, dividida en al menos 10 subcampos mayores de estudio. Hasta el día de hoy, no existe un consenso entre la comunidad científica, en cuanto a que constituye exactamente IA.

IA es parte de la ciencia de la computación, y es importante remarcar en este punto, que la IA, es una diciplina de investigación científica, por lo cual, tiene un lado teórico y otro práctico, e ignorando los tres usos filosóficos de la ciencia de la computación, las diferencias prácticas del la IA, recaen en la programación. de aquí que surge la pregunta ¿ En que modo difiere la IA, de la programación convencional ?

La respuesta es, en el modo de consevir el problema a solucionar. En IA se tiene que pensar en procesamiento simbólico más que en procesamiento numérico, se tiene que introducir un cierto grado de incertidumbre en cual se encuentra relacionado con las imperfecciones del conocimiento.

El programador, debe remplazar el punto de vista algorítmico y la orientación hacía procedimientos, por un punto de vista contrario, debe de permitir la flexibilidad de los procedimientos y los debe de preparar para adecuarse a cambios en su entorno de operación, ya que la programación en IA esta basada fundamentalmente en la emulación de la actividad mental.

Hoy en día, la IA posee dos caras, la del mito la cual ha sido impulsada por algunos filósofos y artistas, y la de la realidad, dura y práctica. Con la IA, se han comenzado a atacar problemas, los cuales por los métodos convencionales, nunca hubieran sido resueltos. Actualmente las áreas de mayor interes en la investigación de la IA, son:

- Teoría de Juegos
- Programación Automática
- Provado y Deducción de Teorémas
- Representación del Conocimiento
- Aprendizaje
- Procesamiento de Lenguaje Natural
- Resolución de Problemas Abstractos
- Sistemas Expertos
- Robótica

Todos ellos, a excepción de los últimos dos, se encuentran todavía en los laboratorios. Los dos últimos, han pasado de los laboratorios al campo de la vida cotidiana, con bastante éxito.

## 7.2 Análisis Ortográfico a través de un Sistema Experto.

Durante el presente documento, se discutieron diversos aspectos tanto de inteligencia artificial como de sistemas expertos, con el objeto de conjugarlos en un sistema de verificación y análisis ortográfico.

El producto que surgió como resultado de este proyecto, es realmente alentador, en cuanto al potencial que ofrece para futuros desarrollos del mismo producto así como de otros similares.

Sin embargo, lamentablemente aún nos encontramos lejos de un producto que sea capaz de hacer una verificación completa tanto sintáctica como semanticamente de un documento completo, lo cual daría pie a la posibilidad aún más interesante, de la creación de un producto para traducción automática de documentos.

Por otra parte y dadas las condiciones actuales de desarrollo tecnológicas en México, una de las pocas oportunidades que tiene nuestro país de competir con los grandes gigantes tecnológicos, es en el campo de desarrollo de software.

Por lo cual en estos momentos de incertidumbre, es de vital importancia que se impulse la industria de desarrollo de software mexicana.

# BIBLIOGRAFIA


AHO, A. V. & J. D. ULLMAN, ''The Theory of Parsing, Translation and compiling, Vol I:Parsing'', Prentice Hall, 1972

BODEN, M., ''Artificial Intelligence and Natural Man'', Basic Books, N. Y. 1977

BRADY, J. M., ''Artificial Intelligence 17 1-3'', Special Issue on Vision, 1981

CHOMSKY, N., ''Syntactic Structures'', Mouton, The Hague, 1969

COLBY, K., ''Artificial Paranoia'', Pergamon Press, N.Y., 1975

CULLINGFORD, R., ''SAM'' in Inside Computer Understanding, R. C. Schank & C. K. Riesbeck (Eds), Erlbaum, Hillsdale, N. J., 1981

DAVIS, M. & H. Putnam, ''A Computing Procedure for Quantification Theory'', Journal of the ACM, Vol.7, 1960

DOYLE, J., ''A Truth Maintenance System'', Artifical Intelligence, Vol. 12, No. 3, 1979

FEIGENBAUM, E. A. & J. A. Feldman (Eds.), ''Computers and Thought'', McGraw-Hill, N. Y., 1963

FEIGENBAUM, E. A. ''The Art of Artificial Intelligence: Themes and Case in Knowledge Engineering'', in Proc. IJCAI 5, 1977

GODINEZ, G., ''Curso Programado de Ortografía'', 4a. edición trillas 1980

KULIKOWSKY, C. A., ''Artifical Intelligence Methods and Systems for Medical Consultation'', IEEE Transactions on Pattern Analysis and Machine Intelligence, Sept. 1980

LEA, W., ''Trends in Speech Recognition'', Prentice-Hall, 1980

LENAT, D. B., ''Theory Formation by Heuristic Search. The Nature of Heuristics II: Background and Examples'', Artificial Intelligence, Mar. 1983

MARR, D., ''Vision'', Freeman, San Francisco, 1982

McDERMOTH, D. & E. CHARNIAK, ''Introduction to Artificial Intelligence'', Addison-Wesley, 1986

MINSKY, M., ''Computation: Finite and Infinite Machines'', Prentice-Hall, N. J., 1968

MINSKY, M., ''Theory of Frames'', 1974

MINSKY, M., '' A Framework for Representing Knowledge'', in The Psychology of Computer Vision, P Winston (Eds.), McGraw-Hill, N. Y., 1975

NILSSON, N. J., ''Principles of Artificial Intelligence'', Tioga, Palo Alto, Calif., 1980

NOVAK, G. S. & A. Araya, ''Research on Expert Problem Solving in Physics'', in Proc. AAAI, 1980

POLYA, G., ''How to Solve It'', Princeton University Press,

Princeton, N. J., 1957
QUILLIAN, R., ''Semantic Memory'', in Semantic Information Processing, M Minsky (Ed.), MIIT Press, Cambridge, Mass., 1968
RAPHAEL, B., ''A computer Program for Semantic Information Retrieval'', in Semantic Information Processing, M Minsky (Ed.), MIT Press, Cambridge, Mass., 1968
REDDY, D. R., ''Speech Recognition by Machine: A Review'', Proc. of the IEEE, Vol. 64, Mayo 1976
RICH, E. A., ''User Modeling via Stereotypes'', Cognitive Science, Vol. 3, pag 329-354, 1979
RICH, E. A., ''Artificial Intelligence'', McGraw-Hill, 1983
SCHANK, R. C. & B. NASH-WEBBER (Eds.), ''Theoretical Issues in Natural Language Processing: An Interdiciplinary Workshop in Computational Linguistic, Psychology, Linguistics, and Artificial Intelligence'', 1975
SCHANK, R. C. & R. P. Abelson, ''Scripsts, Plans, Goals, and Understanding'', Erlbaum, Hillsdale, N. J., 1977
SCHANK, R. C., ''Language and Memory'', Cognitive Science, Vol. 4, 1980
SHORTLIFFE, E. H., B. G. BUCHANAN, & E. A. FEIGENBAUM, ''Knowledge Engineering for Medical Decision Making: A Review of Computer-Based Clinical Decision Aids'', Proceedings of the IEEE, Vol 67, pag 1207-1224, 1979
SORENSON, P. G. & J. P. TREMBLAY, ''The Theory and Practice of Compiler Writing'', McGraw-Hill, 1985
SRIDHARAM, N. S., ''Artificial Intelligence II'', Special Issue on Applications to the Sciences and Medicine'', 1978
STEFIK, M., J. Aikins, R. Balzer, J. Benoit, L. Bimbaum, F. Hayes-Roth, & E. D. Sacerdoti, ''The Organization of Expert Systems'', Artificial Intelligence, Vol. 18, pag 135-173, Marzo 1982
WALKER, D. E., ''Speech Understanding Research'', North-Holand, Amsterdam, 1976
WEIZENBAUM, J., ''ELIZA-A Computer Program for the Study of Natural Languaje Communication between Man and Machine'', Communications of the ACM, Vol 9, No. 1, pag 36-44, Enero 1966
WILKS, Y. A., ''Grammar, Meaning and the Machine Analysis of language'', Routldge & Kegan Paul, London, 1972
WINOGRAD, T., ''Frame Representation and the Declarative-Procedural Controversy'', in Representation and Understanding, D. G. Bobrow & A. Collins (Eds), Academic Press, N. Y. 1975
WINOGRAD, T., ''Language as a Cognitive Process: Syntax'', Addison-Wesley, Reading, Mass, 1983
WINSTON, P. H., ''The Psychology of Computer Vision'', McGraw-Hill, N. Y., 1975

WOODS, W. A., ''Transition Network Grammars for Natural Lan
guage Analysis'', Communications of the ACM, Vol 13, pag 591-
606, 1970

# PROGRAMAS

```
/*****************************************************************

     Turbo Prolog Corrector Ortografico por medio
     de la utilización de técnicas de Inteligencia
                    Artificial
        (C) Copyright 1988 Dinámica Organizacional.

*****************************************************************/
code=4000

/*****************************************************************
      En el siguiente archivo, se incluyen todos
      aquellas declaraciones globales que se usan
      en la parte de interfase de usuario.
*****************************************************************/
include ''tdoms.pro''

DOMAINS
  SELECTOR=INTEGER
  FILE=texto ; texto1 ; archivo

DATABASE
  pdwstate(ROW,COL,SYMBOL,ROW,COL)
  treewindow(ROW,COL)
  treechoice(SELECTOR)
  palabras(string)

global predicates
/*****************************************************************
      Se necesitarón de tres modulos escritos en
      lenguaje c, estos se explican a continuación:

      cpinit: Módulo de interface entre c y prolog
      limpia: Rútina de para limpiar la pantalla
      compara: Rútina de comparación de cadenas
      conviert.bat: Invoca al experto en conversión
*****************************************************************/
   cpinit language c
   limpia language c
   compara (string, string, integer) - (i,i,o) language c

/*****************************************************************
      El sistema es modular, de tal modo que los
      siguientes archivos, contemplan cada uno una
      parte especifica del sistema.
```

```
      tpreds, status y pulldown: Interface de usuario, fueron
                                 tomados directamente de
                                 Turbo Prolog Toolbox
      correcto: Módulo de verificación ortográfica
      analiza:  Módulo de análisis sintáctico-semántico
      xorthos:  Módulo de ayuda
*************************************************************/
include ''tpreds.pro''
include ''status.pro''
include ''pulldown.pro''
include ''xorthos.pro''
include ''correcto.pro''
include ''analiza.pro''
Predicates
  msg(ROW,COL,STRING)
  existe(STRING)
  nuevo(STRING)
CLAUSES
/* Las siguientes clausulas se utilizan en la parte del editor*/
/* ya se para crear un nuevo archivo o para editar uno ya existente
*/

  existe(Archivo):-existfile(Archivo),
                   write(''Este archivo ya existe, ¿ Deseas bor
                         rarlo (S/N) ? ''),
                   readln(Respuesta),
                   upper_lower(Resp, Respuesta),
                   Resp=''N'',
                   makestatus(112,'' Selecciona con las flechas o
                         usa la letra en mayuscula''),
                   removewindow.
  existe(Archivo):-nuevo(Archivo).
  nuevo(Archivo):-clearwindow,
                  makestatus(112,''F1=Ayuda, F10=Salir y Salvar,
                         Esc=Abortar edición.''),
                  Editmode=1, Indent=1,
                  Insert=1, Textmode=1,
                  edit('''', Out, ''Editor'', '''',''¡ Listo
                         !'',0,''prolog.hlp'',
                       Editmode, Indent, Insert,
                         Textmode,_,Retstatus),
                  Retstatus=0,
                  file_str(Archivo, Out),
                  removewindow.
  nuevo(_):-      removewindow.
  pdwaction(1,1):-makewindow(30,7,4,''Orthos Editor'',2,0,22,80),
```

```
                write(''¿ Con que nombre deseas salvar el docu
                  mento ? ''),
                readln(Archivo),
                existe(Archivo).
pdwaction(1,2):-makewindow(30,7,4,''Orthos Editor'',2,0,22,80),
                dir('''', ''*.*'', Archivo),
                removewindow,
                makestatus(112,''F1=Ayuda, F10=Salir y Salvar
                  Esc=Abortar edición.''),
                makewindow(30,7,4,''Orthos Editor'',2,0,22,80),
                Editmode=1, Indent=1,
                Insert=1, Textmode=1,
                file_str(Archivo, Str),
                edit(Str, Out, ''Editor'', '''',''¡ Listo
                  !'',0,''prolog.hlp'',
                     Editmode, Indent, Insert,
Textmode,_,Retstatus),
                makestatus(112,'' Selecciona con las flechas o
                  usa la letra en mayuscula''),
                Retstatus=0,
                file_str(Archivo, Out),
                removewindow.
pdwaction(1,2):-removewindow.
pdwaction(1,3):-system(''CONVIERT.BAT'').
pdwaction(1,4):-msg(3,15,''imprimiendo'').
pdwaction(2,1):-msg(3,22,''alterando'').
pdwaction(2,2):-corrector,
                makestatus(112,'' Selecciona con las flechas o
                  usa la letra en mayuscula'').
pdwaction(3,1):-msg(3,35,''alterando'').
pdwaction(3,2):-analizador,
                makestatus(112,'' Selecciona con las flechas o
                  usa la letra en mayuscula'').
pdwaction(3,3):-msg(5,35,''explicasion'').
pdwaction(3,4):-msg(6,35,''Red de Transicion'').
pdwaction(4,1):-msg(3,49,''capitulo uno'').
pdwaction(4,2):-msg(4,49,''capitulo dos'').
pdwaction(5,0):-orthos.
pdwaction(6,0):-msg(3,10,''Oprime la barra espaciadora''),
                fail.
msg(R,C,S):-
      makestatus(112,''Oprime retorno para continuar''),
   makewindow(1,32,4,'''',R,C,5,30),
   window_str(S),
   readkey(_),
   removewindow,
```

```
.     removestatus.

GOAL
  limpia,
  makewindow(1,7,4,'''',6,25,12,31),
  write(''\n\n          O R T H O S''),
  write(''\n\n\n              Por''),
  write(''\n\n   DINAMICA ORGANIZACIONAL''),
  write(''\n           M C Gloria''),
  write(''\nOprime retorno para comenzar''),
  readchar(_),
  removewindow,
  makewindow(1,7,0,''O R T H O S'',0,0,24,80),
  makestatus(112,'' Selecciona con las flechas o usa la letra en
              mayuscula''),
  pulldown(4,
           [curtain(6, ''Editor'',
[''Nuevo'',''Editar'',''Convertir'',''Imprimir'']),
            curtain(17,''Corrector'', [''Alterando'',''Sin al-
terar'']),
            curtain(30,''Analizador'',[''Alterando'',''Sin al-
terar'',
                                       ''Justificar'',''Rta'']),
            curtain(44,''Maestro'',   [''cap Uno'',''cap
Dos'',''cap Tres'',
                                       ''cap Cuatro'',''cap
cInco'']),
            curtain(56,''Orthos'' ,   []),
            curtain(69,''Salir'',     [])
          ]
                  ,CH,SUBCH ),
  removewindow,
  limpia.
```

```
/*********************************************************

      Módulo experto verificador.

*********************************************************/

domains
   tipoarbol = arbol( string, tipoarbol, tipoarbol ); fin
   db_selector = base_de_datos

predicates
   corrector /* Rútina principal del módulo */
   descompone( string, tipoarbol ) /* Descompone lineas en palabras
*/
   insertar( string, tipoarbol, tipoarbol ) /* Inserta nodos en
árbol binario */
   escribir_arbol( tipoarbol ) /* Recorre el árbol en Inorder */
   carga_archivo( string ) /* Menú de selección de archivo a veri-
ficar */
   lectura( tipoarbol ) /* Lectura de archivo a verificar */
   analiza( bt_selector ) /* Análisis de palabras erroneas */
   busca( string, bt_selector ) /* Selección de palabras alternas
*/
   decide( integer, string ) /* Reporta palabras alternas */

clauses

   corrector:-
      makewindow(30,7,4,''Directorio'',3,0,21,80),
      makestatus(112,''Posicionese sobre el archivo a ser verifi-
cado y oprima <Retorno>''),
      carga_archivo( Archivo ),
      makewindow(31,7,4,''Orthos Analizador'',3,0,21,80),
      makestatus(112,''Verificación en progreso''),
      db_open(base_de_datos, ''dict.ort'', in_file),
      bt_open(base_de_datos, ''indx.ort'', Index),
      openread( archivo, Archivo ),
      readdevice( archivo ),
      lectura( fin ),
      closefile( archivo ),
      analiza( Index ),
      bt_close(base_de_datos, Index),
      db_close(base_de_datos),
      readdevice( keyboard ),
      beep,
      beep,
```

```
      makestatus(112,''** Verificación concluida, oprima cualquier
tecla para continuar **''),
      readchar(_),
      removewindow.

   lectura( Arbol ):- descompone( '''', Arbol ).

   descompone ( Linea, Arbol ) :-  Linea <> '''',
            frontchar( Linea, Car, Linea1),
            Car = ' ',!,
            descompone( Linea1, Arbol ).

   descompone ( Linea, Arbol ) :-  Linea <> '''',!,
                                   fronttoken ( Linea, Palabra,
Linea1 ),
                                   upper_lower( Palabra, Palabras),
                                   insertar( Palabras, Arbol,
Nuevoarbol ),
                                   descompone( Linea1, Nuevoarbol
).

   descompone ( _,Arbol ):- not ( eof ( archivo ) ),!,
                            readln( Linea ),
                            descompone ( Linea, Arbol ).

   descompone ( _,Arbol ):- !,escribir_arbol( Arbol ).

   insertar( Nuevo, fin, arbol(Nuevo, fin, fin) ):-!.

   insertar( Nuevo, arbol(Elemento, Izq, Der), arbol(Elemento,
Nvoizq, Der) ):-
      Nuevo < Elemento, !,
      insertar( Nuevo, Izq, Nvoizq ).

   insertar( Nuevo, arbol(Elemento, Izq, Der), arbol(Elemento, Izq,
Der) ):-
      Nuevo = Elemento, !.

   insertar( Nuevo, arbol(Elemento, Izq, Der), arbol(Elemento, Izq,
Nvoder) ):-
      insertar( Nuevo, Der, Nvoder ).

   escribir_arbol( fin ).

   escribir_arbol( arbol(Elemento, Izq, Der) ):-
      escribir_arbol( Izq ),
```

```
    Elemento >= ''a'',
    assertz(palabras( Elemento )),
    escribir_arbol( Der ).

escribir_arbol( arbol(_, _, Der) ) :-
    escribir_arbol( Der ).

carga_archivo( Archivo ):- dir('''', ''*.asc'', Archivo),
                           removewindow.

analiza( Index ):-
    retract(palabras(Elemento)),!,
    busca( Elemento, Index ),!,
    analiza( Index ).

analiza( _ ).

busca( Elemento, Index ):-
    key_search(base_de_datos, Index, Elemento, _).

busca( Elemento,Index ):-
    write(''\n'', Elemento),
    cursor(R, _),
    cursor(R, 35),
    key_current(base_de_datos, Index, Pos1, _),
    str_len( Elemento, I1),
    str_len( Pos1, I2),
    I3=I1-I2,
    I4=abs(I3),
    compara( Elemento, Pos1, I),
    decide(I4, Pos1),
    key_prev(base_de_datos, Index, _),
    key_current(base_de_datos, Index, Pos2, _),
    cursor(R, 56),
    str_len( Pos2, I5),
    I6=I1-I5,
    I7=abs(I6),
    compara( Elemento, Pos2, J),
    decide(I7, Pos2).

 decide( Numero, Palabra ):-
    Numero < 3,!,
    write(Palabra).

 decide( _,_ ).
```

```
/************************************************************

     Módulo de análisis sintáctico-semántico en base
     al metódo de red de transición aumentada RTA,
     ATN en ingles.

************************************************************/

domains
   list = string*
   form_list = string*
   class = pronombre(symbol) ; verbo(symbol) ;
           adj(symbol) ; det(symbol) ;
           aux(symbol) ; pronom_pro(symbol) ;
           frase_no(form_list) ; frase_prep(form_list) .

predicates
   analizador
   separa( string, tipoarbol )
   carga_archivos( string )
   lee_arc( tipoarbol )
   encuentra( string, bt_selector )
   append(form_list,form_list,form_list)
   diccionario(string, bt_selector)
```

```
   arco(symbol,symbol,string)
   terminal(symbol,list)
   trans(symbol,symbol,form_list,list,list)
   ptrnas(symbol,symbol,symbol,list,string,form_list)

clauses

   analizador:-
      makewindow(30,7,4,''Directorio'',3,0,21,80),
      makestatus(112,''Posicionese sobre el archivo a ser analizado
y oprima <Retorno>''),
      carga_archivos( Archivo ),
      makewindow(31,7,4,''Orthos Analizador'',3,0,21,80),
      makestatus(112,''Análisis en progreso''),
      db_open(base_de_datos, ''dict.ort'', in_file),
      bt_open(base_de_datos, ''indx.ort'', Index),
      openread( archivo, Archivo ),
      readdevice( archivo ),
      lee_arc( fin ),
      closefile( archivo ),
      analiza( Index ),
      bt_close(base_de_datos, Index),
      db_close(base_de_datos),
      readdevice( keyboard ),
      beep,
      beep,
      makestatus(112,''** Análisis concluido, oprima cualquier
tecla para continuar **''),
      readchar(_),
      removewindow.

   lee_arc( Arbol ):- separa( '''', _ ).

   separa ( Linea, [H|T] ) :-  Linea <> '''',
            frontchar( Linea, H, Linea1),
            H = ' ',!,
            separa( Linea1, T ).

   separa ( Linea, [H|T] ) :-  Linea <> '''',!,
                                    fronttoken ( Linea, Palabra,
Linea1 ),
                                    upper_lower( Palabra, H),
                                    separa( Linea1, T ).

   separa ( _,_ ):- not ( eof ( archivo ) ),!,
                    trans(q0,Nq,Analisis,S,S1),
```

```
                        readln( Linea ),
                        separa ( Linea, _ ).

   separa ( _,_ ).

   carga_archivos( Archivo ):- dir('''', ''*.asc'', Archivo),
                               removewindow.

   diccionario( Elemento, Index ):-
      key_search(base_de_datos, Index, Elemento, _).

   diccionario( Elemento,Index ):-
      write( Elemento ),
      makestatus(112,''Palabra desconocida use primero el verifica-
dor'').

    arco(q0,q1, frase_no(_)).
    arco(q0,q2, aux(_)).
    arco(q1,q4, verbo(_)).
    arco(q1,q3, aux(_)).
    arco(q2,q3, frase_no(_)).
    arco(q3,q4, verbo(_)).
    arco(q4,q5, frase_no(_)).
    arco(qnp,qnp1, det(_)).
    arco(qnp,qnp2, pronom_pro(_)).
    arco(qnp1,qnp1, adj(_)).
    arco(qnp1,qnp3, sujeto(_)).

    terminal(q4,[]).
    terminal(q5,[]).

    trans(Lq,Nq,Analisis,S1,S0):-
       diccionario(Class,S1),
       arco(Lq,Nq,Class),
       append( Analisis[Class], P1,_),
       !,
       trans(Nq,_,P1,S0,_).

    trans(Lq,_,Analisis,S1,_):-
       terminal(Lq,S1),
       bound(Analisis),
       nl,nl,
       makestatus(112,''Análisis concluido''),
       beep,beep,
       write(Análisis).
```

```
    trans(Lq,_,Analisis,S0,_):-
       arco(Lq,Nq,frase_no(_)),
       ptrans(qnp,Nq,Lq,S0,frase_no(_),Analisis).

    trans(Lq,_,Analisis,S0,_):-
       bound(Analisis),
       nl,nl,
       makestatus(112,''Estructura no conocida''),
       beep,beep.

    ptrans(Bq,Nq,Lq,S0,frase_no(X),Analisis):-
       diccionario(Class,S0,S1),
       arco(Bq,Zq,Class),
       append(X,[Class],P1),
       !,
       ptrans(Zq,Nq,Lq,S1,frase_no(P1),Analisis).

    ptrans(_,Nq,Lq,S0,frase_no(Pr),Analisis):-
       bound(Pr),
       append(Analisis[frase_no(Pr)],P1,_),
       !,
       trans(Nq,_,P1,S0,_).



/*****************************************************************

     Módulo de Ayuda del sistema ORTHOS
     (C) Copyright 1988 Dinámica Organizacional.

*****************************************************************/



DOMAINS
  TREE = tree(STRING,SELECTOR,TREELIST)
  TREELIST = TREE*

include ''menubig.pro''
include ''tree.pro''

PREDICATES

lee(FILE)
seleccion(SELECTOR)
```

```
CLAUSES
  seleccion(1):-
        makewindow(5,66,24,''Orthos'',5,0,20,80),
        openread( texto1, ''TEXTO1.'' ),
        readdevice( texto1 ),
        lee(texto1),
        readdevice(keyboard),
        write(''\n Oprime return para continuar ''),
        readchar(_),
        removewindow,
        closefile(texto1).

  seleccion(2):-
        makewindow(5,66,24,''Orthos Editor'',5,0,20,80),
        openread( texto1, ''TEXTO2.'' ),
        readdevice( texto1 ),
        lee(texto1),
        readdevice(keyboard),
        write(''\n Oprime return para continuar ''),
        readchar(_),
        removewindow,
        closefile(texto1).

  seleccion(3):-
        makewindow(5,66,24,''Orthos Editor Nuevo'',5,0,20,80),
        openread( texto1, ''TEXTO3.'' ),
        readdevice( texto1 ),
        lee(texto1),
        readdevice(keyboard),
        write(''\n Oprime return para continuar ''),
        readchar(_),
        removewindow,
        closefile(texto1).

  seleccion(4):-
        makewindow(5,66,24,''Orthos Editor Editar'',5,0,20,80),
        openread( texto1, ''TEXTO4.'' ),
        readdevice( texto1 ),
        lee(texto1),
        readdevice(keyboard),
        write(''\n Oprime return para continuar ''),
        readchar(_),
        removewindow,
        closefile(texto1).
```

```
seleccion(5):-
      makewindow(5,66,24,''Orthos Editor Convertir'',5,0,20,80),
      openread( texto1, ''TEXTO5.'' ),
      readdevice( texto1 ),
      lee(texto1),
      readdevice(keyboard),
      write(''\n Oprime return para continuar ''),
      readchar(_),
      removewindow,
      closefile(texto1).

seleccion(6):-
      makewindow(5,66,24,''Orthos Editor Imprimir'',5,0,20,80),
      openread( texto1, ''TEXTO6.'' ),
      readdevice( texto1 ),
      lee(texto1),
      readdevice(keyboard),
      write(''\n Oprime return para continuar ''),
      readchar(_),
      removewindow,
      closefile(texto1).

seleccion(7):-
      makewindow(5,66,24,''Orthos Corrector'',5,0,20,80),
      openread( texto1, ''TEXTO7.'' ),
      readdevice( texto1 ),
      lee(texto1),
      readdevice(keyboard),
      write(''\n Oprime return para continuar ''),
      readchar(_),
      removewindow,
      closefile(texto1).

seleccion(8):-
      makewindow(5,66,24,''Orthos Corrector Modifi-
cando'',5,0,20,80),
      openread( texto1, ''TEXTO8.'' ),
      readdevice( texto1 ),
      lee(texto1),
      readdevice(keyboard),
      write(''\n Oprime return para continuar ''),
      readchar(_),
      removewindow,
      closefile(texto1).

seleccion(9):-
```

```
        makewindow(5,66,24,''Orthos Corrector Sin Al-
terar'',5,0,20,80),
        openread( texto1, ''TEXTO9.'' ),
        readdevice( texto1 ),
        lee(texto1),
        readdevice(keyboard),
        write(''\n Oprime return para continuar ''),
        readchar(_),
        removewindow,
        closefile(texto1).

  seleccion(10):-
        makewindow(5,66,24,''Orthos Analizador'',5,0,20,80),
        openread( texto1, ''TEXTO10.'' ),
        readdevice( texto1 ),
        lee(texto1),
        readdevice(keyboard),
        write(''\n Oprime return para continuar ''),
        readchar(_),
        removewindow,
        closefile(texto1).

  seleccion(11):-
        makewindow(5,66,24,''Orthos Analizador Modifi-
cando'',5,0,20,80),
        openread( texto1, ''TEXTO11.'' ),
        readdevice( texto1 ),
        lee(texto1),
        readdevice(keyboard),
        write(''\n Oprime return para continuar ''),
        readchar(_),
        removewindow,
        closefile(texto1).

  seleccion(12):-
        makewindow(5,66,24,''Orthos Analizador Sin Modifi-
car'',5,0,20,80),
        openread( texto1, ''TEXTO12.'' ),
        readdevice( texto1 ),
        lee(texto1),
        readdevice(keyboard),
        write(''\n Oprime return para continuar ''),
        readchar(_),
        removewindow,
        closefile(texto1).
```

```
  seleccion(13):-
        makewindow(5,66,24,''Orthos Analizador Justifi-
car'',5,0,20,80),
        openread( texto1, ''TEXTO13.'' ),
        readdevice( texto1 ),
        lee(texto1),
        readdevice(keyboard),
        write(''\n Oprime return para continuar ''),
        readchar(_),
        removewindow,
        closefile(texto1).

  seleccion(14):-
        makewindow(5,66,24,''Orthos Analizador RTA'',5,0,20,80),
        openread( texto1, ''TEXTO14.'' ),
        readdevice( texto1 ),
        lee(texto1),
        readdevice(keyboard),
        write(''\n Oprime return para continuar ''),
        readchar(_),
        removewindow,
        closefile(texto1).

  seleccion(15):-
        makewindow(5,66,24,''Orthos Maestro'',5,0,20,80),
        openread( texto1, ''TEXTO15.'' ),
        readdevice( texto1 ),
        lee(texto1),
        readdevice(keyboard),
        write(''\n Oprime return para continuar ''),
        readchar(_),
        removewindow,
        closefile(texto1).

  seleccion(16):-
        makewindow(5,66,24,''Orthos Maestro Cap Uno'',5,0,20,80),
        openread( texto1, ''TEXTO16.'' ),
        readdevice( texto1 ),
        lee(texto1),
        readdevice(keyboard),
        write(''\n Oprime return para continuar ''),
        readchar(_),
        removewindow,
        closefile(texto1).

  seleccion(17):-
```

```
        makewindow(5,66,24,''Orthos Maestro Cap Dos'',5,0,20,80),
        openread( texto1, ''TEXTO17.'' ),
        readdevice( texto1 ),
        lee(texto1),
        readdevice(keyboard),
        write(''\n Oprime return para continuar ''),
        readchar(_),
        removewindow,
        closefile(texto1).

  seleccion(18):-
        makewindow(5,66,24,''Orthos Maestro Cap Tres'',5,0,20,80),
        openread( texto1, ''TEXTO18.'' ),
        readdevice( texto1 ),
        lee(texto1),
        readdevice(keyboard),
        write(''\n Oprime return para continuar ''),
        readchar(_),
        removewindow,
        closefile(texto1).

  seleccion(19):-
        makewindow(5,66,24,''Orthos Maestro Cap Cu-
atro'',5,0,20,80),
        openread( texto1, ''TEXTO19.'' ),
        readdevice( texto1 ),
        lee(texto1),
        readdevice(keyboard),
        write(''\n Oprime return para continuar ''),
        readchar(_),
        removewindow,
        closefile(texto1).

  seleccion(20):-
        makewindow(5,66,24,''Orthos Maestro Cap Cinco'',5,0,20,80),
        openread( texto1, ''TEXTO20.'' ),
        readdevice( texto1 ),
        lee(texto1),
        readdevice(keyboard),
        write(''\n Oprime return para continuar ''),
        readchar(_),
        removewindow,
        closefile(texto1).

  seleccion(21):-
        makewindow(5,66,24,''Orthos Orthos'',5,0,20,80),
```

```
        openread( texto1, ''TEXTO21.'' ),
        readdevice( texto1 ),
        lee(texto1),
        readdevice(keyboard),
        write(''\n Oprime return para continuar ''),
        readchar(_),
        removewindow,
        closefile(texto1).

  seleccion(_):-
        makewindow(5,66,24,''Orthos Salir'',5,0,20,80),
        openread( texto1, ''TEXTO22.'' ),
        readdevice( texto1 ),
        lee(texto1),
        readdevice(keyboard),
        write(''\n Oprime return para continuar ''),
        readchar(_),
        removewindow,
        closefile(texto1).

  treeaction(Opc):-
     cursor(R,C),
     makewindow(_,_,_,_,SR,SC,_,_),
     RR=R+SR, CC=C+SC,
     menu(RR,CC,32,33,[continuar,seleccionar],opción,0,CH),
     CH=2,
     seleccion(Opc).

lee(texto1):-
        not(eof(texto1)),
        readln(Linea),
        write(Linea,''\n''),
        lee(texto1).

lee(_):-!.
PREDICATES
  orthos
  show(SYMBOL)

CLAUSES
  show(DIRECTION):-
      treemenu(DIRECTION,tree(''ORTHOS'',1,
           [tree(''EDITOR'',2,
                [tree(''NUEVO'',3,[]),
                 tree(''EDITAR'',4,[]),
                 tree(''CONVERTIR'',5,[]),
```

```
                tree(''IMPRIMIR'',6,[])]),
            tree(''CORRECTOR'',7,
                [tree(''ALTERANDO'',8,[]),
                 tree(''SIN ALTERAR'',9,[])]),
            tree(''ANALIZADOR'',10,
                [tree(''ALTERANDO'',11,[]),
                 tree(''SIN ALTERAR'',12,[]),
                 tree(''JUSTIFICAR'',13,[]),
                 tree(''RTA'',14,[])]),
            tree(''MAESTRO'',15,
                [tree(''CAPITULO 1'',16,[]),
                 tree(''CAPITULO 2'',17,[]),
                 tree(''CAPITULO 3'',18,[]),
                 tree(''CAPITULO 4'',19,[]),
                 tree(''CAPITULO 5'',20,[])]),
            tree(''ORTHOS'',21,[]),
            tree(''SALIR'',22,[])]),
    CHOICE).

 orthos:- makewindow(5,66,23,''Orthos'',5,5,20,70),
     write(''Te invitamos a realizar una selección de este
menú.\n\n''),
     write(''Este  menú  abarca  todas las  opciones  del  sistema
de\n''),
     write(''correción automática ORTHOS.\n\n''),
     write(''Cada vez que  tu selecciones  una opción, oprimiendo
para\n''),
     write(''ello la tecla RETURN, se abrira una ventana donde se
te\n''),
     write(''explicarán las funciones que realiza esa
opción.\n\n''),
     write(''Tu te puedes mover a tráves del árbol usando las
flechas.\n\n''),
     write(''.. Oprime RETURN para comenzar  ......''),readkey(_),
     removewindow,
     show(right).
```