Nociones de Procesamiento del Lenguaje Natural

Miguel A. Alonso Pardo
Carlos Gómez Rodríguez
Jorge Graña Gil
Jesús Vilares Ferro

Facultad de informática, Universidade da Coruña

2014

Índice general

Índice de figuras

Capítulo 1

Introducción al procesamiento del lenguaje natural

El lenguaje es uno de los aspectos fundamentales del comportamiento humano y constituye un componente esencial de nuestras vidas. En su forma escrita sirve para guardar un registro del conocimiento que se transmite de generación en generación. En su forma hablada constituye el principal medio de comunicación con otras personas en la realización de actividades cotidianas. El objetivo del Procesamiento del Lenguaje Natural (PLN) es crear programas de ordenador que incorporen modelos computacionales del lenguaje y que se aproximen a la capacidad humana en las tareas lingüísticas de lectura, escritura, escucha y habla. Sin embargo, lo importante no es el medio utilizado, sea escritura a mano, textos procesados en un ordenador, o habla. Lo importante son los procesos de comprensión y uso del lenguaje. Para este fin, los modelos computacionales son útiles tanto para explorar la naturaleza de la comunicación lingüística como para la construcción de aplicaciones prácticas que faciliten la comunicación entre los ordenadores y el ser humano. La motivación práctica, o tecnológica, del procesamiento del lenguaje natural radica en que la inclusión de habilidades de comprensión del lenguaje en los ordenadores revolucionaría por completo el modo en que son utilizados. Puesto que la mayor parte del conocimiento humano está registrado en forma lingüística, si los ordenadores pudiesen trabajar con el lenguaje natural, podrían acceder y procesar toda esta información. Adicionalmente, el desarrollo de interfaces en lenguaje natural facilitaría el acceso general a sistemas complejos.

1.1. Relación con otras disciplinas

Varias disciplinas se encargan del estudio del lenguaje. Cada una de ellas define su propio conjunto de problemas así como sus propios métodos para tratarlos. La *lingüística*, por ejemplo, estudia las estructuras del lenguaje en sí, considerando cuestiones tales como porqué ciertas combinaciones de palabras forman frases mientras que otras no lo hacen, o porqué una frase tiene ciertos significados y no otros. La *psicolingüística*, por su parte, se encarga del estudio de los procesos de la producción y comprensión de las lenguas, considerando cuestiones tales como porqué la gente identifica la estructura apropiada para una frase, o cómo es que somos capaces de determinar el significado apropiado de cada palabra. La *filosofía* estudia cómo es que las palabras permiten identificar los objetos del mundo, así como las relaciones entre el lenguaje y las creencias, objetivos e intenciones de cada persona. El objetivo de la *lingüística computacional* es desarrollar una teoría computacional del lenguaje, utilizando para ello las nociones de algoritmos y estructuras de datos propias de la informática. Muchas veces se usa el término *procesamiento de lenguaje natural* como sinónimo de lingüística computacional, aunque lo habitual es considerar que el PLN se refiere a los aspectos más prácticos del tratamiento computacional del lenguaje, centrándose en el diseño e implementación de algoritmos eficientes para el análisis, comprensión y generación de textos. Últimamente ha surgido el término *ingeniería de la lengua* como aglutinante de todas las áreas que tratan el análisis computerizado del lenguaje, incluyendo la lingüística teórica y descriptiva, la lexicología, la informática y la ingeniería. Éste y otros términos pretenden eliminar la distancia existente entre los lingüistas computacionales y aquellos que se dedican a la implementación de aplicaciones prácticas con un uso real potencial.

1.2. Niveles de análisis

Un sistema de PLN debe hacer uso de una cantidad considerable de conocimiento acerca de la estructura del lenguaje: cómo son las palabras que lo forman, cómo se combinan estas palabras para dar lugar a frases, qué significan las palabras, cómo se construye el significado de una frase a partir de los significados de las palabras que la constituyen, etc. A continuación enumeramos los diferentes niveles de conocimiento que deben utilizarse:

Conocimiento léxico. Especifica cómo se forman las palabras a partir de unidades más pequeñas denominadas morfemas.

Conocimiento sintáctico. Especifica cómo deben agruparse las palabras para formar sintagmas, y éstos para formar frases, determina el papel estructural que desempeña cada palabra y cada sintagma en la frase resultante y establece qué frases forman parte de frases mayores.

Conocimiento semántico. Especifica el significado de las palabras y la forma de obtener el significado de una frase a partir del significado de las palabras que la forman, de acuerdo con su estructura sintáctica.

Conocimiento pragmático. Especifica la relación entre el lenguaje y el contexto en el que se utiliza. Incluye aspectos tales como la identificación de personas y objetos, la estructura del discurso, y la utilización de conococimiento del mundo en entornos conversacionales.

En concordancia con estos niveles de conocimiento, podemos establecer cuatro niveles de análisis en los que se incluyen diversos modelos computacionales y algoritmos para su tratamiento:

Análisis léxico. Realiza el análisis de las palabras mediante la utilización de modelos computacionales de la morfología, generalmente basados en autómatas de estado finito y expresiones regulares, transductores de estado finito, transductores ponderados, modelos de Markov ocultos y n-gramas.

Análisis sintáctico. Realiza el agrupamiento de las palabras en sintagmas y frases mediante modelos computacionales como son las gramáticas independientes del contexto, las gramáticas lexicalizadas y las estructuras de rasgos.

Análisis semántico. Determina el significado de las frases de acuerdo con el significado de los sintagmas, palabras y morfemas que las forman, utilizando para ello modelos computacionales tales como la lógica de predicados de primer orden y las redes semánticas.

Análisis pragmático. Establece la identidad de las personas y objetos que aparecen en los textos, determina la estructura del discurso y gestiona el diálogo en un entorno conversacional.

En el caso de que estemos tratando con textos hablados, existiría un nivel previo de *reconocimiento del habla* y posiblemente un nivel posterior de *síntesis del habla*, que harían uso de conocimiento fonético y fonológico.

Es interesante destacar que los distinto niveles de análisis no tienen porqué ser totalmente independientes entre sí, ya que por ejemplo, el análisis

léxico puede ofrecer diferentes etiquetas para una palabra dada, dejando que sean el analizador sintáctico e incluso el semántico los encargados de determinar la más conveniente. Análogamente, ciertas estructuras generadas por un analizador sintáctico pueden no tener una interpretación semántica válida, por lo cual un analizador semántico puede ayudar a realizar la tarea de un analizador sintáctico.

1.3. Ambigüedad

En problema principal con el que se van a enfrentar los analizadores del lenguaje natural es el de la ambigüedad, que se presenta tanto a nivel léxico como sintáctico y semántico.

A nivel léxico, nos encontramos con que una palabra puede recibir diversas etiquetas. Por ejemplo, la palabra "sobre" puede ser un sustantivo masculino singular, una preposición, o la primera o ternera persona del presente de subjuntivo del verbo sobrar. En ciertos contextos la tarea de determinar la etiqueta correcta puede ser relativamente fácil, pero en frase como "pon lo que sobre sobre el sobre" se muestra la complejidad de este proceso. Un tipo de ambigüedad léxica que se da en ciertos idiomas se refiere a la existencia de posibles segmentaciones en palabras de ciertos fragmentos de un texto. Se trata de un problema habitual en las lenguas orientales que también se presenta en idiomas que hacen uso frecuente de contracciones o palabras compuestas. Por ejemplo, *polo* en gallego puede ser segmentado como:

- el sustantivo masculino singular *polo* (pollo);
- la contracción de la preposición *por* y el artículo determinado masculino singular *lo*;
- la contracción de infinitivo del verbo *por* (poner) y el pronombre enclítico de tercera persona *lo* (lo).

En francés, el texto *pomme de terre cuit* se puede segmentar de dos maneras posibles;

- en *pomme de terre* (patata) y *cuit* (cocida);
- en *pomme* (manzana), *de* y *terre cuit* (terracota).

En español, el problema de la segmentación ambigua se da principalmente en las locuciones. Por ejemplo, *sin embargo* se puede considerar en su conjunto como una locución o bien como las palabras individuales *sin* (preposición) y *embargo* (sustantivo).

En el caso de la sintaxis, el hecho de que una frase sea ambigua se traduce en que es posible asociar dos o más estructuras sintagmáticas correctas a dicha frase. Como ejemplo, tomaremos una frase conocida: "Juan vio a un hombre con un telescopio en una colina". Diferentes ubicaciones de las subestructuras correspondientes a los fragmentos "con un telescopio" y "en una colina" llevan a diferentes estructuras sintagmáticas completas para la frase, todas ellas correctas, que se corresponden con los significados siguientes:

- Juan vio a un hombre que estaba en una colina y que tenía un telescopio;
- Juan estaba en una colina, desde donde vio a un hombre que tenía un telescopio;
- Juan estaba en una colina, desde donde miraba con un telescopio, a través del cual vio a un hombre.

A nivel semántico, nos encontramos con que una palabra puede tener diferentes significados. Por ejemplo, la palabra "banda" puede referirse a:

- un grupo de personas;
- una tira de tela;
- los laterales de un barco;
- un conjunto de frecuencias del espectro radioeléctrico.

Como el significado de una frase se construye a partir de las aportaciones semánticas realizadas por las palabras que la componen, es preciso determinar el significado correcto de cada palabra. Sin embargo, el significado de una frase puede ser ambiguo incluso si las palabras que lo componen no lo son. Por ejemplo, la frase "todos los alumnos de la facultad hablan dos idiomas" tiene dos interpretaciones distintas:

- Existen dos idiomas L y L' tales que todos los alumnos de la facultad los hablan.
- Cada uno de los alumnos de la facultad habla un par de idiomas, pero dos estudiantes distintos pueden hablar idiomas distintos.

Las ambigüedades pueden ser locales o globales. Una *ambigüedad local* es aquella que surge en un momento del análisis pero que es eliminada posteriormente, al analizar una porción mayor del texto. Una *ambigüedad global* es aquella que permanece una vez terminado de analizar todo el texto.

1.4. Breve reseña histórica

A continuación, utilizaremos los cuatro niveles de conocimiento y análisis para realizar una visita guiada por el *estado del arte* del procesamiento del lenguaje natural.

1.4.1. El nivel léxico

Los primeros programas para el análisis morfológico utilizaban un enfoque basado en la eliminación de afijos. Por ejemplo, el analizador de Packard [173] para griego antiguo eliminaba iterativamente prefijos y sufijos de las palabras de entrada, buscando la raíz resultante en un lexicón. El sistema AMPLE [254] seguía una estrategia similar, pero utilizando un lexicón con todas las posibles variantes de cada morfema (alomorfos) junto con sus restricciones asociadas.

Los autómatas finitos surgen en los años 50 a partir del modelo de computación algorítmica propuesto por Turing [229], considerado por muchos como el fundamento de la informática moderna. Una máquina de Turing es un dispositivo abstracto con un control finito y una cinta de entrada/salida. En un movimiento, una máquina de Turing puede leer un símbolo de la cinta, escribir un símbolo diferente en la cinta, cambiar el estado y mover el cabezal de la cinta una posición a la derecha o a la izquierda. A diferencia de las máquinas de Turing, los autómatas finitos no pueden cambiar los símbolos de la cinta.

Inspirados por el trabajo de Turing, McCulloch y Pitts [150] definen un modelo simplificado de neurona, similar a los autómatas finitos, que consideraba a cada una de ellas como un *elemento de cálculo* que podía ser descrito en términos de la lógica proposicional. Cada neurona es un dispositivo binario, que puede estar activo o no, que toma entradas excitadoras o inhibidoras de otras neuronas y emite una salida si se supera un cierto umbral de activación. Basándose en la neurona de McCulloch y Pitts, Kleene [120] define el autómata finito y las expresiones regulares, demostrando su equivalencia. Los autómatas finitos no deterministas son introducidos por Rabin y Scott [188], quienes demuestran su equivalencia con respecto a los autómatas finitos deterministas.

Los autómatas finitos se han utilizado con éxito para el análisis léxico del lenguaje natural. El sistema KeCi [88] es un analizador morfológico para el turco que se guía por una representación de estado finito de los morfemas turcos, intentando encontrar un morfema que concuerde con cada una de las partes de la palabra. El sistema descrito por Graña et al. [77] realiza el análisis morfológico del español mediante un autómata finito no determinis-

ta. Vilares et al. muestran en [250, 251] cómo es posible verificar formalmente el correcto funcionamiento de este tipo de analizadores. Los autómatas finitos también constituyen un método eficiente y compacto de representación de diccionarios. Graña et al. [80] muestran cómo la utilización de algoritmos incrementales de compilación [60] hace de los autómatas finitos acíclicos mínimos numerados el método óptimo de implementación de diccionarios.

A pesar de su similitud con los autómatas finitos, los transductores de estado finito tienen un origen diferente. En 1954, Huffman [92] propuso lo que era en esencia una tabla de transición de estados como modelo del comportamiento de los circuitos secuenciales, tomando como base el trabajo de Shannon [209] sobre modelado algebraico de circuitos. Basándose en el trabajo de Turing y Shannon, y desconociendo el trabajo de Huffman, Moore [160] introduce el término *autómata finito* para una máquina con un número finito de estados, un alfabeto finito de símbolos de entrada y un alfabeto finito de símbolos de salida. El trabajo de Moore y Huffman es sintetizado y extendido por Mealy [151]. Aunque ambos tipos de transductores son equivalentes, el autómata finito diseñado por Moore y la extensión propuesta por Mealy difieren en un aspecto importante: en una máquina de Mealy, los símbolos de entrada/salida están asociados con las transiciones entre estados; en una máquina de Moore los símbolos de entrada/salida están asociados con los estados.

La idea de modelar las reglas de creación de palabras como transductores de estado finito se basa en el descubrimiento por parte de Johnson [100] de que cualquier sistema fonológico que no permita que sus reglas se apliquen a su propia salida (esto es, un sistema sin reglas recursivas) puede ser modelado con transductores de estado finito. Esta idea no atrae la atención de la comunidad científica hasta su redescubrimiento por parte de Kaplan y Kay [109, 110], cuyo trabajo es continuado por Koskenniemi [123], quien describe la morfología del finlandés mediante transductores de estado finito, posteriormente implementada por Karttunen [113] en el programa KIMMO. Antworth [18] aplica este enfoque, conocido como morfología de dos niveles, al inglés. Mohri y Pereira [158] introducen un marco probabilístico en los transductores de estado finito mediante la definición de los autómatas finitos ponderados. Mohri [157], así como Roche y Schabes [196], establecen los fundamentos matemáticos y proporcionan algoritmos adicionales, como por ejemplo los de minimización, para la aplicación de los transductores de estado finito en tareas de procesamiento del lenguaje natural.

Una vez reconocida una palabra, surge el problema de asignarle la categoría morfosintáctica (etiqueta) correcta de entre todas las posibles. El primer algoritmo para la asignación de etiquetas que se conoce estaba incorporado en el analizador sintáctico utilizado en el proyecto TDAP, implementado entre 1958 y 1969 en la Universidad de Pennsylvania [89]. Anteriormente, los sistemas de procesamiento del lenguaje natural utilizaban diccionarios con información morfológica de las palabras pero, que se sepa, no realizaban desambiguación de etiquetas. El sistema TDAP realizaba esta desambiguación mediante 14 reglas escritas a mano que eran ejecutadas en un orden basado en la frecuencia relativa de las etiquetas de cada palabra. El analizador del proyecto TDAP ha sido reimplementado recientemente por Joshi y Hopely [102] y por Karttunen [114], quienes destacan el hecho de que el analizador consistía básicamente en una cascada de transductores de estado finito. Curiosamente, se trata de una aproximación muy utilizada en los sistemas más recientes.

Poco después del TDAP surgió el sistema CGC de Klein y Simmons [121], con sus tres componentes: un lexicón, un analizador morfológico y un desambiguador por contexto. El pequeño diccionario de 1.500 palabras incluía aquellas palabras raras que no podían ser tratas por el analizador morfológico, tales como sustantivos, adjetivos y verbos irregulares. El analizador morfológico utilizaba los sufijos flexivos y derivativos para asignar un conjunto de etiquetas a cada palabra. En ese momento entraban en acción un conjunto de 500 reglas encargadas de seleccionar la etiqueta correcta, consultando para ello las islas de palabras contiguas no ambiguas. El juego de etiquetas constaba de 30 etiquetas.

El etiquetador TAGGIT [82] se basa en el trabajo de Klein y Simmons, aplicando la misma arquitectura pero incrementando el tamaño del diccionario y del juego de etiquetas, que pasa a constar de 87 etiquetas. Este etiquetador fue utilizado como herramienta de ayuda en la creación del corpus Brown.

Actuamente, es muy frecuente fundamentar el proceso de etiquetación en la utilización de procedimientos estadísticos basados en la probabilidad de aparición conjunta de secuencias de n palabras o n-gramas. La matemática subyacente a los n-gramas fue propuesta por primera vez por Markov [146], quien utilizó bigramas y trigramas para predecir si la siguiente letra de una palabra rusa sería una vocal o una consonante: Markov clasificó 20.000 letras como vocal o consonante y calculó las probabilidad con respecto a los bigramas y trigramas de que una letra dada vaya seguida de una vocal o una consonante. Shannon [210] aplicó los n-gramas para calcular aproximaciones a las secuencias de palabras en inglés. A partir de los años 50, y gracias al trabajo de Shannon, los modelos de Markov son ampliamente utilizados

para modelar secuencias de palabras. En décadas posteriores su uso decayó, principalmente debido a la argumentación de muchos lingüistas, entre ellos Chomsky [51, 52], de que los modelos de Markov eran incapaces de modelar completamente el conocimiento gramatical humano. El resurgir de los modelos de n-gramas se produce en los años 70 al hacerse públicos los trabajos realizados en el centro de investigación Thomas J. Watson de IBM [97, 23] y en la Universidad de Carnegie Mellon [24], en los que se utilizan con éxito n-gramas para tareas de reconocimiento del habla.

En los años 70 se compiló el corpus Lancaster-Oslo/Bergen (LOB) como un equivalente, en inglés británico, al corpus Brown. Para su etiquetación se utilizó el etiquetador CLAWS [148], basado en un algoritmo probabilístico que puede considerarse una aproximación al enfoque basado en la utilización de modelos de Morkov ocultos. El algoritmo utilizaba la probabilidad de aparición conjunta de dos etiquetas, pero en lugar de almacenar dicha probabilidad directamente, la clasificaba como *rara* ($P(\text{etiqueta}|\text{palabra}) < 0,01$), *infrecuente* ($0,01 \leq P(\text{etiqueta}|\text{palabra}) < 0,10$) o *normalmante frecuente* ($P(\text{etiqueta}|\text{palabra}) \geq 0,10$).

El etiquetador probabilístico de Church [53] seguía una aproximación muy próxima a la de los modelos de Markov ocultos, extendiendo la idea de CLAWS para asignar la probabilidad real a cada combinación palabra/etiqueta, utilizando el algoritmo de Viterbi para encontrar la mejor secuencia de etiquetas. Sin embargo, al igual que CLAWS, almacenaba la probabilidad de una etiqueta dada la palabra para calcular

$$P(\text{etiqueta} \mid \text{palabra}) \times P(\text{etiqueta} \mid n \text{ etiquetas anteriores})$$

en lugar de almacenar la probabilidad de una palabra dada la etiqueta, tal y como hacen los etiquetadores basados en modelos de Markov ocultos para calcular

$$P(\text{palabra} \mid \text{etiqueta}) \times P(\text{etiqueta} \mid n \text{ etiquetas anteriores})$$

Los etiquetadores posteriores ya introdujeron explícitamente la utilización de modelos de Markov ocultos. Los etiquetadores de Brants [39] y Graña [76] constituyen ejemplos de herramientas recientes de alto rendimiento que utilizan modelos de Markov ocultos basados en n-gramas.

En los últimos años, la investigación se ha centrado en definir modelos de n-gramas más sofisticados. Para ello se puede optar por dar más peso a los n-gramas que han aparecido más recientemente [128], utilizar disparadores de larga distancia en lugar de n-gramas locales [198], utilizar n-gramas de longitud variable [170] o definir n-gramas basados en clases [227]. Otros

enfoques tratan de enriquecer los n-gramas mediante la incorporación de información semántica [56]. Graña et al. [81] definen los ajustes necesarios para integrar diccionarios externos en un modelo de Markov oculto. Graña et al. presentan en [78] un método para el manejo de segmentaciones ambiguas en los modelos de Markov ocultos.

Un enfoque alternativo al modelo de n-gramas lo constituye la etiquetación basada en trasformaciones, usualmente llamada etiquetación de Brill por ser éste el autor que la popularizó [42]. Se trata de una instancia del aprendizaje basado en transformaciones. Este enfoque se basa en la utilización de reglas que especifican la etiqueta que deberá ser aplicada a cada palabra, con la particularidad de que las reglas no se codifican manualmente sino que son inferidas automáticamente a partir de textos de entrenamiento ya etiquetados. La principal desventaja de este enfoque radica en los grandes tiempos de entrenamiento necesarios para aprender las reglas y en los grandes tiempos de ejecución necesarios para la etiquetación, puesto que las reglas se aplican iterativamente, primero aquellas más generales, después algunas más específicas y así sucesivamente, de tal modo que una regla puede cambiar etiquetas establecidas anteriormente por otras reglas.

Otra técnica de etiquetación de buen rendimiento es la basada en el formalismo reduccionista no cuantitativo denominado *English Constraint Grammar* (EngCG) desarrollado en la Universidad de Helsinki [253]. EngCG utiliza un conjunto de reglas escritas a mano que manejan contexto global o, en la mayoría de las veces, contexto local. No existe por tanto una verdadera noción de gramática formal, sino más bien un conjunto de restricciones, casi siempre negativas, que van eliminando los análisis imposibles en el contexto [112]. Dicho conjunto de reglas se compila bajo la forma de un transductor de estado finito para su rápida aplicación, como se describe en Graña et al. [79]. La idea básica es similar al aprendizaje basado en transformaciones, excepto por el hecho de que es un humano, y no un algoritmo, el que modifica iterativamente el conjunto de reglas de etiquetación para minimizar el número de errores. En cada iteración, el conjunto de reglas se aplica al corpus y posteriormente se intentan modificar dichas reglas de manera que los errores más importantes queden manualmente corregidos.

La Real Academia Española está desarrollando un formalismo de reglas de restricción denominado RTAG [185], para la anotación automática del Corpus Diacrónico del Español (CORDE) y del Corpus de Referencia del Español Actual (CREA). Este sistema aplica gramáticas de reglas de contexto ponderadas sobre textos anotados ambiguamente. Esto quiere decir que cuando un contexto satisface la descripción estructural de una regla recibe

la puntuación que indica la regla. Esta puntuación puede ser positiva, para promover lecturas, o negativa, para penalizarlas. Finalizado el proceso, permanecen las lecturas con mayor puntuación siempre que aventajen a otra u otras por encima de un umbral definido previamente. El sistema intenta también eliminar lecturas imposibles en función del contexto, sin pérdida de lecturas posibles aunque improbables en ocasiones. Para esta poda de lecturas imposibles en función del contexto se utilizan, básicamente, tres tipos de información: información derivada del propio texto (es decir, de sus características estructurales, tipográficas o secuenciales), información gramatical local (fundamentalmente concordancia y restricciones de aparición conjunta) e información gramatical estructural (toma de decisiones con ayuda de la información estructural derivable de la secuencia lineal del texto). Otro etiquetador basado en gramáticas de restricciones que ha sido utilizado también con éxito sobre el español es el sistema Relax [174]. En este sistema, las reglas de restricción pueden ser escritas tanto de forma manual como generadas automáticamente mediante un algoritmo de adquisición basado en un árbol de decisión [147].

1.4.2. El nivel sintáctico

La tradición gramatical europea se basa en las relaciones entre palabras más que en las relaciones entre constituyentes. El psicólogo Wundt introdujo en 1900 la idea de estructurar las frases en una jerarquía de constituyentes. La introducción de esta idea en el ámbito de la lingüística se debe a Bloomsfield [34]. Dos décadas después, el análisis de constituyentes inmediatos ya era un método de estudio sintáctico bien establecido en los Estados Unidos. Este hecho contrasta con la insistencia por parte de los gramáticos europeos de poner el énfasis en las gramáticas basadas en dependencias entre palabras.

Los estructuralistas americanos buscaban un algoritmo que permitiese describir la sintaxis de un lenguaje, lo que llevó al establecimiento de múltiples definiciones concernientes a los constituyentes inmediatos. Una de las definiciones más conocidas establecía la distribución de similitud a las unidades individuales, utilizando para ello un test de sustitución. Esencialmente, el método consistía en romper una construcción en sus constituyentes, intentando sustituir estructuras simples por posible constituyentes. Por ejemplo, si una forma elemental como la palabra *hombre* puede ser sustituida en una construcción por una forma compleja como *hombre increíblemente joven*, entonces esta última forma compleja es con toda probabilidad un constituyente. Vemos como la intuición subyacente consistía en considerar que los constituyentes formaban una especie de clases de equivalencia.

La primera formalización de esta idea de los constituyentes jerárquicos la

realizó Chomsky [51] al definir en 1956 las gramáticas de estructura de frase. Desde entonces, la mayor parte de las teorías lingüísticas generativas se basan, al menos en parte, en las gramáticas independientes del contexto. Entre ellas podemos citar las gramáticas con estructura de frase dirigidas por el núcleo [184] y las gramáticas léxico-funcionales [111, 168]. Las gramáticas independientes del contexto fueron redescubiertas poco después por Backus [21] y Naur et al. [161] en sus descripciones del lenguaje de programación ALGOL.

A partir de los años 60, la mayor parte de los modelos computacionales para el procesamiento del lenguaje natural se basaron en gramáticas independientes del contexto debido a la disponibilidad de algoritmos eficientes para realizar el análisis de este tipo de gramáticas. Ya en 1955 Yngve [257] describió un algoritmo ascendente de análisis sintáctico como parte de un ejemplo de un procedimiento de traducción automática. Los primeros analizadores descendentes datan de principios de los años 60 [75, 95, 130]. Los analizadores sintácticos más eficientes para el lenguaje natural se basan en la utilización de programación dinámica. El algoritmo CYK se basa en los trabajos realizados por Cocke en los primeros años de la década de los 60. El trabajo se completa con las aportaciones de Younger y Kasami [258, 115]. Un algoritmo relacionado con el CYK es de las *tablas de subcadenas bien formadas*, propuesto por Kuno [129] como una estructura de datos para el almacenamiento de los resultados de todos los cálculos realizados en el curso de un análisis. La utilización de predicción descendente en algoritmos de análisis sintáctico basados en programación dinámica se debe a Earley [70]. Sheil [211] muestra la equivalencia entre el algoritmo de las tablas de subcadenas bien formadas y el algoritmo de Earley. Norvig [171] mostró que la eficiencia ofrecida por los analizadores sintácticos basados en programación dinámica podía obtenerse en cualquier lenguaje que dispusiese de un mecanismo de *memoization*, tal como LISP, simplemente mediante la memorización de los resultados obtenidos por un analizador descendente. Esta técnica es aprovechada por Leermakers [139, 137, 138] para implementar recursivamente varios algoritmos de análisis sintáctico. Otro enfoque seguido a lo largo del tiempo ha sido la utilización de cascadas de autómatas finitos [89], aunque no es hasta principios de los 90 cuando esta aproximación toma fuerza como alternativa real a los anlizadores completos para gramáticas independientes del contexto [3, 2, 38]. La eficiencia de los autómatas finitos hace que sean muy utilizados para realizar el análisis superficial de frases en tareas como la extracción de información [54, 55, 20, 90, 19].

Se ha tratado de aplicar al procesamiento del lenguaje natural extensiones de las técnicas utilizadas en la compilación de lenguajes de programación. A

este respecto destaca la definición por parte de Tomita [226, 190] del algoritmo LR generalizado para el tratamiento de gramáticas independientes del contexto ambiguas. El método utilizado resultó ser un caso particular de la técnica de programación dinámica diseñada por Lang [133] para la ejecución eficiente de autómatas a pila no deterministas, tal y como se observa en [33]. Vilares [245] extiende la técnica propuesta por Billot y Lang [33] a la clase de los autómatas LALR(1) y le añade capacidades incrementales [249, 247]. Alonso et al. [8, 7] explicitan las relaciones existentes entre el algoritmo de Earley y la interpretación de autómatas LR en programación dinámica. Una visión conjunta de la mayor parte de los algoritmos de análisis sintáctico para gramáticas independientes del contexto puede encontrarse en la obra de Sikkel [214].

Una de las extensiones habituales de las gramáticas independientes del contexto consiste en decorar las producciones y los árboles de análisis son probabilidades. Muchas de las propiedades formales de las gramáticas independientes del contexto probabilísticas fueron propuestas por Booth [35] y Salomaa [199] a finales de los años 60. Más tarde, Baker [25] propuso el algoritmo dentro-fuera para el entrenamiento no supervisado de gramáticas probabilísticas, que se basaba en un algoritmo de análisis de estilo CYK para calcular las probabilidades internas. Jelinek y Lafferty [98] extendieron al algoritmo CYK para calcular la probabilidad de los prefijos. Stolcke [219] se basó en ambos algoritmos para diseñar una versión probabilística del algoritmo de Earley. Por su parte, Briscoe y Carrol proponen una versión probabilística del algoritmo LR [43]. Tendeau [224] desarrolla las estructuras algebraicas que permiten manipular bosques de árboles de análisis sintáctico decorados con probabilidades, mostrando como ejemplos el algoritmo de Earley y los algoritmos LR y de la esquina-izquierda. Un enfoque distinto consiste en utilizar conjuntamente un modelo de n-gramas con cascadas de autómatas de estado finito para realizar un análisis sintáctico superficial [38, 260].

Las probabilidades también se han utilizado profusamente para intentar aprender las estructuras de subcategorización de los verbos. Con este fin surgen los algoritmos de Brent [40] y Manning [144], este último basado en la utilización de autómatas finitos. Briscoe y Carrol [44] extraen estructuras de subcategorización más complejas utilizando 160 posibles etiquetas de subcategorización y también aprenden la frecuencia de cada posible subcategorización mediante la utilización de un analizador LR probabilístico junto con cierto postprocesamiento. A este respecto, Roland y Jurafsky [197] mostraron que es importante calcular las probabilidades de cada subcategorización en función del sentido de la palabra más que en función de las realizaciones

ortográficas. El enfoque probabilístico también se ha utilizado para tratar de resolver el problema de la asociación de sintagmas preposicionales. Para ello se han aplicado enfoques tales como el aprendizaje basado en transformaciones [41], la máxima entropía [189], el aprendizaje basado en memoria [259], los árboles de decisión creados en función de la distancia semántica entre los núcleos de los sintagmas [218], y las técnicas de aprendizaje automático [1].

Las gramáticas de adjunción de árboles [105, 104, 106] son uno de los formalismos gramaticales derivados de las gramáticas independientes del contexto más ampliamente difundidos. En este tipo de gramáticas la estructura fundamental es el árbol, en lugar de la producción. Los árboles se clasifican en iniciales y auxiliares. Los árboles iniciales suelen utilizarse para representar las estructuras de las frases elementales, mientras que los árboles auxiliares se utilizan para representar estructuras recursivas mínimas que se pueden añadir a otros árboles. Los árboles se combinan mediante las operaciones de adjunción y sustitución. Desde el punto de vista lingüístico las grandes ventajas de las gramáticas de adjunción de árboles provienen de su carácter lexicalizado, ya que permiten asociar un terminal con cada árbol, y de su dominio de localidad extendido, que posibilitan el establecimiento de relaciones de larga distancia entre los nodos de árboles elementales. Las gramáticas de inserción de árboles [207, 208] constituyen una variante más restringida de las gramáticas de adjunción de árboles que reconoce exactamente la clase de los lenguajes independientes del contexto. Se han creado extensiones probabilísticas tanto para las gramáticas de adjunción de árboles [202, 191, 200, 167] como para las gramáticas de inserción de árboles [206, 208, 93], junto con algoritmos que permiten extraer de un banco de árboles los árboles elementales que componen la gramática [50], así como sus probabilidades asociadas [169]. Un enfoque diferente de aplicación de probabilidades lo constituyen los recientes intentos de fundir el proceso de etiquetación con el de análisis sintáctico de gramáticas de adjunción de árboles, mediante la utilización de un proceso de super-etiquetación que asigna a cada palabra, no sólo su etiqueta morfosintáctica, sino también el árbol elemental que le corresponde en la frase [26, 49, 87].

El primer algoritmo para el análisis sintáctico de gramáticas de adjunción de árboles en tiempo polinomial data de 1985 [235, 234]. Se trataba de una extensión del algoritmo CYK para este tipo de gramáticas, que en lugar de utilizar una tabla bidimensional manejaba una tabla cuadridimensional. Posteriormente aparecieron diversas adaptaciones del algoritmo de Earley, distinguiéndose rápidamente dos variantes, una que preservaba la propiedad del prefijo válido [203] y otra que no lo hacía [204, 201, 134]. Los prime-

ros tenían un coste temporal mayor, por lo que se entabló un debate acerca de si preservar la propiedad del prefijo válido implicaba inexorablemente incrementar la complejidad teórica de los algoritmo de análisis. Esta discusión quedó zanjada con la aparición del algoritmo de Nederhof [162, 165], que preservaba la propiedad del prefijo válido al tiempo que mantenía la complejidad temporal que aquellos que no la preservaban. Las relaciones entre todos estos algoritmos se muestran claramente en los trabajos de Alonso et al. [6, 10]. Otros algoritmos para gramáticas de adjunción de árboles se derivan de la familia de analizadores LR para gramáticas independientes del contexto. El primer intento de Schabes y Vijay-Shanker [205] en esta dirección resultó fallido. Tras varios intentos de reparar el algoritmo, como el llevado a cabo por Kinyon [119], es Nederhof [163] quien obtiene un analizador de tipo LR que funciona correctamente, aunque su aplicación práctica queda limitada por el excesivo tamaño de la tabla de análisis, problema que trata de resolver Prolo [186] modificando el procedimiento de generación de dicha tabla. Existe otro conjunto de estrategias de análisis sintáctico que se distinguen por realizar una lectura bidireccional de la cadena de entrada. Entre los algoritmos más destacados podemos citar el de Lavelli y Satta [136], el de van Noord [232], el de Lopez [141, 142] y el de Díaz et al. [69, 47]. Un estudio pormenorizado de este tipo de algoritmos puede encontrase en la tesis de Díaz [68].

En los últimos años se ha investigado mucho en torno a otro enfoque para realizar el análisis sintáctico de las gramáticas de adjunción de árboles, consistente en el análisis mediante autómatas. Los autómatas a pila embebidos [234, 205] constituyen la primera extensión de los autómatas a pila apta para este tipo de gramáticas. La técnica de programación dinámica que permite ejecutar eficientemente este tipo de autómatas fue propuesta por Alonso et al. [13, 14]. Una extensión distinta es la propuesta por Nederhof [164, 166] y Alonso et al. [17, 12], consistente en asociar una pila a cada elemento almacenado en la pila tradicional de los autómatas a pila. Otro modelo diferente es el propuesto por De la Clergerie y Alonso [63, 64], consistente en trabajar con dos pilas, teniendo cuidado de restringir las operaciones aplicables con el fin de no exceder la potencia generativa de las gramáticas de adjunción de árboles. Aunque la mayoría de las estrategias de análisis implementadas madiante autómatas son unidireccionales, mediante extensiones simples se pueden crear modelos de autómatas bidireccionales, tal y como se muestra en los trabajos de Alonso et al. [15, 16].

Existen multitud de formalismos equivalentes a las gramáticas de adjunción de árboles. Entre ellos destacan las gramáticas lineales de índices [74, 238, 11], las gramáticas categoriales combinatorias [217], las gramáticas de núcleo [193], las gramáticas de concatenación de rangos de aridad

2 [37, 36], los sistemas de matrices recursivos independientes del contexto de índice 2 [30, 29], las gramáticas independientes del contexto acopladas de rango 2 [182] y las gramáticas de control de nivel 2 [240]. Todos estos formalismos se engloban en la clase de los formalismos gramaticales suavemente sensibles al contexto [255, 107, 239].

La operación de unificación en entornos gramaticales fue descubierta independientemente por Kay [117] (unificación de estructuras de rasgos) y Colmerauer [58, 59] (unificación de términos lógicos). Ambos estaban trabajando en traducción automática y buscaban un formalismo reversible para combinar información lingüística. El sistema-Q de Colmerauer era básicamente un analizador ascendente basado en una serie de reglas de reescritura con variables lógicas, diseñadas para ser utilizadas en un sistema de traducción automática inglés-francés. El carácter reversible de las reglas de reescritura permitía que fuesen utilizadas tanto para análisis como para generación. Colmerauer y sus colegas diseñaron el lenguaje Prolog [57, 61, 248] a partir del sistema-Q mediante la adición de un mecanismo completo de unificación basado en el principio de resolución de Robinson [195], con el cual implementaron un analizador para el francés. El uso actual de Prolog y de la unificación de términos lógicos en procesamiento del lenguaje natural mediante las Gramáticas de Cláusulas Definidas descritas por Pereira y Warren [178, 177] tiene sus orígenes en las gramáticas de metamorfosis de Colmerauer [59]. Al mismo tiempo, Kay y Kaplan habían estado trabajando en las redes de transición aumentadas, que no son más que redes de transición recursivas en las cuales los nodos se decoran con registros que contienen rasgos. Con el fin de hacer reversible el proceso de asignación de valores a los registros, a ciertos registros sólo se les podía asignar un valor una sola vez, con lo que se aproximaban al concepto de variable lógica, aunque el algoritmo de unificación de Kay trabajaba con rasgos y no con términos lógicos. Posteriormente, el concepto de unificación se integró en algoritmos de análisis sintáctico como el de Earley [212, 213] o los de tipo LR [246, 9]. Por su parte, Lang [135] propone utilizar los autómatas lógicos a pila [62, 65], una extensión de los autómatas a pila en la que los elementos de la pila son términos lógicos, como modelo básico para la definición de algoritmos de análisis sintáctico para el lenguaje natural. En lo que respecta al trabajo sobre la unificación de estructuras de rasgos, destaca la aparición de las estructuras de rasgos con tipos, formalizadas por Carpenter [46].

Entre los formalismos más conocidos que utilizan unificación de estructuras de rasgos podemos destacar las gramáticas léxico-funcionales [111, 168], las gramáticas con estructura de frase dirigidas por el núcleo [184], las

gramáticas de adjunción de árboles con estruturas de rasgos [236, 237] y las gramáticas categoriales de unificación [231].

Existen formalismos gramaticales que no se basan en las gramáticas independientes del contexto. Las gramáticas contextuales [145] producen un lenguaje comenzando con un conjunto finito de palabras y añadiendo iterativamente contextos a las palabras generadas de acuerdo con un procedimiento de selección. Los lenguajes generados por las gramáticas contextuales no son comparables con los lenguajes de la jerarquía de Chomsky. Otro tipo de formalismo lo constituyen las gramáticas de dependencia [152], que se fundamentan en las relaciones existentes entre palabras y no en las relaciones entre constituyentes.

1.4.3. El nivel semántico

Una de las representaciones formales más utilizadas para capturar el significado de textos en lenguaje natural es la lógica de predicados de primer orden. Los primeros usos de representaciones declarativas del significado aparecen en el contexto de los sistemas de búsqueda de respuestas de los años 60 [215], en los que se empleaban representaciones ad-hoc para los hechos que se necesitaban en la tarea a realizar, mientras que las preguntas se convertían en un formato adecuado para su emparejamiento con los hechos almacenados en la base de conocimiento. Es Woods [256] quien en esos años investiga en la posibilidad de utilizar representaciones basadas en lógica de predicados para los sistemas de búsqueda de respuestas en lugar de las representaciones ad-hoc, plasmando sus ideas en el sistema Lunar de análisis composicional dirigido por la sintaxis. Al mismo tiempo, aquellos investigadores interesados en el modelado cognitivo del lenguaje y de la memoria trabajaban en varias formas de representación basadas en redes asociativas. De hecho, en ese periodo se investiga con profusión en el ámbito de las redes semánticas [149, 187]. Conforme se iban haciendo más sofisticadas y bien definidas, se hizo claro que tal y como se estaban utilizando, las redes semánticas eran en realidad una variante restringida del cálculo de predicados de primer orden acoplada con procedimientos de inferencia especializados.

Los esfuerzos lingüísticos para asignar estructuras semánticas al lenguaje natural en la era generativa comenzaron con el trabajo de Katz y Fodor [116]. Las limitaciones de su representación, basada en estructuras de rasgos simples, y el acomodo natural de la lógica a muchos problemas lingüísticos de la época hizo que se adoptasen diversas estructuras basadas en predicados como representación semántica preferida [132]. La introducción por parte de

Montague [159] de la estructura de modelos teóricos en la teoría lingüística llevó a una integración mucho más fuerte entre las teorías de la sintaxis formal y un amplio rango de estructuras semánticas.

Pronto se constató que resolver el problema del análisis semántico, esto es, de componer las representaciones del significado y asignarlas a las frases, no era tarea fácil. Uno de los enfoques más utilizados es el denominado análisis semántico dirigido por la sintaxis, en el cual la asignación de significados se basa en estructuras estáticas como son el lexicón y la gramática. La clave de este enfoque radica en el principio de composicionalidad, según el cual el significado de una frase puede ser obtenido mediante la composición de los significados de sus sintagmas constituyentes. Montague [159] mostró que el enfoque composicional podía ser aplicado a una parte importante del lenguaje natural. Como ocurrió en el caso del análisis sintáctico de gramáticas independientes del contexto, el análisis composicional se desarrolló en paralelo en el ámbito del procesamiento del lenguaje natural y en el de la compilación de lenguajes de programación. En este último ámbito, Knuth [122] introdujo la noción de gramáticas de atributos en las cuales se establecía una asociación uno-a-uno entre las estructuras semánticas y las estructuras sintácticas.

En ciertas aplicaciones en las que se utiliza un lenguaje controlado en el que la interpretación de las frases depende más de las palabras que intervienen que de la estructura sintáctica, se ha visto que una estructura arborescente detallada dificulta la obtención del significado de la frase. Esta situación se da, por ejemplo, en los sistemas automáticos de atención de pedidos. En estos entornos se suelen utilizar *gramáticas semánticas* [45], en las que las producciones y los constituyentes tratan de corresponder lo más fielmente posible a entidades y relaciones del dominio de aplicación. Se busca por tanto que los componentes semánticos clave ocurran juntos en una misma producción. Las producciones, por su parte, tienden a ser más grandes de lo habitual, lo que permite generar árboles más planos, utilizando categoría gramaticales que están motivadas por su contribución semántica y no por su papel morfosintáctico en la frase.

En ciertas tareas de procesamiento del lenguaje no se precisa obtener un conocimiento detallada sobre el significado de cada frase. Ejemplos de tales tareas son la extracción de información sobre fusiones de empresas a partir de noticias de agencia, la comprensión de informes del tiempo, o la obtención de resúmenes sencillos sobre los acontecimientos de la bolsa en determinados días a partir de boletines informativos. En general, las tareas de *extracción de información* se caracterizan por poseer dos propiedades fundamentales: el conocimiento que se desea extraer se puede describir mediante un conjunto

relativamente simple y fijo de fichas, con apartados que deben ser cubiertos a partir de material extraído de determinados textos, y sólo una pequeña parte de la información contenida en los textos es relevante para completar las fichas, por lo que el resto puede ser ignorado. Los primeros sistemas de extracción de información proporcionaban como salida una única ficha. Actualmente, se pide que obtengan una jerarquía de fichas, de tal modo que ciertos apartados de una fichas pueden contener enlaces a otras fichas.

Muchos sistemas de extracción de información se organizan en torno a cascadas de autómatas finitos. Por ejemplo, el sistema FASTUS [20, 90] utiliza estas cascadas para realizar los siguientes niveles de procesamiento: (1) reconocimiento de palabras complejas, incluyendo nombres propios y términos compuestos; (2) reconocimiento de sintagmas básicos, principalmente sintagmas nominales, grupos verbales y ciertas partículas como preposiciones, conjunciones y pronombres ralativos; y (3) reconocimiento de sintagmas complejos, incluyendo grupos nominales complejos y grupos verbales complejos. Una vez extraídos los sintagmas complejos en el nivel (3), estos son emparejados con los patrones de eventos de interés para la aplicación, de tal modo que cuando un emparejamiento tiene éxito se extrae cierta información que sirve para rellenar unas fichas determinadas. Posteriormente, se aplica una fase de fusión de las diferentes fichas obtenidas, con el fin de producir las fichas finales, teniendo en cuenta las referencias anafóricas entre los elementos del texto [108]. Otros sistemas de éxito, como IE^2 [19] también realizan un reconocimiento de sintagmas mediante autómatas finitos, pasando a continuación a tratar de resolver ciertos tipos de anáfora y a realizar la fusión de fichas, en este caso tomando como base las relaciones temporales y espaciales entre los diferentes eventos. El enfoque basado en análisis sintáctico parcial ha tenido tal éxito que aquellos grupos de investigación que basaban sus sistemas en analizadores sintácticos completos han tenido que cambiar de estrategia [85]. Actualmente, incluso aquellos sistemas de extracción de información que pretenden realizar un análisis completo de las frases aplican primero una cascada de autómatas finitos para reconocer sintagmas que después son combinados con el fin de obtener la representación final de la frase [54, 55].

Otro entorno aplicativo en el cual se trata de capturar la semántica de los textos es el de la *recuperación de información* [233], un campo en constante crecimiento con múltiples facetas relacionadas con el almacenamiento y la recuperación de toda clase de textos. La mayor parte de los sistemas de recuperación de información actuales están basados en una interpretación extrema del principio de la semántica composicional, ya que consideran que

la semántica de los documentos reside únicamente en las palabras que lo forman, sin tener en cuenta el orden de los constituyentes ni su estructura sintáctica. Es lo que se conoce habitualmente como aproximación basada en una bolsa de palabras, para la cual se han desarrollado modelos matemáticos bien establecidos como son el vectorial, el booleano o el probabilístico [22, 73].

Puesto que las colecciones de documentos sobre las que trabajan los sistemas de recuperación de información contienen textos en lenguaje natural, habitualmente mal estructurados y con ambigüedades a nivel léxico, sintáctico y semántico, parece que la aplicación de técnicas de procesamiento de lenguaje natural debería incrementar el rendimiento de estos sistemas [140, 216]. A este respecto, debemos comentar que la mayor parte de la investigación en recuperación de información se ha realizado tomando como lengua base el inglés. Sin embargo, se ha detectado que la aplicación de conocimiento lingüístico al campo de la recuperación de información da mejor resultado cuando se trata con lenguas que poseen una estructura morfosintáctica más compleja, como el español [244, 243], el francés [96] o el holandés [124]. Se ha comprobado que en estos idiomas la aplicación de técnicas de normalización de palabras basadas en la morfología flexiva y derivativa [72, 125], así como de técnicas de normalización de términos multipalabras basadas en la estructura sintáctica básica de los grupos nominales [126, 183] y sus variantes [242] incrementa el rendimiento de los sistemas de recuperación de información [241].

Una arquitectura típica para sistemas de recuperación que aplican conocimiento lingüístico sería la propuesta por Strzalkowski et al. [179], que consta de cinco fases: (1) eliminación de aquellas palabras sin contenido semántico; (2) normalización morfológica, tanto flexiva como derivativa; (3) extracción de nombres compuestos [131, 67]; (4) extracción de relaciones de dependencia entre los núcleos de los sintagmas nominales y los de sus modificadores, entre el núcleo del sujeto y el verbo, y entre el verbo y el núcleo de sus complementos [242, 156]; y (5) extracción de nombres propios [225, 181]. Los resultados obtenidos por cada una de estas fases se indexan tanto por separado [220] como conjuntamente [222]. Adicionalmente, se puede considerar la reformulación de la consulta mediante información extraíada de aquellos documentos que se consideran relevantes [221]. Otras técncias que se pueden aplicar van desde la expansión de las consultas mediantes el empleo de palabras relacionadas [32, 99], al análisis cuidadoso de la estructura sintáctica de las consultas [156] y la utilización combinada de técnicas de extracción y de recuperación de información [28]. Un desafío actual consiste en desarrollar sistemas que puedan buscar automáticamente información en documentos escritos en diferentes idiomas [83, 84, 180, 127].

Las palabras pueden tener diversos significados según el contexto en el que se utilicen, hecho que constituye uno de los principales problemas con que nos encontramos al tratar de realizar el análisis semántico de una frase Las técnicas de *desambiguación del sentido de las palabras* tratan de resolver esta ambigüedad léxica seleccionando el sentido adecuado de cada palabra en una frase. La complejidad de esta tarea viene determinada por la cantidad de palabras homónimas y polisémicas presentes en el vocabulario de todo idioma. En esencia, se aplican técnicas similares a las utilizadas para realizar la etiquetación de las palabras en el nivel léxico, pero en lugar de utilizar etiquetas morfosintácticas se utilizan identificadores del sentido de las palabras. Por tanto se tratará de obtener el sentido más probable de una palabra en relación con los sentidos de las palabras vecinas. En cuanto a su aplicación práctica, en particular en los sistemas de extracción de información, Kilgarriff [118] argumenta que debe considerarse, por una parte el vocabulario propio del dominio de la aplicación, que contendrá aquellas palabras claves para rellenar los apartados de las fichas, y por otra parte el vocabulario general. Las palabras del vocabulario del dominio raramente deberán desambiaguarse, ya que la obtención de una interpretación semántica coherente eliminará automáticamente los sentidos no válidos. En cambio, para el vocabulario general deben aplicarse técnicas generales de desambiguación del sentido de las palabras [94, 192, 176].

Para determinar el sentido correcto de una palabra, primero debemos tener una fuente de la que obtener todos sus posibles sentidos. Uno de los recursos lingüísticos más utilizados para esta tarea es WordNet [154], una red semántica en la que los nodos son *synsets*, conjuntos de palabras sinónimas. Una palabra polisémica estará por tanto representada en varios synsets. Si aceptamos que los posibles sentidos de una palabra vienen determinados por los synsets en los que aparece, la tarea de desambiguación del sentido consistirá en determinar el synset correcto para cada una de las palabras que componen una frase. Para ello también se puede hacer uso de las relaciones jerárquicas de hiponimia-hiperonimia y de meronimia-holonimia existentes entre los synsets, así como de las relaciones de antomia existentes entre las palabras concretas. La WordNet original se diseñó [31] con el fin de relacionar los sustantivos [153], adjetivos [86] y verbos [71] de la lengua inglesa. Posteriormente, surgió EuroWordNet [252] con la intención de realizar redes semánticas interconectadas para las principales lenguas europeas.

1.4.4. El nivel pragmático

La pragmática es el estudio de la relación entre el lenguaje y el contexto en el que se utiliza. El contexto incluye elementos como la identidad de

las personas y los objetos participantes, y por tanto la pragmática incluye el estudio de cómo se utiliza el lenguaje para referenciar a personas y cosas. También incluye el contexto del discurso, y por consiguiente el estudio de cómo se estructura el discurso y de cómo los participantes en una conversación gestionan el diálogo. En consecuencia, para realizar el análisis pragmático se precisa de algoritmos para la resolución de la anáfora, modelos computacionales para recuperar la estructura de monólogos y diálogos, y modelos de gestión del diálogo.

La correcta interpretación de la anáfora es importante para procesar correctamente textos escritos en lenguaje natural [155]. Se trata de un problema que ha atraído el interés de los investigadores en los últimos 10 años, debido principalmente a su utilidad en la realización de tareas como la extracción de información y la creación de resúmenes de textos. Los primeros trabajos sobre resolución de la anáfora trataban de explotar el conocimiento lingüístico y del dominio que se tenía, el cual era difícil tanto de representar como de procesar y requería una notable participación humana. La necesidad de desarrollar soluciones robustas de bajo coste computacional hizo que muchos investigadores se embarcasen en el tratamiento de la anáfora mediante técnicas que hacían uso de pocos recursos lingüísticos. Este enfoque vino también motivado por la existencia de herramientas fiables y eficientes para el tratamiento de corpus, tales como etiquetadores-lematizadores y analizadores sintácticos superficiales. De hecho, la disponibilidad de corpus, tanto si estaban anotados con enlaces referenciales como si no, impulsó en gran medida el desarrollo de técnicas automáticas para el tratamiento de la anáfora, ya que son una fuente insustituible de información empírica que permite desarrollar métodos de aprendizaje automático, a la vez que proporcionan un medio adecuado para realizar la evaluación de las soluciones implementadas. En lo que respecta al tratamiento de la anáfora en español, destaca el trabajo de Palomar et al. [175], que se basa en la estructura obtenida por un analizador sintáctico parcial para manejar los fenómenos anafóricos producidos por los pronombres personales en tercera persona, así como por los pronombres demostrativos, reflexivos y elípticos, estableciendo un conjunto de preferencias para encontrar la entidad referenciada por cada uno de ellos.

El trabajo inicial en procesamiento del lenguaje natural prestó muy poca atención al estudio del diálogo. Uno de los primeros sistemas conversacionales, Eliza, estaba realizado en torno a un sistema trivial de gestión del diálogo: si la última frase del usuario humano emparejaba con una precondición, expresada en la forma de una expresión regular, de una posible respuesta, Eliza

simplemente procedía a generar esa respuesta. El gestor de diálogo para la simulación del agente paranoico Parry era un poco más complejo. Como Eliza, estaba basado en un sistema de producción, pero mientras que las reglas de Eliza estaban basadas en las palabras de la última frase pronunciada por la persona, las reglas de Parry también tenían en cuenta el valor de ciertas variables globales que indicaban su estado emocional. Es más, la salida de Parry también podía hacer uso de guiones para generar secuencias de frases en aquellos casos en los que la conversación derivaba en delirio. No surgieron gestores del diálogo más sofisticados hasta que no se tuvo una mejor comprensión de los mecanismos del diálogo humano. Se estableció el concepto de subdiálogo y se observó que los diálogos orientados a la realización de una determinada tarea presentaban una estructura cercana a la de la tarea que estaba siendo realizada. En general, el tratamiento del diálogo es similar al tratamiento del monólogo, pero con dificultades añadidas, ya que por ejemplo el tratamiento de la anáfora requiere analizar tanto el texto del actuante como de los otros intervinientes en el diálogo.

El análisis pragmático es necesario para realizar sistemas de traducción automática de calidad. Ya a finales de los años 40, poco después del nacimiento del ordenador electrónico, se concibió seriamente la posibilidad de crear sistemas de traducción automática. Las primeras demostraciones públicas de sistemas de este tipo, realizadas a mediados de los años 50, tuvieron un amplio eco en la prensa del momento. En esa década se desarrollaron muchas ideas que prefigurarían la mayor parte de los desarrollos posteriores. Desgraciadamente, el desafío iba muy por delante de los recursos disponibles en la época, ya que, por ejemplo, no existía ningún modo práctico y económico de almacenar la enorme cantidad de información requerida por un diccionario. Se tardó poco tiempo en observar que la creación de sistemas fiables de traducción automática era una tarea más difícil de lo inicialmente pensado, llegándose a la conclusión de que se necesitaba realizar más investigación básica en los nuevos campos de la lingüística formal y computacional, lo que a su vez llevó a un importante recorte en los fondos destinados a la investigación en traducción automática. Se produjo entones una pérdida de interés por este tema, llegando incluso al extremo de cambiar el nombre de la Asociación para la Traducción Automática y la Lingüística Computacional por el de Asociación para la Lingüística Computacional, nombre que se ha mantenido hasta la actualidad. Sin embargo, algunos investigadores perseveraron, consiguiendo mejorar poco a poco sus sistemas e incrementar lentamente su base de clientes. En particular, el sistema Systran fue mejorado continuamente durante 40 años. Sus primeros usos fueron como sistema

de adquisición de información, originalmente escrita en ruso, para el ejército norteamericano. En 1976 su versión inglés-francés fue adoptada por la entonces Comunidad Económica Europea para crear traducciones primarias, pensadas para ser editadas manualmente a posteriori, de varios documentos administrativos. Otro sistema exitoso fue el canadiense Météo, utilizado para traducir boletines meteorológicos del inglés al francés.

Durante los años 80 renace el interés académico por la traducción automática. Se trataba principalmente de utilizar las ideas subyacentes en las técnicas de inteligencia artificial que habían sido desarrolladas originalmente para la comprensión de historias y la ingeniería del conocimiento. Se trataba de enfoques basados en la utilización de una interlíngua, fundamentados en la idea de que los sistemas de traducción automática debían traducir de la misma manera que lo hace la gente, aunque lamentablemente los procesos mentales de la traducción son enormemente complejos, en gran parte desconocidos, y actualmente se duda de su relevancia para la traducción automática. Este interés en técnicas basadas en el significado puede verse también como una reacción al dominio que la sintaxis ejercía sobre la lingüística computacional en aquellos años.

Durante los años 90 se produce un auge en la utilización de métodos estadísticos como consecuencia de la disponibilidad de grandes corpus. En esta época las investigaciones se centran principalmente en la tarea de desarrollar sistemas de traducción automática con interfaz vocal, siguiendo la concepción moderna de los ordenadores como dispositivos fuertemente interactivos. La utilización de los sistemas de traducción automática se ha ido incrementando paulatinamente, debido al aumento de las relaciones comerciales internacionales, a la puesta en práctica de políticas gubernamentales que propician la traducción de documentos oficiales a varias lenguas, a la proliferación de los ordenadores personales y con ellos de aplicaciones de procesamiento de textos disponibles para todo el mundo, y a la difusión mediante Internet de una ingente cantidad de información en formato electrónico.

Capítulo 2

Planificación de un curso de PLN

Tal y como se ha señalado en el capítulo anterior, el procesamiento del lenguaje natural involucra a aquellas técnicas computacionales que procesan el lenguaje humano *en tanto que lenguaje*. Esta definición incluye desde tareas aparentemente sencillas como la separación de las palabras que conforman un texto, hasta tareas extremadamente complejas como la traducción automática. Lo que distingue a las aplicaciones de procesamiento del lenguaje natural de otro tipo de aplicaciones es el uso que hacen del *conocimiento del lenguaje*. Un programa que cuente los caracteres o líneas de un texto es una aplicación ordinaria de procesamiento de datos. Sin embargo, un programa que identifique las palabras escritas en español deberá utilizar conocimiento sobre lo que significa ser una palabra en español, tratando fenómenos tales como las contracciones y la presencia de pronombres enclíticos. Cuanto más elaborada sea la aplicación, más conocimiento de los niveles léxico, sintáctico, semántico y pragmático del lenguaje deberá manejar. Todas estas son consideraciones que deben ser tenidas en cuenta a la hora de planificar un curso sobre PLN.

2.1. El PLN en el Computing Science Curriculum 2008

En el ámbito internacional, el *Computing Science Curriculum* es la obra de referencia para el desarrollo de planes de estudios para la enseñanza de la informática a nivel universitario.

En otoño de 1998, la Education Activities Board de la Computer Society del IEEE (IEEE-CS) y la Education Board de la Association for Computing

Machinery (ACM) establecieron la Joint Task Force on Computing Curricula 2001 con el fin de realizar una revisión general de las líneas curriculares para los planes de estudios universitarios en informática [194]. Su objetivo era revisar el *Computing Curricula 1991* [230, 228, 66] con el fin de incorporar los desarrollos acaecidos en la última década del siglo XX. El resultado fue la definición del *Computing Curricula 2001* (CC001) [172], que a diferencia de la visión integradora presente en el informe de 1991, se divide en cuatro volúmenes diferentes: informática, ingeniería de computadores, ingeniería del software y sistemas de información. Posteriormente se creó una Review Task Force con el fin de mantener actualizado el contenido del CC2001. El resutado final ha sido la definición en diciembre de 2008 del *Computer Science Curriculum 2008* (CS2008) [223].

La propuesta del CC2001 se basa en los siguientes principios, que siguen vigentes en el CS2008:

1. Computación es un campo muy amplio que se extiende más allá de los límites de la informática.

2. La informática se fundamenta en una amplia variedad de disciplinas. Todos los estudiantes de informática deben aprender a integrar teoría y práctica, a reconocer la importancia de la abstracción y a apreciar el valor del buen diseño de ingeniería.

3. La rápida evolución de la informática requiere una revisión continua del correspondiente currículo.

4. El desarrollo del currículo en informática debe ser sensible a los cambios en la tecnología, a los nuevos desarrollos en la pedagogía y a la importancia de la formación contínua.

5. Se debe ir más allá de las unidades de conocimiento para ofrecer una guía significativa en términos de diseño de cursos individuales.

6. Se debe tratar de identificar las habilidades y el conocimiento básico que todo estudiante de informática debe poseer.

7. El cuerpo de conocimiento requerido debe ser tan pequeño como sea posible.

8. Se debe buscar el tener un alcance internacional.

9. Debe haber un amplio apoyo por parte de la industria, la administración y las instituciones de enseñanza superior involucradas en la formación informática.

10. Debe incluirse la práctica profesional como un componente integral del currículo.

11. Deben incluirse discusiones sobre las estrategias y tácticas de implementación además de las recomendaciones de alto nivel.

En lo que respecta a su contenido, el CS2008 se divide en las siguientes 14 *áreas*:

DS Estructuras discretas

PF Fundamentos de programación

AL Algoritmos y complejidad

AR Arquitectura y organización

OS Sistemas Operativos

NC Computación centrada en la red

PL Lenguajes de programación

HC Interacción hombre-máquina

GV Gráficos y computación visual

IS Sistemas inteligentes

IM Gestión de la información

SP Aspectos sociales y profesionales

SE Ingeniería del software

CN Ciencia computacional

Cada una de ellas se divide a su vez en *unidades*, que a su vez se componen de *descriptores*. Siguiendo los principios de diseño enumerados anteriormente y con el fin de crear un cuerpo de conocimiento tan pequeño como sea posible, aquellas unidades para las cuales existe un consenso generalizado de que el material por ellas abarcado es esencial para todo aquel que desee obtener una titulación en informática, son incluidas dentro del *núcleo* del CS2008, mientras que el resto son consideradas *opcionales*.

Aunuqe también aparece un descriptor de PLN en la unidad HC/MultimediaAndMultimodalSystems (sistemas multimedia y multimodales) del área de Interacción Hombre-Máquina, el grueso de los contenidos

sobre PLN se enmarcan en el área *Sistemas Inteligentes*, que se desarrolla a continuación:

IS Sistemas Inteligentes (10 horas núcleo)

IS/FundamntalIssues Cuestiones fundamentales en sistemas inteligentes (1 hora núcleo)

IS/BasicSearchStrategies Búsqueda y satisfacción de restricciones (5 horas núcleo)

IS/KnowledgeBasedReasoning Representación del conocimiento y razonamiento (4 horas nucleo)

IS/AdvancedSearch Búsqueda avanzada

IS/AdvancedReasoning Representación avanzadas del conocimiento y del razonamiento

IS/Agents Agentes

IS/NaturaLanguageProcessing Procesamiento del lenguaje natural

IS/MachineLearning Aprendizaje automático

IS/PlanningSystems Sistemas de planificación

IS/Robotics Robótica

IS/Perception Percepción

Es por ello que Procedemos a describir en detalle la unidad *Procesamiento del lenguaje natural*.

IS/NaturalLanguageProcessing (optativa)
Tiempo mínimo de cobertura: no se especifica
Descriptores:

- Gramáticas deterministas y estocásticas
- Algoritmos de análisis sintáctico
- Métodos basados en corpus
- Recuperación y extracción de información
- Traducción automática
- Reconocimiento del habla

Objetivos de aprendizaje:

1. Definir y contrastar gramáticas deterministas y estocásticas, proporcionando ejemplos para mostrar la adecuación de cada una de ellas.

2. Identificar los algoritmos clásicos de análisis sintáctico del lenguaje natural.

3. Defender la necesidad de un corpus estándar.

4. Dar ejemplos de procedimiento de búsqueda y catalogación en un enfoque basado en corpus.

5. Articular la distinción entre técnicas para recuperación de información, traducción automática y reconocimiento del habla.

En esta obra cubrimos todos los descriptores de la unidad, excepto el correspondiente al reconocimiento del habla, que dejamos aparte por corresponder a los contenidos que se estudian habitualmente en asignaturas relacionadas con el procesado de la señal.

2.2. El PLN en los planes de estudios

La docencia sobre PLN se realiza en asignaturas que con diversas denominaciones se suelen impartir como optativas de segundo ciclo de la titulación de Ingeniería Informática, con una carga docente que varía desde los 4,5 hasta los 7,5 créditos. Con la actual adaptación de las titulaciones universitarias al Espacio Europeo de Educación Superior (EEES), algunas de esas asignaturas estan siendo reconvertidas en optativas de titulaciones de Grado mientras que otras han sido transformadas en asignaturas de títulos de Náster.

Por tomar un caso concreto, en la Facultad de Informática de la Universidade da Coruña se imparte actualmente:

- una asignatura optativa cuatrimestral en el segundo ciclo de la titulación de Ingeniría Informátiva, denominada *Lenguajes Naturales*, con una carga lectiva de 6 créditos (4 de teoría y 2 de práctica);

- una asignatura optativa en el Máster en Informática, denominada también *Lenguajes Naturales*, con una carga lectiva de 3 créditos ECTS (dos de carácter teórico y uno de carácter práctico))

Respecto a la relación del PLN con los contenidos de otras materias que se imparten en los planes de estudios de las titulaciones en informática, debemos mencionar que una base fundamental para la buena comprensión del programa la constituye el bagaje teórico proporcionado por las asignaturas que tratan los conceptos de la *teoría de autómatas y lenguajes formales*, ya que buena parte de las técnicas utilizadas en el procesamiento del lenguaje natural surgen como extensión de técnicas utilizadas en el ámbito de los lenguajes formales. La docencia del PLN permite establecer complementariedades con concocimientos adquiridos en otros materias. Por ejemplo, el PLN permite al alumno aplicar técnicas y principios generales de la *inteligencia artificial* a una problemática concreta. También permite establecer una analogía entre las diferentes fases de procesamiento del lenguaje natural y las diferentes fases de la *compilación* de un programa de ordenador, al tiempo que permite al alumno ver cómo se pueden ampliar algunas de las técnicas utilizadas en la construcción de compiladores para que sean útiles en el análisis de textos escritos en lenguaje natural.

2.3. Nuestra propuesta

Tras tener en cuenta todas las consideraciones mostradas en las secciones previas, creemos que los objetivos generales de una asignatura sobre PLN en titulaciones informáticas deben ser los siguientes:

- Diferenciar entre lenguajes naturales y lenguajes artificiales.
- Comprender la complejidad del lenguaje humano.
- Familiarizarse con la terminología lingüística.
- Conocer los algoritmos, técnicas y métodos más utilizados actualmente para el tratamiento automático de los fenómenos léxicos, sintácticos y semánticos del lenguaje humano.
- Asumir que el procesamiento del lenguaje natural no se puede automatizar completamente, pero que se pueden desarrollar soluciones satisfactorias en la práctica.
- Identificar los aspectos del lenguaje sobre los que debemos trabajar más para obtener sistemas de procesamiento del lenguaje natural útiles.
- Reflexionar sobre los avances realizados en el campo y los errores cometidos a lo largo de las últimas décadas

En la práctica, estos objetivos suponen realizar un estudio de los conceptos, modelos y técnicas para la realización del análisis léxico, sintáctico, semántico y pragmático; y en proporcionar el bagaje suficiente que permita utilizar los conocimientos adquiridos en el desarrollo de aplicaciones basadas en lenguaje natural, en particular, aplicaciones concernientes a los campos de recuperación y extracción de información.

Para ello se ha desarrollado un programa que cubre las recomendaciones del CS2008:

1. Introducción al procesamiento del lenguaje natural
    1.1. Niveles de análisis
    1.2. Ambigüedad
    1.3. Breve reseña histórica
2. Análisis léxico
    2.1. Segmentación de textos
    2.2. Morfología flexiva y derivativa
    2.3. Modelización de grandes diccionarios
    2.4. Autómatas finitos acíclicos deterministas numerados
    2.5. Transductores de estado finito y morfología de dos niveles
3. Etiquetación
    3.1. Modelos de Markov ocultos
    3.2. Ejecución eficiente de los modelos de Markov ocultos
    3.3. Técnicas de suavizado
    3.4. Tratamiento de palabras desconocidas
    3.5. Aprendizaje de etiquetas basado en transformaciones y dirigido por el error
4. Análisis sintáctico: gramáticas independientes del contexto
    4.1. Esquemas de análisis sintáctico
    4.2. Análisis ascendente
    4.3. El algoritmo de Earley
    4.4. Autómatas a pila y programación dinámica
    4.4. Análisis sintáctico LR generalizado

   4.5. Representación compartida de los árboles de análisis sintáctico
   4.6. Análisis sintáctico probabilístico

5. Análisis sintáctico: gramáticas suavemente dependientes del contexto
   5.1. Gramáticas de adjunción de árboles
   5.2. Análisis sintáctico de gramáticas de adjunción de árboles
   5.3. Autómatas para las gramáticas de adjunción de árboles
   5.4. Representación compartida de los árboles de derivación
   5.5. Gramáticas de adjunción de árboles probabilísticas

6. Análisis semántico
   6.1. Estructuras de rasgos y formalismos basados en unificación
   6.2. Lógica de predicados de primer orden
   6.3. Relaciones léxicas: WordNet
   6.4. Desambiguación del sentido de las palabras

7. Recuperación y extracción de información
   7.1. Modelos de recuperación de información
   7.2. Aplicación de la morfología a la normalización de términos simples
   7.3. Aplicación de la sintaxis a la normalización de términos multipalabra
   7.4. Extracción de información [Una hora]

8. Análisis pragmático
   8.1. Resolución de la anáfora
   8.2. Traducción automática

La estructura de este programa nos servirá de guía en el resto de este libro, de tal modo que cada capítulo desarrolla brevemente los contenidos de uno de los temas del programa.

Capítulo 3

Análisis léxico

Las palabras constituyen los ladrillos del lenguaje. Todo lenguaje humano, sea hablado o escrito, se compone de palabras. En este tema nos ocuparemos de aquellas técnicas del procesamiento del lenguaje natural que tratan del reconocimiento de las palabras que forman los textos.

3.1. Segmentación de textos

Muchas aplicaciones de procesamiento del lenguaje natural presuponen que los textos ya están correctamente segmentados en *tokens*, unidades de información de alto nivel que identifican los componentes individuales de los textos. Esta creencia es errónea puesto que el concepto lingüístico de palabra no siempre coincide con la realización gráfica de palabra. Para lograr una segmentación correcta es necesario realizar las siguientes acciones:

- *Tokenización*: cada palabra individual, así como las marcas de puntuación, constituirán un *token* diferente, tratando las abreviaturas, los números en formato decimal y las fechas en formato numérico.
- *Segmentación de frases*: como regla general, se considera que una frase termina en cuanto aparece un punto seguido de una letra en mayúscula, pero hay que tratar las confusiones planteadas por la presencia de abreviaturas y acrónimos.
- *Pre-etiquetación*: al realizar la segmentación se puede asignar ya la etiqueta de ciertas palabras no ambiguas, como ocurre con los números y las fechas.
- *Tratamiento de contracciones*: las contracciones se rompen en sus partes constituyentes, asignando una etiqueta a cada una de ellas. Para

ello se precisa consultar un diccionario con información acerca de la formación de contracciones.

- *Tratamiento de pronombres enclíticos*: se procede a separar la raíz de sus pronombres enclíticos, etiquetando cada uno de ellos por separado. Para ello se necesita consultar un diccionario que contenga el mayor número posible de formas verbales con posibilidad de llevar clíticos, una lista de las combinaciones válidas de pronombres enclíticos y una lista completa de todos estos pronombres, con sus etiquetas y lemas.

- *Identificación de expresiones*: se unen todas las palabras que componen una locución, expresión fija o frase hecha, consultando para ello un diccionario de expresiones. En esta etapa no se puede determinar la segmentación adecuada, por lo que debe simplemente indicarse la existencia de las distintas posibilidades.

- *Identificación de números*: se identifican los números escritos en letra.

- *Identificación de nombres propios*: se identifican los nombres propios, utilizando para ello un diccionario de nombres propios. Se deben considerar reglas para la creación de nombres compuestos, como *Ministerio de Educación, Cultura y Deportes*. Como los diccionarios de nombres propios nunca son completos, se pueden aprender nombres propios mediante un entrenemiento con textos etiquetados, almacenando aquellas secuencias de palabras que empiecen por mayúscula en partes del texto tal que la utilización de mayúsculas indica obligatoriamente que se trata de nombres propios.

3.2. Morfología flexiva y derivativa

Un *morfema* es una unidad gramatical mínima distintiva que ya no puede ser significativamente subdividida en términos gramaticales. Los morfemas antepuestos al primitivo se denominan *prefijos*, y los pospuestos, *sufijos*. Los *infijos* son elementos que aparecen intercalados en el interior de la estructura de un derivado.

Los *morfemas flexivos* son aquellos que se unen al morfema léxico de una palabra para indicar un cierto rasgo morfosintáctico como el género o el número en el caso de sustantivos y adjetivos y el tiempo, modo y persona en el caso de los verbos. Por ejemplo, la palabra *gatos* está formada por el morfema léxico *gat*, el morfema flexivo *o* que indica el género masculino de la palabra, y el morfema flexivo *s* que establece el plural como su número. Obervamos que

no todas las flexiones de una palabra aparecen en el diccionario. Esto se debe a que una de dichas formas, denominada *lema*, se elige como representativa de todas las formas de la palabra. En el caso de sustantivos y adjectivos, el lema es la forma masculina y singular; en el caso de los verbos, es el infinitivo. Los diccionarios comunes utilizan los lemas como entradas. Los distintos tipos de formas diferentes que pueden tener las palabras conforman el juego de etiquetas (*tag-set*), cuyo tamaño puede variar de unas pocas decenas a varias centenas, dependiendo del idioma y de la aplicación en la que se vaya a utilizar.

Los *morfemas derivativos* producen un cambio semántico respecto al lexema base, y frecuentemente también un cambio de categoría sintáctica. Tradicionalmente la formación de palabras ha sido dividida en *composición* y *derivación*. Hablamos de composición cuando combinamos lexemas independientes, y de derivación cuando alguno de los componentes (un morfema derivativo) no puede aparecer como tal lexema independiente, incluso en el caso de que se le pueda asignar un contenido semántico. Se trata en ambos casos de procedimientos morfológicos, unión de morfemas individuales o grupos de morfemas en unidades superiores para formar lexemas complejos. Los nuevos lexemas obtenidos pueden actuar, a su vez, como bases para la formación de nuevas palabras.

Los mecanismos básicos de derivación son la *prefijación*, la *sufijación apreciativa*, la *sufijación no apreciativa*, la *parasíntesis* y la *derivación regresiva*. La prefijación consiste en la adición de un prefijo a una forma base, la sufijación en la adición de un sufijo y la parasíntesis en la adición simultánea de un prefijo y un sufijo (por ejemplo, la derivación de *enrojecer* a partir de *rojo*, que sólo puede realizarse en un único paso al no existir las formas intermedias **enrojo* ni **rojecer*). En el caso de la sufijación, debemos distinguir entre sufijación apreciativa, que altera semánticamente el lexema base de un modo subjetivo emocional pero sin cambiar su categoría gramatical, y sufijación no apreciativa, que involucra un cambio fundamental más que marginal en el significado del lexema base, frecuentemente acompañado de un cambio de categoría sintáctica. Los sufijos apreciativos pueden a su vez subdividirse en diminutivos, que transmiten una idea de pequeñez o afectividad; aumentativos, que implican amplia dimensión, fealdad o grandiosidad; y peyorativos, que implican desagrado o ridiculez.

El repertorio de sufijos no apreciativos del español cuenta con cientos de morfemas derivativos, cuyo inventario no está fijado, ni tampoco sus restricciones, extensión o cambios de toda índole. Uno de los problemas con el que nos encontramos debido al elevado número de sufijos existentes, es el de su clasificación. Por una parte, en función de la categoría gramatical del derivado, podemos hablar de *nominalización* para dar sustantivos (el más común),

adjetivización para dar adjetivos y *verbalización* para obtener verbos. Por otra parte, conforme a la categoría gramatical de la base tenemos sufijos *denominales* (a partir de sustantivos), *deadjetivales* (a partir de adjetivos) y *deverbales* (a partir de verbos, los más frecuentes).

En lo referente a la derivación regresiva, ésta juega un importante papel en el español contemporáneo como mecanismo para la formación de sustantivos a partir de verbos. Su característica principal radica en que, en lugar de incrementar el tamaño del lexema base, provoca una reducción del mismo, al añadir tan solo una vocal a la raíz verbal. Por ejemplo, a partir de *deteriorar* obtenemos el derivado *deterioro*.

Un fenómeno importante a tener en cuenta es la existencia de *alomorfos*, variantes de un mismo morfema derivativo. Por ejemplo: *in*necesario, *im*prudente, *i*rreal. El alomorfo a utilizar en cada caso puede estar determinado por la fonología o venir impuesto por convención o por la etimología.

3.3. Modelización de grandes diccionarios

Un diccionario consiste en una base de datos que almacena la información léxica de las palabras. Sin embargo, la utilización de gestores de bases de datos no es adecuada para esta tarea, ya que no sólo interesa un acceso flexible a los datos, sino también un acceso muy rápido. Existen mecanismos mucho más eficientes para esta última tarea, como pueden ser los autómatas finitos.

Se puede pensar en almacenar en el diccionario tan sólo las raíces de las palabras, utilizando un conjunto de reglas de concatenación de morfemas flexivos para obtener las palabras completas. Aparte de la penalización en el rendimiento, este enfoque no es viable si deseamos incorporar información adicional relativa a las palabras, no a las raíces. Tal es el caso de determinados paradigmas de etiquetación estocástica o de análisis sintáctico estocástico, que necesitan asociar una probabilidad a cada una de las combinaciones posibles palabra-etiqueta.

En la figura 3.1 se muestra la representación más compacta que se puede diseñar para albergar toda la información léxica relativa a las palabras presentes en un diccionario. Es además una representación muy flexible en el sentido de que resulta particularmente sencillo incorporar nuevos tableros, si es que se necesita algún otro tipo de información adicional. Para mostrar su comportamiento, utilizaremos como ejemplo el diccionario del sistema Galena [76], que tiene $M = 291{,}604$ palabras diferentes, con $L = 354{,}007$ etiquetaciones posibles.

La función *Palabra_a_Índice* utiliza un autómata finito acíclico determi-

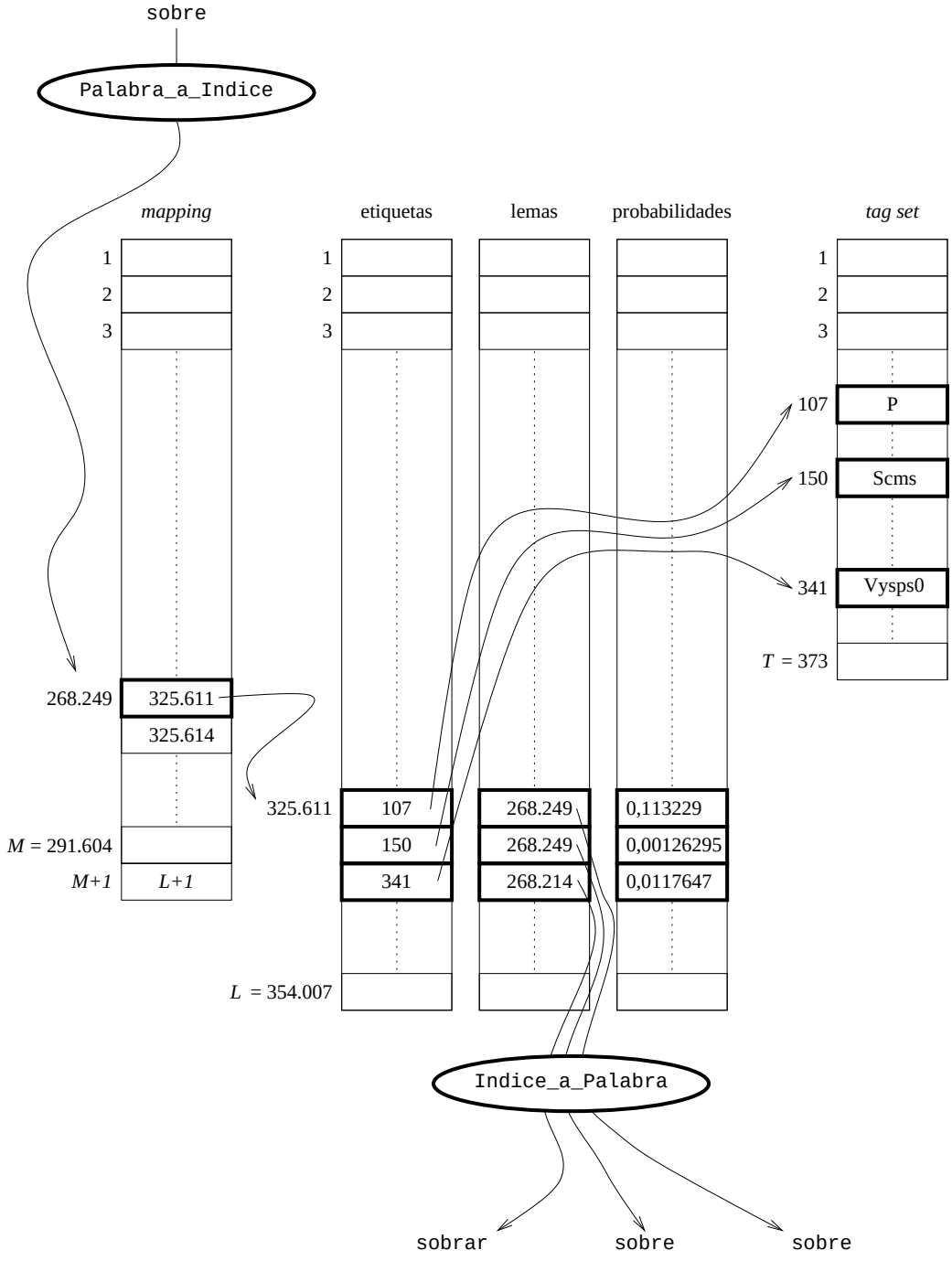

Figura 3.1: Modelización compacta de un diccionario

nista numerado para convertir una palabra en un número que representa la posición que ocupa esa palabra dentro del conjunto de todas las palabras diferentes ordenadas alfabéticamente. Por ejemplo, esta función transforma *sobre* en el número 268,249. Este número sirve para indexar un tablero de correspondencia de tamaño $M + 1$, que transforma la posición relativa de cada palabra en la posición absoluta dentro del lexicón original. En el caso de *sobre*, la posición relativa 268,249 se transforma en la posición absoluta 325,611. Este último número sirve para indexar los tableros de *etiquetas*, *lemas* y *probabilidades*. Todos estos tableros son de tamaño L.

El tablero de *etiquetas* almacena números. Una representación numérica de las etiquetas es más compacta que los nombres de las etiquetas en sí. Las etiquetas originales se pueden recuperar indexando con esos números el tablero del juego de etiquetas o *tag-set*. En el caso del diccionario del sistema Galena, el tamaño de dicho tablero es $T = 373$, el número de etiquetas distintas.

Dado que hemos partido de una ordenación alfabética, está garantizado que las etiquetas de una misma palabra aparecen contiguas. No obstante, necesitamos saber de alguna manera dónde terminan las etiquetas de una palabra dada. Para ello, es suficiente con restarle el valor de la posición absoluta de la palabra al valor de la siguiente casilla en el tablero de correspondencia. Por ejemplo, la palabra `sobre` tiene $325{,}614 - 325{,}611 = 3$ etiquetas. Esta operación es también válida para acceder correctamente a la información de los tableros de *lemas* y *probabilidades*. El tablero de *lemas* almacena también números. Un lema es una palabra que debe estar también presente en el diccionario. El número que la función *Palabra_a_Índice* obtendría para esa palabra es el número que se almacena aquí, siendo esta representación mucho más compacta que el lema en sí. El lema se puede recuperar aplicando la función *Índice_a_Palabra*, utilizando un autómata finito acíclico determinista numerado. El tablero de *probabilidades* almacena directamente las probabilidades, en cuyo caso no es posible realizar ninguna compactación.

3.4. Autómatas finitos acíclicos deterministas numerados

La manera más eficiente de implementar analizadores léxicos es mediante el uso de autómatas finitos. La aplicación más tradicional de esta idea la podemos encontrar en algunas de las fases de construcción de compiladores para los lenguajes de programación. El caso del procesamiento de los lenguajes naturales es cuantitativamente diferente, ya que surge la necesidad de

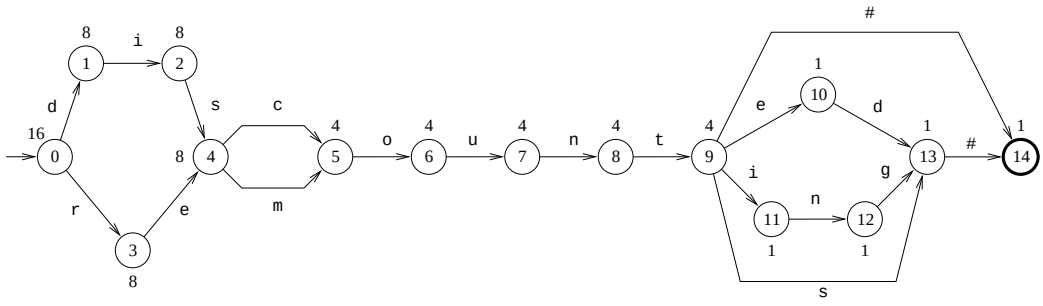

Figura 3.2: Autómata finito acíclico determinista mínimo numerado

representar diccionarios léxicos que muchas veces pueden llegar a involucrar a cientos de miles de palabras.

La estructura más sencilla para implementar un reconocedor de un conjunto finito de palabras dado es un *árbol de letras*. Esta estructura es en sí misma un autómata finito acíclico determinista. Sin embargo, los requerimientos de memoria de esta estructura de árbol pueden llegar a ser muy elevados cuando el diccionario es muy grande. Por ejemplo, el dicionario del sistema Galena necesitaría un árbol de más de un millón de nodos para reconocer las 291,604 palabras diferentes. Para solventar este problema aplicaremos un proceso de minimización de autómatas finitos, que resulta particularmente sencillo en el caso de autómatas finitos acíclicos deterministas [80].

Un autómata finito solamente es capaz de indicarnos si una palabra dada pertenece o no al diccionario, y esto no es suficiente para el esquema de modelización de diccionarios que hemos desarrollado anteriormente. Dicha modelización necesita un mecanismo que transfome cada palabra en una clave numérica unívoca, y viceversa. Esta transformación se puede llevar a cabo fácilmente si el autómata incorpora, para cada estado, un entero que indique el número de palabras que se pueden aceptar mediante el subautómata que comienza en ese estado [143]. Nos referiremos a este autómata como *autómata finito acíclico determinista numerado*. La asignación de los números de indexación a cada estado se puede realizar mediante un sencillo recorrido recursivo sobre el autómata, una vez que éste ha sido correctamente construido y minimizado. La figura 3.2 muestra el autómata finito acíclico determinista mínimo numerado para las diferentes formas de los verbos ingleses *discount*, *dismount*, *recount* y *remount*.

Mediante este tipo de autómatas podemos implementar eficientemente las funciones *Palabra_a_Índice* e *Índice_a_Palabra* del diccionario. La función *Palabra_a_Índice* parte con un índice igual a 1 y va transitando por el autómata desde el estado inicial utilizando cada una de las letras de la palabra a ana-

lizar. En cada uno de los estados por los que pasa el camino correspondiente a dicha palabra, el índice se va incrementando con el número de indexación del estado destino de aquellas transiciones que son lexicográficamente precedentes a la transición utilizada. Si después de procesar todos los caracteres de la palabra llegamos al estado final, entonces el índice contendrá la clave numérica de la palabra. En caso contrario, la palabra no pertenece al lexicón que se está manejando. La función *Índice_a_Palabra* parte del índice y realiza las operaciones análogas para deducir cuáles son las transiciones que dan lugar a ese índice, y a partir de esas transiciones obtiene las letras que forman la palabra que se está buscando.

3.5. Transductores de estado finito y morfología de dos niveles

La morfología de dos niveles considera las palabras como una correspondencia entre el *nivel léxico*, que representa la concatenación de los morfemas que constituyen una palabra, y el *nivel superficial*, que representa la forma escrita real de una palabra. En este paaradigma, las palabras se analizan mediante un conjunto de reglas que hacen corresponder secuencias de letras del nivel superficial a secuencias de morfemas y rasgos morfológicos del nivel léxico. Por ejemplo, la forma superficial *gatos* se convertiría en la forma léxica *gat +S +Masc +Pl* mediante la cual se indica que dicha palabra es un sustantivo masculino singular.

Para realizar la correspondencia entre los niveles superficial y léxico se utilizan *transductores de estado finito*, los cuales se encargan de traducir un conjunto de símbolos en otro, utilizando para ello un autómata finito. En consecuencia, podemos ver un transductor de estado finito como un autómata finito con dos cintas que reconoce o genera pares de cadenas. Mientras que un autómata finito define un lenguaje formal al determinar el conjunto de cadenas que lo forman, un transductor de estado finito define una relación entre dos conjuntos de cadenas. Los transductores de estado finito son cerrados bajo las operaciones de unión, inversión y composición, pero en general no lo son bajo las operaciones de diferencia, complementación e intersección, aunque existen subclases que son cerradas bajo estas tres operaciones.

Para la tarea del análisis morfológico, los transductores de estado finito son utilizado habitualmente en cascada: primero se utiliza un transductor que reconoce el morfema léxico de las palabras y lo convierte en su forma regular, al tiempo que indica su categoría gramatical; posteriormente, se aplican transductores especializados en el reconocimiento de morfemas específicos de

género, número, tiempo, persona, etc., que son transformados en rasgos morfológicos. La potencia de los transductores de estado finito viene determinada por el hecho de que la misma cascada, con las mismas secuencias de estados, puede ser utilizada tanto para obtener la forma léxica a partir de la forma superficial como para generar la forma superficial a partir de la forma léxica. Aunque gracias a la operación de composición podemos sustituir una cascada de transductores relativamente simples por un único transductor complejo, en la práctica el tamaño del transductor resultante puede llegar a ser tan grande que haga que su ejecución sea menos eficiente que la de la cascada.

Capítulo 4

Etiquetación

Una vez que han sido reconocidas las palabras que forman un texto y que ha sido determinado el conjunto de etiquetas válidas para cada una de ellas, es preciso determinar la etiqueta correcta. Este proceso de desambiguación se realiza en función de las etiquetas de las palabras situadas alrededor de la palabra estudiada. En este tema se estudian las técnicas más utilizadas en esta tarea.

4.1. Modelos de Markov ocultos

Consideremos un sistema que en cada instante de tiempo se encuentra en un determinado estado. Dicho estado pertenece a un conjunto finito de estados Q. Regularmente, transcurrido un espacio de tiempo discreto, el sistema cambia de estado de acuerdo con un conjunto de probabilidades de transición asociadas a cada uno de los estados del modelo. Los instantes de tiempo asociados a cada cambio de estado se denotan como $t = 1, 2, \ldots, T$, y el estado actual en el instante de tiempo t se denota como q_t. En general, una descripción probabilística completa del sistema requeriría la especificación del estado actual, así como de todos los estados precedentes. Sin embargo, las cadenas de Markov presentan dos características muy importantes:

1. La *propiedad del horizonte limitado*, que permite truncar la dependencia probabilística del estado actual y considerar, no todos los estados precedentes, sino únicamente un subconjunto finito de ellos. Una cadena de Markov de orden n es la que utiliza n estados previos para predecir el siguiente estado. Para el caso de las cadenas de Markov de tiempo discreto de primer orden tenemos que $P(q_t = j|q_{t-1} = i, q_{t-2} = k, \ldots) = P(q_t = j|q_{t-1} = i)$.

2. La *Propiedad del tiempo estacionario*, que nos permite considerar sólo aquellos procesos en los cuales $P(q_t = j|q_{t-1} = i)$ es independiente del tiempo, lo que a su vez nos lleva a definir una matriz de probabilidades de transición independientes del tiempo $A = \{a_{ij}\}$, donde $\forall i, j; \quad 1 \leq i, j \leq N; \quad a_{ij} = P(q_t = j|q_{t-1} = i) = P(j|i)$ y se cumplen las restricciones estocásticas estándar: $a_{ij} \geq 0$ para todo i y j, y $\sum_{j=1}^{N} a_{ij} = 1$ para todo i. Adicionalmente, es necesario especificar el vector $\pi = \{\pi_i\}$ que almacena la probabilidad $\pi_i \geq 0$ que tiene cada uno de los estados de ser el estado inicial: $\forall i; 1 \leq i \leq N; \pi_i = P(q_1 = i)$.

A un proceso estocástico que satisface estas características se le puede llamar un *modelo de Markov observable*, porque su salida es el conjunto de estados por los que pasa en cada instante de tiempo, y cada uno de estos estados se corresponde con un suceso observable. Esta modelización puede resultar demasiado restrictiva a la hora de ser aplicada a problemas reales. A continuación extenderemos el concepto de modelos de Markov de tal manera que sea posible incluir aquellos casos en los cuales la observación es una función probabilística del estado. El modelo resultante, denominado *modelo de Markov oculto* (HMM), es un modelo doblemente estocástico, ya que uno de los procesos no se puede observar directamente (está oculto), sino que se puede observar sólo a través de otro conjunto de procesos estocásticos, los cuales producen la secuencia de observaciones. Un HMM se caracteriza por la 5-tupla (Q, V, π, A, B), aunque como veremos Q y V son a veces obviados al venir determinados por los demás componentes, donde:

1. $Q = \{1, 2, \ldots, N\}$ es el conjunto de estados del modelo. Aunque los estados permanecen ocultos, para la mayoría de las aplicaciones prácticas se conocen a priori. Por ejemplo, para el caso de la etiquetación de palabras, cada etiqueta del juego de etiquetas utilizado sería un estado. Generalmente los estados están conectados de tal manera que cualquiera de ellos se puede alcanzar desde cualquier otro en un solo paso, aunque existen muchas otras posibilidades de interconexión. El estado actual en el instante de tiempo t se denota como q_t. El uso de instantes de tiempo es apropiado, por ejemplo, en la aplicación de los HMM al procesamiento de voz. No obstante, para el caso de la etiquetación de palabras, no hablaremos de los instantes de tiempo, sino de las posiciones de cada palabra dentro de la frase.

2. V es el conjunto de los distintos sucesos que se pueden observar en cada uno de los estados. Por tanto, cada uno de los símbolos individuales que un estado puede emitir se denota como $\{v_1, v_2, \ldots, v_M\}$. En el caso de

la etiquetación de palabras, M es el tamaño del diccionario y cada $v_k, 1 \leq k \leq M$, es una palabra distinta.

3. $\pi = \{\pi_i\}$, es la distribución de probabilidad del estado inicial, cumpliéndose que $\pi_i \geq 0$, $\forall i$; $1 \leq i \leq N$; $\pi_i = P(q_1 = i)$, y $\sum_{i=1}^{N} \pi_i = 1$.

4. $A = \{a_{ij}\}$ es la distribución de probabilidad de las transiciones entre estados, esto es, $\forall i, j, t$; $1 \leq i \leq N, 1 \leq i \leq N, 1 \leq t \leq T$; $a_{ij} = P(q_t = j | q_{t-1} = i) = P(j|i)$, cumpliéndose que $a_i, j \geq 0$ y que $\sum_{j=1}^{N} a_{ij} = 1$ para todo i.

5. $B = \{b_j(v_k)\}$ es la distribución de probabilidad de los sucesos observables, es decir, $\forall j, k, t$; $1 \leq j \leq N, 1 \leq k \leq M, 1 \leq t \leq T$; $b_j(v_k) = P(o_t = v_k | q_t = j) = P(v_k|j)$, cumpliéndose que $\sum_{k=1}^{M} b_j(v_k) = 1$ para todo j. Este conjunto de probabilidades se conoce también con el nombre de *conjunto de probabilidades de emisión*.

4.2. Ejecución eficiente de los modelos de Markov ocultos

Existen tres cuestiones fundamentales que debemos saber responder para poder utilizar los modelos de Markov ocultos en aplicaciones reales:

1. Dada una secuencia de observaciones $O = (o_1, o_2, \ldots, o_T)$ y un modelo $\mu = (\pi, A, B)$, calcular de manera eficiente $P(O|\mu)$, la probabilidad de dicha secuencia dado el modelo.

   Consideremos la variable $\alpha_t(i)$ definida como $\alpha_t(i) = P(o_1, o_2, \ldots, o_t, q_t = i|\mu)$, es decir, la probabilidad conjunta de obtener $o_1, o_2, \ldots, o_t$, la secuencia parcial de observaciones hasta el instante de tiempo t, y de estar en el estado i en ese instante de tiempo t, dado el modelo μ. Los valores de $\alpha_t(i)$, para los distintos estados y para los distintos instantes de tiempo, se pueden obtener iterativamente, y pueden ser utilizados para calcular $P(O|\mu)$ mediante los pasos del siguiente algoritmo.

   a) Inicialización: $\forall i$; $1 \leq i \leq N$; $\alpha_1(i) = \pi_i \, b_i(o_1)$

   b) Recurrencia: $\forall j, t$; $1 \leq j \leq N, 1 \leq t \leq T-1$; $\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) \, a_{ij}\right] b_j(o_{t+1})$

   c) Terminación: $P(O|\mu) = \sum_{i=1}^{N} \alpha_T(i)$

Los cálculos globales involucrados en este proceso requieren del orden de N^2T operaciones. Más concretamente, se trata de $N(N+1)(T-1)+N$ multiplicaciones y $N(N-1)(T-1)+N-1$ sumas, es decir, un total de $2(T-1)N^2+2N-1$ operaciones.

2. Dada una secuencia de observaciones $O=(o_1,o_2,\ldots,o_T)$ y un modelo $\mu=(\pi,A,B)$, determinar la secuencia de estados $S=(q_1,q_2,\ldots,q_T)$ óptima, la que mejor *explica* la secuencia de observaciones.

   Para encontrar la secuencia de estados $S=(q_1,q_2,\ldots,q_T)$ que maximiza $P(S|O,\mu)$, lo cual es equivalente a maximizar $P(S,O|\mu)$, utilizaremos el algoritmo de Viterbi. Para ello definiremos la variable $\delta_t(i) = \text{máx}_{q_1,q_2,\ldots,q_{t-1}} P(q_1,q_2,\ldots,q_{t-1},q_t=i,o_1,o_2,\ldots,o_t|\mu)$. Como vemos, $\delta_t(i)$ almacena la probabilidad del mejor camino que termina en el estado i, teniendo en cuenta las t primeras observaciones. Se demuestra fácilmente que $\delta_{t+1}(j)=[\text{máx}_{1\le i\le N}\,\delta_t(i)\,a_{ij}]\,b_j(o_{t+1})$. Una vez calculadas las $\delta_t(i)$ para todos los estados y para todos los instantes de tiempo, la secuencia de estados se construye realmente hacia atrás a través de una traza que recuerda el argumento que maximizó la ecuación para cada instante t y para cada estado j. Esta traza se almacena en las correspondientes variables $\psi_t(j)$. La descripción completa del algoritmo es como sigue.

   a) Inicialización: $\forall i;\ 1\le i\le N;\ \delta_1(i)=\pi_i\,b_i(o_1)$

   b) Recurrencia:
   $\forall j,t;\quad 1\le j\le N, 1\le t\le T-1;\quad \delta_{t+1}(j) = [\text{máx}_{1\le i\le N}\,\delta_t(i)\,a_{ij}]\,b_j(o_{t+1})$
   $\forall j,t;\ \ 1\le j\le N, 1\le t\le T-1;\ \ \psi_{t+1}(j)=\arg\text{máx}_{1\le i\le N}\,\delta_t(i)\,a_{ij}$

   c) Terminación: $q_T^*=\arg\text{máx}_{1\le i\le N}\,\delta_T(i)$

   d) Obtención de la secuencia de estados: $\forall t;\ T-1\ge t\ge 1;\ q_t^*=\psi_{t+1}(q_{t+1}^*)$

   El algoritmo de Viterbi es similar al cálculo hacia adelante de la probabilidad de una observación. Las únicas diferencias reseñables son que el sumatorio del paso de recurrencia se ha cambiado por una maximización y que se ha añadido el paso final para construir hacia atrás la secuencia de estados. En todo caso, la complejidad del algoritmo es del orden de N^2T operaciones.

   Es importante destacar que el algoritmo mostrado funciona para HMMs de orden 1. A este tipo de modelos se les denomina también modelos

de *bigramas* de etiquetas. Si se quiere trabajar con un HMM de orden n, sería necesario extender los cálculos del algoritmo para considerar las transiciones de estados desde n posiciones antes de la etiqueta actual. Para ello, se puede cambiar el espacio de estados del modelo por la n-ésima potencia del conjunto de etiquetas. De esta manera, es posible realizar una única implementación del algoritmo que sirva para cualquier orden n, con sólo introducir dicho orden como un parámetro. Por ejemplo, en un HMM de orden 2 el conjunto de estados del modelo se define como el producto cartesiano del conjunto de etiquetas, siendo válidas las transiciones entre un estado (t^i, t^l) y otro (t^m, t^k), donde todas las t^j son etiquetas, sólo cuando $l = m$. En cualquier caso, esto es lo que se conoce como modelo de *trigramas* de etiquetas, ampliamente referenciado y utilizado por la mayoría de los etiquetadores estocásticos.

Dado que el algoritmo de Viterbi no calcula una probabilidad exacta, sino que construye una secuencia de estados, podemos tomar logaritmos sobre los parámetros del modelo e implementarlo sólo con sumas, sin necesidad de ninguna multiplicación. De esta manera, no sólo se evita el problema de la rápida pérdida de precisión debida a las multiplicaciones de números muy próximos a 0, sino que además la velocidad de ejecución aumenta ya que las sumas se realizan más rápido que las multiplicaciones.

3. Dada una secuencia de observaciones $O = (o_1, o_2, \ldots, o_T)$, estimar los parámetros del modelo $\mu = (\pi, A, B)$ que maximiza $P(O|\mu)$, esto es, el modelo que mejor *explica* los datos observados.

   Cuando se dispone de un texto etiquetado, lo más adecuado es calcular directamente las frecuencias relativas a partir de los datos del corpus. Cuando no se dispone de un corpus etiquetado, la solución a esta cuestión es más complicada, ya que no se conoce ningún método analítico definitivo para encontrar un modelo $\mu = (\pi, A, B)$ que maximice $P(O|\mu)$. Sin embargo, podemos elegir un modelo que maximice localmente dicha probabilidad mediante un procedimiento iterativo tal como el algoritmo de Baum-Welch, que es un caso especial del algoritmo EM de maximización de la esperanza. La idea intuitiva es la siguiente. En un primer momento, no sabemos cómo es el modelo, pero podemos trabajar sobre la probabilidad de la secuencia de observaciones utilizando algún modelo inicial, quizás preseleccionado o simplemente elegido de forma aleatoria. A partir de ese cálculo, identificamos qué transiciones y qué símbolos de emisión son los más probables. Incrementando

la probabilidad de esas transiciones y de esos símbolos, construimos un modelo revisado, el cual obtendrá una probabilidad mayor que el modelo anterior para la secuencia de observaciones dada. Este proceso de maximización, denominado normalmente *proceso de entrenamiento*, se repite un cierto número de veces. Finalmente, el algoritmo se detiene cuando no consigue construir un modelo que mejore la probabilidad de la secuencia de observaciones dada.

4.3. Técnicas de suavizado

Un problema con el que nos debemos enfrentar a la hora de trabajar con modelos de Markov ocultos es el de la *dispersión de los datos*, en particular, la dificultad de calcular frecuencias de aparición para todas las posibles combinaciones de n etiquetas. Para evitar el problema de las transiciones nulas, se suelen utilizar *métodos de suavización*. Estos métodos permiten que a partir de muestras pequeñas se puedan estimar unas probabilidades más representativas del comportamiento real de la población que estamos estudiando.

Una forma de implementar la suavización es mediante técnicas de *interpolación lineal*. En general, el esquema de suavizado mediante interpolación lineal funciona como sigue. La distribución $f(x)$ observada en un conjunto de E posibles sucesos se modifica añadiendo un valor muy pequeño procedente de una distribución menos específica $q(x)$ a la cual damos un peso o confianza α. Entonces, la distribución de probabilidad que nos interesa se aproxima de la forma $\hat{p}(x) \approx \frac{f(x)+\alpha\, q(x)}{E+\alpha}$. Si utilizamos el parámetro tradicional de interpolación $\lambda = \frac{\alpha}{E+\alpha}$, entonces también se verifica la aproximación $\hat{p}(x) \approx (1-\lambda)\,\frac{f(x)}{E} + \lambda\, q(x)$. Es decir, si lo que queremos estimar son las probabilidades de transición entre estados de un modelo basado en bigramas, esa distribución menos específica ponderada con α al interpolar puede ser la distribución de probabilidad de los *unigramas*, esto es, la distribución de probabilidad de cada etiqueta individualmente. Por tanto, las probabilidades de los bigramas se pueden aproximar mediante $\hat{p}(t_i|t_{i-1}) = (1-\lambda)\, f(t_i|t_{i-1}) + \lambda\, f(t_i)$ y las de los trigramas mediante $\hat{p}(t_i|t_{i-2}\, t_{i-1}) = \lambda_3\, f(t_i|t_{i-2}\, t_{i-1}) + \lambda_2\, f(t_i|t_{i-1}) + \lambda_1\, f(t_i)$, donde los t_j representan etiquetas (estados del modelo) y todos los *pesos* λ_i deben ser no negativos y deben satisfacer la restricción $\lambda_1 + \lambda_2 + \lambda_3 = 1$. En un esquema de interpolación lineal, el cálculo de los parámetros λ_1, λ_2 y λ_3, conocidas las frecuencias de unigramas, bigramas y trigramas, y conocido N, el tamaño del corpus de entrenamiento, se realiza de mediante el siguiente algoritmo:

1. Inicializar $\lambda_1 = \lambda_2 = \lambda_3 = 0$.
2. Para cada trigrama $t_1\, t_2\, t_3$ con $C(t_1, t_2, t_3) > 0$, localizar el máximo de los tres valores siguientes y realizar la acción correspondiente:
   - $\frac{C(t_1,t_2,t_3)-1}{C(t_1,t_2)-1}$: incrementar λ_3 en $C(t_1, t_2, t_3)$ unidades.
   - $\frac{C(t_2,t_3)-1}{C(t_2)-1}$: incrementar λ_2 en $C(t_1, t_2, t_3)$ unidades.
   - $\frac{C(t_3)-1}{N-1}$: incrementar λ_1 en $C(t_1, t_2, t_3)$ unidades.
3. Normalizar los valores λ_1, λ_2 y λ_3.

Observamos que se proponen valores constantes para los parámetros de interpolación λ_1, λ_2 y λ_3. Es cierto que quizás no es una buena idea utilizar siempre los mismos valores, pero es cierto también que la consideración de un conjunto de valores λ_i, $i \in \{1, 2, 3\}$, para cada posible par de etiquetas eleva muchísimo el número de parámetros del modelo, lo cual no sólo no introduce ninguna mejora en relación con el fenómeno de los datos dispersos, sino que empeora el problema. No obstante, algunos experimentos sugieren que al menos sí sería conveniente una agrupación de los bigramas en un número moderado de clases y una posterior estimación de un conjunto de valores λ_i distinto para cada clase, aunque no queda claro cuál sería un buen criterio para la definición de esas clases.

Existen otros métodos de estimación que son capaces de asignar probabilidades distintas de cero a los sucesos no observados, permitiendo así hacer una importante distinción entre los *ceros no observados*, y los *ceros reales* que serían los correspondientes a los sucesos no posibles (por ejemplo, la aparición de tres preposiciones seguidas). Entre todas ellas, las más conocidas son la fórmula de Good-Turing y el método de *marcha atrás* (*back-off*). La idea intuitiva del método de estimación de Good-Turing es que, en lugar de estimar $P(X)$, intentamos estimar $P(X|C)$, donde C es una clasificación de sucesos definida *a priori* (en nuestro caso, los sucesos X que aparecen en el corpus de entrenamiento exactamente C veces). De esta manera, es más sencillo estimar $P(X)$, y en concreto se puede estimar $P(X|0)$, que es precisamente la probabilidad de los sucesos que no aparecen en el corpus. En resumen, el método de Good-Turing corrige la estimación de máxima verosimilitud en base al número de veces que algo ocurre en el corpus. La idea general del método de estimación de marcha atrás o *back-off* es confiar en las frecuencias si hay un número suficiente de apariciones, utilizar Good-Turing si no lo hay, pero el suceso está todavía presente, y utilizar una hipótesis de menor nivel (como el suavizado lineal) si no está presente en absoluto. La estimación de menor nivel esta basada en la suposición de que $P(X|Y) = P(X)$ cuando X e Y son independientes.

4.4. Tratamiento de palabras desconocidas

El método de manejo de palabras desconocidas que parece ofrecer mejores resultados para los lenguajes flexivos es el análisis de sufijos mediante aproximaciones basadas en inferencias bayesianas. En este método, las probabilidades de las etiquetas propuestas para las palabras no presentes en el diccionario se eligen en función de las terminaciones de dichas palabras. La distribución de probabilidad para un sufijo particular se genera a partir de todas las palabras del corpus de entrenamiento que comparten ese mismo sufijo. El término sufijo tal y como se utiliza aquí significa *secuencia final de caracteres de una palabra*, lo cual no coincide necesariamente con el significado lingüístico de sufijo. Por esta razón, el método requiere una definición previa de la longitud máxima de sufijos que se va a considerar.

Las probabilidades se suavizan mediante un procedimiento de abstracción sucesiva. Esto permite calcular $P(t|l_{n-m+1},\ldots,l_n)$, la probabilidad de una etiqueta t dadas las últimas m letras l_i de una palabra, a través de una secuencia de contextos más generales que omite un caracter del sufijo en cada iteración, de forma que la suavización se lleva a cabo en base a $P(t|l_{n-m+2},\ldots,l_n)$, $P(t|l_{n-m+3},\ldots,l_n)$, $\ldots$, $P(t)$. La fórmula de la recursión es $P(t|l_{n-i+1},\ldots,l_n) = \frac{\hat{p}(t|l_{n-i+1},\ldots,l_n)+\theta_i\, P(t|l_{n-i+2},\ldots,l_n)}{1+\theta_i}$ para $i = m,\ldots,1$, utilizando la estimación de máxima verosimilitud para un sufijo de longitud i calculada a partir de las frecuencias del corpus de entrenamiento como $\hat{p}(t|l_{n-i+1},\ldots,l_n) = \frac{C(t,l_{n-i+1},\ldots,l_n)}{C(l_{n-i+1},\ldots,l_n)}$, utilizando también unos determinados pesos de suavización θ_i, y utilizando como caso base $P(t) = \hat{p}(t)$. Por supuesto, para el modelo de Markov, necesitamos las probabilidades condicionadas inversas $P(l_{n-i+1},\ldots,l_n|t)$, las cuales se obtienen por la regla de Bayes.

La definición de este método no es rigurosa en todos y cada uno de sus aspectos, lo cual da lugar a diferentes interpretaciones a la hora de aplicarlo, ya que es necesario identificar un buen valor para m, la longitud máxima utilizada para los sufijos, realizar una elección adecuada de los pesos de suavización de sufijos θ_i, y elegir adecuadamente las palabras del corpus de entrenamiento que se utilizarán para la extracción de los sufijos.

En general, para las palabras desconocidas, $P(w|t) = 0$ estrictamente hablando. En caso contrario, la palabra no sería desconocida. Sin embargo, una de las principales habilidades que debe incluir un buen etiquetador es asignar una probabilidad $P(w|t)$ distinta de cero para las palabras no presentes en el diccionario. La única restricción que hay que respetar es que, para toda etiqueta t del conjunto de etiquetas, debería cumplirse $\sum_w P(w|t) = 1$ sobre todas las palabras w: las conocidas y las, en teoría, posiblemente desconocidas. Para ello, se podrían combinar las probabilidades de los sufijos con las

probabilidades de emisión de las palabras conocidas, reservando una parte de la masa de probabilidad de estas últimas. Es decir, si conocida la etiqueta t y para toda palabra w del diccionario, $S = \sum_w P(w|t)$, donde $S < 1$, entonces $1 - S$ sería la masa de probabilidad dedicada a las palabras desconocidas, conocida t. $P(\mathit{sufijo}|t)$ debería ser una distribución de probabilidad, de tal forma que $\sum_{\mathit{sufijo}} P(\mathit{sufijo}|t) = 1$, para toda etiqueta t del conjunto de etiquetas utilizado. Y los sufijos deberían estar correctamente definidos, de tal manera que la función $\mathit{sufijo}(w)$ fuera una aplicación. Es decir, el sufijo de w debería ser único para cada palabra w. De esta forma, $P(w|t)$ se define para las palabras desconocidas como $P(w|t) = (1 - S)\, P(\mathit{sufijo}(w)|t)$, lo cual no sólo es una definición formal, sino también sensible a las propiedades morfológicas de cada palabra. Las dificultades, radican en la identificación de los sufijos válidos para las palabras y en la estimación de un buen valor para S.

4.5. Aprendizaje de etiquetas basado en transformaciones y dirigido por el error

Algunas de las hipótesis de funcionamiento de los modelos de Markov no se adaptan del todo bien a las propiedades sintácticas de los lenguajes naturales. Debido a esto, inmediatamente surge la idea de utilizar modelos más sofisticados. Podríamos pensar, por ejemplo, en establecer condiciones que relacionen las nuevas etiquetas, no sólo con las etiquetas precedentes, sino también con las palabras precedentes. Podríamos pensar también en utilizar un contexto mayor que el de los etiquetadores basados en trigramas. Pero la mayoría de estas aproximaciones no tienen cabida dentro de los modelos de Markov, debido a la carga computacional que implican y a la gran cantidad de nuevos parámetros que necesitaríamos estimar. Eric Brill [42] presentó un sistema de etiquetación basado en reglas, el cual, a partir de un corpus de entrenamiento, infiere automáticamente las reglas de transformación, salvando así la principal limitación de este tipo de técnica que es precisamente el problema de cómo obtener dichas reglas. El etiquetador de Brill alcanza una corrección comparable a la de los etiquetadores estocásticos y, a diferencia de éstos, la información lingüística no se captura de manera indirecta a través de grandes tablas de probabilidades, sino que se codifica directamente bajo la forma de un pequeño conjunto de reglas no estocásticas muy simples, pero capaces de representar interdependencias muy complejas entre palabras y etiquetas.

El proceso de etiquetación consta de tres partes, que se infieren automáticamente a partir de un corpus de entrenamiento: un etiquetador léxico, un

etiquetador de palabras desconocidas, y un etiquetador contextual:

- El *etiquetador léxico* etiqueta inicialmente cada palabra con su etiqueta más probable, sin tener en cuenta el contexto en el que dicha palabra aparece. Dicha etiqueta más probable se estima previamente mediante el estudio del corpus de entrenamiento. A las palabras desconocidas se les asigna en un primer momento la etiqueta correspondiente a sustantivo propio si la primera letra es mayúscula, o la correspondiente a sustantivo común en otro caso. Posteriormente, el etiquetador de palabras desconocidas aplica en orden una serie de reglas de transformación léxicas. Si se dispone de un diccionario previamente construido, es posible utilizarlo junto con el que el etiquetador de Brill genera automáticamente.

- El *etiquetador de palabras desconocidas* opera justo después de que el etiquetador léxico haya etiquetado todas las palabras presentes en el diccionario, y justo antes de que se apliquen las reglas contextuales. Este módulo intenta *adivinar* una etiqueta para una palabra desconocida en función de su sufijo, de su prefijo, y de otras propiedades relevantes similares. Básicamente, cada transformación consta de dos partes: una descripción del contexto de aplicación, y una regla de reescritura que reemplaza una etiqueta por otra.

- El *etiquetador contextual* actúa justo después del etiquetador de palabras desconocidas, aplicando en orden una secuencia de reglas contextuales que, al igual que las léxicas, también han sido previamente inferidas de manera automática a partir del corpus de entrenamiento.

El proceso de aprendizaje de las reglas, tanto las léxicas en el caso del etiquetador de palabras desconocidas, como las contextuales en el caso del etiquetador contextual, selecciona el mejor conjunto de transformaciones y determina su orden de aplicación. El algoritmo consta de los pasos que se describen a continuación. En primer lugar, se toma una porción de texto no etiquetado, se pasa a través de la fase o fases de etiquetación anteriores, se compara la salida con el texto correctamente etiquetado, y se genera una lista de errores de etiquetación con sus correspondientes contadores. Entonces, para cada error, se determina qué instancia concreta de la plantilla genérica de reglas produce la mayor reducción de errores. Se aplica la regla, se calcula el nuevo conjunto de errores producidos, y se repite el proceso hasta que la reducción de errores cae por debajo de un umbral dado.

La técnica de etiquetación de Brill resulta considerablemente más lenta que las basadas en modelos probabilísticos. No sólo el proceso de entrenamiento consume muchísimo tiempo, sino que el proceso de etiquetación es

también inherentemente lento. La principal razón de esta ineficiencia computacional es la potencial interacción entre las reglas, de manera que el algoritmo puede producir cálculos innecesarios.

Capítulo 5

Análisis sintáctico: gramáticas independientes del contexto

La estructura de un lenguaje se describe mediante su sintaxis, de tal modo que una frase bien formada puede dividirse en constituyentes de acuerdo con unas determinadas restricciones o reglas sintácticas. Cada uno de los constituyentes puede a su vez dividirse en constituyentes más pequeños de acuerdo con las mismas reglas, hasta que finalmente se obtiene una descripción jerárquica completa de la estructura de la frase. La forma en que se determinan las reglas sintácticas es la que origina que existan diversos formalismos sintácticos. Siguiendo este razonamiento podemos considerar que la construcción de sistemas automáticos capaces de asimilar conocimiento sintáctico requiere ser tratado bajo una doble perspectiva: por un lado, se necesitan formalismos capaces de representar el conocimiento sintáctico; por otro lado, se necesitan algoritmos que determinen si una frase es correcta, y si es el caso, que obtengan su descripción jerárquica. En este tema nos ocuparemos de los algoritmos de análisis sintáctico que utilizan gramáticas independientes del contexto como formalismo para la descripción de la sintaxis.

5.1. Esquemas de análisis sintáctico

En el contexto computacional, la definición de algoritmos para el análisis sintáctico utiliza esencialmente una estrategia constructiva. Un analizador de este género calcula una serie de resultados intermedios que son utilizados para obtener, sucesivamente, nuevos resultados intermedios más avanzados, en el sentido de más cercanos a una posible solución. Si la oración es gramatical, este proceso debería concluir obteniendo un resultado final a partir

del cual podríamos recuperar sus árboles sintácticos. Aunque esta es fundamentalmente la forma en que proceden la mayor parte de los analizadores, la estrategia utilizada y la descripción final de los algoritmos oculta esta similitud. La utilización de *esquemas de análisis sintáctico* nos permitirá estudiar con uniformidad las diversas estrategias aplicadas en la definición de algoritmos para el análisis sintáctico [214]. La idea principal consiste en relacionar el problema de determinar si una oración es gramatical y el problema de demostrar fórmulas en sistemas deductivos. Además de presentar los fundamentos en los que se apoya este nuevo marco teórico, destacaremos sus ventajas a la hora de especificar y comparar, de forma homogénea, distintas estrategias de análisis.

Los esquemas de análisis sintáctico aplican el enfoque deductivo, que se caracteriza por relacionar el proceso de búsqueda de los árboles sintácticos con la demostración de teoremas en un sistema deductivo. La novedad del planteamiento radica en que las fórmulas del sistema deductivo, que denominaremos ítems, tendrán una interpretación específica relacionada con el problema del análisis sintáctico. La forma en que calculamos los ítems en un sistema deductivo de esta índole es similar a la forma en que son deducidas las fórmulas en cualquier otro sistema deductivo. Partiendo de un conjunto de ítems, aplicamos sucesivamente un conjunto de reglas deductivas (o pasos deductivos) de forma que sean calculados todos los ítems que pueden ser deducidos por el sistema. La presencia o no de algunos ítems determinará si una oración es gramatical. En el caso de que lo sea, un examen cuidadoso del conjunto de los ítems calculados nos dará la oportunidad de construir todos sus árboles sintácticos. Una consecuencia interesante de este hecho es la interpretación ambivalente del sistema deductivo como reconocedor o analizador. El propio bosque sintáctico puede ser, a su vez, interpretado mediante una gramática independiente del contexto que genera exclusivamente la oración de entrada [33].

Formalmente, un *sistema de análisis sintáctico* para una gramática independiente del contexto $\mathcal{G}$ y una cadena de entrada $a_1 \dots a_n$ es un triple $\mathbb{P} = \langle \mathcal{I}, \mathcal{H}, \mathcal{D} \rangle$, en el que

- $\mathcal{I}$ es un conjunto de *ítems* que representan resultados intermedios del proceso de análisis. Un conjunto $\mathcal{F} \subseteq \mathcal{I}$ de *ítems finales* indica mediante su presencia el reconocimiento de la cadena de entrada.

- $\mathcal{H}$ es un conjunto inicial de ítems denominados *hipótesis* que representan la cadena que va a ser analizada. No es necesario que $\mathcal{H} \subset \mathcal{I}$. Por el contrario, es habitual que ambos conjuntos sean disjuntos.

- $\mathcal{D} \subseteq \wp_{fin}(\mathcal{H} \cup \mathcal{I}) \times \mathcal{I}$ es un conjunto de reglas de inferencia, denominadas *pasos deductivos*, mediante las cuales se derivan nuevos ítems a partir de los ítems existentes. Estos pasos deductivos tienen la forma $\frac{\eta_1 \ldots \eta_k}{\xi}$ *cond* y su significado es el siguiente: si todos los antecedentes $\eta_i \in \mathcal{H} \cup \mathcal{I}$ de un paso deductivo ya existen y las condiciones *cond* son satisfechas, entonces el consecuente ξ deberá ser generado por el analizador sintáctico. Los pasos deductivos se suponen cuantificados universalmente para todas las posibles variables que aparezcan en los ítems a menos que se indique explícitamente lo contrario.

Un *esquema de análisis sintáctico* es un sistema de análisis $\mathbb{P}$ parametrizado para cualquier gramática y cadena de entrada. Un esquema de análisis puede ser generalizado a partir de otro mediante *refinamiento de los ítems* (rompiendo un ítem individual en varios), *refinamiento de los pasos deductivos* (descomponiendo un paso deductivo simple en una secuencia de pasos) y *extensión* (considerando una clase más amplia de gramáticas). Las operaciones de *filtrado* permiten disminuir el número de ítems y pasos deductivos de un esquema de análisis mediante la eliminación de pasos redundantes (*filtrado estático*) y la utilización de información contextual para determinar la validez de los ítems (*filtrado dinámico*). La operación de *contracción de pasos* permite que una secuencia de pasos deductivos sea reemplazada por un solo paso.

5.2. Análisis ascendente

El algoritmo de Cocke-Younger-Kasami (CYK) para análisis sintáctico de gramáticas independientes del contexto en forma normal de Chomsky [91] fue descubierto por J. Cocke, pero fue publicado independientemente por Younger [258] y Kasami [115], de ahí su nombre. Definiremos un esquema de análisis sintáctico para este algoritmo. Dada una gramática independientes del contexto $G = (V_T, V_N, S, P)$ en forma normal de Chomsky y una cadena de entrada $a_1, \ldots a_n$ con $n \geq 1$, el conjunto de ítems $\mathcal{I}_{\text{CYK}}$ se define de la siguiente forma:

$$\mathcal{I}_{\text{CYK}} = \{[A, i, j] \mid A \in V_N,\ 0 \leq i \leq j\}$$

Cada ítem de este conjunto representa que el símbolo no terminal A reconoce el segmento de la cadena de entrada comprendido entre las posiciones i y j, es decir, $A \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_j$. No se exige que $j \leq n$ con objeto de que la definición del conjunto de ítems no dependa del tamaño de la cadena de

entrada. De esta forma, la definición es general y aplicable a cualquier cadena de entrada. Si la oración es gramatical, y sólo en ese caso, deberá ser deducido el ítem $[S, 0, n]$.

El método CYK utiliza una estrategia ascendente pura: parte de los símbolos terminales incluidos en la entrada y culmina alcanzando la raíz de todos los árboles sintácticos de la oración, si existen. Tan sólo dos reglas deductivas son necesarias en el método que nos atañe. Las denominaremos reglas deductivas de lectura de los terminales $\mathcal{D}_{\text{CYK}}^{\text{Scan}}$ y compleción de no terminales $\mathcal{D}_{\text{CYK}}^{\text{Comp}}$.

La regla deductiva de lectura de los terminales se limita a calcular aquellos ítems relacionados con producciones $A \to a \in P$ cuyo símbolo terminal en el lado derecho concuerde con algún símbolo de la cadena de entrada a_i.

$$\mathcal{D}_{\text{CYK}}^{\text{Scan}} = \frac{[a, i-1, i]}{[A, i-1, i]} \quad A \to a \in P$$

Dada la producción $A \to BC \in P$, suponiendo que han sido deducidos dos ítems asociados a los no terminales B y C tales que reconocen segmentos de cadena colindantes, la regla deductiva de compleción los agrupará en un ítem. Este nuevo ítem estará asociado al no terminal A manifestando que este último reconoce el segmento total reconocido por B y C.

$$\mathcal{D}_{\text{CYK}}^{\text{Comp}} = \frac{[B, i, j]\,[C, j, k]}{[A, i, k]} \quad A \to BC \in P$$

La versión original del algoritmo hace uso de una matriz triangular bidimensional indexada por posiciones de la cadena de entrada para almacenar los resultados parciales obtenidos, de tal modo que el elemento A se encuentra en la posición $[i, j]$ de dicha matriz si y sólo si $A \overset{*}{\Rightarrow} a_i \dots a_{i+j-1}$. Una implementación alternativa almacena el elemento A en la posición $[i, j]$ si y sólo si $A \overset{*}{\Rightarrow} a_{i+1} \dots a_j$. En todo caso, es importante señalar que el esquema de análisis sintáctico es el mismo para cualquier implemenatción, lo que permite abstraerse de los detalles de implementación para centrarse en lo que constituye el comportamiento de la estrategia de análisis.

5.3. El algoritmo de Earley

El algoritmo CYK presenta una importante limitación, ya que sólo es aplicable a gramáticas en forma normal de Chomsky. Nuestro objetivo ahora es extender el esquema CYK a la clase general de CFG, obteniendo un esquema Earley ascendente en el que se construye un conjunto de ítems

$\left\{ \; [A \to \alpha \bullet \beta, i, j] \mid \alpha \stackrel{*}{\Rightarrow} a_{i+1} \ldots j \; \right\}$ que nos permitirán representar el reconocimiento parcial de producciones mediante la utilización de producciones con punto, al contrario de lo que ocurría en el caso del algoritmo CYK, el cual sólo permitía representar producciones binarias completas. En este sentido, un ítem CYK $[A, i, j]$ puede ser equivalentemente representado por $[A \to \alpha\bullet, i, j]$, donde $\alpha = BC$ o bien $\alpha = a_i$. El punto en las producciones indica que los elementos gramaticales situado a su izquierda han sido reconocidos. Las producciones son por tanto reconocidas de izquierda a derecha de tal modo que el punto está a la derecha del último elemento del lado derecho de una producción si y sólo si ésta ha sido completamente reconocida. El esquema de análisis $\mathbb{P}_{\text{buE}}$ correspondiente al algoritmo Earley ascendente es el siguiente:

$$\mathcal{I}_{\text{buE}} = \left\{ \; [A \to \alpha \bullet \beta, i, j] \mid A \to \alpha\beta \in P, \; 0 \leq i \leq j \; \right\}$$

$$\mathcal{H}_{\text{buE}} = \mathcal{H}_{\text{CYK}}$$

$$\mathcal{D}_{\text{buE}}^{\text{Init}} = \frac{}{[A \to \bullet\alpha, i, i]}$$

$$\mathcal{D}_{\text{buE}}^{\text{Scan}} = \frac{[A \to \alpha \bullet a\beta, i, j] \; [a, j, j+1]}{[A \to \alpha a \bullet \beta, i, j+1]}$$

$$\mathcal{D}_{\text{buE}}^{\text{Comp}} = \frac{[A \to \alpha \bullet B\beta, i, k] \; [B \to \gamma\bullet, k, j]}{[A \to \alpha B \bullet \beta, i, j]}$$

$$\mathcal{D}_{\text{buE}} = \mathcal{D}_{\text{buE}}^{\text{Init}} \cup \mathcal{D}_{\text{buE}}^{\text{Scan}} \cup \mathcal{D}_{\text{buE}}^{\text{Comp}}$$

$$\mathcal{F}_{\text{buE}} = \left\{ \; [S \to \alpha\bullet, 0, n] \; \right\}$$

Efectivamente, el sistema de análisis $\mathbb{P}_{\text{buE}}$ se deriva del sistema de análisis $\mathbb{P}_{\text{CYK}}$ mediante la aplicación de un refinamiento de los ítems y de un refinamiento de pasos deductivos, puesto que se ha descompuesto el paso $\mathcal{D}_{\text{CYK}}^{\text{Comp}}$ en dos pasos $\mathcal{D}_{\text{buE}}^{\text{Init}}$ y $\mathcal{D}_{\text{buE}}^{\text{Comp}}$. Finalmente se ha realizado una extensión del sistema de análisis al considerar, no ya gramáticas en forma normal de Chomsky, sino en forma arbitraria. La prueba formal puede encontrarse en [214].

Con respecto al comportamiento del algoritmo, podemos resumirlo indicando que el análisis comienza con la creación por parte de los pasos $\mathcal{D}_{\text{bu}}^{\text{Init}}$ de

los ítems $[A \to \bullet\alpha, i, i]$ para toda regla de la gramática y para toda posición en la cadena de entrada. A continuación se aplican los pasos $\mathcal{D}_{\text{buE}}^{\text{Scan}}$ y $\mathcal{D}_{\text{buE}}^{\text{Comp}}$ con el fin de ir desplazando el punto de las producciones hacia la derecha. Un paso deductivo $\mathcal{D}_{\text{buE}}^{\text{Scan}}$ es aplicable a ítems de la forma $[A \to \alpha \bullet a\beta, i, j]$ cuando $a_{j+1} = a$, obteniéndose un nuevo ítem $[A \to \alpha a \bullet \beta, i, j+1]$. Es decir, se ha reconocido el símbolo terminal que estaba justo a la derecha del punto. Un paso deductivo $\mathcal{D}_{\text{buE}}^{\text{Comp}}$ se aplica cuando un ítem tiene una regla con el punto en el extremo derecho. Dado un ítem $[B \to \gamma\bullet, k, j]$ se buscan todos los posibles $[A \to \alpha \bullet B\beta, i, k]$ y se generan nuevos ítems $[A \to \alpha B \bullet \beta, i, j]$ que representan que la subcadena $a_{k+1} \ldots a_j$ puede ser reducida a B y por consiguiente, como la subcadena $a_{i+1} \ldots a_k$ se reduce a α, la subcadena $a_{i+1} \ldots a_j$ se reduce a αB. El proceso termina cuando no se pueden combinar más ítems. En tal caso, si se ha generado algún ítem final la cadena de entrada pertenece al lenguaje generado por la gramática.

Se puede derivar un esquema de análisis correspondiente al algoritmo de Earley [70] a partir del esquema Earley ascendente mediante la aplicación de un *filtrado dinámico* a este último, de modo que los ítems de la forma $[A \to \bullet\alpha, i, i]$ no sean generados por los pasos Init para todas las posibles producciones y posiciones en la cadena de entrada, sino que sean generados o no dependiendo de la validez de otros ítems mediante un paso deductivo predictivo, encargándose el paso Init únicamente de la creación de los ítems correspondientes a las producciones del axioma de la gramática. Los ítems válidos son por tanto de la forma $\left\{ \; [A \to \alpha \bullet \beta, i, j] \mid \alpha \overset{*}{\Rightarrow} a_{i+1} \ldots j, \;\; S \overset{*}{\Rightarrow} a_1 \ldots a_i A\delta \; \right\}$. El esquema de análisis $\mathbb{P}_{\text{Earley}}$ correspondiente al algoritmo de Earley es el que se muestra a continuación:

$$\mathcal{I}_{\text{Earley}} = \mathcal{I}_{\text{buE}}$$

$$\mathcal{H}_{\text{Earley}} = \mathcal{H}_{\text{CYK}}$$

$$\mathcal{D}_{\text{Earley}}^{\text{Init}} = \frac{}{[S \to \bullet\alpha, 0, 0]}$$

$$\mathcal{D}_{\text{Earley}}^{\text{Scan}} = \mathcal{D}_{\text{buE}}^{\text{Scan}}$$

$$\mathcal{D}_{\text{Earley}}^{\text{Pred}} = \frac{[A \to \alpha \bullet B\beta, i, j]}{[B \to \bullet\gamma, j, j]}$$

$$\mathcal{D}_{\text{Earley}}^{\text{Comp}} = \mathcal{D}_{\text{buE}}^{\text{Comp}}$$

$$\mathcal{D}_{\text{Earley}} = \mathcal{D}^{\text{Init}}_{\text{Earley}} \cup \mathcal{D}^{\text{Scan}}_{\text{Earley}} \cup \mathcal{D}^{\text{Pred}}_{\text{Earley}} \cup \mathcal{D}^{\text{Comp}}_{\text{Earley}}$$

$$\mathcal{F}_{\text{Earley}} = \mathcal{F}_{\text{buE}}$$

Con respecto al comportamiento del algoritmo, lo resumiremos indicando que el proceso de análisis comienza con la generación, por parte del conjunto de pasos $\mathcal{D}^{\text{Init}}_{\text{Earley}}$, de los ítems de la forma $[S \to \bullet\alpha, 0, 0]$, donde $S \to \alpha \in P$ es una regla del axioma de la gramática. Estos ítems indican que se está tratando de reconocer el axioma de la gramática desde el inicio de la cadena de entrada. Los pasos deductivos $\mathcal{D}^{\text{Scan}}_{\text{Earley}}$ y $\mathcal{D}^{\text{Comp}}_{\text{Earley}}$ se comportan como en el caso del algoritmo de Earley ascendente, mientras que el conjunto de pasos $\mathcal{D}^{\text{Pred}}_{\text{Earley}}$ se encarga de realizar la fase descendente o predictiva del algoritmo, ya que a partir de ítems de la forma $[A \to \alpha \bullet B\beta, i, j]$ genera los ítems $[B \to \bullet\gamma, j, j]$, esto es, se predicen todas las reglas que potencialmente pueden ser útiles en el reconocimiento de la cadena de entrada.

5.4. Autómatas a pila y programación dinámica

La utilización de autómatas para realizar el análisis sintáctico es interesante porque permite separar el problema de la definición de un algoritmo de análisis sintáctico del problema de la ejecución del mismo. En el caso particular de las gramáticas independientes del contexto es posible optar por este diseño modular puesto que es posible definir un algoritmo de análisis sintáctico como un conjunto de transiciones de un autómata a pila, probablemente no determinista, el cual puede ser interpretado eficientemente mediante las técnicas de tabulación disponibles. Este enfoque presenta ventajas evidentes, entre la cuales cabe citar la simplificación de las pruebas de corrección de los algoritmos y la disponibilidad de un entorno homogéneo para la comparación experimental de su comportamiento.

La definición tradicional de autómata a pila [91] incluye un control finito que se puede eliminar, puesto que el estado de una configuración dada puede almacenarse en el elemento situado en la cima de la pila [27]. Como consecuencia obtenemos una definición alternativa equivalente [135, 62], que juzgamos más simple y homogénea, según la cual un autómata a pila es una tupla $(V_T, V_S, \Theta, \$_0, \$_f)$, donde V_T es un conjunto finito de símbolos terminales, V_S es un conjunto finito de símbolos de pila, $\$_0 \in V_S$ es el símbolo inicial

de la pila, $\$_f \in V_S$ es el símbolo final de la pila y Θ es un conjunto de transiciones, cada una de las cuales pertenece a uno de los tres tipos siguientes, donde $C, F, G \in V_S$, $\xi \in V_S^*$ y $a \in V_T \cup \{\epsilon\}$:

- *SWAP*: Transiciones de la forma $C \overset{a}{\longmapsto} F$ que reemplazan el elemento C de la cima de la pila por el elemento F mientras se lee a de la cadena de entrada.

- *PUSH*: Transiciones de la forma $C \overset{a}{\longmapsto} CF$ que apilan un nuevo elemento F en la pila mientras se lee a de la cadena de entrada.

- *POP*: Transiciones de la forma $CF \overset{a}{\longmapsto} G$ que eliminan los dos elementos C y F de la cima de la pila y los sustituyen por G mientras se lee a de la cadena de entrada.

Según la nueva definición, la *configuración* de un autómata a pila en un momento dado viene determinada por el par (ξ, w), donde ξ es el contenido de la pila y w es la parte de la cadena de entrada que resta por leer. Una configuración (ξ, aw) deriva una configuración (ξ', w), hecho que denotamos mediante $(\xi, aw) \vdash (\xi', w)$, si y sólo si existe una transición que aplicada a ξ devuelve ξ' y consume a de la cadena de entrada.

La independencia del contexto de las transiciones de los autómatas a pila permite tabular la ejecución de los mismos. Presentamos a continuación la técnica propuesta por Lang [133, 135]. En una gramática independiente del contexto, si $B \overset{*}{\Rightarrow} \delta$ entonces $\alpha B \beta \overset{*}{\Rightarrow} \alpha \delta \beta$ para todo $\alpha, \beta \in (V_N \cup V_T)^*$. Esta misma independencia del contexto se traslada a los autómatas a pila, de tal modo que si $(B, a_{i+1} \dots a_j \dots a_n) \overset{*}{\vdash} (BC, a_{j+1} \dots a_n)$ entonces también se cumple que $(\xi B, a_{i+1} \dots a_j \dots a_n) \overset{*}{\vdash} (\xi BC, a_{j+1} \dots a_n)$ para todo $\xi \in V_S^*$. Denominaremos *derivaciones independientes del contexto* a este tipo de derivaciones. Observamos que este tipo de derivaciones presenta gran semejanza con las transiciones PUSH.

Para representar una derivación independiente del contexto sólo precisamos almacenar los símbolos de pila B y C más las posiciones de la cadena de entrada i y j, puesto que la derivación es independiente de ξ. La técnica de tabulación se basa precisamente en reemplazar la manipulación de pilas por la manipulación de ítems de la forma $[B, i, C, j]$ que representan de forma compacta el conjunto de derivaciones independientes del contexto que comparten los elementos de la cima de la pila. Los ítems se combinan mediante *reglas de combinación* de la forma $\frac{ants}{cons}\ trans$, donde *cons* es el ítem consecuente que se obtiene al aplicar la transición *trans* sobre los ítems antecedentes *ants*. La manipulación de configuraciones mediante la aplicación

de transiciones es equivalente a la manipulación de ítems mediante las reglas de combinación correspondientes a cada transición [133, 135]:

- *Transiciones SWAP*: Dada la derivación independiente del contexto $(\xi' B, a_{i+1} \dots a_n) \overset{*}{\vdash} (\xi' BC, a_{j+1} \dots a_n)$ que da lugar a la configuración $(\xi' BC, a_{j+1} \dots a_n)$, tras la aplicación de la transición $C \overset{a}{\longmapsto} F$ obtendremos la derivación independiente del contexto $(\xi' B, a_{i+1} \dots a_n) \overset{*}{\vdash} (\xi' BF, a_{k+1} \dots a_n)$. La regla de combinación de ítems es

$$\frac{[B, i, C, j]}{[B, i, F, k]} \; C \overset{a}{\longmapsto} F$$

- *Transiciones PUSH*: la aplicación de una transición $C \overset{a}{\longmapsto} CF$ produce la derivación $(\xi C, a_{j+1} \dots a_n) \vdash (\xi CF, a_{k+1} \dots a_n)$, donde $k = j$ si $a = \epsilon$ y $k = j + 1$ si $a = a_{j+1}$. Esta derivación es por sí misma una derivación independiente del contexto, por lo que la correspondiente regla de combinación de ítems es

$$\frac{[B, i, C, j]}{[C, j, F, k]} \; C \overset{a}{\longmapsto} CF$$

- *Transiciones POP*: La configuración $(\xi CF, a_{k+1} \dots a_n)$ refleja una derivación independiente del contexto $(\xi C, a_{j+1} \dots a_n) \overset{*}{\vdash} (\xi CF, a_{k+1} \dots a_n)$, pero además necesitamos la derivación independiente del contexto $(\xi' B, a_{i+1} \dots a_n) \overset{*}{\vdash} (\xi' BC, a_{j+1} \dots a_n)$ que colocó C en la cima de la pila para obtener la derivación independiente del contexto $(\xi' B, a_{i+1} \dots a_n) \overset{*}{\vdash} (\xi' BG, a_{l+1} \dots a_n)$ resultante de la aplicación de la transición $CF \overset{a}{\longmapsto} G$, donde $l = k$ si $a = \epsilon$ y $l = k + 1$ si $a = a_{k+1}$. La regla de combinación de ítems es

$$\frac{\begin{array}{c}[C, j, F, k] \\ [B, i, C, j]\end{array}}{[B, i, G, l]} \; CF \overset{a}{\longmapsto} G$$

5.5. Análisis sintáctico LR generalizado

Los algoritmos de análisis LR, que constituyen una de las más poderosas familias de algoritmos de análisis sintáctico para gramáticas independientes del contexto, constan de dos fases diferenciadas: una de *compilación*, en la que la gramática es compilada en una máquina de estado finito denominada

autómata LR y dos tablas de acciones e ir-a [5, 4]; y una segunda de *ejecución* en la que un autómata a pila, utilizando el autómata LR como memoria de estado finito, determina la pertenencia o no de las sentencias de entrada al lenguaje generado por la gramática. Las gramáticas que pueden ser analizadas determinísticamente mediante analizadores LR con k símbolos de preanálisis constituyen la clase de las *gramáticas LR*, muy útiles a la hora de describir lenguajes de programación pero que resultan insuficientes cuando se trata de analizar el lenguaje natural. Para ampliar el ámbito de los analizadores LR al caso de lenguajes ambiguos, debemos admitir la posibilidad de que las entradas de la tabla de acciones indiquen más de una acción a realizar en un momento dado. En este caso, nos encontramos con los algoritmos LR no deterministas, también conocidos como *LR generalizado*.

A partir del algoritmo de Earley es posible derivar la familia de algoritmos LR para el análisis de gramáticas independientes del contexto sin restricciones [8, 9]. Como resultado, se obtienen algoritmos LR generalizados con complejidad $\mathcal{O}(n^3)$, donde n es la longitud de la cadena de entrada, en el peor caso pero con un mejor comportamiento en casos prácticos. La obtención de esta complejidad se debe a la utilización de programación dinámica para representar la evolución no determinista de la pila, en lugar de representaciones de pilas estructuradas en grafo. Para ello utilizaremos una *binarización implícita de producciones* [101, 135, 138]. Básicamente, este proceso consiste en transformar una reducción de una producción con m elementos en su parte derecha en m reducciones de producciones que poseen a lo sumo 2 elementos en su parte derecha. Siguiendo este enfoque, la reducción de la producción $A_{r,0} \to A_{r,1} \dots A_{r,n_r}$ se transformaría en la reducción de las siguientes $n_r + 1$ producciones: $A_{r,0} \to \nabla_{r,0}$, $\nabla_{r,0} \to A_{r,1}\nabla_{r,1}$, $\dots$, $\nabla_{r,n_r-1} \to A_{r,n_r}\nabla_{r,n_r}$, y $\nabla_{r,n_r} \to \varepsilon$ donde los símbolos nabla son *frescos*, esto es, diferentes de cualquier otro símbolo de la gramática. Un aspecto importante a considerar es que no es necesario tratar explícitamente la existencia de esas $n_r + 1$ nuevas producciones. Bien al contrario, el algoritmo trabaja siempre sobre las producciones originales. Esto se consigue introduciendo los símbolos ∇ directamente en el interior de las producciones. En efecto, la producción $A_{r,0} \to A_{r,1} \dots A_{r,n_r}$ pasa a ser vista por el algoritmo como constituida por los elementos $A_{r,0} \to \nabla_{r,0}\ A_{r,1}\ \nabla_{r,1}\ \dots A_{r,n_r}\ \nabla_{r,n_r}$ de tal modo que los símbolos ∇ sirven de indicadores para señalar la parte de la producción que ha sido reducida, o equivalentemente, cuáles de las reducciones binarias han sido aplicadas. Por ejemplo, un ítem conteniendo ∇_{r,n_r} indicará que se ha reducido la producción $\nabla_{r,n_r} \to \varepsilon$, mientras que un ítem conteniendo ∇_{r,n_r-1} indicará que ya han sido reducidas las producciones $\nabla_{r,n_r} \to \varepsilon$ y $\nabla_{r,n_r-1} \to A_{r,n_r}\ \nabla_{r,n_r}$. La presencia de $\nabla_{r,0}$ indicará que toda la parte derecha de la producción original ha sido reducida y que ya sólo queda por

generar el símbolo $A_{r,0}$, correspondiente al lado izquierdo de la producción[1].

La interpretación en programación dinámica del autómata a pila correspondiente al algoritmo LR generalizado para una gramática independiente del contexto $\mathcal{G}$ y una cadena de entrada $a_1 \dots a_n$ queda definido por el el sistema de análisis $\mathbb{P}_{\text{LR}}$ que se muestra a continuación:

$$\mathcal{I}_{\text{LR}} = \left\{ \; [A, st, i, j] \cup [\nabla_{r,s}, st, i, j] \mid A \in V_N \cup V_T, \;\; st \in \mathcal{S}, \;\; 0 \leq i \leq j \; \right\}$$

$$\mathcal{H}_{\text{LR}} = \mathcal{H}_{\text{Earley}}$$

$$\mathcal{D}_{\text{LR}}^{\text{Init}} = \frac{}{[-, st_0, 0, 0]}$$

$$\mathcal{D}_{\text{LR}}^{\text{InitShift}} = \frac{[A, st, i, j]}{[A_{r,1}, st', j, j+1]} \quad \begin{array}{l} \exists [a, j, j+1] \in \mathcal{H}_{\text{LR}^{\text{S1}}} \\ A_{r,1} = a \\ \text{shift}_{st'} \in \text{acción}(st, a), \;\; A \in V \end{array}$$

$$\mathcal{D}_{\text{LR}}^{\text{Shift}} = \frac{[A_{r,s}, st, i, j]}{[A_{r,s+1}, st', i, j+1]} \quad \begin{array}{l} \exists [a, j, j+1] \in \mathcal{H}_{\text{LR}^{\text{S1}}} \\ A_{r,s+1} = a \\ \text{shift}_{st'} \in \text{acción}(st, a) \end{array}$$

$$\mathcal{D}_{\text{LR}}^{\text{Sel}} = \frac{[A, st, i, j]}{[\nabla_{r,n_r}, st, i, j+1]} \quad \begin{array}{l} \exists [a, j, j+1] \in \mathcal{H}_{\text{LR}^{\text{S1}}} \\ \text{reduce}_r \in \text{acción}(st, a), \;\; A \in V \end{array}$$

$$\mathcal{D}_{\text{LR}}^{\text{Red}} = \frac{[\nabla_{r,s}, st, i, k] \quad [A_{r,s}, st, k, j]}{[\nabla_{r,s-1}, st', i, j]} \quad st' \in \text{revela}(st)$$

$$\mathcal{D}_{\text{LR}}^{\text{Head}} = \frac{[\nabla_{r,0}, st, i, j]}{[A_{r,0}, st', i, j]} \quad st' \in \text{ir\_a}(st, A_{r,0})$$

$$\mathcal{D}_{\text{LR}} = \mathcal{D}_{\text{LR}}^{\text{Init}} \cup \mathcal{D}_{\text{LR}}^{\text{InitShift}} \cup \mathcal{D}_{\text{LR}}^{\text{Shift}} \cup \mathcal{D}_{\text{LR}}^{\text{Sel}} \cup \mathcal{D}_{\text{LR}}^{\text{Red}} \cup \mathcal{D}_{\text{LR}}^{\text{Head}}$$

$$\mathcal{F}_{\text{LR}^{\text{S1}}} = \left\{ \; [S, st_f, 0, n] \; \right\}$$

[1] Es interesante señalar la similitud entre $\nabla_{r,i}$ y la producción con punto $A_{r,0} \rightarrow \alpha \bullet \beta$, donde $\alpha = A_{r,1} \dots A_{r,i}$ y $\beta = A_{r,i+1} \dots a_{r,n_r}$.

A la hora de realizar un desplazamiento, diferenciamos entre el desplazamiento del primer elemento del lado derecho de una producción (*InitShift*) y el de cualquier otro elemento del lado derecho (*Shift*). Por su parte, las operaciones de reducción se realizan en tres pasos: *Sel* para indicar que una producción a sido seleccionada para reducción, *Red* para realizar la reducción implícita de una producción binaria y *Head* para indicar la finalización de la reducción y el reconocimiento del lado izquierdo de la producción. El conjunto de estados del autómata LR se donota por $\mathcal{S}$, $st_0 \in \mathcal{S}$ es el estado inicial y Φ es el axioma de la gramática aumentada.

5.6. Representación compartida de los árboles de análisis sintáctico

Los algoritmos de análisis sintáctico tal y como han sido descritos son realmente reconocedores y no analizadores, ya que no crean una representación de los árboles de análisis. Sin embargo, es fácil modificarlos para que creen un bosque de análisis sintáctico que satisfaga las dos características siguientes: que almacene de una forma compacta todos los árboles de análisis y que permita recuperar cada uno de ellos en tiempo lineal con respecto al tamaño del bosque. Billot y Lang [33] definen el bosque compartido para una gramática independiente del contexto $\mathcal{G} = (V_T, V_N, P, S)$ y una cadena de entrada $a_1 \dots a_n$ como una gramática independiente del contexto en la cual los no terminales son de la forma $\langle A, i, j\rangle$, donde $A \in V_N$ y $i, j \in 0..n$, y las producciones son de la forma $\langle A_0, j_0, j_m\rangle \to w_0\ \langle A_1, j_0, j_1\rangle\ w_1\ \langle A_2, j_1, j_2\rangle\ \dots\ w_{m-1}\ \langle A_m, j_{m-1}, j_m\rangle\ w_m$, donde $A_0 \to w_0 A_1 w_1 A_2 \dots w_{m-1} A_m w_m \in P$ y $w_i \in V_T^*$. Estas producciones indican que A_0 reconoce la parte $a_{j_0+1} \dots a_{j_m}$ de la cadena de entrada mediante la aplicación de la producción $A_0 \to w_0 A_1 w_1 A_2 \dots w_{m-1} A_m w_m$ tal que cada A_i reconoce la parte $a_{j_{i-1}+1} \dots a_{j_i}$ de la cadena. Como ejemplo, en el algoritmo de Earley se puede obtener un bosque de análisis de este tipo añadiendo la siguiente salida a los pasos deductivos:

$$\mathcal{D}^{\text{Pred}}_{\text{Earley}} = \frac{[A \to \alpha \bullet B\beta, i, j]}{[B \to \bullet\gamma, j, j]}$$

$$\text{salida: } \langle B \to \bullet\gamma, j, j\rangle \to \epsilon$$

$$\mathcal{D}^{\text{Scan}}_{\text{buE}} = \frac{[A \to \alpha \bullet a\beta, i, j]\ [a, j, j+1]}{[A \to \alpha a \bullet \beta, i, j+1]}$$

$$\text{salida: } \langle A \to \alpha a \bullet \beta, i, j+1\rangle \to \langle A \to \alpha \bullet a\beta, i, j\rangle\ \ a$$

$$\mathcal{D}_{\text{buE}}^{\text{Comp}} = \frac{[A \to \alpha \bullet B\beta, i, k]\ [B \to \gamma\bullet, k, j]}{[A \to \alpha B \bullet \beta, i, j]}$$

salida: $\langle B \to \gamma\bullet, k, j\rangle \to \langle A \to \alpha \bullet B\beta, i, k\rangle\ \langle B \to \gamma\bullet, k, j\rangle$

Una vez obtenida la gramática que representa el bosque de análisis, se puede reducir con el fin de eliminar los no terminales inútiles. La extracción de los árboles individuales se puede realizar aplicando el algoritmo de la figura 5.1, en el que los ítems, limitados por paréntesis, contienen producciones del bosque de análisis. La función *insert* almacena uno de estos ítems, mientras que la función *delete* lo borra. La función *show_tree* k es la encargada de mostrar los ítems almacenados que tienen un tercer componente de valor menor o igual que k.

5.7. Análisis sintáctico probabilístico

Los algoritmos de análisis sintáctico utilizados en procesamiento del lenguaje natural son capaces de obtener todos los árboles que una gramática dada asigna a una frase. Sin embargo, en ciertas aplicaciones puede ser más conveniente obtener un sólo árbol, el más plausible de acuerdo con el uso habitual del lenguaje. Para ello se pueden aumentar las gramáticas independientes del contexto con probabilidades, para establecer las gramáticas independientes del contexto probabilísticas, en las que se asigna una probabilidad a cada producción, denotada habitualmente como $P(A \to \alpha)$ o como $P(A \to \alpha|A)$. Formalmente, la probabilidad asociada a una producción $A \to \alpha$ es la probabilidad condicional de una determinada expansión dado el no terminal del lado izquierdo de la producción. En consecuencia, debe cumplirse que $\forall A; A \in V_N; \sum_{A\to\alpha} P(A \to \alpha) = 1$.

Mediante la utilización de este tipo de gramáticas podemos establecer la probabilidad asociada a cada árbol de análisis. La probabilidad de un árbol T para una cadena de entrada w se define como el producto de las probabilidades asociadas a las producciones involucradas en la construcción de dicho árbol: $P(T, w) = \prod_{A\to\alpha\in T} P(A \to \alpha)$. La probabilidad $P(T, w)$ es tanto la probabilidad conjunta del árbol y la cadena como la probabilidad $P(T)$ del árbol. Esto se sigue del hecho de que $P(T, w) = P(T)P(w|T)$, pero puesto que el árbol T debe incluir todas las palabras de w se cumple que $P(w|T) = 1$, por lo que $P(T, w) = P(T)P(w|T) = P(T)$. Esta probabilidad se puede calcular al extraer cada árbol de análisis individual del bosque de análisis.

También podemos asignar una probabilidad a cada cadena w del lenguaje definido por la gramática. Para cadenas no ambiguas, su probabilidad es igual

a la del único árbol que se puede obtener: $P(w) = P(T, w) = P(T)$. En el caso de una cadena ambigua, su probabilidad es la suma de las probabilidades de cada uno de los árboles de análisis: $P(w) = \sum_{T \in \tau(w)} P(T, w) = \sum_{T \in \tau(w)} P(T)$, donde $\tau(w)$ es el conjunto de árboles de análisis para w. Esta probabilidad se puede calcular al tiempo que se generan las producciones del bosque de análisis.

Una gramática probabilística G es consistente si la suma de las probabilidades de cada una de las cadena del lenguaje definido por la gramática es 1: $\sum_{w \in L(G)} P(w) = 1$. La presencia de ciertas producciones recursivas puede causar que una gramática sea inconsistente, al generar una pérdida de la masa de probabilidad por la presencia de derivaciones que nunca terminan. En la práctica, no es necesario verificar la consistencia, considerándose que una gramática es adecuada para el tratamiento probabilístico si no genera árboles con ciclos, todos los terminales son productivos y $\forall A; A \in V_N; \sum_{A \to \alpha} P(A \to \alpha) = 1$.

La forma más sencilla de obtener una gramática probabilística es a partir de un corpus de oraciones previamente analizadas. Tales corpus reciben el nombre de *bancos de árboles* o *treebanks*. La probabilidad de expansión de un no terminal se puede calcular contando las apariciones de las producciones: $P(A \to \alpha) = \frac{C(A \to \alpha)}{\sum_{\gamma} C(A \to \gamma)} = \frac{C(A \to \alpha)}{C(A)}$.

Si no se dispone de un banco de árboles, deberá utilizarse un procedimiento iterativo para estimar las probabilidades. Se partirá de una gramática inicial con unas estimaciones iniciales, probablemente realizadas aleatoriamente. Mediremos la confianza en nuestra gramática como la probabilidad de cada análisis obtenido al analizar un conjunto de oraciones de entrenamiento. Sumaremos la probabilidad de cada producción utilizada para estimar la frecuencia de aplicación de cada una de ellas y utilizaremos estas probabilidades para refinar las probabilidades asignadas a las producciones, con el fin de incrementar la probabilidad de las oraciones de entrenamiento dada la gramática.

```
function reduce (A → γ•, i, k)
    if A = S
    then show_tree k
    else foreach (B → α • Aβ, i', i) do
            build (B → α A • β, i', k)

function closure k B
    foreach B → γ do
       insert (B → •γ, k, k)
       if γ = a_{k+1}γ'
       then build (B → a_{k+1} • γ', k, k+
1)
       else if γ = Cγ'
            then closure k C
            else reduce (B → •, k, k)
       delete (B → •γ, k, k)

function build (A → α • β, i, k)
    insert (A → α • β, i, k)
    if β = a_{k+1} β'
    then build (A → α a_{k+1} • β', i, k+1)
    else if β = Bβ'
         then closure k B
         else reduce (A → α•, i, k)
    delete (A → α • β, i, k)

function main
    build (S → •α, 0, 0)
```

Figura 5.1: Algoritmo de extracción de los árboles del bosque de análisis

Capítulo 6

Análisis sintáctico: gramáticas suavemente dependientes del contexto

La potencia expresiva de las gramáticas independientes del contexto es habitualmente suficiente para describir la sintaxis de los lenguajes de programación. Sin embargo, las lenguas naturales presentan construcciones que no pueden ser descritas mediante gramáticas independientes del contexto. Surge entonces la pregunta de si existen otro género de formalismos más apropiados. Un obstáculo importante en esta búsqueda es que no se sabe a ciencia cierta qué lugar ocuparían las lenguas naturales en la jerarquía de lenguajes definida por Chomsky, aunque se cree que estarían situadas entre los lenguajes independientes del contexto y los lenguajes dependientes del contexto, posiblemente más cerca de los primeros que de los segundos. Esta suposición se basa en el hecho de que la mayoría de las construcciones sintácticas sólo dependen *suavemente* del contexto en el cual son aplicadas. En este tema se presentan las gramáticas de adjunción de árboles, que constituyen el formalismos suavemente dependiente del contexto más popular, y se estudian diferentes técnicas de análisis sintáctico para ellas.

6.1. Gramáticas de adjunción de árboles

Puesto que la estructura sintáctica asociada a las frases es una estructura jerárquica representada normalmente como un árbol o, en el caso de frases ambiguas, como un conjunto de árboles, parece natural pensar que un formalismo que manipule árboles y que presente cierta dependencia suave del contexto debe facilitar la descripción de la sintaxis de las lenguas natu-

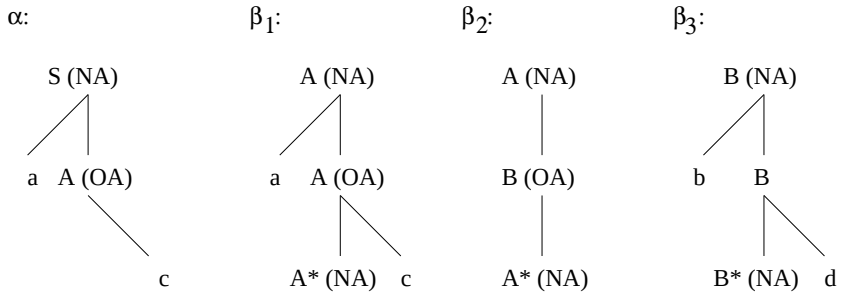

Figura 6.1: Gramática de adjunción de árboles que genera el lenguaje $a^n b^m c^n d^m$

rales. En esta dirección las gramáticas de adjunción de árboles (TAG), un formalismo suavemente dependiente del contexto que manipula árboles, se ha mostrado adecuado para la descripción de los fenómenos sintácticos que aparecen en el lenguaje natural [103]. El conjunto de los lenguajes generados por las gramáticas de adjunción de árboles constituye la clase de los lenguajes de adjunción de árboles.

Formalmente, una gramática de adjunción de árboles es una quíntupla $(V_T, V_N, \boldsymbol{I}, \boldsymbol{A}, S)$ donde V_T es un conjunto finito de símbolos *terminales*, V_N es un conjunto finito de símbolos *no terminales*, con $V_T \cap V_N = \emptyset$, $\boldsymbol{I}$ es un conjunto finito de *árboles iniciales*, $\boldsymbol{A}$ es un conjunto finito de *árboles auxiliares*, y $S \in V_N$ es el axioma de la gramática. Los árboles en $\boldsymbol{I} \cup \boldsymbol{A}$ se denominan *árboles elementales* de la gramática. Los árboles iniciales se caracterizan porque su raíz está etiquetada por el axioma de la gramática, sus nodos interiores están etiquetados por no terminales y sus nodos hoja están etiquetados por terminales o por la palabra vacía, denotada por ϵ. Los árboles auxiliares son como los árboles iniciales con la excepción de que la etiqueta de su raíz puede ser un no terminal arbitrario y porque uno de sus nodos hoja, que recibe el nombre de *pie* está etiquetado por el mismo no terminal que etiqueta su raíz. El camino desde el nodo raíz hasta el nodo pie recibe el nombre de *espina*. En la figura 6.1 se muestra la gramática de adjunción de árboles $(V_T, V_N, \boldsymbol{I}, \boldsymbol{A}, S)$ donde $V_T = \{a, b, c, d\}$, $V_N = \{S, A, B\}$, $\boldsymbol{I} = \{\alpha\}$, $\boldsymbol{A} = \{\beta_1, \beta_2, \beta_3\}$ y S es el axioma de la gramática. Dicha gramática genera el lenguaje $a^n b^m c^n d^m$ para $n, m \geq 1$.

Los árboles elementales se pueden combinar entre sí para crear *árboles derivados*, los cuales a su vez se pueden combinar con otros árboles para formar árboles derivados más grandes. La operación mediante la cual se combinan los árboles se denomina *adjunción*. Mediante una adjunción se construye un nuevo árbol a partir de un árbol auxiliar β y de otro árbol γ, que puede ser un árbol inicial, auxiliar o derivado de adjunciones realizadas previamente.

En su forma más simple, una adjunción puede tener lugar si la etiqueta de un nodo del árbol γ (denominado *nodo de adjunción*) coincide con la etiqueta del nodo raíz de un árbol auxiliar β. En tal caso, el árbol derivado resultante se construye como sigue: el subárbol de γ dominado por el nodo de adjunción se escinde de γ, aunque se deja una copia del nodo de adjunción en γ; el árbol auxiliar β se pega a la copia del nodo de adjunción de tal forma que la raíz del árbol auxiliar se identifica con dicha copia; finalmente, el subárbol escindido de γ se pega al nodo pie del árbol auxiliar β de tal modo que la raíz del subárbol escindido (el nodo de adjunción) se identifica con el nodo pie de β.

La aplicabilidad de una adjunción tal y como ha sido descrita sólo depende de las etiquetas de los nodos. Sin embargo, por conveniencia se puede especificar para cada nodo un conjunto de restricciones que permite indicar con más precisión los árboles auxiliares que pueden ser adjuntados. Las restricciones asociadas a un nodo, que se denominan *restricciones de adjunción*, pueden ser de los tipos siguientes: restricciones de *adjunción selectiva* (SA) que especifican el subconjunto de árboles auxiliares que pueden participar en una operación de adjunción; restricciones de *adjunción nula* (NA) que impiden la realización de adjunciones; y restricciones de *adjunción obligatoria* (OA) que especifica un subconjunto de árboles auxiliares, uno de los cuales ha de ser utilizado obligatoriamente en una operación de adjunción.

Desde una perspectiva lingüística, las características más destacadas de las gramáticas de adjunción de árboles son las tres siguientes:

- *Dominio de localidad extendido*: los lenguajes descritos por las gramáticas de adjunción de árboles son fruto de la composición de un conjunto de árboles elementales. Los árboles son estructuras con un dominio de localidad suficientemente amplio como para establecer las dependencias locales entre elementos lingüísticos dentro del mismo.

- *Factorización de la recursión*: los árboles elementales se dividen en iniciales y auxiliares, representando los primeros las estructuras básicas frente a los segundos que representan las estructuras recurrentes. Para obtener nuevos árboles se hace uso de una operación denominada adjunción que consiste en la inclusión de un árbol auxiliar dentro de otro árbol. Dicha operación permite expandir las dependencias locales representadas en los árboles elementales, de tal modo que son capturados de forma directa fenómenos como las dependencias de larga distancia.

- *Lexicalización*: Las gramáticas de adjunción de árboles están muy en consonancia con las teorías lingüísticas modernas en las que se otorga

un papel preponderante a la información léxica. El carácter lexicalista de estas gramáticas procede de que los árboles elementales deben contener al menos un elemento léxico. Esta restricción conduce a una cierta simbiosis entre los conceptos de gramática y de diccionario en el sentido de que cada palabra está asociada con un conjunto de árboles.

6.2. Análisis sintáctico de gramáticas de adjunción de árboles

En la descripción de los algoritmos de análisis para TAG, tenemos que representar el reconocimiento parcial de los árboles elementales. En las gramáticas independientes del contexto se utilizan habitualmente reglas punteadas para este fin. En el caso de TAG, consideraremos cada árbol elemental γ como constituido por un conjunto de reglas independientes del contexto $\mathcal{P}(\gamma)$ e indicaremos la posición del punto en el árbol indicando la posición en la regla correspondiente. Los elementos de las reglas serán los nodos del árbol, excepto en el caso de los elementos del lado derecho de las reglas cuyas etiquetas pertenecen a $V_T \cup \varepsilon$, puesto que al no poder tener descendientes ni ser susceptibles de ser nodos de adjunción consideraremos directamente su etiqueta a la hora de escribir las reglas.

Con el fin de simplificar la descripción de los algoritmos seguiremos el enfoque de [162] de considerar la regla adicional $\top \rightarrow \mathbf{R}^{\alpha}$ para cada árbol inicial α y las dos reglas adicionales siguientes para cada árbol auxiliar β: $\top \rightarrow \mathbf{R}^{\beta}$ y $\mathbf{F}^{\beta} \rightarrow \bot$, donde $\mathbf{R}^{\beta}$ y $\mathbf{F}^{\beta}$ se refieren a los nodos raíz y pie de β, respectivamente. Con el fin de no modificar la capacidad generativa de las gramáticas, los nuevos nodos $\top$ y $\bot$ no pueden ser nodos de adjunción.

Como punto de partida tomaremos una extensión del algoritmo CYK. Asumiremos que todo nodo de un árbol elemental de la gramática tiene a lo sumo dos descendientes. Definimos el siguiente conjunto de ítems de la forma $[A^{\gamma}, i, j \mid p, q \mid adj]$, tal que $A^{\gamma} \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_p \; \mathbf{F}^{\gamma} \; a_{q+1} \ldots a_j \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_j$ si y sólo si $(p, q) \neq (-, -)$ mientras que $A^{\gamma} \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_j$ si y sólo si $(p, q) = (-, -)$. El elemento *adj* toma su valor del conjunto $\{\text{true}, \text{false}\}$ e indica, si es *true* que el ítem es el resultado de una operación de adjunción ya totalmente realizada, si su valor es *false* que el ítem no es el resultado de una adjunción. Con ello podremos limitar a una sola la cantidad de adjunciones que se pueden realizar sobre cada nodo de un árbol elemental. Describimos aquí los pasos más relevantes, aquellos correspondientes al inicio de una adjunción en el nodo pie y a la terminación de la misma en el nodo raíz de un árbol auxiliar:

$$\mathcal{D}_{\text{CYK}}^{\text{Foot}} = \frac{}{[\mathbf{F}^{\gamma}, i, j \mid i, j \mid \text{false}]}$$

$$\mathcal{D}_{\text{CYK}}^{\text{Adj}} = \frac{\begin{array}{c}[\mathbf{R}^{\beta}, i', j' \mid i, j \mid adj], \\ [N^{\gamma}, i, j \mid p, q \mid \text{false}]\end{array}}{[N^{\gamma}, i', j' \mid p, q \mid \text{true}]} \quad \beta \in \text{adj}(N^{\gamma})$$

Para superar la limitación de las gramáticas binarias definiremos un algoritmo de tipo Earley ascendente. Como primer paso precisamos introducir puntos en las producciones que representan los árboles elementales, por lo que los ítems serán de la forma $[A^{\gamma} \rightarrow \delta \bullet \nu, i, j \mid p, q]$, tal que $\delta \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_p \, \mathbf{F}^{\gamma} \, a_{q+1} \ldots a_j \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_j$ si y sólo si $(p, q) \neq (-, -)$ mientras que $\delta \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_j$ si y sólo si $(p, q) = (-, -)$. Al incluirse reglas con punto en la forma de los nuevos ítems ya no es necesario el elemento que indicaba si se había realizado una adjunción en el nodo situado en el lado izquierdo de la producción contenida en dicho ítem. Los pasos deductivos más importantes son:

$$\mathcal{D}_{\text{buE}}^{\text{Foot}} = \frac{}{[\mathbf{F}^{\beta} \rightarrow \bot\bullet, i, j \mid i, j]}$$

$$\mathcal{D}_{\text{buE}}^{\text{AdjComp}} = \frac{\begin{array}{c}[\top \rightarrow \mathbf{R}^{\beta}\bullet, k, j \mid l, m], \\ [M^{\gamma} \rightarrow \upsilon\bullet, l, m \mid p', q'], \\ [N^{\gamma} \rightarrow \delta \bullet M^{\gamma}\nu, i, k \mid p, q],\end{array}}{[N^{\gamma} \rightarrow \delta M^{\gamma} \bullet \nu, i, j \mid p \cup p', q \cup q']} \quad \beta \in \text{adj}(M^{\gamma})$$

Incorporando predicción descendente podemos obtener un algoritmo de tipo Earley. Para ello debemos aplicar dos filtros dinámicos: el paso deductivo $\mathcal{D}_{\text{buE}}^{\text{Init}}$ sólo contendrá producciones cuyo lado izquierdo corresponda con la raíz de un árbol inicial mientras que un nuevo conjunto de pasos predictivos controlará la generación de nuevos ítems tratando de limitarla únicamente a aquellos que puedan resultar útiles en el proceso de análisis. Los pasos deductivos más importantes son:

$$\mathcal{D}_{\text{E}}^{\text{AdjPred}} = \frac{[N^{\gamma} \rightarrow \delta \bullet M^{\gamma}\nu, i, j \mid p, q]}{[\top \rightarrow \bullet\mathbf{R}^{\beta}, j, j \mid -, -]} \quad \beta \in \text{adj}(M^{\gamma})$$

$$\mathcal{D}_{\text{E}}^{\text{FootPred}} = \frac{[\mathbf{F}^{\beta} \rightarrow \bullet\bot, k, k \mid -, -]}{[M^{\gamma} \rightarrow \bullet\delta, k, k \mid -, -]} \quad \beta \in \text{adj}(M^{\gamma})$$

$$\mathcal{D}_{\mathrm{E}}^{\mathrm{FootComp}} = \frac{[M^\gamma \to \delta\bullet, k, l \mid p, q],\ [\mathbf{F}^\beta \to \bullet\bot, k, k \mid -, -]}{[\mathbf{F}^\beta \to \bot\bullet, k, l \mid k, l]} \quad \beta \in \mathrm{adj}(M^\gamma)$$

$$\mathcal{D}_{\mathrm{E}}^{\mathrm{AdjComp}} = \mathcal{D}_{\mathrm{buE}}^{\mathrm{AdjComp}} = \frac{\begin{array}{c}[\top \to \mathbf{R}^\beta\bullet, j, m \mid k, l],\\ [M^\gamma \to \upsilon\bullet, k, l \mid p, q],\\ [N^\gamma \to \delta \bullet M^\gamma\nu, i, j \mid p', q']\end{array}}{[N^\gamma \to \delta M^\gamma \bullet \nu, i, m \mid p \cup p', q \cup q']} \quad \beta \in \mathrm{adj}(M^\gamma)$$

Los analizadores sintácticos que satisfacen la *propiedad del prefijo válido* garantizan que, en tanto que leen la cadena de entrada de izquierda a derecha, las subcadenas leídas son prefijos válidos del lenguaje definido por la gramática. Más formalmente, un analizador sintáctico satisface la propiedad del prefijo válido si al leer la subcadena $a_1 \ldots a_k$ de la cadena de entrada $a_1 \ldots a_k a_{k+1} \ldots a_n$ garantiza que hay una cadena $b_1 \ldots b_m$, donde b_i no tiene porque formar parte de la cadena de entrada, tal que $a_1 \ldots a_k b_1 \ldots b_m$ es una cadena válida del lenguaje. Para obtener un esquema de análisis que se corresponda con un algoritmo del tipo Earley que posea la propiedad del prefijo válido es necesario refinar los ítems incluyendo un nuevo elemento que indique la posición del extremo izquierdo de la frontera del árbol al que se refieren los nodos de cada ítem: $[h, A^\gamma \to \delta \bullet \nu, i, j \mid p, q]$, tal que $\mathbf{R}^\gamma \overset{*}{\Rightarrow} a_{h+1} \ldots a_i \delta\nu\upsilon$ y además $\delta \overset{*}{\Rightarrow} a_i \ldots a_p$ $\mathbf{F}^\gamma$ $a_{q+1} \ldots a_j \overset{*}{\Rightarrow} a_i \ldots a_j$ si y sólo si $(p, q) \neq (-, -)$ mientras que $\delta \overset{*}{\Rightarrow} a_i \ldots a_j$ si y sólo si $(p, q) = (-, -)$. Los pasos deductivos más importantes son:

$$\mathcal{D}_{\mathrm{Nederhof}}^{\mathrm{AdjPred}} = \frac{[h, N^\gamma \to \delta \bullet M^\gamma\nu, i, j \mid p, q]}{[j, \top \to \bullet\mathbf{R}^\beta, j, j \mid -, -]} \quad \beta \in \mathrm{adj}(M^\gamma)$$

$$\mathcal{D}_{\mathrm{Nederhof}}^{\mathrm{FootPred}} = \frac{[j, \mathbf{F}^\beta \to \bullet\bot, k, k \mid -, -],\ [h, N^\gamma \to \delta \bullet M^\gamma\nu, i, j \mid p, q]}{[h, M^\gamma \to \bullet\delta, k, k \mid -, -]} \quad \beta \in \mathrm{adj}(M^\gamma)$$

$$\mathcal{D}_{\mathrm{Nederhof}}^{\mathrm{FootComp}} = \frac{\begin{array}{l}[h, M^\gamma \to \upsilon\bullet, k, l \mid p, q],\\ [j, \mathbf{F}^\beta \to \bullet\bot, k, k \mid -, -],\\ [h, N^\gamma \to \delta \bullet M^\gamma\nu, i, j \mid p', q']\end{array}}{[j, \mathbf{F}^\beta \to \bot\bullet, k, l \mid k, l]} \quad \begin{array}{l}\beta \in \mathrm{adj}(M^\gamma),\\ p \cup p' \text{ y } q \cup q' \text{ está definido}\end{array}$$

$$\mathcal{D}_{\mathrm{Nederhof}}^{\mathrm{AdjComp}^0} = \frac{\begin{array}{c}[j, \top \to \mathbf{R}^\beta\bullet, j, m \mid k, l],\\ [h, M^\gamma \to \upsilon\bullet, k, l \mid p, q],\end{array}}{[[M^\gamma \to \upsilon\bullet, j, m \mid p, q]]} \quad \beta \in \mathrm{adj}(M^\gamma)$$

$$\mathcal{D}^{\text{AdjComp}^1}_{\text{Nederhof}} = \frac{\begin{array}{c}[[M^\gamma \to \upsilon\bullet, j, m \mid p, q]],\\ [h, \mathbf{F}^\gamma \to \bot\bullet, p, q \mid p, q],\\ [h, N^\gamma \to \delta \bullet M^\gamma\nu, i, j \mid -, -]\end{array}}{[h, N^\gamma \to \delta M^\gamma \bullet \nu, i, m \mid p, q]}$$

$$\mathcal{D}^{\text{AdjComp}^2}_{\text{Nederhof}} = \frac{\begin{array}{c}[[M^\gamma \to \upsilon\bullet, j, m \mid -, -]],\\ [h, N^\gamma \to \delta \bullet M^\gamma\nu, i, j \mid p', q']\end{array}}{[h, N^\gamma \to \delta M^\gamma \bullet \nu, i, m \mid p', q']}$$

Todos estos algoritmos trabajan en una complejidad $\mathcal{O}(n^6)$ con respecto a la longitud n de la cadena de entrada. La complejidad espacial varía entre $\mathcal{O}(n^4)$ y $\mathcal{O}(n^5)$.

6.3. Autómatas para el análisis de TAG

Los autómatas lineales de índices (LIA) [17] son una extensión de los autómatas a pila en la cual cada símbolo de pila tiene asociado una pila con información acerca de las adjunciones pendientes de realizar. Formalmente, un autómata de este tipo es una tupla $(V_T, V_S, \$_0, \$_f, V_I, \mathcal{T})$, donde: V_T es un conjunto finito de símbolos terminales, V_S es un conjunto finito de símbolos de pila, $\$_0 \in V_S$ es el símbolo inicial de la pila, $\$_f \in V_S$ es el símbolo final de pila, V_I es un conjunto finito de índices que se almacenan en la pila de adjunciones, y $\mathcal{T}$ es un conjunto finito de transiciones, que pueden ser de alguno de los siguientes tipos:

- $C[\,] \overset{a}{\longmapsto} F[\,]$
- $C[\circ\circ\gamma] \overset{a}{\longmapsto} F[\circ\circ\gamma']$
- $C[\circ\circ] \overset{a}{\longmapsto} F[\circ\circ]\ G[\,]$
- $C[\circ\circ\gamma] \overset{a}{\longmapsto} F[\,]\ G[\circ\circ\gamma']$
- $F[\circ\circ\gamma]\ G[\,] \overset{a}{\longmapsto} C[\circ\circ\gamma']$
- $F[\,]\ G[\circ\circ\gamma] \overset{a}{\longmapsto} C[\circ\circ\gamma']$

donde $C, F, G \in V_S$, $a \in V_T \cup \{\epsilon\}$, $\gamma, \gamma' \in V_I \cup \{\epsilon\}$ y en cada transición, bien $\gamma = \epsilon$, bien $\gamma' = \epsilon$ o bien $\gamma = \gamma' = \epsilon$. Dada una transición, decimos que el elemento de la parte derecha que comparte la pila de adjunciones

con el elemento de la parte izquierda es su *hijo dependiente*. La relación de *descendiente dependiente* es el cierre reflexivo y transitivo de la relación de hijo dependiente.

Una *configuración* de un autómata lineal de índices es un par (Υ, w), donde $\Upsilon \in (V_S[V_I^*])^*$ y $w \in V_T^*$, tal que Υ representa el contenido de la pila del autómata y w representa la parte de la cadena de entrada que resta por leer. Una configuración (Υ_1, aw) deriva una configuración (Υ_2, w), denotado mediante $(\Upsilon_1, aw) \vdash (\Upsilon_2, w)$ si y sólo si existe una transición que transforma la pila Υ_1 en la pila Υ_2 leyendo $a \in V_T \cup \{\epsilon\}$ de la cadena de entrada.

En el caso que nos ocupa, un esquema de compilación es un conjunto de reglas que permite, a partir de una gramática de adjunción de árboles y de una estrategia de análisis sintáctico, construir un autómata lineal de índices que describa los cálculos que se pueden realizar con dicha gramática utilizando la estrategia de análisis elegida. El juego de instrucciones requerido por los esquemas de compilación nos permitirán definir diferentes modelos de autómata especializados. En la literatura se han descrito las clases de los *autómatas lineales de índices orientados a la derecha*, que permiten definir estrategias en las cuales las pilas de adjunciones se construyen de modo ascendente, de los *autómatas lineales de índices orientados a la izquierda*, en los cuales las pilas de adjunciones se construyen de modo descendente, para finalizar con los *autómatas lineales de índices universales*, que permiten la descripción de estrategias mixtas. A modo de ejemplo, incluimos las reglas de compilación correspondientes al tratamiento del nodo pie y del nodo raíz de un árbol auxiliar con motivo de una adjunción, para una estrategia de tipo Earley sin la propiedad del prefjo válido:

AdjPred	$\nabla^{\gamma}_{r,s}[\circ\circ] \longmapsto \nabla^{\gamma}_{r,s}[\circ\circ] \;\; \top^{\beta} \;\; [\,]$	$\beta \in \mathrm{adj}(N^{\gamma}_{r,s+1})$
FootPred	$\nabla^{\beta}_{f,0}[\circ\circ] \longmapsto \nabla^{\beta}_{f,0}[\circ\circ] \;\; N^{\gamma}_{r,s+1}[\,]$	$N^{\beta}_{f,0} = \mathbf{F}^{\beta},\ \beta \in \mathrm{adj}(N^{\gamma}_{r,s+1})$
FootComp	$\nabla^{\beta}_{f,0}[\,] \;\; N^{\gamma}_{r,s+1}[\circ\circ] \longmapsto \nabla^{\beta}_{f,1}[\circ\circ N^{\gamma}_{r,s+1}]$	$N^{\beta}_{f,0} = \mathbf{F}^{\beta},\ \beta \in \mathrm{adj}(N^{\gamma}_{r,s+1})$
AdjComp	$\nabla^{\gamma}_{r,s}[\,] \;\; \top^{\beta}[\circ\circ N^{\gamma}_{r,s+1}] \longmapsto \nabla^{\gamma}_{r,s+1}[\circ\circ]$	$\beta \in \mathrm{adj}(N^{\gamma}_{r,s+1})$

La técnica de programación dinámica utilizada para ejecutar los autómatas a pila en tiempo polinomial se puede extender al caso de los autómatas lineales de índices, manteniendo las complejidades estándar del análisis sintáctico de gramáticas de adjunción de árboles: $\mathcal{O}(n^6)$ en tiempo y $\mathcal{O}(n^4)$ en espacio.

6.4. Representación compartida de los árboles de derivación

Los algoritmos mostrados hasta el momento, tal y como han sido descritos, son realmente reconocedores y no analizadores sintácticos, puesto que no construyen árboles de derivación. Sin embargo, cada uno de los pasos deductivos contiene la información necesaria para generar la parte correspondiente de un árbol de derivación y, puesto que todos los algoritmos recorren todas las posibles derivaciones, se pueden reconstruir todos los posibles árboles de derivación.

Es posible representar el bosque compartido mediante una gramática independiente del contexto que capture la independencia de la operación de adjunción. Los no terminales de la gramática serán de la forma $\langle tb, N^\gamma, i, j, p, q\rangle$ donde $tb \in \{\top, \bot\}$ se utiliza para indicar si el no terminal representa al nodo N^γ antes ($\bot$) o después ($\top$) de una adjunción. Es interesante observar que los no terminales son casi idénticos a los ítems utilizados en el esquema de análisis CYK. Puesto que los ítems de los restantes esquemas son un refinamiento de los ítems de CYK la información necesaria para los no terminales se puede obtener directamente a partir de los ítems. Respecto a la forma de las producciones, a modo de ejemplo, mostramos la producción correspondiente a la adjunción del árbol auxiliar β en el nodo N^γ:

$$\langle \top, N^\gamma, i, j, r, s\rangle \rightarrow \langle \top, \mathbf{R}^\beta, i, j, p, q\rangle \ \langle \bot, N^\gamma, p, q, r, s\rangle$$

la cual se corresponde con el paso deductivo de adjunción del esquema de análisis CYK. Al igual que ocurría con los no terminales, las producciones del bosque de análisis se pueden obtener directamente a partir de los pasos deductivos en los diferentes esquemas de análisis.

El número de producciones es $\mathcal{O}(n^6)$ y la construcción de la gramática tienen una complejidad temporal $\mathcal{O}(n^6)$, por lo que la complejidad temporal de los algoritmos permanece inalterable, aunque la complejidad espacial aumenta de $\mathcal{O}(n^4)$ ó $\mathcal{O}(n^5)$ a $\mathcal{O}(n^6)$. Un aspecto interesante a destacar es que aunque el bosque de análisis construido de esta forma codifica las derivaciones para una cadena de entrada dada, el lenguaje derivado por la gramática independiente del contexto no es importante. Lo que importa es que el lenguaje generado es no vacío si la cadena pertenece a la TAG original y en tal caso las derivaciones para la TAG original pueden ser obtenidas en tiempo lineal a partir de las derivaciones de la gramática independiente del contexto que codifica el bosque compartido, siempre que ésta haya sido podada para eliminar los símbolos inútiles. El algoritmo de extracción de derivaciones es similar al de la figura 5.1.

6.5. Gramáticas de adjunción de árboles probabilísticas

Se han propuesto cuatro modelos diferentes de TAG estocásticas que dependen del esquema de asignación de probabilidades [48]. Estos modelos son enumerados a continuación, según su capacidad creciente para describir fenómenos derivacionales:

1. El primer modelo sólo asocia probabilidades a los árboles elementales de tal modo que la la suma de las probabilidades de todos los árboles auxiliares con la misma etiqueta en la raíz sume 1 y la suma de las probabilidades de los árboles iniciales con la misma etiqueta en la raíz sume también 1.

2. El segundo modelo es equivalente a las TAG probabilísticas (*Probabilistic Tree Adjoining Grammars*, PTAG) definidas por Resnik en [191] y a las TAG lexicalizadas estocásticas (*Stochastic Lexicalized Tree Adjoining Grammars*, SLTAG) propuestas por Schabes en [202], en las que la suma de las probabilidades de que una derivación comience por un árbol inicial debe ser 1, la suma de las probabilidades de adjunción en un nodo debe ser igual a 1 y la suma de las probabilidades de sustitución en un nodo deber ser también igual a 1. Este tipo de gramáticas puede ser extraído automáticamente a partir de un banco de árboles [169].

3. En el tercer modelo se asocian probabilidades con una meta-gramática independiente del contexto que codifica las posibles derivaciones de la gramática. La suma de las probabilidades de las meta-producciones asociadas a un árbol dado debe ser 1.

4. El cuarto modelo considera una gramática de sustitución de árboles estocástica obtenida a partir de un banco de árboles (*treebank*) en la que cada árbol tiene una probabilidad asociada. Las probabilidades de todos los árboles con el mismo símbolo no terminal deben sumar 1.

Si denotamos mediante $\mathrm{op}(\gamma, \gamma', N^\gamma),\ \mathrm{op} \in \{\mathrm{subs}, \mathrm{adj}\}$, cada una de las operaciones que toman parte en una derivación y si denotamos mediante $\alpha_0 \in \boldsymbol{I}$ el árbol por el que comienza una derivación, la probabilidad $P(\tau)$ de una derivación $\tau = (\alpha_0, \mathrm{op}_1(\ldots), \ldots, \mathrm{op}_m(\ldots))$ se calcula mediante la fórmula

$$P(\tau) = P_I(\alpha_0) \prod_{1 \leq i \leq m} P_{\mathrm{op}}(\mathrm{op}_i(\ldots))$$

que captura la propiedad de independencia de las operaciones de adjunción y sustitución. Estas proabilidades se pueden calcular al extraer los árboles individuales del bosque de análisis, o bien se puede ir calculando, en el propio curso del análisis sintáctico, la mejor derivación en cada momento. Asumiendo que la probabilidad asociada a cada derivación de una cadena de entrada está bien definida, la probabilidad de una cadena es igual a la suma de las probabilidades de todas las derivaciones de dicha cadena. Se dice que una gramática probabilística es *consistente* si las probabilidades asociadas a todas las cadenas del lenguaje suman 1. Al igual que ocurría en el caso de las gramáticas independientes del contexto probabilísticas, la consistencia no es excesivamente importante en la práctica.

Capítulo 7

Análisis semántico

La semántica estudia el significado de las frases. En este tema nos dedicaremos a estudiar las representaciones formales que nos permiten capturar el significado y los algoritmos que nos permiten trabajar con tales representaciones.

7.1. Estructuras de rasgos y formalismos basados en unificación

Una estructura de rasgos es un conjunto de pares atributo-valor, donde los atributos son símbolos atómicos y los valores pueden ser símbolos atómicos o estructuras de rasgos. Tales estructuras son tradicionalmente ilustradas mediante matrices atributo-valor como la siguiente

$$\begin{bmatrix} \text{CAT} & \text{S} \\ \text{HEAD} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

La utilización de estructuras de rasgos como valores de los atributos permite agrupar ciertos atributos con el fin de que reciban tratamientos conjuntos. Por ejemplo, los rasgos de número y persona se suelen tratar al mismo tiempo puesto que son los que establecen las restricciones de concordancia entre el sujeto y el verbo de una oración. La utilización de estructuras de rasgos como valores también nos lleva a la noción de *camino de rasgos*, que no es más que la lista de rasgos que se tienen que recorrer para llegar a un

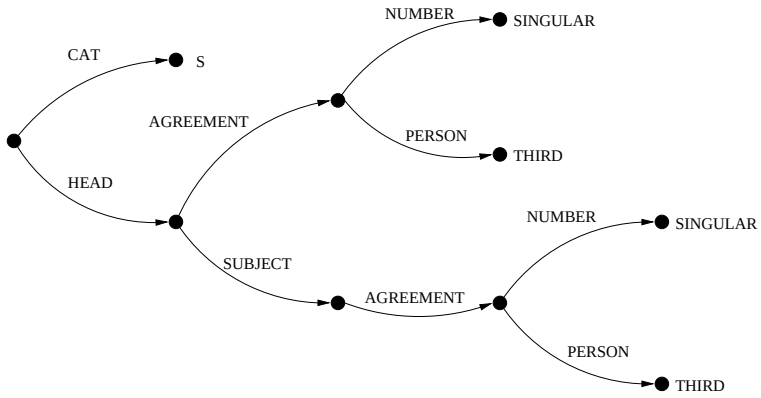

Figura 7.1: Grafo dirigido de una estructura de rasgos

determinado valor. En la estructura anterior, <HEAD AGREEMENT> es el camino que nos lleva al valor del rasgo AGREEMENT, que a su vez forma parte del rasgo HEAD. Esta noción de camino nos lleva a una representación alternativa, en la forma de un *grafo*, de las estructuras de rasgos. En estos grafos, los nodos representan valores mientras que los arcos representan rasgos. Como ejemplo, la anterior matriz atributo-valor y el grafo de la figura 7.1 son equivalentes.

Una estructura de rasgos es *reentrante* si comparte valores con otra estructura de rasgos. La compartición implica no sólo que un determinado rasgo tenga el misma valor que otro rasgo de otra estructura de rasgos, sino que comparten exactamente la misma estructura. En la representación en forma de grafo, esto implica que comparten un mismo nodo. Por ejemplo, en la estructura de rasgos anterior, los rasgos <HEAD AGREEMENT> y <HEAD SUBJECT AGREEMENT> de la oración tienen el mismo valor, lo que resulta en nodos distintos en el grafo de la figura 7.1. En cambio, en la siguiente estructura de rasgos el rasgo <HEAD AGREEMENT> comparte su valor con el rasgo <HEAD SUBJECT AGREEMENT>. En una matriz, este hecho se representa mediante un puntero $\boxed{1}$ que denota el valor del rasgo. En la representación mediante el grafo de la figura 7.2, ambos rasgos apuntan al mismo nodo valor. La compartición implica que una modificación del valor en uno de los rasgos también afecta al otro rasgo.

$$\begin{bmatrix} \text{CAT} & \text{S} \\ \text{HEAD} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Las producciones de las gramáticas independientes del contexto se pue-

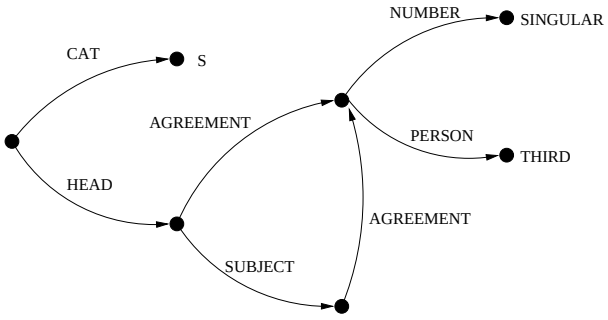

Figura 7.2: Grafo dirigido de una estructura de rasgos reentrante

den *aumentar* mediante la asociación de estructuras de rasgos a los distintos terminales y no terminales que las componen. En el proceso de análisis se *unifican* las distintas estructuras de rasgos de los elementos de la parte derecha de una producción con el fin de garantizar que se satisfacen las restricciones de compatibilidad establecidas y para componer la estructura de rasgos asociada al no terminal del lado izquierdo. Las estructuras de rasgos se utilizan habitualmente para establecer las restricciones de concordancia entre las distintas partes de la oración, para la determinación de la estructura de subcategorización verbal, y para representar las dependencias de larga distancia.

En una producción independiente del contexto aumentada con rasgos como la siguiente $S \rightarrow NP \quad VP \quad < NP \quad$ AGREEMENT $>=< VP \quad$ AGREEMENT $>$ se establece la restricción de que el rasgo AGREEMENT del no terminal NP debe *unificar* con el correspondiente rasgo del no terminal VP. La operación de *unificación* toma como argumentos dos estructuras de rasgos y devuelve la estructura de rasgos resultado de unir dichas estructuras de rasgos. Para que una unificación tenga éxito, los valores de todos los caminos existentes en ambas estructuras de rasgos deben ser compatibles. Si un camino existe un una de las estructuras de rasgos pero no en la otra, simplemente pasa a formar parte del resultado. La unificación se puede implementar de forma *no destructiva*, en la cual la estructura de rasgos resultante se crea mediante la copia de las estructuras de entrada, o *destructiva*, en la cual la estructura final se crea a partir de la modificación de las estructuras de entrada. La operación de unifcación puede ser incorporada a prácticamente todos los algoritmos de análisis sintáctico para gramáticas independientes del contexto. El grado de dificultad de la adaptación depende en gran medida de las características propias de cada algoritmo.

7.2. Lógica de predicados de primer orden

La lógica de predicados de primer orden proporciona un marco flexible, bien conocido y computacionalmente viable para el tratamiento de la semántica del lenguaje natural, ya que nos provee de una base computacional sólida al tiempo que satisface los requerimientos de inferencia y expresividad requeridos para esta tarea, aunque su característica más atractiva viene dada por el hecho de que prácticamente no establece limitaciones en el modo en que se deberán representar las cosas, que únicamente deben formar parte de un universo de objetos, propiedades de objetos y relaciones entre objetos, que adquirirán su significado en virtud de su correspondencia con los objetos, propiedades y relaciones del mundo real que se tratan de modelar.

En consecuencia, capturar el significado de una frase involucrará identificar los términos y predicados que se correspondan con los elementos gramaticales de la frase, así como crear las fórmulas lógicas que capturen las relaciones implicadas por las palabras y la sintaxis de la frase. Por ejemplo, la semántica de la frase *Carrefour está cerca de la Facultad de Informática* podría representarse como *Cerca(UbicaciónDe(Carrefour), UbicaciónDe(FacultadDeInformática))* como resultado de relacionar los términos *UbicaciónDe(Carrefour)* y *UbicaciónDe(FacultadDeInformática)* mediante el predicado *cerca*, así como de la relación entre dichos términos y predicados con los objetos y relaciones del mundo real.

La utilización de *variables* nos permite representar objetos anónimos y también referirnos genéricamente a todos los objetos de una colección. Para ello necesitamos utilizar los *cuantificadores* existencial y universal. La necesidad del cuantificador existencial viene dada por la presencia de frases nominales indefinidas. Por ejemplo, la frase *un hipermercado vende ordenadores cerca de la Faculta de Informática* se representaría como $\exists x$, *Hipermercado(x)* $\wedge$ *Vende(x,Ordenador)* $\wedge$ *Cerca(UbicaciónDe(x), UbicaciónDe(FacultadDeInformática))*. Informalmente, el cuantificador existencial nos indica que, para que la frase sea cierta, debe existir al menos un objeto tal que si lo sustituimos por la variable x la frase resultante sería cierta. El operador universal se utiliza para indicar que, para que una fórmula lógica sea cierta, la sustitución de cualquier objeto disponible en la base de conocimiento por la variable cuantificada universalmente deberá resultar en una fórmula cierta. Por ejemplo, la frase *todos los hipermercados venden ordenadores* se representará mediante $\forall x$, *Hipermercado(x)* $\Rightarrow$ *Vende(x,Ordenador)*.

La utilización de predicados para representar categorías no es siempre lo más adecuado, por lo que es preferible utilizar una relación de membresía entre la propiedad y el objeto involucrado. Por ejemplo, es preferible representar el hecho de que Carrefour es un hipermercado mediante la rela-

ción *Es-un(Hipermercado, Carrefour)* que mediante el predicado *Hipermercado(Carrefour)*.

En la representación de eventos se pueden utilizar predicados simples con tantos argumentos como se necesite. En el caso de los verbos, este enfoque simplemente asume que el predicado que representa el significado del verbo tiene el mismo número de argumentos que la estructura de subcategorización del verbo. Sin embargo, este enfoque conlleva una serie de inconvenientes, entre los que destacan la dificultad para determinar el número correcto de argumentos de un evento determinado y la dificultad de garantizar que todas las inferencias correctas se pueden derivar de la representación del evento y que no es posible realizar ninguna inferencia incorrecta. Por ejemplo, diferentes eventos del verbo *comer* necesitan ser representados mediante diferentes predicados: *Comer_1(Persona)*, *Comer_2(Persona, Bocadillo)*, *Comer_3(Persona, Bocadillo, Mesa)*, *Comer_4(Persona, Mesa)*, *Comer_5(Persona, Almuerzo)*, *Comer_6(Persona, Bocadillo, Almuerzo)*, *Comer_7(Persona, Bocadillo, Almuerzo, Mesa)*. Para indicar las relaciones entre ellos podemos utilizar *postulados de significado*, que establecen explícitamente las relaciones entre la semántica de dos o más predicados. Un ejemplo de postulado sería el siguiente: $\forall w, x, y, z$, *Comer_7(w,x,y,z)* $\Rightarrow$ *Comer_7(w,x,y)*. Otro enfoque más escalable consiste en tomar la estructura del predicado más general y utilizar cuantificadores y variables lógicas para representar los demás. En el caso de nuestro ejemplo: $\exists x, y, z$, *Comer(Persona, x, y, z)*; $\exists y, z$, *Comer(Persona, Bocadillo, y, z)*; $\exists y$, *Comer(Persona, Bocadillo, y, Mesa)*; $\exists x, y$, *Comer(Persona, x, y, Mesa)*; $\exists x, z$, *Comer(Persona, x, Almuerzo, z)*; $\exists z$, *Comer(Persona, Bocadillo, Almuerzo, z)*; *Comer(Persona, Bocadillo, Almuerzo, Mesa)*.

Más complicada se presenta la representación del tiempo, mediante la cual se trata de establecer la secuencia temporal en la que tienen lugar los eventos, y del aspecto, mediante la cual se trata de indicar por ejemplo si un determinado evento se está realizando o ya se ha terminado de realizar.

La lógica de predicados nos proporciona los mecanismos necesarios para inferir nuevos hechos que añadir a nuestra base de conocimiento. En particular, el mecanismo de inferencia más utilizado en el tratamiento de la semántica es el *modus ponens*:

$$\frac{\begin{array}{c}\alpha \\ \alpha \Rightarrow \beta\end{array}}{\beta}$$

que nos indica que si α es cierto, y $\alpha \Rightarrow \beta$, entonces β es también cierto. El modus ponens puede ser aplicado mediante encadenamiento hacia adelante

o hacia atrás. En el *encadenamiento hacia adelante*, según los hechos individuales se van añadiendo a la base de conocimiento, se activan todas las reglas de implicación aplicables. En el *encadenamiento hacia atrás*, primero se busca si la fórmula que se desea probar está ya en la base de conocimiento. Si no es así, se busca una regla de implicación tal que la fórmula consecuente coincida con aquella que se dea probar, pasando a tratar de probar su fórmula antecedente.

La combinación de la lógica de predicados con el lambda-cálculo proporciona un método adecuado para aumentar las gramátias independientes del contexto. Por ejemplo, en la siguiente producción asignamos una semántica parametrizada al verbo "vender":

$$Verbo \rightarrow \text{“}vender''\ \ \{\lambda x \lambda y\ \ \exists e\ \ \textit{Es-un}(e, vender) \wedge Sujeto(e, y) \wedge Objeto(e, x)\}$$

Los argumentos de dicha semántica se pueden ir realizando mediante la aplicación de producciones como la siguiente:

$$VP \rightarrow Verb\ \ NP \quad \{Verb.sem(NP.sem)\}$$

en la que la semántica del sintagma nominal situado a continuación del verbo reemplazará al primer parámtro de la semántica del verbo involucrado. En particular, en nuestro ejemplo se trataría de la representación semántica del objeto vendido.

7.3. Relaciones léxicas: WordNet

La utilidad de las relaciones léxicas ha llevado a la creación de grandes bases de datos de tales relaciones. WordNet, una de estas bases de datos, creada de cero y no a partir de diccionarios ya existentes, se ha convertido en pocos años en la más usada para el inglés. Su éxito ha propiciado la aparición de una versión para diversas lenguas europeas, entre ellas el español, denominada EuroWordNet.

WordNet está formada por un conjunto de ficheros en los que se almacena información de sustantivos, verbos, adjetivos y adverbios. Como se puede observar, no incluye las palabras pertenecientes a categorías cerradas. Cada uno de estos ficheros consta de un conjunto de entradas léxicas correspondientes a formas ortográficas únicas, acompañadas de los conjuntos de *sentidos* asociados a cada forma. Cada sentido de WordNet contiene un conjunto de sinónimos, una definición como las que habitualmente se encuentran en los diccionarios, y algunos ejemplos de uso. A diferencia de los diccionarios comunes, WordNet no proporciona transcripciones fonéticas y tampoco intenta

distinguir entre homonimia y polisemia, simplemente lista todos los posibles sentidos para una palabra. En lo referente a formas léxicas, las últimas versiones almacenan en torno a 95.000 sustantivos, 10.000 verbos, 20.000 adjetivos y 4.500 adverbios. En cuanto a sentidos, contienen aproximadamente 116.000 para sustantivos, 22.000 para verbos, 30.000 para adjetivos y 5.700 para adverbios.

Respecto a las relaciones léxicas, la más importante en WordNet es la de *sinonimia*. De hecho, WordNet se organiza en torno a la noción de *synset*, que no más que un conjunto de sentidos sinónimos, considerando que dos sentidos son sinónimos si pueden ser sustituidos con éxito en algún contexto. Podemos considerar que cada synset representa un concepto que ha sido lexicalizado en el lenguaje. Cada synset está relacionado con los synsets inmediatamente más generales y específicos mediante enlaces directos de *hiperonimia* y de *hiponimia*. Estas dos relaciones transitivas y antisimétricas establecen una jerarquía de tipo es-un que es muy útil en el caso de los sustantivos. Las relaciones de *meronimia* y *holonimia* también son transitivas y antisimétricas, pero en este caso establecen una jerarquía de tipo parte-de, utilizada principalmente en el caso de los sustantivos. La relación de *antonimia* es especial en el sentido de que se establece entre formas y no entre sentidos, siendo útil principalmente en el caso de los adjetivos.

En el futuro se esperan añadir también enlaces entre distintas categorías, de modo que sea posible obtener el sustantivo que se corresponde con la acción de un determinado verbo, o el adjetivo correspondiente a un sustantivo.

7.4. Desambiguación del sentido de las palabras

Si no se dispone de ningún medio para seleccionar los sentidos correctos de las palabras de un texto, la enorme cantidad de homonimia y polisemia presente en los diccionarios producirá una avalancha de posibles interpretaciones diferentes de palabras y frases. La desambiguación del sentido de las palabras se puede realizar conjuntamente con la fase de análisis semántico, reduciendo el conjunto de posibles sentidos válidos a medida que se van rechazando interpretaciones incorrectas, o bien se puede realizar en una etapa independiente, previa al análisis semántico. Si bien el primer enfoque tiene una clara motivación lingüística, resulta bastante problemático en la práctica, debido en primer lugar a que las restricciones de selección son habitualmente demasiado generales, con lo que lo más probable es que se obtengan muchas interpretaciones semánticas diferentes, y en segundo lugar porque el coste del

análisis semántico es elevado, con lo cual es necesario podar cuanto antes las interpretaciones no válidas.

En el caso de realizar la desambiguación como una etapa independiente, se puede seguir un enfoque basado en diccionarios o uno basado en aprendizaje automático. En el primer caso, todas las definiciones de sentidos de las palabras a desambiguar se tomarán de un diccionario. Cada uno de estos sentidos será comparado con las definiciones en el diccionario de las palabras ubicadas en un contexto cercano, eligiendo aquel sentido que presente un mayor solapamiento. El rendimiento se puede mejorar incorporando las definiciones de las palabras relacionadas.

En el segundo enfoque se trata de entrenar el sistema para realizar la tarea de la desambiguación. Si se dispone de un corpus anotado con el sentido de cada una de las palabras, se puede realizar un aprendizaje supervisado. Como los corpus anotados de este tipo suelen ser pequeños, es habitual utilizar un enfoque de *bootstrapping*, en el cual se crea un desambiguador a partir del corpus original, el cual es utilizado para anotar un corpus mayor, que una vez corregido puede ser utilizado para generar un desambiguador mejor, y así sucesivamente. Si no se dispone de un corpus anotado, se puede aplicar un aprendizaje no supervisado. En cualquier caso, la *palabra objeto* de la desambiguación se considerará conjuntamente con un cierto número de palabras situadas a su alrededor, que formarán el *contexto*. Las operaciones complementarias más habituales son la etiquetación de las palabras del texto, el reemplazo del contexto original por otro más pequeño o más grande, la obtención de palabras relacionadas por morfología derivativa y la realización de un análisis sintáctico superficial para establecer las relaciones de dependencia entre las palabras.

Capítulo 8

Recuperación y extracción de información

Los campos de la recuperación y extracción de información adquieren cada día mayor importancia debido a la disponibilidad de textos en formato electrónico y al aumento del acceso público a información contenida en CD-ROM e Internet. La creación de sistemas de recuperación de información involucra la realización de múltiples tareas relativas tanto al almacenamiento como a la recuperación de información escrita y audiovisual. En este tema nos centraremos en la información textual, mostrando diferentes técnicas de procesamiento del lenguaje natural que parecen incrementar la efectividad de este tipo de sistemas.

8.1. Modelos de recuperación de información

Los sistemas tradicionales de recuperación de información adoptan generalmente términos índices para indexar y recuperar los documentos. Un término índice es una palabras clave, o un grupo de palabras relacionadas, que posee un significado por sí misma. En su forma más general, un término índice es simplemente cualquier palabra que aparece en el texto de un documento. La recuperación de información basada en palabras clave se fundamenta en la idea de que tanto la semántica de los documentos como la de la consulta del usuario puede ser expresada naturalmente a través de conjuntos de términos índice. Esto supone una simplificación considerable del problema puesto que gran parte de la semántica de un documento se pierde cuando se considera que está formado únicamente por un conjunto de palabras sin relaciones estructurales entre ellas.

El problema principal con el que se enfrenta un sistema de recuperación

de información es el de predecir qué documentos son relevantes y cuáles no lo son con respecto a la necesidad de información del usuario, establecida en la forma de una consulta. Tal decisión depende generalmente de un algoritmo de ordenación que intenta establecer un orden simple entre los documentos recuperados. Aquellos documentos que aparecen en las primeras posiciones de la clasificación son considerados más relevantes que los otros. Existen diversos *modelos* para determinar la relevancia de un documento. Los más conocidos y utilizados son el modelo booleano, el vectorial y el probabilístico.

Sea t el número de términos índice y k_i uno de esos términos. $K = \{k_1, \ldots, k_t\}$ es el conjunto de todos los términos índice. Se asociará un peso $w_{i,j} > 0$ para cada término k_i de un documento d_j. Si el término no aparece en el documento, entonces $w_{i,j} = 0$. Todo documento d_j tiene asociado un vector de términos índice $\vec{d_j} = (w_{i,j}, w_{2,j}, \ldots, w_{t,j})$. La función g_i devuelve el peso asociado con el término k_i en cualquier vector t-dimensional: $g_i(\vec{d_j}) = w_{i,j}$. El *modelo booleano* considera que todos los pesos son binarios, esto es, $w_{i,j} \in \{0, 1\}$. Una consulta q estará formada por términos índice ligados por conectivas lógicas. Sea $\vec{q}_{dnf}$ la forma normal disyuntiva para la consulta q y sea $\vec{q}_{cc}$ cualquiera de los componentes conjuntivos de $\vec{q}_{dnf}$. La similitud de un documento d_j a una consulta se define como

$$sim(d_j, q) = \begin{cases} 1 \text{ si } \exists \vec{q}_{cc} \mid \vec{q}_{cc} \in \vec{q}_{dnf} \wedge \forall k_i, g_i(\vec{d_j}) = g_i(\vec{q}_{cc}) \\ 0 \text{ en otro caso} \end{cases}$$

si $sim(d_j, q) = 1$ entonces el modelo booleano predice que el documento d_j es relevante a la consulta q. En otro caso, se considera que el documento no es relevante.

A diferencia del modelo booleano, el *modelo vectorial* permite concordancias parciales de los documentos con las partes de la consulta. En este caso, el peso $w_{i,j}$ asociado con un par (k_i, d_j) es positivo y no binario. Los pesos de los términos índice en la consulta son también ponderados. Sea $w_{i,q}$ el peso asociado con el par $[k_i, q]$, donde $w_{i,q} \geq 0$. En tal caso podemos definir el vector de la consulta como $\vec{q} = (w_{i,q}, \ldots, w_{t,q})$. Como antes, el vector para un documento es $\vec{d_j} = (w_{i,j}, w_{2,j}, \ldots, w_{t,j})$. En consecuencia, tanto los documentos como las consultas se representan como vectores en un espacio t-dimensional. El modelo vectorial propone evaluar el grado de similitud entre el documento d_j con respecto a la consulta q como el coseno del ángulo entre ambos vectores:

$$sim(d_j, q) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

Normalmente se utiliza la fórmula tf-idf para calcular los pesos de los términos

en los documentos: $w_{i,j} = f_{i,j} \times \log \frac{N}{n_i}$, donde $f_{i,j} = \frac{freq_{i,j}}{max_l\ freq_{i,j}}$ es la frecuencia normalizada de aparición de un término en un documento mientras que $\log \frac{N}{n_i}$ es la frecuencia inversa de documento.

En el *modelo probabilístico*, todos los pesos son binarios, esto es, $w_{i,j} \in \{0,1\}$ y $w_{i,q} \in \{0,1\}$. Una consulta q es un subconjunto de términos índice. Sea R el conjunto de documentos relevantes, mientras que $\bar{R}$ es el complemento de R. Sea $P(R|\vec{d_j})$ la probabilidad de que el documento d_j sea relevante para la consulta q y $P(\bar{R}|\vec{d_j})$ la probabilidad de que d_j no sea relevante. La medida de similitud entre d_j y la consulta q se define como

$$sim(d_j, q) = \frac{P(R|\vec{d_j})}{P(\bar{R}|\vec{d_j})} = \frac{P(\vec{d_j}|R) \times P(R)}{P(\vec{d_j}|\bar{R}) \times P(\bar{R})}$$

Para evaluar la bondad de una determinado sistema con respecto a un conjunto de consultas y documentos, se utilizan las medidas de *precisión*, que mide el porcentaje de documentos relevantes recuperados con respecto al número total de documentos recuperados y la *cobertura*, que mide el porcentaje de documentos relevantes recuperados con respecto al total de documentos relevantes. También suele ser útil realizar un gráfico de la evolución de la precisión con respecto a la cobertura. Además de estas medidas básicas, existen otras como son la *precisión-R*, la precisión en un determinado punto de la salida (el punto de corte) y la *precisión media con respecto a los documentos relevantes*.

8.2. Aplicación de la morfología a la normalización de términos simples

Aunque las nuevas generaciones de ordenadores están haciendo posible la representación de un documento por su conjunto completo de palabras, al trabajar con grandes colecciones de documentos todavía debemos seguir limitando el conjunto de palabras clave representativas. Para ello se recurre a operaciones tales como la eliminación de *stopwords* (palabras excesivamente frecuentes y sin significación aparente) o técnicas de *stemming* (las cuales reducen las palabras a una supuesta raíz gramatical). A dicho tipo de operaciones se las denomina *operaciones de texto*, y generan una *vista lógica* del documento procesado. En efecto, los sistemas de recuperación de información normalizan los documentos antes de su indexación para decrementar su variabilidad lingüística mediante la agrupación de términos referentes a conceptos similares explotando para ello similitudes gráficas, thesaurus, etc. Desafortunadamente, las técnicas más empleadas carecen habitualmente de

base lingüística. Incluso aquellas técnicas que presumiblemente gozan de ella (por ejemplo *stemming*), si bien consiguen resultados muy aceptables para el inglés, se muestran del todo insuficientes cuando se aplican a idiomas más ricos desde el punto de vista morfológico, caso del español.

Una *familia morfológica* es el conjunto de palabras obtenidas a partir de una misma raíz mediante la aplicación de mecanismos de derivación. Es de esperar que exista una relación semántica entre las palabras de dicho conjunto, relación que normalmente es de tipo proceso-resultado (por ejemplo *fijación-fijado*), proceso-agente (por ejemplo *inhibición-inhibidor*), y similares.

Dadas dos palabras w y w' del diccionario, denotaremos por $w \triangleright w'$ el hecho de que w' sea obtenida a partir de w mediante algún mecanismo de derivación, teniendo en cuenta los posibles alomorfos y los ajustes fonológicos que se deben satisfacer al realizar la derivación. La familia morfológica de w se calcula como closure(w), su cierre reflexivo y transitivo mediante derivación. Esta operación de cierre se define recursivamente de la siguiente forma:

- $w \in \text{closure}(w)$.
- Si $w \triangleright w'$ entonces $w' \in \text{closure}(w)$.
- Si $w' \triangleright w$ entonces $w' \in \text{closure}(w)$.

Observamos que el método propuesto opta por sobregenerar, es decir, aplica todos los sufijos posibles obteniendo todos los derivados morfológicamente válidos, los cuales pueden ser filtrados mediante el lexicón. De este modo se resuelve el problema de la decisión sobre la validez y aceptación del término derivado a través únicamente de la forma léxica y de su etiqueta, sin considerar otros aspectos.

El conjunto de familias morfológicas de un diccionario el igual a la colección de conjuntos distintos obtendios tras aplicar la operación closure(w) a cada una de las palabras w que lo componen.

Con respecto al coste computacional, las familias morfológicas, al haber sido generadas a priori, no influyen en el coste final de indexación y consulta. El coste de ejecución de un stemmer es lineal en la longitud de la palabra, siendo el del lematizador sólo ligeramente superior debido al proceso previo de desambiguación. Dicho coste será también lineal respecto a la longitud de la palabra, pero cúbico respecto al tamaño del conjunto de etiquetas.

8.3. Aplicación de la sintaxis a la normalización de términos multipalabra

En el ámbito de la recuperación de información se denomina *término multipalabra* a aquel término que contiene dos o más palabras con contenido (sustantivos, verbos y adjetivos)[1]. En la literatura se describen varios métodos para su obtención. Uno de los más utilizados es el denominado *simplificación del texto*: en una primera fase, se realiza un stemming de las palabras individuales, y se procede a eliminar las stopwords; posteriormente se extraen y normalizan los términos empleando para ello patrones, técnicas estadísticas, etc. Existe, pues, una clara falta de base lingüística en dichas operaciones[2], lo que redunda frecuentemente en simplificaciones erróneas. Sin embargo, es el método más sencillo y menos costoso. Por otra parte, existen otros métodos de sólida base lingüística que realizan un *análisis sintáctico* del texto mediante un analizador sintáctico, el cual devuelve a su salida un conjunto de árboles sintácticos que denotan relaciones de dependencia entre las palabras involucradas. De este modo, estructuras con relaciones de dependencia similares pueden ser normalizadas a una misma forma. A medio camino estaría la *correspondencia de patrones*, que se basa en la hipótesis de que las partes más informativas del texto siguen unas construcciones sintácticas bastante bien definidas que se pueden aproximar mediante patrones.

Una aproximación que conjuga los dos últimos métodos es la que se basa en la indexación de sintagmas nominales y de sus *variantes sintácticas y morfosintácticas* [96]. Una variante sintáctica o morfosintáctica de un término multipalabra es una frase perteneciente al texto, tal que:

- Las variantes sintácticas son producto de la flexión de palabras individuales y de la modificación de la estructura sintáctica del término original. Por ejemplo, *medidas de longitud y tiempo* es una variante de *medida de tiempo*.

- Las variantes morfosintácticas difieren de las anteriores en que al menos una de las palabras con contenido del término original se transforma en otra que deriva de la misma raíz morfológica. Por ejemplo, *medición del contenido* es una variante de *medir el contenido*.

- La variante puede ser sustituida por el término original del que deriva en lo que respecta al acceso a la información.

[1] Por ejemplo, *el perro grande del vecino* o *la casa de mis padres*.

[2] Por ejemplo, algunas stopwords tales como artículos y preposiciones son componentes clave de la estructura sintáctica.

Desde el punto de vista morfológico, las variantes sintácticas hacen referencia a la morfología flexiva, mientras que las morfosintácticas entran además en el ámbito de la morfología derivativa. En lo referente a la sintaxis, las variantes sintácticas tienen un campo de actuación mucho más restringido, el sintagma nominal, mientras que las variantes morfosintácticas lo amplían a prácticamente toda la oración, incluyendo los verbos y sus objetos, tal y como se observa en las frases *negociaciones entre los ministros* y *los ministros negocian*.

Las consultas realizadas a un sistema de recuperación de información suelen expresarse en forma de grupos nominales de complejidad diversa. Es por ello que se tomarán los grupos nominales como términos base a partir de los cuales, y mediante la aplicación de las transformaciones correspondientes, se obtendrán sus variantes sintácticas y morfosintácticas, no necesariamente grupos nominales. Todos estos términos multipalabra, tanto los grupos nominales originales como sus variantes, son susceptibles de ser utilizados como términos índice.

Las estructuras básicas de los grupos nominales para el caso del español son Adjetivo-Sustantivo, Sustantivo-Adjetivo, Sustantivo-Preposición-Sustantivo, Sustantivo-Preposición-Determinante-Sustantivo. Nos interesará, por tanto, identificar tanto dichos sintagmas como sus variantes para así indexarlos. Para extraer de los documentos dichos términos, puede emplearse un analizador sintáctico superficial implementado mediante cascadas de autómatas finitos, lo que garantiza su eficiencia computacional, indispensable cuando se trata con grandes cantidades de texto.

8.4. Extracción de información

Extracción de información es el nombre dado a cualquier proceso que selectivamente estructura y combina datos que se encuentran, explícita o implícitamente, en uno o más textos. La aplicación de un sistema de extracción de información puede ir precedida por la de un sistema de recuperación de información que se encargue de obtener aquellos textos relevantes para la consulta. En general, las tareas de extracción de información se caracterizan por poseer dos propiedades fundamentales: el conocimiento que se desea extraer se puede describir mediante un conjunto relativamente simple y fijo de fichas, con apartados que deben ser cubiertos a partir de material extraído de determinados textos, y sólo una pequeña parte de la información contenida en los textos es relevante para completar las fichas, por lo que el resto puede ser ignorado.

Los sistemas que actualmente tienen mayor éxito en la tarea de extracción

de información son aquellos que utilizan un enfoque basado en autómatas finitos y que buscan patrones especificados por expertos o derivados automáticamente a partir de datos de entrenamiento y corpus. Un sistema genérico constaría de los siguientes componentes:

- Un *segmentador de textos* que selecciona las partes de los textos que serán tratadas por los demás componentes, utilizando para ello información de formato y de marcado.

- Un *preprocesador* que convierte los segmentos de texto en una secuencia de frases y palabras.

- Un *filtro* que selecciona los párrafos en los cuales debe concentrase la labor de los restantes componentes. Su misión es reducir la sobrecarga computacional total del sistema.

- Un *preanalizador sintáctico* que identifica pequeñas estructuras sintácticas (fechas, números) e identifica los nombres propios de personas, ciudades y entidades en general.

- Un *analizador sintáctico* que obtiene árboles de análisis sintáctico que pueden estar completos o no. Normalmente se restringen al análisis de frases nominales simples y complejas, así como de algunas cláusulas subordinadas simples.

- Un *combinador de fragmentos* que intenta obtener una estructura que represente el análisis de una oración completa a partir de la combinación de los árboles sintácticos parciales.

- Un *intérprete semántico* que realiza la correspondencia entre las estructuras sintácticas y las estructuras semánticas de las fichas que se deben rellenar.

- Un *desambiguador léxico* que obtiene el sentido correcto de cada una de las palabras.

- Un *resolutor de correferencias* que detecta aquellos nombres diferentes que se refieren a la misma entidad, aquellas entidades que se corresponden con partes de otras entidades, y aquellos eventos aparentemente diferentes que realmente son uno.

- Un *generador de fichas* que alimenta la base de datos con los elementos extraídos.

El campo de la extracción de información tiene puntos en común con el de la recuperación de información. En particular, ha adaptado varias métricas de este último campo, incluyendo la precisión, cobertura y una medida combinada llamada medida-F. La *precisión* P se obtiene como resultado de dividir el número de respuestas correctas obtenidas por el número total de respuestas obtenidas. La *cobertura* R se obtiene de dividir el número de respuestas correctas por el número de posibles respuestas correctas contenidas en los textos. La *medida-F* combina ambos valores mediante un parámetro β aplicando la fórmula $F = \frac{(\beta^2+1)PR}{(\beta^2 P+R)}$. Una medida útil para determinar la habilidad de un sistema para ignorar información irrelevante es el *fallout*, que se obtiene de dividir el número de respuestas incorrectas obtenidas por el número de hechos irrelevantes presentes en el texto.

Capítulo 9

Análisis pragmático

La pragmática es el estudio de la relación entre el lenguaje y el contexto en el que se utiliza. El contexto incluye elementos como la identidad de las personas y los objetos participantes, y por tanto la pragmática incluye el estudio de cómo se utiliza el lenguaje para referenciar a personas y cosas. En este tema se tratan la recuperación de la anáfora y los principios de la traducción automática.

9.1. Resolución de la anáfora

Las expresiones anafóricas hacen un texto más legible y enfatizan la conexión entre las partes del texto. La clase más habitual de anáfora es aquella producida por la utilización de pronombres que se refieren a partes de la frase que han aparecido anteriormente. Las referencias anafóricas se pueden realizar con pronombres personales, demostrativos, reflexivos, y en español con pronombres omitidos, también denominados pronombres-cero. Otro tipo de anáfora es aquel provocado por expresiones definidas, como por ejemplo *el alumno*, en la cual la persona concreta a la que nos referimos probablemente habrá sido citada anteriormente en el texto. En español, la anáfora pronominal se puede clasificar en las siguientes categorías:

- Pronombres personales clíticos *lo*, *la*, *le*, *los*, *las*, *les* que juegan el papel de los complementos. Un ejemplo sería la frase *Ana abre [la puerta]$_i$ y la$_i$ cierra.*

- Pronombres personales *él*, *ella*, *ello*, *ellos*, *ellas* no incluidos en un sintagma preposicional. Un ejemplo sería la frase *Andrés$_i$ es mi vecino. Él$_i$ vive en el segundo piso.*

- Pronombres personales *él*, *ella*, *ello*, *ellos*, *ellas* incluidos en un sintagma preposicional. Un ejemplo sería la frase *Juan$_i$ debe asistir pero Pedro lo hará por él$_i$*.

- Pronombres demostrativos *éste*, *ésta*, *esto*, *éstos*, *éstas*, *ése*, *ésa*, *aquél*, *aquélla*, *ésos*, *ésas*, *aquéllos*, *aquéllas*, no incluidos en un sintagma preposicional. Un ejemplo sería la frase *El Ferrari$_i$ ganó al Ford. Éste$_i$ es el mejor*.

- Pronombres demostrativos *éste*, *ésta*, *esto*, *éstos*, *éstas*, *ése*, *ésa*, *aquél*, *aquélla*, *ésos*, *ésas*, *aquéllos*, *aquéllas*, incluidos en un sintagma preposicional. Un ejemplo sería la frase *Ana vive con Paco$_i$ y cocina para éste$_i$ diariamente*.

- Pronombres reflexivos *se*, *sí*, *sí mismo*, *consigo*, *consigo mismo*. Un ejemplo sería la frase *Ana$_i$ abre la verja y la cierra tras de sí$_i$*.

- Pronombres omitidos. En español, sólo pueden aparecer en posición de sujeto. Un ejemplo sería la frase *Ana$_i$ abre la verja y $\emptyset_i$ la cierra tras de sí*.

Un algoritmo de resolución de la anáfora pronominal debe identificar todos los tipos de anáfora de izquierda a derecha en el orden en que aparecen en la frase. Mediante la aplicación de un conjunto de restricciones se descartan algunos de los antecedentes candidatos. Los que quedan son ordenados mediante unas determinadas preferencias. Una *restricción* define una propiedad que debe ser satisfecha para que un candidato sea considerado una posible solución de una anáfora. Por ejemplo, los pronombres que generan la anáfora y sus antecedentes deben concordar en persona, género y número, en caso contrario el antecedente debe ser descartado. También se pueden establecer restricciones sintácticas. Por ejemplo, un pronombre enclítico no puede referenciar a un nombre que está incluido en una sintagma preposicional, tal y como ocurre en la frase *Con Juan$_i$ lo$_j$ compré*. Una *preferencia* es una característica que no tiene porqué ser siempre satisfecha por la solución pero que nos permite diseñar heurísticas mediante las cuales es posible ordenar los distintos candidatos. Un ejemplo de preferencia es, en el caso de los pronombres omitidos, favorecer a aquellos sintagmas nominales que están en la misma frase y que han sido propuestos para otro pronombre omitido, o en su defecto aquellos sintagmas nominales que están en la frase precedente.

La resolución de las expresiones definidas implica igualmente mantener una lista de todas las posibles expresiones candidatas que han aparecido anteriormente en el texto. Una diferencia con respecto al tratamiento de la

anáfora pronominal radica en que estas últimas parecen tener un alcance mucho más local, mientras que el antecedente de una anáfora de expresión definidas puede aparecer muchas frases antes. Otra diferencia radica en el hecho de que la anáfora pronominal se puede resolver utilizando fundamentalmente información morfológica y sintáctica, mientras que la semántica parece jugar un papel primordial en las expresiones definidas.

9.2. Traducción automática

La traducción automática es uno de los campos más antiguos del procesamiento del lenguaje natural. Caracterizada por la utilización de ordenadores para realizar la traducción de textos escritos en lenguaje natural, con asistencia humana o sin ella, la traducción automática tiene que lidiar con prácticamente todos los problemas del procesamiento automático del lenguaje, en dos lenguas diferentes. Estos problemas se pueden resumir en una palabra, ambigüedad, que se presenta tanto a nivel léxico como sintáctico y semántico.

Para realizar la tarea de la traducción automática se pueden tomar dos enfoques principales, la traducción basada en interlíngua o la traducción basada en transferencia. El enfoque basado en *interlíngua* es el más puro desde el punto de vista lingüístico, ya que pretende obtener una representación abstracta, neutral con respecto a la lengua, del significado del texto origen que captura toda la información lingüística necesaria para generar un texto adecuado en la lengua destino. En la práctica, esta meta se ha mostrado casi imposible de conseguir, ya que las representaciones más elaboradas que se han obtenido no logran representar el significado de los textos, sino que más bien se basan en la estructura textual, lo que hace necesario disponer de algún mecanismo que transforme las representaciones obtenidas en un lenguaje a las correspondientes en el otro. A pesar de la ventaja que podría suponer un sistema basado en interlíngua, sobre todo en entornos con múltiples lenguas como el europeo, los problemas prácticos encontrados parecen favorecer el enfoque basado en *transferencia*. En este último enfoque, el proceso de traducción consta de tres etapas:

1. Análisis de la entrada con el fin de obtener la representación sintáctico-semántica del texto origen.

2. Transferencia de esta representación en una equivalente para el lenguaje destino.

3. Generación, a partir de esta estructura, del texto en lenguaje destino.

Aunque requiere de una etapa extra de procesamiento, este enfoque tiene la ventaja de que se centra en la etapa de transferencia, que es aquella en la cual se manifiestan las diferencias entre las lenguas que participan en la traducción. Los sistemas actuales que incorporan este enfoque reducen la tarea de transferencia tanto como sea posible, limitándola idealmente a la traducción de las palabras, mediante la adopción de estructuras intermedias profundas que neutralizan, en la medida de lo posible, las particularidades de las formas superficiales de las lenguas, como por ejemplo las concordancias de género y número, el orden de las palabras, el tiempo y aspecto de los verbos, etc.

Desde el punto de vista del usuario, con el tiempo se ha producido un desplazamiento desde la traducción totalmente automática a la traducción asistida por ordenador, pudiéndose distinguir dos tendencias principales, una en la que el traductor humano utiliza herramientas de ordenador parcialmente automatizadas y otra en la cual es el ordenador el que realiza la traducción, aunque todavía bajo el control del usuario, quien en este caso no es necesariamente un traductor. Los sistemas basados en *lenguajes controlados* son aquellos que trabajan bajo la hipótesis de que los textos que se van a traducir han sido restringidos de alguna manera, principalmente evitando palabras o construcciones que el sistema es incapaz de traducir. Aunque a primera vista pueda parecer bastante negativo que el ordenador nos dicte la forma de lo que debemos escribir, hemos de tener en cuenta que los escritores técnicos están habituados a disponer de un vocabulario y de un estilo preestablecido. Muchos de los sistemas actuales de mayor éxito trabajan con lenguajes controlados o con *sublenguajes* restringidos, observándose en ciertos casos un aumento de la legibilidad de los textos con respecto a los originales escrito en lenguaje *libre*. De todas formas, la salida de un sistema de traducción automática necesita habitualmente ser post-editada por un traductor humano, aunque este proceso también ocurre en las traducciones manuales, donde la actividad de unos traductores es supervisada por otros.

Bibliografía

[1] S. P. Abney, R. E. Schapire, and Y. Singer. Boosting applied to tagging and PP attachment. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very large Corpora (EMNLP/VLC'99)*, pages 38–45, College park, MD, 1999. ACL.

[2] Steven Abney. Parsing by chunks. In Robert Berwick, Steven Abney, and Carol Tenny, editors, *Principle-Based Parsing*. Kluwer Academic Publishers, Dordrecht, 1991.

[3] Steven Abney. Partial parsing via finite-state cascades. *Natural Language Engineering*, 2(4):337–344, 1997.

[4] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, USA, 1986.

[5] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation and Compiling*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1972.

[6] Miguel A. Alonso, David Cabrero, Eric de la Clergerie, and Manuel Vilares. Tabular algorithms for TAG parsing. In *Proc. of EACL'99, Ninth Conference of the European Chapter of the Association for Computational Linguistics*, pages 150–157, Bergen, Norway, June 1999. ACL.

[7] Miguel A. Alonso, David Cabrero, and Manuel Vilares. A new approach to the construction of Generalized LR parsing algorithms. In Ruslan Mitkov, Nicolas Nicolov, and Nikolai Nikolov, editors, *Proc. of Recent Advances in Natural Language Processing (RANLP'97)*, pages 171–178, Tzigov Chark, Bulgaria, September 1997.

[8] Miguel A. Alonso, David Cabrero, and Manuel Vilares. Construction of efficient generalized LR parsers. In Derick Wood and Sheng Yu, editors, *Automata Implementation*, volume 1436 of *Lecture Notes in Computer Science*, pages 7–24. Springer-Verlag, Berlin-Heidelberg-New York, 1998.

[9] Miguel A. Alonso, David Cabrero, and Manuel Vilares. Generalized LR parsing for extensions of context-free grammars. In Nicolas Nicolov and

Ruslan Mitkov, editors, *Recent Advances in Natural Language Processing II*, volume 189 of *Current Issues in Linguistic Theory*, pages 81–92. John Benjamins Publishing Company, Amsterdam & Philadelphia, 2000.

[10] Miguel A. Alonso, Eric De la Clergerie, Víctor J. Díaz, and Manuel Vilares. Relating tabular parsing algorithms for LIG and TAG. In Harry Bunt, John Carroll, and Giorgio Satta, editors, *New Developments in Parsing Technology*, volume 23, chapter 8, pages 157–184. Kluwer Academic Publishers,, Dordrecht-Boston-London, 2004.

[11] Miguel A. Alonso, Eric de la Clergerie, Jorge Graña, and Manuel Vilares. New tabular algorithms for LIG parsing. In *Proc. of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, pages 29–40, Trento, Italy, February 2000.

[12] Miguel A. Alonso, Eric de la Clergerie, and Manuel Vilares. Automata-based parsing in dynamic programming for Linear Indexed Grammars. In A. S. Narin'yani, editor, *Proc. of DIALOGUE'97 Computational Linguistics and its Applications International Workshop*, pages 22–27, Moscow, Russia, June 1997.

[13] Miguel A. Alonso, Eric de la Clergerie, and Manuel Vilares. A redefinition of Embedded Push-Down Automata. In *Proc. of 5th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+5)*, pages 19–26, Paris, France, May 2000.

[14] Miguel A. Alonso, Eric de la Clergerie, and Manuel Vilares. A formal definition of Bottom-up Embedded Push-Down Automata and their tabulation technique. In Philippe de Groote, Glyn Morril, and Christian Retoré, editors, *Logical Aspects of Computational Linguistics*, volume 2099 of *Lecture Notes in Artificial Intelligence*, pages 44–61. Springer-Verlag, Berlin-Heidelberg-New York, 2001.

[15] Miguel A. Alonso, Víctor J. Díaz, and Manuel Vilares. Bidirectional automata for tree adjoining grammars. In *Proc. of the Seventh International Workshop on Parsing Technologies (IWPT-2001)*, pages 42–53, Beijing, China, October 2001. Tsinghua University Press.

[16] Miguel A. Alonso, Víctor J. Díaz, and Manuel Vilares. Bidirectional push-down automata. In Jean-Marc Champarnaud and Denis Maurel, editors, *Seventh International Conference on Implementation and Application of Automata (CIAA 2002)*, pages 41–50, Tours, France, July 2002.

[17] Miguel A. Alonso, Mark-Jan Nederhof, and Eric de la Clergerie. Tabulation of automata for tree adjoining languages. *Grammars*, 3(2/3):89–110, 2000.

[18] E. L. Antworth. PC-KIMMO: A two-level processor for morphological analysis, 1990. Summer Institute of Linguistics, Dallas, TX.

[19] Chinatsu Aone, Lauren Halverson, Tom Hampton, and Mila Ramos-Santacruz. SRA: Description of the IE^2 system used for MUC-7. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, 1998.

[20] Douglas Appelt, Jerry R. Hobbs, John Bear, David Israel, Megumi Kameyama, and Mabry Tyson. SRI: Description of the JV-FASTUS system used for MUC-5. In B. Sundheim, editor, *Proc. of the Fifth Message Understanding Conference (MUC-5)*, Tokyo, Japan, August 1993.

[21] J. W. Backus. The syntax and semantics of the proposed international algebraic language of the Zurch ACM-GAMM Conference. In *Information Processing: Proceedings of the International Conference on Information Processing*, pages 125–132, Paris, France, 1959. UNESCO.

[22] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley and ACM Press, Harlow, England, 1999.

[23] L. R. Bahl, F. Jelinek, and R. L. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2):179–190, 1983.

[24] J. K. Baker. The DRAGON system — an overview. *IEEE Transactions on Acoustic, Speech, and Signal Processing, ASSP*, 23(1):24–29, 1975.

[25] J. K. Baker. Trainable grammars for speech recognition. In D. H. Klatt and J. J. Wolf, editors, *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550, 1979.

[26] Srinivas Bangalore. Transplanting supertags from English to Spanish. In *Proc. of Fourth International Workshop on Tree-Adjoining Grammars and Related Frameworks (TAG+4)*, pages 5–8, Philadelphia, PA, USA, August 1998.

[27] François Barthélemy. *Outils pour l'analyse syntaxiqye contextuelle*. PhD thesis, Université d'Orléans, Orleans, France, February 1993.

[28] John Bear, David Israel, Jeff Petit, and David Martin. Using information extraction to improve document retrieval. In *NIST Special Publication 500-240: The Sixth Text REtrieval Conference (TREC 6)*, pages 367–377, Gaithersburg, MD, USA, 1998. Department of Commerce, National Institute of Standards and Technology.

[29] Tilman Becker and Dominik Heckmann. Recursive matrix systems (RMS) and TAG. In *Proc. of Fourth International Workshop on Tree Adjoining*

Grammars and Related Frameworks (TAG+4), pages 9–12, Philadelphia, PA, USA, August 1998.

[30] Tilman Becker and Dominik Heckmann. Parsing mildly context-sensitive RMS. In *Proc. of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, pages 293–294, Trento, Italy, February 2000.

[31] Richard Beckwith and Georges A. Miller. Implementing a lexical network. *International Journal of Lexicography*, 3(4):302–312, 1990.

[32] Romaric Besançon, Jean-Cédric Chappelier, Martin Rajman, and Antoine Rozenknop. Improving text representations through probabilistic integration of synonymy relations. In N. Limnios G. Govaert, J. Janssen, editor, *Proceedings of the Xth International Symposium on Applied Stochastic Models and Data Analysis (ASMDA'2001)*, volume 1, pages 200–205, Compiègne, France, 2001.

[33] Sylvie Billot and Bernard Lang. The structure of shared forest in ambiguous parsing. In *Proc. of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 143–151, Vancouver, British Columbia, Canada, June 1989. ACL.

[34] Leonard Bloomfield. *An Introduction to the Study of Language*. Henry Holt and Company, New York, 1914.

[35] T. L. Booth. Probabilistic representation of formal languages. In *IEEE Conference Record of the 1969 Tenth Annual Symposium on Switching and Automata Theory*, pages 74–81, 1969.

[36] Pierre Boullier. Proposal for a natural language processing syntactic backbone. Rapport de recherche 3342, INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex, France, January 1998.

[37] Pierre Boullier. Range concatenation grammars. In *Proc. of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, pages 53–64, Trento, Italy, February 2000.

[38] Thorsten Brants. Cascade Markov models. In *Proc. of EACL'99, Ninth Conference of the European Chapter of the Association for Computational Linguistics*, pages 118–125, Bergen, Norway, June 1999. ACL.

[39] Thorsten Brants. TnT - a statistical part-of-speech tagger. In *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP'2000)*, Seattle, 2000.

[40] M. R. Brent. From grammar to lexicon: Unsupervised learning of lexical syntax. *Computational Linguistics*, 19(2):243–262, 1993.

[41] E. Brill and P. Resnik. A rule-based approach to prepositional phrase attachment disambiguation. In *Proc. of COLING'94*, pages 1198–1204, Kyoto, Japan, 1994.

[42] Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4):543–566, 1995.

[43] T. Briscoe and J. Carrol. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–59, 1993.

[44] T. Briscoe and J. Carrol. Automatic extraction of subcategorization from corpora. In *Proc. of the Fifth Conference on Applied Natural Language Processing*, pages 356–363, Washington, DC, 1997. ACL.

[45] J. S. Brown and R. R. Burton. Multiple representations of knowledge for tutorial reasoning. In D. G. Bobrow and A. Collins, editors, *Representation and Understanding*, pages 311–350. Academic Press, New York, NY, USA, 1975.

[46] Bob Carpenter. *The Logic of Typed Feature Structures*. Number 32 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge/New York/Melbourne, 1992.

[47] Vicente Carrillo, Víctor J. Díaz, and Miguel A. Alonso. Análisis sintáctico ascendente de TAGs guiado por la esquina izquierda. *Procesamiento del Lenguaje Natural*, 27:47–54, September 2001.

[48] John Carrol and David Weir. Encoding frequency information in lexicalized grammars. In *Proc. of the 5th International Workshop on Parsing Technologies (IWPT-97)*, Boston/Cambridge, MA, USA, 1997. ACL/SIGPARSE.

[49] John Chen, Srinivas Bangalore, and K. Vijay-Shanker. New models for improving supertag disambiguation. In *Proc. of EACL'99, Ninth Conference of the European Chapter of the Association for Computational Linguistics*, pages 188–195, Bergen, Norway, June 1999. ACL.

[50] John Chen and K. Vijay-Shanker. Automated extraction of TAGs from the Penn treebank. In *Proc. of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, pages 65–76, Trento, Italy, February 2000.

[51] Noam Chomsky. Three models for the description of language. *IRI Transactions on Information Theory*, 2(3):113–124, 1956.

[52] Noam Chomsky. *Syntactic Structures*. Mouton, The Hague, The Netherlands, 1957.

[53] K. W. Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proc. of the Second Conference on Applied Natural Language Processing*, pages 136–143. ACL, 1988.

[54] Fabio Ciravegna and Alberto Lavelli. Full text parsing using cascades of rules: an information extraction perspective. In *Proc. of EACL'99, Ninth Conference of the European Chapter of the Association for Computational Linguistics*, pages 102–109, Bergen, Norway, June 1999. ACL.

[55] Fabio Ciravegna, Alberto Lavelli, Nadia Mana, Johannes Matiasek, Luca Gilardoni, Silvia Mazza, Massimo Ferraro, William J. Black, Fabio Rinaldi, and David Mowatt. FACILE: Classifying texts integrating pattern matching and information extraction. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, 1999.

[56] N. Coccaro and D. Jurafsky. Towards better integration of semantic predictors in statistical language modeling. In *Proc. of ICSLP'98*, volume 6, pages 2403–2406, Sydney, Australia, 1998.

[57] Jacques Cohen. A view of the origins and development of Prolog. *Communications of the ACM*, 31(1):26–36, January 1988.

[58] A. Colmerauer. Les systèmes-q ou un formalisme pour analyser et synthétiser des phrases sur ordinateur. Internal Publication 43, Département d'Informatique de l'Université de Montréal, 1970.

[59] A. Colmerauer. Metamorphosis grammars. In L. Bolc, editor, *Natural Language Communication with Computers*, volume 63 of *Lecture Notes in Computer Science*, pages 133–189. Springer-Verlag, Berlin, 1978.

[60] Jan Daciuk, Stoyan Mihov, Bruce W. Watson, and Richard E: Watson. Incremental construction of minimal acyclic finite-state automata. *Computational Linguistics*, 26(1):3–16, 2000.

[61] Veronica Dahl. Natural language processing and logic programming. *Journal of Logic Programming*, 19,20:681–714, 1994.

[62] Eric de la Clergerie. *Automates à Piles et Programmation Dynamique. DyALog : Une Application à la Programmation en Logique.* PhD thesis, Université Paris 7, Paris, France, 1993.

[63] Eric de la Clergerie and Miguel A. Alonso. A tabular interpretation of a class of 2-Stack Automata. In *COLING-ACL'98, 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Proceedings of the Conference*, volume II, pages 1333–1339, Montreal, Quebec, Canada, August 1998. ACL.

[64] Eric de la Clergerie, Miguel A. Alonso, and David Cabrero Souto. A tabular interpretation of bottom-up automata for TAG. In *Proc. of Fourth International Workshop on Tree-Adjoining Grammars and Related Frameworks (TAG+4)*, pages 42–45, Philadelphia, PA, USA, August 1998.

[65] Eric de la Clergerie and Bernard Lang. LPDA: Another look at tabulation in logic programming. In Van Hentenryck, editor, *Proc. of the 11th International Conference on Logic Programming (ICLP'94)*, pages 470–486. MIT Press, June 1994.

[66] Peter J. Denning, Douglas E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R. Young. Computing as a discipline. *Communications of the ACM*, 32(1):9–23, January 1989.

[67] Gaël Dias, Sylvie Guilloré, Jean-Claude Bassano, and José Gabriel Pereira Lopes. Extraction automatique d'unités lexicales complexes: un enjeu fondamental pour la recherche documentaire. *Traitement automatique des langues*, 41(2):447–472, 2000.

[68] Víctor J. Díaz. *Gramáticas de Adjunción de Árboles: un Enfoque Deductivo en el Análisis Sintáctico*. PhD thesis, Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla, Seville, Spain, 2000.

[69] Víctor J. Díaz, Vicente Carrillo, and Miguel A. Alonso. A bidirectional bottom-up parser for TAG. In *Proc. of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, pages 299–300, Trento, Italy, February 2000.

[70] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

[71] Christiane Fellbaum. English verbs as a semantic net. *International Journal of Lexicography*, 3(4):278–301, 1990.

[72] Carlos G. Figuerola, Raquel Gómez, Angel F. Zazo Rodríguez, and José Luis Alonso Berrocal. Stemming in Spanish: A first approach to its impact on information retrieval. In Carol Peters, editor, *Working notes for the CLEF 2001 workshop*, Darmstadt, Germany, September 2001.

[73] William B. Frakes and Ricardo Baeza-Yates, editors. *Information Retrieval. Data Structures and Algorithms*. Prentice Hall, Upper Saddle River, New Jersey, 1992.

[74] Gerald Gazdar. Applicability of indexed grammars to natural languages. In U. Reyle and C. Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, pages 69–94. D. Reidel Publishing Company, 1987.

[75] A. Glennie. On the syntax machine and the construction of a universal compiler. Technical Report 2, Carnegie Mellon University, Pittsburgh, PA, 1960.

[76] Jorge Graña. *Técnicas de Análisis Sintáctico Robusto para la Etiquetación del Lenguaje Natural*. PhD thesis, Departamento de Computación, Universidade da Coruña, A Coruña, Spain, December 2000.

[77] Jorge Graña, Miguel A. Alonso, and Alberto Valderruten. Análisis léxico no determinista: Etiquetación eficiente del lenguaje natural. Technical Report 16, Departamento de Computación, Facultade de Informática, Universidade da Coruña, Campus de Elviña s/n, 15071 La Coruña, Spain, 1994.

[78] Jorge Graña, Miguel A. Alonso, and Manuel Vilares. A common solution for tokenization and part-of-speech tagging: One-pass Viterbi algorithm vs. iterative approaches. In Petr Sojka, Ivan Kopecek, and Karel Pala, editors, *Text, Speech and Dialogue*, volume 2448 of *Lecture Notes in Computer Science*, pages 3–10. Springer-Verlag, Berlin-Heidelberg-New York, 2002.

[79] Jorge Graña, Gloria Andrade, and Jesús Vilares. Compilation of constraint-based contextual rules for part-of-speech tagging into finite state transducers. In Jean-Marc Champarnaud and Denis Maurel, editors, *Seventh International Conference on Implementation and Application of Automata (CIAA 2002)*, pages 131–140, Tours, France, July 2002.

[80] Jorge Graña, Fco. Mario Barcala, and Miguel A. Alonso. Compilation methods of minimal acyclic automata for large dictionaries. In Bruce W. Watson and Derick Wood, editors, *Implementation and Application of Automata*, volume 2494 of *Lecture Notes in Computer Science*, pages 135–148. Springer-Verlag, Berlin-Heidelberg-New York, 2002.

[81] Jorge Graña, Jean-Cédric Chappelier, and Manuel Vilares. Integrating external dictionaries into stochastic part-of-speech taggers. In Galia Angelova, Kalina Bontcheva, Ruslan Mitkov, Nicolas Nocolov, and Nokolai Nikolov, editors, *EuroConference Recent Advances in Natural Language Processing. Proceedings*, pages 122–128, Tzigov Chark, Bulgaria, 2001.

[82] B. B. Greene and G. M. Rubin. Automatic grammatical tagging of English, 1971. Departament of Linguistics, Brown University, Providence, Rhode Island.

[83] Gregory Grefenstette, editor. *Cross-Language Information Retrieval*. The Kluwer International Series on Information Retrieval. Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1998.

[84] Gregory Grefenstette. The problem of cross-language information retrieval. In Gregory Grefenstette, editor, *Cross-Language Information Retrieval*, The

Kluwer International Series on Information Retrieval, chapter 1, pages 1–9. Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1998.

[85] Ralph Grishman. The NYU system for MUC-6 or where's the syntax? In *Proc. of the Sixth Message Understanding Conference*. Morgan Kaufmann Publishers, 1995.

[86] Derek Gross and Katherine J. Miller. Ajectives in WordNet. *International Journal of Lexicography*, 3(4):265–277, 1990.

[87] Ariane Halber. Tree-grammar linear typing for unified supertagging/probabilistic parsing models. In *Proc. of Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*, pages 54–57, Philadelphia, PA, USA, August 1998.

[88] J. Hankamer. Finite atate morphology and left to right phonology. In *Proceedings of the Fifth West Coast Conference on Formal Linguistics*, pages 29–34, 1986.

[89] Z. S. Harris. *String Analysis of Sentence Structure*. Mouton, The Hague, The Netherlands, 1962.

[90] Jerry R. Hobbs, Douglas Appelt, John Bear, David Israel, Megumi Kameyama, Mark Stickel, and Mabry Tyson. FASTUS: A cascaded finite-state transducer for extracting information from natural-language text. In Enmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*. MIT Press, Cambridge, MA, USA, 1997.

[91] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Series in Computer Science. Addison-Wesley Publishing Company, Reading, Massachussetts, USA, 1979.

[92] D. A. Huffman. The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, 3:161–191, 1954. Continued in Volume 4.

[93] Rebecca Hwa. An empirical evaluation of probabilistic lexicalized tree insertion grammars. In *COLING-ACL'98, 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Proceedings of the Conference*, volume I, pages 557–563, Montreal, Quebec, Canada, August 1998. ACL.

[94] Nancy Ide and Jean Véronis. Word sense disambiguation: The state of the art. *Computational Linguistics*, 24(1):1–40, 1998.

[95] E. T. Irons. A syntax directed compiler for ALGOL 60. *Communications of the ACM*, 4:51–55, 1961.

[96] Christian Jacquemin and Evelyne Tzoukermann. NLP for term variant extraction: synergy between morphology, lexicon and syntax. In Tomek Strzalkowski, editor, *Natural Language Information Retrieval*, volume 7 of *Text, Speech and Language Technology*, pages 25–74. Kluwer Academic Publishers, Dordrecht/Boston/London, 1999.

[97] F. Jelinek. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–557, 1976.

[98] F. Jelinek and J. D. Lafferty. Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, 17(3):315–323, 1991.

[99] Yufeng Jing and W. Bruce Croft. An association thesaurus for information retrieval. In *Proc. of RIAO'94*, pages 146–160, New York, USA, 1994.

[100] C. D. Johnson. *Formal Aspects of Phonological Description*. Number 3 in Monographs on Linguistic Analysis. Mouton, The Hague, The Netherlands, 1972.

[101] Mark Johnson. The computational complexity of GLR parsing. In Masaru Tomita, editor, *Generalized LR Parsing*, chapter 3, pages 35–42. Kluwer Academic Publishers, Boston/Dordrecht/London, 1991.

[102] A. K. Joshi and P. Hopely. A parser from antiquity. In A. Kornai, editor, *Extended Finite State Models of Language*, pages 6–15. Cambridge University Press, Camdridge, UK, 1999.

[103] Aravind K. Joshi. Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In David R. Dowty, Lauri Karttunen, and Arnold M. Zwicky, editors, *Natural Language Parsing. Psychological, Computational and Theoretical Perspectives*, chapter 6, pages 206–250. Cambridge University Press, 1985.

[104] Aravind K. Joshi. An introduction to tree adjoining grammars. In Alexis Manaster-Ramer, editor, *Mathematics of Language*, pages 87–115. John Benjamins Publishing Co., Amsterdam/Philadelphia, 1987.

[105] Aravind K. Joshi, Leon S. Levy, and M. Takahashi. Tree adjunt grammars. *Journal of Computer and System Sciences*, 10(1):136–162, February 1975.

[106] Aravind K. Joshi and Yves Schabes. Tree-adjoining grammars. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages. Vol 3: Beyond Words*, chapter 2, pages 69–123. Springer-Verlag, Berlin/Heidelberg/New York, 1997.

[107] Aravind K. Joshi, K. Vijay-Shanker, and David Weir. The convergence of mildly context-sensitive grammar formalisms. In P. Sells, Shieber S. M., and T. Warsow, editors, *Foundational Issues in Natural Language Processing*, pages 31–81. MIT Press, Cambridge, MA, USA, 1991.

[108] Megumi Kameyama. Recognizing referential links: An information extraction perspective. In Mitkov and Bugaraev, editors, *Proc. of ACL/EACL Wokshop on Operational Factors in Practical, Robust Anaphora Resolution for Unrestricted texts*, pages 46–53, Madrid, Spain, 1997.

[109] Roland M. Kaplan and Martin Key. Phonological rules and finite-state transducers. In *Annual Meeting of the Linguistics Society of America*, New York, 1981.

[110] Roland M. Kaplan and Martin Key. Regular models of phonological rules systems. *Computational Linguistics*, 20(3):331–378, 1994.

[111] Ronald M. Kaplan. The formal architecture of lexical-functional grammar. In Mary Darymple, Ronald M. Kaplan, John T. Maxwell III, and Annie Zaenen, editors, *Formal Issues in Lexical-Functional Grammar*. Stanford University, 1994.

[112] F. Karlsson, A. Voutilainen, J. Heikkilä, and A. Anttila. *Constraint Grammar: a Language-Independent System for Parsing Unrestricted Text*. Mouton de Gruyter, Berlin, 1995.

[113] L. Karttunen. KIMMO: A general morphological processor. *Texas Linguistics Forum*, 22:165–186, 1983.

[114] L. Karttunen. Comments on Joshi. In A. Kornai, editor, *Extended Finite State Models of Language*, pages 16–18. Cambridge University Press, Cambridge, UK, 1999.

[115] T. Kasami. An efficient recognition and syntax algorithm for context-free languages. Scientific Report AFCRL-65-758, Air Force Cambridge Research Lab., Bedford, Massachussetts, 1965.

[116] S. M. Katz and J. A. Fodor. The structure of a semantic theory. *Language*, 39:170–200, 1963.

[117] Martin Kay. Functional grammar. In *BLS'79*, pages 142–158, Berkeley, CA, USA, 1979.

[118] Adam Kilgarriff. Foreground and background lexicons and word sense disambiguation for information extraction. In *Proc. of the International Wokshop on Lexicon Driven Information Extraction*, Frascati, Italy, 1997.

[119] Alexandra Kinyon. Un algorithme d'analyse LR(0) pour les Grammaires d'Arbres Adjoints Lexicalisées. In D. Genthial, editor, *Actes de la quatrième conférence annuelle sur Le Traitement Automatique du Langage Naturel*, pages 93–102, Grenoble, France, June 1997.

[120] S. C Kleene. Representation of events in nerve nets and finite automata. In C. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, Princeton, NJ, 1956. Also available as Technical Report RM-704, RAND Corporation, 1951.

[121] S. Klein and R. F. Simmons. A computational approach to grammatical coding of English words. *Journal of the Association for Computing Machinery*, 10(3):334–347, 1963.

[122] Donald E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968.

[123] K. Koskenniemi and K. W. Church. Complexity, two-level morphology and Finnish. In *Proc. of COLING'88*, pages 335–339, Budapest, Hungary, 1988.

[124] Wessel Kraaij and Renée Pohlmann. Using linguistic knowledge in information retrieval. OTS Working Paper OTS-WP-CL-96-001, Research Institute for Language and Speech (OTS), Utrecht University, Utrecht, The Netherlands, 1996.

[125] Wessel Kraaij and Renée Pohlmann. Viewing stemming as recall enhancement. In Hans-Peter Frei, Donna Harman, Peter Schauble, and Ross Wilkinson, editors, *Proc. of the 19th annual international ACM SIGIR conference on Research and Development in Information Retrieval*, pages 40–48, Zurich, Switzerland, 1996.

[126] Wessel Kraaij and Renée Pohlmann. Comparing the effect of syntactic vs. statistical phrase indexing strategies for Dutch. In Christos Nicolaou and Constantine Stephanidis, editors, *Research and Advanced Technology for Digital Libraries*, volume 1513 of *Lecture Notes in Computer Science*, pages 605–614. Springer-Verlag, Berlin/Heidelberg/New York, 1998.

[127] Wessel Kraaij, Renée Pohlmann, and Djoerd Hiemstra. Twenty-One at TREC-8: using language technology for information retrieval. In E. M. Voorhees and D. K. Harman, editors, *NIST Special Publication 500-246: The Eighth Text REtrieval Conference (TREC 8)*, pages 285–299, Gaithersburg, MD, USA, 2000. Department of Commerce, National Institute of Standards and Technology.

[128] R. Kuhn and R. De Mori. A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):570–583, 1990.

[129] S. Kuno. The predictive analyzer and a path elimination technique. *Communications of the ACM*, 8(7):453–462, 1965.

[130] S. Kuno and A. G. Oettinger. Multiple-path syntactic analyzer. In C. M. Poplewell, editor, *Information processing 1962: Proceedings of the IFIP Congress 1962*, pages 306–312, Munich, 1963. Noth-Holland.

[131] Byung-Kwan Kwak, Jee-Hyub Kim, Geunbae Lee, and Jung Yun Seo. Corpus-based learning of compound noun indexing. In J. Klavans and J. Gonzalo, editors, *Proc. of the ACL'2000 workshop on Recent Advances in Natural Language Processing and Information Retrieval*, Hong Kong, October 2000.

[132] G. Lakoff. Linguistics and natural logic. In D. Davidson and G. Harman, editors, *Semantics for Natural Language*, pages 545–665. D. Reidel, Dordrecht, 1972.

[133] Bernard Lang. Deterministic techniques for efficient non-deterministic parsers. In *Proc. of 2nd Colloquium on Automata, Languages and Programming (ICALP'74), Saarbrüecken, Germany*, volume 14 of *Lecture Notes in Computer Science*, pages 255–269. Springer Verlag, Berlin-Heidelberg-New York, 1974.

[134] Bernard Lang. The systematic construction of Earley parsers: Application to the production of $\mathcal{O}(n^6)$ Earley parsers for tree adjoining grammars. In *Proc. of the 1st International Workshop on Tree Adjoining Grammars*, August 1990.

[135] Bernard Lang. Towards a uniform formal framework for parsing. In Masaru Tomita, editor, *Current Issues in Parsing Technology*, pages 153–171. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[136] Alberto Lavelli and Giorgio Satta. Bidirectional parsing of lexicalized tree adjoining grammars. In *Proceedings of the 5th Conference of the European Chapter of the Association for Computational Linguistics (EACL'91)*, pages 27–32, Berlin, Germany, April 1991. ACL.

[137] René Leermakers. A recursive ascent Earley parser. *Information Processing Letters*, 41:87–91, February 1992.

[138] René Leermakers. *The Functional Treatment of Parsing.* The Kluwer International Series in Engineering and Computer Science. Natural Language Processing and Machine Translation. Kluwer Academic Publishers, Boston/Dordrecht/London, 1993.

[139] René Leermakers. Recursive ascent parsing: from Earley to Marcus. *Theoretical Computer Science*, 104:299–312, 1996.

[140] David D. Lewis and Karen Sparck-Jones. Natural language processing for information retrieval. *Communications of the ACM*, 39(1):92–101, 1996.

[141] Patrice Lopez. Connection driven parsing of lexicalized TAG. In *Proc. of International Workhop on Text, Speech and Dialog (TSD'98)*, Brno, Czech Republic, September 1998.

[142] Patrice Lopez. *Analyse d'énoncés oraux pour le Dialogue Homme-Machine à l'aide de Grammaires Lexicalisées d'Arbres*. PhD thesis, Université Henri Poincaré – Nancy 1, Nancy, France, October 1999.

[143] C. L. Lucchesi and T. Kowaltowski. Applications of finite automata representing large vocabularies. *Software Practice and Experience*, 23(1):15–30, 1993.

[144] C. D. Manning. Automatic acquisition of a large subcategorization dictionary from corpora. In *Proc. of ACL'93*, pages 235–242, Columbus, OH, 1993.

[145] Solomon Marcus, Carlos Martín-Vide, and Gheorghe Păun. Contextual Grammars as generative models of natural languages. *Computational Linguistics*, 24(2):245–274, June 1998.

[146] A. A. Markov. Essai d'une recherche statistique sur le texte du roman "Eugene Onegin" illustrant la liaison des epreuve en chain. *Izvistia Imperatorskoi Akademi Nauk (Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg)*, 7:153–162, 1913.

[147] Lluís Màrquez and Horacio Rodríguez. Automatically acquiring a language model for POS tagging using decision trees. In Rusland Mitkov, Nicolas Nicolov, and Nikolai Nokolov, editors, *International Conference on Recent Advances in Natural Language Processing. Proceedings*, pages 27–34, Tzigov Chark, Bulgaria, September 1997.

[148] I. Marshall. Choice of grammatical word-class without global syntactic analysis: Tagging words in the LOB corpus. *Computer and the Humanities*, 17:139–150, 1983.

[149] M. Masterman. The thesaurus in syntax and semantics. *Machine Translation*, 4(1):1–2, 1957.

[150] W. S. McCulloch and W. Pitts. A logical calculus of ideas inmanent in nervous activity. *Bulletin of Mathematical Biophysics*, pages 115–133, 1943. Reprinted in *Neurocomputing: Foundations of Research*, edited by J. A. Anderson and E. Rosenfeld. MIT Press, 1988.

[151] G. H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955.

[152] I. A. Mel'čuk. *Studies in Dependency Syntax*. Karoma Publishers, Ann Arbor, 1979.

[153] George A. Miller. Nouns in Wordnet: A lexical inheritance system. *International Journal of Lexicography*, 3(4):245–264, 1990.

[154] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J. Miller. Introduction to WordNet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–244, 1990.

[155] Ruslan Mitkov, Branimir Boguraev, and Shalom Lappin. Introduction to the special issue on computational anaphora resolution. *Computational Linguistics*, 27(4):473–477, December 2001.

[156] Markus Mittendorfer and Werner Winiwarter. A simple way of improving traditional IR methods by structuring queries. In *Proc. of the 2001 IEEE International Workshop on Natural Language Processing and Knowledge Engineering*, Tucson, Arizona, October 2001.

[157] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–312, June 1997.

[158] Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. A rational design for a weighted finite-state transucers library. In *Proc. of Second International Workshop on Implementing Automata (WIA'97)*, pages 43–53, London, Ontario, Canada, September 1997.

[159] Richard Montague. The proper treatment of quantification in ordinary English. In R. Thomason, editor, *Formal Philosophy: Selected Papers of Richard Montague*, pages 247–270. Yale University Press, New Haven, CT, USA, 1973.

[160] E. F. Moore. Gedanken-experiments on sequential machines. In C. Shannon and J. McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, Princeton, NJ, 1956.

[161] P. Naur, J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Winjnagaarden, and M. Woodger. Report on the algorithmic language ALGOL 60. *Communicactions of the ACM*, 3(5), 1960. Revised in CACM 6(1):1–17, 1963.

[162] Mark-Jan Nederhof. Solving the correct-prefix property for TAGs. In T. Becker and H.-V. Krieger, editors, *Proc. of the Fifth Meeting on Mathematics of Language*, pages 124–130, Schloss Dagstuhl, Saarbruecken, Germany, August 1997.

[163] Mark-Jan Nederhof. An alternative LR algorithm for TAGs. In *COLING-ACL'98, 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Proceedings of the Conference*, volume II, pages 946–952, Montreal, Quebec, Canada, August 1998. ACL.

[164] Mark-Jan Nederhof. Linear indexed automata and tabulation of TAG parsing. In *Proc. of First Workshop on Tabulation in Parsing and Deduction (TAPD'98)*, pages 1–9, Paris, France, April 1998.

[165] Mark-Jan Nederhof. The computational complexity of the correct-prefix property for TAGs. *Computational Linguistics*, 25(3):345–360, 1999.

[166] Mark-Jan Nederhof. Models of tabulation for TAG parsing. In *Proc. of the Sixth Meeting on Mathematics of Language (MOL 6)*, pages 143–158, Orlando, Florida, USA, July 1999.

[167] Mark-Jan Nederhof, Anoop Sarkar, and Giorgio Satta. Prefix probabilistic from stochastic tree adjoining grammars. In *COLING-ACL'98, 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Proceedings of the Conference*, volume II, pages 953–959, Montreal, Quebec, Canada, August 1998. ACL.

[168] Carol Neidle. Lexical functional grammars. In *Encyclopaedia of Language and Linguistics*. Pergamon Press, New York, NY, USA, 1994.

[169] Günter Neumann. Automatic extraction of stochastic lexicalized tree grammars from treebaks. In *Proc. of Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*, pages 120–123, Philadelphia, PA, USA, August 1998.

[170] H. Ney, U. Essen, and R. Kneser. On structuring probabilistic dependencies in stochastic language modelling. *Computer Speech and Language*, 8:1–38, 1994.

[171] P. Norvig. Techniques for automatic memorization with applications to context-free parsing. *Computational Linguistics*, 17(1):91–98, 1991.

[172] The Joint Task Force on Computing Curricula. Computing Curricula 2001 Computer Science — Final Report —, December 2001. IEEE Computer Society and Association for Computing Machinery.

[173] D. W. Packard. Computer assited morphological analysis of ancient Greek. In A. Zampolli and N. Calzonari, editors, *Computational and Mathematical Linguistics: Proceedings fo the International Conference on Computational Linguistics*, pages 343–355, Pisa, Italy, 1973.

[174] L. Padró. POS tagging using relaxation labelling. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING'96)*, Copenhagen, Denmark, 1996.

[175] Manuel Palomar, Antonio Ferrández, Lidia Moreno, Patricio Martínez-Barco, Jesús Peral, Maximiliano Saiz-Noeda, and Rafael Muñoz. An algorithm for anaphora resolution in Spanish texts. *Computational Linguistics*, 27(4):545–567, December 2001.

[176] Ted Pedersen. Assessing system agreement and instance difficulty in the lexical sample tasks of SENSEVAL-2. In *Proc. of the Workshop on Word Sense Disambiguation: Recent Successes and Future Directions*, Philadelphia, PA, USA, July 2002.

[177] Fernando C. N. Pereira and Stuart M. Shieber. *Prolog and Natural Language Analysis*. Number 10 in CSLI Lecture Notes. Center for the Study of Language and Information, Stanford, CA, USA, 1987.

[178] Fernando C. N. Pereira and David H. D. Warren. Definite Clause Grammars for language analysis — a survey of the formalism and a comparison with Augmented Transition Networks. *Artificial Intelligence*, 13:231–278, 1980.

[179] Jose Perez-Carballo and Tomek Strzalkowski. Natural language information retrieval: progress report. *Information Processing and Management*, 36(1):155–178, 2000.

[180] Carol Peters and Páraic Sheridan. Multilingual information access. In M Agosti, F. Crestani, and G. Pasi, editors, *Lectures on Information Retrieval*, volume 1980 of *Lecture Notes in Computer Science*, pages 51–80. Springer-Verlag, Berlin-Heidelberg-New York, 2000.

[181] Ulrich Pfeifer, Thomas Poersch, and Norbert Fuhr. Retrieval effectiveness of proper name search methods. *Information Processing and Management*, 32(6):667–679, 1996.

[182] Gisela Pitsch. $LL(k)$ parsing of coupled-context-free grammars. *Computational Intelligence*, 10(4):563–578, 1994.

[183] Renée Pohlmann and Wessel Kraaij. The effect of syntactic phrase indexing on retrieval performance for Dutch texts. In L. Devroye and C. Chrisment, editors, *Proc. of Computer-Aided Information Searching on the Internet (RIAO'97)*, pages 176–187, Montreal, Canada, 1997.

[184] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. The University of Chicago Press, Chicago & London, 1994.

[185] J. Porta. Rtag. Technical report, Grupo de Investigación de Lingüística Computacional, Universidad de Barcelona, Bercelona, Spain, 1996.

[186] Carlos A. Prolo. An efficient LR parser generator for tree adjoining grammars. In *Proc. of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, pages 207–218, Trento, Italy, February 2000.

[187] M. R. Quillian. Semantic memory. In M. Minsky, editor, *Semantic Information Processing*, pages 227–270. MIT Press, Cambridge, MA, USA, 1968.

[188] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

[189] A. Ratnaparkhi, J. Reynar, and S. Roukos. A maximum entropy model for prepositional phrase attachment. In *ARPA Human Language Technologies Worskhop*, pages 250–255, Plainsboro, NJ, 1994.

[190] Jan Rekers. *Parsing Generation for Interactive Environments.* PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, 1992.

[191] Philip Resnik. Probabilistic tree-adjoining grammar as a framework for natural language processing. In *Proc. of Fifteenth International Conference on Computational Linguistics (COLING'92)*, pages 418–424, Nantes, France, August 1992.

[192] Philip Resnik and David Yarowsky. A perspective on word sense disambiguation methods and their evaluation. In *Proc. of SIGLEX'97*, pages 79–86, Washington, DC, USA, 1997.

[193] Kelly Roach. Formal properties of Head Grammars. In Alexis Manaster-Ramer, editor, *Mathematics of Language*, pages 293–347. John Benjamins Publishing Company, Amsterdam/Philadephia, 1987.

[194] Eric Roberts, Russ Shackelford, Rich LeBlanc, and Peter J. Denning. Curriculum 2001: Interim report from the ACM/IEEE-CS task force. In *The proceedings of the thirtieth SIGCSE technical symposium on Computer Science Education*, pages 343–344, New York, 1999. ACM Press.

[195] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12:23–41, 1965.

[196] Enmanuel Roche and Yves Schabes, editors. *Finite-State Devices for Natural Language Processing.* MIT Press, Cambridge, MA, 1997.

[197] D. Roland and D. Jurafsky. How verb subcategorization frequencies are affected by corpus choice. In *COLING-ACL'98, 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Proceedings of the Conference*, pages 1122–1128, Montreal, Quebec, Canada, August 1998. ACL.

[198] R. Rosenfeld. A maximum entropy approach to adaptive statistical language modelling. *Computer Speech and Language*, 10:187–228, 1996.

[199] A. Salomaa. Probabilistic and weighted grammars. *Information and Control*, 15:529–544, 1969.

[200] Anoop Sarkar. Conditions on consistency of probabilistic tree adjoining grammars. In *COLING-ACL'98, 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Proceedings of the Conference*, volume II, pages 1164–1170, Montreal, Quebec, Canada, August 1998. ACL.

[201] Yves Schabes. The valid prefix property and left to right parsing of tree-adjoining grammar. In *Proc. of II International Workshop on Parsing Technologies, IWPT'91*, pages 21–30, Cancún, Mexico, 1991.

[202] Yves Schabes. Stochastic lexicalized tree-adjoining grammars. In *Proc. of Fifteenth International Conference on Computational Linguistics (COLING'92)*, pages 426–432, Nantes, France, August 1992.

[203] Yves Schabes and Aravind K. Joshi. An Earley-type parsing algorithm for tree adjoining grammars. In *Proc. of 26th Annual Meeting of the Association for Computational Linguistics*, pages 258–269, Buffalo, NY, USA, June 1988. ACL.

[204] Yves Schabes and Aravind K. Joshi. Parsing with lexicalized tree adjoining grammar. In Masaru Tomita, editor, *Current Issues in Parsing Technologies*, chapter 3, pages 25–47. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[205] Yves Schabes and K. Vijay-Shanker. Deterministic left to right parsing of tree adjoining languages. In *Proc. of 28th Annual Meeting of the Association for Computational Linguistics*, pages 276–283, Pittsburgh, Pennsylvania, USA, June 1990. ACL.

[206] Yves Schabes and Richard C. Waters. Stochastic lexicalized context-free grammar. In *Proc. of the Third International Workshop on Parsing Technologies (IWPT'93)*, pages 257–266, Tilburg (The Netherlands) and Durbuy (Belgium), August 1993. Also as Technical Report TR-93-12, July 1993, Mitsubishi Electric Research Laboratories, Cambridge, MA, USA.

[207] Yves Schabes and Richard C. Waters. Tree insertion grammar: A cubic-time parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Computational Linguistics*, 21(4):479–513, December 1995. Also as Technical Report TR-94-13, June 1994, Mitsubishi Electric Research Laboratories, Cambridge, MA, USA.

[208] Yves Schabes and Richard C. Waters. Stochastic lexicalized tree-insertion grammar. In Harry Bunt and Masaru Tomita, editors, *Recent Advances in Parsing Technology*, volume 1 of *Text, Speech and Language Technology*, chapter 15, pages 281–294. Kluwer Academic Publishers, Dordrecht/Boston/London, 1996.

[209] C. E. Shannon. A symbolic analysis of relay and switching circuits. *Transactions of the American Institute of Electrical Engineers*, 57:713–723, 1938.

[210] C. E. Shannon. Prediction and entropy of printed English. *Bell System Technical Journal*, 30:50–64, 1951.

[211] B. A. Sheil. Observations on context-free grammars. *SMIL: Statistical Methods in Linguistics*, 1:71–109, 1976.

[212] Stuart M. Shieber. Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *Proc. of the 23th Annual Meeting of the Association for Computational Linguistics*, pages 145–152. ACL, June 1985.

[213] Stuart M. Shieber. *Constraint Based Grammar Formalisms*. MIT Press, Cambridge, MA, USA, 1992.

[214] Klaas Sikkel. *Parsing Schemata — A Framework for Specification and Analysis of Parsing Algorithms*. Texts in Theoretical Computer Science — An EATCS Series. Springer-Verlag, Berlin/Heidelberg/New York, 1997.

[215] R. F. Simmons. Answering English questions by computer. *Communications of the ACM*, 8(1):53–70, 1965.

[216] Karen Sparck-Jones. What is the role of NLP in text retrieval? In Tomek Strzalkowski, editor, *Natural Language Information Retrieval*, volume 7 of *Text, Speech and Language Technology*, pages 1–24. Kluwer Academic Publishers, Dordrecht/Boston/London, 1999.

[217] M. Steedman. Combinators and grammars. In R. Oehrle, E. Bach, and D Wheeler, editors, *Categorial Grammars and Natural Language Structures*, pages 417–442. Foris, Dordrecht, 1986.

[218] J. Stetina and M. Nagao. Corpus based PP attachment ambiguity resolution with a semantic dictionary. In J. Zhou and K. W. Church, editors, *Proceedings of the Fifth Workshop on Very large Corpora*, pages 66–80, Beijing, China, 1997.

[219] A. Stolcke. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–202, 1995.

[220] Tomek Strzalkowski, Louise Guthrie, Jussi Karlgren, Jim Leistensnider, Fang Lin, Jose Perez-Carballo, Troy Straszheim, Jin Wang, and Jon Wilding. Natural language information retrieval: TREC-5 report. In E. M. Voorhees and D. K. Harman, editors, *NIST Special Publication 500-238: The Fifth Text REtrieval Conference (TREC 5)*, pages 291–313, Gaithersburg, MD, USA, 1997. Department of Commerce, National Institute of Standards and Technology.

[221] Tomek Strzalkowski, Jose Perez-Carballo, Jussi Karlgren, Anette Hulth, Pasi Tapanainen, and Timo Lahtinen. Natural language information retrieval: TREC-8 report. In E. M. Voorhees and D. K. Harman, editors, *NIST Special Publication 500-246: The Eighth Text REtrieval Conference (TREC 8)*, pages 381–390, Gaithersburg, MD, USA, 2000. Department of Commerce, National Institute of Standards and Technology.

[222] Tomek Strzalkowski, Gees Stein, G. Bowden Wise, Jose Perez-Carballo, Pasi Tapanainen, Timo Jarvinen, Atro Voutilainen, and Jussi Karlgren. Natural language information retrieval: TREC-7 report. In E. M. Voorhees and D. K. Harman, editors, *NIST Special Publication 500-242: The Seventh Text REtrieval Conference (TREC 7)*, pages 217–226, Gaithersburg, MD, USA, 1999. Department of Commerce, National Institute of Standards and Technology.

[223] The CS2008 Review Taskforce. Computer Science Curriculum 2008: An interim revision of CS 2001, December 2008. IEEE Computer Society and Association for Computing Machinery.

[224] Frédéric Tendeau. *Analyse syntaxique et sémantique avec évaluation d'attributs dans un demi-anneau. Applications à la linguistique calculatoire.* PhD thesis, Université d'Orléans, France, June 1997.

[225] Paul Thompson and Christopher C. Dozier. Name recognition and retrieval performance. In Tomek Strzalkowski, editor, *Natural Language Information Retrieval*, volume 7 of *Text, Speech and Language Technology*, pages 261–272. Kluwer Academic Publishers, Dordrecht/Boston/London, 1999.

[226] Masaru Tomita. *Efficient Parsing for Natural Language. A Fast Algorithm for Practical Systems.* The Kluwer International Series in Engineering and Computer Science. Natural Language Processing and Machine Translation. Kluwer Academic Publishers, Boston/Dordrecht/Lancaster, 1986.

[227] J. L. Triviño Rodríguez and R. Morales Bueno. A Spanish POS tagger with variable memory. In *Proc. of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, pages 254–265, Trento, Italy, February 2000.

[228] Allen B. Tucker, Barnes Bruce H., Robert M. Aiken, Keith Barker, Kim B. Bruce, J. Thomas Cain, Susan E. Conry, Gerald L. Engel, Richard G. Epstein, Doris K. Lidtke, Michael C. Mulder, Jean B. Rogers, Eugene H. Spafford, and A. Joe Turner. Computing Curricula 1991. Report of the ACM/IEEE-CS joint curriculum task force, December 1990.

[229] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1937. Read to the Society in 1936, but published in 1937. Correction in volume 43:544–546.

[230] A. Joe Turner. A summary of the ACM/IEEE-CS joint curriculum task force report. Computing Curricula 1991. *Communications of the ACM*, 34(6):69–84, June 1991.

[231] H. Uszkoreit. Categorial unification grammar. In *Proc. of COLING'86*, pages 187–194, Bonn, Germany, 1986.

[232] Gertjan van Noord. Head-corner parsing for TAG. *Computational Intelligence*, 10(4):525–534, 1994.

[233] C. J. van Rijsbergen. Getting into information retrieval. In M Agosti, F. Crestani, and G. Pasi, editors, *Lectures on Information Retrieval*, volume 1980 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag, Berlin-Heidelberg-New York, 2000.

[234] K. Vijay-Shanker. *A Study of Tree Adjoining Grammars*. PhD thesis, University of Pennsylvania, January 1988. Available as Technical Report MS-CIS-88-03 LINC LAB 95 of the Department of Computer and Information Science, University of Pennsylvania.

[235] K. Vijay-Shanker and Aravind K. Joshi. Some computational properties of tree adjoining grammars. In *23rd Annual Meeting of the Association for Computational Linguistics*, pages 82–93, Chicago, IL, USA, July 1985. ACL.

[236] K. Vijay-Shanker and Aravind K. Joshi. Feature structures based tree adjoining grammars. In Dénes Vargha, editor, *Proc. of the 12nd International Conference on Computational Linguistics (COLING'88)*, pages 714–719, Budapest, Hungary, 1988.

[237] K. Vijay-Shanker and Aravind K. Joshi. Unification-based tree adjoining grammars. In J. Wedekind, editor, *Unification Based Grammars*. MIT Press, Cambridge, MA, USA, 1991.

[238] K. Vijay-Shanker and David J. Weir. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27:511–545, 1994.

[239] K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. Characterizing structural descriptions produced by various grammatical formalisms. In *Proc. of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 104–111, Buffalo, NY, USA, June 1987. ACL.

[240] K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. On the progression from context-free to tree adjoining languages. In Alexis Manaster-Ramer, editor, *Mathematics of Language*, pages 389–401. John Benjamins Publishing Company, Amsterdam/Philadelphia, 1987.

[241] Jesús Vilares, Miguel A. Alonso, Francisco J. Ribadas, and Manuel Vilares. COLE experiments at CLEF 2002 Spanish monolingual track. In Carol Peters, editor, *Results of the CLEF 2002 Cross-Language System Evaluation Campaign, Working Notes for the CLEF 2002 Workshop*, pages 153–160, Rome, Italy, September 2002.

[242] Jesús Vilares, Fco. Mario Barcala, and Miguel A. Alonso. Using syntactic dependency-pairs conflation to improve retrieval performance in Spanish. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volume 2276 of *Lecture Notes in Computer Science*, pages 381–390. Springer-Verlag, Berlin-Heidelberg-New York, 2002.

[243] Jesús Vilares, David Cabrero, and Miguel A. Alonso. Applying productive derivational morphology to term indexing of Spanish texts. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volume 2004 of *Lecture Notes in Computer Science*, pages 336–348. Springer-Verlag, Berlin-Heidelberg-New York, 2001.

[244] Jesús Vilares, Manuel Vilares, and Miguel A. Alonso. Towards the development of heuristics for automatic query expansion. In Heinrich C. Mayr, Jiri Lazansky, Gerald Quirchmayr, and Pavel Vogel, editors, *Database and Expert Systems Applications*, volume 2113 of *Lecture Notes in Computer Science*, pages 887–896. Springer-Verlag, Berlin-Heidelberg-New York, 2001.

[245] Manuel Vilares. *Efficient Incremental Parsing for Context-Free Languages*. PhD thesis, Université de Nice, France, 1992.

[246] Manuel Vilares and Miguel A. Alonso. An LALR extension for DCGs in dynamic programming. In Carlos Martín Vide, editor, *Mathematical and Computational Analysis of Natural Language*, volume 45 of *Studies in Functional and Structural Linguistics*, pages 267–278. John Benjamins Publishing Company, Amsterdam & Philadelphia, 1998.

[247] Manuel Vilares, Miguel A. Alonso, and Víctor M. Darriba. Generation of incremental parsers. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volume 2588 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin/Heidelberg/New York, 2003.

[248] Manuel Vilares, Miguel A. Alonso, and Alberto Valderruten. *Programación Lógica*. Editorial Tórculo, Santiago de Compostela, Spain, 2nd edition, 1996.

[249] Manuel Vilares and Bernard A. Dion. Efficient incremental parsing for context-free languages. In *Proc. of the* 5^{th} *IEEE International Conference on Computer Languages*, pages 241–252, Toulouse, France, 1994.

[250] Manuel Vilares, Jorge Graña, and Pilar Alvariño. Finite-state morphology and formal verification. *Journal of Natural Language Engineering, special issue on Extended Finite State Models of Language*, 3(4):303–304, 1997.

[251] Manuel Vilares, Jorge Graña, and Pilar Alvariño. Finite-state morphology and formal verification. In András Kornai, editor, *Extended Finite State Models of Language*. Cambridge University Press, 1997.

[252] Piek Vossen, editor. *EuroWordNet. A Multilingual Database with Lexical Semantic Networks*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998. Reprinted from Computers and the Humanities, Volume 32, Nos. 2–3, 1998.

[253] A. Voutilainen and J. Heikkilä. An English constraint grammar (EngCG): a surface-syntactic parser of English. In Fries, Tottie, and Schneider, editors, *Creating and using English language corpora*. Rodopi, 1994.

[254] D. J. Weber, H. A. Black, and S. R. McConnel. AMPLE: A tool for exploring morphology. Occasional Publications in Academic Computing 12, Summer Institute of Linguistics, Dallas, TX, 1988.

[255] David J. Weir. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, University of Pennsylvania, 1988. Available as Technical Report MS-CIS-88-74 of the Department of Computer and Information Sciences, University of Pennsylvania.

[256] W. A. Woods. *Semantics for a Question-Answering System*. PhD thesis, Harvard University, 1967.

[257] V. H. Yngve. Syntax and the problem of multiple meaning. In W. N. Locke and A. D. Booth, editors, *Machine Translation of Languages*, pages 208–226. MIT Press, Cambridge, MA, 1955.

[258] D. H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208, 1967.

[259] J. Zavrel and W. Daelemans. Memory-based learning: Using similarity for smoothing. In *Proc. of ACL/EACL'97*, pages 436–443, Madrid, Spain, 1997. ACL.

[260] K. Zechner and A. Waibel. Using chunk based partial parsing of spontaneous speech in unrestricted domains for reducing word error rate in speech recognition. In *COLING-ACL'98, 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Proceedings of the Conference*, pages 1453–1459, Montreal, Quebec, Canada, August 1998. ACL.

Índice alfabético