UNIVERSIDADE DA CORUÑA

FACULTADE DE INFORMÁTICA

DEPARTAMENTO DE COMPUTACIÓN

TESIS DE LICENCIATURA

SUPRESIÓN DE AMBIGÜEDADES LÉXICAS EN GALENA MEDIANTE MÉTODOS ESTADÍSTICOS

Memoria dirigida por
D. Alberto Valderruten Vidal
y presentada por
D. Javier Andrade Garda

para optar al
Grado de Licenciado en Informática

Octubre de 1997

BASTA YA

A mis padres, hermano y amigos,
que siempre me apoyan y animan.

Quid opus est verbis ?

TERRENCE

UNIVERSIDADE DA CORUÑA

DEPARTAMENTO DE
COMPUTACIÓN

Facultade de Informática.
Campus de Elviña, s/n.
15071. A Coruña.
España - Spain.
Telf.: +34 (9)81 16 70 00
Fax: +34 (9)81 16 71 60

D. JOSÉ MARÍA BARJA PÉREZ, catedrático y director del Departamento de Computación de la Universidade da Coruña,

AUTORIZA a **D. JAVIER ANDRADE GARDA** a presentar el trabajo titulado **"SUPRESIÓN DE AMBIGÜEDADES LÉXICAS EN GALENA MEDIANTE MÉTODOS ESTADÍSTICOS"**, para optar al Grado de Licenciado en Informática.

Y para que así conste a los efectos oportunos, firma la presente en A Coruña a diez de septiembre de mil novecientos noventa y siete.

Fdo.: Prof. Dr. D. José Mª. Barja Pérez.

UNIVERSIDADE DA CORUÑA

DEPARTAMENTO DE
COMPUTACIÓN

Facultade de Informática.
Campus de Elviña, s/n.
15071. A Coruña.
España - Spain.
Telf.: +34 (9)81 16 70 00
Fax: +34 (9)81 16 71 60

D. ALBERTO VALDERRUTEN VIDAL, profesor doctor del Departamento de Computación de la Universidade da Coruña,

CERTIFICA que el presente trabajo, titulado **"SUPRESIÓN DE AMBIGÜEDADES LÉXICAS EN GALENA MEDIANTE MÉTODOS ESTADÍSTICOS"**, se ha realizado bajo su dirección en el Departamento de Computación de la Universidade da Coruña.

Y para que así conste a los efectos oportunos, firma la presente en A Coruña a diez de septiembre de mil novecientos noventa y siete.

Fdo.: Prof. Dr. D. Alberto Valderruten Vidal.

AGRADECIMIENTOS

Un Proyecto de Licenciatura que no tiene carácter obligatorio pasa por algunos momentos en los que se tienen tentaciones de abandonarlo. Afortunadamente, en esos momentos, siempre hay personas que te ayudan, e incluso muchas veces te fuerzan, a seguir. Muchas no verán su nombre aquí, pero es simplemente para no hacer este apartado más largo que el propio proyecto. Estas líneas están dedicadas a todas y cada una de ellas, sin las cuales este proyecto difícilmente habría alcanzado su fin.

Por todo ello este apartado es tan importante, o más, que cualquiera de los capítulos que siguen. A todos, mi más sincero agradecimiento.

En primer lugar, agradecer a Manuel Vilares Ferro su confianza en mí al proponerme este proyecto primero, y por permitirme realizarlo con su ayuda después.

A Alberto Valderruten Vidal por ser, más que un magnífico director de proyecto, un inestimable amigo, y por saber guiarme de la mejor manera que cabría esperar.

Al personal del Departamento de Filología Española de la Universidad de Santiago de Compostela integrado en el proyecto donde se encuadra este trabajo, por su colaboración, y especialmente a Susana y Conchi por pasarse alguna que otra noche en vela echándome una mano, y a veces las dos.

A todos los, más que mandos, compañeros de la Comandancia Militar de Obras de la R.M. Noroeste, por permitirme invertir mucho tiempo de la mili en este trabajo y por ayudarme en todo lo posible.

Por último deseo expresar mi gratitud a todos los amigos, compañeros y profesores que de alguna forma me han ayudado, guiado y animado, especialmente a Santi, por ayudarme siempre que lo he necesitado, y a Jorge, Miguel, David, Teresa, Luis y Óscar, por sufrir mis dudas y enseñarme muchas cosas con mucha paciencia.

A todos ellos y por todo:

Muchas gracias.

ÍNDICE

1. INTRODUCCIÓN

Uno de los objetivos más perseguidos por la inteligencia artificial es la creación de sistemas capaces de entender el lenguaje humano. No es sólo que la habilidad para entender el lenguaje natural parezca ser uno de los aspectos fundamentales de la inteligencia humana, sino que también sería un éxito en la automática que tendría un increíble impacto en la utilidad y efectividad de los ordenadores.

El interés en el tratamiento de los lenguajes naturales ha experimentado un fuerte incremento desde que fue declarado tema prioritario de investigación en las últimas convocatorias del proyecto comunitario ESPRIT[1]. En efecto, en un mundo en el que la informática tiende a acercarse al usuario inexperto, no se puede ignorar el desarrollo de proyectos de investigación viables y realistas sobre el tema.

Se han invertido muchos esfuerzos para escribir programas que entendiesen el lenguaje natural. Sin embargo, estos programas sólo han obtenido éxito dentro de contextos restringidos. Los sistemas capaces de usar el lenguaje natural con la flexibilidad y la generalidad que caracterizan el discurso humano están más allá de las actuales perspectivas.

El entendimiento del lenguaje natural involucra mucho más que al análisis sintáctico de las frases y al análisis léxico de las palabras, siendo estos dos aspectos fundamentales a abordar. Sin embargo, aún cuando hayamos logrado efectuar eficientemente esos dos análisis, el entendimiento real depende de un conocimiento acumulado acerca del dominio en que nos movamos y de los idiomas usados en ese dominio, así como de la habilidad para aplicar conocimiento contextual general para resolver omisiones y ambigüedades, que son elementos asiduos de la forma de hablar de los humanos.

El trabajo que se refleja en esta memoria consiste en el desarrollo y estudio de un módulo, denominado “Supresor de ambigüedades léxicas mediante métodos estadísticos”, para el proyecto “Desarrollo de un generador de analizadores para el tratamiento informático de los lenguajes naturales” (GALENA) que se desarrolla conjuntamente en las tres universidades gallegas: Departamento de Filología Española de la Universidad de Santiago de Compostela, Departamento de Traducción, Lingüística y Teoría de la Literatura de la Universidad de Vigo y, finalmente, el Departamento de Computación de la Universidad de A Coruña.

El proyecto GALENA hace referencia al tratamiento eficaz de la interpretación de los lenguajes naturales (esto es, los lenguajes humanos de comunicación como por ejemplo el español, el gallego, el inglés, el francés, ...) en su forma escrita.

Como soporte básico del anterior proyecto, se ha hecho uso del sistema ICE (Incremental Context-Free Environment), desarrollado por el Prof. Dr. D. Manuel Vilares Ferro en el seno del INRIA[2] para el proyecto EUREKA.

Sobre esta base se distinguieron, inicialmente, tres etapas en el desarrollo:

- Incorporación a ICE de un algoritmo de corrección automática de errores.
- Incorporación de una interfaz gráfica.

[1] *European Strategic Programme for Research in Information Technology*.
[2] *Institut National de Recherche en Informatique et en Automatique*.

- La extensión de ICE mediante técnicas de filtro semántico con el fin de desechar aquellas interpretaciones de los mensajes analizados que no se correspondan con el "uso natural" del lenguaje.

En una fase más avanzada del proyecto se observó que los analizadores léxicos, aunque eran objeto de estudio por parte de diversos grupos de trabajo, no se mostraban en sus niveles actuales con los niveles de calidad exigibles, sobre todo en lo que hace referencia a las lenguas de origen latino. Por esta razón hubo que incluir, en los puntos a desarrollar, un software que realizase un análisis morfológico de las palabras al mismo tiempo que la etiquetación de las mismas.

Llegados a este punto surgió la problemática de que a las palabras, en ese análisis léxico, se les proporcionaban unas etiquetas -es decir, descripciones que contienen toda la información (tipológica, morfológica, ...) necesaria para caracterizarlas en el conjunto del léxico de la lengua- y a cada palabra aislada, esto es, fuera de contexto, pueden corresponderle varias etiquetas, lo que no era viable para llevar a cabo el análisis sintáctico. Surge por tanto la necesidad de eliminar la ambigüedad en el proceso de etiquetación para permitir realizar la interpretación de un texto. Para resolver la ambigüedad es necesario incorporar al sistema un módulo, el "Supresor de ambigüedades léxicas mediante métodos estadísticos", que permita asignar a una palabra una etiqueta (forma verbal, adverbio, preposición, nombre, ...), de forma única, para permitir continuar con el análisis de un texto escrito en lenguaje natural.

En el módulo de Eliminación de ambigüedades se distinguen dos fases:

- Fase de aprendizaje.
- Fase operacional.

En la primera de dichas fases es necesario que el módulo "aprenda" el estilo literario con el que se desarrolló el texto en estudio para permitir una correcta y satisfactoria eliminación de las ambigüedades. El aprendizaje deberá llevarse a cabo con textos, del mismo estilo literario que el que se desee tratar, previamente etiquetados sin ambigüedades por expertos lingüistas.

La segunda de las fases aquí expuestas es la que efectúa la supresión, propiamente dicha, de las ambigüedades haciendo uso de las "enseñanzas" recibidas en la fase previa. Basándonos en las estadísticas elaboradas en la fase anterior, en la historia del texto y las posibilidades que presenta una determinada palabra se seleccionará la etiquetación más probable. En caso de que la elección no haya sido la correcta para el caso tratado y el analizador sintáctico no lograse efectuar el árbol de análisis, en razón del contexto gramatical, se seleccionará la segunda etiqueta más probable, y así sucesivamente.

La implementación que permita llevar a cabo este proceso debe verse avalada por:

- Investigación de los métodos susceptibles de aplicarse.
- Evaluación de dichos métodos.
- Optimización en la implementación del método elegido, que conllevará así mismo la optimización en el uso de los recursos disponibles.
- Validación del software para lograr un corpus lingüístico de entidad suficiente.

Por último, y a modo de simpática curiosidad comentada en el seno del equipo GALENA, indicar que en el Diccionario de la Real Academia Española el término Galena tiene dos acepciones:

- Mineral compuesto de azufre y plomo, de color gris y lustre intenso. Es la mejor mena del plomo.
- Viento suave y apacible.

La sensación que tienen los miembros del proyecto GALENA es que dicho proyecto se ajusta más a la primera acepción, por el pesado trabajo que conlleva, que a la segunda. Sin embargo todo parece indicar que gran parte del camino ya está andado y que todo se volverá algo más suave y apacible para ajustarse a la segunda acepción.

2. PROCESAMIENTO DEL LENGUAJE NATURAL

2.1 INTELIGENCIA ARTIFICIAL Y LENGUAJE NATURAL

2.1.1 CONCEPTOS Y CARACTERÍSTICAS DE LA INTELIGENCIA ARTIFICIAL

Si bien el concepto clave en este apartado es el de inteligencia artificial, se puede retroceder un poco y definir, para tener una mejor visión del término, qué es la inteligencia natural. Una vez aclarado este aspecto derivar el concepto de inteligencia artificial será muy intuitivo.

La inteligencia natural no es un atributo exclusivo de los hombres, puesto que existen otros seres vivos que pueden ser considerados también como inteligentes, pero para lo que nos ocupa nos centraremos en la inteligencia natural de los seres humanos.

La inteligencia natural humana es muy difícil de explicar, evaluar y, por tanto, definir. Se puede pensar en ella como la capacidad de respuesta de un ser inteligente ante determinadas circunstancias. Dicha capacidad depende de muchos factores, entre los que se podrían destacar los siguientes [Angulo y del Moral, 1986]:

- La experiencia.
- La facilidad de adquirir información externa.
- La rapidez de encadenar pensamientos.
- La facilidad en deducir consecuencias.
- ...

Una forma de comprender la inteligencia natural es fijarse en las características que presentan las personas inteligentes, para dar lo que se conoce como una definición fenomenológica del término. Siguiendo con esta filosofía, cabe indicar que un ser humano inteligente se caracteriza por poseer las siguientes facultades:

- **Aprendizaje**

  Es capaz de recoger información del exterior (a través de la vista, del oído, etc.) y guardarla en la memoria. De esta forma, la información que posee en su interior es cada vez mayor, a medida que percibe a través de sus sentidos.

- **Razonamiento y deducción**

  Manipula los conocimientos e información que posee, aplicando una serie de reglas y experiencias, ya formales ya intuitivas, para deducir y solucionar los problemas que se le presentan en el discurrir de su vida.

- **Realización de las soluciones elaboradas**

  Comunica sus deducciones al exterior bien por medio de los movimientos de sus órganos (brazos, piernas, pies, manos, etc.) o bien por medio de la palabra (lenguaje natural).

- **Generación de sentimientos**

  La alegría, la tristeza, la bondad o el amor son sentimientos que se desprenden de los razonamientos inteligentes y que se encuentran en los seres

considerados con cierto grado de inteligencia. De hecho son cualidades que se apuntan como típicas de los seres humanos.

Desde hace siglos el hombre ha mostrado un enorme interés en la construcción de máquinas e ingenios que imitasen lo mejor posible, y subsanando los posibles defectos, a los seres vivos, y fundamentalmente, al hombre con especial hincapié en lo que se refiere a la inteligencia. De esta forma, y ya desde la antigüedad, el hombre se quiere parecer a los dioses creando seres a su imagen y semejanza. Tales inquietudes las encontramos en obras literarias como por ejemplo Frankenstein, de Mary Shelley o 2001: Odisea en el espacio, de Arthur Clarke, con el malvado ordenador llamado HAL, o incluso con C-3PO y R2-D2 en "La guerra de las galaxias".

Dejando a un lado la ficción, y entrando en la realidad del siglo XX, nos encontramos con que el ordenador cubre la necesidad imperiosa de almacenar y procesar la ingente cantidad de información que exigen los procesos modernos de nuestra era, y es el candidato idóneo para conseguir imitar a los seres humanos.

La inteligencia artificial (en adelante IA) ha surgido con la idea de copiar, en cierta medida, la inteligencia natural. Para conseguirlo, se ha escogido la herramienta disponible más potente y sofisticada en la actualidad: el ordenador o computadora.

Esta máquina es capaz de procesar gran cantidad de datos en muy poco tiempo, siguiendo las directrices definidas en un programa. Por esta razón, hasta nuestros días, se ha considerado a la computadora una máquina tonta ya que, como se ha dicho con anterioridad, sólo realiza aquello que se haya programado previamente. Esta manera de operar suprime una de las cualidades principales de la inteligencia, que consiste en un comportamiento diferente para idénticas situaciones, comportamiento que no ha sido previamente programado, sino que se produce como consecuencia de una manipulación inteligente de los conocimientos y experiencias que se poseen. Lo que se pretende en la actualidad es dotar de cierta inteligencia a los ordenadores intentando desterrar el mito de la máquina tonta.

Una de las formas más comunes para medir los avances logrados en este campo de investigación es el juego del ajedrez, debido a su propia naturaleza. Un fiel reflejo de este hecho se pudo observar, no hace mucho tiempo, con el duelo entre el maestro de ajedrez Kasparov y el ordenador Deep Blue, en el cual venció la máquina. Esta victoria, más que a la inteligencia, se pudo deber según los expertos a la enorme potencia de cálculo del ordenador y a su falta de condiciones humanas, como por ejemplo su estado de ansiedad, de humor, etc.

Llegados a este punto surge la pregunta de cuándo considerar una máquina inteligente. Tradicionalmente los autores entendidos en la materia consideran una máquina inteligente cuando cumple los siguientes requisitos:

- Es capaz de percibir visualmente los objetos que la rodean y reconocer sus formas.
- Es quien de elaborar actuaciones de acuerdo con las condiciones cambiantes del entorno y llevarlas a cabo mediante los correspondientes elementos físicos.
- Cuando puede almacenar información y conocimientos, que manipula mediante reglas y algoritmos para alcanzar soluciones a los problemas que plantea su funcionamiento.
- Una máquina se puede calificar como inteligente si es capaz de entender el lenguaje natural, hablado o escrito, así como de producir respuestas en dicho lenguaje.

¿Cómo sabremos que hemos desarrollado una máquina inteligente? es la siguiente pregunta que cabe plantearse. Para responder a esta pregunta, y por tanto saber si un ordenador es inteligente, el matemático británico Alan Turing propuso la siguiente prueba en un artículo titulado “Computing machinery and intelligence” en 1950, que en principio denominó “juego de imitación”, pero que actualmente, y en su honor, se denomina prueba de Turing (Fig. 1):

"Un ser humano aislado se comunica a través de dos terminales; con uno se comunica con otro ser humano con capacidad normal de pensamiento y con el otro se comunica con una máquina sospechosa de tener capacidad de razonar. Si el ser humano no consigue distinguir claramente, mediante el análisis de las respuestas o resultados, la máquina del ser humano, se deduce que aquella máquina tenía cierto grado de inteligencia."

Lógicamente no tienen cabida aquellas preguntas que permitan detectar el potencial de cálculo del ordenador, como por ejemplo preguntar el resultado de un cálculo complejo, ya que la diferencia en este aspecto es claramente favorable al ordenador y no es el objetivo perseguido. Para solventar este inconveniente Turing propone que la máquina no siempre tenga que dar una respuesta correcta a tales preguntas.

Esta prueba se podría actualizar de la siguiente forma para liberarla de la apariencia tan artificial que posee:

En vez de comunicarse a través de terminales, ¿por qué no hacerlo a través del lenguaje natural hablado?

Cuando Turing planteó su prueba esta posibilidad era impensable, pero actualmente, basándose en los sistemas desarrollados de habla artificial, sí es posible. Manteniendo el aislamiento, para que el interrogador no tenga prejuicios derivados del aspecto de la máquina, podría entrar en juego un sistema de reproducción de voz que debería eliminar cualquier indicio mecánico de la voz para evitar su descubrimiento.

La distinción, sin embargo, en nuestros días se haría evidente, ya que la máquina no es capaz de construir expresiones en lenguaje natural de la misma forma que lo hacen los seres humanos, y mantener por tanto una conversación sin ser descubierta. Es decir, el dominio que en la actualidad tienen las máquinas del lenguaje natural es muy reducido. Todo esto sin olvidar que la forma de razonar delatará a la máquina ya que, en un ámbito general, las máquinas no han conseguido equipararse a los humanos en este aspecto, u otros, como por ejemplo, juzgar un poema o una obra de arte.

La principal conclusión que se obtiene es que hay que procurar la comunicación hombre-máquina mediante el uso del lenguaje natural hablado o, por extensión, del escrito, para conseguir lo que hace tanto tiempo que el hombre persigue: construir ingenios inteligentes semejantes a él.

En segundo término aparece la necesidad de igualar la calidad en el razonamiento de las máquinas en comparación con el hombre, aspecto éste que en algunos campos de actuación ya se ha logrado (sistemas expertos en medicina, etc.).

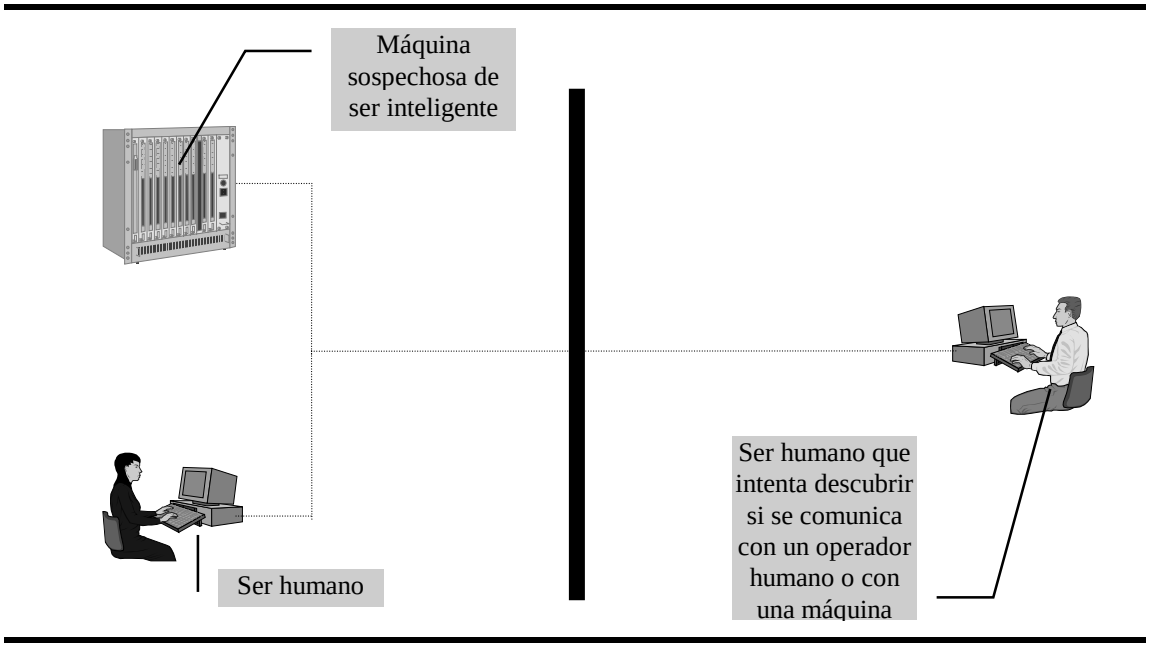


Figura 1. Prueba de Turing.

2.1.2 EL LENGUAJE NATURAL VISTO DESDE LA INTELIGENCIA ARTIFICIAL

Cuando se inventaron los primeros ordenadores la única forma de poder darles instrucciones, para que realizasen una determinada operativa, era conectar y reconectar una serie de cables; lo que de alguna manera recordaba una antigua centralita telefónica. Aunque la facilidad de manejo en los ordenadores actuales se ha incrementado notablemente, la comunicación con ellos todavía no es tan sencilla como la forma más natural de comunicación que existe entre las personas: el lenguaje natural. Cuando nos comunicamos con otras personas no tenemos que utilizar ningún lenguaje técnico especializado, simplemente usamos los lenguajes naturales.

La investigación en lingüística computacional, esto es, el uso de los ordenadores en el estudio del lenguaje, empezó tan pronto como los ordenadores estuvieron disponibles en los años 40. La habilidad de la máquina (ordenador en nuestro caso) para manipular símbolos fue enseguida aplicada a texto escrito para recopilar índices de palabras (listas de ocurrencias de palabras) y concordancias (índices que incluían una línea de contexto para cada ocurrencia). Este procesamiento de texto es de un nivel muy superficial pero sin embargo fue de algún valor en la investigación realizada en el campo de la lingüística, si bien es cierto que rápidamente se vio que el ordenador podía realizar funciones lingüísticas más importantes que una simple cuenta u ordenación de datos.

En 1949 Warren Weaver propuso que los ordenadores podrían ser útiles para "la solución de los problemas de traducción del mundo entero". La investigación resultante, en lo que se dio en llamar "*machine translation*" (traducción por máquina o traducción mecánica), intentó simular con un ordenador las funciones que se intuía que debía tener un traductor humano: buscar cada palabra en un diccionario bilingüe, escoger una palabra equivalente en el lenguaje de salida y, antes de procesar cada sentencia, ordenar

la cadena resultante de palabras para imponer el orden de palabras correcto según el lenguaje que se utiliza como salida. Todos los proyectos de esa época se basaban en la falsa creencia de que la traducción era un simple proceso mecánico, que podría ser realizado fácil y rápidamente por los ordenadores. Además hay que tener en cuenta el momento sociológico y político de aquella época, en la que la guerra fría y la falta de comunicación eran constantes en el desarrollo de las relaciones. Todo ello favoreció el tema ya que ofrecía el intercambio de ideas e información entre personas de distintos países, sin tener que depender del intérprete, que estaba sujeto, como toda persona, a errores que podían resultar muy embarazosos en determinadas conversaciones. Otro objetivo perseguido con la traducción mecánica era evitar cualquier factor humano (odio, favoritismo, etc.) que pudiese desestabilizar las ya de por sí tensas relaciones.

Los primeros sistemas desarrollados constaban de dos partes fundamentales:

- Un diccionario bilingüe, que relacionaba las palabras de los dos idiomas entre sí.
- Una gramática para cada idioma, que añadía las terminaciones a las palabras y ordenaba los elementos de la oración.

Los escasos avances que se alcanzaron hasta la primera parte de la década de los 60 convencieron a algunos especialistas, tales como Bar Hillel, Weaver y Booth, de que la comprensión del lenguaje natural precisaba de algo más que un simple diccionario y una gramática como se pensaba en los sistemas pioneros.

Entonces la comprensión del significado de las palabras se basaba en la sintaxis (gramática), sin tener en cuenta la semántica. Por esta razón los investigadores de la época tropezaron con problemas a la hora de utilizar palabras cuyo significado dependía del contexto de la frase en que fuesen usadas, también con las expresiones idiomáticas propias y con las frases hechas o las ambigüedades sintácticas.

Muchos expertos que trabajaban en el ámbito del lenguaje natural abandonaron su empeño ante estas y otras dificultades, y otros variaron la orientación de sus estudios y experiencias. En vez de intentar realizar la traducción del lenguaje por medio de ordenadores, se emprendió la tarea de hacer que el ordenador comprendiera el lenguaje. Casi todas las complicadas técnicas usadas en la comprensión del lenguaje natural derivan de la IA, de ahí la íntima relación entre estos dos campos, que a veces deriva en la subordinación del tratamiento del lenguaje natural a la IA por parte de algunos autores.

Hoy en día los estudios que se realizan sobre el tema se encaminan hacia la intercomunicación entre el hombre y la máquina a través del lenguaje natural, y no a la mera traducción asistida por ordenador.

El entendimiento del lenguaje natural se convirtió en el foco de la investigación de la IA en el área del lenguaje -si la máquina puede entender el significado de una frase, podría, presumiblemente, parafrasearla, contestar preguntas acerca de ella o traducirla a otro lenguaje. Sin embargo, la naturaleza de entender es, en sí mismo, un problema muy difícil.

Nuevas aproximaciones de la IA al procesamiento del lenguaje natural (en adelante PLN) se vieron influidas por algunos desarrollos científicos en los años 60: lenguajes de programación de alto nivel, procesamiento de listas, mayor capacidad de procesamiento y memoria y los adelantos de Chomsky en la teoría lingüística. En esta época los investigadores de IA desarrollaron un nuevo grupo de programas de ordenador intentando dar solución a algunos de los asuntos que habían frustrado los intentos con la idea de "*machine translation*". Estos nuevos programas de lenguaje natural marcaron el

inicio del trabajo serio de la IA en el entendimiento del lenguaje. Se empezó a ver el lenguaje humano como una compleja habilidad cognitiva que involucraba conocimiento de diferentes clases: estructura de las sentencias, el significado de las palabras, un modelo del oyente, las reglas de conversación, y una extensa información acerca del mundo en general.

La aproximación general de la IA ha sido al modelo del lenguaje humano como un sistema basado en el conocimiento para procesar comunicaciones y para crear programas que sirvan como instancias de trabajo de esos modelos.

Los investigadores en el PLN esperan que su trabajo sirva tanto para el desarrollo de sistemas de entendimiento del lenguaje natural útiles y prácticos, como para un mejor entendimiento del lenguaje (interés de los lingüistas) y de la naturaleza de la inteligencia (interés de los informáticos).

Desarrollando y probando modelos, basados en computadoras, de procesamiento de lenguaje que se aproximen al comportamiento humano, los investigadores esperan entender mejor cómo trabaja el lenguaje humano, a la vez que avanzar en el campo de la IA.

2.1.2.1 EL ENTORNO DEL PLN

La capacidad de comunicarse a través de un lenguaje es uno de los rasgos distintivos de la especie humana. Es por ello que el tratamiento del lenguaje natural, procesamiento del lenguaje natural también llamado, ha constituido desde sus mismos orígenes (el primer sistema de traducción automática, de Weaver y Booth, se remonta a 1946, es decir, coincide, prácticamente, con el nacimiento de los ordenadores comerciales) una parte importante de la inteligencia artificial e, incluso, de la informática en general.

Es un poco absurdo pretender que el PLN esté englobado en la IA de una forma tajante y sin posibilidad de discusión, lo mismo que pretender que se pueda calificar de una parte de la lingüística o, incluso, de la lingüística computacional. En realidad el PLN utiliza técnicas y formalismos tomados de éstas y otras disciplinas para conseguir sus fines, que no son otros que la construcción de sistemas computacionales para la comprensión y la generación de textos en lenguaje natural. Llegados a este punto conviene precisar estos conceptos:

La primera consideración que hemos de hacer es que al hablar de lenguaje natural nos estamos refiriendo al lenguaje humano. Es decir, lenguaje natural se opone a lenguaje artificial o a lenguaje formal. De ahí proviene la primera, importante, diferencia. El lenguaje natural es, si no inabarcable, sí mucho más difícil de abarcar que cualquier lenguaje formal.

El PLN se ha alimentado durante décadas de las técnicas diseñadas para el tratamiento de los lenguajes artificiales (especialmente con los compiladores de los lenguajes de programación), provenientes de la Teoría Formal de Lenguajes o de la Teoría de la Compilación. Por supuesto, el lenguaje natural no es un lenguaje de programación ni mucho menos y de ahí que el PLN se haya convertido en una disciplina diferenciada debido a su propia naturaleza.

Si el PLN se refiere al lenguaje humano, otra disciplina con la que debe tener afinidades es la lingüística. La lingüística es, probablemente, la más formalizada de las ciencias humanísticas. La razón de ello es que el lenguaje, objeto de su estudio, presenta unas características altamente estructuradas y, por lo tanto, es más susceptible de formalización que otros aspectos de la actividad humana. La lingüística trata del estudio

del lenguaje tanto en general como de las diferentes lenguas particulares que los seres humanos practican. E, incluso, trata de los sublenguajes: subconjuntos más o menos cerrados de un lenguaje que permiten un tratamiento formal o computacional más eficientes.

Durante años, el desarrollo de la lingüística (lógicamente anterior) y el del PLN siguieron caminos casi paralelos con apenas transferencia de resultados de uno a otro. Los lingüistas construían sus teorías, básicamente sintácticas, sin tener en cuenta la posible realización computacional de las mismas, y los informáticos trataban las aplicaciones informáticas del lenguaje natural como un problema más de ingeniería del software. No se llegó muy lejos en ninguna de las disciplinas (a pesar de éxitos inicialmente notables en ambas) y finalmente se impuso la sensatez. En la actualidad cualquier propuesta de teoría lingüística lleva aparejado el estudio de su componente computacional (básicamente expresada en cómo aplicarla al análisis y/o a la generación del lenguaje y con qué coste computacional hacerlo). Cualquier aplicación seria del lenguaje natural, descansa, por otra parte, en teorías o modelos lingüísticos previamente establecidos.

En cuanto el PLN adquirió un cierto nivel de utilización y las exigencias de calidad aumentaron, se hizo patente que una adaptación pura y simple de los métodos y técnicas provenientes del tratamiento de los lenguajes artificiales no conducía a nada. Se vio que el proceso de comprensión (y, más adelante, de generación) del lenguaje natural implicaba el manejo de una cantidad ingente de conocimiento de índole diversa y la utilización de procesos inteligentes. La IA proporcionó la base para elevar decisivamente el nivel y las prestaciones de los sistemas de PLN.

Un último punto a mencionar, a caballo entre la Psicología y la Lingüística, es el de la Psicolingüística, que estudia los mecanismos humanos de comprensión y generación del lenguaje. Aquí se reproduce el eterno dilema de la IA, ¿debe la IA modelar formas humanas de tratamiento de los problemas, o debe limitarse a obtener resultados similares a los que los seres humanos, enfrentados a los mismos problemas, obtendrían? Ciñéndonos a la comprensión o la generación del lenguaje natural: ¿deben las máquinas intentar reproducir las formas humanas de tratamiento, de acuerdo a lo que la Psicolingüística proponga, o deben explorar sus propios métodos de procesamiento, más adecuados a sus peculiaridades? Como es lógico podemos encontrar abundantes ejemplos de uno y otro signo.

La Figura 2 expresa gráficamente las consideraciones realizadas.

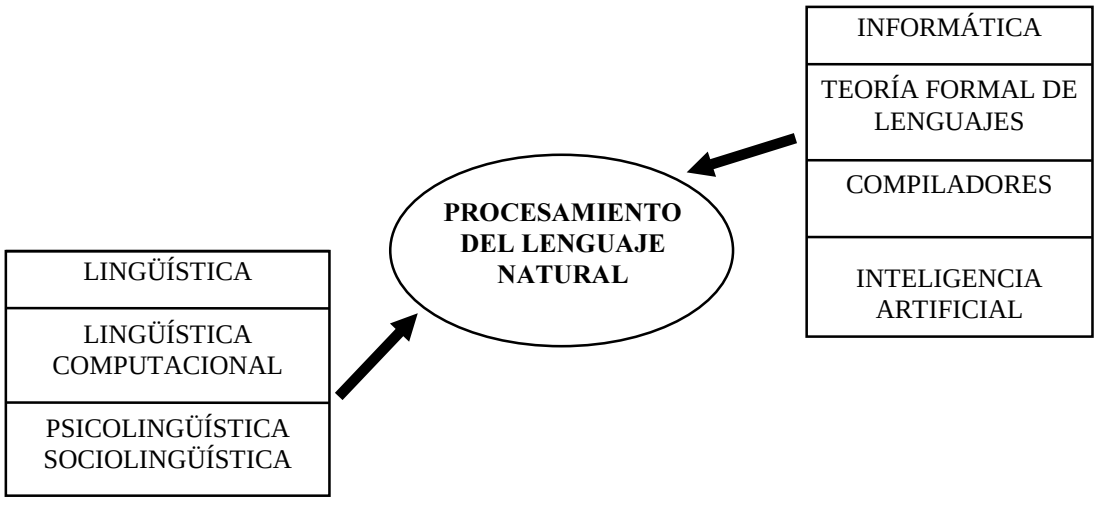


Figura 2. El entorno de PLN.

2.1.2.2 BREVE HISTORIA DEL PLN

Como ya se ha apuntado, la historia del PLN se remonta a los orígenes mismos de la informática. En 1946, Weaver y Booth presentaron el primer sistema de traducción automática, seguido poco después por el GAT ("Georgetown Automatic Translator") y, ya en 1961, por CETA ("Centre d´etudes pour la Traduction Automatique") en Grénoble.

En 1964 el informe ALPAC, cancelando los fondos para los proyectos de traducción automática en los EE.UU. y negando su viabilidad, supuso un gran freno para estos primeros intentos pero no impidió el nacimiento de programas tan importantes como METAL o SYSTRAN [Toma, 1977]. Era la época en que el PLN se apoyaba en métodos bastante rudimentarios como los analizadores de *pattern matching*[3], utilizados en sistemas como ELIZA, SIR, STUDENT o BASEBALL [Barr y Feigenbaum, 1989].

En los años 70 continúan los esfuerzos en traducción automática como SUSY, en Saarbrucken, GETA en Grénoble o TAUM-METEO en Montreal. Paralelamente comienzan a construirse las primeras interfaces en lenguaje natural, primero con bases de datos y más adelante con otras aplicaciones. LUNAR [Woods, 1970], Rendezvous [Codd, 1974] o LADDER [Hemdrix et al., 1978] son algunos ejemplos. En paralelo aparecen sistemas más evolucionados de PLN como las Gramáticas Procedurales (Winograd, SHRDLU, 1971) y una larga lista de analizadores de muy diversa índole (MARGIE, SAM, etc.).

Ya en los 80 asistimos a la incorporación de los nuevos formalismos sintácticos a los que antes nos referíamos, a la aparición de las gramáticas lógicas y los formalismos de unificación y a la construcción de sistemas cada vez más evolucionados.

Sistemas como Ariane-78, EUROTRA o ATLAS, en el campo de la traducción automática, interfaces como TEAM, CHAT-80, FRED o DATALOG, en el campo de las Interfaces con bases de datos y otros como XCALIBUR, XTRA, INKA, en otras aplicaciones, completan el panorama de estos años.

Con respecto a la época de los 90 hay que resaltar varios puntos:

En primer lugar, se han ido incorporando a los sistemas de PLN algunos de los formalismos, sobre todo sintácticos, que se introdujeron como novedad en los 80.

En segundo lugar, igualmente han seguido desarrollándose algunos de estos formalismos (las gramáticas de cláusulas definidas han evolucionado para dar lugar a formalismos más adecuados a la descripción lingüística).

Se ha detectado la necesidad de adquisición masiva de información léxica y gramatical y se han construido sistemas para el aprovechamiento de fuentes de información léxica ya existentes, como los diccionarios para uso humano en soporte informático (proyecto Acquilex) o los córpora textuales.

Las interfaces han sido perfeccionadas en las líneas de mejora de la calidad del diálogo, de las cualidades de transportabilidad de los productos y construcción de sistemas integrados (NAT, FRED).

Se han establecido criterios de separación clara entre la descripción de la estructura de la lengua (*Competence*) y el tratamiento, usando esa descripción, de los casos concretos (*Performance*). La descripción debe estar orientada al usuario: lingüista, lexicógrafo, usuario final, etc. y favorecer, por lo tanto las características de claridad,

[3] Reconocimiento de patrones.

amplia cobertura, cohesión, reutilización, etc. El tratamiento estará orientado a la máquina y deberán, por lo tanto, primar los criterios de eficiencia, tanto espacial como temporal. Los sistemas deberán proveer mecanismos de transferencia entre uno y otro formalismo.

2.1.2.3 APROXIMACIONES HISTÓRICAS AL PLN

Los proyectos de investigación en lenguaje natural han tenido diversos énfasis y han empleado diversos métodos haciendo su clasificación difícil. Un esquema coherente surgido de Winograd (1972) agrupa los programas de lenguaje natural de acuerdo a cómo representan y usan el conocimiento de su materia. Partiendo de esta base los programas de lenguaje natural pueden ser divididos en cuatro grandes categorías históricas que a continuación se exponen.

Los programas de lenguaje natural más antiguos buscaron únicamente alcanzar resultados limitados en dominios muy específicos y restringidos. Estos sistemas, como BASEBALL, STUDENT o ELIZA, usan estructuras de datos "*ad-hoc*" para almacenar hechos acerca de un dominio limitado. Las sentencias de entrada se restringieron a simples formas declarativas o interrogativas, las cuales se examinaban para ver si existían palabras claves predeclaradas o patrones que indicasen objetos o relaciones conocidas. Las reglas específicas del dominio tratado, llamadas heurísticas, son usadas para derivar la respuesta requerida empleando las palabras claves de la frase y el conocimiento en la base de datos. Como el dominio de discurso de estos sistemas era muy restringido se ignoraban muchas de las complejidades del lenguaje, lo que derivaba, a veces, en resultados imprevisibles en las respuestas a las preguntas realizadas.

La segunda categoría histórica está representada por sistemas tales como PROTOSYNTHEX-I y SEMANTIC MEMORY. Estos sistemas, esencialmente, almacenaban una representación del propio texto en sus bases de datos usando una gran variedad de esquemas de indexación muy ingeniosos para recuperar material que tuviese palabras o frases específicas. En esta aproximación basada en el texto (*text-based*) los sistemas no nacen, por su construcción, para un dominio específico, como sucedía con la categoría anterior, ya que la base de datos textual puede cubrir cualquier tema. Sin embargo, todavía se encuentran con restricciones muy fuertes ya que sólo pueden responder con material que haya sido prealmacenado explícitamente. Al igual que sus predecesores, estos programas siguen fallando incluso en implicaciones obvias de las sentencias que están en la base de datos. Esto es debido a que no tratan el significado de la entrada en lenguaje natural, esto es, no tienen capacidad deductiva.

Para acercarse al problema de cómo caracterizar y usar el significado de las sentencias surge un tercer grupo de programas a mediados de los años 60. En estos sistemas de lógica limitada[4], donde se incluyen sistemas como SIR, DEACON y CONVERSE, la información en la base de datos se almacena en alguna notación formal y se proporcionan mecanismos para traducir sentencias de entrada en esa forma interna. La principal ventaja de estos sistemas es que se

[4] *Limited-logic systems*.

pueden realizar inferencias en la base de datos con el objetivo de encontrar respuestas a las preguntas que no están explícitamente almacenadas en la base de datos. Por ejemplo si a un sistema se le dice que Sultán es un lobo y que todos los lobos son mamíferos el sistema debería ser capaz de responder a la pregunta ¿es Sultán un mamífero? Los sistemas de esta época estuvieron limitados en el sentido de que las deducciones que se hacían sólo eran un subconjunto de todo el amplio rango de inferencias lógicas que típicamente se usan en una conversación.

El cuarto grupo de programas entendedores de lenguaje natural podría llamarse sistemas basados en el conocimiento[5]. Su desarrollo está íntimamente relacionado con la investigación en el campo de la representación del conocimiento y se encuadran en los años 70. Estos programas usan una gran cantidad de información acerca del dominio de discurso para ayudar a entender las sentencias. Este conocimiento se almacena con el programa usando algún tipo de esquema de representación del conocimiento, como pueden ser *frames*, redes semánticas, lógica, etc. Ejemplos de sistemas encuadrados en este grupo son LUNAR, SHRDLU, MARGIE, SAM, LIFER, ... Todos estos sistemas de alguna manera tratan aspectos sintácticos y semánticos del procesamiento del lenguaje.

2.1.2.4 APORTACIONES DE LA IA AL PLN

La razón básica para incluir el PLN dentro de la IA es que la comprensión del lenguaje natural se considera una forma clara de comportamiento inteligente. Hemos de admitir, sin embargo, que buena parte de las técnicas y métodos que se aplican para tratar el lenguaje natural poco o nada tienen que ver con las propias de la IA, sino que se importan de disciplinas ligadas al tratamiento de los lenguajes formales.

Sin embargo, y a medida que las exigencias de calidad en el PLN han aumentado, la necesidad de utilizar fuentes de conocimiento extensas y complejas y de emplear tratamientos no estrictamente algorítmicos ha dado lugar a una utilización creciente de la IA, como se ha podido apreciar en el apartado anterior.

Entre las herramientas de la IA que se emplean extensamente en el PLN encontramos los siguientes:

- Sistemas de representación del conocimiento tanto basados en lógica (usados en los niveles sintáctico y lógico), como en redes semánticas (gramáticas de casos [Barr y Feigenbaum, 1989]) o en modelos de objetos estructurados, *frames* (modelos de representación conceptual y léxica, formas complejas de inferencia, herencia, etc.).
- Sistemas de planificación (planificación de diálogos, generación de lenguaje natural a partir de planes, generación de explicaciones, etc.).
- Sistemas de búsqueda heurística (estrategias de análisis sintáctico, cooperación sintaxis/semántica, etc.).
- Sistemas de razonamiento (resolución de anáforas, determinación del ámbito de los cuantificadores, etc.).

[5] *Knowledge-based systems*.

- Sistemas de representación y razonamiento aproximado e incierto (lógica de modalidades, cuantificadores difusos, información incierta, analizadores probabilísticos, etc.).

Se ha de tener en cuenta, por otro lado, que un gran número de aplicaciones de PLN actúa como interfaz de sistemas inteligentes, en los que se integran. Buena parte de los sistemas de representación de conocimiento e información que se utilizan al tratar el lenguaje natural deben tener en cuenta esta doble función.

2.2 PROCESAMIENTO DEL LENGUAJE NATURAL

Actualmente el ámbito de la IA está logrando un considerable auge dentro de la informática y, aunque su campo de acción es amplio, se puede considerar que los temas fundamentales que abarca son:

- Sistemas expertos.
- Resolución de problemas.
- Lógica e incertidumbre.
- Robótica.
- Aprendizaje de las máquinas.
- Visión y reconocimiento de modelos.
- Procesamiento del lenguaje natural.

El proyecto de Fin de Licenciatura que aquí se expone se puede encuadrar dentro del área de estudio denominada procesamiento del lenguaje natural.

Para muchos investigadores en IA, el PLN es uno de los fines principales que la IA debe lograr, ya que permite a la computadora la entrada del lenguaje humano de forma directa. Este fin no está exento de obstáculos debido al gran tamaño y complejidad de los lenguajes humanos, como ya se podrá comprobar en apartados sucesivos. Además se tiene el problema añadido de que la computadora debe de considerar la información contextual que pueda aparecer en cualquier situación, que no necesariamente tiene que ser de las más simples.

Una vez se logren avances en el PLN se abre una puerta a los diálogos directos entre el hombre y la computadora, lo que evitaría la programación convencional y el protocolo del sistema operativo que gobierna la máquina.

Dar una definición de PLN resulta complicado, pero se podría considerar que es el área de estudio de la IA que trata de hacer a la computadora capaz de entender órdenes escritas en lenguaje natural, esto es, en lenguaje humano.

Si las computadoras fuesen capaces de comprender el lenguaje natural, podríamos decirles en lenguaje ordinario lo que querríamos que hicieran, y si pudieran generar lenguaje natural, nos podrían hacer preguntas y darnos información en un lenguaje fácil de entender para nosotros. La comprensión y la generación del lenguaje natural son los dos componentes del PLN, cuyo objetivo final es hacer posible que la comunicación con los ordenadores sea tan sencilla como la comunicación con las personas.

El trabajo que aquí se presenta se encuentra bajo el área de trabajo de la comprensión del lenguaje natural.

El PLN se refiere, normalmente, al lenguaje escrito sobre un teclado, que se imprime o se visualiza en pantalla, más que al lenguaje hablado. De este último se ocupan las tecnologías de reconocimiento y entendimiento del habla.

2.2.1 ENTENDER EL LENGUAJE NATURAL

Llegar a entender el lenguaje natural es una ardua tarea de la que a primera vista no se es consciente. A pesar de que los niños pequeños aprenden en un tiempo relativamente corto, no debemos de olvidarnos que poseen la inteligencia propia de los humanos y que ésta es muy difícil de trasladar al ordenador. A través de los siguientes puntos se reflejará la dificultad subyacente al intentar entender el lenguaje natural.

2.2.1.1 LA COMPRENSIÓN DE LAS PALABRAS

La mayor parte de la comunicación humana se realiza mediante el lenguaje oral. Sin embargo, es mucho más fácil entender el lenguaje escrito que el oral. Para construir un sistema que entendiese el lenguaje oral, habría que considerar, además del mecanismo que fuese capaz de entender el contexto, la entonación, el acento, las pausas e, incluso, el ruido ambiental.

Existen dos campos en el estudio del entendimiento del lenguaje natural. Por una parte nos encontramos con el entendimiento del lenguaje escrito, que utiliza el conocimiento léxico, sintáctico y semántico del lenguaje, unido a la información o conocimiento del dominio. Por otra parte se encuentra el entendimiento del lenguaje oral, que comprende todo lo anterior junto con la fonología.

Nos centraremos en el primer caso observando las técnicas que se pueden emplear para entender oraciones simples y no relacionadas entre sí, puesto que para poder entender un párrafo con varias frases habría que proporcionar a la máquina la habilidad de buscar las relaciones entre las oraciones.

Para entender una oración hay que considerar tres aspectos:

- Comprender el significado de cada palabra de la oración (análisis léxico).
- Comprender la estructura de la frase (análisis sintáctico).
- Tener conocimiento del contexto en que se pronunció y lo que significa (análisis semántico).

Para entender el significado de cada palabra, por tanto en el análisis léxico, se puede incluir en el sistema un diccionario que tenga a todas ellas junto a lo que quieren decir. Dejando a un lado el hecho de que esto sea o no viable, hay palabras con varios significados, y dependiendo del contexto en que aparezcan significarán una cosa u otra. ¿Cómo sabría la máquina cuál de los significados tomar?

2.2.1.2 LA COMPRENSIÓN DE LAS FRASES

Otro de los objetivos en la comprensión del lenguaje natural escrito, aparte del entendimiento de las palabras por separado, es entender la estructura de la oración, que también posee una complejidad muy elevada. Esto es lo que se conoce como el análisis sintáctico, el cual determina la estructura gramatical de la frase. Se efectúa mediante un

proceso llamado *parsing* o análisis sintáctico de la oración, que consiste en descomponerla en sus elementos constituyentes.

Para realizar el análisis de la oración se precisa de una gramática que describa ese lenguaje, para luego poder construir el árbol de derivación que represente a la sentencia de entrada.

Una gramática, formalmente definida, es una cuádrupla que viene representada por la siguiente expresión:

$$G = (Vt, Vn, P, S)$$

siendo

- Vt: alfabeto terminal, es decir, el conjunto de cadenas que conforman el lenguaje generado por esa gramática: las palabras.
- Vn: es el alfabeto no terminal que consiste en el conjunto de clases gramaticales que describen el lenguaje generado.
- P es el conjunto de reglas de producción o reglas de reescritura que nos indican cómo se pueden combinar los anteriores elementos gramaticales.
- S es el símbolo de comienzo de la gramática.

Según sean las reglas de producción se obtiene un tipo u otro de gramática.

Un lenguaje (L(G)) es el conjunto de cadenas que se pueden generar partiendo del símbolo inicial de la gramática (S) aplicando las reglas de producción (P).

La mayor dificultad que se encuentra en este punto es dar una gramática para el idioma que se esté considerando. Esta labor es inmensa y el mayor esfuerzo cae del lado de los lingüistas. Posteriormente la dificultad recae en el campo de la informática, ya que hay que proveer de sistemas reconocedores eficientes para esa gramática.

2.2.1.3 LA INTERPRETACIÓN SEMÁNTICA DE LAS FRASES

Una vez descompuesta la oración y comprobado que sigue la gramática establecida se procede a la interpretación semántica.

Existen varias formas de resolver esta fase:

- **Gramáticas semánticas**

  Son gramáticas que combinan todo tipo de conocimientos, tanto sintácticos como semánticos, en un único tipo de reglas de producción. Consiste en una muy exitosa modificación a la aproximación tradicional de las gramáticas con estructura de frase. Implica cambiar la concepción de las clases gramaticales desde el tradicional <NOMBRE>, <VERBO>, etc., a clases que están motivadas por conceptos en el dominio del discurso.

  Por ejemplo, una gramática semántica para un sistema que trate de reservas de vuelos puede tener clases como <DESTINO>, <VUELO> u <HORA-DE-VUELO>. Las reglas de producción usadas por el analizador sintáctico (*parser*) deberían describir frases y cláusulas en términos de estas categorías semánticas.

- **Gramáticas basadas en el caso**

Son gramáticas en las que el análisis aplicado contiene cierta información semántica. Un ejemplo sencillo sería el que se presenta a continuación.

Dadas las siguientes frases:

"Javier hizo la tortilla"
"La tortilla fue hecha por Javier"

Los papeles semánticos de "tortilla" y "Javier" son los mismos, pero los papeles sintácticos están cambiados en las frases. Si se usa una gramática de casos se evita este problema puesto que la asignación de casos es la misma, aunque haya cambiado la estructura superficial.

La representación del ejemplo anterior sería:

(hacer (AGENTE: Javier) (DATIVO: tortilla))

Por último también hay que realizar lo que se conoce como el análisis interpretativo o el análisis pragmático, que traduce lo que se ha escrito a lo que se quería decir. Esto es, a la típica frase de ¿sabe usted qué hora es? la respuesta no debería ser SÍ o NO, ya que en realidad eso no es lo que queremos que se nos conteste. Queremos saber la hora concreta. Una respuesta afirmativa o negativa descubriría al ordenador en la anteriormente mencionada prueba de Turing.

2.2.2 TÉCNICAS DE COMPRENSIÓN DEL LENGUAJE NATURAL

Como todas las aproximaciones racionales a los problemas de gran complejidad, la dificultad del lenguaje natural se considera rompiendo la complejidad del problema en una colección de subproblemas de menor problemática.

Una división tradicional y simplificada del problema considera los siguientes niveles de trabajo:

- Nivel léxico.
- Nivel sintáctico.
- Nivel semántico.

Según Herbert Schildt el corazón de cualquier sistema de PLN es el analizador. El analizador, según Schildt y de forma muy sucinta, es la sección el código que lee cada oración, palabra por palabra, para decidir lo que es.

Si se es más concreto, y siguiendo con la concepción en niveles anteriormente presentada, habría que indicar que realmente, en un sistema de PLN, no se está hablando de un único analizador sino de tres, que juntos conformarían un macro-analizador. Esos tres analizadores son los mismos que nos encontramos en un compilador de lenguajes de programación. Esta situación era previsible ya que un compilador debe entender un programa escrito en un determinado lenguaje, que es, salvando las distancias, lo que se

pretende con el PLN, pero teniendo el lenguaje empleado por los seres humanos como objetivo.

Los tres analizadores, que se han mencionado anteriormente y que siguen los niveles expuestos, son los siguientes:

- Analizador léxico (análisis léxico).
- Analizador sintáctico (análisis sintáctico).
- Analizador semántico (análisis semántico).

El analizador léxico concierne a las palabras y al vocabulario del lenguaje, mientras la sintaxis trata de las estructuras permitidas a los ítems lexicales, y por tanto es cuestión de gramática.

El estudio de la sintaxis sólo nos proporciona un armazón para la comprensión de una frase y, según Elaine Rich, es el primer paso que hay que dar para llegar a ese objetivo. Posteriormente es necesario realizar su interpretación semántica.

El análisis semántico es aquel que interpreta la frase de acuerdo con su significado y no de acuerdo con su forma. La semántica es un aspecto particularmente complicado en su estudio, ya que trata del significado de las estructuras en el lenguaje, significado que incluso para nosotros muchas veces es oscuro.

2.2.3 NIVELES DE DESCRIPCIÓN Y TRATAMIENTOS LINGÜÍSTICOS

Depende del tipo de aplicación el que deban utilizarse más o menos niveles de descripción y tratamiento lingüístico. En alguno de los niveles hay amplia coincidencia mientras que otros producen cierta controversia según el autor consultado.

Se suelen presentar los niveles de descripción en forma estratificada, comenzando por los más próximos a la realización superficial (voz, frases escritas) y acabando por los más próximos a las capacidades cognitivas de quien produce el lenguaje.

Veamos muy brevemente una posible clasificación más completa que la tradicional anteriormente citada:

- **Nivel fonético:** Trata de los sonidos en el ámbito de sus características físicas (frecuencia, intensidad, modulación, etc.).
- **Nivel fonológico:** Trata de los sonidos desde el punto de vista de la realización de las palabras en forma de voz. Sus unidades son los fonemas.
- **Nivel léxico:** Trata de las palabras en cuanto depositarias de unidades de significado. Sus unidades serían los lexemas. A este nivel se plantean los problemas de segmentación del texto en palabras, la diferencia entre palabra ortográfica y lexema, las lexías, etc.
- **Nivel morfológico:** Trata de la formación de las palabras a partir de fenómenos como la flexión, la derivación o la composición. Sus unidades son los morfemas.
- **Nivel sintáctico:** Trata de la forma en que las palabras se agrupan para formar frases. Las relaciones estructurales (sintácticas) entre los componentes de la frase son variadas. En los formalismos de tipo sintagmático la estructura

sintáctica refleja la propia estructura de constituyentes de la frase (que se denominan sintagmas). Esta estructura se corresponde con el árbol de análisis. Los sintagmas terminales (las hojas del árbol) se corresponden con los elementos de los niveles inferiores de la descripción lingüística de la oración (morfemas, palabras, cadenas de palabras, etc.).

- **Nivel lógico:** Trata el significado literal de la frase (sin tener en cuenta el contexto). A este nivel se introduce el concepto de forma lógica. Las unidades dependen del formalismo lógico empleado -predicados, funciones, constantes, variables, conectivos lógicos, cuantificadores, etc.
- **Nivel semántico:** Se dota a la forma lógica de una interpretación semántica, es decir, se conectan los elementos de la forma lógica con los elementos del mundo sobre el que la aplicación informática debe trabajar.
- **Nivel pragmático:** Trata de la forma en que las oraciones se usan (y se interpretan) dentro del contexto en el que se expresan.
- **Nivel ilocutivo:** Trata de las intenciones que persigue quien produce la frase. Se pueden estudiar los actos de habla directos e indirectos. Otros fenómenos ligados a los objetivos, intenciones, planificación de los diálogos, etc. también se explican a este nivel.

La figura 3 nos permite tener la visión en niveles que se plantea en el párrafo anterior.

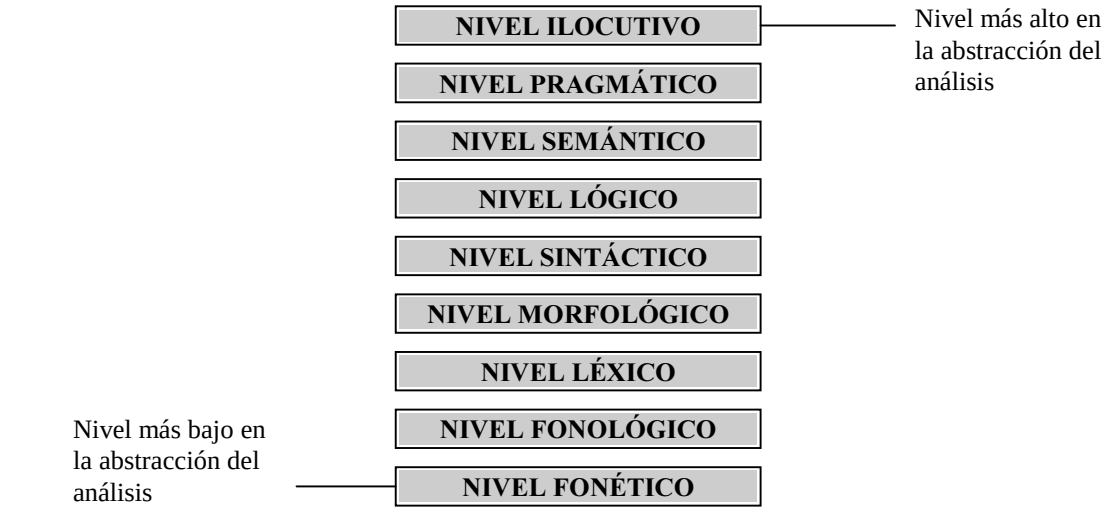


Figura 3. Pila de tratamiento lingüístico.

2.3 APLICACIONES DEL ENTENDIMIENTO DEL LENGUAJE NATURAL

El administrativo, el director y el científico rodeados de archivadores repletos de documentos y calculando con lápiz y papel son anacronismos del pasado en la era actual, llamada era de la informática.

Cada vez es mayor el número de individuos que descubre que los ordenadores pueden hacer los cómputos con mayor rapidez y libres de errores, al mismo tiempo que los libera de pesadas y aburridas cargas de trabajo.

Este es el perfil bueno de la informática. Por el contrario, también existe uno no tan bueno. Este inconveniente tiene que ver con el hecho de que los hombres hablan el lenguaje natural, medio que utilizan exclusivamente las personas de todo el mundo para comunicarse entre sí. Por contra, los ordenadores hablan un lenguaje formal y críptico, que casi todo el mundo encuentra difícil de entender y utilizar.

La capacidad de conversar en lenguaje natural con el ordenador hace de éste un elemento mucho más accesible a los no profesionales de la informática. Los lenguajes informáticos formales (C, PASCAL, FORTRAN, ...) son ideales para expresar algoritmos y estructuras de datos en forma fácilmente comprensible por un sistema informático, mas son lenguajes complejos y muy estructurados; en consecuencia son difíciles de comprender por los no profesionales de la informática.

Lo que se persigue es compaginar la potencia de cálculo de los ordenadores con la facilidad de manejo, ¿y qué forma más fácil de manejarlos que haciéndolo como si se tratase de un semejante superdotado hablando con él en nuestro lenguaje? La idea [Rauch-Hindin, 1989] es la que acuñó Boeing Computer Services al llamar a los anteriormente mencionados “no profesionales de la informática” usuarios de artefactos, ya que quieren utilizar la potencia y versatilidad de los ordenadores del mismo modo que se valen del automóvil, la radio, la televisión, la nevera, o cualquier otra máquina de nuestra civilización -es decir, como un medio de realizar su trabajo sin tener que aprender el funcionamiento interno, la teoría subyacente, o en nuestro caso, los lenguajes especiales de los ordenadores-. Este tipo de usuario es tan abundante que los fabricantes compiten duramente para hacer que los sistemas informáticos capaces de comunicarse con el usuario, en lenguaje natural, sean una realidad palpable.

En este apartado se verán algunos de las aplicaciones que los fabricantes han derivado como fruto de la investigación en el tema.

2.3.1 INTERFACES DE LENGUAJE NATURAL

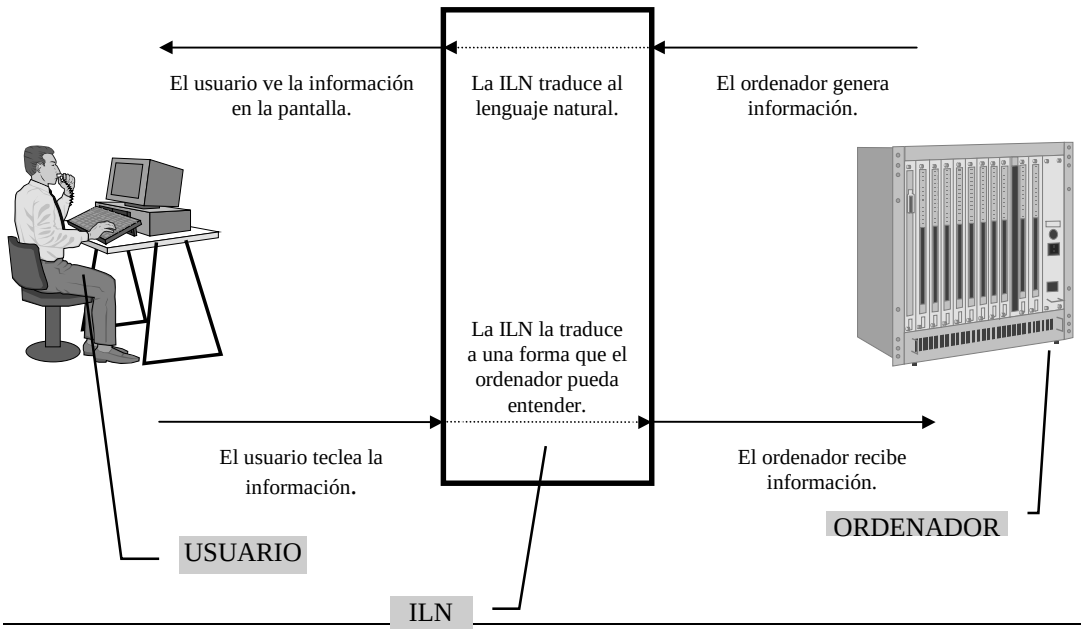


Figura 4. Comunicación con un ordenador vía una interfaz de lenguaje natural (ILN).

La mayor parte de la investigación que se ha efectuado en el entendimiento del lenguaje natural tiene que ver con el desarrollo de interfaces de lenguaje natural (ILN), las cuales permiten comunicarnos con un ordenador en lenguaje humano. También se les suele llamar terminales de lenguaje natural, ya que tienen posibilidades de realizar el entendimiento y la generación de lenguaje natural, permitiendo al ordenador entender lo que se escribe en el teclado y visualizar en pantalla un texto que sea comprensible para el usuario, respectivamente. La ILN se encuentra entre el usuario y el ordenador como se puede apreciar en la figura 4. En esta figura se puede observar claramente como el usuario no se comunica directamente con el ordenador, sino que la ILN traduce y transmite la información comunicada por el usuario al ordenador, o por éste al usuario.

Uno de los principales usos de una ILN es recuperar la información almacenada en una base de datos situada en un ordenador [Luger y Stubblefield, 1993]. Hace algunos años sólo los profesionales de la informática usaban los sistemas de gestión de bases de datos para gestionar y acceder a los datos almacenados en los ordenadores. Sin embargo la naturaleza humana se adapta rápidamente a los nuevos desarrollos. En consecuencia, de forma casi inmediata, otros profesionales quisieron manejar y acceder a sus datos a través de la computadora.

En la actualidad casi la totalidad de los sistemas de lenguaje natural son interfaces con bases de datos. Permiten que el usuario emita peticiones en forma de frases relativamente aleatorias para solicitar información a la base de datos. Si no se tiene una ILN para obtener la información almacenada en una base de datos, se tienen que usar un conjunto de órdenes o comandos (SQL[6] en el caso de ser una base de datos relacional) y estructurar la petición en un formato técnico muy preciso.

El hecho de que los sistemas de lenguaje natural se hayan propuesto fundamentalmente para actuar de interfaces con las bases de datos no deja de ser una forma de atacar el problema. En una gran parte de los sistemas informáticos las bases de datos son un elemento destacable, de tal forma que si permitimos actuar sobre ellas fácilmente, con lenguaje natural, nuestro trabajo interesará a la mayoría de las personas y su salida al mercado será más factible. Esto es debido a que se aumenta la productividad, puesto que su facilidad de manejo es evidente, y no existe aprendizaje para adecuarse al nuevo sistema informático.

Esta manera de abordar el problema permite que el trabajo realizado no se quede en un mero proyecto de investigación que nace y muere en los tubos de ensayo de las instituciones de investigación, sino que será un trabajo con una salida en el mundo real inmediata y con muchos ámbitos de aplicabilidad; por citar algunos:

- Interfaces con bases de datos.
- Sistemas de programación en lenguaje natural.
- Sistemas inteligentes de conversación.
- Sistemas avanzados de aprendizaje.

2.3.2 ENTENDIMIENTO DE TEXTOS

[6] *Structured Query Language.*

Actualmente se encuentra en investigación otro de los usos del entendimiento del lenguaje natural: la comprensión de textos impresos. Se han desarrollado sistemas de visión por ordenador que pueden interpretar con exactitud cartas y palabras impresas en diferentes tipos de letra; pero evidentemente existe una gran diferencia entre reconocer texto y comprenderlo.

Hay ya algunos programas de entendimiento de texto que pueden leerlo y analizarlo en dominios limitados, resumiéndolo y hasta respondiendo a preguntas sobre él. Por ejemplo el grupo de IA de Yale que dirige Roger Schank [Schank, 1984 a] [Schank, 1984 b] ha desarrollado programas que comprenden artículos de periódicos sobre temas tan distintos como accidentes de automóviles o política internacional.

Aunque normalmente los programas de entendimiento de textos no tienen, en la actualidad, una gran utilidad presentan un gran futuro. Por ejemplo, un programa que pueda examinar automáticamente publicaciones en busca de un tema determinado y que al mismo tiempo haga un resumen de ello será de gran valor para un hombre de negocios que siempre se encuentre muy ocupado. El Departamento de Defensa de los Estados Unidos también se encuentra muy interesado en encontrar y resumir artículos significativos que aparezcan en los medios de comunicación.

El PLN es el procesamiento del lenguaje en todas sus formas y a todos sus niveles según Derek Partridge [Partridge, 1991]. El PLN así definido incluye el entendimiento del lenguaje natural así como la lingüística computacional.

El PLN puede ser el último logro de esta rama de la IA, pero está lejos de describir cualquier éxito actual logrado en este campo.

En el área del PLN encontramos interfaces de lenguaje natural, quizá la parte más comercial, junto con sistemas tutoriales, entendedores de historias y una serie de sistemas que se pueden comunicar con el hombre.

Hay una cualitativa diferencia entre las capacidades lingüísticas de un sistema de PLN y los humanos. Sin embargo, los humanos, frecuentemente, entienden de forma muy pobre el lenguaje que se está utilizando. Pensemos en un diálogo entre dos personas que hablan en un idioma que no es el suyo propio y que no dominan a la perfección. En este diálogo habrá muchas interpretaciones y significados no comprendidos, pero no se puede decir que no es un diálogo en lenguaje natural. No se puede trazar una línea separadora del uso natural y no natural del lenguaje, pero también es cierto que los actuales sistemas de PLN están algo lejos de situarse en una zona borrosa.

2.4 ANÁLISIS LÉXICO

El componente léxico es fundamental en cualquier sistema de PLN. En última instancia las frases que el sistema debe comprender están formadas por palabras y en ellas se deposita la información (morfológica, sintáctica, semántica) que se precisará en procesos posteriores.

El tratamiento léxico no es sencillo, a pesar de lo que en una primera aproximación pudiera parecer (de hecho, y en esto el PLN no se diferencia de otras disciplinas, construir un pequeño lexicón para una aplicación “de juguete” es trivial, construir uno real para una aplicación real no lo es). La razón de esta situación es doble:

- Por una parte tenemos los problemas intrínsecos al propio léxico: segmentación (e identificación) de las palabras, homonimia y polisemia

(multiplicidad de acepciones), desplazamientos de significado (por ejemplo, los usos metafóricos de las palabras), lexías, locuciones, etc.
- Por otra parte los problemas ligados al volumen de la información léxica necesaria: representación, compacidad, redundancia, acceso eficiente, adquisición, etc.

Dividiremos esta sección en tres subsecciones que tratarán, respectivamente, de la información léxica, los diccionarios (principales depositarios de dicha información) y la morfología.

2.4.1 DESCRIPCIÓN DE LA INFORMACIÓN Y EL CONOCIMIENTO LÉXICO

Si la función del nivel léxico en la descripción lingüística de una oración (de una secuencia de palabras) es la de asignar información a las unidades elementales de la frase, la fuente de conocimiento básica será el lexicón.

Un lexicón es, simplemente, un repositorio de información léxica. El modelo de representación más sencillo sería una tabla cuyo dominio estaría constituido por las unidades léxicas (lexemas) y cuyo rango consistiría en la información necesaria asociada a cada entrada léxica.

El lexicón actúa en:

- La asignación de la categoría sintáctica.
- La asignación de propiedades de diverso tipo: morfológico, sintáctico y semántico.
- La invocación del análisis semántico de nivel léxico (el asociado a la semántica léxica).
- Desplegado de la flexión, composición y derivación morfológicas.

2.4.1.1 LA SEGMENTACIÓN DE LA ORACIÓN

El proceso de análisis léxico comienza por la segmentación del texto en unidades (palabras). Si identificáramos las palabras ortográficas con las unidades léxicas el proceso no plantearía dificultad alguna: Podemos definir, simplemente, la palabra ortográfica como una secuencia de caracteres delimitada por espacios o signos de puntuación. Construir un segmentador en este supuesto, es, pues, trivial.

Ahora bien, esta simplificación no es aceptable: Existen, por una parte, palabras gramaticales que se realizan ortográficamente en más de una palabra (por ejemplo, “sin embargo”, “no obstante”) y, por otra parte, existen palabras ortográficas que contienen más de una palabra gramatical (por ejemplo, “del” = “de él”, “dámelo” = “da me lo”).

Se podría entrar en la discusión de si “amó”, “amar”, “amaría” son palabras diferentes o formas léxicas diferentes que corresponden a la misma palabra (“lema” o “lexema”), “amar”.

Se plantea el problema de la polisemia (palabras con varios significados), el de la homonimia (palabras diferentes con la misma grafía): “separado” puede ser nombre, adjetivo o participio, “banco”, nombre, admite varias acepciones (mueble, entidad bancaria, agrupación de peces), etc.

Una aproximación aceptable es la de identificar el lexema con la unidad de información léxica no sintáctica (es decir, básicamente semántica) en tanto que los

posibles afijos, que modificarían al lexema para construir la forma léxica, aportarían el resto de la información. La figura 5 nos muestra el proceso con la palabra "reconstrucciones".

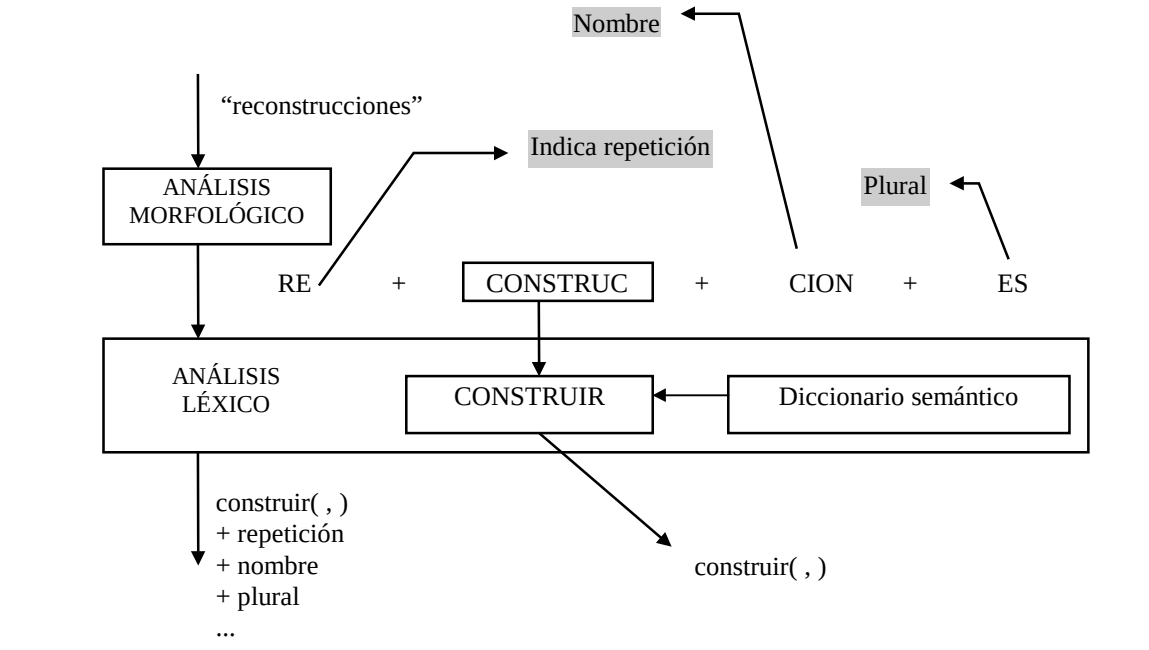


Figura 5. Análisis léxico de "reconstrucciones".

2.4.1.2 EL CONTENIDO DE LA INFORMACIÓN LÉXICA

¿Qué cabe incorporar a las unidades léxicas como resultado del análisis léxico y, por lo tanto, qué información debe incluirse en el lexicón?

Ante la anterior pregunta las respuestas surgidas son muchas, pero podrían agruparse de la siguiente forma:

- Categorización Sintáctica: Se trata de un etiquetado dependiente del tipo de formalismo sintáctico utilizado. Existen categorías cerradas (preposición, determinante, etc.) y otras abiertas (nombre, adjetivo, etc.). A veces es importante subcategorizar (nombre común y nombre propio).
- Propiedades sintácticas de concordancia, como el género, el número, la persona, el caso y el tiempo verbal, entre otras posibilidades.
- Otras propiedades sintácticas, como las restricciones selectivas (tipo de argumentos que una palabra, normalmente un verbo, admite), el tipo de complementos que una palabra rige, las preposiciones que acepta, etc.
- La información morfológica, como el patrón de formación de la palabra.
- La información semántica, como la categoría semántica, la forma lógica asociada y los rasgos semánticos, por ejemplo.

Por supuesto, no todas las combinaciones, ni el tipo de información, ni los valores posibles, son admisibles. Si, por ejemplo, la categoría es un verbo, la concordancia no incluirá seguramente el género (excepto para algunos tiempos); si la categoría es una preposición, no existirán restricciones selectivas, y así sucesivamente.

Pustejowsky, por ejemplo, sugiere para los verbos la siguiente información:

- Aridad: Número de argumentos indicativos de agente, objeto, etc.
- Argumentos internos y externos.
- Opcionalidad de cada uno de ellos.
- Restricciones selectivas sobre los argumentos:
    1. Sintácticas (por ejemplo: preposiciones regidas).
    2. Semánticas (características exigibles al argumento: si impone algo animado, si se refiere a humanos, si es contable, medible, etc.)
- Casos de los argumentos.
- Tipo aspectual del verbo (suceso, estado, proceso).
- Categoría (tipo) de la información.

2.4.1.3 ORGANIZACIÓN DEL LEXICÓN

Todos los investigadores coinciden en que conviene incorporar a los lexicones mecanismos para:

- Reducir la redundancia.
- Capturar y expresar generalizaciones.
- Realizar con facilidad la interfaz con las aplicaciones.

Veamos, brevemente, alguno de estos mecanismos:

- **Convenciones abreviadoras**

Suponen el uso de símbolos que se expanden, a la manera de las macros de algunos lenguajes de programación, permitiendo una descripción más concisa.

La mayoría de los lexicones admiten mecanismos de este estilo. Podemos citar, por ejemplo, los *clusters* de rasgos del LSP de Sager, los "alias" de Gazdar en GPSG o las "templetas" de Shieber en PATR-II. Veamos un ejemplo de estas últimas:

let nombre-masc-sing **be**:
<sin cat> = n
<sin concordancia gen> = masculino
<sin concordancia num> = singular
<sin concordancia persona> = 3.

Con esta declaración previa la definición de "pepe" se reduciría a:

word pepe:
nombre-masc-sing
<sem> = pepe.

- **Herencia**

Diferentes mecanismos de herencia se han venido utilizando en la construcción de lexicones: herencia simple, herencia múltiple, herencia monotónica, herencia por omisión e incluso combinaciones entre ellas o formas más complejas de herencia.

Shieber, en PATR-II, permite la herencia simple de propiedades:

let verbo **be**:
<sin cat> = v
<sin suj cat> = gn
<sin suj caso> = nominativo.

let vt **be**:
verbo
<sin objl cat> = gn
<sin objl caso> = acusativo.

let vdat **be:**
vt
<sin obj2 cat> = gprep
<sin obj2 prep> = a.

De forma que “vdat” heredaría las propiedades de “vt”, que, a su vez, lo habría hecho de “verbo”.

Haciendo uso de estas declaraciones previas podríamos definir, en forma más concisa, las entradas léxicas de tipo verbal:

word reir:
verbo
<sem pred> = reir
<sem argl> = humano.
(alguien ríe)

word dar:
vdat
<sem pred> = dar
<sem argl> = humano
<sem arg2> = cosa
<sem arg3> = humano.
(alguien da algo a alguien).

- **Transformaciones Léxicas**

Se trata de reglas que permiten derivar entradas léxicas no presentes explícitamente en el lexicón a partir de las sí existentes, mediante la aplicación de determinados procedimientos (normalmente reglas de producción). La aplicación de la regla permite modificar parte del contenido de la entrada léxica fuente para incorporarlo al resultado.

La utilización de reglas léxicas permite obviar la presencia de un módulo específico de análisis morfológico para las lenguas (como el inglés) en las que la flexión o derivación morfológicas no plantean problemas graves.

Son numerosos los sistemas de tratamiento léxico que incorporan reglas léxicas (Kaplan, Bresnan, Ritchie, Jackendoff, Copestake, etc.).

- **Mecanismos ligados al formalismo sintáctico utilizado**

Dependiendo del formalismo sintáctico utilizado se pueden incorporar al tratamiento léxico mecanismos de simplificación específicos. Suele tratarse de mecanismos ligados a formalismos sintácticos fuertemente lexicalistas.

- **Otros mecanismos**

Existen, finalmente, mecanismos ligados a la propia aplicación informática de la que el sistema de PLN es componente.

Lexicones Computacionales

Toda aplicación de PLN incorpora el uso de algún lexicón. En la mayoría de los casos el lexicón se desarrolla expresamente para la aplicación que lo utiliza. Se han desarrollado, sin embargo, algunos lexicones computacionales de uso general. Entre otros podemos citar el BBN-CFG de R. Ingria, el IRUS de M. Bates o el incorporado al programa ALVEY. Uno de los más conocidos es debido a Ritchie. El sistema de Ritchie se basa en la existencia de un lexicón básico y una colección de reglas. El efecto de la aplicación de las reglas es el de combinar la información que el lexicón asocia a la entrada léxica con la aportada por las reglas aplicadas. El efecto es incremental de forma que las sucesivas reglas que se aplican con éxito van enriqueciendo la información asociada a la entrada.

Lexicones Frasales

No es corriente el uso de lexicones cuya entrada sean frases, es decir, cadenas de palabras, y no palabras aisladas. La razón de ello es que aunque el empleo de tal tipo de lexicones resolvería alguno de los problemas planteados por algunas lexías o construcciones léxicas que implican a más de una palabra ortográfica, dejaría otros sin resolver, por ejemplo, la inclusión dentro de la lexía de palabras ajenas a ella o la existencia de flexión interna en la misma. Por otra parte, el empleo de cadenas de palabras puede complicar la forma de representación y acceso al lexicón.

No obstante podemos encontrar algunos ejemplos de uso de lexicones frasales. Carbonell-Hayes constituyen un ejemplo notable.

La adquisición del Conocimiento Léxico

Uno de los puntos clave del tratamiento léxico es el problema de su adquisición. Se han seguido básicamente dos enfoques para abordar este problema: la adquisición manual y la adquisición (semi) automática.

La adquisición manual se basa en la existencia de entornos interactivos de ayuda al lexicógrafo que permiten a éste incorporar y estructurar la información léxica. Se trata

del sistema más utilizado ya que posibilita, en forma relativamente sencilla, crear lexicones de tamaño medio/bajo, suficientes para muchas aplicaciones.

La alternativa es la utilización de recursos léxicos ya existentes (diccionarios para uso humano, corpus textuales, etc.) para extraer de ellos de forma automática la información léxica que se precisa.

2.4.2 IMPLEMENTACIONES DE LOS DICCIONARIOS

Un problema importante, en relación con el tratamiento léxico, es el de la implementación de los diccionarios o lexicones. En cuanto el volumen se hace importante las exigencias de eficiencia en el acceso obligan a una implementación cuidadosa.

Es importante precisar las características funcionales y operativas en cada caso: volumen, acceso sólo para consultas o también para actualización, diccionario de palabras o de lexemas (o incluso de frases), acceso incremental o global, tipo de información asociada, mecanismos de expansión presentes (herencia, reglas, morfología, etc.)...

En cuanto a organizaciones en memoria, se han utilizado listas (para lexicones "de juguete") de varios tipos, árboles, tablas y árboles de búsqueda entre otros mecanismos. Especial interés tiene la utilización de árboles binarios con búsqueda digital.

En lo que se refiere a implementaciones en disco, se ha utilizado toda la gama de ficheros indexados o bases de datos.

Un punto importante se refiere a la implementación del rango, es decir, de las propiedades o de la información correspondiente a cada entrada. Suelen utilizarse notaciones externas, de tipo simbólico, accesibles al usuario, y representaciones internas, más compactas. Se han utilizado listas, árboles de propiedades y representaciones compactas del tipo de los mapas de bits.

2.4.3 MORFOLOGÍA

Se ha prestado poca importancia a la morfología en el PLN debido, básicamente, a la poca complejidad de los problemas morfológicos en la lengua inglesa. Otras lenguas, sin embargo, con gran capacidad aglutinante, flexiva o derivativa, presentan graves problemas en cuanto a la identificación de las formas léxicas y por ello es importante el empleo de analizadores morfológicos.

El tratamiento morfológico puede afectar a la adquisición del conocimiento léxico, al almacenamiento del mismo, o a ambos. El tratamiento morfológico se basa en los mecanismos de formación de las palabras. En cierto modo un analizador morfológico trata de realizar el proceso inverso al que condujo a la formación de la palabra ortográfica que se examina.

Las palabras pueden formarse por concatenación o composición de formas más simples ("rascacielos") o por adjunción. La adjunción supone que a una raíz se le adjunten uno o varios afijos. Los afijos, de acuerdo con el punto de adjunción, pueden ser prefijos, infijos o sufijos. Estos últimos pueden ser flexivos o derivativos. Así por

ejemplo, “adormecedoras” presenta la adjunción del prefijo “a”, de la raíz “dorm”, del sufijo derivativo “ecedor” y de los sufijos flexivos “a” y “s”.

No todos estos fenómenos tienen la misma importancia en la formación de las palabras en las distintas lenguas y, por lo tanto, no todos se tratan en un analizador morfológico concreto. En castellano, por ejemplo, es bastante complicada la casuística de la flexión verbal y la de los derivados (nominalizaciones verbales, por ejemplo), mientras que no tiene demasiada importancia la composición.

El punto básico en un analizador morfológico es la obtención de la descomposición (*pattern*) de la palabra en una cadena de morfemas. A menudo se trata, además, de obtener el lexema asociado a la forma léxica para, a través de él, acceder a la información semántica. Existen dos tipos básicos de analizadores morfológicos:

Los **analizadores de dos niveles** (Ritchie y Koskeniemi han trabajado mucho en este tema) funcionan como transductores (normalmente de estados finitos) de forma que existe un nivel de entrada (superficial) y otro de salida (léxico). La entrada correspondería a la palabra que se analiza y la salida al lexema correspondiente. Las reglas morfológicas establecen las condiciones sobre las letras que se van consumiendo y las que se van produciendo.

Los **analizadores de un solo nivel** trabajan solamente a nivel superficial. Las reglas, en este caso, establecen condiciones válidas de concatenación entre los diferentes morfemas. Existen abundantes ejemplos que utilizan técnicas muy variadas. MARS, por ejemplo, utiliza una red de transición para definir las transiciones posibles. SIPA utiliza autómatas de estados finitos con condiciones ligadas a las transiciones entre estados. Ben Hamadou, por su parte, hace uso de matrices para definir las combinaciones válidas de afijos.

El trabajo realizado en GALENA se puede enmarcar dentro de los analizadores en dos niveles compilados. El calificador de “compilado” se debe a que las mencionadas reglas no se buscan en el momento del análisis léxico, como es lo habitual, sino que realmente se encuentran embebidas en el código, consideradas ya en los casos adecuados.

Un sistema interesante es SEGWORD. En este sistema, de un solo nivel, se permite no sólo la concatenación de afijos sino también la eliminación de afijos de la forma léxica. El resultado es que el sistema no consta de raíz y afijos concatenados a ella, sino de forma base y de afijos concatenados o eliminados de la misma. El interés reside en que entonces es posible sustituir el posible diccionario de raíces por un diccionario de formas base (o formas canónicas) que puede ser un diccionario de uso humano.

2.5 LA DIMENSIÓN SINTÁCTICA

El análisis sintáctico constituye, para muchos autores, el núcleo de cualquier sistema de PLN. El objetivo del analizador sintáctico, en contra de lo que generalmente se cree, es doble:

- Determinar si una frase es correcta (es gramatical).

- Proporcionar una estructura de la frase que refleje sus relaciones sintácticas y que pueda ser usada como base de los tratamientos posteriores.

En general, lo que pretende el análisis sintáctico es determinar si una frase pertenece o no al lenguaje que se trata de analizar. Si los elementos de la frase son palabras, podemos decir que una frase w está constituida por una cadena de palabras, es decir, $w \in Vt^*$, donde Vt nota al vocabulario terminal de la gramática o lo que es lo mismo, al conjunto de palabras válidas. Por supuesto, no toda concatenación de elementos de Vt se puede considerar como una frase correcta (es decir, perteneciente al lenguaje). En general tendremos que el lenguaje es un subconjunto estricto de las cadenas posibles, $L \subset Vt^*$. La manera de definir el lenguaje L puede ser a través de una lista de frases correctas o a través de una gramática.

2.5.1 LAS REDES DE TRANSICIÓN

Los primeros procedimientos que se emplearon para realizar el análisis (o mejor dicho, el reconocimiento) sintáctico fueron las redes de transición, o autómatas de estados finitos también llamadas. Consideremos la red de la figura 6 a modo de ejemplo. La red consta de una serie de nodos y una serie de arcos. El nodo origen (se pueden admitir varios) aparece señalado con una flecha, mientras que los nodos finales aparecen señalados con un doble círculo. Los nodos representan estados y los arcos transiciones entre estados. Los arcos están etiquetados con nombres de categorías (elementos del vocabulario terminal de la gramática).

Cada transición es aceptable si la palabra de la cadena de entrada tiene una categoría igual a la etiqueta del arco. El tránsito de un estado a otro hace que paralelamente se consuma una palabra en la cadena a analizar (es frecuente utilizar el símil de una cadena de palabras y una ventana abierta sobre la palabra actual que se desplaza al realizarse la transición). El proceso de reconocimiento comienza situándose la ventana sobre la primera palabra y arrancando en el estado de inicio. El proceso continúa, realizándose transiciones válidas entre estados y desplazándose paralelamente la ventana sobre la cadena de entrada. Si al consumir completamente la cadena a analizar estamos en un estado final, entonces la frase es correcta. Si no fuera así, o bien si en el curso del recorrido nos encontramos en una situación en que ninguna transición es posible, entonces la frase es incorrecta.

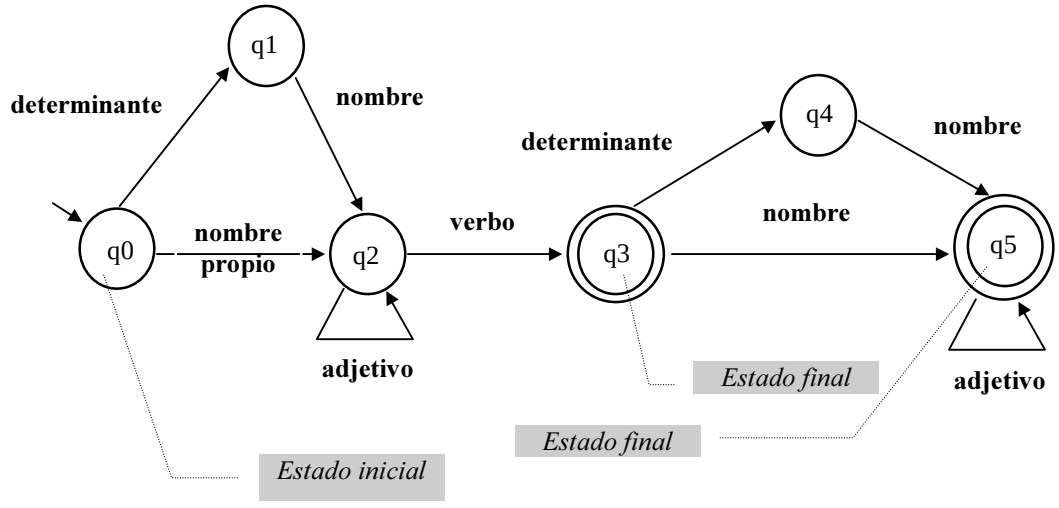


Figura 6. Ejemplo de red de transición.

En este formalismo podemos admitir el no-determinismo, bien porque exista más de un estado de inicio, bien porque de un estado salgan varios arcos con la misma etiqueta o bien porque alguna de las palabras de la cadena de entrada sea ambigua (tenga más de una categoría posible de forma aislada). En este caso el algoritmo de análisis debería incluir un sencillo mecanismo de retroceso[7] que gestionara el no-determinismo.

Es fácil comprobar que la capacidad expresiva de las redes de transición no va mucho más allá de lo conseguido mediante analizadores de *pattern-matching* u otras técnicas.

El siguiente hito lo supusieron las redes de transición recursivas [Gevarter, 1987] [Barr y Feigenbaum, 1989]. Una red de transición recursiva consiste en una colección de redes de transición etiquetadas con un nombre. Los arcos de cualquiera de las redes de transición pueden estar etiquetados con categorías (como en el caso precedente) o con identificadores de las redes de transición. El procedimiento de reconocimiento se complica algo. Cuando la etiqueta del arco es de tipo categoría se procede como hasta ahora; cuando es de tipo redes de transición entonces se continúa el reconocimiento situándose en el estado de inicio de la red referenciada. Cuando se alcanza un estado final, si la red que estamos recorriendo no era la activada inicialmente, lo que se hace es continuar el recorrido en el estado destino del arco cuya transición motivó la llamada. En el fondo la transición es una llamada recursiva a una nueva red de transición (de ahí la denominación) y el llegar a un estado final implica el final de la llamada recursiva y la continuación dentro de la red de transición llamadora. El mecanismo de retroceso también se complica algo ya que se ha de gestionar una pila de estados de retorno o dejar que la recursividad se ocupe de ello, ya que evidentemente se pueden producir llamadas recursivas cruzadas. Como ejemplo se puede indicar que un grupo nominal puede tener un grupo preposicional que a su vez posee otro grupo nominal.

Las redes de transición recursivas constituyeron una herramienta seria de PLN pero tenían varias limitaciones importantes. La primera era que al hacer depender la posibilidad de una transición exclusivamente de la categoría esperada el sistema era en exceso “local” y se dejaba de utilizar información no categorial. Por otra parte, las redes de transición recursivas eran reconocedoras y no analizadoras del lenguaje.

La siguiente extensión la constituyeron las redes de transición aumentadas, propuestas por Woods en 1970. Las redes de transición aumentadas son, básicamente, redes de transición recursivas a las que se añaden operaciones en los arcos. Las operaciones son de dos tipos:

- Condiciones, que permiten filtrar la transición entre estados.
- Acciones, que permiten construir estructuras de salida y convertir, de esta forma, el reconocedor en un verdadero analizador.

Los formalismos presentados, especialmente las redes de transición aumentadas, han sido y son ampliamente utilizados en el análisis sintáctico pero todos ellos presentan graves dificultades de expresividad notacional. Incluso incorporando mecanismos más legibles para expresar las características de las redes: formas de definir las transiciones válidas, formas de nombrar estados y arcos, etc. e incluso facilitando medios

[7] *Backtracking*.

interactivos, por ejemplo, editores sintácticos y/o gráficos, difícilmente podemos considerar cómoda la forma de definir las condiciones de pertenencia al lenguaje o las estructuras de salida.

Por ello, la mayoría de los sistemas actuales de análisis sintáctico se apoyan en dos componentes diferentes y diferenciados: la gramática y el analizador, facilitando al lingüista la tarea de escribir la gramática como componente separada.

2.5.2 LOS FORMALISMOS SINTÁCTICOS: LAS GRAMÁTICAS

El concepto de gramática, o más precisamente el de gramática de estructura sintagmática, es bien conocido. Una gramática G es una tupla de 4 elementos:

$$G = <Vt, Vn, P, S>$$

donde:

- Vt es el vocabulario Terminal, es decir, el conjunto de elementos terminales de la gramática.
- Vn es el vocabulario No Terminal, es decir, el conjunto de elementos no terminales de la gramática.
- La intersección de Vt y Vn ha de ser nula.
- La unión de Vt y Vn es el Vocabulario (V) de la gramática.
- S, perteneciente a Vn, es el axioma o símbolo de inicio de la gramática.
- P es el conjunto de reglas de producción de la gramática.

La gramática de un lenguaje es un esquema que especifica las frases permitidas en el lenguaje, indicando las reglas sintácticas que combinan las palabras en frases y cláusulas bien estructuradas.

Chomsky ha tenido un gran impacto en la investigación lingüística diseñando una estructura matemática del lenguaje. Ha definido una serie de gramáticas basadas en reglas para reescribir frases en función de sus componentes. Según las características restrictivas de P propuso una clasificación ya clásica de las gramáticas en 4 tipos, denominados 0, 1, 2 y 3. Esto es lo que se conoce como Jerarquía de Chomsky:

- **Tipo 0:**

  En las cuales los elementos de P son reglas de reescritura del tipo

  $$u \rightarrow w$$

  donde “u” y ”w” pertenecen a V*.

- **Tipo 1 (gramáticas sensitivas):**

  En las cuales se establece la limitación de ser la longitud de la cadena de “u” menor o igual que la de “w”.

- **Tipo 2 (gramáticas de contexto libre):**

Establecen la limitación adicional de permitir sólo reglas del tipo

$$A \rightarrow w$$

donde "A" es un no terminal y "w", como en el caso anterior, pertenece a V* (de hecho la aceptación o no de la cadena nula a la derecha de las producciones, es decir, la aceptación de elementos opcionales, es un punto importante que conviene precisar en los diferentes formalismos).

- **Tipo 3 (gramáticas regulares):**

Son aquellas que admiten únicamente reglas del tipo:

$$A \rightarrow a$$
$$A \rightarrow aB$$

donde "A" y "B" son elementos de Vn y "a" lo es de Vt.

Se define el concepto de derivación directa ($_G\Rightarrow$), en una gramática G, como la operación de reescritura uxw $\Rightarrow$ uyw (que podemos leer como "uxw" deriva directamente a "uyw") si la cadena "uxw" se reescribe en forma "uyw" a través de la aplicación de la producción x $\rightarrow$ y, que debe pertenecer, obviamente, a P.

Definimos la relación $_G\Rightarrow^*$ como la clausura transitiva de la relación $_G\Rightarrow$. Llamaremos a esta relación derivación.

Diremos que una cadena de elementos de Vt (es decir, una frase), w, pertenece al lenguaje generado por la gramática G ($w \in L(G)$) si es posible encontrar una derivación de w a partir de S, es decir si $S \ _G\Rightarrow^* \ w$.

Se puede demostrar que las redes de transición recursivas son equivalentes a las gramáticas de contexto libre, es decir que a cada red de transición recursiva le podemos asociar una gramática de contexto libre y viceversa, de forma que generen -reconozcan- el mismo lenguaje.

Es preciso establecer un compromiso entre las capacidades expresivas de cada tipo de gramática, las necesidades de expresión necesarias para el tratamiento del lenguaje natural y la existencia de mecanismos computacionales que hagan tratable el formalismo. En general se considera que el lenguaje natural es demasiado complejo para ser expresado mediante gramáticas de contexto libre. Sin embargo, dado que éstas son aquellas que permiten un tratamiento más eficiente (dejamos de lado las gramáticas regulares cuya cobertura es claramente insuficiente) la aproximación más extendida en el PLN, a nivel sintáctico, es la de proporcionar una gramática de contexto libre nuclear y tratar los fenómenos sensitivos en forma de añadidos locales, como por ejemplo los filtros (a menudo implementados "*ad-hoc"*).

Supongamos la siguiente gramática:

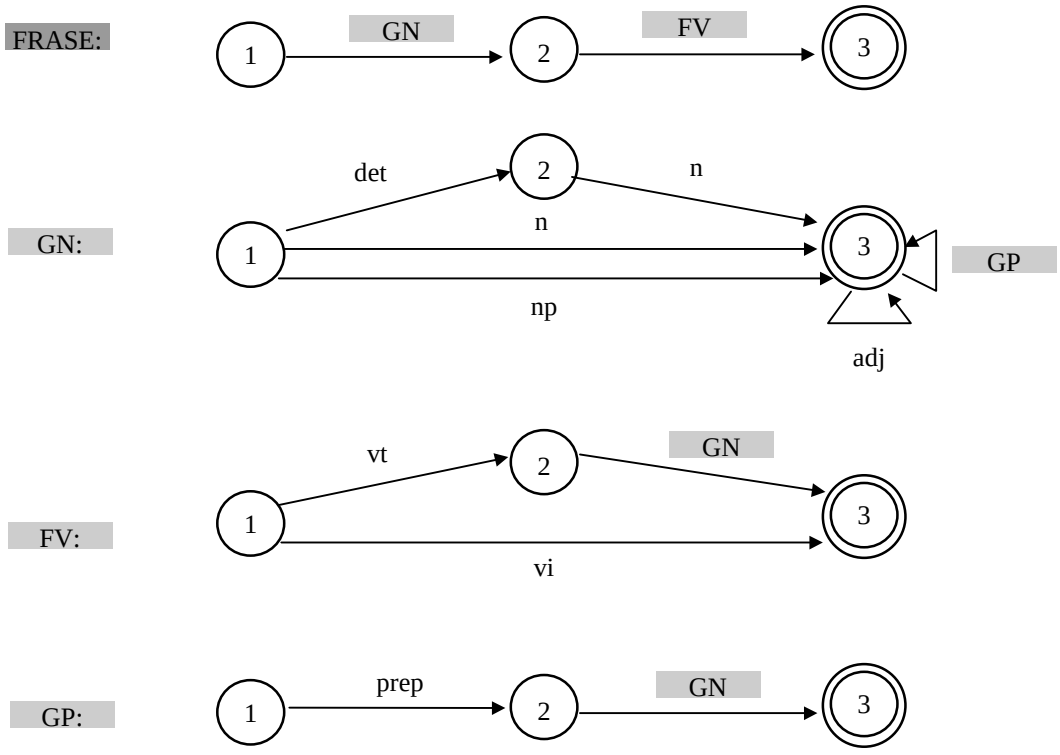
G1 = <Vt1, Vn1, P1, FRASE>

donde:

Vt1 = {det, n, np, adj, vi, vt, prep}
Vn1= {FRASE, GN, FV, RGN, GP}
P1 = {
FRASE $\rightarrow$ GN FV

GN → det n RGN
GN → n RGN
GN → np RGN
RGN → ε
RGN → GP RGN
RGN → adj RGN
FV → vi
FV → vt GN
GP → prep GN
}

La gramática G1 anterior es, evidentemente, una gramática de contexto libre que incluye categorías opcionales (RGN). Se puede comprobar fácilmente que la cobertura es la misma que la de la siguiente red de transición recursiva:



La frase “el gato come pescado” será correctamente analizada (siendo fácil hacerlo). Ahora bien, si analizamos “la gato come pescado” o “las gatas come pescado” también se llegará a la conclusión de que son frases correctas según la gramática expresada con anterioridad, cuando es evidente que son incorrectas.

¿Cabe una solución a este problema dentro del formalismo de las gramáticas de contexto libre? Por supuesto podríamos multiplicar los elementos de V incorporando a las categorías la información de género, número o persona (nmassing, nmasplu, ... para indicar, respectivamente, un nombre masculino singular o un nombre masculino plural). Deberíamos aumentar también el número de reglas de P (GN → detmassing nmassing) pero esta multiplicación se haría totalmente insoportable en cuanto el número de fenómenos a tratar (concordancia, dependencias a larga distancia, movimiento de

constituyentes, conjunción, etc.) y la información requerida para su tratamiento se incorporase a los elementos de V.

La solución sugerida, y adoptada por la mayoría, es la del enriquecimiento de las gramáticas con componentes (a menudo de tipo procedural) que traten estos temas. Con esta visión, la segunda producción de G1 podría sustituirse por la siguiente:

$$GN \rightarrow det\ n\ RGN\ \{concordancia\_1\}$$

donde concordancia_1 sería algún filtro capaz de verificar la concordancia.

2.5.3 ANALIZADORES BÁSICOS

Una vez definido un formalismo gramatical hemos de ocuparnos ahora del analizador.

Lo primero que cabe precisar es lo que esperamos del analizador. Por supuesto no nos conformaremos con una variable booleana que nos informe sobre la gramaticalidad de la frase. Existen dos resultados básicos a obtener: la estructura sintáctica y la estructura lógica o semántica básica.

La forma habitual de estructura sintáctica es el árbol de derivación, o árbol de análisis (a menudo decorado con fragmentos de estructuras semánticas). El árbol de derivación nos refleja la estructura sintagmática o de componentes de la oración que analizamos. Una alternativa son las estructuras funcionales que permiten expresar relaciones sintácticas no estrictamente ligadas a la estructura de la frase. Los modelos que representan la frase en términos de acciones, estados, situaciones y actantes, como las estructuras de casos, son un ejemplo conocido. Respecto a las estructuras semánticas producidas por el análisis sintáctico, las fórmulas lógicas o las redes conceptuales son los ejemplos más corrientes.

El siguiente punto de interés es el de la interacción sintaxis/semántica. Se han seguido básicamente las siguientes aproximaciones:

- **Ausencia de Semántica**

  Modelos exclusivamente sintácticos que producen un árbol de análisis a partir del cual actúa la aplicación informática. Muchos sistemas de traducción automática siguen este modelo.

- **Ausencia de Sintaxis**

  Modelos básicamente lexicalistas que prescinden de la sintaxis, obteniendo toda la información necesaria para el tratamiento de las palabras que forman la oración. Modelos aplicados a dominios muy restringidos siguen esta aproximación.

- **Síntesis de las dos aproximaciones anteriores**

  La categorización es sintáctico-semántica y el árbol de análisis tiene en sí mismo un etiquetado que incluye la semántica. Las gramáticas semánticas son un ejemplo relevante.

- **Actuación en secuencia**

El análisis sintáctico produce un árbol de análisis y, a partir de él, el intérprete semántico produce la estructura semántica. Se trata del modelo más sencillo. Permite una modularización del conocimiento y de los tratamientos implicados sin graves complicaciones.

- **Actuación en paralelo**

Modelos cooperativos. Existen diversas maneras de realizar la cooperación:

- Uso de interpretaciones semánticas parciales para validar algunas construcciones sintácticas, igualmente parciales. Normalmente estas interpretaciones se incorporan al árbol de análisis (en lo que se suelen llamar decoraciones del árbol) para evitar la redundancia en los cálculos.
- Uso de restricciones selectivas (información léxica de tipo semántico que determinadas componentes sintácticas deben poseer: el verbo "reír", por ejemplo, exige un sujeto de tipo humano).
- Valencia sintáctica adecuada a determinadas relaciones semánticas.

El tercer punto anterior se refiere a las fuentes de conocimiento implicadas en el análisis. La primera, por supuesto, es la gramática. No es la única sin embargo. El lexicón juega, asimismo, un papel predominante no sólo como fuente de categorización sintáctica, sino como fuente de asignación de las diferentes propiedades que pueden ser utilizadas por la "extensión" de la gramática de contexto libre.

Son igualmente frecuentes sistemas que incorporan diferentes tipos de reglas de formación al margen de las expresadas en la gramática: sistemas basados en principios, transformaciones, metarreglas, etc.

2.5.4 LA TÉCNICA DEL ANÁLISIS SINTÁCTICO

La técnica básica del análisis sintáctico es la habitual de cualquier sistema de reescritura. Se parte de un objetivo (el axioma, S, de la gramática) y de unos hechos (la frase a analizar). El objetivo del analizador es lograr una derivación que conduzca del objetivo a los hechos. Conviene mencionar en este proceso varios puntos:

- **Estrategia del Análisis**

Las estrategias básicas son la descendente (dirigida por objetivos, *top-down*, o también llamada predictiva) y la ascendente (dirigida por hechos, *bottom-up*). Ambas tienen ventajas e inconvenientes y la elección de una u otra dependerá de las características de la gramática, del grado de ambigüedad léxica y de la técnica de análisis a emplear. Existen métodos que combinan ambas estrategias, estática o dinámicamente. En ellos la dificultad estriba en dotar al

mecanismo de control del suficiente conocimiento para que la mejora de las prestaciones compense el *overhead* propio de efectuar el análisis.

- **Dirección del Análisis**

La mayoría de los analizadores operan de izquierda a derecha. Existen, sin embargo, excepciones. La más conocida es la de los analizadores activados por islas en los cuales la localización de determinadas palabras (no ambiguas, muy denotativas, etc.) activa el proceso ascendente en forma de capas concéntricas alrededor de dichas islas. Otro ejemplo notable de bidireccionalidad lo constituyen los analizadores regidos por el núcleo (*head driven*) en los cuales la activación se lleva a cabo, para cada regla, a partir del núcleo (*head*) o componente principal del sintagma correspondiente (el verbo para la frase, el nombre para el grupo nominal, etc.).

- **Orden de aplicación de las reglas**

El azar o el orden de escritura de las reglas de la gramática suele ser el criterio que se sigue al seleccionar las reglas a aplicar en cada caso. Existen, sin embargo, sistemas más sofisticados en los que las reglas se ponderan o en los que se incluyen formas de selección inteligentes.

- **La ambigüedad**

Dos tipos de ambigüedad tienen importancia durante el análisis sintáctico: la ambigüedad léxica, que hace que una palabra pueda poseer más de una categoría sintáctica, y la ambigüedad sintáctica, que hace que pueda producirse más de un árbol de análisis correcto para alguna de las componentes (incluyendo, por supuesto, el propio axioma).

Lo único que debemos señalar es que cualquier sistema de análisis sintáctico debe poseer mecanismos de gestión de estos tipos de ambigüedad. Existen sistemas que asignan probabilidades a las categorizaciones y a las reglas. De esta forma los árboles de análisis obtenidos pueden dotarse de alguna medida de probabilidad relativa y ordenarse de acuerdo ella. En la mayoría de los casos la selección del árbol adecuado se deja a procesos posteriores.

- **No-determinismo**

El análisis sintáctico presenta, como hemos visto, varias fuentes de indeterminismo. En cualquier caso, e incluso para analizadores calificados de deterministas, es necesario gestionar un mayor o menor grado de indeterminismo. Las herramientas que se emplean para ello son el retroceso y el (pseudo) paralelismo.

Como antes señalábamos, las redes de transición recursivas son equivalentes a las gramáticas de contexto libre. Una de las maneras de analizar una gramática de contexto libre es transformarla en una red de transición recursiva e implementar esta última. El procedimiento es sencillo pero el método presenta algún problema:

- El mecanismo de análisis de las redes de transición recursivas es puramente descendente. Intentar implementar estrategias ascendentes o híbridas es antinatural.
- Cualquier transformación de una gramática hace que la estructura resultante, sobre todo si se trata del árbol de análisis, refleje no la estructura de la frase de acuerdo a la gramática original (que es la escrita y prevista por el lingüista), sino de la transformada. Ello puede producir problemas no sólo de depuración de la gramática, sino a la hora de describir y llevar a cabo los procesos posteriores al análisis sintáctico. Por supuesto una alternativa es escribir directamente la red de transición recursiva, con los inconvenientes que en su momento se señalaron.

Ya indicamos anteriormente que el no-determinismo constituye el gran problema del análisis sintáctico. Se han hecho numerosos intentos de superarlo, algunos restringiendo las gramáticas utilizadas y otros proponiendo formalismos nuevos.

Si nos limitamos al ámbito de las gramáticas de contexto libre podemos intentar aplicar alguno de los métodos que la teoría de compiladores nos proporciona mediante la creación de mecanismos (normalmente tablas) que asocien a cada estado la acción (o acciones) a efectuar. Asociada a las estrategias descendentes nos encontramos en primer lugar con la familia de los analizadores LL(k), es decir, analizadores descendentes, de izquierda a derecha y con visión hacia adelante (*lookahead*) de k palabras.

Asociada a la estrategia ascendente nos encontramos con la gran familia de los analizadores LR(k). Existe abundante literatura sobre estos formalismos y las muchas subclases que encontramos dentro de ellos.

Los analizadores LL o LR se basan en la construcción de tablas, construidas a partir de la gramática, y que dirigen, en forma determinista, el análisis posterior de las oraciones. La construcción de la tabla depende sólo de la gramática, se realiza sólo cuando ésta cambia y permite, por lo tanto, implementaciones sumamente eficientes. El problema es que, desgraciadamente, no todos los lenguajes de contexto libre están incluidos en la clase de los LL(k) o LR(k), incluso para valores elevados de k.

Una alternativa muy utilizada la constituyen los analizadores denominados Tabulares. En ellos las tablas que dirigen el análisis se construyen dinámicamente en paralelo con el análisis de la frase.

Existen numerosos ejemplos. Los más conocidos son CKY (Cocke, Kasami, Younger) y Earley. Este último, sobre todo, se ha convertido en una referencia indispensable cuando de análisis sintáctico se trata. Los dos métodos tienen un coste cúbico en tiempo respecto a la longitud de la frase a analizar (en algún caso Earley puede ser cuadrático e incluso lineal). La ventaja es que son aplicables a cualquier gramática de contexto libre sin limitación.

Una alternativa que ha adquirido gran relevancia en los últimos tiempos es el analizador de M. Tomita. Se trata, básicamente, de un analizador LR(0) que admite indeterminismo. Es decir, las entradas de las tablas LR pueden ser listas de acciones en vez de acciones únicas. Tomita propone una estructura de datos bastante peculiar (una especie de grafo dirigido sin ciclos cuyos nodos terminales operan como pilas) y un tratamiento de pseudoparalelismo en los puntos de indeterminación. Los resultados son sumamente interesantes. Recientemente se han propuesto mejoras al método incorporándole *lookahead* y ponderando las reglas.

Un último sistema a mencionar es PARSIFAL. Se trata de un sistema que en su momento levantó grandes expectativas. PARSIFAL se aparta bastante de cuanto estamos describiendo. Funciona con dos estructuras básicas de datos para describir el

estado de proceso de la frase: una pila en la que se almacenan componentes aún incompletos y un buffer en el que se guardan componentes ya completados pero cuyas relaciones con otros componentes pueden estar aún sin determinar (se incluyen las palabras que forman la frase a analizar junto a la información léxica que tengan asociada).

El buffer tiene tres elementos accesibles (por eso Marcus, su creador, habla de *lookahead* a distancia tres). Las reglas de la gramática son del tipo de las reglas de producción. La condición de la regla es un patrón que se tratará de emparejar, mediante un mecanismo de *pattern-matching*, con la cima de la pila y los valores de los tres elementos accesibles del *buffer*. Las acciones implican manipulación de pila y *buffer*, activación de reglas y otras operaciones.

2.6 LA INTERPRETACIÓN SEMÁNTICA

La semántica se refiere al significado de las frases. La interpretación semántica es el proceso de extracción de dicho significado. No existe unanimidad en cuanto a los limites de la interpretación semántica. En esta presentación consideraremos que interpretación semántica se refiere a la representación del significado literal de la frase considerada ésta en forma aislada, es decir, sin tener en cuenta el contexto en que ha sido enunciada. El conseguir a partir de aquí la interpretación final de la oración será una cuestión que resolveremos a nivel de la pragmática.

Existen numerosos estudios sobre diferentes modelos semánticos, aunque ni de lejos se ha llegado en semántica al mismo grado de formalización que en sintaxis. Desde el punto de vista del PLN se suelen exigir una serie de cualidades al modelo semántico utilizado y, por lo tanto, al proceso de la interpretación. Allen, por ejemplo, propone:

- La Semántica ha de ser compositiva. Es decir, la representación semántica de un objeto debe poder realizarse a partir de la de sus componentes.
- Debe utilizarse una teoría, no basarse en una semántica "*ad-hoc*".
- Se deben utilizar objetos semánticos. Se impone definir un sistema de representación semántico.
- Debe existir un mecanismo de interfaz entre sintaxis y semántica.
- La interpretación semántica debe ser capaz de tratar fenómenos complejos como la cuantificación, la predicación en sus diversas formas, modalidades, negación, etc.
- La interpretación semántica debe ser robusta frente a las ambigüedades léxicas (polisemia) y sintácticas. El sistema de representación ha de ser no ambiguo.
- El sistema de representación debe permitir realizar inferencias (herencia, conocimiento no explícito, etc.).

En cuanto al mecanismo de la representación semántica, se han utilizado formas variadas. Quizás lo más corriente sea el uso de formas de representación basadas en lógica, especialmente formas diversas del cálculo de predicados de primer orden. También las redes semánticas han sido ampliamente utilizadas. Representaciones más profundas, como los grafos de dependencia conceptual de Schank (u otros de más alto nivel, como los MOPs o SCRIPTs) tienen también sus partidarios. Los modelos más

basados en semántica (con poca o nula incidencia de la sintaxis) encuentran especial acomodo en representaciones estructuradas, como por ejemplo las *frames*.

Es habitual que la compositividad de la semántica se realice en el ámbito sintáctico. Es decir, que la interpretación semántica de un objeto sea función de la interpretación semántica de sus componentes sintácticos. Ésta es la hipótesis que se adoptará en lo que sigue.

Si hablamos de semántica compositiva hemos de definir el nivel atómico, es decir el que no admite mayor descomposición. En la hipótesis adoptada éste no puede ser sino el nivel léxico (es decir, consideraremos al lexema como unidad de significado). En este sentido cabría asignar a las palabras, a través del lexicón, estas unidades de significado.

Consideremos, por ejemplo, un sistema de representación basado en lógica. En él parece razonable que los nombres propios se interpreten como las constantes del formalismo ("Pedro" $\rightarrow$ pedro). Los verbos intransitivos, por su parte, se podrían interpretar como predicados unarios ("ríe" $\rightarrow$ (lambda(x), reir(x))), etc.

La composición de significados de estas unidades en la frase "Pedro ríe" podría, si la función de composición fuera la aplicación lambda, dar lugar a:

(lambda(x), reir(x)) pedro $\rightarrow$ reir(pedro).

¿Cómo llevar a cabo la interpretación semántica? Hay que decidir dos factores, la función (o funciones) de composición y el momento de llevar a cabo la interpretación. Respecto a la función de composición, el enfoque más elegante sería el de admitir una sola función de composición de forma que las diferencias de comportamiento quedasen reflejadas en los parámetros de tal función y, en último extremo, en el léxico.

Si siguiéramos este enfoque no tendría demasiada importancia el momento de llevar a cabo la interpretación semántica. Si lo hacemos con posterioridad al análisis sintáctico resultará que la interpretación semántica dará lugar a un recorrido en postorden del árbol de análisis (del que ya dispondremos) llamando recursivamente al interpretador semántico, que aplicaría la misma función de interpretación sobre los componentes sintácticos del nodo (es decir, sobre sus hijos en el árbol de análisis).

El enfoque opuesto sería el de la aplicación regla a regla. Cada vez que se aplique una regla con éxito, es decir, cada vez que se construya un nuevo nodo en el árbol de análisis, podemos calcular la interpretación semántica del constituyente.

La ventaja del primer caso es que no se presentan problemas con el *backtracking* (el árbol de análisis ya ha sido construido) y, por lo tanto, las acciones del interpretador semántico serán ya definitivas. El inconveniente es que muy probablemente no tengamos un árbol de análisis sino varios y que alguno de ellos pudiera haber sido eliminado por consideraciones de tipo semántico.

El inconveniente de la aplicación regla a regla es que se puede llevar a cabo procesamiento semántico inútil (que el *backtracking* deshará). La ventaja es que la interpretación semántica constituye una forma más de restricción de la regla (además de las posibles restricciones de tipo sintáctico).

Elegir una u otra aproximación se puede basar en consideraciones sobre el coste relativo de los procesos sintáctico y semántico.

Una forma extendida de implementar la función de composición es limitarla a una lambda evaluación. En este caso, la única información semántica que se debería asociar a las reglas de la gramática sería el orden de aplicación de los componentes.

Es habitual, si se emplea la forma de interpretación regla a regla, el almacenamiento de las interpretaciones parciales como etiquetas del árbol de análisis (existencia de un atributo `sem`, por ejemplo).

Si empleamos el proceso a posteriori, lo único que necesitamos almacenar es el orden de aplicación de los argumentos de la composición.

No siempre es adecuado un sistema de interpretación como el aquí presentado. A veces conviene utilizar más de una función de composición para atenuar la rigidez del método. En esos casos la información asociada no es únicamente el orden de aplicación sino también el tipo de operación a efectuar. En cualquier caso, si el número de operaciones diferentes no es grande las mismas consideraciones hechas serían aplicables.

En todos los casos anteriores, la información básica de tipo semántico o es aportada por el léxico o se reduce a un mínimo (operación y orden de aplicación) que se debe incorporar a cada regla. Independientemente de que la interpretación semántica se realice en paralelo o al final del análisis sintáctico se produce una interacción muy fuerte sintaxis-semántica. Difícilmente podrá escribir la parte semántica de las reglas quien no haya escrito la parte sintáctica. Existe, por supuesto, la aproximación contraria: la que se basa en una descripción independiente de las reglas sintácticas y semánticas.

En esta otra aproximación el análisis sintáctico produce un árbol de análisis y sobre él se aplicará la interpretación semántica. No existe una correspondencia regla a regla.

Las reglas de interpretación semántica, en este último caso, suelen ser reglas que producen interpretaciones semánticas parciales que luego se combinarán entre sí (compositividad) para formar interpretaciones semánticas más complejas.

El mecanismo de activación y actuación de estas reglas suele ser del tipo de los sistemas de producción. Las reglas constan de una condición y una acción. La condición se aplica sobre un fragmento del árbol de análisis y cuando tiene éxito se ejecuta la acción. Ésta suele incorporar nueva información al árbol de análisis. La condición acostumbra a implicar la actuación de un mecanismo de *pattern-matching*.

Así expuesto el sistema parece sencillo. No lo es. En primer lugar, un mecanismo de *pattern-matching* sobre los nodos de un árbol (potencialmente a diferente profundidad) que suponga el examen de todos los posibles cortes del árbol para intentar asociarlos a los patrones de todas las reglas de interpretación es, a todas luces, computacionalmente impracticable.

Es necesario limitar el espacio de búsqueda en ambos sentidos: no todos los cortes y no todas las reglas. Ello obliga, en primer lugar, a establecer algún tipo de indicación de las reglas (agrupación en paquetes, acceso eficiente, posiblemente a través de multiíndices, etc.) y, en segundo lugar, a incluir algún tipo de mecanismo de control que establezca cuándo deben aplicarse qué reglas.

Ahora bien, esto no quiere decir que la interpretación semántica se deba llevar a cabo obligatoriamente después del análisis sintáctico. Podemos perfectamente interpretar semánticamente cualquier fragmento de árbol de análisis, no necesariamente el completo, una vez esté construido, y almacenar las interpretaciones parciales en el árbol. De esta manera es posible implementar una estrategia cooperativa entre sintaxis y semántica sin vernos obligados a desarrollar en paralelo las dos fuentes de conocimiento. Si usáramos esta aproximación iríamos construyendo y decorando semánticamente el árbol de análisis en forma incremental. Los mismos argumentos que se adujeron al hablar de la interpretación semántica regla a regla serían aplicables aquí.

2.7 LA PRAGMÁTICA

Si la semántica representaba el significado literal de las expresiones, la pragmática se asocia a la utilización del lenguaje en un contexto. En ocasiones a lo que aquí hemos llamado nivel semántico se le denomina nivel lógico para enfatizar que se trata de las condiciones de verdad de las expresiones. Cuando decimos "el gato come pescado" y se interpreta en forma de:

existe(X, y (gato(X), existe(Y, y (pescado(Y), come(X, Y)))))

lo que hacemos es definir como cierta esta aserción.

La pragmática trataría de interpretar la frase en su contexto de forma que produjera los efectos deseados.

El primer objetivo de la pragmática será el de establecer el marco en el que deben ser interpretadas las expresiones. No es sencillo tratar de delimitar, ni aún tratar de definir, tal marco. Que el dominio semántico en el que se enmarcan las expresiones del lenguaje natural forma parte de ese marco es evidente. Que las características de los emisores y de los eventuales receptores de dichas expresiones también se incluyen en dicho marco puede aceptarse. Que algunas formas de organización de las expresiones en lenguaje natural, como las conversaciones, suponen marcos más o menos cerrados es asimismo cierto. Pero poco más podemos precisar.

2.7.1 UN EJEMPLO CONCRETO: LA REFERENCIA

La referencia constituye el problema más interesante a nivel pragmático. Básicamente consiste en ligar los conceptos que aparecen en las expresiones que se tratan con los conceptos del mundo (real o simbólico) que manejamos.

Se deben buscar los referentes de todos los conceptos contenidos en la oración (objetos, acciones, estados, situaciones, etc.), aunque los mayores esfuerzos se dirigen a la búsqueda de referentes de los objetos, normalmente expresados en frases nominales, que sean relevantes al contexto de la conversación.

Denominaremos referencia a la relación entre descripciones conceptuales, más o menos complejas, de un objeto con la entidad del dominio que este objeto denota.

Existen dos tipos de referencia: la referencia directa y la anafórica (o simplificada).

La referencia directa implica que la descripción que se hace del objeto referido es completa (o puede ser completada mediante inferencias), mientras que la referencia anafórica implica que parte de la información, eventualmente toda, se omite.

La referencia directa puede ser definida (grupos nominales con artículos determinados, nombres propios, etc.) o indefinida (grupos nominales con artículos indefinidos). Normalmente la primera se refiere a objetos ya mencionados y la segunda a nueva información.

Suele considerarse que la primera referencia a un objeto es directa y las restantes anafóricas. La referencia anafórica puede ser oracional, si el referente está descrito en la propia frase, o interoracional, si no lo está (el caso extremo, la catáfora, es cuando el referente se introducirá con posterioridad).

Es importante la distinción entre referencia, es decir, una relación que liga conceptos lingüísticos con elementos del dominio, y correferencia, relación entre diferentes descripciones lingüísticas de objetos del dominio. Resolver una referencia implica dos fases:

1. Establecer una(s) expectativa(s) conceptual(es) restringida(s) a nivel semántico y usando información morfosintáctica.
2. Satisfacer esta(s) expectativa(s).

Consideremos los siguientes ejemplos:

- Puse el libro en la mesa. Más tarde lo cogí.
- Puse el libro en la mesa. Más tarde la limpié.

Fijémonos en que “lo” y “la” deben tener como referentes objetos, susceptibles de ser “cogidos” y “limpiados” y cuyo correferente sea un nombre masculino y femenino respectivamente.

- Dejé el libro a Pedro. Luego se lo pedí.
- Dejé el libro a Pedro. Luego le pedí otro.

“Le” admite como referente a una persona.

- Compré un gato. El animal no me dejaba dormir.
- Compré un coche. Las ruedas estaban gastadas.

En estos casos el referente debe inferirse a través de una relación (clase-subclase y todo-parte, respectivamente).

3. LA PROBLEMÁTICA DE LA AMBIGÜEDAD

3.1 INTRODUCCIÓN

Ya que el lenguaje natural es inherentemente ambiguo, la ambigüedad debe manejarse con cierta inteligencia en un sistema que trabaje en el campo del lenguaje natural. Fijémonos, para ilustrar la anterior afirmación, en la siguiente frase:

"Vi un hombre con un telescopio."

Ante la pregunta ¿quién tiene el telescopio? la respuesta puede ser tanto el sujeto de la oración como el complemento, y ninguna de las respuestas es errónea o correcta. Ejemplos similares se pueden encontrar en la vida cotidiana:

- "¿Qué escribiste?"
- "No lo sé."

La pregunta que permite mostrar el grado de ambigüedad es si escribió el texto "No lo sé." o si no se acuerda de lo que escribió.

En estos ejemplos, que son muy normales en nuestra forma de expresarnos, lo que nos impide dar una respuesta, en la mayoría de los casos, a las preguntas planteadas es la falta de contexto. El contexto es lo único que nos permitirá razonar para proporcionar una respuesta a las preguntas anteriores, permitiendo así deshacer las ambigüedades, que mayoritariamente son intrínsecas y generan discusiones bizantinas. La razón de ello es que muchas veces ni siquiera el contexto permite tener un punto de apoyo para optar por una u otra interpretación, siendo todas las posibles interpretaciones tan válidas como cualquiera.

Al contrario de lo que sucede con el lenguaje natural, los lenguajes de programación no tienen ningún problema de ambigüedad, puesto que se diseñan cuidadosamente para ser inambiguos. Es evidente que al escribir un programa, el programador debe saber exactamente lo que el programa va a hacer. No puede haber lugar a diferentes interpretaciones, esto es, programas ejecutables, para un mismo código fuente.

Las ambigüedades léxicas en los lenguajes naturales, en las que nos centraremos más adelante por ser éste el objeto de estudio, ocurren porque una misma palabra tiene significados diferentes cuando se utiliza en contextos diferentes, es decir, una palabra, por sí misma y fuera de contexto, tiene varias acepciones o significados.

Se puede definir formalmente ambigüedad como el fenómeno de falta de información, donde hay incertidumbre entre dos o más descripciones alternativas. Llevado al ámbito que nos ocupa, esas descripciones alternativas serían etiquetaciones léxicas posibles para una misma palabra. Evidentemente el objetivo perseguido es deshacer la ambigüedad indicando la alternativa por la que se opta.

3.2 PROBLEMAS EN LA COMPRENSIÓN DEL LENGUAJE NATURAL

El lenguaje natural, con el que normalmente nos comunicamos, es informal y a veces puede ser muy ambiguo, impreciso e incompleto, aunque esté escrito correctamente. Cuando a estos problemas añadimos el hecho de que a veces usamos el lenguaje incorrectamente, esto es, sin que esté totalmente de acuerdo con las reglas de la gramática y de la sintaxis, no nos deben sorprender las dificultades con las que se encuentra el ordenador para entender el lenguaje natural.

Se pueden clasificar los obstáculos que existen para entender el lenguaje natural en cuatro [Mishkoff, 1988]:

- Imprecisión.
- Inexactitud.
- Formas incompletas.
- Ambigüedad.

3.2.1 IMPRECISIÓN

Muy a menudo las personas expresan los conceptos con una terminología inexacta.

Por ejemplo, ¿cuánto tiempo es “mucho tiempo”? en las siguientes frases:

“La cosecha se ha perdido porque ha estado mucho tiempo nevando.”
“He estado en esa situación mucho tiempo.”
“Hace mucho tiempo los dinosaurios poblaban la tierra.”

El término “mucho tiempo” va creciendo en cuanto a intervalo de tiempo implicado en las sentencias anteriores, pero esto lo sabemos porque estamos familiarizados con los conceptos incluidos en estas frases. Sin embargo, el ordenador no posee esa familiaridad conceptual. No sería capaz de distinguir los intervalos de tiempo implicados en las frases anteriores.

3.2.2 INEXACTITUD

Dejando aparte los problemas inherentes al entendimiento del lenguaje natural estructurado y correcto, están también los problemas surgidos del hecho de que el lenguaje natural raramente se atiene a reglas fijas y bien definidas. Por ejemplo, podemos encontrar errores en un texto escrito como los que siguen:

- Errores de ortografía.
- Palabras traspuestas.
- Construcciones no gramaticales.
- Sintaxis incorrecta.
- Puntuación inapropiada.

Si se cometen muchas faltas el texto puede resultar del todo incomprensible, pero si el número de faltas no es demasiado elevado, el lector, o el oyente en su caso, puede obviar esos errores y ser capaz de entender el significado. Si se diseña un sistema para llegar a entender el lenguaje natural debe ser capaz de solventar las inexactitudes en la misma forma que lo hace una persona.

En definitiva, el principal problema que plantea la comprensión del lenguaje natural, es decir, el proceso por el cual la máquina es capaz de comprender lo que su interlocutor le dice (oralmente o por escrito), es precisamente su característica de natural. El lenguaje natural es el hablado por las personas y, como tal, se rebela a ser apresado en términos de una gramática.

Se han hecho esfuerzos considerables por construir gramáticas de cobertura extensa que describan fragmentos importantes de lenguas naturales (gran parte de los esfuerzos que se han llevado a cabo lo son sobre la lengua inglesa) y, aunque hay resultados notables, el lenguaje natural acaba por desbordarlos; surgen, o se descubren, nuevas construcciones sintácticas no previstas inicialmente, se utilizan palabras nuevas y aparece una retahíla de excepciones en cuanto se intenta tratar un nuevo dominio (los nombres propios y su tipología, las siglas, fórmulas o abreviaturas, las jergas, las diferencias entre lengua reglamentada y lengua real: faltas ortográficas y gramaticales, usos restringidos de la lengua, barbarismos, etc.).

3.2.3 FORMAS INCOMPLETAS

Cuando las personas hablan no siempre proporcionan todos los datos completos. Esto se debe a que comparten muchas experiencias comunes y pueden omitir muchos detalles sin que por ello se las comprenda incorrectamente, pudiéndose en este caso decir que el oyente es capaz de “leer entre líneas”.

Elaine Rich [Rich, 1983], como ejemplo, sugiere la siguiente situación:

“Anoche Juan fue a un restaurante. Pidió un filete. Cuando fue a pagar se dio cuenta de que no llevaba dinero.”

¿Se comió Juan el filete? Aunque no está expresado explícitamente, suponemos que sí lo hizo; si no, ¿por qué iba a pagarlo?

Nuestras experiencias en situaciones parecidas nos permiten entender la información que no está incluida en el texto. Por ello el ordenador deberá poseer la misma forma de experiencia sobre la situación para poder comprender la información incompleta; aunque tengamos asociado el riesgo de no entender correctamente (en el caso anterior suponemos que Juan se comió el filete, pero no es posible asegurarlo).

3.2.4 NIVELES DE AMBIGÜEDAD

Muchas de las cosas que decimos pueden ser interpretadas en más de una forma. Esta ambigüedad puede dar origen a interpretaciones erróneas entre personas y constituyen uno de los problemas básicos para programar al ordenador del que esperamos que comprenda el lenguaje natural. Algunos de los factores que contribuyen a la ambigüedad del lenguaje natural son los siguientes:

- **Múltiples significados de las palabras**

Esta situación se da a nivel léxico.

No es extraño que una palabra dada tenga más de un significado, tal es el caso de la palabra “sobre” en las siguientes frases:

1. “La carta está sobre la mesa.”
2. “Coge un sobre y ponle un sello.”
3. “¡Que no sobre nada de lo que se trajo!”

Es evidente que en las anteriores frases el rol desempeñado por la partícula “sobre” es distinto en cada una de ellas. Si el ordenador no conoce alguna característica más de la palabra, no sabrá qué significado darle en cada una de las frases. Sin embargo, si es capaz de detectar el contexto en que la palabra está embebida, sí podría elegir una alternativa.
Otros ejemplos similares son:

1. “Se sentó en el banco.”
2. “Entró en el banco y fue a la ventanilla.”
3. “El avión localizó el banco y comunicó su posición.”

¿Qué tipo de conocimiento hay que utilizar y cómo debemos utilizarlo para conjeturar que (probablemente) la aparición de “banco” en (l) se refiere a un mueble que sirve para sentarse, mientras que la aparición en (2) se refiere a una oficina en la que una entidad financiera realiza operaciones a través de una ventanilla y la aparición en (3) se refiere, una vez examinado el contexto, a un banco de pesca? La respuesta no es otra que un conocimiento contextual.

- **Ambigüedad sintáctica**

Este tipo de ambigüedad aparece, como su propio nombre da a entender, en el nivel sintáctico. Este tipo de ambigüedades surge por las peculiaridades en la sintaxis. Por ejemplo:

“Golpeé al hombre con el martillo.”

¿Cómo podemos interpretarla?, ¿cogí un martillo para golpear al hombre?, o por el contrario ¿golpeé al hombre que tenía un martillo?

“La vendedora de periódicos del barrio es muy guapa.”

¿Queremos indicar aquí que la vendedora es del barrio, o bien son los periódicos los que son del barrio?

El ordenador será incapaz de determinar el verdadero significado si no puede entender el contexto en que la frase está inmersa.

- **Ambigüedad semántica**

Este tipo de ambigüedad se da a nivel semántico y se puede ejemplificar con la siguiente frase:

"Pedro dio un pastel a los niños."

Las preguntas que muestran la ambigüedad existente son las siguientes:

¿Uno para todos?
¿Uno a cada uno?

- **Ambigüedad de referencia**

También llamada antecedentes confusos. En el lenguaje natural solemos usar pronombres en lugar de los nombres que ya hemos utilizado con anterioridad. Esto puede derivar en una ambigüedad, como en:

"Juan pegó a Pedro porque simpatizaba con Beatriz."

¿Quién simpatizaba con Beatriz? Igual que en los casos anteriores necesitamos conocer el contexto para comprender el significado real de la frase.

"Le dijo, después, que lo pusiera encima."

¿Quién dijo?, ¿a quién?, ¿cuándo?, ¿después de qué?, ¿que pusiera qué?, ¿encima de dónde?

Los ejemplos que hemos propuesto anteriormente plantean problemas que aparecen a diferentes niveles de la descripción lingüística, y cuya solución requiere conocimientos diversos, a menudo expresados en otros niveles.

3.3 TIPOS DE AMBIGÜEDAD

Se pueden distinguir cuatro vertientes distintas del término ambigüedad en el ámbito de la etiquetación morfosintáctica [EAGLES, 1994]:

1. Homonimia gramatical

Normalmente este tipo de ambigüedad, considerada como tal, no aparece en un texto comentado, puesto que la ambigüedad se resuelve para decidir la etiqueta apropiada.

Esta es la situación que se encontraba en GALENA antes de que el presente trabajo se llevase a cabo, ya que se proporcionaban todas las etiquetas posibles de una palabra, sin más.

Lógicamente, en este tipo de ambigüedad, como en todas, una palabra puede tener varias etiquetaciones posibles, y se trata de seleccionar la adecuada según el contexto donde se encuentre.

2. Etiquetas de "palabras híbridas" o etiquetas *portemanteau*

En el caso de tratar con textos de gran tamaño la etiquetación se hace automáticamente, y puede no haber necesidad u oportunidad para el retoque manual posterior de todo el corpus. Puede ser práctico, en tales casos, conservar más de una etiqueta en el corpus comentado, donde los algoritmos de etiquetación automática no tienen las suficientes evidencias para eliminar las ambigüedades. Se asume que un experto lingüista con posterioridad no tendrá problema, en general, para suprimir la ambigüedad.

3. Ambigüedades de incertidumbre humana

Otro tipo de ambigüedad puede surgir cuando el etiquetador humano no puede decidirse por la etiqueta apropiada. Puede haber buenas razones para este tipo de indecisión:

- Varios expertos tienen opiniones diferentes.
- Las categorías, por sí mismas, pueden no establecer unos límites claros.

4. Ambigüedades genuinamente textuales

Por este tipo entendemos los casos donde el texto no proporciona información suficiente para eliminar la ambigüedad entre dos o más categorías perfectamente definidas.

3.4 UN EJEMPLO: AMBIGÜEDAD EN UNA ILN

Este apartado mostrará, a modo de ejemplo, cómo puede influir la ambigüedad en el comportamiento de un sistema de lenguaje natural. El sistema de lenguaje natural tomado como ejemplo es una ILN (Interfaz de Lenguaje Natural) que ya ha sido introducida en el apartado **2.3.1** de la presente memoria.

En la actualidad casi la totalidad de las ILN están orientadas a interfaces con bases de datos. Permiten que el usuario emita peticiones en forma de frases relativamente aleatorias para solicitar información de la base de datos.

Simplemente hay que pensar cuan deseable es que el usuario emita peticiones en forma de frases relativamente aleatorias (el objetivo final es, lógicamente, una arbitrariedad total), disponiendo de una ILN, para solicitar información contenida en la base de datos, como por ejemplo:

"¿Cuántos pintores trabajan en el departamento de chapa de esta filial?"
"¿Cuántos de ellos son mujeres casadas?"

Sin tener que ser experto en SQL (en el caso de tratarse de una base de datos relacional) con preguntas de este estilo sería posible obtener información de la base de datos.

Por supuesto, el lenguaje natural no se comprende aún a la perfección, y dado que la naturaleza, interpretación y representación del significado dista de ser una ciencia exacta, los sistemas de este estilo pueden malinterpretar los contenidos de las

instrucciones. Fijémonos en la situación siguiente propuesta por Rauch-Hindin [Rauch-Hindin, 1989].

Si efectuamos la siguiente consulta a un sistema que tenga la capacidad de responder al lenguaje natural, se presentan problemas de ambigüedad:

"How many cooks work at the company?"

Esta pregunta puede interpretarse de diferentes formas según nos refiramos a un restaurante o a una empresa siderúrgica, ya que las dos posibles interpretaciones son:

¿Cuántos cocineros trabajan en esta empresa?
¿Cuántos empleados de apellido COOK trabajan en la empresa?

Una forma inmediata de solucionar este problema es efectuar la consulta por escrito y poner mayúsculas y minúsculas, permitiendo diferenciar así apellido de profesión. Sin embargo esta forma de trabajar plantearía varios inconvenientes:

- Estamos reduciendo el lenguaje natural a su forma escrita, sin permitir la oral.
- Se introduce mayor complejidad y rigidez en el sistema.
- La solución adoptada sólo será eficaz para situaciones de este estilo, donde la diferenciación sea posible gracias a escribir con mayúsculas o no.

La realidad es que estos problemas son inevitables debido a que estamos tratando un elemento que intrínseca e inherentemente es muy complicado y ambiguo, como es el lenguaje natural.

La ambigüedad, y su tratamiento, es uno de los aspectos fundamentales a la hora de plantearse un sistema enmarcado dentro del procesamiento de lenguaje natural.

3.5 APROXIMACIONES A LA RESOLUCIÓN DE AMBIGÜEDADES LÉXICAS

En ocasiones las palabras, fuera de todo contexto, pueden pertenecer a más de una categoría léxica o *Part-of-Speech* (POS en adelante). El proceso de resolver ambigüedades léxicas y etiquetar cada palabra, en una sentencia, con una etiqueta POS se ha denominado POS *tagging*, o simplemente etiquetación.

Considérese, entre los muchos existentes, el ya mencionado caso de la palabra "sobre". Esta palabra posee múltiples acepciones:

- *Preposición*: Encima de, acerca de, además de, aproximadamente, ...
- *Nombre masculino*: Cubierta en que se incluye una carta u otra comunicación.
- *Verbo*: Dentro de esta categoría podemos distinguir, a su vez, dos posibilidades:
    - *Voz del presente de subjuntivo* del verbo *sobrar*.
    - *Voz imperativa* del verbo *sobrar*.

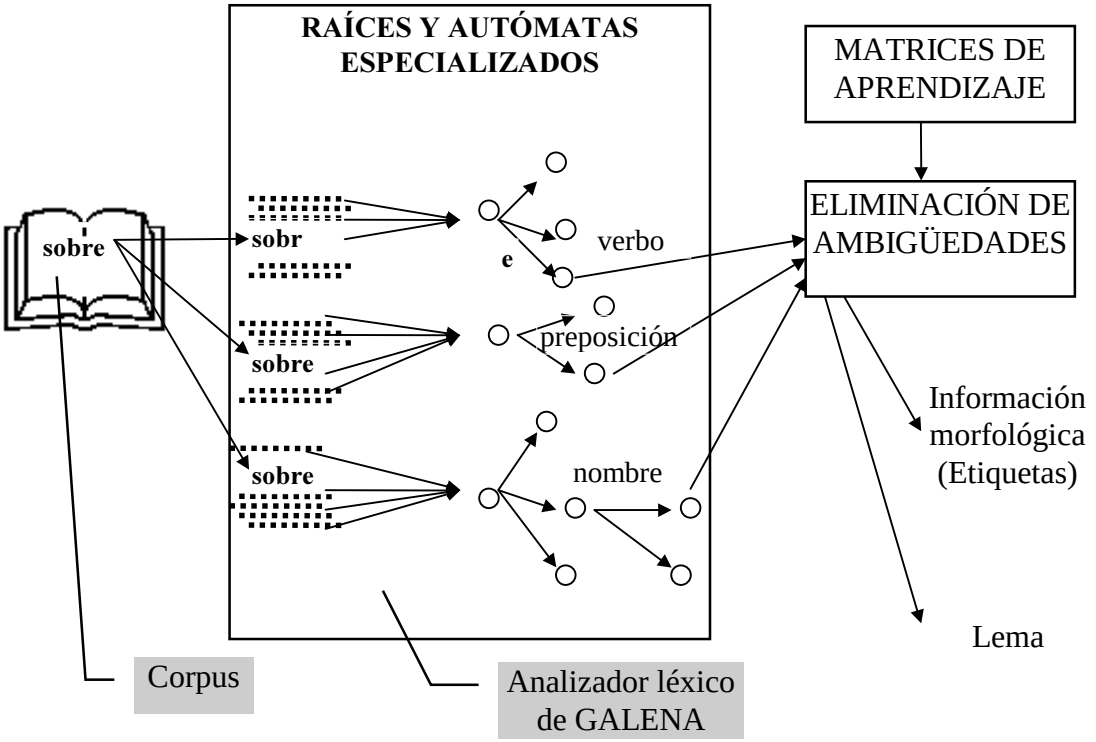


Figura 7. Posibilidades de la partícula “sobre”.

Sin embargo, en una determinada frase, la partícula “sobre” sólo tendrá un papel léxico. La cuestión radica en saber qué etiqueta es la correcta para la frase que se está analizando. La situación en nuestro sistema, para este caso en concreto, podría verse como indica la figura 7.

En esta figura se refleja el hecho de que al encontrar una palabra en un texto ésta va a sufrir un proceso de análisis léxico. En GALENA el análisis léxico proporcionado permite analizar la palabra en todas sus acepciones. Para proporcionar una única etiquetación de esa palabra, una vez conocidas todas sus acepciones, el módulo de supresión de ambigüedades hará uso de un aprendizaje previo, reflejado en matrices de aprendizaje, para seleccionar la que en cada caso resulte más verosímil.

La información que se proporciona al final de todo el proceso es el lema, así como la referente a los accidentes morfológicos de la palabra en relación a la etiqueta escogida. La etiquetación de una palabra recoge los valores que se hayan determinado para todos los campos de la estructura `token` (apartado **4.2.4.1**), que a su vez está determinada por los fenómenos léxicos considerados en GALENA (ver tabla 2 en pág. 90).

En muchos trabajos que abordan este problema se presentan resultados que evalúan el rendimiento de las diferentes aproximaciones. La dificultad del problema de la etiquetación depende de la naturaleza, grado de homogeneidad del entrenamiento, si lo hay, y evaluación de los textos, el número de etiquetas en el conjunto de etiquetas[8], y muchos otros factores. Sin un corpus[9] de evaluación de etiquetación POS estandarizado, o una forma de cuantificar y compensar esas variaciones, las comparaciones de unos

[8] *Tagset*.
[9] Conjunto de textos homogéneos.

resultados con otros no son muy significativas. Además hay que tener en cuenta que casi la totalidad de los resultados se ofrecen para el idioma inglés. El idioma español tiene mucha más variedad y complejidad de formas y expresiones, lo que deriva en una mayor complejidad de tratamiento en los sistemas.

En los puntos siguientes se presentarán las posibles aproximaciones más relevantes para abordar el problema de la supresión de ambigüedades léxicas. Se pretende dar una visión general de las filosofías existentes, a la vez que unas medidas de comparación, que se deben de tomar con cautela por las indicaciones ya mencionadas. Se han propuesto diferentes alternativas y todas ellas dividen el proceso en varias etapas [Klein y Simmons, 1963]:

1. Selección inicial de etiquetas, en donde se identifican todas las posibles etiquetaciones para una palabra.
2. Supresión de ambigüedades en las etiquetas, en donde se selecciona la etiqueta más apropiada.

- **Resolución de ambigüedades léxicas basada en reglas**

Green y Rubin, en 1971 [Green y Rubin, 1971], crearon el sistema TAGGIT, un programa de etiquetación para etiquetar el corpus Brown que resultó ser un buen ejemplo de resolución de ambigüedades léxicas basada en reglas. La selección que tiene lugar en TAGGIT se desarrolla como sigue:

Primero, los elementos léxicos se buscan en un diccionario de cerca de tres mil entradas, el cual incluye la mayoría de las clases de palabras. A mayores TAGGIT maneja una lista de casos especiales, como contracciones, símbolos especiales, etc. Finalmente, la palabra se confronta a una lista de sufijos, de aproximadamente cuatrocientas cincuenta entradas. Es decir, se logra asignar a cada palabra las etiquetas que en cada caso se determinen.

La supresión de ambigüedades en TAGGIT se basa en reglas heurísticas creadas manualmente. Estas reglas hacen uso de un contexto local para determinar la etiqueta correcta en cada caso. TAGGIT incorpora tres mil trescientas reglas de contexto que dependen de las dos palabras anteriores y de las dos posteriores de la palabra objetivo. Una forma general para estas reglas se muestra a continuación, donde "?" indica la palabra que se desea desambiguar[10]:

$$tag_i\ tag_j\ ?\ tag_k\ tag_l \rightarrow result\_tag$$
$$tag_i\ tag_j\ ?\ tag_k\ tag_l \rightarrow not\ tag$$

Un ejemplo de una regla de TAGGIT es el siguiente:

$$?\ VBZ \rightarrow not\ NNS$$

La regla anterior indica que la palabra objetivo, que precede a una tercera persona de singular (VBZ), no puede ser un nombre plural (NNS).

[10] Los términos desambiguar, desambiguación y similares no están reconocidos por la RAE. Se hace uso de ellos, sin embargo, para facilitar la lectura y no recurrir a giros que podrían resultar incómodos en la lectura.

Lo que se persigue en este tipo de sistemas es definir unas reglas que regulen las posibles combinaciones de etiquetas que puedan surgir. La desventaja de esta filosofía es que es necesario un esfuerzo muy considerable para crear, depurar y refinar las reglas, ya que la casuística que hay en juego es enorme.

Conviene aclarar que posteriormente los errores que se habían producido se corrigieron a mano para completar la etiquetación del corpus Brown, es decir, observaremos que el experto lingüista siempre será necesario, eso sí, en mayor o menor medida dependiendo del método.

• Etiquetación basada en frecuencias

Garside, en 1987 [Garside et al., 1987], muestra un trabajo que se basa en el corpus Lancaster-Oslo-Bergen (LOB) del inglés británico escrito.

El programa de etiquetación del corpus LOB se llama CLAWS[11] y hace uso de ciento treinta etiquetas diferentes. Su diccionario se basa en el corpus Brown etiquetado y contiene cerca de siete mil entradas. La lista de sufijos tiene cerca de setecientas entradas.

Un concepto previo muy importante en el estudio de este tipo de sistemas es el concepto de bigrama. Un bigrama es un par de etiquetas léxicas, como puede ser el caso, por ejemplo, del par (Determinante, Nombre). Resulta fácil la extensión al concepto de trigrama, y N-grama en general.

El procedimiento para la desambiguación léxica en CLAWS fue diseñado pensando en la importante observación de que cerca del 80% de los intentos de aplicaciones de las reglas de TAGGIT concernían a reglas que sólo hacían referencia a la palabra inmediatamente anterior o siguiente. En otras palabras, el 80% de las aplicaciones de reglas en TAGGIT concernían sólo a bigramas.

Basándose en esto, CLAWS fue diseñado para usar una matriz de posibilidades de colocación; un precursor de las actuales probabilidades de bigramas. Las probabilidades de bigramas en CLAWS se aproximan usando estimaciones de máxima probabilidad[12] basadas en las frecuencias de los bigramas de una gran porción del corpus Brown ya etiquetado. Inicialmente la fórmula que a continuación se presenta era la usada para las probabilidades de los bigramas; en ella t_i denota etiquetas y $f(t_i)$ denota la frecuencia observada de la etiqueta t_i:

$$P(t_i, t_{i+1}) \approx \frac{f(t_i, t_{i+1})}{f(t_i)} \quad \textbf{(1)}$$

Sin embargo, con los estudios realizados por Garside, se puso de manifiesto que esta forma de cálculo favorecía demasiado a etiquetas con muy alta frecuencia, pasándose a adoptar la siguiente definición:

$$P(t_i, t_{i+1}) \approx \frac{f(t_i, t_{i+1})}{f(t_i) \times f(t_{i+1})}$$

[11] *Constituent-Likelihood Automatic Word-Tagging System.*
[12] *Maximum likelihood.*

Una vez establecida la base de cálculo estamos en condiciones de estudiar la forma en que se desambiguaron las secuencias de etiquetas:

Considérese una secuencia de palabras ambiguas de longitud n. Esto define un número de posibles secuencias de etiquetas desambiguadas T_1, T_2, ..., T_k. El valor de una secuencia de etiquetas T_j, $V(T_j)$, se define como el producto de las probabilidades de los bigramas en la secuencia:

$$V(T_j) = \prod_{i=1}^{n-1} P(t_i, t_{i+1})$$

La probabilidad de una secuencia de etiquetas, T_j, se estima en CLAWS como el valor de la secuencia dividida por la suma de los valores de todas las posibles secuencias:

$$P(T_j) \approx \frac{V(T_j)}{\sum_{i=1}^{k} V(T_i)}$$

Para mejorar el rendimiento se llevaron a cabo algunas extensiones incluyendo la incorporación de factores basados en trigramas.

Los aciertos en CLAWS se encuentran en unos niveles considerados como típicos para esta clase de sistemas, y esto supone unos valores de 96%-97% de aciertos. Sobre estos niveles hay que indicar que se han obtenido haciendo uso de todo el corpus LOB, usando probabilidades de bigramas junto con unas pocas excepciones de trigramas.

• Desambiguación léxica con apoyo en modelos de Markov

Kupiec, en el año 1992 [Kupiec, 1992], describe una aproximación novedosa y alternativa a todas las existentes anteriormente para llevar a cabo la desambiguación. Ésta se basa en un modelo de Markov.

Considérense los puntos discretos en el tiempo $\{t_k\}$ para k=1, 2, 3, ..., y sea ξ_{tk} la variable aleatoria que caracteriza el estado del sistema en t_k. La familia de variables aleatorias $\{\xi_{tk}\}$ forma un proceso estocástico.

Un proceso de Markov es un sistema estocástico para el cual la ocurrencia de un estado futuro depende del estado inmediatamente anterior y únicamente de él.

Un modelo de Markov es un modelo matemático de un proceso estocástico definido como una colección de estados conectados por transiciones. El movimiento a través de la red de transiciones ocurre de acuerdo a una función de transición probabilística, produciéndose las etiquetas al atravesar un arco. Esto ocurre de acuerdo a una segunda función probabilística, la función salida, que define la probabilidad condicional de emitir un símbolo dado, una vez se ha originado una determinada transición.

Un modelo de Markov para la etiquetación léxica tiene la ventaja de que puede ser entrenado a partir de un corpus no etiquetado, sin embargo, se necesita un lexicón que describa las posibles etiquetas para las palabras.

En el modelo inicial hay un estado por cada categoría de POS, siendo la topología que nos encontramos en un modelo de Markov la de una red totalmente interconectada. Las palabras son emitidas a la vez que se atraviesan los arcos, pero como la cantidad de palabras es desbordante, no es factible estimar parámetros para cada palabra simple y en todos los arcos. Esto condujo a Kupiec a considerar el hecho de que cuando se desea reducir la complejidad de una situación es muy habitual definir una relación de equivalencia. Efectivamente, este autor dividió las palabras en clases de equivalencia dependiendo de sus posibles etiquetaciones, llegando a definir ciento veintinueve clases para las palabras, con clases adicionales para las palabras más comunes (aproximadamente cuatrocientas clases). La progresión de etiqueta a etiqueta se modeliza progresando a través de los estados del modelo de Markov, y las palabras se generan de acuerdo con las funciones de salida de los arcos.

La complejidad de esta aproximación es debida al propio modelo y al establecimiento de las clases. Sin embargo, el corpus de entrenamiento no necesita estar etiquetado, sino que puede ser, como así ha sido, algo tan poco habitual como un corpus de entrenamiento de cerca de un millón de palabras sacadas del correo electrónico, y teniendo como tema el diseño del lenguaje de programación Common Lisp.

Pese a que lo anterior puede ser, y de hecho así es, considerado como una ventaja debemos de indicar que la desambiguación manual de textos, que es una tarea ardua y no exenta de errores, sirve a varios propósitos [Aldezabal et al., 1996]:

- La depuración del sistema de etiquetas y la mejora de la representación gramatical en los análisis.
- Como texto para la validación de los resultados obtenidos con el etiquetador automático basado en el conocimiento lingüístico.
- Y, principalmente, como base para el aprendizaje en la desambiguación.

Por estas razones la desambiguación manual, aunque limitada, a nuestro entender debe seguir considerándose.

Los resultados que Kupiec presenta para su modelo son los siguientes: con un diccionario cerrado (estimado directamente desde el corpus), entrenando con cuatrocientas cuarenta mil palabras del corpus Brown, y probando con la misma cantidad de palabras del mismo corpus se obtiene un porcentaje de 95.7% de aciertos en total y 88.2% de acierto en palabras ambiguas.

Con un diccionario abierto (con cerca del 3% de palabras desconocidas en el texto de entrenamiento), entrenando con hasta ciento diez mil palabras procedentes de columnas de humor y probando con la misma cantidad de palabras que en el caso anterior del corpus Brown, el acierto total fue del 94.7% y el acierto en palabras ambiguas fue del 85.2%.

- **Etiquetación estocástica basada en N-gramas**

En 1988 Church [Church, 1988] notó ciertas características comunes entre las reglas de TAGGIT, la mayoría concernientes a bigramas, y el *parser* FIDDITCH de Donald Hindle [Hindle, 1983]. FIDDITCH es un *parser* determinista con un rango de cobertura amplio que hace uso de la desambiguación por reglas. Bastantes reglas de desambiguación en FIDDITCH pueden ser reformuladas en términos de probabilidades de bigramas y trigramas. Estimando automáticamente tales probabilidades podría resultar todo más sencillo que crear manualmente, probar y depurar las heurísticas simbólicas de desambiguación.

En ese mismo año Church describe el etiquetador PARTS que implementa el modelo anteriormente esbozado. Aquí a la palabra actual se le asigna una etiqueta basándose parcialmente en las etiquetas de las dos palabras siguientes. Se usan dos tipos de parámetros para ello:

- Probabilidades léxicas: P(t|w) - la probabilidad de observar la etiqueta t sabiendo que se ha dado la palabra w.
- Probabilidades contextuales: $P(t_i|t_{i+1}, t_{i+2})$ - la probabilidad de observar la etiqueta t_i dado que (condicionada a que) las dos siguientes etiquetas son t_{i+1} y t_{i+2}.

Nótese que en esta forma de trabajo el contexto es un contexto “futuro”, mientras que en los modelos anteriores el contexto hacía referencia a las etiquetas previas a la palabra objetivo, es decir, hacía referencia al pasado.

Las probabilidades fueron estimadas a partir del corpus Brown etiquetado usando el criterio de estimación de máxima probabilidad como sigue:

- Probabilidad léxica:

$$P(t|w) \approx \frac{f(t,w)}{f(w)}$$

donde f(t, w) es la frecuencia de observar la palabra w con etiqueta t.

- Probabilidad contextual:

$$P(t_i|t_{i+1},t_{i+2}) \approx \frac{f(t_i|t_{i+1},t_{i+2})}{f(t_{i+1},t_{i+2})}$$

El etiquetador de Church intenta encontrar las etiquetas que maximizan el producto de la probabilidad léxica y la probabilidad contextual, siendo la complejidad del algoritmo empleado lineal en el número de palabras de la sentencia. Dada una sentencia de entrada de longitud n la asignación de etiquetas es tal que maximiza la siguiente cantidad:

$$\prod_{i=1}^{n} P(t_i|w_i) \times P(t_i|t_{i+1},\ t_{i+2})$$

Con trabajos similares al etiquetador de Church la precisión lograda está cercana al 96% [Franz, 1996].

3.6 NECESIDAD DE SUPRESIÓN DE LAS AMBIGÜEDADES A NIVEL LÉXICO

Las razones por las que es necesaria la supresión de las ambigüedades léxicas son varias y resultan evidentes. Sin embargo, no está de más que se expresen de forma explícita para que las argumentaciones sean más concretas. Esto es lo que se va a realizar en este apartado.

De alguna manera las razones que se van a dar aquí constituirían las justificaciones de la realización de este proyecto. Dichas razones se argumentan para un caso general, pero debe estar presente el hecho de que este proyecto de encuadra dentro de un sistema concreto: GALENA.

La primera y más potente de las razones es que, hasta la llegada de este módulo de supresiones, el etiquetador que se poseía no podía ser considerado como tal, ya que para un *token* dado proporcionaba todas sus posibles etiquetaciones. Esta situación, técnicamente hablando, no se debería de producir, ya que en los etiquetadores se debe de proporcionar una categoría por palabra. Es decir, se busca una relación 1:1 entre palabra y etiquetación léxica a la salida del etiquetador. Como ya se ha dicho esta situación no se daba y es una de las razones por las que se abordó el presente proyecto.

La segunda razón que se argumenta supone considerar la fase posterior al análisis léxico en el entendimiento del lenguaje natural: el análisis sintáctico.

Al analizador sintáctico, para que éste pueda realizar el *parsing*, hay que darle una única etiquetación por palabra. Esto está directamente relacionado con la primera razón, es decir, se impone la necesidad de un etiquetador, en el sentido puro, para que al analizador sintáctico le llegue un *token* con sólo una categoría. Sería impensable, para un análisis sintáctico, que un *token* tuviese varias alternativas de etiquetación (¿qué etiquetación cogería?, y ¿en qué se basaría para ello?). El *parser* no tiene la capacidad de seleccionar una etiqueta de entre varias, es labor del etiquetador realizar este trabajo; y en el caso de que el *parser* pudiese hacerlo estaríamos introduciendo trabajo, que se encuadra a nivel léxico, en el nivel sintáctico. Esto repercute en un acoplamiento muy peligroso, dejando aparte el hecho de que no tiene sentido alguno.

Como tercera razón, que refuerza las dos anteriores, se puede esgrimir el hecho de que el proyecto GALENA se está llevando en paralelo con otro proyecto, de su misma filosofía, pero teniendo el gallego como lengua objetivo: el proyecto XIADA[13].

En el idioma gallego se da una característica que hace que el supresor de ambigüedades tenga, si cabe, una mayor personalidad. Esta particular característica es debida a que, por el modo de articulación, los fonemas vocálicos se pueden distinguir en tres grupos [Costa et al., 1988]:

[13] Xerador Incremental de Analizadores De gAlego.

- De apertura máxima o **abiertas**: donde la posición de la boca es la más abierta (/a/).

- De apertura mínima o **cerradas**: la posición de la boca es la más cerrada (/i/, /u/).

- De apertura media o **medias**: la posición de la boca es intermedia entre la más abierta y la más cerrada. En este caso tenemos dos tipos de vocales medias, unas que se aproximan más a la apertura máxima, y otras que lo hacen a la apertura mínima. Son, respectivamente:

  - **medias abiertas**: (/ɛ/, /ɔ/).
  - **medias cerradas**: (/e/, /o/).

No hay ninguna regla fija que permita discernir si una "e" o una "o" gráficas se pronuncian abiertas o cerradas. Pero lo más importante es que hay muchas palabras que, escribiéndose igual, tienen significados distintos y que se diferencian sólo fonéticamente debido a la apertura de las vocales tónicas. Tal es el caso de las siguientes palabras:

- **pe**:

  Con e tónica cerrada es la letra del alfabeto.
  Con e tónica abierta es la extremidad del cuerpo humano.

- **besta**:

  Con e tónica cerrada es un animal.
  Con e tónica abierta se refiere al arma.

- **bola**:

  Con o tónica cerrada es la pieza de pan.
  Con o tónica abierta se refiere a un cuerpo esférico.

- **polo**:

  Con o tónica cerrada es el gallo joven.
  Con o tónica abierta se refiere a un extremo o al juego.

Esta peculiaridad hace que el módulo supresor de ambigüedades adquiera mayor importancia si se piensa en diseñar un sintetizador de voz, aspecto éste que se está estudiando en el Centro de Investigacións Lingüísticas e Literarias Ramón Piñeiro de Santiago de Compostela con el que se está colaborando. Esto es debido a que la pronunciación será distinta dependiendo del papel que juegue la palabra (etiquetación léxica), y por tanto el escoger una u otra posibilidad de etiquetación, y es obligación escogerla para pronunciar de una forma u otra[14], influirá en el comportamiento posterior del sintetizador de voz, esto es, en la mejor o peor cercanía a la pronunciación humana.

[14] En castellano la pronunciación no depende de la acepción, por lo que esta cuestión pierde importancia.

Lógicamente las dos razones anteriores prevalecen y esta razón aumenta la importancia del supresor de ambigüedades, para aquellos sistemas que trabajan con lenguajes que poseen estas peculiaridades.

El sistema de supresión de ambigüedades desarrollado tiene su origen en estas razones y cumple con la necesidad de seleccionar una única etiqueta léxica y guiar al analizador sintáctico. Como se verá más adelante, no se descartan las etiquetas no seleccionadas, sino que se mantiene un sistema de prioridades para evitar que si la elección no permite construir el árbol de análisis sintáctico se pueda coger la siguiente, y así sucesivamente, hasta permitir construir el citado árbol.

4. EL SISTEMA GALENA

4.1 INTRODUCCIÓN AL SISTEMA GALENA

El entorno GALENA[15] es un proyecto conjunto de las tres universidades gallegas: Universidad de Santiago de Compostela, de Vigo y de A Coruña, y es el resultado de la inquietud y colaboración de un equipo de lingüistas y de un equipo de informáticos. Este trabajo se inscribe tanto en el terreno de la lingüística computacional como en el de la lingüística de corpus. Dicho en pocas palabras, el objetivo es desarrollar herramientas de análisis lingüístico automático que permitan un tratamiento eficaz de la interpretación de lenguajes naturales en su forma escrita.

Una vez logrado el objetivo de construir estas herramientas, las puertas que se abren son muchas y muy variadas:

- Traducción automática.
- Sintetización de voz.
- Análisis de las construcciones lingüísticas.
- Sistemas de aprendizaje.
- Interfaces de lenguaje natural.
- Entendedores de textos.
- ...

GALENA tiene como idioma objetivo el español, pero paralelamente se está desarrollando el sistema para adecuarlo a las muchas particularidades que presenta el gallego. De esta forma los avances son, generalmente, más rápidos ya que la filosofía de desarrollo va paralela. La única diferencia, que no es poca, es la debida a las diferencias que existen entre una lengua y la otra.

El objetivo es construir programas informáticos que sean capaces de etiquetar gramaticalmente cada forma (*tagging*), vincularla al lema que le corresponde y, en el paso siguiente, realizar el análisis sintáctico de las cláusulas en los elementos funcionales que las constituyen. Posteriormente, en un futuro muy cercano, habrá que abordar la parte siguiente: el análisis semántico, para permitir así que el ordenador pueda asociar acciones efectivas al texto interpretado.

Las herramientas resultantes, de interés ya en sí mismas en tanto que son instrumentos de análisis lingüístico automatizado, permitirán abordar todas las posibilidades que se enumeraron con anterioridad en esta misma página.

[15] Generador de Analizadores de LEnguaje NAtural.

4.2 ANÁLISIS LÉXICO EN GALENA

4.2.1 INTRODUCCIÓN

El análisis léxico constituye el primer paso en el procesamiento informático de las lenguas naturales, entendido como el etiquetado de clases de palabras, así como la detección y posible eliminación de ambigüedades en la determinación de dichas clases. En el estudio de los lenguajes de programación dicha fase suele tener un tratamiento práctico totalmente independiente de las fases de análisis sintáctico y semántico. Sin embargo, en el caso de las lenguas naturales esta parte del proceso general de análisis tiene especial importancia en el tratamiento subsiguiente. Es por ello necesario asegurar un tratamiento eficiente del mismo.

En este contexto, el trabajo que a continuación se describe pretende dar un tratamiento práctico al problema de la etiquetación de lenguas naturales [Graña, Alonso y Valderruten, 1994], tanto en lo que se refiere a la velocidad de tratamiento como a su dominio de aplicación. Para ello, se ha partido de la base representada por la experiencia acumulada durante años en el tratamiento de este mismo problema, a nivel de los lenguajes formales de programación. En concreto, la técnica propuesta representa un sustancial distanciamiento de las aplicadas tradicionalmente en la etiquetación de textos, fundamentadas en el uso de bases de datos, para orientarse hacia métodos basados en el uso de autómatas finitos de diseño específico, que incorporan mecanismos operacionales para el tratamiento del conocimiento lingüístico disponible acerca de la estructura morfológica de las palabras.

El entorno aplicativo más estudiado en relación al tratamiento del análisis léxico lo constituyen sin duda los lenguajes de programación. El conocimiento obtenido en esta área podría resultar de gran provecho en el tratamiento de otro tipo de lenguajes, tales como los utilizados para la comunicación humana, si tenemos en cuenta las diferencias que separan a unos y otros:

- En un lenguaje de programación, cada palabra se corresponde con un único componente léxico. En el caso de las lenguas naturales, una palabra, por sí misma y aislada (sin contexto), puede tener múltiples acepciones que se corresponden con diferentes categorías gramaticales.
- Una palabra puede contener más de un componente léxico (tal es el caso de los verbos con pronombres enclíticos: cógelo) e incluso puede suceder que un único componente léxico englobe a más de una palabra (es el caso de las locuciones prepositivas: de cara a, ...).
- Las lenguas naturales poseen un complejo entramado de reglas morfológicas que controlan la construcción de las variantes válidas de cada palabra, aspecto éste que no aparece en los lenguajes de programación.

En consecuencia, un analizador léxico para lenguas naturales debe proveer mecanismos no-deterministas que permitan obtener todas las posibles interpretaciones de una palabra, así como mecanismos operacionales que le permitan rentabilizar las posibilidades que ofrece el análisis morfológico. Más exactamente, el analizador debe proveer todos los posibles reconocimientos para una palabra dada y no debe pasar a reconocer un nuevo componente léxico hasta que no se hayan agotado todas las

posibilidades para el precedente. Sólo de esta manera se puede asegurar que el posterior análisis sintáctico y semántico sea capaz de tratar correctamente el texto, seleccionando de entre todas las opciones aquellas que mejor concuerden con la estructura general del mismo.

Un caso particular interesante es el de la lengua española, especialmente rica en lo que a estructuras morfológicas se refiere, con decenas de miles de lemas que se corresponden con las entradas en el diccionario. Cada lema representa lo que podríamos llamar la forma canónica de un conjunto de palabras. Así, el lema de un sustantivo lo constituye la forma masculina singular. Las demás formas, femenino y plurales en masculino y singular, pueden ser derivadas a partir de la forma canónica utilizando las reglas morfológicas correspondientes al tipo de sustantivo del que se trate. En el caso de los verbos, la necesidad de utilización de estas reglas se hace más acusada. En los verbos regulares, nos basta con conocer la forma del infinitivo para poder conjugar todos los posibles tiempos verbales. Sin embargo, los verbos irregulares siguen sus propias reglas léxicas. En este caso, es incluso probable que sea preciso utilizar más de una raíz para su conjugación. Un ejemplo típico es el del verbo **pensar,** para cuya conjugación es necesario utilizar las raíces **pens y piens.**

Una forma de solucionar los problemas planteados consiste en utilizar una inmensa base de datos en la cual estén almacenadas todas las formas de todas las palabras. Cada vez que se tenga que reconocer una palabra, se accede a esta base de datos y se recuperan todas las posibles alternativas. Este enfoque presenta serios inconvenientes:

- El tamaño desproporcionado que deberá tener la base de datos para ser capaz de obtener buenos resultados con textos grandes. Ello, o bien obliga a que residan en almacenamiento secundario gran parte de los recursos del sistema, o bien limita notablemente sus prestaciones.
- Cada vez que se encuentra una nueva palabra se deben introducir todas las formas derivadas de su lema, si queremos que la base de datos sea consistente. Por ejemplo, para cada verbo sería necesario introducir todas las formas verbales relativas a su conjugación. Se podría evitar la entrada manual de las formas mediante un generador de formas derivadas, pero aún así se tiene el problema del crecimiento exponencial del tamaño de la base de datos con respecto al número de lemas, lo que degrada el rendimiento ya que introduce retardos en el tiempo de acceso, siempre considerando que se posee una buena estructura de índices.

Para evitar esos inconvenientes hemos considerado que la mejor solución es utilizar un generador de analizadores léxicos en el que se puedan incorporar las reglas morfológicas y el comportamiento no-determinista, adoptando el concepto de autómata como formalismo operacional. El resultado es un programa ejecutable que se encarga de realizar las labores de etiquetación con un buen rendimiento temporal y un consumo de memoria mínimo. Como punto de partida para el diseño de esta nueva herramienta se ha optado por utilizar Flex [Mason y Brown, 1990], un generador de analizadores léxicos estándar en el sistema operativo Unix [Kaare, 1990].

Pese a todo lo anterior en el sistema actual existe una base de datos de lemas que nos facilita mucho las tareas de consulta, inserción y depuración de los lemas que se introducen en el sistema. Sin embargo, no se cae en los problemas mencionados con anterioridad puesto que a partir de la base de datos de lemas se generan los autómatas en

código Flex a partir de una serie de programas. De esta manera se consigue mantener una interfaz agradable de cara al usuario a la vez que una eficiencia muy alta.

En secciones posteriores se describe el formalismo utilizado para implementar los mecanismos operacionales derivados de las reglas morfológicas y se muestra un pequeño ejemplo. También se describirán los mecanismos no-deterministas incorporados en el generador de analizadores léxicos propuesto, permitiendo de esta manera analizar múltiples interpretaciones de una palabra. La integración del analizador léxico con el analizador sintáctico también será objeto de estudio, mostrando los mecanismos de comunicación entre ambos. Un breve estudio acerca de la arquitectura concreta del analizador construido para la lengua española concluirá esta presentación.

4.2.2 LAS REGLAS LÉXICAS

Las palabras de una lengua como la española se pueden dividir en dos partes: el lexema y los afijos. El lexema es la parte invariable de la palabra. En ciertos casos la palabra está formada únicamente por el lexema. Tal es el caso de las preposiciones (a, de, para, ...). Sin embargo, normalmente las palabras se forman mediante la combinación de una serie de sufijos y/o prefijos (afijos en general) junto con el lexema, como ocurre, por ejemplo, en el caso de los sustantivos, los adjetivos y los verbos. Actualmente los prefijos no son tratados de una manera especial sino que son introducidos como lemas aparte. Por el contrario, como se podrá observar a lo largo de esta exposición, sí se ha hecho un estudio exhaustivo para el tratamiento de los sufijos.

Cuando de un lexema se puede obtener un conjunto de palabras que derivan de él, se toma una de esas palabras para representar el conjunto: es lo que se denomina el lema de ese conjunto de palabras. En el caso de los sustantivos y los adjetivos el lema lo constituye la forma masculina singular, y en el de los verbos el infinitivo.

En ciertos casos, como en el de los verbos irregulares, las cosas no son tan simples, puesto que no todas las formas de la conjugación tienen la misma raíz. Consideremos por ejemplo el verbo **pensar**. Parte de la conjugación utiliza como raíz **pens** (pensé, pensaré), pero otras utilizan **piens** (pienso, piensas) como raíz. Sin embargo, todas las formas de la conjugación tienen un único lema: **pensar**.

Por lo tanto, a la hora de reconocer las palabras válidas del español se deberá partir del reconocimiento de los lexemas para posteriormente ir aplicando las reglas léxicas que indican cómo se formarán las palabras pertenecientes al conjunto de un determinado lema.

Mención y tratamiento especial, que aquí no se discutirá, merecen los verbos compuestos, los pronombres enclíticos y los elementos léxicos de varias palabras (en concreto las locuciones prepositivas).

4.2.2.1 UNA APROXIMACIÓN INTUITIVA

Para aclarar el significado de las reglas léxicas utilizaremos como ejemplo los sustantivos. En español se han detectado, por parte de los expertos, hasta veinte formas distintas de formar el género de una palabra, catalogadas como G1, G2, ..., G20. Hay grupos muy comunes, como por ejemplo el G1, que añade una “o” en el caso del masculino y una “a” en el del femenino. Otros son más inusuales, como el grupo G8, que forma el masculino añadiendo la terminación “que” y el femenino mediante la

terminación “ca”. Un ejemplo de este grupo lo constituyen las palabras “cacique” y “cacica”.

Para formar el número se han detectado diez formas, catalogadas como N1, N2, ..., N10. El grupo más común es el N1, que añade una “s” para formar el plural y ningún carácter para el singular. Un grupo bastante inusual es el N4, que utiliza la terminación “c” para el singular y la “ques” para el plural. Es el caso de las palabras “frac” y “fraques”.

La mayoría de los sustantivos forman tanto el género como el número, por lo que es necesario combinar los dos tipos de reglas léxicas. En cualquier caso, siempre se forma primero el género y después el número. Además, no son válidas todas las combinaciones de grupos de formación género-número. Por ejemplo, los lexemas que utilizan el grupo G1 para formar el género siempre utilizan el grupo N1 para el número.

En el caso de los verbos, en español, como es fácilmente previsible, existen muchas más reglas léxicas que determinan cómo se construye cada una de las conjugaciones y la treintena de grupos de verbos irregulares.

Todo este conocimiento debe ser incorporado en la etapa de análisis léxico para determinar los componentes que se han de reconocer. Puesto que las reglas léxicas son fijas, para ampliar posteriormente el conjunto de palabras que es capaz de reconocer el analizador tan solo es necesario introducir el lexema y establecer los grupos con los cuales se forman las palabras válidas que comparten dicho lexema. Estas modificaciones se ven también facilitadas por el hecho de utilizar un generador de analizadores, en lugar de programar directamente en un lenguaje de propósito general. Actualmente estas modificaciones carecen de dificultad ya que, como se ha dicho con anterioridad, se dispone de una base de datos que facilita este tipo de operaciones. Esta base de datos mantiene la mínima información para, a partir de ella, poder generar los ficheros que posteriormente serán compilados para generar el analizador léxico [Vilares, Graña y Pan, 1996].

La interfaz de usuario con respecto al analizador léxico

El mantenimiento de la información léxica almacenada en la base de datos puede realizarse a través de una interfaz gráfica especialmente diseñada para tal efecto. Las posibles operaciones a realizar por el usuario se corresponden con cada uno de los tres modos de la interfaz:

- **Modo de consulta, modificación y borrado**

  Todos los campos, con todos sus valores, se ponen a disposición del usuario a través de unos menús, de forma que se puede construir fácilmente una transacción y realizar operaciones de *browser* sobre las raíces ya introducidas. La figura 8 muestra la pantalla asociada con este aspecto.

- **Modo de introducción**

  Este modo es válido para la introducción de nuevas raíces en todas las categorías, salvo en los verbos irregulares, los cuales, debido a su complejidad, disponen de un modo propio. Se incluye un pequeño generador de formas, mediante el cual el usuario puede verificar que el lema introducido se asigna al grupo de flexión correcto. La figura 9 presenta la pantalla asociada al modo de introducción.

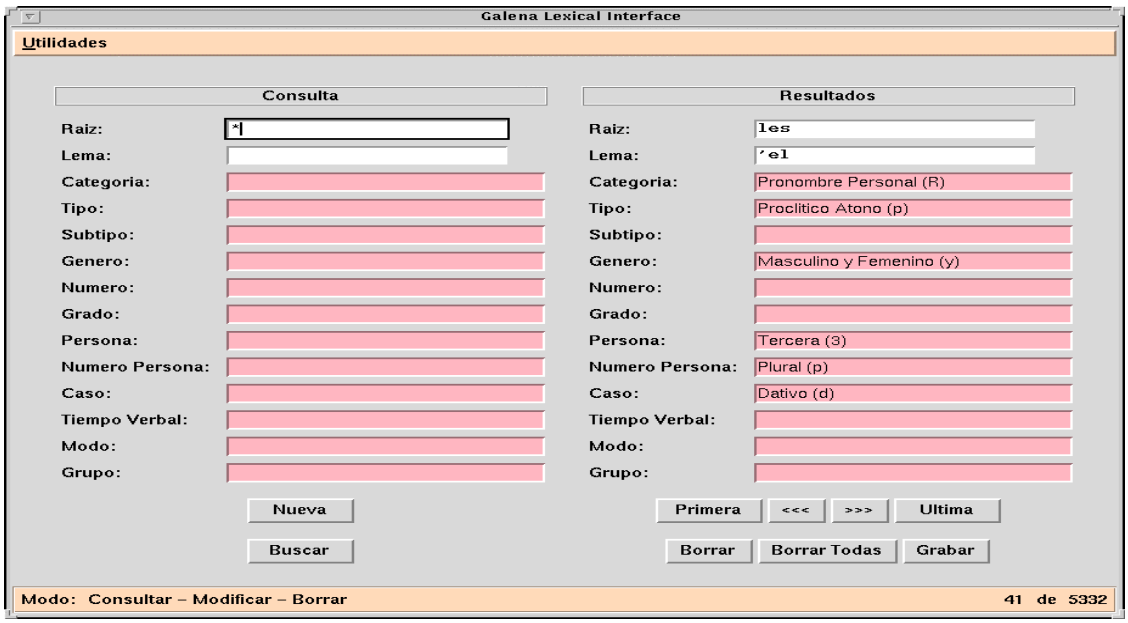


Figura 8. Modo de consulta, modificación y borrado.

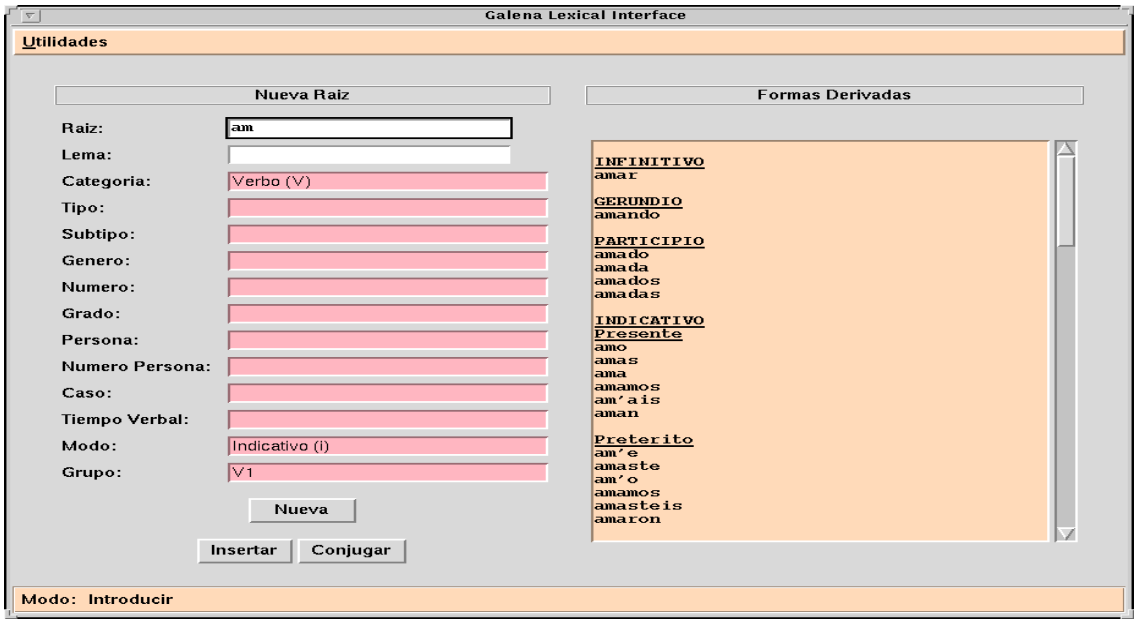


Figura 9. Modo de introducción.

• **Modo de introducción de verbos irregulares**

Debido al hecho de que cada verbo irregular involucra a más de una raíz, se proporciona al usuario un modelo de cada grupo de irregularidad verbal, con

un juego de raíces por defecto. La otra problemática principal se centra en el hecho de que los pronombres enclíticos pueden introducir cambios de acentuación en las raíces. Sin embargo, la interfaz es capaz de generar automáticamente las raíces extra que deben ser introducidas para estos casos. El generador de formas muestra algunas de las formas con enclíticos, tal como puede verse en la figura 10.

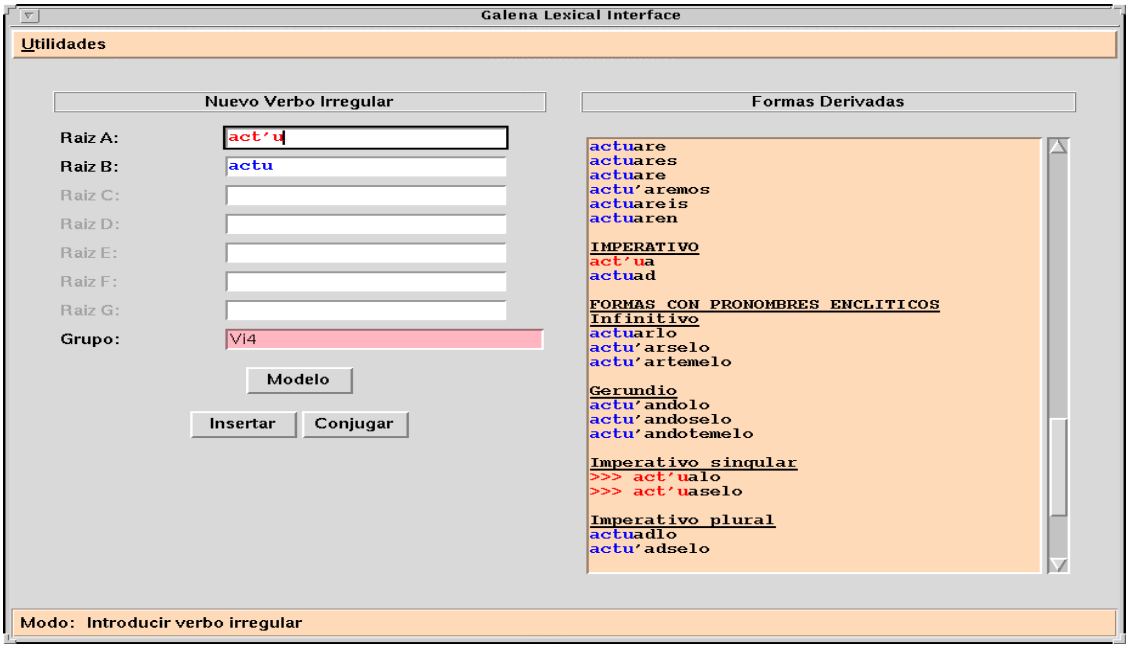


Figura 10. Modo de introducción de verbos irregulares.

Posteriormente, a partir de toda esta información léxica, el proceso de compilación automáticamente construye el etiquetador, que en nuestro caso supone construir el código Flex del autómata finito que actuará como etiquetador léxico.

4.2.2.2 LAS CONDICIONES DE ARRANQUE

Para facilitar la implementación de las reglas léxicas, se pueden utilizar las condiciones de arranque (*start conditions*) de Flex, que permiten continuar la búsqueda de expresiones regulares una vez que ha sido detectado el lexema de una palabra.

Las condiciones de arranque proporcionan un mecanismo para establecer la ejecución condicional de las reglas. Cualquier regla cuyo patrón comience por `<sc>`, se activará solamente cuando la condición de arranque `sc` esté activa.

La utilización de condiciones de arranque es similar a definir miniautómatas especializados en el examen de las reglas léxicas. Su utilización permite flexibilizar la introducción, modificación o eliminación de palabras. En efecto, una vez diseñados los miniautómatas correspondientes a la detección de accidentes gramaticales, la inclusión de una familia léxica se reduce a la inclusión de una entrada en el autómata correspondiente a su raíz. Por tanto, sólo nos resta ser exhaustivos a la hora de detectar los casos que nos obligan a definir esos miniautómatas especializados. Se podría pensar en los miniautómatas como subfunciones que se encargan de comprobar la existencia o

no de accidentes gramaticales, sin embargo, no se programan directamente a bajo nivel, sino que hacemos uso del generador de analizadores.

4.2.2.3 IMPLEMENTACIÓN DE LAS REGLAS LÉXICAS

Una vez introducidas las ideas fundamentales en cuanto al diseño de nuestro analizador, podemos comenzar la descripción de los mecanismos operacionales que constituyen las reglas léxicas. Así, el procedimiento mediante el cual hemos implementado en Flex las reglas es el siguiente:

- Utilizar reglas sin condición de arranque para reconocer los lexemas, los cuales constituirán los patrones de tales reglas.
- Definir una condición de arranque para cada grupo de reglas léxicas. Utilizar los sufijos como patrones de las reglas incluidas en esa condición.
- En las acciones que se disparan por el reconocimiento de un lexema, lanzar la condición de arranque que represente al grupo en el cual se reconocen las palabras derivadas de ese lexema.
- En las acciones de las reglas con condiciones de arranque se puede tomar una de las siguientes alternativas:
    1. Lanzar una condición de arranque que indique los sufijos que deben ser reconocidos a continuación. Esto se utiliza, por ejemplo, cuando después de reconocer el género de un sustantivo es preciso reconocer su número.
    2. Si no es necesario comprobar la existencia de más sufijos, dar por terminado el reconocimiento de la palabra.

Es conveniente considerar la existencia de una condición `ERROR`, a la que se envíe el analizador cuando una palabra no pueda ser reconocida (puede ser que no se reconozca el lexema como válido, o que se reconozca el lexema pero no se pueda aplicar ninguna de las reglas léxicas para reconocer la palabra). Cuando el analizador entra en este estado de error, debe actuar en modo pánico (*panic mode* en la literatura anglosajona), saltando todos los caracteres siguientes hasta que encuentra un carácter de sincronización, que en nuestro caso será un separador (un blanco, un *enter*, un tabulador, un punto, una coma, ...).

4.2.2.4 UN EJEMPLO SENCILLO

Para que se pueda comprender mejor la implementación de las reglas léxicas, vamos a mostrar un pequeño ejemplo simplificado. Se trata de reconocer los sustantivos que forman el género mediante G1 y el número mediante N1 y se puede ver en detalle en [Graña, Alonso y Valderruten, 1994].

Lo primero que debemos hacer es definir las condiciones de arranque en la sección inicial de declaraciones del programa Flex. Para nuestro ejemplo, es suficiente con definir G1, N1 y E. Esta última constituye la condición de error a la cual se enviará el autómata del reconocedor siempre que una palabra no pueda ser reconocida. Esto se traduce en código de la siguiente manera:

```
/* Error */
%x E
/* Gender */
%x G1
/* Number */
%x N1
```

Después definimos los dos patrones siguientes:

- **sep**, formado por los caracteres separadores. Estos caracteres se utilizan para evitar reconocer una palabra dentro de otra. El conjunto de caracteres separadores, simplificado, está formado por los siguientes elementos: espacio, tabulador, coma, punto y coma, dos puntos, punto y el carácter de nueva línea.
- **nosep**, en el que se engloban todos los caracteres que no son separadores y que por tanto pueden aparecer dentro de una palabra.

En las siguientes línea de código se definen estos dos patrones:

```
/* Separator characters */
sep [" "\t,;:\.\n]
/* Non-separator characters */
nosep [^" "\t,;:\.\n]]
```

Con esto ya podemos comenzar a definir reglas en la sección de reglas del programa Flex. Como ejemplo vamos a mostrar cómo se reconocen las palabras cuyos lemas son “perro” y “gato”. Los lexemas de estas palabra son “perr” y “gat”, respectivamente. Como dijimos anteriormente, deberán ser utilizados como patrones en la parte izquierda de una regla sin condición de arranque, como se muestra a continuación:

```
gat |
perr {yymore(); BEGIN G1;}
```

Mediante la llamada a la función `yymore` indicamos a Flex que queremos seguir añadiendo en `yytext` (aquí es donde se almacena el texto reconocido en cada regla) los caracteres que sean reconocidos a continuación.

Ahora es preciso definir las reglas mediante las cuales se reconoce el género de las palabras incluidas en el grupo G1. Estas palabras forman el masculino añadiendo una “o” al lexema y el femenino añadiendo una “a”. En el siguiente código se muestra la definición de tales reglas:

```
/* Terminations for gender: Case 1 */
<G1>o       {   /* Masculine */
              printf(" masculine");
              yymore();
              BEGIN N1;
             }

<G1>a       {   /* Feminine */
              printf(" feminine");
              yymore();
              BEGIN N1;
            }
```

El prefijo `<G1>` antes de los patrones “o” y “a” indica que ambas reglas sólo se activarán cuando la condición de arranque G1 haya sido activada mediante `BEGIN G1`. Después de `BEGIN G1` únicamente estas dos reglas estarán activas. Como todas las

palabras que forman el género en G1 forman el número en N1, ambas reglas utilizan `BEGIN N1` para activar la condición de arranque N1.

Las reglas para el reconocimiento del número son las siguientes:

```
/* Terminations for number: Case 1 */
<N1>s/{sep}       {/* Plural */
                    printf(", plural --> %s",yytext);
                    BEGIN (INITIAL)
                   }

<N1>""/{sep}      {/* Singular */
                    printf(", singular --> %,s", yytext);
                    BEGIN(INITIAL);
                   }

<N1>.             {/* Bad word. Error. */
                    yymore();
                    BEGIN E;
                   }
```

La terminación `/{sep}` constituye un contexto de cola (*trailing context*). Se utiliza para indicar que el patrón de esa regla deberá estar seguido por un separador, pero el carácter separador no es incluido en `yytext` y será reconocido en patrones de reglas posteriores, pero no en la actual. Se utiliza en estos patrones para evitar reconocer una palabra incompleta.

La expresión `""` se utiliza para indicar un carácter vacío, puesto que las palabras de N1 no añaden ningún carácter para formar el plural.

El patrón “.” empareja con cualquier carácter excepto con un avance de línea. Se utiliza para detectar palabras erróneas o aquellas que no han sido incluidas en el lexical. En las acciones de esta regla se utiliza `yymore` para que al activar el modo pánico, en `yytext` se almacene todo el texto no reconocido como válido. La regla del error es la siguiente:

```
<E>{nosep}*/{sep}       {/* Panic mode. Recognise every character
                            until the next separator. */
                          printf(", ERROR --> %s", yytext);
                          BEGIN(INITIAL);
                         }
```

La acción `BEGIN(INITIAL)` hace que el reconocedor vuelva a las reglas que no tienen condición de arranque. Realmente es como si existiese una condición de arranque implícita llamada `INITIAL` en todas aquellas reglas que carecen de condición de arranque.

Por último las siguientes reglas permiten tratar los separadores.

```
" "   |
\t    |
\n          {/* non significative separators */ ; }
","   |
";"   |
":"   |
"."         {/* Punctuation marks */
              printf(" Punctuation mark -> %s", yytext);
              BEGIN(INITIAL);}
```

Los espacios, tabuladores y caracteres de nueva línea se ignoran, pero las marcas de puntuación se reconocen como un componente léxico, ya que son significativas en el proceso de análisis sintáctico para detectar los límites de las oraciones.

En este ejemplo sólo se muestra por pantalla el género y el número. En el analizador real, se almacenan los datos en una estructura que se pasa al analizador sintáctico. El modo en que se enlazan los analizadores léxico y sintáctico se muestra en el apartado **4.2.4**.

4.2.3 EL COMPORTAMIENTO NO-DETERMINISTA

En los lenguajes naturales existen ambigüedades a todos los niveles: léxico, sintáctico y semántico. En esta sección nos interesa ver el modo en que se pueden tratar las ambigüedades del nivel léxico. Algunos autores proponen utilizar un enfoque de ventana deslizante, generalmente de tamaño reducido, empleando técnicas estadísticas, como las cadenas de Markov o los autómatas probabilísticos, para determinar el componente léxico más probable que se corresponde con una palabra dada a la hora de reconocerla. Este enfoque tiene varios inconvenientes serios:

- La carga computacional asociada es muy elevada, lo que disminuye el rendimiento. Para mantenerlo entre límites aceptables, se precisaría ejecutar el programa en máquinas con una disponibilidad generosa de recursos.
- Muchas veces estos modelos trabajan en conjunción con bases de datos, lo que degrada todavía más el rendimiento, ignorando además el conocimiento disponible acerca de la formación de las palabras a nivel léxico.

Ello justifica que el enfoque que hemos adoptado haya sido el de ampliar las capacidades de los reconocedores clásicos en teoría de los lenguajes formales, incorporándoles un comportamiento no-determinista, para posteriormente poder guiar al analizador sintáctico dando una serie de prioridades a cada una de las posibilidades de etiquetación. Esta forma de operar no conlleva la obligación de seleccionar una única etiqueta en el proceso de análisis.

4.2.3.1 EL COMPORTAMIENTO DETERMINISTA DE LOS RECONOCEDORES CLÁSICOS

Los reconocedores generados para los lenguajes de programación son deterministas. Esto quiere decir que una palabra es analizada una sola vez y las acciones de las reglas disparadas se llevan a cabo también una sola vez. Sin embargo, en nuestro caso, mediante la utilización de la acción `REJECT` se puede indicar al reconocedor que rechace la entrada reconocida en los patrones de las reglas de la condición de arranque actual y que seleccione la segunda mejor regla cuyo patrón coincida con la entrada[16]. Aunque el propósito para el que fue creada esta acción se refiere fundamentalmente a la recuperación de errores, su utilización permite conseguir un comportamiento no-determinista local en la condición de arranque en la que se ejecutó. Sin embargo, mediante la utilización de `REJECT` no es posible ir atrás en el análisis más que hasta el comienzo de la condición de arranque actual. Si el reconocedor pasa por varias de estas condiciones de arranque hasta llegar al final de la palabra, como es nuestro caso, no es posible ir retrocediendo paso a paso hasta la condición inicial. La dificultad se encuentra

[16] Recordemos que en el caso de Flex, la mejor regla (la mejor elección) es aquella cuyo patrón coincide con la mayor cantidad de caracteres de la entrada. En caso de que dos patrones empaten, se considera mejor el situado antes en el código fuente.

en la falta absoluta de dependencia contextual a nivel léxico de los lenguajes de programación, aspecto este ineludible en los lenguajes de comunicación humana.

4.2.3.2 LA INCORPORACIÓN DEL NO-DETERMINISMO A LOS RECONOCEDORES CLÁSICOS

Puesto que la carencia de no-determinismo en los reconocedores clásicos viene dada porque no se almacena información global sobre el texto reconocido, la solución consiste en encontrar algún método para incorporar a los reconocedores tal información. A priori, ésta puede parecer una tarea desbordante, ya que la consecución de no-determinismo en todas las condiciones de arranque significaría tener que guardar una estructura arborescente, puesto que habría múltiples caminos mediante los cuales reconocer una palabra.

Sin embargo, si enfocamos bien el problema, se puede observar que éste radica en la iteración continuada en las reglas iniciales. Por lo tanto, de lo que se trata es de buscar un mecanismo que indique al reconocedor que un patrón ya ha sido reconocido para evitar que vuelva a lanzar la regla correspondiente.

De este modo, mediante la combinación de `yyless(0)`, la activación de la condición de arranque y la información del *matching* de los patrones, se puede conseguir un comportamiento no-determinista del autómata reconocedor. La solución adoptada se ha mostrado simple y eficiente, pues tan sólo es necesario mantener las dos variables siguientes:

- **`match_no`**, que indica el número de patrones que han sido emparejados con éxito hasta el momento en el proceso de reconocimiento de la palabra actual. Por tanto también indica el orden que ocupa en cuanto a mejor elección en términos del generador clásico.
- **`semaphore`** se utiliza para determinar cuándo se puede reconocer un determinado patrón. Su función es la de un semáforo clásico:
    1. Si su valor es 0, el patrón que intentaba reconocer el analizador léxico es un patrón válido, puesto que no se había utilizado antes para intentar reconocer la palabra actual.
    2. Si su valor es mayor que cero, indica la distancia al patrón correcto, en cuanto a mejor elección en términos de Flex, en relación al patrón que se está intentando reconocer.

Inicialmente ambas variables establecen sus valores a 0. Cuando se reconoce un patrón para una palabra, se incrementa en 1 el valor de `match_no` y si el valor de `semaphore` es 0, se iguala el valor de ésta al de `match_no`. Cuando se termine de reconocer esa palabra o se deseche por ser errónea, se llama a `yyless(0)` y `BEGIN(INITIAL)`, con lo cual el analizador léxico intentará reconocer la palabra otra vez desde el principio.

El reconocedor entrará otra vez por el mismo patrón de antes, pero esta vez `semaphore` tendrá el valor 1, lo que indica que no debemos escoger ese patrón, sino otro que es la siguiente mejor elección. Se decrementa su valor en 1 y se hace un `REJECT`, con lo cual se busca precisamente la siguiente mejor elección. El siguiente patrón reconocido será el correcto puesto que `semaphore` tendrá valor 0. Se incrementa el valor de `match_no` y se prosigue con el proceso. Además, se debe contemplar la posibilidad de una salida para el caso en que ya no haya más patrones correctos para la palabra

examinada. Esto se consigue mediante una llamada a `yymore()` y a `BEGIN S` desde una regla con el punto como patrón. La condición de arranque `S` contiene la regla siguiente:

```
/* Success condition */
<S>{nosep}*/{sep}       { BEGIN 0;
                          return (1);
                        }
```

donde la acción `BEGIN 0` es equivalente a la activación de la condición de arranque inicial. Esta regla es la única en la que se realiza un `return`, por lo que una vez que se llama a la función `yylex` desde el analizador sintáctico, no se devuelve el control hasta que se llega a la condición `S`, hecho que se da cuando ya se han examinado todos los posibles componentes léxicos válidos para una palabra.

Existe también una condición de error, llamada `E`, a la que se envía al reconocedor cuando una palabra no puede ser construida mediante el camino examinado:

```
/* Error condition */
<E>{nosep}*/{sep}       { yyless(0);
                          BEGIN(INITIAL);
                        }
```

cuya interpretación es igual que la de la regla con `<S>` excepto que ésta no devuelve el control al analizador sintáctico, sino que intentará encontrar otro camino para reconocer la palabra.

Un ejemplo clásico de ambigüedad es el de la palabra “para”, que puede ser reconocida como:

- Una preposición.
- La tercera persona del presente de indicativo del verbo "parar".
- La segunda persona del imperativo del verbo "parar".
- La primera persona del presente de subjuntivo del verbo "parir".
- La tercera persona del presente de subjuntivo del verbo "parir".

Estas situaciones hacen del no-determinismo una capacidad fundamental y absolutamente necesaria en el tratamiento de los lenguajes naturales en general y del español en particular. El siguiente paso, una vez se tienen todas las posibilidades de etiquetación para una palabra dada, es seleccionar la correcta en el contexto en el que se encuentra. Precisamente este es el objetivo del trabajo que constituye el núcleo del Proyecto de Fin de Licenciatura aquí presentado.

4.2.4 INTEGRACIÓN CON EL ANALIZADOR SINTÁCTICO

Los analizadores léxicos generados se integran perfectamente con los analizadores sintácticos generados mediante Yacc, Bison [Mason y Brown, 1990] o ICE. Esta integración se basa fundamentalmente en:

- El valor devuelto por las llamadas a la función `yylex`, que es un entero representando el componente léxico reconocido, excepto el valor 0, que representa la condición de fin de fichero.

- El valor semántico de cada componente léxico, almacenado en la variable `yylval`.

El problema surge cuando queremos incorporar el no-determinismo al reconocimiento de los componentes léxicos. La función `yylex` sólo es capaz de devolver un valor entero. Cuando se trata de un reconocimiento no-determinista, el analizador sintáctico debe recibir una lista de componentes léxicos por cada palabra que ha sido reconocida por el lexical.

4.2.4.1 LA VARIABLE TOKEN

Para facilitar la interacción y extensibilidad del analizador sintáctico con el reconocedor léxico no-determinista, se ha optado por mantener intactas las estructuras generadas por Flex y añadir una nueva variable, denominada `token`, que almacene la lista de componentes léxicos reconocidos en cada ejecución de la función `yylex`.

La variable `token` tiene como tipo una nueva estructura denominada **`ice_lex_object`**. Esta estructura, que se define en el fichero de cabecera que contiene las declaraciones de tipos y funciones C utilizadas en el reconocedor, se muestra en el siguiente fragmento de código:

```
struct ice_lex_object

            {     STRING_LIST word;
                  STRING_LIST lemma;
                  INT_LIST    category;
                  INT_LIST    type;
                  INT_LIST    subtype;
                  INT_LIST    gender;
                  INT_LIST    number;
                  INT_LIST    degree;
                  INT_LIST    person;
                  INT_LIST    person_number;
                  INT_LIST    lcase;
                  INT_LIST    verbal_tense;
                  INT_LIST    mode;
                  INT_LIST    counter;
                  INT_LIST    priority;
            };
```

donde el significado de los distintos campos es el siguiente:

- **`word`**. Este campo almacena la cadena de caracteres correspondiente a la palabra que ha sido reconocida en la última llamada a la función `yylex`. Es necesario que sea una lista ya que existen locuciones que presentan ambigüedades que obligan a ello. Un ejemplo es el de la locución “de cara a”. Al analizar ésta puede considerarse todo junto (como locución), o bien pueden analizarse los tres componentes que la forman por separado (“de” + ”cara” + ”a”). Por tanto el primer *token* que se obtiene contendrá dos posibilidades: locución[17] (con este campo conteniendo “de cara a”) o bien preposición (con este campo conteniendo “de”).
- **`lemma`**. Este campo es una lista de cadenas de caracteres que indica el lema de cada palabra según cada tipo de componente léxico que ha sido reconocido

[17] El juego de etiquetas finalmente considerado no comprende las locuciones, sin embargo la discusión es válida ya que lo único que variará es la forma de etiquetar la locución.

para dicha palabra. Por ejemplo, tomemos la palabra “para”, que es reconocida como:

- Una preposición, en cuyo caso el lema es “para”.
- Dos formas verbales del verbo parar, en cuyo caso el lema es “parar” para ambas. Cada forma verbal se corresponde con un componente léxico distinto y por lo tanto con dos elementos en cada una de las listas presentes en la estructura `token`.
- Dos formas verbales del verbo parir, en cuyo caso el lema es “parir” para ambas.

- **`category`**. Es una lista de valores enteros que almacena el tipo de componente léxico que se ha reconocido. Cada uno de los valores se corresponde con una declaración `%token` en el fichero de la gramática. Para comprender mejor su significado, diremos que cada valor de esta lista se corresponde con un valor válido que podría ser devuelto por la función `yylex` cuando se trata de reconocedores léxicos deterministas convencionales.

Los campos **`word`** y **`category`** constituyen la información básica que se debe almacenar en `token` para permitir el enlace entre los analizadores léxico y sintáctico. Por tanto, deben ser incorporados en cualquier programa que pretenda utilizar el reconocimiento no-determinista de los componentes léxicos. El resto de la información almacenada en la variable `token` es específica de la aplicación que se trata en este trabajo. Diferentes aplicaciones pueden redefinir la estructura `ice_lex_object` para adaptar el contenido de los campos a sus necesidades específicas. Otra posible solución hubiese sido situar esta información dependiente de la aplicación en la variable `yylval`. Para ello habría que definir dicha variable como un puntero a una lista de estructuras en las que se almacenaría tal información. El problema surge al tratar de mantener la consistencia entre la lista de componentes léxicos y la lista de valores de `yylval`. Para facilitar el mantenimiento de esta consistencia se ha considerado más adecuado almacenar toda la información que tiene relación con las capacidades no-deterministas en una única estructura. Con ello se consigue adicionalmente un alto grado de aislamiento de estas nuevas características con respecto a las capacidades estándar de los reconocedores generados por Flex en el caso de los lenguajes de programación, es decir, se logra una mayor robustez desde el punto de vista de ingeniería del software. Los campos con información adicional dependiente de la aplicación que se han utilizado son[18]:

- **`type`.** Indica el tipo léxico de una palabra, si es aplicable.
- **`subtype`.** Indica el subtipo léxico de una palabra, si es aplicable. Complementa la descripción proporcionada por el campo anteriormente descrito.
- **`gender`**. Almacena el código que identifica el género de aquellas palabras en las que es aplicable esta característica. Para aquellos componentes léxicos en los que no es aplicable recibe el valor No-Aplicable. Esta lista está coordinada con la lista **`category`**, lo que quiere decir que el primer elemento de la lista de género indica el género del primer *token* en la lista de categorías, el

[18] Todos ellos son listas de enteros.

segundo elemento en la lista de géneros el valor del género para el segundo componente léxico, y así sucesivamente.

- **`number`**. Este campo indica el valor del número para aquellas palabras en las cuales es aplicable.
- **`degree`**. Con este campo podremos saber el grado comparativo de la palabra.
- **`person`**. Al igual que los campos anteriores, y sucesivos, es una lista coordinada con la lista de categorías, que en este caso indica la persona.
- **`person_number`**. Indica el número que corresponde a la persona de la palabra reconocida.
- **`lcase`**. Similar a todos los demás campos, en este caso indicando el caso de la palabra. Se denomina **`lcase`** puesto que `case` es una palabra reservada de C. El nombre del campo se tomó debido al hecho de que estamos tratando un caso léxico (*lexical case*).
- **`verbal_tense`**. Indica el tiempo verbal de una palabra, si es aplicable.
- **`mode`**. Indica el modo del tiempo verbal si procede.
- **`counter`**. Indica el número de palabras que conforman el *token*. En los verbos compuestos la etiquetación ya considera que se trata de verbos compuestos (“haber amado” por ejemplo) y de existir un valor en este campo es debido a que existen pronombres enclíticos. Este campo no sólo es necesario por los enclíticos sino también por las locuciones (de cara a, en aras de, ...).
- **`priority`**. En este campo se indica qué prioridad, sobre las demás ambigüedades que presenta la palabra reconocida, posee la alternativa de etiquetación a la que se corresponde la posición actual en esta lista.

El objetivo básico del trabajo que se presenta es cumplimentar adecuadamente este último campo sobre la base de un aprendizaje restringido por el usuario.

Como se ha podido observar en la definición de la estructura `ice_lex_object`, los campos que son listas de enteros tienen asignado el tipo `INTLIST`, y los que son listas de cadenas de caracteres el tipo `STRINGLIST`. Estos dos tipos han sido definidos para facilitar el manejo de tales listas, ya que además de definir el tipo en sí se define un conjunto de funciones asociadas que permiten trabajar de un modo seguro y estándar sobre estas listas.

Para poder realizar una actualización consistente de la información del componente léxico, lo que implica mantener la coordinación entre todas las listas de la variable `token`, se ha definido un conjunto de funciones que evitan la realización de actualizaciones manuales de tal variable.

Estas funciones han sido testadas y se han mostrado seguras en el tratamiento de la información del componente léxico. Siempre que se deba modificar cualquier dato almacenado en `token` se debe hacer uso de ellas, evitando el acceso “*ad-hoc”*, fuente potencial de problemas de inconsistencia de la información.

4.2.4.2 PROCESO DE ACTUALIZACIÓN DE LA INFORMACIÓN DEL COMPONENTE LÉXICO

El valor de la variable `token` debe ser inicializado y mantenido por el usuario. El esquema que se ha seguido es el de ir actualizando su contenido tan pronto como esté disponible algún tipo de información relevante.

El valor retornado por la función `yylex` pierde, con esta operativa, gran parte de su significado, puesto que ahora tan sólo se consideran dos posibles opciones:

- Que el valor devuelto sea 0, en cuyo caso indica que se ha alcanzado el final del fichero sobre el cual se está realizando el reconocimiento de los componentes léxicos.
- Cualquier otro valor indica el reconocimiento de alguna palabra. Por defecto el valor devuelto se establece a 1. El reconocimiento de una palabra puede tener tres orígenes diferentes:

    1. La detección de una correspondencia de la palabra con alguno de los tipos de componentes léxicos declarados en el analizador sintáctico.
    2. El reconocimiento como una palabra errónea, esto es, una palabra que está mal construida según las reglas léxicas.
    3. Una palabra desconocida.

El valor contenido en la variable `yylval` es irrelevante, ya que toda la información semántica concerniente al proceso de reconocimiento de componentes léxicos se encuentra almacenada en la variable `token`.

4.2.4.3 ACTUALIZACIÓN MEDIANTE REGLAS LÉXICAS

Según se van aplicando las reglas léxicas, se va obteniendo información acerca de la palabra que está siendo reconocida.

4.2.4.3.1 EL RECONOCIMIENTO DE LOS LEXEMAS

La detección del lexema de una palabra indica que posiblemente se ha encontrado un nuevo componente léxico en que encuadrarla. Por ello, las reglas encargadas de realizar el reconocimiento de los lexemas llaman a la función de inicialización `new_column_token`. En ese momento, ya estamos en condiciones de poder establecer la siguiente información concerniente al nuevo análisis que se está realizando de la palabra:

- La categoría de la palabra.
- El lema. La información del lema puede establecerse ahora, ya que una vez que se conoce la categoría y el lexema, se pueden aplicar las reglas léxicas que determinan el lema.

Para localizar esta información en la variable `token` se utilizan las funciones `set_cat_token` y `set_lem_token`.

4.2.4.3.2 EL RECONOCIMIENTO DE LOS SUFIJOS

En cada una de las condiciones de arranque a las que se envía al reconocedor, se obtiene algún tipo de información relevante que debe ser almacenada en la variable `token`. Por ejemplo, cuando se alcanza una de las condiciones que reconoce el género de una palabra, se obtiene la información que determina si el género es el masculino o el femenino. Lo mismo ocurre cuando se alcanza una condición referente al número con respecto al singular o al plural. En el caso de los verbos, las condiciones de arranque que identifican el tiempo verbal y la persona. Por tanto, cuando se alcanza una condición de arranque, se debe realizar una llamada a la función que actualiza la lista en la cual se almacena la información que se acaba de obtener. El encadenamiento de las condiciones de arranque conlleva la actualización sucesiva de diferentes listas.

Aquella información que no se obtiene en el proceso de reconocimiento de una palabra queda igualada a No-Aplicable, puesto que así se estableció cuando fue inicializada mediante las llamadas a `new_column_token` o `copy_column_token`. Por ejemplo, los sustantivos no tienen ni tiempo ni persona verbal, por lo que los campos `person` y `verbal_tense` tendrán valor `NA` (No-Aplicable).

4.2.4.4 ACTUALIZACIÓN NO-DETERMINISTA

La actualización no-determinista de los componentes léxicos está relacionada con la utilización de las condiciones de arranque correspondientes a la salida y a la detección de errores.

4.2.4.4.1 LA CONDICIÓN DE SALIDA

A esta condición de arranque se llega a partir de la regla sin condición de arranque cuyo patrón es el punto. Con ello se indica que ya se han reconocido todos los posibles componentes léxicos para una palabra dada.

En este momento se dispone de la información concerniente a la porción de texto que constituye la palabra, por lo que se puede establecer el campo `word` de la variable `token`. Ciertamente, esta información ya se conoce cuando se finaliza la cadena de condiciones de arranque para cada análisis léxico de la palabra. Sin embargo, este es el único lugar en el que se puede determinar con seguridad que no se va a realizar ningún análisis más, a nivel léxico, de esta palabra. Por tanto es el lugar idóneo para guardar esta información, ya que se garantiza que sólo será actualizada una vez para cada palabra. Pero lo que es aún más importante, es la única condición de arranque por la que se garantiza que pasará el análisis de toda palabra, por lo que las dos alternativas para actualizar el campo `word` serían:

- Actualizar el campo `word` en cada una de las condiciones de arranque que finalizan una cadena de reconocimiento de reglas léxicas.
- Actualizarlo en la condición de salida, por la que siempre se pasa.

La solución más simple y eficiente consiste en realizar el almacenamiento de la información concerniente a la cadena de caracteres que conforma la palabra, que acaba de ser reconocida, en la condición <S> de salida.

Si se llega a esta condición con los campos de la variable `token` vacíos, significa que la palabra no ha sido reconocida como una de las incorporadas al analizador léxico. En este caso, se utiliza la categoría Desconocido para la palabra. Para establecerla, es necesario llamar a la función `new_column_token` y posteriormente a `set_cat_token`, pasándole a esta última, además de la dirección de `token`, el valor `UKN` que identifica a la categoría Desconocido.

Cuando se alcanza la condición de salida, el analizador léxico devuelve el control al sintáctico. En este momento, este último ya dispone en la variable `token` de toda la información, concerniente a la palabra, que se ha podido obtener.

4.2.4.4.2 LA CONDICIÓN DE ERROR

Esta condición se alcanza cuando una palabra ha comenzado a ser reconocida aplicando algunas reglas léxicas, pero se ha detectado que no verifica ninguna de las alternativas para la continuación del proceso de reconocimiento. En tal caso, en las acciones correspondientes a la regla incluida en esta condición de arranque se llama a la función `rm_column_token` para eliminar la información del análisis erróneo de la variable `token`. En caso de que se desee realizar una depuración del analizador léxico, podría ser interesante mantener tal información, pero sumando al campo categoría un valor `ERR` que indique que dicha información es errónea.

4.2.5 ARQUITECTURA DEL ETIQUETADOR

A continuación mostramos la estructura formal de un etiquetador de palabras en español construido según las técnicas y métodos mostrados en las secciones precedentes. La primera tarea a realizar consiste en la determinación de todas las posibles categorías en que se pueden etiquetar las palabras reconocidas y, obviamente, para cada categoría todos los accidentes que se puedan presentar. Esta tarea produce como resultado un juego de etiquetas, cuya calidad estará avalada por la experiencia de un equipo de lingüistas, que ha sido quien ha desarrollado la clasificación. En la tabla 1 se muestra, de forma resumida, el sistema de etiquetas que se ha considerado.

El siguiente paso consiste en la definición de los miniautómatas que realizarán el reconocimiento de los morfemas. Mediante está técnica se incorpora el conocimiento lingüístico relativo a la construcción del léxico español, y es aquí donde los informáticos tienen un gran campo por explorar.

A modo de curiosidad en la tabla 2 se expone el juego de etiquetas, totalmente detallado, que usa actualmente el analizador léxico incorporado a GALENA.

En esta tabla es posible observar todas las posibilidades que se pueden dar para cada campo de la variable `token`. Cuando en una intersección aparece `last` esto quiere decir que, al usar el etiquetador, esa información aparecerá al final de toda la etiquetación que la herramienta proporciona al usuario. Una interrogación (?) en un cruce indica que es posible esa información para el caso considerado, pero que está en

proceso de implantación en la herramienta. Y por último aclarar que el punto (•) indica en qué situaciones tiene sentido el campo `counter` de la variable `token`.

Campo	Valores posibles	
Word	La palabra en la forma introducida.	
Lemma	La forma canónica de la palabra.	
Category	Adjective	Sin tipo.
	Adverb	exclamative, modifier, nuclear, nuclear & modifier, interrogative y relative.
	Article	Sin tipo.
	Conjunction	coordinate y subordinate.
	Demonstrative	Sin tipo.
	Indefinite	Sin tipo.
	Interjection	Sin tipo.
	Interrogative	Sin tipo.
	Numeral	cardinal, ordinal, partitive y multiple.
	Peripheral	foreign word, formula, abbreviation, symbol, acronym y other.
	Preposition	Sin tipo.
	Personal pronoun	tonic, proclitic atonic y enclitic atonic.
	Possessive	Sin tipo.
	Punctuation mark	dot, comma, semicolon, colon, open | close question mark, open | close exclamation mark, open | close parenthesis, dash, quotes y dots.
	Relative	Sin tipo.
	Substantive	common y proper.
	Unknown	Sin tipo.
	Verb	Sin tipo.
Subtype	determiner, non-determiner y both.	
Gender	masculine, feminine, both, neutral, y non-applicable.	
Number	singular, plural, both y non-applicable.	
Degree	comparative y non-applicable.	
Person	first, second, third, first & third y non-applicable.	
Person number	singular, plural, both y non-applicable.	
Case	nominative, accusative, dative, accusative & dative, case with preposition y nominative & case with preposition.	
Verbal tense	present, preterite, copreterite, copreterite_se, future, postpreterite, antepresent, antepreterite, antecopreterite, antecopreterite_se, antefuture, antepostpreterite y non-applicable.	
Mode	indicative, subjunctive, imperative, infinitive, compound infinitive, gerund, compound gerund y participle.	

Tabla 1. Sistema de etiquetas considerado en el analizador léxico.

Tabla 2. Juego de etiquetas detallado del analizador léxico.

4.3 ANÁLISIS SINTÁCTICO EN GALENA

4.3.1 INTRODUCCIÓN

Muchos autores sostienen que un algoritmo de *parsing* eficiente es crucial cuando se está construyendo sistemas de lenguaje natural prácticos. Esta afirmación es lógica desde el momento en que sabemos que el análisis sintáctico construye el árbol de análisis del texto en estudio, sin el cual es difícil realizar una aproximación al entendimiento del texto tratado.

En este apartado se pretende dar una visión general del analizador sintáctico que está usándose actualmente en el proyecto GALENA. Dicho analizador se ha construido a partir de la herramienta ICE[19], la cual es capaz, a partir de la definición de una gramática de contexto libre, de generar un analizador sintáctico no-determinista e incremental para dicha gramática.

Inicialmente ICE sólo realizaba análisis de gramáticas de contexto libre, pero posteriormente ha sido extendida a gramáticas de cláusulas definidas, concepto este último que se introducirá más adelante a la hora de mostrar el sistema.

Se ha construido además una interfaz gráfica asociada al *parser*, desarrollada utilizando AÏDA[20], que permite interactuar de forma sencilla e intuitiva con el analizador sintáctico explotando sus facilidades.

En este capítulo se presentará, sin entrar en profundidad, la herramienta ICE, el *parser* generado y la interfaz desarrollada [Alonso, 1994], más que centrarse en la escritura de la gramática, lo cual cae dentro de las habilidades del equipo de lingüistas.

La necesidad de introducir este capítulo en la presente tesis de licenciatura es debido al hecho de que, ante una palabra que presente ambigüedades, el supresor de ambigüedades desarrollado no optará por una etiquetación determinada, eliminando de esta manera el resto de las opciones. Por el contrario se permitirá que el *parser* tenga conocimiento de todas las ambigüedades en la etiquetación, indicándole para cada posibilidad, la prioridad, según estudios estadísticos previos, que tiene en el proceso de selección para obtener todos los posibles árboles sintácticos. De esta forma no se eliminan alternativas que, estadísticamente, no serían prioritarias pero que en la situación actual podrían ser las correctas para generar el árbol de análisis.

Con esta filosofía se explota la habilidad del analizador para construir todos los árboles sintácticos (análisis no-determinista), pero añadiéndole un lazarillo que hace que considere ordenadamente las posibles etiquetaciones de una palabra, evitando que coja una tras otra sin ningún método. Con esta forma de actuar se explora todo el espacio de posibilidades ordenadamente y se consigue:

- Por una parte, generar primero los árboles con las etiquetas más probables antes que los árboles con las menos probables.
- Por otra, permitir hacer un estudio léxico de las ambigüedades, ya que se dan todas las opciones marcadas con una determinada probabilidad contextual.

[19] *Incremental Context-Free Environment*.
[20] Una herramienta para la construcción de interfaces gráficas desarrollada sobre la base de Le-Lisp.

Puede suceder que para la etiquetación prioritaria de una palabra no sea posible construir ningún árbol sintáctico pero, siguiendo esta filosofía, al morir ese intento de construir el árbol por parte del *parser*, se puede pasar a la siguiente etiquetación en orden de prioridad. Esta posibilidad no podría llevarse a cabo si se descartan las posibilidades que las estadísticas no revelan como las más frecuentes dado un determinado contexto.

Estudios realizados muestran que los supresores estadísticos de ambigüedades léxicas no sobrepasan el 96%-97% de aciertos. La forma de actuar, en definitiva, permite llegar a construir, si es posible, el árbol de análisis aunque el supresor se encuentre en ese margen del 4%-3% de error. Este aspecto es del todo imposible si en vez de establecer una política de prioridades tomamos una decisión absoluta, esto es, comprobamos si el análisis sintáctico es posible con la única opción que tenemos (etiquetación elegida).

Por todo lo anterior es conveniente mostrar, aunque sea de forma muy breve, el trabajo desarrollado en el campo del analizador sintáctico, de cara a tener una visión completa del trabajo que aquí se presenta.

4.3.1.1 EL ANÁLISIS INCREMENTAL NO-DETERMINISTA

Dentro del ámbito de la informática, el desarrollo de técnicas para el diseño de analizadores sintácticos siempre ha tenido especial relevancia, principalmente debido a su estrecha relación con el campo de los compiladores. Desde hace años, se han estudiado con detenimiento una serie de técnicas que permiten realizar de un modo eficiente el análisis sintáctico (o *parsing*) de un texto fuente de un lenguaje. Entre estas técnicas podemos citar, como las más conocidas, la LL(k), de carácter descendente, y las distintas variantes de la técnica LR(k), de carácter ascendente.

Las técnicas ascendentes se han mostrado como las más eficientes en el tratamiento de la entrada, y de entre éstas destacan las LR(k). De hecho, Yacc, el conocido programa de generación de analizadores sintácticos ampliamente usado en el mundo Unix (generalmente en compañía del generador de analizadores léxicos conocido por Lex), implementa un algoritmo LALR, extrapolado directamente a partir de los LR(k).

Sin embargo, los principales desarrollos, incluido Yacc, presentan dos características que limitan su aplicación: su carácter determinista y no incremental.

4.3.1.1.1 EL ANÁLISIS INCREMENTAL

En el proceso tradicional, tras analizar la salida generada por el compilador (que puede estar constituida por un conjunto de listados con los errores encontrados y sus referencias al texto fuente), si efectivamente éste ha detectado errores, será necesario repetir el ciclo edición-compilación, lo que conllevará que el texto fuente sea reanalizado completamente, aunque el error tan sólo afecte a una mínima porción del programa. Ciertos compiladores no proporcionan un listado de todos los errores encontrados, sino que paran el proceso de compilación al encontrar el primer error, lo que agrava el problema. El usuario debe entonces modificar el texto y recompilar el programa tantas veces como errores posea en el código fuente.

En este punto no nos interesa si el compilador es llamado desde la línea de comandos o si por el contrario dispone de un entorno de programación que permite realizar la compilación directamente desde un editor. Lo que realmente interesa resaltar aquí es que, cada vez que se invoca al compilador, **todo** el texto fuente es reanalizado **completamente**.

Inmediatamente se puede pensar que reconstruir totalmente el árbol de análisis sintáctico constituye un derroche de recursos cuando la corrección del error tan sólo provocará la modificación de una rama de dicho árbol. De acuerdo con esto, lo ideal sería que tan sólo se reconstruyesen (o mejor dicho, se reanalizasen) aquellas ramas afectadas por el error. Sin embargo, para conseguir esto que aparentemente es tan sencillo se deben observar una serie de condiciones:

- El analizador sintáctico debe construir realmente una representación completa del árbol de análisis sintáctico, que debe estar disponible para el siguiente análisis.
- El analizador sintáctico debe conocer exactamente qué componentes léxicos han sido modificados por el usuario desde el último análisis.
- Debe de existir un entorno de compilación que mantenga el texto, el árbol de análisis sintáctico y las relaciones existentes entre ambos. Esto es, un editor interactivo.

El cumplimiento de estas condiciones implica una modificación sustancial del análisis sintáctico clásico, ya que:

- Los analizadores sintácticos más comúnmente usados en la actualidad no mantienen una representación completa de las estructuras de cálculo utilizadas en el análisis sintáctico, sino que suelen utilizar una pila en la que se van almacenando valores que representan el avance del proceso de análisis en un momento dado. El movimiento entre estados del autómata asociado al analizador provoca la localización de nuevos elementos, o su eliminación, de la pila. Generalmente, la realización de desplazamientos conlleva la introducción de más elementos en la pila, mientras que las reducciones implican la eliminación de la pila de un cierto número, n, de elementos a partir del tope. Dicho número n suele estar relacionado con la longitud de la parte derecha de la regla. De este modo se consigue un reconocedor muy eficiente tanto en tamaño como en velocidad, pero al finalizar el proceso de análisis se carece de una representación completa del árbol. Conviene recordar que éste era uno de los objetivos que debía de cumplir un análisis sintáctico como se indicó en el apartado **2.5**.

- Para que en un análisis incremental, de un texto previamente analizado, el analizador pueda saber qué parte del árbol debe ser reconstruida, éste debe poseer algún conocimiento sobre las modificaciones que se han realizado sobre el texto fuente y cómo han afectado a los componentes léxicos. Para conseguirlo es necesario integrar el analizador léxico con el texto de modo que el editor sea capaz de establecer las conexiones componente léxico-texto y pueda guiar al usuario en las operaciones de modificación, al mismo tiempo que debe ser capaz de indicar al analizador sintáctico qué porciones del análisis anterior han de ser revisadas.

En el procesamiento del lenguaje natural el uso de analizadores incrementales presenta más ventajas incluso que en el campo de los compiladores de lenguajes de programación, ya que permiten que ante una entrada errónea (una falta de ortografía, un error al realizar el OCR[21] de un documento digitalizado mediante un escáner, etc.) sólo se tenga que reanalizar, como mucho, la frase en la cual está contenido el error. En este contexto, sería prohibitivo que para subsanar un error se tuviese que realizar un nuevo análisis completo de todo el texto de entrada.

4.3.1.1.2 EL ANÁLISIS NO-DETERMINISTA

La utilización práctica de técnicas no-deterministas en la construcción de compiladores para lenguajes de programación es escasa. Las técnicas clásicas como LL(k) y LR(k) son deterministas. Esto quiere decir que para una entrada dada (un programa) sólo se obtendrá un árbol de análisis sintáctico. Como no todas las gramáticas de contexto libre pueden ser transformadas de modo tal que sean reconocidas por un autómata determinista, estas técnicas no reconocen todas las gramáticas de contexto libre, aunque sí las utilizadas en todos los lenguajes de programación. Algunas herramientas basadas en técnicas determinísticas, como Yacc, permiten que la gramática sea ambigua, pero a la hora de realizar el análisis, cuando se presentan los denominados conflictos, siguen normas bien conocidas para resolverlos. Por ejemplo, cuando Yacc detecta un conflicto de desplazamiento/reducción, opta siempre por realizar el desplazamiento, lo cual le lleva a comportarse bien ante casos como el del `else` ambiguo.

En el PLN el uso del no-determinismo es obvio ya que los idiomas presentan por su propia naturaleza cierto número de ambigüedades a nivel léxico y sintáctico. Para que un analizador sintáctico sea capaz de tratar con el no-determinismo debe ser capaz de tratar con más de un árbol sintáctico, en concreto con todos los que aplicando la gramática a un programa sean posibles. Para no perder eficiencia se debe buscar un método que permita compartir la mayor cantidad posible de información, ya que mantener varias representaciones completas del mismo árbol cuando tan sólo se diferencien en una serie de ramas (aquellas que corresponden a los distintos caminos a tomar cuando tenemos una ambigüedad) representaría un consumo de recursos excesivamente alto.

Para gramáticas de contexto libre, ICE es capaz de generar eficientes analizadores no-deterministas incrementales, como se puede comprobar a partir de los resultados empíricos mostrados en los siguientes artículos: [Vilares y Alonso, 1996] y [Vilares, Alonso y Cabrero, 1996 b]. En definitiva, ICE permite incorporar la incrementalidad y el no-determinismo al proceso del análisis sintáctico sin que ello suponga un coste elevado en cuanto a eficiencia, e incrementa en gran medida la flexibilidad en lo concerniente al diseño de gramáticas. Actualmente, la herramienta ICE permite generar analizadores no-deterministas y eficientes para gramáticas de cláusulas definidas. Se está trabajando en incorporar la incrementalidad al análisis de gramáticas de cláusulas definidas.

[21] *Optical Character Recognition.*

4.3.2 INTERFAZ GRÁFICA DEL ANALIZADOR SINTÁCTICO

El sistema GALENA, en lo que se refiere al analizador sintáctico, posee una interfaz de usuario dirigida por menús y multiventana. Para explicar el comportamiento de este entorno de trabajo se ha construido un *parser* para el subconjunto del español descrito por la gramática de cláusulas definidas que se expone a continuación en esta misma página.

Una gramática de cláusulas definidas se puede definir como una gramática de contexto libre en la que los no terminales son reemplazados por elementos de lógica de predicados de primer orden.

Este conjunto de nueve cláusulas describe un simple subconjunto del español, estableciendo algunas restricciones de concordancia en género y número [Vilares, Alonso y Cabrero, 1996 a]. Así, la segunda cláusula garantiza la congruencia del número entre el predicado nominal y el verbal al analizar una frase. El mecanismo para el paso de información se basa en la unificación.

Para esto evidentemente necesitamos recuperar información morfológica de las palabras que conforman la frase a ser analizada. Así, por ejemplo, la primera cláusula para NP (cláusula 4) recupera la cadena y el número del nombre, usando las palabras clave `word` y `number`. Los resultados obtenidos se guardan, en este caso, en los atributos `wrd` y `nmbr` respectivamente. El número `nmbr` se usará para inicializar el correspondiente atributo del predicado nominal, mientras que la cadena `wrd` se usará para construir el árbol abstracto `np(s(wrd))`.

```
(1) esp(tree)                          :      PHRASE(tree)
(2) PHRASE(phr(tree1, tree2))          :      NP(tree1, nmbr), VP(tree2, nmbr)
(3) PHRASE(phr(tree1, tree2))          :      PHRASE(tree1), PP(tree2)
(4) NP(np(s(wrd), nmbr)                :      NOUN(wrd:word, nmbr:number)
(5) NP(pr(wrd), nmbr)                  :      PRONOUN(wrd:word, nmbr:number)
(6) NP(np(det(wrd1), s(wrd2)), nmbr) :      DETERMINER(wrd1:word,nmbr:number,
                                                       gndr:gender),
                                       NOUN(wrd2:word, nmbr:number, gndr:gender)
(7) NP(np(tree1, tree2), nmbr)         :      NP(tree1, nmbr), PP(tree2)
(8) PP(pp(prep(wrd), tree))            :      PREPOSITION(wrd:word), NP(tree, nmbr)
(9) VP(vp(verb(wrd), tree), nmbr)      :      VERB(wrd:word,nmbr:number), NP(tree, nmbr)
```

La gramática anterior es una gramática de cláusulas definidas, pero tiene asociado un esqueleto de contexto libre que se obtiene eliminando los argumentos de los predicados anteriores. Dicho esqueleto es el siguiente[22]:

(0) $\Phi \rightarrow$ S
(1) S $\rightarrow$ PHRASE
(2) PHRASE $\rightarrow$ NP VP
(3) | PHRASE PP
(9) VP $\rightarrow$ verb NP
(4) NP $\rightarrow$ noun
(5) | pronoum
(6) | determiner noun
(7) | NP PP
(8) PP $\rightarrow$ preposition NP

Como primer paso de todo el proceso se debe crear el fichero describiendo la gramática. Para este propósito el sistema proporciona la interacción con la mayoría de

[22] Los números indican la correspondencia de las reglas.

los editores de texto de Unix. Así, por ejemplo, la figura 11 muestra al sistema trabajando con el editor `emacs`.

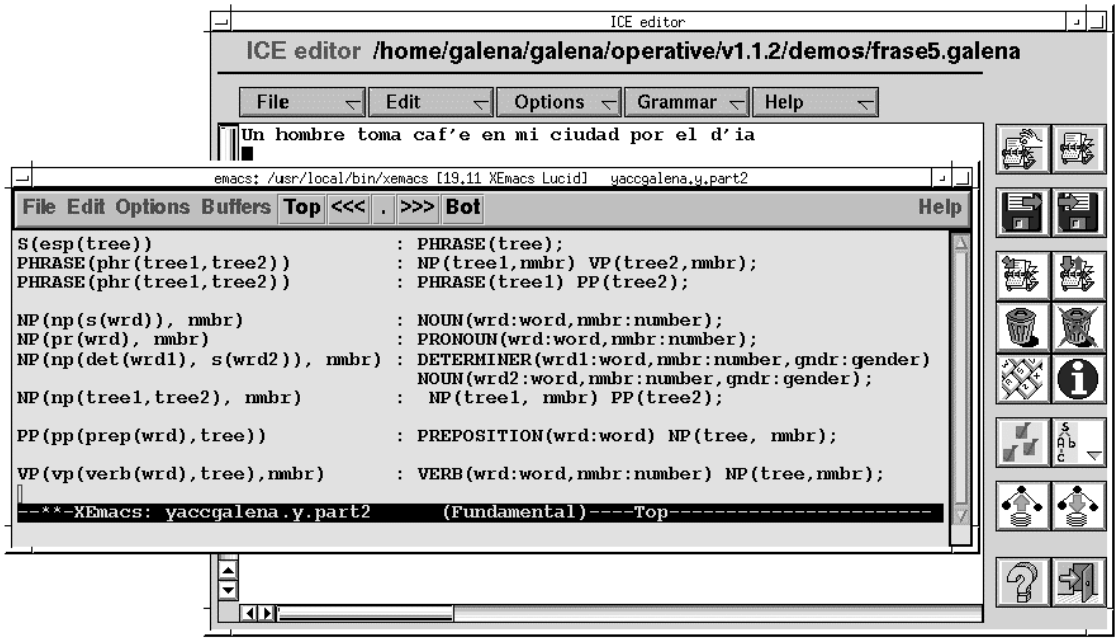


Figura 11. Interfaz interactuando con el editor EMACS.

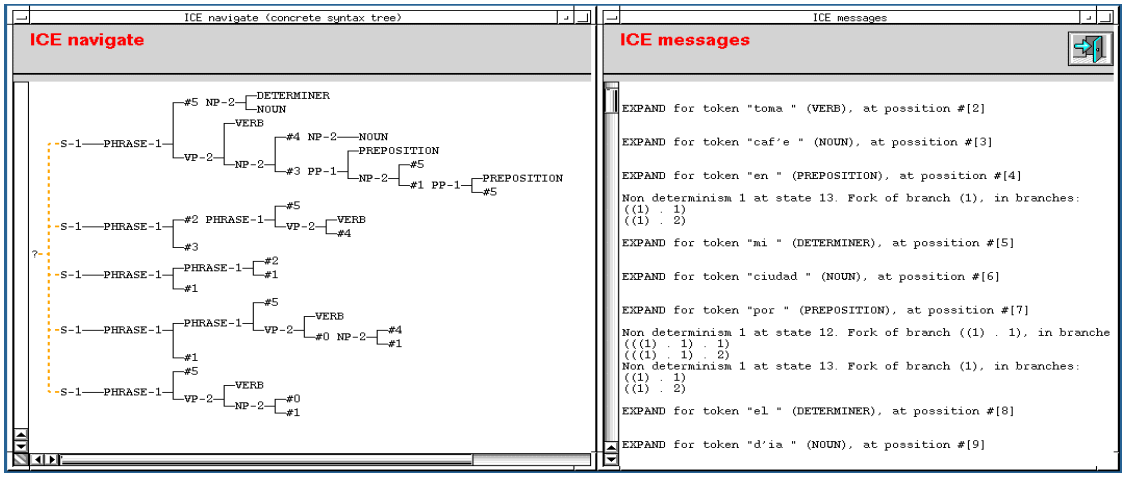


Figura 12. Ejemplo de bosque concreto.

La herramienta ICEeditor tiene un editor para escribir el texto a analizar, el cual se muestra en la pantalla posterior de la figura 11. Se puede apreciar que el texto que se ha escrito es: “Un hombre toma café en mi ciudad por el día”. El texto de entrada puede también ser directamente recogido desde un fichero previamente editado.

Cada vez que analizamos el texto, la ventana ICEmessages se activa con los mensajes resultantes, como se muestra en la parte derecha de las figuras 12 y 13. El usuario puede escoger el nivel de información que será mostrada en esta ventana: desde

nada, hasta las acciones más elementales, pasando por todas las posibilidades intermedias.

Se puede también navegar por el bosque compartido, el cual se muestra en la ventana ICEnavigate, mostrada en la parte izquierda de las figuras 12 y 13, conteniendo respectivamente los bosques compartidos concretos y abstractos.

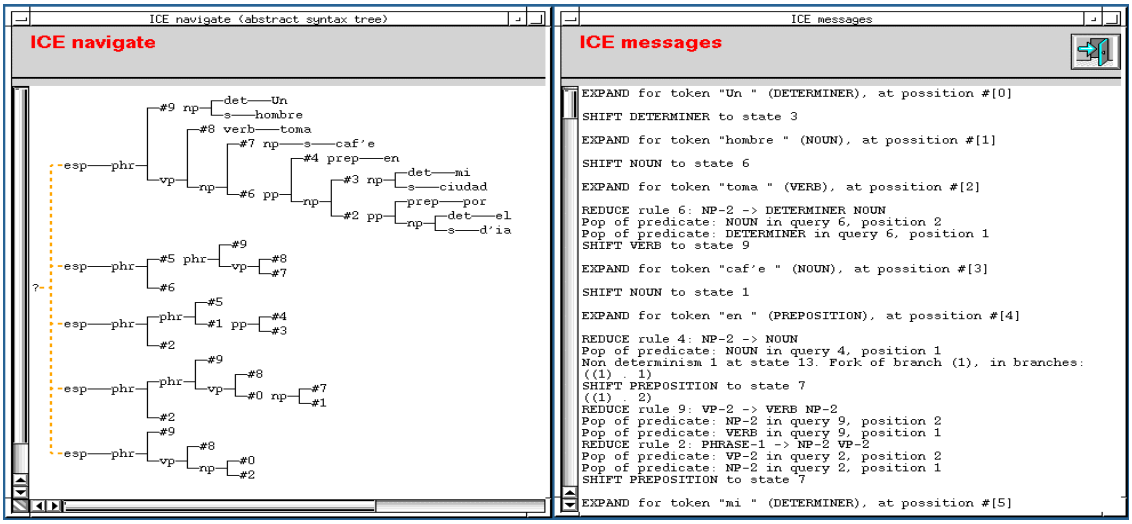


Figura 13. Ejemplo de bosque abstracto.

El bosque concreto es el que resulta de la aplicación directa de la gramática y el abstracto es el que se obtiene pensando en el tratamiento posterior. En el bosque abstracto nos interesa generar una estructura que esté orientada a nuestro tratamiento posterior. Dos frases dadas pueden tener dos bosques concretos distintos pero el abstracto igual. El bosque concreto, o de superficie también llamado, de "Yo hice la casa" y el de "La casa fue hecha por mí" son evidentemente distintos, ya que irán por distintas derivaciones. El bosque abstracto podría ser el mismo, si así nos interesa, y podría ser el relacionado con la estructura:

"`CONSTRUIR(AGENTE(yo) COMPLEMENTO(casa))`"

En ambos casos las ambigüedades se presentan por líneas punteadas y los nodos compartidos por expresiones de la forma `#n`.

Si es necesario, las estructuras generadas durante el proceso de análisis pueden ser guardadas en disco y recuperarlas en sesionas futuras.

Para obtener un entorno más amigable, se puede seleccionar el lenguaje en el que el sistema interactúa con el usuario: inglés, francés, español y gallego son los disponibles actualmente. Además una facilidad de ayuda está disponible en todo momento para resolver dudas acerca de los editores, generación del *parser* y facilidades de *parsing*.

5. EL SISTEMA PROPUESTO

5.1 INTRODUCCIÓN

En los últimos años se ha observado un aumento considerable de la investigación en el campo de la etiquetación automática. La justificación de este hecho es que el etiquetador sirve como punto de apoyo al analizador sintáctico [Barr y Cohen, 1989] (el etiquetador asigna las etiquetas a las palabras y por ello el *parser* puede trabajar de forma efectiva a su nivel, donde se trabaja con las etiquetas) y, como se deriva de los trabajos de Church, todo problema se puede reducir a un problema de sintaxis.

En este capítulo se presentará el trabajo realizado para conseguir un sistema que, integrado en el proyecto GALENA, logre eliminar de forma estadística las ambigüedades que provocan las palabras al ser tratadas por el módulo de análisis léxico.

La etiquetación se realiza de forma automática. Es el método más natural si tratamos textos de gran tamaño. Para evitar la necesidad de un retoque manual de las etiquetas que proporciona el analizador, cuando nos encontramos ante algoritmos de etiquetación automática que no cuentan con las suficientes evidencias para eliminar ambigüedades, resulta práctico conservar más de una etiqueta por palabra encontrada en el texto. Nuestro analizador léxico siempre etiqueta cada *token* con todas las posibilidades que ofrece, como se expuso en el apartado **4.2**. Ésta es una de sus características más importantes, ya que permite poner de manifiesto las ambigüedades.

Esta estrategia imposibilita el análisis sintáctico de los textos, ya que para construir el árbol de análisis es imprescindible que cada palabra juegue un único papel dentro de la sentencia, esto es, posea una única etiqueta. Surge, por tanto, la necesidad de eliminar las ambigüedades en el proceso de etiquetación dentro de nuestro sistema, para poder construir el árbol de análisis.

El objetivo del presente trabajo, y de un supresor de ambigüedades en general, es proveer de una única etiqueta a cada palabra, asignando de entre todas las posibles etiquetas, la que en cada caso sea más probable según el texto en estudio. Para ello, una opción es realizar un análisis estadístico de textos del mismo estilo literario que el que va a ser tratado, y posteriormente, aplicar ese estudio al texto en cuestión.

El supresor de ambigüedades que proponemos permite la selección, por parte del usuario, de la información léxica a utilizar en la supresión de ambigüedades, y ofrece funcionalidades complementarias para el tratamiento de las matrices de aprendizaje.

Para conseguir estos objetivos, inicialmente se entrena el sistema con textos, denominados textos de aprendizaje, del mismo estilo literario que el texto que se va a tratar. Estos textos son etiquetados por lingüistas, de tal forma que no presenten ambigüedades. El procedimiento que sigue nuestro sistema consiste, básicamente, en obtener de los textos de aprendizaje las frecuencias de aparición de las etiquetas con relación al contexto. Posteriormente, en una segunda fase, esas frecuencias de aprendizaje se aplican al texto en estudio y determinan la etiqueta más probable para cada palabra ambigua. Los resultados de la selección están avalados por la calidad de la fase de aprendizaje.

En consonancia con lo expuesto, el sistema que aquí se presenta consta de dos módulos para lograr una operatividad total. El primero constituye la fase de aprendizaje y el segundo la operacional.

5.2 EL PROBLEMA DE LA ETIQUETACIÓN

Supondremos, para exponer formalmente el problema que se plantea en este trabajo, que el usuario ha definido un conjunto de etiquetas vinculadas a las palabras. El juego de etiquetas que se ha considerado en este proyecto se puede ver en la tabla 2 (pág. 90).

Considérese una secuencia $W = w_1w_2...w_n$, donde cada w_i es una palabra, y una secuencia de etiquetas $T = t_1t_2...t_n$. Llamemos al par (W, T) una alineación (*alignment* para Bernard Merialdo [Merialdo, 1994]). Se dice que a la palabra w_i se le ha asignado la etiqueta t_i en esta alineación.

Se asume que las etiquetas tienen algún significado para el usuario, de forma que entre todas las posibles alineaciones para una secuencia sólo hay una correcta desde el punto de vista gramatical. El objetivo final en la eliminación de ambigüedades es encontrar la alineación correcta haciendo uso, en nuestro caso, de la información adquirida en la fase de aprendizaje.

Un procedimiento de etiquetación (*tagging procedure*) es un procedimiento ϕ que selecciona una secuencia de etiquetas, esto es, define una alineación para la secuencia W dada. La forma de expresarlo formalmente sería:

$$\phi: W \rightarrow T = \phi(W)$$

Hay al menos dos medidas para evaluar la calidad de un procedimiento de etiquetación[23]:

- A nivel de frase:

  $Q_s(\phi)$ = porcentaje de frases correctamente etiquetadas.

- A nivel de palabra:

  $Q_w(\phi)$ = porcentaje de palabras correctamente etiquetadas.

En la práctica $Q_s(\phi)$ es generalmente menor que $Q_w(\phi)$ ya que, para que una frase sea considerada como correctamente etiquetada, todas las palabras que la componen deben estar etiquetadas de forma adecuada. Por esta razón, $Q_s(\phi)$ es una medida poco útil debido a su carácter tan estricto.

El sistema que presentamos se puede definir como un procedimiento de etiquetación, y la medida de calidad que consideraremos es $Q_w(\phi)$, que es la medida estándar usada en la literatura.

[23] Calidad se representa mediante Q, del inglés *quality*.

5.3 REQUERIMIENTOS PARA LA RESOLUCIÓN EFECTIVA DE LA AMBIGÜEDAD

El sistema de supresión de ambigüedades propuesto para GALENA se ha desarrollado teniendo en cuenta que existen cuatro requerimientos principales para un procedimiento efectivo de resolución de ambigüedades. Dichos requerimientos fueron establecidos por Alexander Franz [Franz, 1996] y son los siguientes:

- **Entrenamiento automático**

El primer objetivo es evitar el cuello de botella en la adquisición de conocimiento, que surge cuando se necesita una creación manual de extensas fuentes de conocimiento; situación ésta que se daba en los métodos tradicionales de desambiguación por reglas. Por esta razón se ha optado por un sistema basado en una aproximación estadística, cuyo bagaje de conocimiento se obtiene del estudio automático de las frecuencias de aparición de las etiquetas léxicas en textos previamente etiquetados de forma correcta, esto es, se obtienen por un procedimiento de análisis de los corpus.

Todavía se precisa de la pericia manual para etiquetar esos textos de aprendizaje a partir de los cuales se obtendrán las frecuencias que serán aplicadas con posterioridad para resolver las ambigüedades que se presenten. Sin embargo, el principal escollo, la adquisición de conocimiento, se ha visto sensiblemente reducido.

- **Manejo de múltiples características**

De cara a la obtención de una forma efectiva de resolución de la ambigüedad es necesario permitir combinar múltiples características de desambiguación.

El sistema propuesto no impide en ningún momento combinar la filosofía propia del sistema con otra aproximación que se desee poner en juego, como puede ser el caso de la desambiguación por reglas, por ejemplo. Además se ha desarrollado de tal forma que el usuario tiene, en todo momento, la posibilidad de refinar u obviar detalles de las etiquetas léxicas en la fase de aprendizaje, y por ende en la fase operacional. En la práctica esto significa que el usuario puede variar el número de etiquetas que son consideradas. Esta posibilidad no es contemplada por la literatura pero resulta obvio que permite establecer la granularidad del conocimiento obtenido[24].

- **Características de dependencia de modelización**

En el ámbito del PLN no es suficiente modelizar los principales efectos de un determinado número de características por aislado. El lenguaje natural resulta ser un campo muy complicado y no muy bien entendido, y parece cierto que algunos aspectos del lenguaje no son independientes. Por esta razón es importante elegir una técnica de modelización que permita modelizar explícitamente las interacciones. Este es el caso de los métodos estadísticos.

- **Robustez**

[24] Los expertos lingüistas han mostrado una gran aceptación por esta idea, aunque con las reservas propias de algo que necesita un estudio muy preciso para calibrar su impacto en el funcionamiento del sistema.

Un procedimiento de resolución de ambigüedades efectivo exige ser robusto en dos sentidos:

Primero, no debe limitarse a un dominio específico para el cual ha sido construido y probado, y debe tener una amplia aplicabilidad. El procedimiento presentado en este trabajo está englobado dentro del proyecto GALENA, pero podría perfectamente englobarse en otro sistema sin cambiar su operativa y, por otro lado, no se limita en ningún modo a un determinado dominio del lenguaje natural, ya que el aprendizaje puede realizarse con textos de diversos estilos literarios. Las estadísticas obtenidas serán propias de cada estilo y podrán aplicarse a todos los textos. Lógicamente se aplicarán en la desambiguación de textos que sigan el mismo patrón de estilo que aquellos que se usaron para realizar el aprendizaje, aunque esto no es obligatorio.

En segundo lugar, el procedimiento no puede ser una caja negra monolítica; debe poder compaginarse con otros componentes que puedan compensar algunas de sus inadecuaciones. En nuestro caso será el analizador sintáctico que se está desarrollando para GALENA.

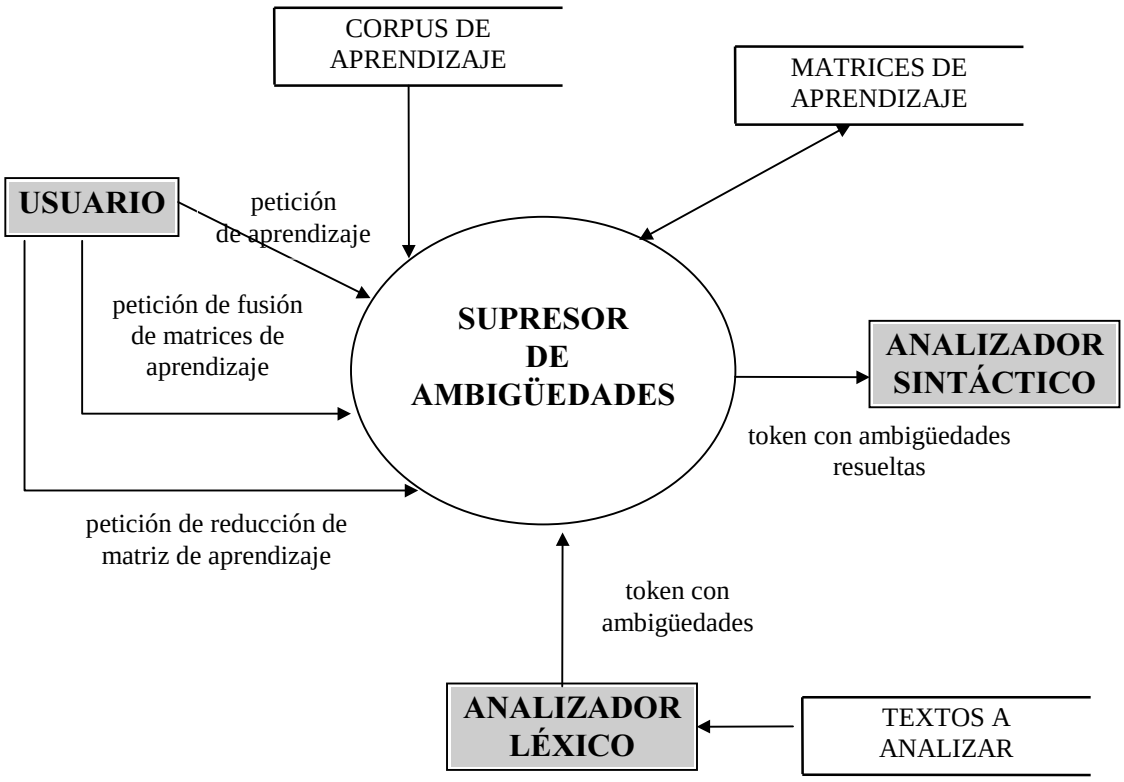


Figura 14. Diagrama de contexto del supresor de ambigüedades[25].

Esto es fácil de entender si se tiene una idea de la situación en la que se localiza la herramienta construida. Esta localización se puede observar en el

[25] Notación inspirada en Yourdon [Yourdon, 1989].

anterior diagrama de contexto (figura 14). La situación reflejada en este diagrama de contexto es la situación que se busca cuando todo el sistema esté integrado. En la actualidad el supresor de ambigüedades funciona conjuntamente con el analizador léxico, formando un etiquetador. Sin embargo, aún funciona separadamente del analizador sintáctico, a la espera de que éste lo integre en su operativa.

5.4 ELECCIÓN DE LA APROXIMACIÓN CONSIDERADA

La resolución de ambigüedades está considerada como el mayor problema del análisis del lenguaje natural [Franz, 1996], y se ha realizado un enorme esfuerzo para lograr un esquema de resolución de ambigüedades eficiente.

Un esquema de resolución de ambigüedades requiere una fuente de información para determinar qué elecciones son más probables, y por tanto nos ayude a suprimir las ambigüedades en el proceso de análisis de la sentencia.

Algunos de los esquemas de resolución de ambigüedades se apoyan en fuentes de conocimiento muy elaboradas. La experiencia ha demostrado que esas estrictas restricciones sintácticas y las reglas de preferencia semánticas son difíciles de conseguir en análisis de amplia cobertura. Además, las fuentes de conocimiento semántico o pragmático pueden requerir enormes esfuerzos de desarrollo manual y pueden ser difíciles, a la vez que lentos, de aplicar en tiempos de análisis. Por esta razón, este tipo de información de supresión de ambigüedades es difícil de proporcionar, a no ser que se encuentren métodos de compilación y de adquisición del conocimiento eficientes que eviten el cuello de botella en la adquisición del mismo.

Estos métodos se han revelado muy frágiles si se usan fuera de dominios bien definidos, lo que hace que su aplicación sea menos generalizada. Se ha probado la dificultad de lograr análisis robustos en el ámbito del lenguaje natural haciendo uso de aproximaciones tradicionales que se apoyan en restricciones sintácticas codificadas a mano, reglas semánticas de dominios específicos y reglas pragmáticas de cualquier dominio [Franz, 1996].

Los modelos basados en la estadística tienen la ventaja de ser robustos, ya que proporcionan una medida de generalización más allá de los datos de entrenamiento, y generalmente exhiben una degradación que Alexander Franz califica de elegante. Esta filosofía permite el entrenamiento automático de cara a la estimación del modelo. Por ello, si se encuentran disponibles los datos de entrenamiento adecuados, se puede evitar el cuello de botella de la adquisición manual de conocimiento.

Para la eliminación de ambigüedades, en nuestro sistema, se han manejado métodos estadísticos. Más adelante, se considerará la posibilidad de utilizar reglas de desambiguación. El método actual de supresión de ambigüedades guiará, de alguna forma, al que se desarrolle posteriormente.

En el pasado se ha dedicado una gran cantidad de esfuerzos al problema de la etiquetación de textos. En líneas generales, y a grandes rasgos, recordemos que ha habido dos aproximaciones (apartado **3.5**):

- Basada en reglas (Klein y Simmons 1963; Brodda 1982; Paulussen y Martin 1992; Brill et al. 1990);

- Probabilística (Bahl y Mercer 1976; Debili 1977; Stolz, Tannenbaum y Carstensen 1965; Marshall 1983; Leech, Garside y Atwell 1983; Derouault y Merialdo 1986; DeRose 1988; Church 1989; Beale 1988; Marcken 1990; Merialdo l991; Cutting et al.).

Hoy en día se afirma con frecuencia, y es comúnmente aceptado [Aldezabal et al., 1996], que la segunda aproximación es preferible. Esta opinión se debe al coste que tiene escribir "a mano" un conjunto de reglas con la cobertura suficiente para obtener resultados que igualen los de los sistemas estadísticos. Estos, evitando el "trabajo manual" llegan a obtener unos resultados de hasta un 96% de acierto. Sin embargo, el etiquetador de Brill constituye un contraejemplo: a partir de un conjunto de patrones de reglas escritos a mano, este sistema extrae automáticamente las instancias de las reglas y obtiene resultados tan buenos como los de un etiquetador estadístico. Otro ejemplo que contradice la afirmación anterior es el sistema diseñado por Voutilainen [Voutilainen y Tapanainen, 1993]. Está basado en el formalismo sintáctico *Constraint Grammar* y obtiene un resultado del 99% para el inglés. Atro Voultilainen, para conseguir este nivel de resultados, diseñó a mano aproximadamente mil reglas, lo cual le supuso cerca de un año de trabajo.

Utilizando un supresor de ambigüedades que opere con reglas, podemos encontrarnos con situaciones en que son aplicables varias de ellas. Para solventar este tipo de situaciones, es necesario asignar prioridades a las reglas de forma que la selección pueda hacerse incluso cuando se pueda aplicar más de una regla. La desventaja de esta solución es doble. Por una parte, complica el algoritmo y, por otra, surge el problema de cómo asignar las prioridades a las reglas para lograr una operatividad final.

Actualmente la tendencia mayoritaria es tomar como punto de partida los resultados de un desambiguador estadístico e intentar reducir el 4% de error subsistente[26], mediante información lingüística. Esta es la filosofía que se sigue dentro del proyecto en el que se enmarca esta tesis de licenciatura. Por ello, en el futuro se desarrollará un sistema que, valiéndose del conocimiento aportado por el supresor de ambigüedades actual, lo mejore.

Recientemente, se han propuesto trabajos que usan redes de neuronas artificiales (Benello, Mackie y Anderson 1989; Nakamura y Shikano 1989). Todo el proceso que se ha descrito aquí encaja con la forma de operar de las redes neuronales. Sin embargo, aún es pronto para plantearnos el desarrollo de un módulo de esas características mientras no podamos asegurar un buen rendimiento con la utilización de estas técnicas.

5.4.1 SELECCIÓN DEL TAMAÑO DE LA VENTANA TEMPORAL

Church y Mercer [Church y Mercer, 1993] afirman que la etiqueta de un *token* se puede estimar teniendo en cuenta sólo las etiquetas de los dos *tokens* anteriores, debido a que el cálculo en base a los n *tokens* anteriores es demasiado complejo. En el caso del inglés, esta aproximación parece suficiente; se puede asegurar que la etiqueta de un *token* depende únicamente de las dos anteriores. Pese a que no está demostrado para otros idiomas, como por ejemplo las lenguas latinas, esa aproximación ha servido de base para muchos trabajos, entre los cuales se encuentra el nuestro.

[26] Con métodos puramente estadísticos, no se ha logrado superar el 96-97% de aciertos.

En la mayoría de los sistemas se utilizan ventanas de un único *token*, bigramas, puesto que la cantidad de datos que hay que almacenar es considerablemente menor. En ocasiones, se utilizan ventanas de dos *tokens*, que se denominan trigramas o trietiquetas[27]. Podríamos pensar en ampliar la ventana temporal; sin embargo, se han realizado experimentos utilizando ventanas mayores y la mejoría de los resultados no compensa la cantidad de datos que hay que almacenar y utilizar. Además, para hacer uso de 4-gramas, o n-gramas en general, en el aprendizaje, el tamaño del corpus debería ser muchísimo mayor que el necesario para obtener resultados aceptables con bigramas o trigramas. Con una ventana de dos *tokens*, la cantidad de datos que es necesaria entra dentro de los límites de lo razonable. No ocurre lo mismo con ventanas de tres o más *tokens*.

En nuestro sistema se han elegido las trietiquetas, que proporcionan una información útil para la supresión de ambigüedades sin imponer un precio excesivo en el almacenamiento y manejo de datos.

5.4.2 SELECCIÓN DE LA OPERATIVA DE TRABAJO

El sistema propuesto en este trabajo se encuadra dentro de los sistemas denominados de etiquetación basados en frecuencias. Se usan trigramas, como se ha mencionado en el apartado anterior, en vez de bigramas, como en el sistema CLAWS (apartado **3.5**), para obtener una mayor cantidad de información. El proceso de selección es similar al de CLAWS, ya que dada una ventana temporal histórica de dos *tokens*, anteriores por tanto a la palabra objetivo, se seleccionará la etiqueta de la tercera palabra del trigrama que más probable sea, en función de las estadísticas adquiridas en una fase previa de aprendizaje.

Las fórmulas presentadas en relación al sistema CLAWS se han extendido de forma natural de la siguiente manera (fórmula 1, pág. 62):

$$P(t_i, t_{i+1}, t_{i+2}) \approx \frac{f(t_i, t_{i+1}, t_{i+2})}{f(t_i, t_{i+1})}$$

El sistema propuesto, internamente y para mejorar el tiempo de respuesta, no realiza la computación anterior. La expresión anterior[28] se podría seguir desarrollando de la siguiente manera:

$$P(t_i, t_{i+1}, t_{i+2}) \approx \frac{\frac{n(t_i, t_{i+1}, t_{i+2})}{n_1}}{\frac{n(t_i, t_{i+1})}{n_2}} = \frac{n(t_i, t_{i+1}, t_{i+2})}{n(t_i, t_{i+1})} \times \frac{n_2}{n_1}$$

donde n(...) indica el número de observaciones de la secuencia entre paréntesis y n_1 y n_2 representan las observaciones totales de trigramas y bigramas respectivamente.

[27] No se incluye el *token* actual.
[28] Haciendo uso de la definición de frecuencia.

Ya que $\frac{n_2}{n_1}$ es un coeficiente invariable que multiplica a todas las probabilidades se puede descartar puesto que, aunque se desvirtúa la distribución estadística, la relación de orden de precedencia entre las probabilidades no varía.

Dado que en nuestro sistema lo que se busca es indicar al analizador sintáctico qué etiqueta, para una palabra, es la más probable y por tanto la que primero debe tomar[29]para intentar realizar el árbol de análisis, nos llega con establecer un orden de preferencia ante las diferentes alternativas. Evidentemente este orden de preferencia estará establecido en relación directa con la probabilidad obtenida por la fórmula anterior. Sin embargo, para llevar a cabo los cálculos y establecer el orden de prioridad, el denominador de la última fórmula se puede obviar ya que el único efecto que tiene es el de normalizar. Por esta razón con sólo saber el número de veces que se ha dado en el aprendizaje la situación (t_i, t_{i+1}, t_{i+2}) tenemos información suficiente para indicar si la etiqueta t_{i+2} es la más factible para asignar a la palabra en la que estamos intentando eliminar la ambigüedad, o es otra, t_k, ya que el trigrama (t_i, t_{i+1}, t_k) aparece un número de veces superior.

En definitiva, realmente con lo que se trabaja en el sistema desarrollado es con las frecuencias absolutas de aparición de las etiquetas, las cuales han sido obtenidas en una fase previa de aprendizaje.

En [Lin, Y.-C. et al., 1995] se muestra que los modelos probabilísticos aplicados en PLN tienen, muy frecuentemente, una gran cantidad de parámetros. Por este motivo, los parámetros necesitan ser relajados para reducir el error de estimación debido a la insuficiencia de datos de entrenamiento. Es más, los modelos probabilísticos son generalmente simplificados para hacer factible la estimación de los parámetros del modelo. Consecuentemente alguna información discriminante podría sacrificarse al hacer suposiciones para simplificar un modelo probabilístico.

Para evitar estos problemas se ha optado por una aproximación en la que el modelo utilizado se basa, sencillamente, en las frecuencias absolutas de aparición para estimar las probabilidades; además la filosofía de trabajo elegida se encuentra también apoyada por el hecho de que en el analizador sintáctico se posee un *token* de *lookahead.* Esto también facilitaría restringir las posibilidades de etiquetación de una palabra, esta vez basándose en la gramática establecida en el analizador sintáctico. Además de las frecuencias históricas con las que trabaja el sistema actual existiría, por tanto, la posibilidad de incorporar un conocimiento añadido que hace referencia a lo que viene después, en vez de hacer referencia a lo que vino, que es lo que actualmente se considera. La idea sería combinar conocimiento sintáctico, adquirido con los lingüistas, con conocimiento estadístico, adquirido por el estudio de textos; sin embargo, el prototipo desarrollado aún no lo hace.

5.5 FASE DE APRENDIZAJE

En esta fase se parte de un texto sin ambigüedades léxicas, resultado del trabajo de un equipo de lingüistas, y a partir de este texto se elaboran las estadísticas que sirven para suprimir las ambigüedades que puedan aparecer en la etiquetación de una palabra.

[29] Realmente se le indica un orden de prioridades para las diversas alternativas existentes para una palabra.

5.5.1 OBJETIVO DE LA FASE DE APRENDIZAJE

Como resultado de la fase de aprendizaje se crea una matriz que refleja las situaciones que se han producido en el texto [Andrade et al., 1997]. La estructura creada debe ser una matriz debido a que la historia del texto se mantiene en una ventana temporal de dos *tokens*. Las dimensiones de la matriz se corresponden con cada uno de los *tokens*.

En cada acceso a la matriz pueden presentarse distintas posibilidades para la misma ventana temporal que fija el acceso a la matriz; la situación actual puede ser distinta de las anteriormente aprendidas, o bien puede repetirse. Esto obliga a mantener una dimensión más en la matriz de aprendizaje, que refleja todas las situaciones aprendidas, así como el número de veces que han aparecido, posibilitando la fase operativa posterior basada en esas estadísticas. Esta dimensión a mayores la denominaremos en lo sucesivo **tercera dimensión**.

Al finalizar el proceso, se generará un fichero que contendrá la matriz de aprendizaje. Esto posibilita:

- Que se puedan generar matrices de aprendizaje para los diferentes estilos literarios que se deseen tener en cuenta, y
- Que realizando el proceso de aprendizaje una única vez, la matriz se pueda utilizar en diferentes momentos de la fase operacional.

El código desarrollado para esta fase permite combinar varios textos en los que ya no se presentan ambigüedades, para generar una sola matriz de aprendizaje. De esta forma se consigue:

- No tener que almacenar todo un corpus en un único fichero[30], y
- Poder reflejar, en una única matriz, el aprendizaje aportado por varios textos que posean estilos literarios similares.

La forma de tratar los diferentes ficheros es idéntica a la forma en que se trataría un único fichero que resultase de la concatenación de todos los anteriores en el orden en que se proporcionan. Esta facilidad resulta muy interesante en el caso de encontrarnos en alguna de las dos situaciones enumeradas en el párrafo anterior.

5.5.2 DESARROLLO DEL SISTEMA

5.5.2.1 PROTOTIPO

Para desarrollar el código necesario en el aprendizaje, inicialmente se utilizó el lenguaje de reconocimiento de patrones Gawk [Kernighan y Pike, 1987] y el lenguaje de programación C [Kernighan y Ritchie, 1991]. La situación de esta primera versión de la fase de aprendizaje, construida con el concepto de prototipo en mente, se refleja en el esquema propuesto en la figura 16 (pág. 111), sin embargo, antes de continuar es

[30] El aprendizaje debe estar avalado por el estudio de una amplia casuística, lo cual significa manejar textos de aprendizaje muy grandes.

conveniente aclarar ciertos aspectos relacionados con la creación de la matriz de aprendizaje:

- La definición de la ventana temporal supone mantener la historia correspondiente a los dos *tokens* anteriores y es la siguiente:

```
struct windowtype       {
                                int beforelast;
                                int last;
                        };

struct windowtype window;
```

La definición de la ventana temporal se hace con enteros, esto hace que deba de existir una función que permita pasar de una etiqueta (de la información léxica) a un entero, que será el reflejo de toda la información contenida en la etiqueta. Este número servirá para acceder a la matriz tanto en la fase operacional como en la fase de aprendizaje. También será lo que aparezca en las matrices de aprendizaje ya que realmente es el número que resulta de la traducción de las etiquetas léxicas, esto es, es su código.

Esta función, biyectiva, se podría describir de la siguiente forma, donde T es el conjunto de etiquetas y N representa en conjunto de los números formado por {1, 2, 3, ..., n}, donde n es el número de etiquetas posibles con la configuración considerada para la desambiguación:

$$\tau: T \rightarrow N = \tau(T)$$

La implementación de esta función inicialmente se realizó a través de un fichero denominado ETICONF.MTZ (para más detalle ver pág. 112), con posterioridad se consiguió a través de un switch, el cual considera todas las elecciones referidas a las partes de las etiquetas que se desean considerar en la desambiguación. Precisamente este código es el que planteaba mayores problemas si se deseaba cambiar la información léxica considerada (configuración) en la desambiguación, ya que el código había que cambiarlo a mano. En el código esta función τ recibe el nombre de GetIndex() y se verá en más detalle en este mismo apartado.

Con la intención de facilitar la operativa con el sistema, se hace una comprobación para verificar que la configuración actual es compatible con la que se utilizó para construir la matriz de aprendizaje que se utiliza para la desambiguación. Esta comprobación resulta en verificar que el conjunto N de la función τ es el mismo para ambas configuraciones.

- Inicialmente, para arrancar el proceso, la ventana temporal posee los siguientes valores de inicialización:

```
window.beforelast=0;
window.last=0;
```

Por esta razón, si se consideran n etiquetas para operar con el desambiguador, obtendremos una matriz de orden $(n+1)\times(n+1)$.

- La forma de desplazar (refrescar) la ventana temporal, una vez que ya se ha determinado la etiqueta correcta para una palabra, es la siguiente:

```
RefreshTemporalWindow(GetIndex(actual_token))
```

Siendo la definición de la función RefreshTemporalWindow():

```
void RefreshTemporalWindow (int label)
{
  window.beforelast=window.last;
  window.last=label;
}
```

- Las dos dimensiones principales de una matriz de aprendizaje son estáticas en el sentido de que no varían en tamaño a lo largo de un aprendizaje con una configuración dada. Es la configuración elegida la que determina el número de etiquetas que se van a considerar, y es este número el que determina el tamaño de estas dos dimensiones; razón por la cual no varía su tamaño ante una determinada configuración. Sin embargo, hay una tercera dimensión, dinámica, que sí varía de tamaño al avanzar el aprendizaje. Esto se debe a que es la dimensión que refleja las situaciones producidas en el texto. El tamaño de esta tercera dimensión varía para cada posición (i, j) de la matriz, ya que refleja las situaciones acaecidas para las que la forma de los trigramas es (i, j, k).

- La definición del tipo de la matriz en C se corresponde con las siguientes líneas de código:

```
struct no_ma    {
                        int labelno_ma;
                        struct no_ma * ptr;
                };
struct no_ma ***matrix;
```

Esta definición está asociada con la idea presentada en la figura 15:

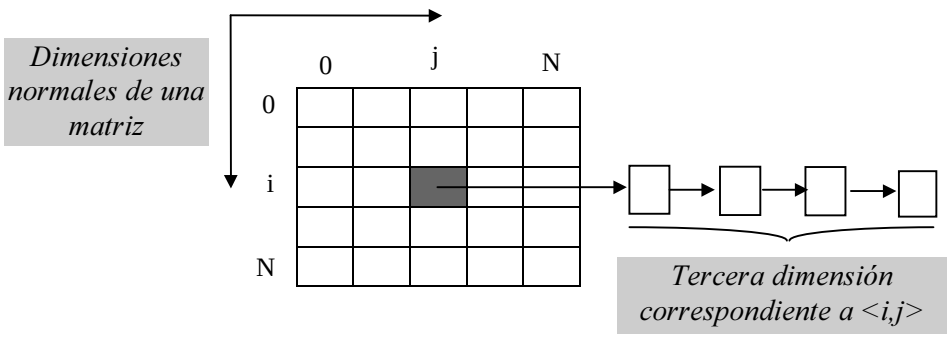


Figura 15. Conceptualización de la matriz de aprendizaje.

- La matriz, una vez creada, se almacenará en un fichero nombrado por el usuario. El almacenamiento se llevará a cabo por filas y la tercera dimensión se almacenará ya ordenada para favorecer la posterior operativa.

- La lectura de un fichero que contiene una matriz de aprendizaje es sencilla:

  (1) Hay una cabecera en la primera línea que indica el número de etiquetas consideradas en la desambiguación. Este dato es necesario más adelante para comprobar, en la fase operacional, si estamos trabajando con una matriz originada con la misma configuración que la que en la fase operacional se considera.

  (2) Posteriormente se procederá a almacenar la matriz por filas finalizando cada tercera dimensión por un `NULL`[31].

  (3) Cada elemento de la tercera dimensión es un par (label, times) donde el primer elemento del par es el código de la etiqueta y el segundo es el número de veces que con la historia (i, j), considerando que estamos almacenando la información correspondiente a la posición (i, j) de la matriz, apareció esa etiqueta.

  (4) Los elementos de las terceras dimensiones ya se almacenan en el fichero ordenados por el segundo ítem del par, esto es, times. En la fase operacional este dato ya no hace falta puesto que la información se encuentra ordenada, sin embargo sí es necesaria, y por eso se mantiene, para realizar las operaciones de *fusión* y *reducción* de matrices.

  Un ejemplo de una parte de una matriz de aprendizaje podría ser el siguiente:

```
NUM_LABELS,370
232     1
N
…
N
N
88      2
56      1
252      1
N
N
N
N
370      1
N
N
N
N
…
```

[31] En la codificación este `NULL` se abrevia por una `N`.

En esta porción de una matriz de aprendizaje cualquiera podemos observar como la primera línea indica que se han considerado 370 posibles etiquetas para la configuración del desambiguador. Con la ventana temporal (0,0), correspondiente a la primera posición de la primera fila de la matriz, ha aparecido una palabra cuya etiqueta se corresponde con el número 232, esto es, la primera palabra que aparece en el aprendizaje tiene por $\tau(T)$ el número 232. También podemos observar en el ejemplo que para una ventana temporal dada hay una tercera dimensión que está compuesta por las etiquetas cuyos códigos son 88, 56 y 252. La etiqueta 88 apareció dos veces precedida de las dos mismas etiquetas, es decir, con la misma ventana temporal.

Una vez comentados estos aspectos estamos en situación de mostrar el esquema prometido de la fase de aprendizaje (figura 16).

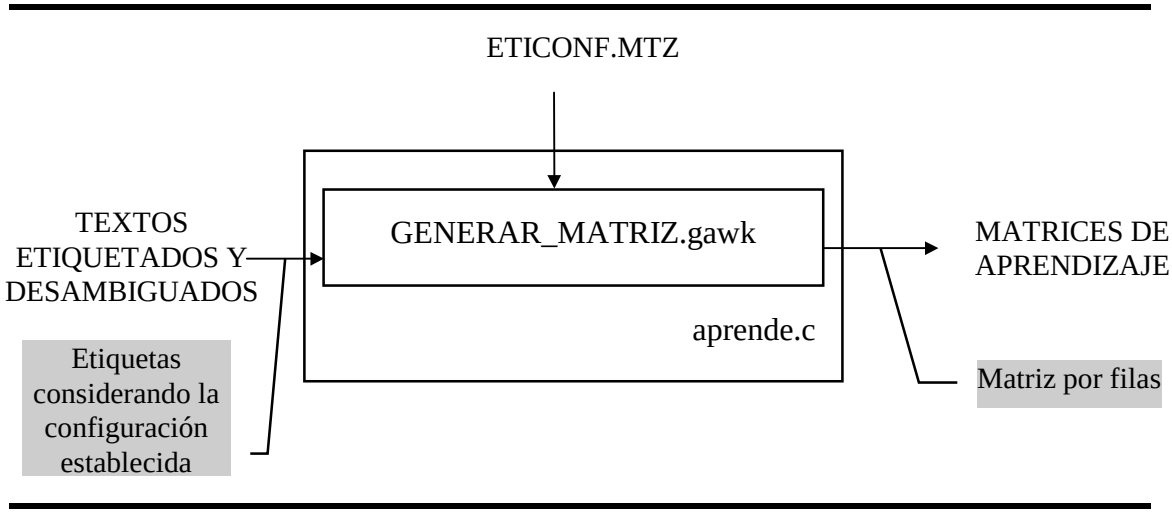


Figura 16. Esquema de la fase de aprendizaje.

Como se refleja en el esquema anterior se parte de textos de aprendizaje ya etiquetados y desambiguados por expertos lingüistas. El hecho de que estén desambiguados simplemente quiere decir que se encuentra marcada la etiqueta correcta para cada palabra del texto. De estos textos se extraen las etiquetas marcadas para cada palabra, obviando las demás, para así conocer el discurrir correcto del texto, en lo que a secuencia de etiquetas se refiere.

El código Gawk denominado GENERAR_MATRIZ.gawk lee un fichero de configuración, ETICONF.MTZ, que le indica el número de etiquetas consideradas con el que opera el desambiguador y especifica cada una de ellas, así como un código asignado a cada una de ellas. Es decir, se incorpora en un fichero el resultado de la función τ descrita anteriormente. Una muestra de este fichero de configuración es el siguiente:

```
NUM_ETIQUETAS, 47
Sustantivo comun, 1
Sustantivo propio, 2
Adjetivo, 3
Demostrativo, 4
Relativo, 5
Indefinido, 6
Interrogativo, 7
Posesivo, 8
Numeral cardinal, 9
Numeral ordinal, 10
Numeral partitivo, 11
Numeral multiplo, 12
Articulo, 13
Pronombre Personal tonico, 14
Pronombre Personal proclitico atono, 15
Pronombre Personal enclitico atono, 16
Verbo, 17
Preposicion, 18
Conjuncion coordinada, 19
Conjuncion subordinada, 20
Adverbio nuclear, 21
     .
     .
     .
```

Una vez leído este fichero, el código Gawk procede a leer las etiquetas marcadas del texto de aprendizaje (corpus) y a construir la matriz de aprendizaje en memoria, manteniendo en todo momento actualizada la ventana temporal. Una vez construida la matriz correspondiente a un texto se procederá a almacenarla en disco con el nombre que el usuario desee.

Un código C, aprende.c, es la interfaz de usuario que invoca al código Gawk y renombra la matriz, entre otras operaciones.

El código Gawk es tan sencillo como sigue:

```
# ! /usr/bin/gawk -f GENERAR_MATRIZ.gawk ETICONF.MTZ {ficheros_desambiguados}

FILENAME != prevfile    {
                  NR = 1
                  prevfile = FILENAME
                  }

BEGIN             {
                  system("clear")
                  printf"\n\n\tProcesando...\n\n\n"
                  FS = ", "
                  anteanterior = 0
                  anterior = 0
                  }

FILENAME ~/.*ETICONF.MTZ/ && NR==1 && $1=="NUM_ETIQUETAS"  {
                  cont = $2 + 0
                  printf"NUM_ETIQUETAS,%d\n",cont > matriz.gln
                  close("matriz.gln")
                  limite = cont + 1  #  EOF del fichero de configuracion
                  }

(NR > 1 && (NR <= limite)) && FILENAME ~/.*ETICONF.MTZ/
                  {
                  etiquetas[$1] = $2
                  }

# A continuacion se procesan los ficheros desambiguados y se construye la
# matriz

((( $1 ~/^=> \*\[.*/ ) || ($1 ~/^   \*\[.*/ )) && (FILENAME
!~/.*ETICONF.MTZ/) && ($1 !~/^$/)) {
                  presente = etiquetas[$3] + 0
                  matriz[anteanterior"."anterior"."presente]++
                  anteanterior = anterior
                  anterior = presente
                  }

END               {

      for (i=0 ; i<=cont ; i++)
         for (j=0 ; j<=cont ; j++)
                     {
                        for (k=1 ; k<=cont ; k++)
                              if (matriz[i"."j"."k] != 0)
                                 printf"%d %5d\n",k,matriz[i"."j"."k]
                         | "sort -r +1 -2 >> matriz.gln"

                         printf"NULL\n" | "sort -r +1 -2 >> matriz.gln"
                         close ("sort -r +1 -2 >> matriz.gln")
                     }
                  }
```

En esta muestra de código se puede observar el carácter críptico del lenguaje Gawk a la vez que su potencia.

A pesar de su correcto funcionamiento surgen inconvenientes de lentitud e incomodidad:

- Los tiempos obtenidos en la generación de matrices de aprendizaje resultan bastante elevados, como se puede apreciar en los resultados presentados más adelante (tabla 3, pág. 118).
- Con el código Gawk se necesita una lista de la información léxica que el desambiguador debe de considerar. Esta lista se escribe en un fichero de

configuración que permite cierta flexibilidad, pero a cambio redunda en un tedioso trabajo manual añadido en el caso de tener que crear el fichero de configuración a mano, o bien tener que retocarlo en el caso de que la configuración se cambie.

- Otro inconveniente añadido está en el hecho de que si se desea tener en cuenta más o menos información de las etiquetas léxicas, no sólo hay que cambiar ETICONF.MTZ, sino que hay que modificar varios códigos, entre los que está el código que permite generar los ficheros de los textos de aprendizaje etiquetados. Esto es debido a que Gawk trabaja con marcas que hay que cambiar de lugar en los textos etiquetados.

5.5.2.2 LA VERSIÓN ACTUAL

En la búsqueda de una solución a los problemas detectados nos pareció conveniente que el usuario pudiese decidir, sin ningún tipo de restricción ni cambio manual en los códigos existentes, qué campos de las etiquetas léxicas quisiera considerar en el proceso de supresión de ambigüedades definiendo así qué etiquetas considerar. Permitimos así al usuario que determine qué información le interesa tener en cuenta para llevar a cabo el proceso, en lugar de preestablecerla. Se puede decir que ésta es la principal motivación: lograr, dentro de lo posible, una cierta flexibilidad frente a la rigidez que cualquier juego de etiquetas establece inherentemente.

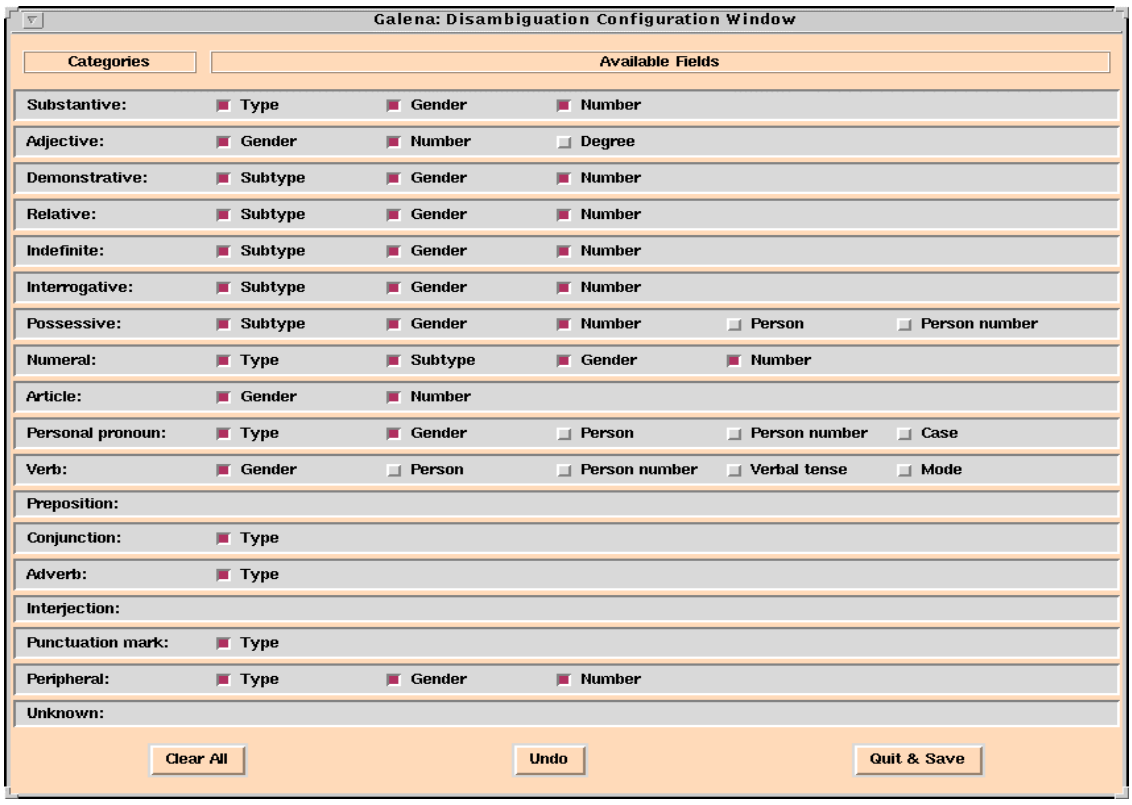


Figura 17: Ventana de configuración del módulo de supresión de ambigüedades.

Esta funcionalidad resulta además muy útil para controlar la cantidad de datos que el sistema debe almacenar. En el caso de considerar toda la información posible, la matriz de aprendizaje tendrá un tamaño aproximado de 5700×5700 posiciones, sin tener en cuenta la tercera dimensión, que es variable y depende del volumen del corpus utilizado para el aprendizaje. Una matriz de tales proporciones resulta poco conveniente.

En relación a las etiquetas léxicas a considerar en la supresión de ambigüedades léxicas, hemos comprobado que seleccionando sólo la información de Category y Type, la desambiguación no hila tan fino como cabría esperar. Por ejemplo, en los casos en que hay que discernir entre varias etiquetas de una misma palabra, de las cuales varias (o todas) corresponden a la categoría Verb, lo interesante sería tener en cuenta los campos Person, Verbal tense y Mode. También se podrían considerar Gender y Number en las categorías Personal pronoun y Substantive para que en el proceso de desambiguación se tenga en cuenta la concordancia que debe existir entre género y número.

Una vez que el usuario decida que información quiere considerar, se genera automáticamente el código C adecuado para permitir la desambiguación. Todo esto se realiza a través de una interfaz gráfica, creada con Tcl/Tk [Ousterhout, 1994], donde el usuario puede, de forma sencilla e intuitiva, tomar sus decisiones (figura 17). Acto seguido, se genera en el sistema, de forma totalmente transparente, un fichero de configuración, que será tratado con Flex y Bison[32] para generar el código C que mencionamos.

El fichero de configuración al que nos acabamos de referir no sería necesario si se usase una herramienta que permitiese generar interfaces gráficas con la posibilidad de integrar fácilmente código C. Hemos decidido, sin embargo, usar las herramientas descritas para no limitar nuestro sistema al uso de estaciones gráficas. Con este fichero mantenemos la total operatividad del sistema aún en el caso de poseer sólo terminales de texto. Basta con editarlo y colocar un "-" a continuación de la información que no deseamos.

Un ejemplo de este fichero de configuración, denominado disambiguation.ini, es el siguiente:

```
Substantive: Type, Gender, Number
Adjective: Gender, Number, Degree-
Demonstrative: Subtype, Gender, Number
Relative: Subtype, Gender, Number
Indefinite: Subtype, Gender, Number
Interrogative: Subtype, Gender, Number
Possessive: Subtype, Gender, Number, Person-, Person number-
Numeral: Type, Subtype, Gender, Number
Article: Gender, Number
Personal pronoun: Type, Gender, Person-, Person number-, Case-
Verb: Gender, Person-, Person number-, Verbal tense-, Mode-
Preposition:
Conjunction: Type
Adverb: Type
Interjection:
Punctuation mark: Type
Peripheral: Type, Gender, Number
Unknown:
```

En este fichero se recogen todos los posibles accidentes léxicos para una categoría léxica dada. Se puede observar cómo a continuación de algunos de estos

[32] Herramientas de GNU equivalentes a Lex y Yacc.

accidentes existe un "-" indicando que en la ventana de configuración del desambiguador no se han seleccionado, es decir, no se desea tener en cuenta esta información en el módulo estadístico. La información restante es la que establece las etiquetas léxicas consideradas en la desambiguación. Este fichero se corresponde con la ventana de configuración presentada en la figura 17.

Este fichero es leído por un programa, creado usando Flex, Bison y C, que genera un case, denominado switch en lenguaje C, que permite obviar el fichero ETICONF.MTZ ya que los códigos de las etiquetas, que sirven para acceder a la matriz, son asignadas ahora de forma automática, en vez de a mano como se hacía al generar el fichero ETICONF.MTZ.

Un ejemplo de parte de este código generado de forma automática se puede observar a continuación:

```
   /*################################ BE CAREFUL ###################################*/
   /*     THIS FILE WAS GENERATED AUTOMATICALLY. ALL CHANGES WILL BE DISCARD      */
   /*###############################################################################*/

int GetIndex (struct ice_lex_object * tkn)

/* Get the matrix index number. This number is a correlative number beginning in 1 */
{
switch(CAR_INT_LIST(tkn->category))
      {
            case SUSTANTIVE : switch (CAR_INT_LIST(tkn->type))
                              {
                                    case COMMON : return(1);
                                    case PROPER : return(2);
                                    default : printf(" Function
                                    GetIndex :Label not exists\n");
                                    exit();
                              }
            case ADJECTIVE : return(3);
            case DEMONSTRATIVE : return(4);
            case RELATIVE : return(5);
            case INDEFINITE : return(6);
            case INTERROGATIVE : return (7);
            case POSSESSIVE : return(8);
            case NUMERAL : switch (CAR_INT_LIST(tkn->type))
                              {
                                    case CARDINAL : return(9);
                                    case ORDINAL : return(10);
                                    case PARTITIVE : return(11);
                                    case MULTIPLE : return(12);
                                    default : printf(" Function
                                    GetIndex :Label not exists\n");
                                    exit();
                              }
            case ARTICLE : return(13);
            case PERSONAL_PRONOUN : switch (CAR_INT_LIST(tkn->type))
                              {
                                    case TONIC : return(14);
                                    case PROCLITIC_ATONIC :
                                    return(15);
                                    case ENCLITIC_ATONIC : return(16);
                                    default : printf(" Function
                                    GetIndex :Label not exists\n");
                                    exit();
                              }
            case VERB :
            case VERB_SER :
            case VERB_HABER : return(17);
            case PREPOSITION : return(18);
            case CONJUNCTION : switch (CAR_INT_LIST(tkn->type))
                              {
                                    case COORDINATE : return(19);
                                    ...
```

Para generar este switch se hace uso de una serie de estructuras internas que luego se recorren para originar el fichero desambswitch.c que lo contiene.

Este switch no sólo sirve para la fase de aprendizaje sino también para la fase operacional:

- En la fase de aprendizaje se utiliza porque, en la nueva versión, la etiqueta marcada como correcta para una palabra se usa para regenerar la variable token (pág. 66). Al regenerar la variable token a partir de una etiqueta podemos aplicar el switch[33] para obtener su código y llevar a cabo el aprendizaje. Este rodeo no era necesario en la versión que usaba Gawk debido a sus prestaciones.

La regeneración es un proceso idéntico sea cual sea la configuración elegida. Es en el momento de la aplicación del switch donde se restringe la etiqueta a la porción de información que el usuario decida. La aplicación del switch es sinónimo de aplicar la función int GetIndex(struct ice_lex_object * tkn) que lo contiene.

- En la fase operacional se emplea para obtener el índice, de acceso a la matriz, de una determinada etiqueta. Pero en este caso la etapa de regeneración no es necesaria ya que trabajamos al mismo tiempo que se van obteniendo todas las posibles etiquetas léxicas, y entonces la variable token ya posee en los valores de sus campos esas etiquetas.

En definitiva, la nueva versión permite la generación automática del código susceptible de ser modificado debido al cambio de las configuraciones, a la vez que mejora el tiempo de creación de las matrices de aprendizaje.

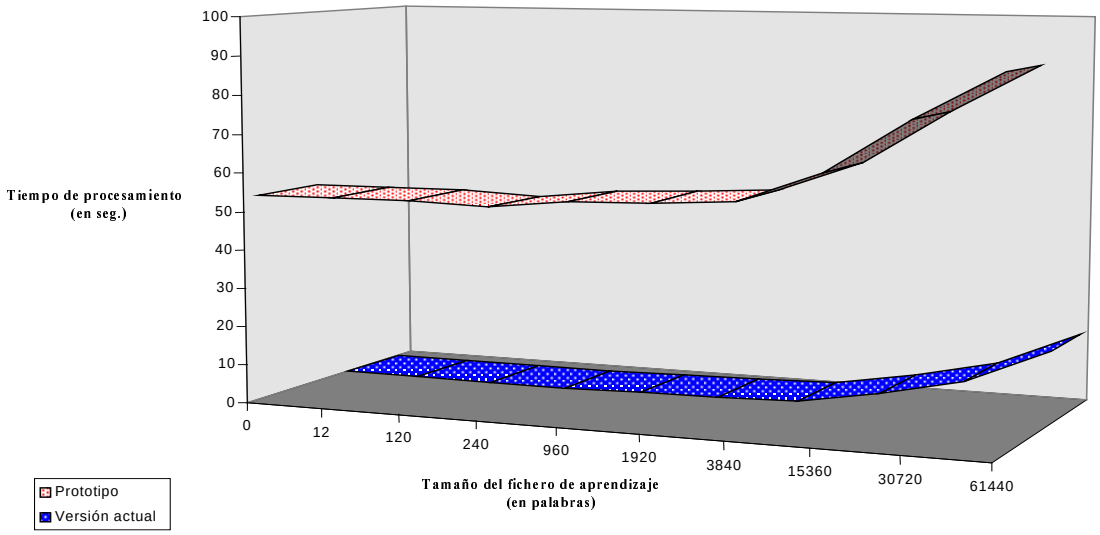
5.5.3 TIEMPOS DE APRENDIZAJE

Los resultados que se presentan a continuación han sido obtenidos con el comando time de Unix en una SUN SPARC STATION 10 bajo SUNOS 4.1.3 y reflejan el tiempo de generación de la matriz de aprendizaje, contrastando el prototipo que utiliza Gawk y C con la versión que hace uso de Tcl/Tk, Flex, Bison y C.

Se puede apreciar, los tiempos se han reducido de tal forma que, en la actualidad, la generación de la matriz sigue una función linealmente creciente frente al tamaño del texto de aprendizaje.

La gráfica 1 muestra la comparación gráfica de los datos presentados en la tabla 3. En ella se puede apreciar que la curva es mucho menos pronunciada en el *Versión actual* que no en el *Prototipo,* así como que el origen de la curva se sitúa en el origen de coordenadas. En la versión inicial el origen de la curva no se sitúa en el origen de coordenadas porque, aunque no haya nada que procesar, el código Gawk toma un tiempo de ejecución inherente.

[33] Nótese el tipo del argumento de la función que lo implementa.



Gráfica 1. Resultados de la fase de aprendizaje.

Tamaño de fichero de aprendizaje		**Prototipo**	**Versión actual**
en palabras	*en bytes aprox.*	*Tiempo de usuario en seg.*	*tiempo de usuario en seg.*
Sin texto	0	53.5	0.0
12	1400	53.9	0.0
120	14000	54.0	0.0
240	28000	53.2	0.1
960	111900	55.5	0.3
1920	223700	56.1	0.6
3840	447400	57.5	1.1
15360	1800000	65.2	4.7
30720	3600000	78.6	9.4
61440	7000000	90.3	18.6

Tabla 3: Resultados de la fase de aprendizaje.

5.5.4 REDUCCIÓN Y FUSIÓN DE MATRICES DE APRENDIZAJE

Además de lo expuesto para la fase de aprendizaje, hemos desarrollado dos funciones adicionales -reducción y fusión- que pasamos a describir.

El **módulo de reducción** permite reducir una matriz de aprendizaje, de tal forma que sólo nos quedemos con la información de aprendizaje que supere o iguale una calidad que el propio usuario puede especificar en términos de porcentajes de aparición (mayores o iguales a 50%). Esto provoca que si se consulta una matriz de aprendizaje reducida para deshacer una ambigüedad puedan darse dos situaciones:

1. O bien, la eliminación de la ambigüedad no es posible, lo que se debería a que en la matriz inicial ninguna alternativa supera la calidad solicitada por el usuario[34];
2. O bien, el proceso de eliminación de ambigüedades sólo tiene en cuenta la situación presentada por la matriz: la entrada de la matriz sólo refleja la alternativa que supera la calidad impuesta por el usuario.

En el primero de los casos anteriores, la matriz reducida no nos permite discernir qué etiqueta es la más probable (se elegirá la primera para continuar con el proceso). En el segundo, se elegirá de forma definitiva la alternativa que presente la matriz reducida, siempre que dicha alternativa sea una de las posibles para la palabra que presentó la ambigüedad.

Lo que se consigue con esta reducción de la matriz de aprendizaje es una mayor agilidad a la hora de elegir entre las posibles etiquetas de una palabra: o no se puede deshacer la ambigüedad, o bien sólo hay que determinar si la única alternativa que se refleja al indexar la matriz, se encuentra entre las varias posibilidades de etiquetación que presenta la palabra. En definitiva, lo que se logra es mejorar el tiempo de respuesta y tomar una elección absoluta.

El **módulo de fusión** permite fusionar, en una única matriz final, varias matrices que hayan sido generadas con anterioridad. En realidad, a través de este bloque se posibilita una suma de matrices, o lo que es equivalente, una suma de aprendizajes.

Ambos programas han sido realizados usando Flex y C, originando códigos sumamente rápidos con un tiempo de ejecución casi instantáneo.

Como punto final, se deben recalcar las ventajas que ofrece guardar en un fichero la matriz que se genera:

- Cada estilo literario puede corresponderse, en el sistema, con una matriz que se haya generado a partir de textos de ese mismo estilo. Se pone especial énfasis en los diferentes estilos de los textos porque, obviamente, los patrones de escritura difieren de un poema a un artículo periodístico, por poner dos ejemplos. Los datos que se desprendan del aprendizaje sobre un estilo y otro serán también claramente diferentes.

- El propio usuario puede tener varias matrices en las que la información léxica de las etiquetas que se toma en consideración difiera de unas a otras. Las matrices resultan propicias para aplicar sobre ellas operaciones, como la de suma o fusión de matrices, de las que ya hemos hablado.

5.6 FASE OPERACIONAL

Una vez finalizada la fase de aprendizaje, poseemos una matriz generada a partir de un texto desambiguado, que nos permitirá abordar la presente fase operacional [Andrade et al., 1997].

El código necesario para esta fase se ha incorporado en un programa que permite visualizar los resultados del analizador léxico y usar la opción de supresión de

[34] En esa posición existe un `NULL` en la matriz.

ambigüedades, a la espera de que el analizador sintáctico considere la desambiguación. En caso de invocar esta opción, se deberá especificar la matriz de aprendizaje que se usará para el proceso. Esta matriz se cargará en memoria y el sistema ya estará listo para proceder a eliminar las ambigüedades léxicas que se encuentren en el tratamiento del texto que se está estudiando.

El analizador léxico mostrará todas las posibles alternativas de una palabra, y al final de cada etiqueta da un número que indica la seguridad de esa etiqueta según el aprendizaje realizado. Cuanto más bajo sea este número, más alta será la confianza que ofrece esa etiqueta, siempre de acuerdo con la matriz de aprendizaje que haya indicado el usuario. Existen situaciones, como hemos explicado anteriormente, en que la matriz no nos permite deshacer las ambigüedades. En estos casos se escogerá la primera etiqueta para poder seguir con el proceso.

Mientras la mayoría de los etiquetadores devuelven una única etiqueta, que es considerada como la mejor, para cada palabra[35], algunos trabajos, entre ellos el presente, se han desarrollado en el campo de los etiquetadores que devuelven una lista de las posibles etiquetas en aquellos casos donde una segunda (y subsiguientes) posibilidad podría elegirse de acuerdo con la métrica empleada[36] [Charniak et al., 1996]. La razón es permitir al analizador sintáctico tomar la decisión final, ya que en ese nivel se tiene una visión más amplia de la situación, y esto permitirá tener un mejor criterio de selección. Según [Weischedel et al., 1993] incluso con una tasa de error realmente pequeña de un 3.7%, hay casos en los que el sistema devuelve una etiqueta errónea y se produce una situación muy difícil para el módulo de análisis sintáctico, que intenta trabajar con sentencias, por término medio, de veinte palabras de longitud, en donde se pueden encontrar varias situaciones similares.

Hay que destacar que algunos elementos léxicos, como los pronombres enclíticos o las preposiciones de varias palabras[37], no ofrecen la posibilidad de varias etiquetaciones, es decir, son casos en que no existe ambigüedad. Este hecho ha de tenerse en cuenta. También hay que tener presentes otros casos en los que hay que hacer una gestión especial de los *tokens* que devuelve el analizador léxico. Por ejemplo la palabra "velo" presenta dos posibilidades de etiquetación:

1. ve (verbo ver) + lo (pronombre enclítico).
2. velo (paño utilizado para cubrir la cara).

Si se escoge la segunda alternativa, la siguiente llamada al analizador léxico deberá ignorarse, puesto que devolverá un pronombre enclítico que no tiene sentido ante la elección tomada. Esto no ocurrirá si, guiados por la matriz de aprendizaje, escogemos la primera alternativa. En las líneas siguientes presentamos éste y otro ejemplo tal y como se presentan en la salida de nuestro analizador léxico. El analizador ha sido invocado con la opción de desambiguación y con una matriz de aprendizaje creada con anterioridad:

[35] Llamados *single taggers*.
[36] Se llaman *multiple taggers*.
[37] Tradicionalmente llamadas locuciones prepositivas, del tipo *de cara a*, *en aras de*, etc.

1. Palabra introducida: “velo”

```
velo
=> ["ve", Verbo, 2da, sg, presente, imperativo, prioridad 1, "ver"]
    ["ve", Verbo, 2da, sg, presente, imperativo, prioridad 1, "ir"]
    ["velo", Verbo, primera, sg, presente, indicativo, prioridad 4, "velar"]
    ["velo", Sustantivo comun, masc, sg, "velo"]
=> ["lo", Pron Per encl ato, 3ra, sg, acus, masc, prioridad 1, "'el"]
```

También podríamos tener, si así lo indicase el aprendizaje, la salida siguiente:

```
velo
=> ["ve", Verbo, 2da, sg, presente, imperativo, prioridad 3, "ver"]
    ["ve", Verbo, 2da, sg, presente, imperativo, prioridad 3, "ir"]
    ["velo", Verbo, primera, sg, presente, indicativo, prioridad 3, "velar"]
    ["velo", Sustantivo comun, masc, sg, prioridad 1, "velo"]
```

2. Palabra introducida: “ténselo”

```
t'enselo
=> ["ten", Verbo, 2da, sg, presente, imperativo, prioridad 1, "tener"]
    ["t'ense", Verbo, 2da, sg, presente, imperativo, prioridad 1, "tensar"]
=> ["se", Pron Per encl ato, 3ra, sg y pl, acus y dat, masc y fem. prioridad 1, "el"]
=> ["lo", Pron Per encl ato, 3ra, sg, acus, masc, prioridad 1, "el"]
```

La etiquetación, basada en otra matriz de aprendizaje, también podría ser como se indica a continuación:

```
t'enselo
=> ["t'ense", Verbo, 2da, sg, presente, imperativo, prioridad 2, "tensar"]
    ["ten", Verbo, 2da, sg, presente, imperativo, prioridad 2, "tener"]
=> ["lo", Pron Per encl ato, 3ra, sg, acus, masc, prioridad 1, "'el"]
```

La situación que se da en la fase operacional se podría representar como se indica en la figura 18.

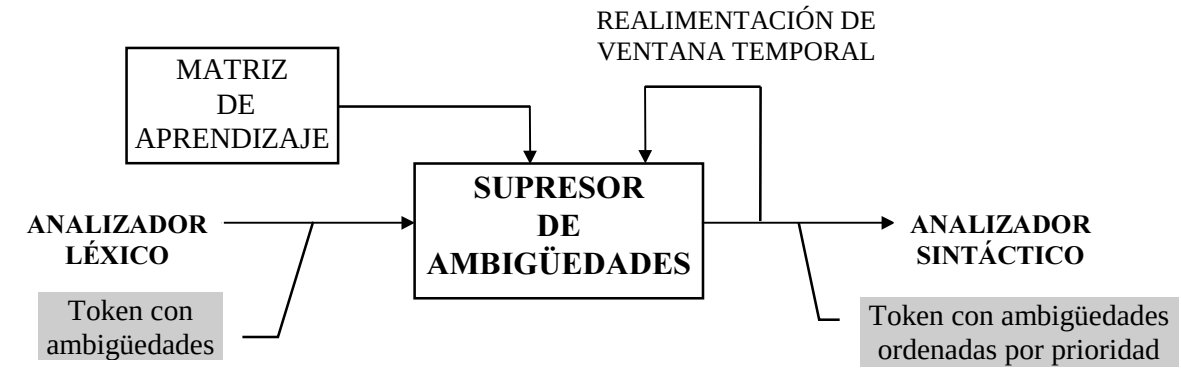


Figura 18: Visión del sistema en la fase operacional.

La forma de incorporar la desambiguación a un analizador léxico, en concreto al de GALENA, resulta tan sencilla como lo siguiente:

```
                                    …
case 'd':   /* disambiguation option */
      if ((zzin = fopen (optarg, "r")) != NULL)
        {
          disambiguation_flag = 1;
          ReadNumLabel();
          AllocateMatrix();
          ReadMatrix();
          InitTemporalWindow();
          tkn_aux=AllocateTknAux();
        }
      else
        fprintf (stderr, "Galena lexer warning: cannot open file %s
for input (-d ignored).\n", optarg);
      break;
                                    …

while (get_category_token(actual_token=yygalenalex()))
    {
      if (disambiguation_flag)
      DisambiguatePrintToken (actual_token);

                                    …
  if (disambiguation_flag)
    {
      FreeMatrix();
      free(tkn_aux);
    }
                                    …
```

Existen datos que se guardan en las matrices pero que no son necesarios para la fase operacional. Estos datos son los números indicativos del número de apariciones. Estos sirven para permitir la operativa de fusión y reducción de las matrices pero una vez que la matriz se desea leer en memoria para abordar la fase operacional ya no son necesarios. Esto es debido a que su utilidad es la de permitir la ordenación de la tercera dimensión, como esta dimensión ya se almacena ordenada y sólo se desea establecer para cada etiqueta su prioridad se pueden obviar; será el orden en la tercera dimensión el que establezca directamente la prioridad, sin necesidad de observar el número de veces que ha aparecido en el aprendizaje esa etiqueta. Llega simplemente con saber que ha aparecido un número de veces, que no conocemos en la fase operacional, que determinó su lugar en la ordenación de la tercera dimensión.

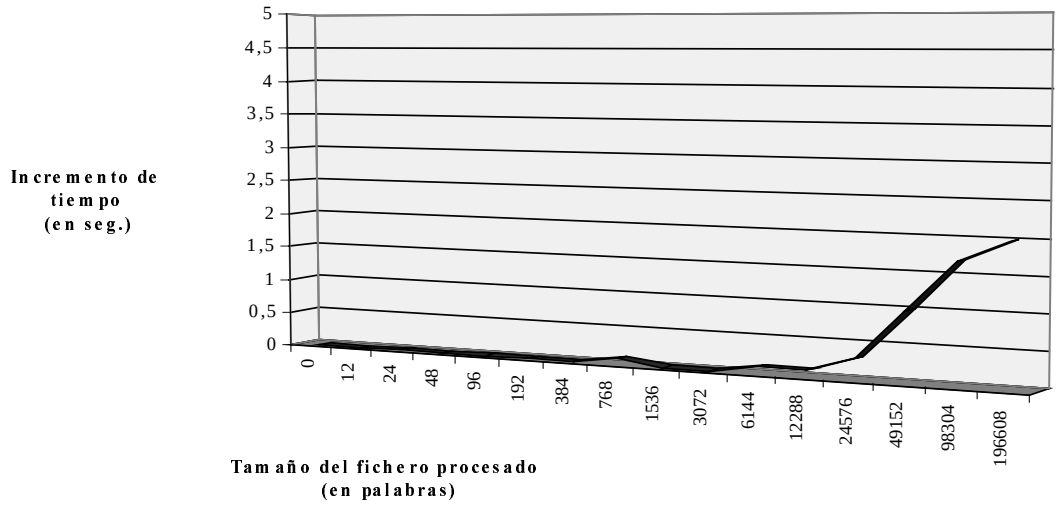
5.6.1 TIEMPOS DE LA FASE OPERACIONAL

En este apartado se comprobará el impacto que tiene la desambiguación en el lematizador.

Considerando una matriz de aprendizaje, resultante de una configuración de 48 etiquetas léxicas, esto es, una matriz de 48 x 48 más la tercera dimensión resultante del aprendizaje[38],sobre una Sun SPARCstation 10 bajo SunOS 4.1.4 y con una carga de trabajo media, la influencia de la opción de supresión de ambigüedades se puede observar en la gráfica 2, mostrada a continuación.

[38] Variable dependiendo del corpus de aprendizaje empleado en el mismo.

Los tiempos mostrados incluyen: tiempo de escritura/lectura en soporte físico y tiempo invertido en las acciones de etiquetación, así como los posibles errores del cronómetro Unix[39]. Se puede observar que la influencia del desambiguador en la *performance* del lematizador no es significativa.



Gráfica 2. Influencia del supresor en la fase operacional

Tamaño del fichero *en palabras*	**Incremento de tiempo** *(en seg.)*
Sin texto	0
12	0
24	0
48	0
96	0
192	0
384	0
768	0.1
1536	0
3072	0
6144	0.1
12288	0.1
24576	0.3
49152	1
98304	1.7
196608	2

Tabla 4: Datos de la gráfica anterior.

Debe tenerse muy en consideración, en la lectura de la gráfica anterior, los siguientes puntos:

[39] Estadísticas obtenidas con el comando `time` de Unix.

1. El porcentaje de ambigüedad en la entrada a analizar es determinante. En esta evaluación elegimos un texto con una ambigüedad de un 65%.
2. La calidad del aprendizaje influye notablemente en el tamaño de la matriz, en concreto son dos los aspectos fundamentales:
   - Tamaño del corpus de aprendizaje: determina el tamaño de la tercera dimensión de la matriz de aprendizaje (en nuestro caso la matriz es de 5 Kb.).
   - La configuración establecida para el supresor: determina las dimensiones de la matriz.

5.7 UN EJEMPLO DE PRUEBA

En este apartado presentamos una prueba de la fase operacional, que ha sido realizada tomando como texto de aprendizaje un fragmento de la obra Crónica de una muerte anunciada de Gabriel García Márquez [García, 1988], constituido por 500 palabras y que fue previamente desambiguado a mano. Estos resultados han sido publicados en [Andrade et al., 1997].

Category	Type	Etiquetaciones correctas	Etiquetaciones erróneas
Adjective		29	Verb: 1
Adverb	nuclear	11	Verb: 1
	otros	5	
Article		44	
Conjunction	coordinate	12	
	subordinate	23	Verb: 1 Relative: 1
Demonstrative		3	
Indefinite		23	Numeral cardinal: 4 Verb: 2
Interjection		0	
Interrogative		0	
Numeral	Cardinal	7	Verb: 1
	Otros	0	
Peripheral	Todas	0	
Preposition		91	Verb: 3
Personal pronoun	Tonic	3	
	Proclitic atonic	19	Article: 2
	Enclitic atonic	2	
Possessive		5	
Punctuation mark	todas	47	
Relative		7	Conjunction subordinate: 6
Substantive	common	95	Verb: 3 Adjective: 1
	proper	10	
Verb		66	otros Verb: 3

Tabla 5: Resultados de la fase operacional.

La primera tarea fue configurar el supresor estadístico de ambigüedades tomando como información léxica la mostrada en la figura 17 (pág. 114). Esta configuración originó cerca de 400 etiquetas léxicas, que fueron las consideradas tanto para el aprendizaje como para la fase operacional. La tarea siguiente fue la llamada al analizador léxico, con la opción de desambiguación y la matriz generada, a partir del texto de aprendizaje anterior.

El resultado que se obtuvo, sobre un fragmento de cerca de 550 palabras de la obra citada, fue de aproximadamente un 95% de acierto en la elección de la etiqueta correcta. Si bien este porcentaje resulta alentador, es evidente que con el tamaño del texto de aprendizaje que se ha podido manejar, las estadísticas no son significativas (ver la tabla 5). Todo el proceso se ha desarrollado con una base de datos léxica, constituida por 4.500 lemas.

Los resultados de la tabla 5 nos sugieren varias reflexiones:

1. Algunas categorías, como por ejemplo Demonstrative, Possessive y Punctuation mark, no plantean complicaciones de elección entre varias alternativas.

2. Habrá que centrar los esfuerzos en eliminar los errores referentes a las categorías Relative y Conjunction subordinate[40]. Tanto en este tipo de error, como en otros que pudieran aparecer, se puede proceder de varias maneras:

> (a) Manejando textos de aprendizaje mayores que, en breve, se deben tener disponibles. Esto es lo que se ha venido haciendo hasta ahora.
>
> (b) Seleccionando las etiquetas más adecuadas para llevar a cabo la desambiguación, a través de la ventana de configuración presentada en la figura 17 (pág. 114). Esta tarea debe llevarse a cabo con ayuda de un equipo de lingüistas, ya que son ellos los que pueden aportar conocimiento lingüístico sobre este aspecto.

3. La selección mencionada en el punto anterior debe estar orientada a minimizar la elección de Verb como etiqueta, ya que ésta constituye la mayor fuente de errores o selecciones incorrectas.

4. La desambiguación mediante reglas, proyectada para el futuro, deberá centrarse fundamentalmente en el estudio de las situaciones en que el desambiguador actual no permite reducir el margen de error.

[40] Los errores se deben, fundamentalmente, a la aparición de la partícula *que*.

6. CONCLUSIONES

A cada palabra, fuera de contexto, le puede corresponder, o bien una sola etiqueta, o bien un conjunto limitado de etiquetas. Un análisis léxico de una palabra dada nos proporciona el conjunto de etiquetas que le atañen.

Cuando una palabra ofrece varias alternativas de etiquetación, generalmente se puede escoger la etiqueta correcta usando estadísticas contextuales que definen la secuencia válida de etiquetas. El usuario puede, utilizando el sistema propuesto, seleccionar la información léxica más adecuada para refinar el aprendizaje, y por ende, las anteriores estadísticas.

Dicho aprendizaje se manifiesta en las matrices de aprendizaje que se generan y que son características de cada estilo literario.

Una aproximación estadística del problema ofrece las siguientes ventajas:

- Proporciona un marco de trabajo teórico firme: el de la estadística.
- Los resultados de la aproximación son claros.
- Las probabilidades proporcionan un camino sencillo para eliminar las ambigüedades.
- Las probabilidades pueden estimarse automáticamente a partir de los datos.

La aproximación estadística presentada combina la sencillez, a la hora de hallar las probabilidades, junto con la facilidad de implementación y los buenos resultados obtenidos.

El supresor de ambigüedades que proponemos permite al usuario seleccionar la información léxica, en definitiva, las etiquetas, que se desea considerar en la desambiguación a través de una sencilla e intuitiva interfaz. También se ofrecen funcionalidades complementarias para el tratamiento de las matrices de aprendizaje.

Con este sistema se ha resuelto el asociar a cada palabra una única etiqueta en el sistema GALENA, integrándolo con el analizador que hasta el momento existía. El trabajo se verá logrado completamente cuando el analizador sintáctico considere el trabajo realizado por el módulo de desambiguación.

Las posibles ampliaciones a realizar en el ámbito de este trabajo se pueden concretar en los siguientes puntos:

- Ajustar el sistema construido a los nuevos fenómenos léxicos que se incorporen al analizador léxico de GALENA.
- Reducir el error subsistente mediante información lingüística recogida en un sistema de desambiguación basada en reglas. Esto supone desarrollar dicho sistema e integrarlo con el trabajo actual de forma que ambos sistemas actúen conjuntamente, aprovechando de esta manera las ventajas de uno para solventar los inconvenientes del otro.
- Desarrollar herramientas cómodas e intuitivas para obtener los resultados de acierto en la etiquetación de las palabras. En la actualidad la forma de obtener estos resultados del comportamiento del supresor de ambigüedades es muy rudimentaria, además de ser extremadamente lenta y sujeta a numerosos errores.

Por último, hay que mencionar el hecho de que este módulo de supresión de ambigüedades ha sido adaptado al sistema XIADA (pág. 66), del idioma gallego. Esto

ha permitido comprobar, de forma fehaciente, que un cambio en el juego de etiquetas es fácilmente absorbido con la arquitectura lograda para el desambiguador. Este hecho justifica plenamente la evolución del sistema, presentada en esta memoria, para lograr una total flexibilidad ante modificaciones similares, y que era uno de los objetivos que desde el principio se perseguía.

APÉNDICES

A. LOS LENGUAJES DE PROGRAMACIÓN ELEGIDOS

En este apéndice se pretende justificar la elección de los lenguajes de programación que se han empleado a lo largo del desarrollo de la presente tesis de licenciatura.

Los lenguajes que se han utilizado han sido los siguientes:

- C [Kernighan y Ritchie, 1991].
- Flex y Bison [Mason y Brown, 1990].
- Tcl/Tk [Ousterhout, 1994].
- Gawk [Kernighan y Pike, 1987].

A continuación se ofrecerán unos breves razonamientos de los motivos de haber hecho uso de cada uno de los lenguajes.

- **C**

1. La primera, y más importante razón, es que el sistema de desambiguación debe integrarse con otros ya existentes, con los que se debe entender y que se encuentran ya desarrollados en C.

2. C es uno de los lenguajes de programación más populares en uso. Proporciona un esqueleto estructurado sin imponer limitaciones a la creatividad del programador; además los compiladores C producen programas muy rápidos y eficientes al ejecutarlos.

3. Ya que hay más programadores que conocen C que los que conocen algún otro lenguaje habitualmente usado en investigación en IA, parece claro que si las técnicas de IA van a ser usadas en el mundo real necesitan ser codificadas en C.

4. No hay ni una sola técnica de IA que no pueda ser desarrollada usando un lenguaje procedimental como lo es C [Schildt, 1990].

5. Aprender a desarrollar aspectos de IA a través de C es importante porque estos elementos pueden adaptarse a un gran número de aplicaciones existentes. Muchas aplicaciones han sido, y continuarán siéndolo, escritas en C, y muchas de ellas pueden beneficiarse de la suma de los distintos avances de la IA, y en concreto del PLN. Por ejemplo un gestor de base de datos o un sistema operativo podrían usar un procesador de lenguaje natural como un *front-end* para crear un mejor procesador de órdenes.

• **FLEX y BISON**

1. Ambas herramientas, usadas generalmente de forma conjunta, permiten construir analizadores léxicos, la primera, y sintácticos, la segunda, muy útiles para realizar el tratamiento de todos los ficheros del sistema que están relacionados con el módulo supresor de ambigüedades.

2. Su uso es muy sencillo, permitiendo absorber fácil y rápidamente las modificaciones surgidas, tanto en la fase de construcción como en la fase de mantenimiento.

3. Ambas herramientas generan código C, por esta razón son directamente integrables con el código C desarrollado.

• **TCL/TK**

1. Permite generar interfaces gráficas de forma relativamente rápida y sencilla.

2. Era el lenguaje, para el uso expuesto, conocido por los miembros del grupo, lo que hacía más sencillo el desarrollo.

3. Y, principalmente, se seleccionó porque la parte gráfica del analizador léxico (figs. 7, 8 y 9) estaba desarrollada con Tcl/Tk lo que invitaba a seguir usándolo para, no sólo aprovechar conocimientos, sino también para mantener la compatibilidad.

• **GAWK**

1. Es un lenguaje sencillo, similar a C, algo más simplificado en su sintaxis, y con una carga semántica mayor.

2. Resulta ser muy potente en el tratamiento de patrones y en el manejo de ficheros. Estas dos características hicieron que nos decantásemos por su uso, ya que eran dos aspectos muy necesarios en la primera versión desarrollada del desambiguador.

3. El hacer prototipos, esto es, la primera versión del desambiguador, resulta muy fácil y rápido debido a su potencia y relativa sencillez. Además no impone exigencias que después, al descartar el prototipo o al realizar modificaciones, sean difíciles de superar.

4. El manejo que posee de las variables favorecía su utilización en nuestro ámbito, ya que no hay que definirlas y sólo al usarlas se les reserva espacio. Esta característica, si se piensa en la creación de las matrices de aprendizaje, resulta muy importante.

5. Permite indexar *arrays* por medio de cadenas de caracteres usando, internamente, una función *hash*. Esta característica resultó en su momento salvadora ya que facilitaba el uso de las etiquetas directamente, y determinó su

uso, a pesar de lo críptico que es y lo poco eficiente que resulta a tenor de los resultados mostrados con anterioridad.

B. ORGANIZACIÓN DEL SISTEMA

En este apartado se refleja la estructura de directorios Unix, así como el contenido de los mismos, que se ha creado para el módulo de supresión de ambigüedades. Se pretende, de esta manera, ofrecer una visión del sistema que clarifique su funcionamiento, a la vez que se presenta su organización y división en módulos. Sin entrar en excesivos detalles se comentarán los componentes que integran cada fichero con el afán de familiarizarse con el sistema.

En la figura 19 se presenta el árbol de directorios que tiene relación con el desambiguador.

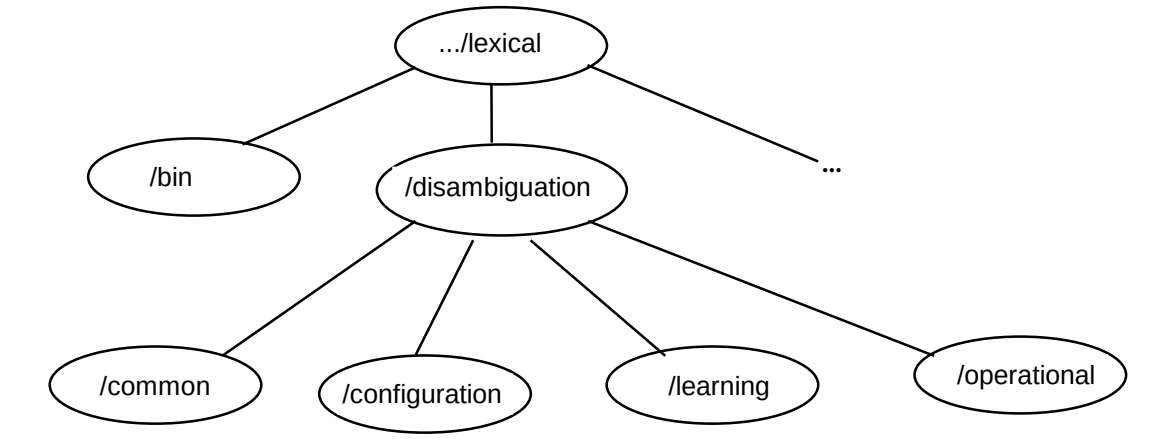


Figura 19. Organización de los códigos del supresor de ambigüedades.

El primer detalle que se desea hacer notar es que el sistema desarrollado forma parte del analizador léxico (disambiguation es subdirectorio de lexical). Esto reitera el hecho de que aunque está pensado para facilitar la labor del analizador sintáctico, realiza una tarea encuadrada en la parte léxica.

En un primer subdirectorio llamado bin residen los *links* simbólicos a los diferentes programas ejecutables, de la parte léxica, desarrollados para GALENA. Entre estos programas se encuentran los relacionados con el módulo de supresión de ambigüedades:

- **gldi**[41]: muestra la ventana de configuración del supresor de ambigüedades (figura 17, pág. 114).

- **learn**: ejecutable para crear la matriz de aprendizaje cuya sintaxis es:

```
learn {file_without_ambiguities} matrix_file
```

[41] Abreviación de *Galena Lexer Disambiguation Interface*.

- **reduce**: ejecutable que reduce una matriz dada y cuya sintaxis es:

  reduce <m1> <p> <m2>

  donde <m1> es la matriz de entrada, <p> es el porcentaje de reducción (>=50%) y <m2> es la matriz que se crea de salida.

- **fusion**: ejecutable para la fusión de matrices cuya sintaxis es:

  fusion {matrix_file_in} matrix_file_out

- **gld**: analizador léxico que al ser invocado con la opción -d, seguida de un fichero que contiene una matriz de aprendizaje, se convierte en un etiquetador con la operativa descrita.

Estos ejecutables están preparados para consultar la variable de entorno GALENA_LEXER_DIR, que contiene el directorio donde se instala toda la parte léxica de GALENA, de tal forma que el sistema se puede instalar en cualquier punto del árbol de directorios de un sistema y todo seguirá funcionando correctamente.

En el subdirectorio common se localizan todos los módulos que pueden ser llamados por los diferentes programas ejecutables desarrollados. De esta manera se facilita la labor de mantenimiento del sistema, ya que los cambios se realizarán en un único código y se verán propagados al compilar todo el sistema. Esto es, se pretende evitar la redundancia de código. Ni que decir tiene que estos códigos se han desarrollado de tal manera que sean perfectamente flexibles, permitiendo un núcleo común y la incorporación de las particularidades de forma muy sencilla.

En este subdirectorio nos encontramos los siguientes ficheros:

defines.h: contiene los defines e includes comunes a todos los códigos.

disambswitch.c: contiene la función int GetIndex (struct ice_lex_object *tkn), que implementa el switch que permite realizar la traducción de una etiqueta a un código de indexación en la matriz de aprendizaje, tal y como lo definía la función τ.

files.c: contiene las funciones utilizadas para saber si los ficheros, con los que trabajan los diferentes ejecutables, existen.

matrix.c: contiene las funciones usadas para manipular la matriz de aprendizaje. Incluye las funciones de reserva de memoria para la matriz, void AllocateMatrix (void), de liberación de memoria, void FreeMatrix (void), y de acceso a la matriz, struct no_ma* AccedeMatrix (int line, int column).

matrixlearn.c: incorpora las funciones usadas en la fase de aprendizaje relacionadas con operaciones sobre la matriz. Estas funciones son las utilizadas para volcar a un fichero la matriz, void PrintMatrix (char * file), para ordenar la matriz, void OrderMatrix (void), etc.

window.c: contiene la definición de la estructura que implementa la ventana temporal y todas las funciones relacionadas: void InitTemporalWindow (void) y void RefreshTemporalWindow (int label).

En el subdirectorio configuration nos encontramos con todos los códigos que sirven para lograr la configuración del módulo de supresión de ambigüedades:

Makefile: fichero, de la herramienta make, que permite automatizar la compilación de los códigos contenidos en este directorio.

ReadIniFile.h: fichero header usado en el código Bison ReadIniFile.y. Contiene la definición de tipos y estructuras necesarias para generar automáticamente el código que contiene el switch que implementa la función τ.

ReadIniFile.l: fichero Flex que describe el analizador léxico usado por el analizador sintáctico descrito por ReadIniFile.y. El fichero analizado es disambiguation.ini.

ReadIniFile.y: analizador sintáctico, descrito en código Bison, que procesa el fichero de configuración del desambiguador: disambiguation.ini.

ReadIniFile.tcl: código Tcl/Tk que crea la interfaz gráfica de configuración.

disambiguation.ini: fichero ASCII que mantiene la configuración establecida para el desambiguador.

setup: código ejecutable, llamado desde el código Tcl/Tk, que procesa el fichero de configuración y genera el código del switch. Resulta de la compilación conjunta de ReadIniFile.h, ReadIniFile.l y ReadIniFile.y.

En el subdirectorio learning se encuentran todos los códigos que se encuadran en la fase de aprendizaje, es decir, los códigos realizados para crear, reducir y fusionar las matrices de aprendizaje.

Los ficheros incluidos en este subdirectorio son los que a continuación se indican.

Makefile: fichero, de la herramienta make, que permite automatizar la compilación de los códigos contenidos en este directorio.

fusion: ejecutable que permite sumar el aprendizaje debido a varias matrices de aprendizaje.

fusion.h: establece las definiciones necesarias para realizar la fusión de matrices.

fusion.l: código Flex que permite leer las matrices, y ayudado de funciones C auxiliares, sumarlas.

reduce: código ejecutable que permite reducir una matriz de aprendizaje de acuerdo con una calidad impuesta por el usuario.

reduce.h: establece las definiciones necesarias para realizar la reducción de matrices.

reduce.l: código Flex que permite leer las matrices, y ayudado de funciones C auxiliares, reducirlas según los parámetros establecidos por el usuario.

learn: código ejecutable que permite efectuar el aprendizaje a partir de una serie de ficheros de texto etiquetados y desambiguados, reflejándolo en un fichero que contiene la matriz de aprendizaje resultante. Resulta de la compilación conjunta de los siguientes ficheros:

regeneration.h: define las estructuras y prototipos de funciones que se utilizarán en regeneration.y, del que es fichero de cabecera.

regeneration.l: código Flex que describe el analizador léxico empleado para regenerar los valores de los campos de la variable `token` a partir de la etiqueta marcada como correcta para una palabra dada.

regeneration.y: código Bison que describe el analizador sintáctico empleado para regenerar la variable token.

El subdirectorio operational contiene los códigos empleados en la fase operacional. Estos códigos, por tanto, permiten llevar a cabo la desambiguación en base a una matriz de aprendizaje previamente creada.

Los ficheros localizados en este subdirectorio son los siguientes:

disambi.h: es el fichero de cabecera del código C disambi.c.

disambi.c: contiene todas las funciones que permiten llevar a cabo la supresión de ambigüedades: void SetPriority (struct ice_lex_object *position, int priority_number), etc. La función principal que refleja la desambiguación es void DisambiguatePrintToken (struct ice_lex_object *actual_token).

readmatrixfile: código Flex que permite leer la matriz descartando aquellos datos que para la fase operacional no son necesarios: números de aparición de las etiquetas.

BIBLIOGRAFÍA

[Aldezabal et al., 1996] Aldezabal, I., Alegría, I., Ezeiza, N., Urizar, R. y Aduriz, I. Del analizador morfológico al etiquetador/lematizador: Unidades léxicas complejas y desambiguación. Procesamiento del Lenguaje Natural, Septiembre de 1996. Revista nº 19. pp. 90-100.

[Alonso, 1994] Alonso Pardo, M.A. Edición interactiva en entornos incrementales. Tesis de Licenciatura de la Fac. de Informática de la Univ. de A Coruña. Septiembre de 1994.

[Andrade et al., 1997] Andrade G., J., Valderruten V., A., Álvarez L., Mª.C. y Sotelo D., S. Un Supresor de Ambigüedades Léxicas mediante Métodos Estadísticos. Revista de la SEPLN, Marzo de 1997.

[Angulo y del Moral, 1986] Angulo, J.M. y del Moral, A. Guía fácil: INTELIGENCIA ARTIFICIAL. 1ª edición. Madrid: Paraninfo, 1986. Series Guía fácil. ISBN 84-283-1451-9.

[Barr y Cohen, 1989] Barr, A., R. Cohen, P. y A. Feigenbaum, E. The handbook of Artificial Intelligence. Volumen IV. USA: Addison-Wesley Publishing Company, Inc, 1989. ISBN 0-201-51731-0.

[Barr y Feigenbaum, 1989] Barr, Avron y A. Feigenbaum, Edward. The handbook of Artificial Intelligence. Volumen I. USA: Addison-Wesley Publishing Company, Inc, 1989. ISBN 0-201-11811-4.

[Codd, 1974] Codd, T. Seven steps to RENDEZVOUS with the casual user. Database Management. North-Holland: Hoffmann Eds., 1974.

[Costa et al., 1988] Costa C., X.X., González R., Mª dos Anxos, Morán F., C.C. y Rábade C., X.C. Nova gramática para a aprendizaxe da língua. 1ª edición. A Coruña: Vía láctea, Septiembre 1988. Series de manuales. ISBN 84-86531-48-9.

[Charniak et al., 1996] Charniak, E. et al. Taggers for parsers. Artificial Intelligence, 1996. Número 85. pp. 45-57.

[Church, 1988] Church, K. W. A stochastic parts program and Noun Phrase parser for unrestricted text. Proceedings de la II Conferencia del Procesamiento del Lenguaje Natural Aplicado, 1988. pp. 136-143.

[Church y Mercer, 1993] Church y Mercer. Introduction to the Special Issue on Computational Linguistics Using Large Corpora. Computational Linguistics, 1993. Volumen 19, Número 1. pp. 1-24.

[EAGLES, 1994] EAGLES. Morphosyntactic Annotation. EAGLES document EAG-CSG/IR-T3.1. Versión draft de Octubre de 1994.

[Franz, 1996] Franz, Alexander. Automatic Ambiguity Resolution in Natural Language Processing. An Empirical Approach. Lecture Notes in Artificial Intelligence. Número 1171. Subseries de Lecture Notes in Computer Science. Berlin: Springer, 1996. ISBN 3-540-62004-4.

[García, 1988] García Márquez, G. Crónica de una muerte anunciada. Madrid: Mondadori, 1988.

[Garside et al., 1987] Garside, R., Leech, G. y Sampson, G. The Computational Analysis of English. Longman, Londres, 1987.

[Gevarter, 1987] M. Gervarter, W. Máquinas inteligentes: Una panorámica de la inteligencia artificial y de la robótica. 1ª edición. Madrid: Díaz de Santos, 1987. ISBN 84-86251-63-X.

[Graña, Alonso y Valderruten, 1994] Graña G., J., Alonso P., M.A. y Valderruten V., A. Análisis léxico no determinista: Etiquetación eficiente del leguaje natural. Technical Report del Depto. de Computación de la Univ. de A Coruña, Octubre de 1994.

[Green y Rubin, 1971] Green, B. B. y Rubin, G. M. Automated grammatical tagging of English. Depto. de Lingüística de la Univ. de Brown. Isla Rhode, 1971.

[Hemdrix et al., 1978] Hemdrix, G., Sacerdoti, D., Sagalowicz, D., Slocom, J. Developing a natural language interface to complex data. ACM Transactions on Database Systems, 3, Vol. 2, 1978. pp. 105-147.

[Hindle, 1983] Hindle, D. User manual for FIDDITCH, a deterministic parser. Technical Memorandum, 1983. Número 7590-142. Naval Research Laboratories.

[Kaare, 1990] Kaare, Christian. Unix. Madrid: Paraninfo, 1990. ISBN 84-283-1783-6.

[Kernighan y Pike, 1987] Kernighan, B.W. y Pike, R. El entorno de programación Unix. México: Prentice-Hall Hispanoamericana, S.A., 1987. ISBN 958-880-067-8.

[Kernighan y Ritchie, 1991] Kernighan, B.W. y Ritchie, D.M. El lenguaje de programación C. 2ª edición. México: Prentice-Hall Hispanoamericana, S.A., 1991. ISBN 968-880-205-0.

[Klein y Simmons, 1963] Klein, S. y Simmons, R. A computational approach to grammatical coding of English words. JACM, 1963.

[Kupiec, 1992] Kupiec, J. Robust part-of-speech tegging using a hidden Markov model. Computer Speech and Language, 1992. Volumen 6. pp. 225-242.

[Lin, Y.-C. et al., 1995] Lin, Y.-C. et al. The effects of learning, parameter tying and model refinement for improving probabilistic tagging. Computer Speech and Language, 1995. Ed. Academic Press Limited. Volumen 9. pp. 37-61.

[Luger y Stubblefield, 1993] F. Luger, G. y A. Stubblefield, W. Artificial Intelligence. Structures and strategies for complex problem solving. 2ª edición. California: The Benjamin/Cummings Publishing Company, Inc, 1993. ISBN 0-8053-4780-1.

[Mason y Brown, 1990] Mason, Tony y Brown, Doug. Lex & Yac. 1ª edición. Sebastopol: O'Reilly & Associates, Inc., Mayo de 1990.

[Merialdo, 1994] Merialdo, B. Tagging English Text with a Probabilistic Model. Computational Linguistics, 1994. Volumen 20, Número 2. pp. 155-171.

[Mishkoff, 1988] C. Mishkoff, Henry. A fondo: Inteligencia artificial. Madrid: Anaya, 1988. ISBN 84-7614-165-3.

[Ousterhout, 1994] Ousterhout, J.K. Tcl and the Tk Toolkit. Massachusetts: Addison-Wesley, 1994. Profesional Computing Series. ISBN 0-201-63337-X.

[Partridge, 1991] Partridge, Derek. A new guide to Artificial Intelligence. Norwood, New Jersey: Ablex Publishing Corporation Norwood, 1991. ISBN 0-89391-607-2.

[Rauch-Hindin, 1989] Rauch-Hindin, B. Wendy. Aplicaciones de la Inteligencia Artificial en la actividad empresarial, la ciencia y la industria (Fundamentos-Aplicaciones). Madrid: Ediciones Díaz de Santos, S.A., 1989. ISBN 84-87189-07-5.

[Rich, 1983] Rich, Elaine. Artificial Intelligence. Nueva York: McGraw-Hill. Computer Science Series, 1983. ISBN 0-07-Y66508-7. (Ref. a la página 236).

[Schank, 1984 a] Schank, Roger. Intelligent Advisory Systems. The AI Business, MA: MIT Press, 1984. pp. 133-148.

[Schank, 1984 b] Schank, R., Childers, P. G. The Cognitive Computer. Reading MA: Addison-Wesley, 1984.

[Schildt, 1990] Schildt, Herbert. Utilización de C en Inteligencia Artificial. Madrid: Osborne/McGraw-Hill, 1990. ISBN 84-7615-290-6.

[Toma, 1977] Toma, P. SYSTRAN as a multilingual machine translation system. Overcoming the language barrier, CCEE. Munich, Verlag Documentation, 1977. pp. 569-581.

[Vilares y Alonso, 1996] Vilares Ferro, M. y Alonso Pardo, M.A. Towards Analyzers Based on Efficient Logical Frames. Proc. fo the II International Conference on Mathematical Linguistics, 1996, Tarragona. pp. 97-98.

[Vilares, Alonso y Cabrero, 1996 a] Vilares F., M., Alonso P., M.A. y Cabrero S., D. An Experience on Natural Language Parsing. Actas del XII Congreso de lenguajes naturales y lenguajes formales, 1996. pp. 555-562.

[Vilares, Alonso y Cabrero, 1996 b] Vilares F., M., Alonso P., M.A. y Cabrero S., D. Autómatas lógicos y lenguaje natural. V Congreso Iberoamericano de Inteligencia Artificial, Editorial Limusa, S.A. de C.V., Mexico D.F., Mexico, 1996. ISBN 968-18-5429-2, pp. 341-351.

[Vilares, Graña y Pan, 1996] Vilares F. M., Graña G. J. y Pan B. A. Building Friendly Architectures for Tagging. Proc. of SEPLN, 1996, Sevila. pp. 127-132.

[Voutilainen y Tapanainen, 1993] Voutilainen, A. y Tapanainen, P. Ambiguity resolution in a reductionistic parser. EACL, 1993. pp. 394-403.

[Weischedel et al., 1993] Weischedel, R., et al. Coping with ambiguity and unknown words through probabilistic models. Computational Linguistics, 1993. Volumen 19. pp. 359-382.

[Woods, 1970] Woods, W.A. Transition network grammars for natural language analysis. Communications of the ACM, Vol. 13, 1970.

[Yourdon, 1989] Yourdon, Edward. Modern structured analysis. Republic of Singapore: Prentice-Hall. Yourdon Press Computing Series, 1989. ISBN 0-13-598632-X.

Esta memoria fue finalizada en el Departamento de Computación de la Universidade de A Coruña en el mes de Septiembre de mil novecientos noventa y siete.