UNIVERSIDAD DE A CORUÑA
Departamento de Computación

Proyecto Fin de Carrera en Ingeniería Informática

Extensiones del Formalismo HMM para la Etiquetación del Lenguaje Natural

Autor:

GLORIA ANDRADE SANCHEZ

Director:

JORGE GRAÑA GIL

Enero de 2002

A mi fami

Dr. Jorge Graña Gil
Profesor Titular de Universidad Interino
Departamento de Computación
Universidad de A Coruña

CERTIFICA:

Que la memoria titulada "*Extensiones del Formalismo HMM para la Etiquetación del Lengua*
Natural" ha sido realizada por Dña. Gloria Andrade Sánchez bajo mi dirección en
Departamento de Computación de la Universidad de A Coruña, y concluye el Proyecto F
de Carrera que presenta para optar al título de Ingeniero en Informática.

A Coruña, 24 de enero de 2002

Fdo.: Jorge Graña Gil

Autor: Gloria Andrade Sánchez

Director: Jorge Graña Gil

Fecha:

Tribunal

Presidente:

Vocal 1°:

Vocal 2°:

Vocal 3°:

Secretario:

Resumen

Extensiones del Formalismo HMM para la Etiquetación de Textos en Lenguaje Natural

El objetivo último que persigue el Procesamiento del Lenguaje Natural es el perfecto análi y entendimiento de los lenguajes humanos. Actualmente, estamos todavía lejos de conseg este objetivo. Por esta razón, la mayoría de los esfuerzos de investigación de la lingüísti computacional han sido dirigidos hacia tareas intermedias que dan sentido a alguna de l múltiples características estructurales inherentes a los lenguajes, sin requerir un entendimien completo. Una de esas tareas es la asignación de categorías gramaticales a cada una de l palabras del texto. Este proceso se denomina también etiquetación.

La eliminación de ambigüedades es una tarea crucial durante el proceso de etiquetación un texto en lenguaje natural. Si tomamos aisladamente, por ejemplo, la palabra `sobre`, vem que puede tener varias categorías posibles en español: sustantivo, preposición o verbo. S embargo, si examinamos el contexto en el que aparece dicha palabra, seguramente sólo una ellas es posible. Por otra parte, el interés se centra también en asignar una etiqueta a tod aquellas palabras que aparecen en los textos, pero que no están presentes en nuestro diccionar y garantizar de alguna manera que ésa es la etiqueta correcta. Un buen rendimiento en esta fa asegura la viabilidad de procesamientos posteriores tales como los análisis sintáctico y semántic

Tradicionalmente, el problema de la etiquetación se aborda a partir de recursos lingüístic bajo la forma de diccionarios y textos escritos, previamente etiquetados o no. Esta línea desarrollo se denomina lingüística basada en corpus. Dichos textos se utilizan para ajust los parámetros de funcionamiento de los etiquetadores. Este proceso de ajuste se denomi entrenamiento. Las técnicas tradicionales engloban métodos basados en reglas, tales como aprendizaje de etiquetas mediante transformaciones y dirigido por el error, y también métod estocásticos, tales como los modelos de Markov ocultos o HMM,s (*Hidden Markov Models*), q constituyen actualmente la aproximación que ofrece mejores rendimientos.

Aún con todo esto, un pequeño porcentaje de palabras etiquetadas erróneamente (2-3 es una característica que está siempre presente en los sistemas de etiquetación puramen estocásticos. Este proyecto aborda el diseño e implementación de un sistema de etiquetaci híbrido que considere la aplicación combinada de reglas contextuales y algoritmos programación dinámica para intentar eliminar estos errores residuales. Dichas reglas deb ser proporcionadas por un usuario experto, y el sistema se ocupa de compilarlas bajo la forn de una estructura matemática denominada traductor de estado finito, que es la que permite q una ejecución eficiente de las mismas.

Por otra parte, la mayoría de los etiquetadores actuales asumen que los textos aparecen correctamente segmentados, es decir, divididos en *tokens* que identifican perfectamente ca componente. Sin embargo, esta hipótesis no es realista debido a la naturaleza heterogén de los textos. Las mayores dificultades surgen cuando la segmentación es ambigua. P ejemplo, la expresión `sin embargo` se etiquetará normalmente de manera conjunta como u

r a cada *token*, sino que además debería saber decidir si algunos de ellos constituyen o
a misma entidad y asignar, en cada caso, la cantidad de etiquetas adecuada en función de
ferentes alternativas de segmentación. Así pues, este proyecto aborda también la fase de
o de una extensión del algoritmo de Viterbi (el algoritmo de programación dinámica sobre
,s que realiza la etiquetación) capaz de etiquetar flujos de *tokens* de distinta longitud con
omplejidad temporal comparable a la del algoritmo de clásico.

oras Clave: Procesamiento del Lenguaje Natural (*Natural Language Processing*),
ografía Computacional (*Computational Lexicography*), Etiquetación de las Partes del
rso (*Part-of-Speech Tagging*), Modelos de Markov Ocultos (*Hidden Markov Models*),
áticas de Restricciones (*Constraint Grammars*), Traductores de Estado Finito (*Finite-
Transducers*).

Agradecimientos

Ha llegado el momento de expresar mi más sincero agradecimiento a todas las personas q
me han ayudado a llegar hasta aquí, compartiendo conmigo su sabiduría, su formación,
entusiasmo, su amistad y su confianza durante estos últimos años.

A mi director de proyecto, Jorge Graña, por toda la ayuda y dedicación que me ha presta
durante el desarrollo de este proyecto.

A mis compañeros de laboratorio, con los que he compartido estos últimos meses de traba

A mis compañeros de clase, con los que he vivido muchos momentos.

A Jano, mi compañero de prácticas, con el que he formado un buen equipo de trabajo, y
que me ha aguantado en los momentos de máxima tensión.

A mis padres, que me han dado la oportunidad de recibir una buena educación y siemp
me han apoyado en mis decisiones.

A mi hermano, que siempre ha estado ahí cuando lo necesitaba.

A mis amigos y amigas, por las dosis de distracción que uno siempre necesita.

Y a Suso, por haber confiado siempre en mis posibilidades.

A todos y por todo, muchísimas gracias.

Índice General

Parte I

Etiquetación del lenguaje natural: conceptos previos

Capítulo 1

Introducción

El objetivo último que persigue el Procesamiento del Lenguaje Natural o NLP[1] es el perfec
análisis y entendimiento de los lenguajes humanos. Su utilidad es obvia, ya que consiguien
esto dispondríamos, por ejemplo, de interfaces amigables hombre-máquina utilizando lengua
natural, de buenos sistemas de búsqueda de información, de traductores, etc. Actualmen
estamos todavía lejos de conseguir este objetivo. Por esta razón, la mayoría de los esfuerz
de investigación en lo que al NLP se refiere han sido dirigidos hacia tareas intermedias q
dan sentido a alguna de las múltiples características estructurales inherentes a los lenguaj
sin requerir un entendimiento completo. Una de esas tareas es la asignación de categori
gramaticales a cada una de las palabras del texto. Este proceso se denomina también *etiquetaci*
de las partes del discurso o POST[2]. En definitiva, se trata de decidir si cada palabra es
sustantivo, un adjetivo, un verbo, etc. Por ejemplo, si consideramos aisladamente la palab
`sobre`, vemos que puede ser:

- Un sustantivo, como por ejemplo en la frase: `mételo en ese sobre`.
- Una preposición, como por ejemplo en la frase: `déjalo sobre la mesa`.
- O un verbo en primera o tercera persona del presente de subjuntivo del verbo `sobra`
  como por ejemplo en la frase: `dame lo que te sobre`.

Es decir, si echamos un vistazo al contexto en el que dicha palabra aparece, es muy probab
que sólo una de esas tres etiquetas sea la correcta. El proceso de etiquetación debe elimin
por tanto este tipo de ambigüedades y encontrar cuál es el papel más probable que juega ca
palabra dentro de una frase. Es más, dicho proceso debe ser capaz también de asignar u
etiqueta a cada una de las palabras que aparecen en un texto y que no están presentes
nuestro diccionario, y garantizar de alguna manera que ésa es la etiqueta correcta.

Como ya hemos visto, la etiquetación es un problema de ámbito limitado. En lugar
construir un análisis completo, simplemente se establecen las categorías de las palabras, y
dejan de lado problemas tales como por ejemplo el de encontrar la correcta ligadura de l
frases preposicionales. Debido a esto, la etiquetación es más sencilla de resolver que los análi
sintáctico o semántico, y el rendimiento es bastante elevado. Las aproximaciones más exitos
son capaces de etiquetar correctamente alrededor del 95-97% de las palabras. Sin embargo,

[1] A lo largo de la presente memoria intentaremos hacer referencia a todos los conceptos propios del lengua
especializado de la materia a tratar con su correspondiente término en español; no obstante, la mayoría de
acrónimos utilizados constituirán una excepción a este principio, ya que muchas veces su traducción resu
excesivamente artificial; en este caso, utilizaremos la sigla NLP, que responde al término inglés *Natural Langu*
Processing.

n aparecer entre una y dos palabras mal etiquetadas en cada frase. Además, estos errores mpre se localizan en las categorías más pobladas, tales como sustantivos, adjetivos o verbos, en principio parece más probable el encontrarse con palabras desconocidas. Muchas veces, rores aparecen asociados a las partículas que conectan los sintagmas entre sí, tales como siciones, conjunciones o relativos, y pueden hacer que una frase tome un significado muy to del original.

ún con todo esto, y a pesar de sus limitaciones, la información que se obtiene mediante la tación es muy útil. Las potenciales aplicaciones de los textos en lenguaje natural aumentan o dichos textos están anotados, y el primer nivel lógico de anotación es normalmente la ación de este tipo de etiquetas gramaticales a cada una de las palabras. Por tanto, los ya no serán vistos como una mera secuencia de caracteres, sino como una secuencia de ades lingüísticas con algún tipo de significado natural. El texto etiquetado puede utilizarse ces para introducir nuevos tipos de anotaciones, normalmente mediante posteriores análisis ticos o semánticos, o puede utilizarse también para recoger datos estadísticos sobre el e un idioma que pueden servir para múltiples aplicaciones. El trabajo a partir de las ras etiquetadas hace mucho más viables tareas tales como el proceso y síntesis del diálogo, la grafía computacional, o el más reciente y novedoso tema de la recuperación de información.

Fuentes de información relevantes para la etiquetación

decidir cuál es la etiqueta correcta de una palabra existen básicamente dos fuentes de nación:

La primera de ellas consiste en mirar las etiquetas de las otras palabras que pertenecen al contexto en el que aparece la que nos interesa. Esas palabras podrían ser también ambiguas, pero el hecho de observar secuencias de varias etiquetas nos puede dar una idea de cuáles son comunes y cuáles no lo son. Por ejemplo, en inglés, una secuencia como artículo-adjetivo-sustantivo es muy común, mientras que otras secuencias como artículo-adjetivo-verbo resultan muy poco frecuentes o prácticamente imposibles. Por tanto, si hubiera que elegir entre sustantivo o verbo para etiquetar la palabra `play` en la frase `a new play`, obviamente optaríamos por la primera de las etiquetas.

Este tipo de estructuras constituyen la fuente de información más directa para el proceso de etiquetación, pero por sí misma no resulta demasiado exitosa: uno de los primeros etiquetadores basado en reglas deterministas que utilizaba este tipo de patrones sintagmáticos etiquetaba correctamente sólo el 77% de las palabras [Greene y Rubin 1971]. Una de las razones de este rendimiento tan bajo es que en inglés las palabras que pueden tener varias etiquetas son efectivamente muy numerosas, debido sobre todo a procesos productivos como el que permite a casi todos los sustantivos que podamos tener en nuestro diccionario transformarse y funcionar como verbos, con la consiguiente pérdida de la información restrictiva que es necesaria para el proceso de etiquetación.

Sin embargo, existen palabras que, aunque puedan ser usadas como verbos, su aparición es mucho más probable cuando funcionan como sustantivos. Este tipo de consideraciones sugiere la segunda fuente de información: el simple conocimiento de la palabra concreta

posteriores de esa época.

La información léxica de las palabras resulta tan útil porque la distribución de uso de u palabra a lo largo de todas sus posibles etiquetas suele ser rara. Incluso las palabras c un gran número de etiquetas aparecen típicamente con un único uso o etiqueta particul

Efectivamente, esta distribución es tan peculiar que casi siempre existe esa etiqueta denomina *básica*, mientras que las otras representan usos derivados de ésta, y como resultado se h producido algunos conflictos en relación con la manera en la que el término *etiqueta*, *categor* o *parte del discurso* debe ser utilizado. En las gramáticas tradicionales, se pueden encontr palabras clasificadas como *un sustantivo que está siendo utilizado como un adjetivo*, lo cu confunde la etiqueta básica del lexema de la palabra, con la función real que dicha palabra es desempeñando dentro del contexto. Aquí, al igual que en la lingüística moderna en general, n centraremos siempre en el segundo concepto, es decir, en el uso real de las palabras dentro cada frase concreta.

En cualquier caso, la distribución de uso de las palabras proporciona una informaci adicional de gran valor, y es por ello por lo que parece lógico esperar que las aproximacion estadísticas al proceso de etiquetación den mejores resultados que las aproximaciones basad en reglas deterministas. En éstas últimas, uno sólo puede decir que una palabra puede o puede ser un verbo, mientras que en una aproximación estadística se puede decir *a priori* q una palabra tiene una gran probabilidad de ser un sustantivo, pero también que existe u posibilidad, por remota que sea, de ser un verbo o incluso cualquier otra etiqueta. Sin embarg en los últimos años, se han recuperado los métodos simbólicos y, con un pequeño replanteamient se han logrado unos resultados muy interesantes. Mientras que antes las reglas se utilizaban pa cambiar la etiqueta de la palabra o para asignar una etiqueta dada según el contexto, ahora idea consiste en reducir el uso de estas reglas y potenciar un nuevo tipo de reglas de men compromiso, que consisten en eliminar etiquetaciones imposibles en lugar de asignar posibl etiquetas. Por ejemplo, si tenemos una palabra que puede ser verbo o sustantivo, y ant una que sólo puede ser determinante, debemos eliminar la posibilidad de etiquetar la palab ambigua como verbo. Esto se puede llevar a cabo gracias a lenguajes de alta expresivid en los que se pueden contemplar casos conocidos del idioma y refinar así el conjunto etiquetas que se le asignan a una palabra a través de un diccionario [Voutilainen y Heikkilä 199 Tapanainen y Voutilainen 1994, Samuelsson y Voutilainen 1997].

De esta manera, se han conseguido resultados que superan al 99% de aciertos con un conjun de unas 1.000 reglas aproximadamente. A pesar de esto, nos encontramos con el problema que, como se trata de un método de menos compromiso que el anterior, no se garantiza obtención de una única etiqueta por palabra, pudiendo quedar algunas palabras ambiguas a largo del texto. Por ello, este método se suele utilizar en combinación con otros sistemas etiquetación.

En este proyecto describiremos los etiquetadores estocásticos y simbólicos más conocid y abordaremos el diseño de un sistema híbrido que considere la aplicación combinada reglas contextuales con técnicas estocásticas. Es decir, utilizaremos de manera combina la información sintagmática proporcionada por las secuencias de etiquetas y la informaci léxica proporcionada por las palabras, junto con todas las ventajas que nos ofrecen amb

r sistema conocido que realmente intentaba asignar etiquetas en función del contexto
mático es el programa basado en reglas presentado en [Klein y Simmons 1963], aunque
imadamente la misma idea fue presentada también en [Salton y Thorpe 1962]. Klein
mons utilizan los términos *código* y *codificación*, aunque son aparentemente de uso
ambiable con *etiqueta* y *etiquetación*. El primer etiquetador probabilístico conocido
olz *et al.* 1965]. Este sistema asignaba inicialmente etiquetas a algunas palabras mediante
de un diccionario, de reglas morfológicas y de otras reglas confeccionadas a medida. El
de las palabras se etiquetaban entonces usando probabilidades condicionadas calculadas a
de secuencias de etiquetas. Ni que decir tiene que no se trataba de un modelo probabilístico
lefinido.

os grandes grupos de trabajo, uno en la Universidad Brown y otro en la Universidad de
ster, emplearon considerables recursos para etiquetar dos grandes *corpora* de texto: el
s BROWN y el corpus LOB (Lancaster-Oslo-Bergen). Ambos grupos coincidieron en que
stencia de un corpus anotado sería de incalculable valor para la investigación en el campo
etiquetación, y es cierto que sin estos dos *corpora* etiquetados el progreso de esta línea
abajo hubiera sido extremadamente duro, si no imposible. La disponibilidad de grandes
lades de texto etiquetado es sin lugar a dudas una importante razón que explica el hecho
e la etiquetación haya sido un área de investigación tan activa.

corpus BROWN fue preetiquetado automáticamente con el etiquetador basado en reglas
IT [Greene y Rubin 1971]. Esta herramienta utilizaba información léxica sólo para limitar
iquetas de las palabras y sólo aplicaba reglas de etiquetación cuando las palabras del
xto no presentaban ambigüedades. La salida de este etiquetador se corrigió entonces
almente. Este esfuerzo duró años, pero finalmente proporcionó los datos de entrenamiento
onstituyeron la base de numerosos trabajos posteriores.

no de los primeros etiquetadores basado en modelos de Markov ocultos o HMM,s[3] fue
o en la Universidad de Lancaster como parte del proyecto de etiquetación del corpus
[Garside *et al.* 1987, Marshall 1987]. El punto central de este etiquetador era el manejo
obabilidades para las secuencias de bigramas de etiquetas, con uso limitado de un contexto
ayor orden, y las probabilidades de asignación de una palabra a sus diferentes etiquetas
gestionadas mediante factores de descuento diseñados a medida. Los etiquetadores
os en modelos de Markov que etiquetan utilizando ambos tipos de información, las
bilidades de las palabras y las probabilidades de transición entre etiquetas, fueron
ucidos en [Church 1988] y [DeRose 1988].

pesar de que los trabajos de Church y DeRose fueron la clave del resurgir de los métodos
sticos en lingüística computacional, la aplicación de los HMM,s al proceso de etiquetación
comenzado realmente mucho antes en los centros de investigación de IBM en Nueva York
ís [Jelinek 1985, Derouault y Merialdo 1986]. Otras referencias de los primeros trabajos
etiquetación probabilística incluyen [Bahl y Mercer 1976, Baker 1975, Foster 1991].

n los últimos años, la etiquetación ha sido un área con una gran actividad dentro
investigación en NLP, lo que ha permitido la aparición de otras técnicas para
quetación, como por ejemplo pueden ser las redes nueronales [Benello *et al.* 1989,
ues y Pereira 1996], los árboles de decisión [Schmid 1994], y el aprendizaje basado
emoria [Daelemans *et al.* 1996, Zavrel y Daelemans 1999]. De todas las técnicas más
osas merece mención la que se basa en reglas reduccionistas, ya que uno de los objetivos

no tenemos ningún corpus de entrenamiento equiquetado, o cuando a pesar de tenerlo, l aplicaciones trabajan con textos muy diferentes y los datos de entrenamiento son de po utilidad. Otros estudios se han centrado en la manera de construir un conjunto de etiquet automáticamente, con el propósito de crear categorías apropiadas para un idioma o para estilo de texto particular [McMahon y Smith 1996].

1.3 Rendimiento y precisión de los etiquetadores

Las cifras de rendimiento conocidas para los etiquetadores se encuentran casi siempre dentro d rango del 95 al 97% de acierto, cuando se calculan sobre el conjunto de todas las palabras de texto. Algunos autores proporcionan la precisión sólo para los términos ambiguos, en cuyo ca las cifras son por supuesto menores. Sin embargo, el rendimiento depende considerablemente una serie de factores, tales como los siguientes:

- La cantidad de texto de entrenamiento disponible. En general, cuanto más texto se teng mejor.

- El juego de etiquetas[4]. Normalmente, cuanto más grande es el conjunto de etiquet considerado, existe más ambigüedad potencial, y la tarea de etiquetación se vuelve m compleja. Por ejemplo, en inglés, algunos juegos de etiquetas hacen una distinción ent `to` como preposición y `to` como marca de infinitivo, y otros no. En el primero de los cas la palabra `to` podría etiquetarse incorrectamente.

- La diferencia entre, por un lado el corpus de entrenamiento y el diccionario, y por ot el corpus de aplicación. Si los textos de entrenamiento y los textos que posteriormente van a etiquetar proceden de la misma fuente, por ejemplo, textos de la misma época, extraídos de un mismo periódico particular, entonces la precisión será alta. Normalmen los resultados que los investigadores proporcionan sobre sus etiquetadores provienen situaciones como ésta. Pero si los textos de aplicación pertenecen a un periodo de tiem distinto, a una fuente distinta, o a un género o estilo distinto, por ejemplo, textos científi contra textos periodísticos, entonces el rendimiento será bajo.

- Las palabras desconocidas. Un caso especial del punto anterior es la cobertura d diccionario. La aparición de palabras desconocidas puede degradar el rendimiento. U situación típica en la cual el porcentaje de palabras fuera de vocabulario puede ser alto cuando se intenta etiquetar material procedente de algún dominio técnico.

Un cambio en cualquiera de estas cuatro condiciones puede producir un impacto muy fuerte la precisión de los etiquetadores, pudiendo provocar que el rendimiento se reduzca de mane dramática. Si el conjunto de entrenamiento es pequeño, el juego de etiquetas grande, y el corp a etiquetar significativamente diferente del corpus de entrenamiento, o si nos enfrentamos a gran número de palabras desconocidas, el rendimiento puede caer muy por debajo del rango cifras citado anteriormente.

Es importante también señalar que efectivamente estos factores son externos al proceso etiquetación y al método elegido para realizar dicho proceso. Es por ello que el efecto q producen es a menudo mucho mayor que la influencia ejercida por el propio método en sí.

onociendo esta motivación, resulta sorprendente el hecho de que existan más trabajos y ncias bibliográficas sobre el proceso de etiquetación aisladamente, que sobre la aplicación etiquetadores a tareas de interés inmediato. A pesar de esto, intentaremos enumerar aquí licaciones más importantes en las que la etiquetación ha jugado y está jugando un papel nte. Además de las citadas anteriormente, estas aplicaciones podrían ser las siguientes:

La mayoría de las aplicaciones requieren un paso de procesamiento adicional posterior a la etiquetación: el análisis sintáctico parcial[5], que puede reflejar varios niveles de detalle dentro del análisis sintáctico. Los analizadores parciales más simples se limitan a buscar las frases nominales de una oración. Otras aproximaciones más sofisticadas asignan funciones gramaticales a esas frases nominales (sujeto, objeto directo, objeto indirecto, etc.) y al mismo tiempo proporcionan información parcial sobre las ligaduras[6]. Una presentación conjunta del análisis sintáctico parcial y de la etiquetación puede verse en [Abney 1996].

Un uso importante de la combinación de la etiquetación y el análisis sintáctico parcial es el proceso de adquisición automática de información léxica[7]. El objetivo general de este proceso es desarrollar algoritmos y técnicas estadísticas para rellenar los huecos de información sintáctica y semántica que existen en los diccionarios electrónicos, mediante el estudio de los patrones de palabras que se pueden observar en grandes *corpora* de textos. Existen multitud de problemas relacionados con la adquisición de información léxica: la colocación de las palabras, las preferencias de selección[8], los marcos de subcategorización[9], o la categorización semántica[10]. La mayoría de este tipo de propiedades de las palabras no se suele cubrir completamente en los diccionarios. Esto es debido principalmente a la productividad de los lenguajes naturales: constantemente inventamos nuevas palabras o nuevos usos de las palabras que ya conocemos. Incluso aunque fuera posible crear un diccionario que reflejara todas las características del lenguaje actual, inevitablemente estaría incompleto en cuestión de pocos meses. Esta es la razón por la cual la adquisición automática de información léxica es tan importante en el NLP estadístico.

Otra aplicación importante es la extracción de información[11]. El objetivo principal de la extracción de información es encontrar valores para un conjunto predeterminado de ranuras de información de una plantilla. Por ejemplo, una plantilla de información meteorológica podría tener ranuras para el tipo de fenómeno (tornado, tormenta de nieve, huracán), la localización del evento (la bahía coruñesa, Europa Central, Venezuela), la fecha (hoy, el próximo domingo, el 27 de diciembre de 1999), y el efecto causado (apagón general, inundaciones, accidentes de tráfico, etc.). La etiquetación y el análisis sintáctico parcial ayudan a identificar las entidades que sirven para rellenar las ranuras de información y las relaciones entre ellas. En cierta manera, podría verse la extracción de información como

---

mbién denominado *partial parsing*.

r ejemplo, *esta frase nominal está ligada a otra frase de la derecha*, la cual puede aparecer especificada o

mbién denominado *lexical acquisition*.

r ejemplo, el verbo `comer` normalmente lleva como objeto directo elementos relacionados con la comida.

r ejemplo, el beneficiario del verbo `contribuir` se expresa mediante una frase preposicional encabezada preposición `a`.

r ejemplo, cuál es la categoría semántica de una nueva palabra no presente en nuestro diccionario.

information extraction, y a veces también referenciada como *message understanding* (entendimiento del

- La etiquetación y el análisis sintáctico parcial se pueden utilizar también para encontr los términos de indexación adecuados en los sistemas de recuperación de información La mejor unidad de tratamiento para decidir qué documentos están relacionados con l consultas de los usuarios a menudo no es la palabra individual. Frases como `Estad Unidos de América` o `educación secundaria` pierden gran parte de su significado se rompen en palabras individuales. El rendimiento de la recuperación de informaci se puede incrementar si la etiquetación y el análisis sintáctico parcial se dirigen hac el reconocimiento de la frase nominal y si las asociaciones consulta-documento realizan apoyándose en este tipo de unidad de información de más alto rango significado [Fagan 1987, Smeaton 1992, Strzalkowski 1995]. Otra línea de investigaci relacionada se ocupa de la normalización de frases, es decir, del estudio de las múltipl variantes de términos que representan realmente la misma unidad de informaci básica[13] [Jacquemin *et al.* 1997].

- Por último, existen también trabajos relacionados con los llamados sistemas de respues de preguntas[14], los cuales intentan responder a una cuestión del usuario devolviendo u frase nominal concreta, como por ejemplo un lugar, una persona o una fecha [Kupiec 199 Burke *et al.* 1997]. Es decir, ante una pregunta como `¿quién mató a John Lenno` obtendríamos como respuesta `Chapman`, en lugar de una lista de documentos tal y cor ocurre en la mayoría de los sistemas de recuperación de información. Una vez más, análisis de la consulta para determinar qué tipo de entidad está buscando el usuario y cór está relacionada con las frases nominales que aparecen en la pregunta requiere etiquetaci y análisis sintáctico parcial.

Terminamos, sin embargo, con un resultado no del todo positivo: los analizadores sintáctic probabilísticos mejor lexicalizados son hoy en día suficientemente buenos como para trabajar c texto no etiquetado y realizar la etiquetación por sí mismos, en lugar de utilizar un etiquetad como preprocesador [Charniak 1997]. Por tanto, el papel de los etiquetadores parece ser m bien el de un componente que aligera rápidamente la complejidad de los textos y proporcio suficiente información para multitud de interesantes tareas de NLP, y no el de una fase preprocesado deseable e imprescindible para todas y cada unas de las aplicaciones.

1.5 Etiquetación mediante gramáticas de restricciones

En esta sección, hacemos una mención especial a las gramáticas de restricciones, porque en presente proyecto se estudiará, analizará y mejorará la sintaxis de las reglas utilizadas por est gramáticas, y además se desarrollará una forma de compilación y ejecución eficiente basada traductores de estado finito. Por esta razón, resulta interesante el presentar una recopilaci histórica de todos aquellos etiquetadores que hacen uso de este tipo de gramáticas.

Las gramáticas basadas en restricciones son una técnica en alza en estos últimos años. E concreto, una de las técnicas de etiquetación de mejor rendimiento para el inglés es un formalism

[12] O también *information retrieval*.

[13] Por ejemplo, `book publishing` y `publishing of books`.

xto local. No existe una verdadera noción de gramática formal, sino más bien un conjunto stricciones, casi siempre negativas, que van eliminando los análisis imposibles según el xto [Karlsson *et al.* 1995, Samuelsson *et al.* 1996]. La idea es similar al aprendizaje basado ansformaciones, excepto por el hecho de que es un humano, y no un algoritmo, el que ica iterativamente el conjunto de reglas de etiquetación para minimizar el número de s. En cada iteración, el conjunto de reglas se aplica al corpus y posteriormente se an modificar dichas reglas de manera que los errores más importantes queden manualmente gidos. Esta metodología propone la construcción de un pequeño sistema experto para la etación, y parece ofrecer mejores rendimientos que los etiquetadores basados en modelos de ov ocultos, especialmente cuando los *corpora* de entrenamiento y de aplicación no provienen misma fuente. Sin embargo, la comparación de estos dos modelos es difícil de realizar, ya uando el sistema EngCG no es capaz de resolver determinadas ambigüedades devuelve un nto de más de una etiqueta. Por otra parte, para aquellas personas ya familiarizadas con metodología, la construcción de este tipo de etiquetadores no requiere más esfuerzo que la rucción de un etiquetador basado en un HMM [Chanod y Tapanainen 1995], aunque quizás, ltimo sea más accesible y automático. El problema de esta técnica es que hace falta un de lingüistas que proporcionen las reglas, lo que supone un problema en comparación con endizaje automático de los HMM,s. Sin embargo, los resultados son muy interesantes, aún a de no decidir en todas las etiquetas.

Real Academia Española está desarrollando también un formalismo de reglas de cciones denominado sistema Rtag, para la anotación automática de los *corpora* Corde[16] y [17] [Porta 1996, Sánchez *et al.* 1999]. Este sistema aplica gramáticas de reglas de contexto eradas sobre textos anotados ambiguamente. Esto quiere decir que cuando un contexto ace la descripción estructural de una regla, recibe la puntuación que indica la regla. Esta ación puede ser positiva, para promover lecturas, o negativa, para penalizarlas. Una nalizado el proceso, permanecen las lecturas con mayor puntuación siempre que estén ncima de un umbral definido previamente. El sistema también intenta eliminar lecturas ibles en función del contexto, sin pérdida de lecturas posibles aunque éstas sean poco bles. Para la poda de lecturas en función del contexto se utiliza información derivada del o texto (características estructurales, tipográficas o secuenciales), información gramatical e todo concordancia y restricciones de aparición conjunta) e información gramatical tural (toma de decisiones con ayuda de la información estructural derivable de la secuencia del texto).

tro etiquetador basado en gramáticas de restricciones que ha sido utilizado con éxito sobre añol es el sistema Relax [Padró 1996]. Aquí las reglas pueden ser escritas tanto de forma al como generadas automáticamente mediante un algoritmo de adquisición basado en un de decisión [Màrquez y Padró 1997, Màrquez y Rodríguez 1997].

Motivación y objetivos del proyecto

quetación, como vimos anteriormente, se aborda a partir de recursos lingüísticos bajo la de diccionarios y textos previamente etiquetados o no, que serán utilizados para ajustar

glish Constraint Grammar.
rpus Diacrónico del Español.

a nivel de frase. Para una nueva frase a etiquetar, se accede primeramente a un diccionario, busca de todas las posibles etiquetas candidatas para cada una de las palabras que conform dicha frase. Todas estas etiquetas individuales pueden involucrar a un gran número de posibl caminos o secuencias de etiquetación globales para la frase. Así pues, seguidamente, se constru una estructura en forma de enrejado capaz de albergar todas esas combinaciones. Por último, algoritmo de programación dinámica, el algoritmo de Viterbi, es capaz de seleccionar de mane eficiente la secuencia o secuencias de etiquetas más probables.

No obstante, un pequeño porcentaje de palabras etiquetadas erróneamente (2-3%) es u característica siempre presente en los sistemas de etiquetación puramente estocásticos. Con objetivo de reducir este porcentaje de errores residuales se acude a las reglas contextuales. esta manera, el enrejado sobre el que opera el algoritmo de Viterbi va a poder ser podado p estas reglas, bien antes de aplicar el etiquetador estocástico, o bien después de aplicarlo.

A continuación indicamos de una forma más detallada los puntos que abordaremos en es primer objetivo del proyecto:

- Tomando como punto de partida el proyecto de fin de carrera [Reboredo y Graña 200 se estudiarán y analizarán las reglas contextuales aquí desarrolladas, que se basan en gramáticas de restricciones, con el propósito de simplificar su sintaxis y así facilitar proceso de familiarización, entendimiento y uso de las mismas. Lo que se persigue c esto es facilitar el árido camino de comprensión de las reglas, para que así los lingüist no sean tan reacios a proporcionar un conjunto de reglas contextuales, lo suficientemen grande y adecuado, para poder evaluar de una manera fiable, eficiente y completa nuestr sistemas.

- El siguiente punto a desarrollar consiste en la compilación de las reglas contextuales pa lograr una ejecución lo más eficiente posible. Este proceso de compilación consiste representar dichas reglas mediante una estructura matemática más compacta, que denomina *traductor de estado finito*. Unido a esto surge la dificultad de que la teo de traductores no está tan claramente especificada como la teoría de autómatas. Es dec problemas tales como la minimización o determinización de los traductores constituy líneas de trabajo vigentes hoy en día, ya que no existen algoritmos universales pa resolverlos. Por tanto, tendremos que enfrentarnos al estudio y análisis en profundidad todos estos aspectos que la teoría de traductores no refleja con claridad.

- Por último, se llevará a cabo la implementación de un prototipo que cumpla con especificaciones anteriores. Cabe destacar que, en el momento de la realización este proyecto, no se ha podido disponer de reglas contextuales elaboradas por expert lingüistas. Si bien éstos han contribuido parcialmente en la mejora del formalismo reglas, lo que no ha sido posible es evaluar el sistema sobre un conjunto de reglas real No obstante, la aplicación del prototipo sobre un caso de estudio sencillo permitirá valid el correcto funcionamiento del mismo. Por tanto, con el diseño e implementación esta herramienta, queda preparado un sistema híbrido de etiquetación al que sólo le res ser alimentado con reglas contextuales verdaderamente representativas del idioma que quiera tratar.

Como segundo objetivo del presente proyecto abordaremos otra extensión natural de la aplicaci

ás concretamente, la mayoría de los etiquetadores actuales asumen que los textos aparecen rrectamente segmentados, es decir, divididos en *tokens* que identifican perfectamente cada onente. Sin embargo, esta hipótesis de trabajo no es realista debido a la naturaleza ogénea de los textos. Las mayores dificultades surgen cuando la segmentación es ambigua. emplo, la expresión `sin embargo` se etiquetará normalmente de manera conjunta como una nción, pero en algún otro contexto podría ser una secuencia formada por una preposición sustantivo. De igual forma, la palabra `ténselo` puede ser una forma del verbo `tener` con ronombres enclíticos, o bien una forma del verbo `tensar` con un solo pronombre.
ebido a estas segmentaciones ambiguas, un etiquetador debería ser capaz de enfrentarse a de *tokens* de distinta longitud. Es decir, no sólo debería ser capaz de decidir qué etiqueta r a cada *token*, sino que además debería saber decidir si algunos de ellos constituyen o no isma entidad y asignar, en cada caso, la cantidad de etiquetas adecuada en función de las ntes alternativas de segmentación.
ásicamente, este segundo objetivo del proyecto involucra a las tareas que se enumeran a nuación:

En primer lugar, será necesario definir una estructura capaz de representar coherentemente las distintas alternativas de segmentación que puede proporcionar un módulo de preprocesamiento ante una nueva frase. Veremos que los enrejados clásicos dejan de ser adecuados, por lo que serán sustituidos por una nueva estructura de representación denominada retículo[18].

Posteriormente, abordaremos la adaptación del algoritmo tradicional de Viterbi, para que pueda trabajar sobre retículos, en lugar de sobre enrejados.

Por último, este algoritmo sufrirá una nueva adaptación, que es la que efectivamente permitirá etiquetar flujos de *tokens* de distinta longitud con una complejidad temporal comparable a la del algoritmo de clásico.

uiente sección permitirá al lector situar los objetivos anteriormente mencionados a lo largo capítulos desarrollados en este documento.

Estructura de la presente memoria

contenidos fundamentales de este trabajo se dividen en cuatro partes claramente nciadas. A continuación introducimos cada una de ellas, centrándonos en el contenido capítulos que las conforman:

La parte I engloba los conceptos preliminares sobre NLP en general, y sobre etiquetación en particular, necesarios para abordar el resto de los contenidos del documento. Además de la presente introducción, esta parte consta también del siguiente capítulo:

- El capítulo 2 esboza las distintas técnicas existentes para la realización del análisis léxico de los textos. Este análisis transforma los caracteres de entrada en unidades de más alto nivel de significado, normalmente las palabras, y obtiene rápida y cómodamente todas las etiquetas candidatas de esas palabras. Por tanto, el análisis

actual del arte en lo que al proceso de etiquetación se refiere. Los capítulos que conform esta parte son los siguientes:

- El capítulo 3 se ocupa de los etiquetadores basados en modelos de Markov ocultos HMM,s[19]. Los HMM,s presentan un conjunto de estados que normalmente coinci con el conjunto de etiquetas que se está considerando. La dependencia probabilísti del estado o etiqueta actual generalmente se trunca para considerar sólo uno o d de los estados o etiquetas precedentes (propiedad del horizonte limitado) y es dependencia no varía a lo largo del tiempo (propiedad del tiempo estacionario). Es quiere decir que si, por ejemplo, se ha establecido en 0,2 la probabilidad de que verbo venga después de un pronombre al principio de una frase, dicha probabilid es la misma para el resto de la frase e incluso para otras frases posteriores. Cor ocurre con la mayoría de los modelos probabilísticos, las dos propiedades de Mark no formalizan la realidad a la perfección, pero constituyen una buena aproximació
- El capítulo 4 describe el sistema de etiquetación presentado por Eric Brill, que utili reglas de transformación tanto léxicas como contextuales [Brill 1993b]. Las reglas se escriben a mano, sino que se generan automáticamente a partir de un corpus entrenamiento mediante un procedimiento iterativo que, en cada etapa, selecciona l transformaciones que minimizan el número de errores cometidos. El procedimien se repite hasta que dicho número cae por debajo de un cierto umbral. Este méto de entrenamiento se denomina *aprendizaje basado en transformaciones y dirigido p el error*[20].

- La parte III engloba el primer objetivo del proyecto, con lo cual aquí presentaremo estudiaremos y desarrollaremos todos los puntos necesarios que conllevan a la resoluci del mismo. Para ello, nos apoyaremos en gramáticas de restricciones y *traductores estado finito* principalmente. Los capítulos que conforman esta parte son los siguientes

  - El capítulo 5 enfatiza la necesidad de acudir al contexto para mejorar la precisión los métodos de etiquetación estocástios, e introduce la necesidad de un formalism de reglas basado en gramáticas de restricciones. A continuación se estudia lenguaje de reglas desarrollado para este propósito, que se conoce como lengua LEMMERS (Lenguaje de Etiquetado MEdiante Restricciones Simples). Sin embar este formalismo, que es una mejora de las reglas que presentan las gramáticas restricciones, todavía es complejo y difícil de entender. Por esta razón nosotr plantearemos aquí una mejora sobre la sintaxis de estas reglas para hacerlas m sencillas, intuitivas y expresivas.
  - El capítulo 6 se ocupa de la representación de los esquemas generales de regl introducidos en el capítulo anterior, a través de *traductores de estado finito*, ya que compilación de las reglas bajo esta estructura matemática redunda en una ejecuci más eficiente. Con lo cual, se indicarán los traductores generales que represetan l esquemas generales de las reglas junto con algunos ejemplos. También abordaremos desarrollo y la implementación del prototipo encargado de llevar a cabo la compilaci y la ejecución de las reglas contextuales, así como un caso de estudio para verific

[19] *Hidden Markov Models.*

La parte IV se centra en el segundo objetivo de este proyecto, de manera que presentaremos el proceso de segmentación, los problemas a los que se enfrenta y las soluciones que propone. También incluimos aquí el capítulo de conclusiones finales y trabajo futuro.

- El capítulo 7 trata de explicar en qué consiste el proceso de segmentación y refleja que la identificación de los *tokens* que participan en una frase no es una operación tan sencilla como se cree de antemano, ya que puede haber ambigüedades. Para esto es necesario elaborar un sistema de etiquetación que asigne las etiquetas adecuadas en función de las diferentes alternativas de segmentación. Por esta razón, consideraremos una extensión del algoritmo de Viterbi para que tenga en cuenta estas indicaciones y aborde el problema de manera que no supere la complejidad temporal del algoritmo tradicional.
- El capítulo 8 finaliza este estudio presentando las principales conclusiones extraídas y las líneas de trabajo futuro.

mente, se incluyen los apéndices que contienen los aspectos complementarios, así como la le las referencias bibliográficas utilizadas a lo largo de este trabajo.

Capítulo 2

Análisis léxico de grandes diccionarios

El problema de la correcta etiquetación o del análisis sintáctico de una frase dada puede result
complejo si se aborda tratando directamente el flujo de los caracteres de entrada que conform
esa frase. Para evitar dicha complejidad, normalmente existirá un paso de procesamiento prev
cuya misión es la de transformar el flujo de caracteres de entrada en un flujo de elementos
más alto nivel de significado[1], que típicamente serán las palabras de la frase en cuestión, y
de obtener rápida y cómodamente todas las etiquetas candidatas de esas palabras. Dicho pa
previo se denomina análisis léxico[2].

Este capítulo está destinado a esbozar las distintas técnicas existentes para la realización
esta tarea. No obstante, comenzaremos presentando en detalle la visión particular que se
utilizado a lo largo de este trabajo para modelizar los diccionarios, y la técnica concreta con
que se han implementado.

2.1 Modelización de un diccionario

Si bien es cierto que muchas de las palabras que aparecen en un diccionario se pued
capturar automáticamente a partir de los textos etiquetados, otras muchas serán introducid
manualmente por los expertos lingüistas, con el fin de cubrir de manera exhaustiva algun
categorías de palabras poco pobladas, que constituyen el núcleo invariable de un idion
(artículos, preposiciones, conjunciones, etc.), o alguna terminología particular correspondien
a un determinado ámbito de aplicación. Sin embargo, cuando nos enfrentamos a lenguaj
naturales que presentan un paradigma de inflexión de gran complejidad, resulta impensable q
el usuario tenga que introducir en el diccionario todas y cada una de las formas derivadas
un lema dado[3]. En lugar de esto, resulta mucho más conveniente realizar un estudio prev
que identifique los diferentes grupos de inflexión (género, número, irregularidad verbal, etc.),
manera que a la hora de introducir posteriormente un nuevo término en el diccionario, el usua
sólo tenga que hacer mención de las raíces involucradas en él y de los grupos de inflexión median
los cuales se realiza la derivación de formas desde cada una de esas raíces [Graña *et al.* 1994]

Por supuesto, una interfaz gráfica que genere temporalmente las formas correspondientes
la operación de inserción que se va a realizar, tal y como se muestra en la figura 2.1, pue
resultar de gran ayuda para el usuario al permitirle comprobar si efectivamente está realizan

[1] Normalmente denominados *tokens*.

[2] O también *scanning*.

[3] Por ejemplo, el paradigma de conjugación verbal del español puede utilizar hasta 118 formas distintas pa

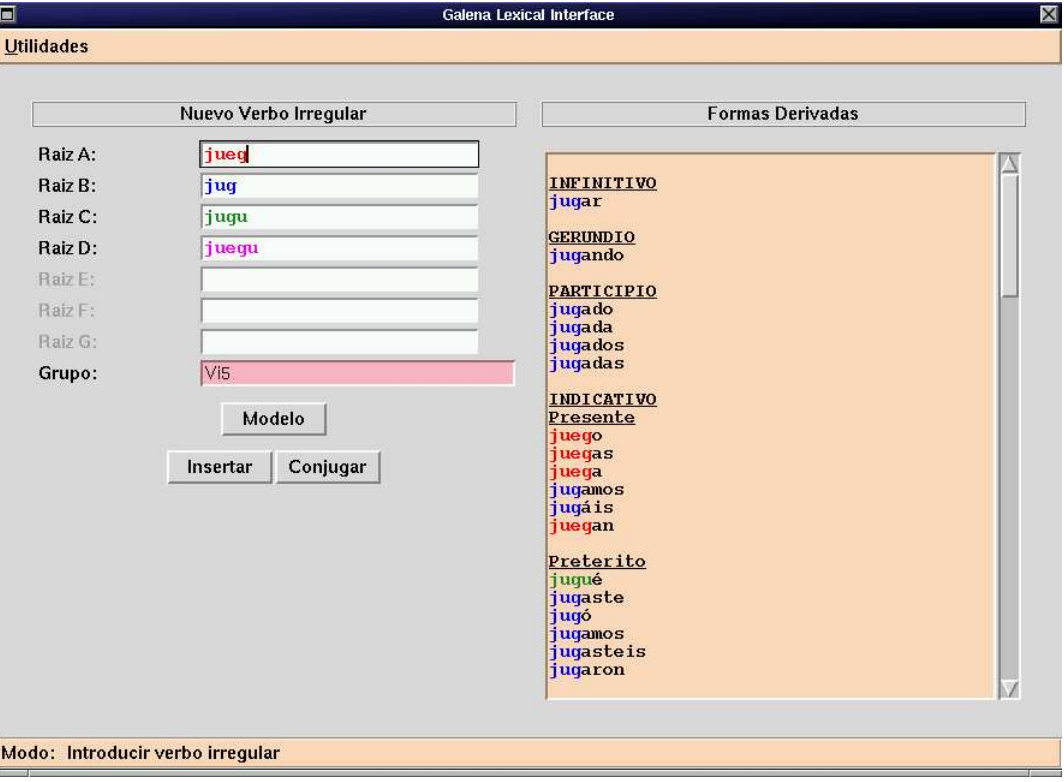


Figura 2.1: Interfaz gráfica para la introducción de términos en el diccionario

mbargo, en este punto surgen dos problemas:

El primero de ellos es un problema inherente al uso de bases de datos. Las bases de datos se presentan como herramientas válidas para el almacenamiento de gran cantidad de información estructurada, ya que resultan muy flexibles a la hora de realizar tareas de gestión y mantenimiento: las inserciones, actualizaciones, borrados y consultas se realizan mediante operaciones relacionales muy potentes que pueden implicar a uno o varios campos de información, de la misma o de diferentes tablas de datos. Pero cuando se procesan textos grandes, quizás de millones de palabras, no sólo interesa un acceso flexible a los datos, sino también un acceso muy rápido, y una base de datos no es el mejor mecanismo a la hora de recuperar este tipo de información, y menos aún si en ella residen las raíces y no las palabras concretas que aparecen en los textos. Existen mecanismos mucho más eficientes para esta última tarea, como pueden ser los autómatas finitos que describiremos en la siguiente sección.

El segundo de los problemas es que muchas aplicaciones necesitan 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. Por ejemplo, en el contexto de los modelos de Markov ocultos, que veremos más adelante, esa probabilidad representa la *probabilidad de emisión* de la palabra dentro del conjunto de

raíces se puede también utilizar aquí para generar todas las palabras y, posteriormen
a través de otro procedimiento para la estimación de las probabilidades, que veremos c
detalle en el próximo capítulo, podemos integrar toda la información necesaria, de mane
que resida junta en un mismo recurso.

Por lo tanto, en este segundo momento, nuestra visión de un diccionario o lexicón es simplemen
un fichero de texto, donde cada línea tiene el siguiente formato:

```
palabra etiqueta lema probabilidad
```

Las palabras ambiguas, es decir, con varias etiquetaciones posibles, utilizarán una línea diferen
para cada una de esas etiquetas.

Ejemplo 2.1 Sin pérdida de generalidad, las palabras podrían estar ordenadas alfabéticamen
de tal manera que, en el caso del diccionario del sistema GALENA [Vilares *et al.* 1995], el pun
donde aparece la ambigüedad de la palabra `sobre` presenta el siguiente aspecto[4]:

```
...
sobraste V2sei0 sobrar 0.00162206
sobrasteis V2pei0 sobrar 0.00377715
sobre P sobre 0.113229
sobre Scms sobre 0.00126295
sobre Vysps0 sobrar 0.0117647
sobrecarga Scfs sobrecarga 0.00383284
sobrecarga V2spm0 sobrecargar 0.00175131
sobrecarga V3spi0 sobrecargar 0.000629723
sobrecargaba Vysii0 sobrecargar 0.0026455
...
```

El diccionario del sistema GALENA presenta 291.604 palabras diferentes, con 354.0
etiquetaciones posibles. Esta última cifra es precisamente el número de líneas del fichero anteri
Para una discusión posterior, diremos ahora que la primera etiquetación de la palabra `sobr`
es decir, como preposición,

```
sobre P sobre 0.113229
```

aparece en la línea 325.611 dentro de ese fichero. Y diremos también que, en el conjunto de tod
las 291.604 palabras diferentes ordenadas alfabéticamente, la palabra `sobre` ocupa la posici
268.249.

Por supuesto, ésta tampoco es todavía la versión operativa final de un diccionario.
problema del acceso eficiente a los datos sigue aún presente, pero éste es un problema que, co
ya hemos dicho, resolveremos en la siguiente sección. Lo realmente importante ahora es obte
una versión compilada que represente de una manera más compacta todo este gran volumen
información. Esta versión compilada y su forma de operar se muestran en la figura 2.2, don
se pueden identificar cada uno de los siguientes elementos:

[4]Las etiquetas a las que se hace referencia en este y futuros capítulos pertenecen al sistema GALENA y pa

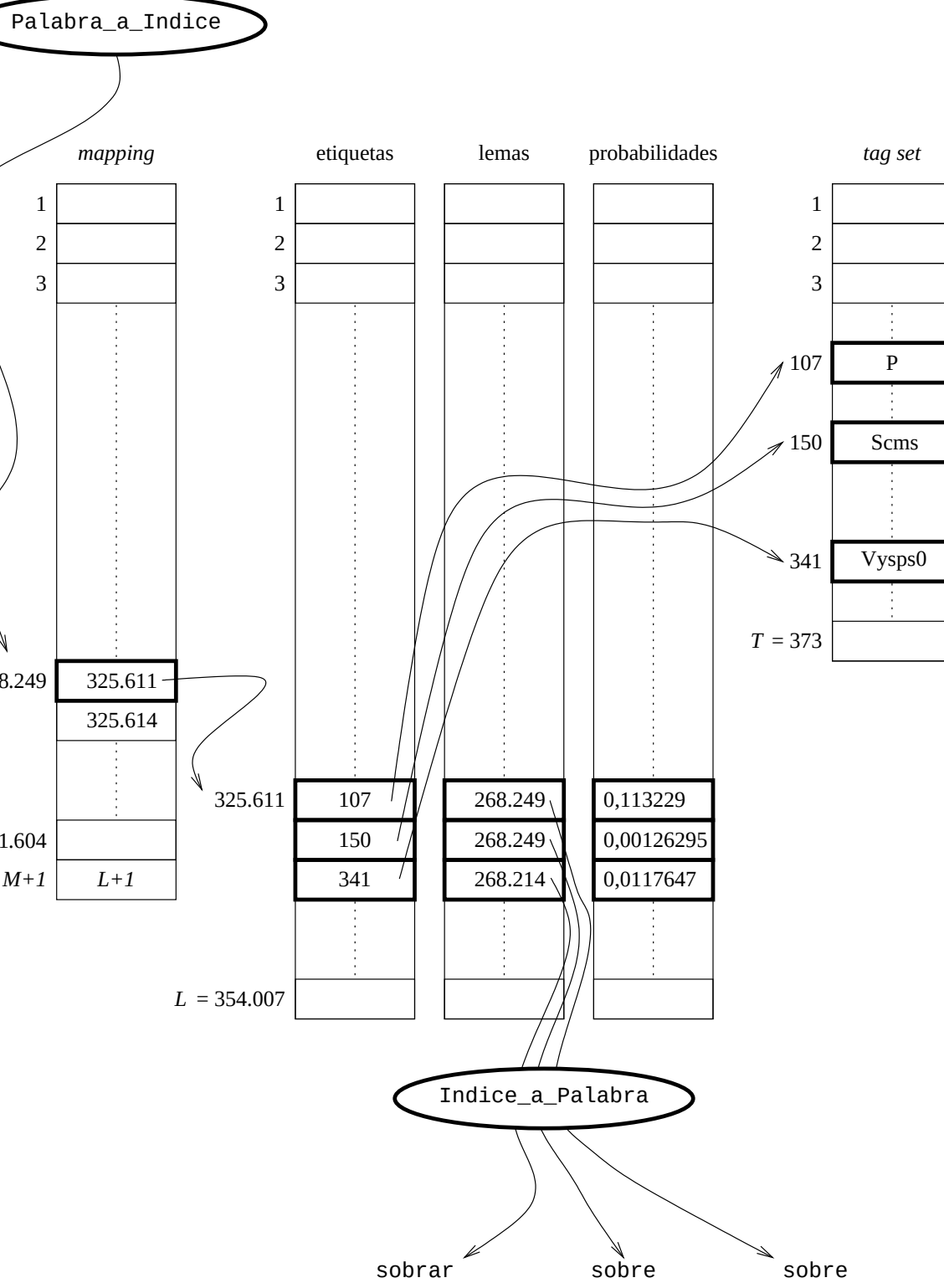


Figura 2.2: Modelización compacta de un diccionario

- Ese número sirve para indexar un tablero de correspondencia[5] de tamaño $M + 1$, q transforma la posición relativa de cada palabra en la posición absoluta dentro del lexic original. En el caso del diccionario del sistema GALENA, M es igual a 291.604, el núme de palabras distintas. En el caso de `sobre`, la posición relativa 268.249 se transforma la posición absoluta 325.611.

- Ese número sirve para indexar los tableros de *etiquetas*, *lemas* y *probabilidades*. Tod estos tableros son de tamaño L. En el caso del diccionario del sistema GALENA, L es igu a 354.007, el número de etiquetaciones distintas.

- El tablero de *etiquetas* almacena números. Una representación numérica de las etiquet es más compacta que los nombres de las etiquetas en sí. Las etiquetas originales se pued recuperar indexando con esos números el tablero del juego de etiquetas[6]. El tablero d juego de etiquetas tiene un tamaño T. En el caso del diccionario del sistema GALENA, es igual a 373, el número de etiquetas distintas.

  Dado que hemos partido de una ordenación alfabética, está garantizado que las etiquet de una misma palabra aparecen contiguas. No obstante, necesitamos saber de algu manera dónde terminan las etiquetas de una palabra dada. Para ello, es suficiente c restarle el valor de la posición absoluta de la palabra al valor de la siguiente casilla en tablero de correspondencia. En nuestro caso, la palabra `sobre` tiene $325.614 - 325.611 =$ etiquetas. Esta operación es también válida para acceder correctamente a la informaci de los tableros de *lemas* y *probabilidades*.

- El tablero de *lemas* almacena también números. Un lema es una palabra que de estar también presente en el diccionario. El número que la función `Palabra_a_Índi` obtendría para esa palabra es el número que se almacena aquí, siendo esta representaci mucho más compacta que el lema en sí. El lema se puede recuperar aplicando la funci `Índice_a_Palabra`, que explicaremos con detalle en la siguiente sección.

- El tablero de *probabilidades* almacena directamente las probabilidades. En este caso no posible realizar ninguna compactación.

Ésta es por tanto la representación más compacta que se puede diseñar para albergar to la información léxica relativa a las palabras presentes en un diccionario. Es además u representación muy flexible en el sentido de que resulta particularmente sencillo incorpor nuevos tableros, si es que se necesita algún otro tipo de información adicional. Por ejemp algunas aplicaciones de lexicografía computacional que realicen estudios sobre el uso de idioma podrían utilizar un tablero de números enteros que almacene la frecuencia de aparici de las palabras en un determinado texto. De igual manera, aquellos tableros que no se utilic se pueden eliminar, con el fin de ahorrar su espacio correspondiente[7].

Por otra parte, ideando un nuevo método de acceso o una nueva disposición de los element de los tableros, es también sencillo transformar esta estructura en un generador de formas,

[5] O tablero de *mapping*.

[6] O tablero de *tag set*.

[7] Por ejemplo, no todas las aplicaciones hacen uso del lema y de la probabilidad, pudiendo ser suficiente s

al [Vilares *et al.* 1996a].

tro aspecto más de la flexibilidad de esta representación es el que se describe a continuación. do al procesar un texto aparece una palabra que no está presente en nuestro diccionario, tamiento normal será asumir que es desconocida y posteriormente intentar asignarle la ta que describe su papel en la frase, igual que si se tratara de una palabra normal. ferencia radica en que para la palabra desconocida no podemos obtener un conjunto de etiqueta candidatas, de manera que habrá que definir una serie de categorías *as* que permitan inicializar dicho conjunto. Sin embargo, hay aplicaciones en las uede ser más conveniente suponer que todas las categorías están *cerradas* y que por esa palabra es realmente una palabra del diccionario, pero que presenta algún error gráfico que hay que corregir. Ocurre entonces que el hecho de tener la información quetación totalmente separada del mecanismo de reconocimiento de las palabras en sí a la posterior incorporación de algoritmos de corrección automática de este tipo de errores gráficos [Vilares *et al.* 1996b].

nalmente, para completar la modelización de diccionarios que hemos presentado, sólo esta detallar la implementación de las funciones `Palabra_a_Índice` e `Índice_a_Palabra`. s funciones trabajan sobre un tipo especial de autómatas finitos, los autómatas finitos cos deterministas numerados, que se describen a continuación.

Autómatas finitos acíclicos deterministas numerados

anera más eficiente de implementar analizadores léxicos[8] es quizás mediante el uso de natas finitos [Hopcroft y Ullman 1979]. La aplicación más tradicional de esta idea la nos encontrar en algunas de las fases de construcción de compiladores para los lenguajes ogramación [Aho *et al.* 1985]. El caso del procesamiento de los lenguajes naturales es ntativamente diferente, ya que surge la necesidad de representar diccionarios léxicos que as veces pueden llegar a involucrar a cientos de miles de palabras. Sin embargo, el uso de tómatas finitos para las tareas de análisis y reconocimiento de las palabras sigue siendo écnica perfectamente válida.

nición 2.1 Un *autómata finito* es una estructura algebraica que se define formalmente una 5-tupla $A = (Q, \Sigma, \delta, q_0, F)$, donde:

Q es un conjunto finito de estados,

Σ es un alfabeto finito de símbolos de entrada, es decir, el alfabeto de los caracteres que conforman las palabras,

δ es una función del tipo $Q \times \Sigma \rightarrow P(Q)$ que define las transiciones del autómata,

q_0 es el estado inicial del autómata, y

F es el subconjunto de Q al que pertenecen los estados que son finales.

ado o conjunto de estados que se alcanza mediante la transición etiquetada con el símbolo le el estado q se denota como $q.a = \delta(q, a)$. Cuando este estado es único, es decir, cuando ción δ es del tipo $Q \times \Sigma \rightarrow Q$, se dice que el autómata finito es *determinista*. □

Definición 2.2 Se denota como $L(A)$ el *lenguaje reconocido por el autómata* A, es decir, conjunto de todas las palabras w tales que $q_0.w \in F$.

Definición 2.3 Un autómata finito es *acíclico* cuando su grafo subyacente es acíclico. L autómatas finitos acíclicos reconocen lenguajes formados por conjuntos finitos de palabras.

2.2.1 Construcción del autómata

La primera estructura que nos viene a la mente para implementar un reconocedor de un conjun finito de palabras dado es un *árbol de letras*.

Ejemplo 2.2 El árbol de letras de la figura 2.3 reconoce todas las formas de los verbos ingles `discount`, `dismount`, `recount` y `remount`, es decir, el conjunto finito de palabras[9]:

`discount`	`discounted`	`discounting`	`discounts`
`dismount`	`dismounted`	`dismounting`	`dismounts`
`recount`	`recounted`	`recounting`	`recounts`
`remount`	`remounted`	`remounting`	`remounts`

Esta estructura es en sí misma un autómata finito acíclico determinista, donde el estado inic es el 0 y los estados finales aparecen marcados con un círculo más grueso.

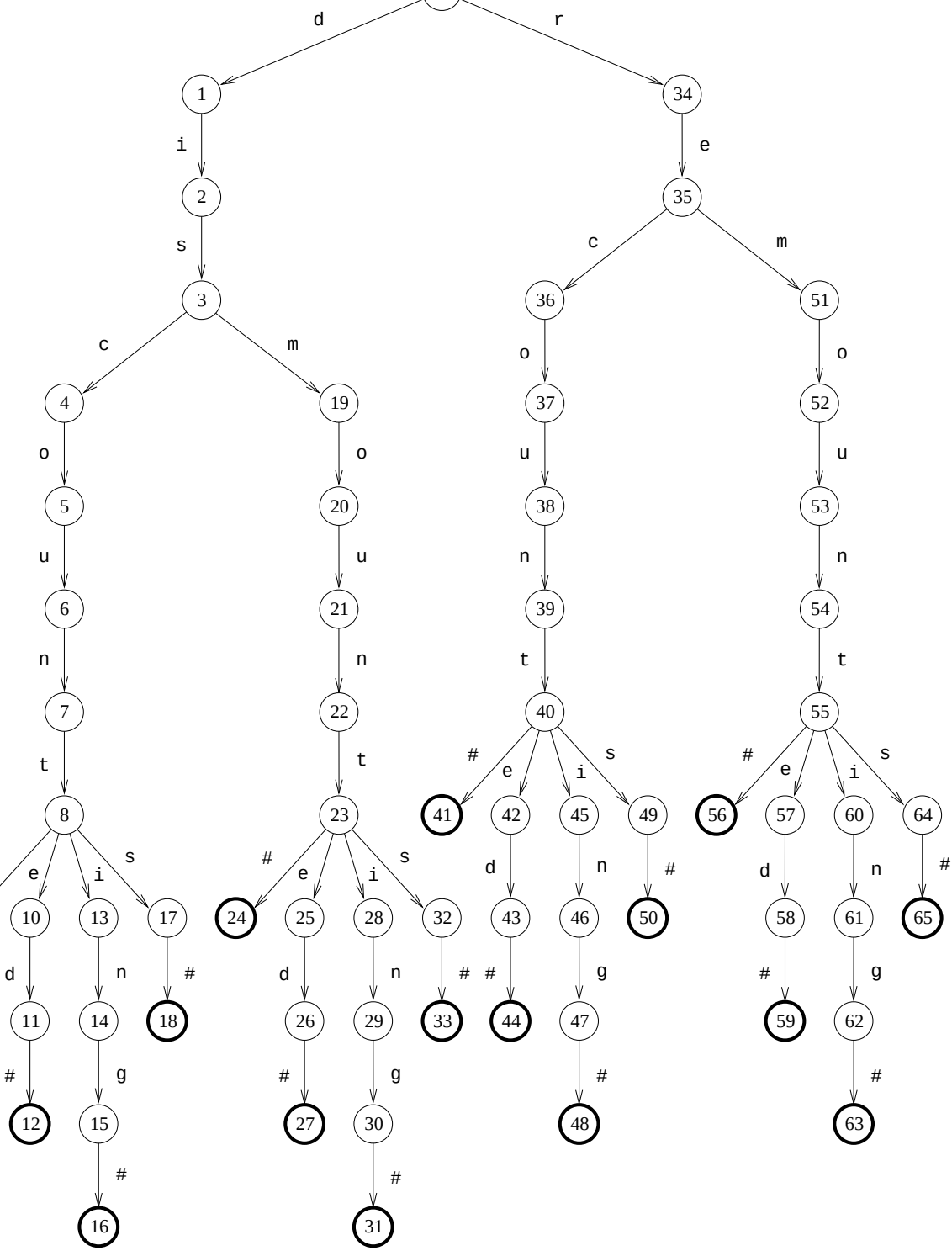
Como en todo autómata finito, la complejidad de reconocimiento de un árbol de letras lineal respecto a la longitud de la palabra a analizar, y no depende para nada ni del tamaño d diccionario, ni del tamaño de dicho autómata.

Sin embargo, los requerimientos de memoria de esta estructura de árbol pueden llegar ser elevadísimos cuando el diccionario es muy grande. Por ejemplo, el dicionario del siste GALENA necesitaría un árbol de más de un millón de nodos para reconocer las 291.6 palabras diferentes. Por tanto, en lugar de utilizar directamente esta estructura, le aplicarem un proceso de minimización para obtener otro autómata finito con menos estados y men transiciones. Los autómatas finitos tienen una propiedad que garantiza que este proceso minimización siempre se puede llevar a cabo, y que además el nuevo autómata resultante equivalente, es decir, reconoce exactamente el mismo conjunto de palabras que el autóma original [Hopcroft y Ullman 1979]. Por otra parte, en el caso de los autómatas finitos acíclic deterministas, este proceso de minimización es particularmente sencillo, tal y como veremos m adelante.

Ejemplo 2.3 El autómata finito acíclico determinista mínimo correspondiente al árbol de letr de la figura 2.3 es el que se muestra en la figura 2.4.

Por otra parte, y debido una vez más a esos mismos requerimientos de memoria, no resul conveniente construir un diccionario insertando primero todas y cada una de las palabras en árbol de letras y obteniendo después el autómata mínimo correspondiente a dicho árbol. lugar de esto, es mucho más aconsejable realizar varias etapas de inserción y minimización. P tanto, la construcción del autómata se realiza de acuerdo con los pasos básicos del siguie algoritmo.

[9] El árbol de letras que reconoce todas las formas del verbo inglés



a 2.3: Árbol de letras para las formas de los verbos `discount`, `dismount`, `recount` y `nt`

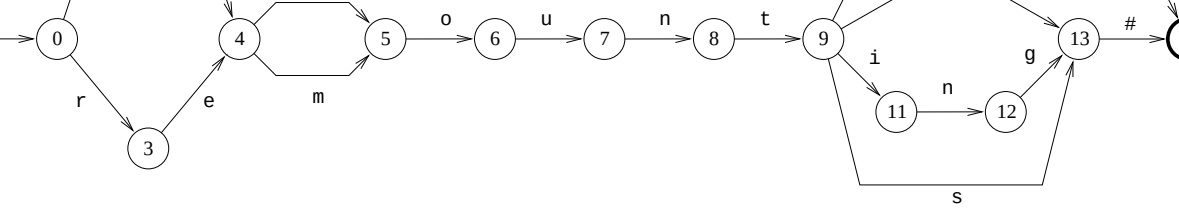


Figura 2.4: Autómata finito acíclico determinista mínimo para las formas de los verb
`discount`, `dismount`, `recount` y `remount`

Algoritmo 2.1 Algoritmo para la construcción de un autómata finito acíclico determinista
partir de un lexicón:

function *Construir_Autómata* (*Lexicón*) =
  begin
    $A \leftarrow$ *Autómata_Vacío*;

    while (*queden palabras del Lexicón por insertar*) **do**
      begin
        if (*A está lleno*) **then**
          $A \leftarrow$ *Minimizar_Autómata* (*A*);
        Insertar la siguiente palabra del Lexicón en A
      end;

    $A \leftarrow$ *Minimizar_Autómata* (*A*);
    return *A*
  end;

Eligiendo un tamaño máximo previo para el autómata, este algoritmo de construcción permi
que los consumos tanto de memoria como de tiempo por parte de los procesos de inserción
minimización sean muchísimo más moderados.

Ejemplo 2.4 Fijando el tamaño máximo en 65.536 estados[10], el proceso de construcción d
autómata finito acíclico determinista mínimo para el diccionario del sistema GALENA necesi
10 etapas de inserción y minimización. La evolución de dicho proceso se puede observar en
figura 2.5.

En este momento es importante indicar cómo se debe realizar la correcta inserción de nuev
palabras en un autómata durante el proceso de construcción del mismo. Dicha inserción se pue
llevar a cabo mediante una sencilla operación recursiva que hace uso de las transiciones que
han aparecido, con el fin de compartir los caminos ya existentes en el grafo. Pero ocurre que es
procedimiento *estándar* de inserción sólo es válido para los árboles de letras, y realmente nuest

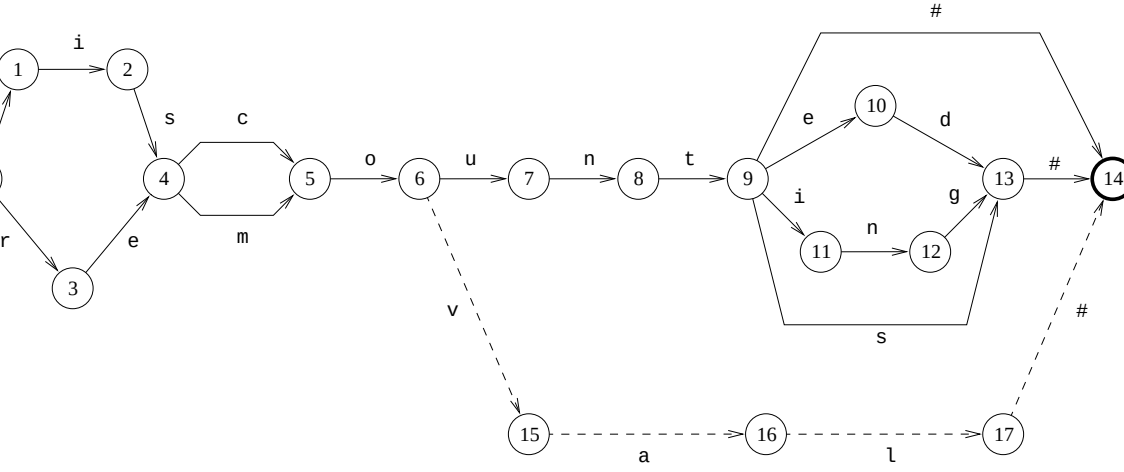
[10]La razón para elegir este número como cota superior del tamaño del autómata es que es el número máximo
enteros diferentes que se pueden almacenar en dos *bytes* de memoria, y además se ha comprobado empíricame
que resulta adecuado ya que con cotas mayores los pasos de minimización se alargan excesivamente, y con co
menores el número de palabras que se pueden insertar en cada etapa de construcción es muy pequeño y co

2	68.556	65.527	101.580	4.641	11.451
3	100.325	65.526	104.285	6.419	16.163
4	132.094	65.530	107.043	7.377	18.560
5	163.426	65.532	108.047	8.350	21.094
6	193.368	65.525	108.211	9.858	24.877
7	223.743	65.530	110.924	10.646	27.166
8	253.703	65.527	112.007	11.221	28.850
9	283.281	65.528	112.735	11.779	30.617
10	291.604	26.987	54.148	11.985	31.258

a 2.5: Evolución del proceso de construcción del autómata finito acíclico determinista
no para el diccionario del sistema GALENA

nata sólo es un árbol de letras antes de la primera minimización. Si el procedimiento de
ión estándar se aplica después de minimizar, es decir, cuando el autómata ya no es un
de letras, las inserciones pueden dar lugar a errores de construcción.

plo 2.5 Tal es el caso de la inserción que se plantea en la figura 2.6. Nuestra intención
es incorporar la palabra `removal` en el autómata de la figura 2.4. Si añadimos esta
palabra mediante un procedimiento de inserción estándar, éste aprovecha la existencia
mino $0 \xrightarrow{\texttt{r}} 3 \xrightarrow{\texttt{e}} 4 \xrightarrow{\texttt{m}} 5 \xrightarrow{\texttt{o}} 6$ para añadir el menor número posible de nuevos estados y
ciones, y la operación degenera recursivamente en la inserción de la subpalabra `val` en el
tómata que comienza en el estado 6. Aparentemente todo es correcto, pero si nos fijamos
bservaremos que de repente hemos hecho válida no sólo la palabra `removal` sino también
labras `discoval`, `dismoval` y `recoval`, lo cual no era nuestra intención, y además dichas
ras no existen en inglés. □



a 2.6: El problema de la inserción de nuevas palabras en un autómata finito acíclico
minista

solución al problema planteado pasa por diseñar un nuevo procedimiento de inserción

se cambia por el estado copia, y se continúa recursivamente la inserción de la palabra en
subautómata que comienza en este nuevo estado.

Ejemplo 2.6 Como se puede observar en la figura 2.7, esta operación de duplicado de
estado podría ser necesario realizarla más de una vez durante el proceso de inserción de u
misma palabra, ya que es posible que se alcancen varias veces estados con más de una transici
entrante, en nuestro caso, los estados 4, 5 y 6.

Aparentemente, esta operación de rotura y duplicación de determinadas partes del autóma
contradice el objetivo perseguido, que es el de obtener el autómata mínimo. Pero a medida q
se inserten más y más palabras irán apareciendo nuevos fenómenos léxicos subceptibles de s
compartidos durante la siguiente aplicación del proceso de minimización.

En cualquier caso, lo que sí es importante señalar es que este procedimiento de inserci
especial es más complejo que el procedimiento de inserción estándar. Pero ocurre que
insertamos las palabras en el autómata según su orden alfabético, entonces está garantizado q
sólo es necesario aplicar el procedimiento de inserción especial a la primera palabra que apare
después de cada minimización, pudiéndose realizar el resto de inserciones con el procedimien
estándar. La demostración de esta afirmación es sencilla de razonar: si después de una inserci
especial, al intentar insertar una nueva palabra, aparece algún estado problemático con m
de una transición entrante, necesariamente dicha inserción ha de corresponder a una palab
lexicográficamente menor a la última palabra insertada en el autómata, y dado que las palabr
se insertan en orden alfabético dicha palabra no puede aparecer.

Ejemplo 2.7 Si observamos otra vez la figura 2.7, es fácil comprobar intuitivamente que
inserción de cualquier palabra lexicográficamente mayor que `removal` caerá por debajo d
camino de líneas punteadas y no pasará por ningún estado problemático, y por tanto se pue
insertar normalmente en el autómata como si de un árbol de letras se tratara.

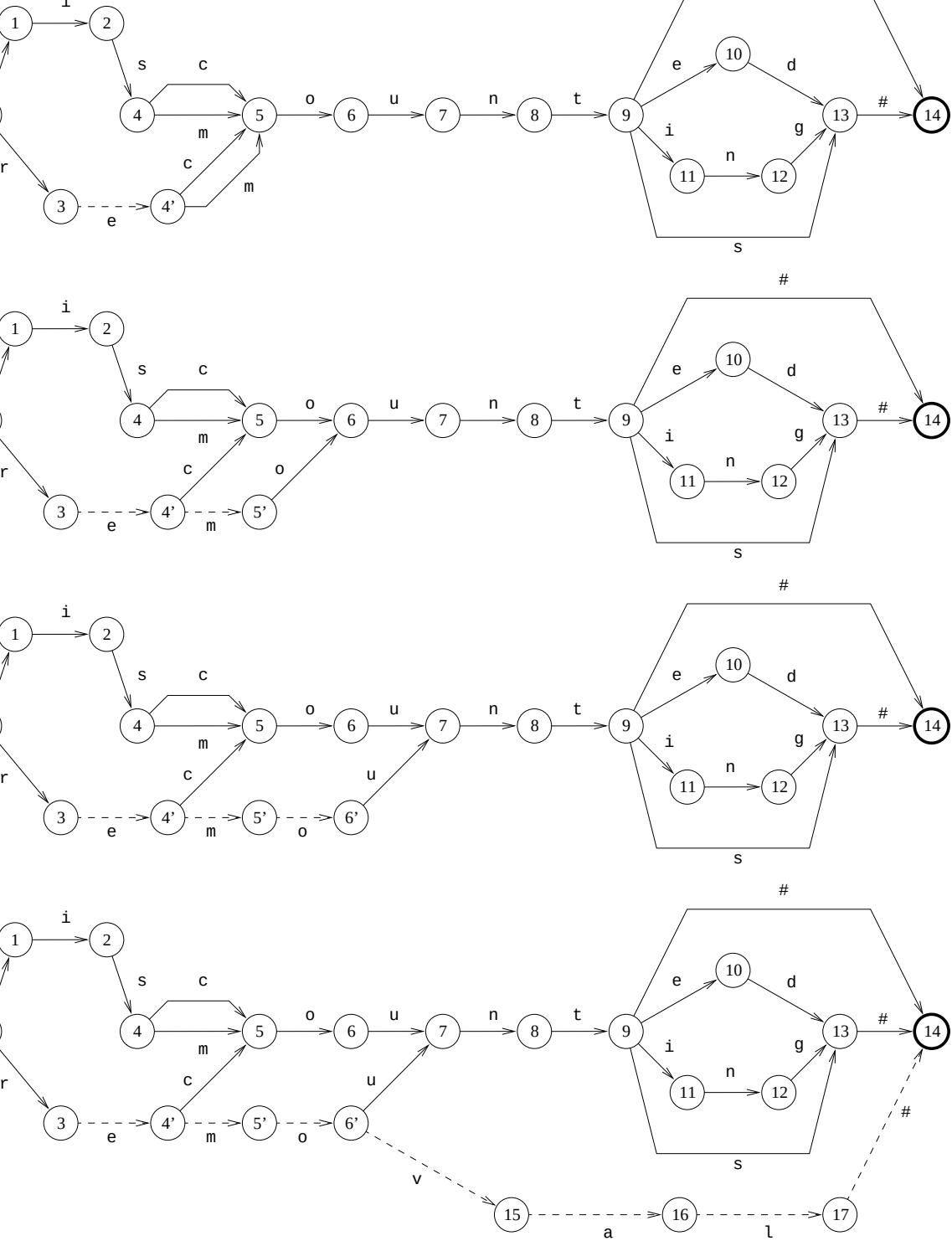
En el momento en que el autómata se llena de nuevo, se realiza una minimización, u
inserción especial, y se continúa con el proceso hasta que todas las palabras del lexicón hay
sido insertadas.

2.2.2 El algoritmo de minimización de la altura

Para definir formalmente el algoritmo de minimización, es necesario introducir primero l
siguientes conceptos.

Definición 2.4 Se dice que dos autómatas son *equivalentes* si y sólo si reconocen el misr
lenguaje. Se dice también que dos estados p y q de un autómata dado son *equivalentes* si
sólo si el subautómata que comienza con p como estado inicial y el que comienza con q s
equivalentes. O lo que es lo mismo, si para toda palabra w tal que $p.w$ es un estado fin
entonces $q.w$ es también un estado final, y viceversa.

Definición 2.5 El concepto contrario es que dos estados p y q se dicen *distinguibles* o
equivalentes si y sólo si existe una palabra w tal que $p.w$ es un estado final y $q.w$ no lo es,



ra 2.7: Inserción correcta de nuevas palabras en un autómata finito acíclico determinista

todos los autómatas que reconocen L.

Definición 2.7 Si denotamos la *altura* de un estado s como $h(s)$, entonces $h(s)$ $max\{|w|$ tal que $s.w \in F\}$. Es decir, la altura de un estado s es la longitud del cami más largo de entre todos los que empiezan en dicho estado s y terminan en alguno de los estad finales. Esta función de altura establece una partición Π sobre Q, de tal manera que Π_i denc el conjunto de todos los estados de altura i. Diremos que el conjunto Π_i es *distinguible* si tod sus estados son *distinguibles*, es decir, si no contiene ningún par de estados equivalentes.

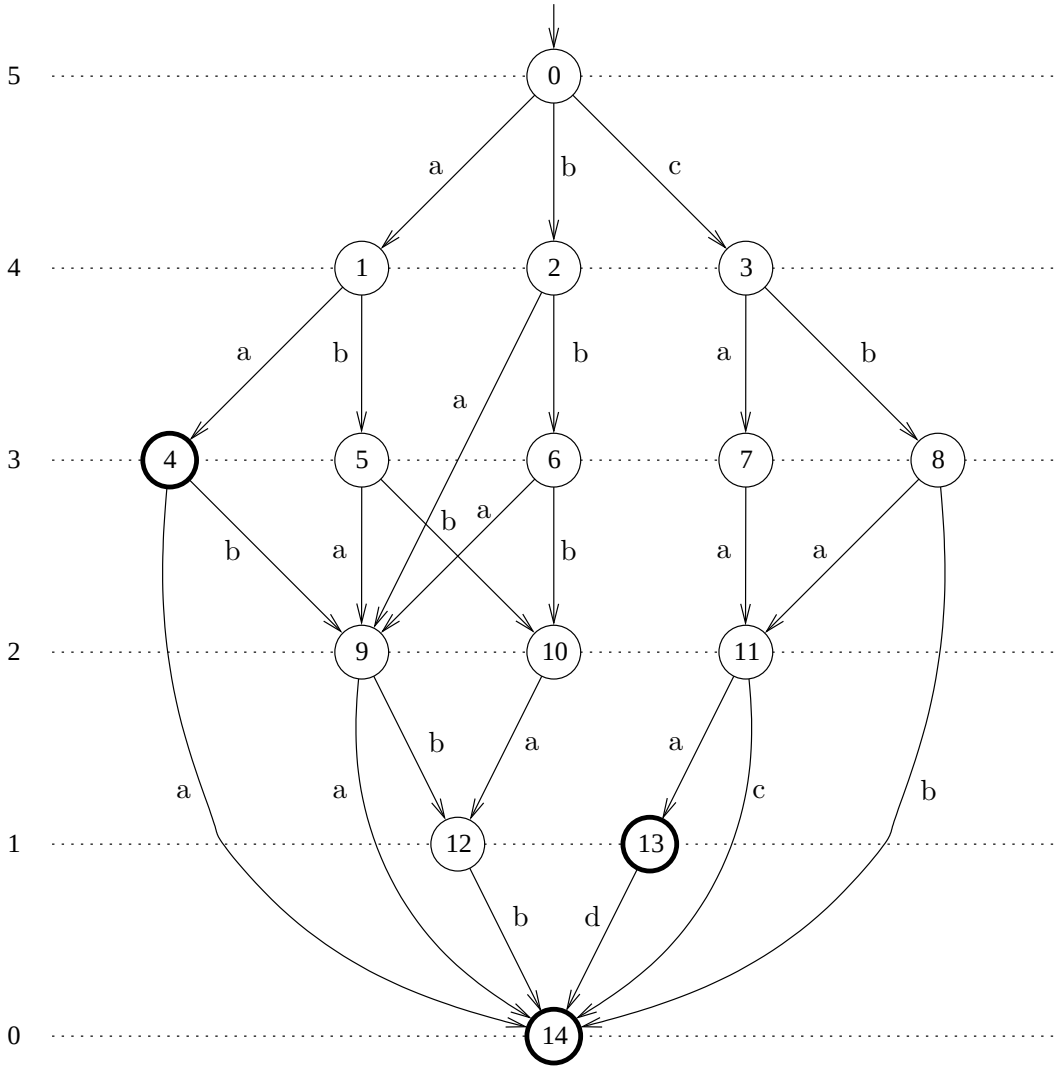


Figura 2.8: Un autómata finito acíclico determinista no mínimo

Ejemplo 2.8 La figura 2.8 muestra un autómata finito acíclico determinista, donde el esta inicial es el 0 y los estados finales son el 4, el 13 y el 14, y que por tanto reconoce el lengua

ando la transición $1 \xrightarrow{b} 5$ por $1 \xrightarrow{b} 6$. □

na vez que ha sido definida la altura de un estado, estamos en condiciones de introducir el nte teorema.

ema 2.1 Si todos los Π_j con $j < i$ son distinguibles, entonces dos estados p y q ecientes a Π_i son equivalentes si y sólo si para cualquier letra $a \in \Sigma$ la igualdad $p.a = q.a$ ifica.

emostración: Si dicha igualdad se verifica, los estados son equivalentes, por la propia ción de estado equivalente. Así que para la demostración de esta propiedad, basta estudiar sos en los que dicha igualdad no se verifica, y comprobar que efectivamente los estados n equivalentes. Entonces, dados p y q dos estados de Π_i con $p.a \neq q.a$, tenemos dos lidades:

Cuando $p.a$ y $q.a$ pertenecen al mismo Π_j. Dado que $j < i$, que el autómata es acíclico, y que por hipótesis todos los estados de Π_j son distinguibles, entonces p y q también son distinguibles.

Cuando $p.a \in \Pi_j$ y $q.a \in \Pi_k$, $j \neq k$. Supongamos sin pérdida de generalidad que $k < j$. Entonces, por la definición de Π, existe una palabra w de longitud j tal que $(p.a).w$ es final y $(q.a).w$ no lo es. Por tanto, los estados $p.a$ y $q.a$ son distinguibles, y entonces p y q también son distinguibles.

esultado se conoce también como la *propiedad de la altura.* □

algoritmo de minimización se puede deducir ahora de manera sencilla a partir de la edad de la altura [Revuz 1992].

ritmo 2.2 Los pasos básicos del algoritmo de minimización de un autómata finito acíclico minista son los siguientes:

```
procedure Minimizar_Autómata (Autómata) =
  begin
    Calcular Π;

    for i ← 0 to h(q0) do
      begin
        Ordenar los estados de Π_i según sus transiciones;
        Colapsar los estados equivalentes
      end
  end;
```

ro creamos la partición del conjunto de estados según su altura. Esta partición se puede ar mediante un recorrido recursivo estándar sobre el autómata, cuya complejidad temporal t), donde t es el número de transiciones del autómata. No obstante, si el autómata no es ool, se puede ganar algo de velocidad mediante una marca que indique si la altura de un ha sido ya calculada o no. De igual manera, los estados no útiles no tendrán ninguna asignada y ya se pueden eliminar durante este recorrido. Posteriormente, se procesa cada e los Π_i, desde $i = 0$ hasta la altura del estado inicial, ordenando los estados según sus

$$\mathcal{O}(t + \sum_{i=0} f(|\Pi_i|))$$

que en el caso de los autómatas finitos acíclicos deterministas es menor que la complejid del algoritmo general de Hopcroft para cualquier tipo de autómatas finitos determinista $\mathcal{O}(n \times \log n)$, donde n es el número de estados [Hopcroft y Ullman 1979].

2.2.3 Asignación y uso de los números de indexación

Hemos visto que los autómatas finitos acíclicos deterministas son la estructura más compacta q se puede diseñar para el reconocimiento de un conjunto finito de palabras dado. Los resultad de compresión son excelentes, y el tiempo de reconocimiento es lineal respecto a la longitud la palabra a analizar, y no depende ni del tamaño del diccionario, ni del tamaño del autómat

Sin embargo, si detenemos el proceso de construcción del autómata en este punt dispondremos de una estructura que solamente es capaz de indicarnos si una palabra da pertenece o no al diccionario, y esto no es suficiente para el esquema de modelización diccionarios que hemos desarrollado anteriormente. Dicha modelización necesita un mecanis que transfome cada palabra en una clave numérica unívoca, y viceversa.

Definición 2.8 Esta transformación se puede llevar a cabo fácilmente si el autómata incorpor para cada estado, un entero que indique el número de palabras que se pueden aceptar median el subautómata que comienza en ese estado [Lucchesi y Kowaltowski 1993]. Nos referiremos este autómata como *autómata finito acíclico determinista numerado*.

Ejemplo 2.9 La versión numerada del autómata de la figura 2.4 es la que se muestra en figura 2.9.

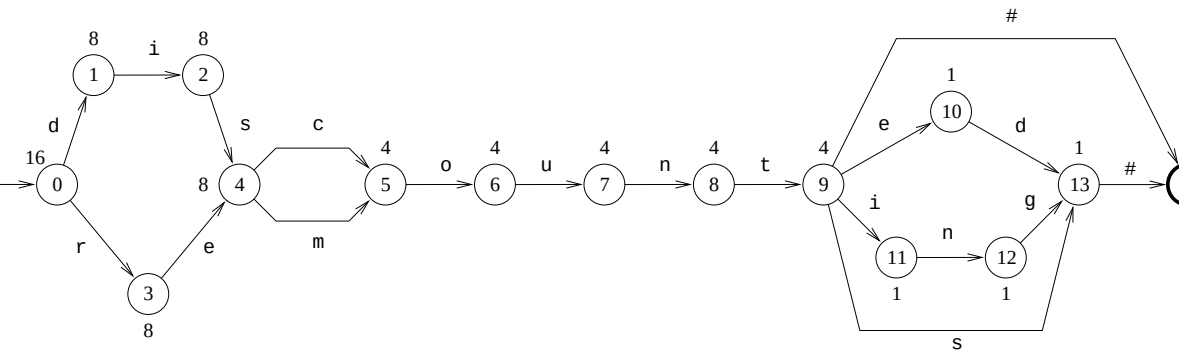


Figura 2.9: Autómata finito acíclico determinista mínimo numerado para las formas de los verb `discount`, `dismount`, `recount` y `remount`

La asignación de los números de indexación a cada estado se puede realizar mediante sencillo recorrido recursivo sobre el autómata, una vez que éste ha sido correctamente construi y minimizado. Por tanto, la versión definitiva de la función `Construir_Autómata` es la que

```
begin
    A ← Autómata_Vacío;

    while (queden palabras del Lexicón por insertar) do
        if (A está lleno) then
            begin
                A ← Minimizar_Autómata (A);
                Inserción especial de la siguiente palabra del Lexicón en A
            end
        else
            Inserción estándar de la siguiente palabra del Lexicón en A;

    A ← Minimizar_Autómata (A);
    Asignar los números de indexación a los estados de A;
    return A
end;
```

uevo proceso de construcción del autómata completa y substituye al visto anteriormente algoritmo 2.1. □

na vez que el autómata ha sido numerado, podemos ya escribir las funciones `ra_a_Índice` e `Índice_a_Palabra`, que son las que realizan la correspondencia uno a uno las palabras del diccionario y los números 1 a M, donde M es el número de total de ras distintas aceptadas por el autómata.

ritmo 2.4 Pseudo-código de la función `Palabra_a_Índice`:

```
function Palabra_a_Índice (Palabra) =
  begin
    Índice ← 1;
    Estado_Actual ← Estado_Inicial;

    for i ← 1 to Longitud (Palabra) do
      if (Transición_Válida (Estado_Actual, Palabra[i])) then
        begin
          for c ← Primera_Letra to Predecesor (Palabra[i]) do
            if (Transición_Válida (Estado_Actual, c)) then
              Índice ← Índice + Estado_Actual[c].Número;
          Estado_Actual ← Estado_Actual[Palabra[i]];
        end
      else
        return palabra desconocida;

    if (Es_Estado_Final (Estado_Actual)) then
      return Índice
    else
      return palabra desconocida
```

precedentes a la transición utilizada. Si después de procesar todos los caracteres de la palab
llegamos al estado final, entonces el índice contendrá la clave numérica de la palabra. En ca
contrario, la palabra no pertenece al lexicón que se está manejando. Como consecuencia, el val
del índice no es correcto, y en su lugar la función devuelve un valor que indica que la palabra
desconocida.

Algoritmo 2.5 Pseudo-código de la función `Índice_a_Palabra`:

```
function Índice_a_Palabra (Índice) =
  begin
    Estado_Actual ← Estado_Inicial;
    Número ← Índice;
    Palabra ← Palabra_Vacía;
    i ← 1;

    repeat
      for c ← Primera_Letra to Última_Letra do
        if (Transición_Válida (Estado_Actual, c)) then
          begin
            Estado_Auxiliar ← Estado_Actual[c];
            if (Número > Estado_Auxiliar.Número) then
              Número ← Número − Estado_Auxiliar.Número
            else
              begin
                Palabra[i] ← c;
                i ← i + 1;
                Estado_Actual ← Estado_Auxiliar;
                if (Es_Estado_Final (Estado_Actual)) then
                  Número ← Número − 1;
                exit forloop
              end
          end
    until (Número = 0);

    return Palabra
  end;
```

Esta función parte del índice y realiza las operaciones análogas a las del algoritmo 2.4, pa
deducir cuáles son las transiciones que dan lugar a ese índice, y a partir de esas transicion
obtiene las letras que forman la palabra que se está buscando.

Ejemplo 2.10 En el caso del autómata numerado de la figura 2.9, la correspondencia individu
de cada palabra con su índice es como sigue:

```
   1 <-> discount   2 <-> discounted    3 <-> discounting   4 <-> discounts
   5 <-> dismount   6 <-> dismounted    7 <-> dismounting   8 <-> dismounts
   9 <-> recount   10 <-> recounted    11 <-> recounting   12 <-> recounts
```

os requerimientos de almacenamiento y la complejidad temporal extra en el proceso conocimiento implicada por la incorporación de los números de indexación resulta sta. Así que finalmente sólo nos queda hacer una pequeña referencia a las prestaciones estro analizador léxico: el autómata finito acíclico determinista numerado correspondiente cionario GALENA consta de 11.985 estados y 31.258 transiciones; el tamaño del fichero lado correspondiente a dicho autómata es de 3.466.121 *bytes*; el tiempo de compilación aproximadamente 29 segundos; y la velocidad de reconocimiento en una máquina con un sador Pentium II a 300 MHz bajo sistema operativo Linux es de aproximadamente 40.000 ras por segundo.

Algoritmos de construcción incrementales

hemos visto, los métodos tradicionales para la construcción de autómatas finitos acíclicos ministas mínimos a partir de un conjunto de palabras consisten en combinar dos fases de ción: la construcción de un árbol o de un autómata parcial, y su posterior minimización. mbargo, existen métodos de construcción incremental, capaces de realizar las operaciones de nización en línea, es decir, al mismo tiempo que se realizan las inserciones de las palabras en ómata [Daciuk *et al.* 2000]. Estos métodos son mucho más rápidos, y sus requerimientos de oria son también significativamente menores que los de los métodos descritos anteriormente. ara construir un autómata palabra a palabra de manera incremental es necesario combinar ceso de minimización con el proceso de inserción de nuevas palabras. Por tanto, hay dos ntas cruciales que hay que responder:

¿Qué estados, o clases de equivalencia de estados, son subceptibles de cambiar cuando se insertan nuevas palabras en el autómata?

¿Existe alguna manera de minimizar el número de estados que es necesario cambiar durante la inserción de una palabra?

ya sabemos, si las palabras están ordenadas lexicográficamente, cuando se añade una , sólo pueden cambiar los estados que se atraviesan al aceptar la palabra insertada amente. El resto del autómata permanece inalterado, ya que la nueva palabra:

O bien comienza con un símbolo diferente del primer símbolo de todas las palabras ya presentes en el autómata, con lo cual el símbolo inicial de la nueva palabra es situado lexicográficamente después de todos esos símbolos.

O bien comparte algunos de los símbolos iniciales de la palabra añadida previamente. En este caso, el algoritmo localiza el último estado en el camino de ese prefijo común y crea una nueva rama desde ese estado. Esto es debido a que el símbolo que etiqueta la nueva transición será lexicográficamente mayor que los símbolos del resto de transiciones salientes que ya existen en ese estado.

anto, cuando la palabra previa es prefijo de la nueva palabra a insertar, los únicos estados ueden cambiar son los estados del camino de reconocimiento de la palabra previa que no en el camino del prefijo común. La nueva palabra puede tener una finalización igual a la ras palabras ya insertadas, lo cual implica la necesidad de crear enlaces a algunas partes

el autómata mínimo en cada instante.

Algoritmo 2.6 El algoritmo de construcción incremental de un autómata finito acícli determinista, a partir de un conjunto de palabras ordenado lexicográficamente, cons básicamente de dos funciones: la función principal *Construir_Autómata_Incremental* y la funci *Reemplazar_o_Registrar*. Los pasos de la función principal son los siguientes:

```
function Construir_Autómata_Incremental (Lexicón) =
  begin
    Registro ← ∅;

    while (queden palabras del Lexicón por insertar) do
      begin
        Palabra ← siguiente palabra del Lexicón en orden lexicográfico;
        Prefijo_Común ← Prefijo_Común (Palabra);
        Último_Estado ← q0.Prefijo_Común;
        Sufijo_Actual ← Palabra[(Longitud (Prefijo_Común) + 1) ... Longitud (Palabra
        if (Tiene_Hijos (Último_Estado)) then
          Registro ← Reemplazar_o_Registrar (Último_Estado, Registro);
        Añadir_Sufijo (Último_Estado, Sufijo_Actual);
      end;

    Registro ← Reemplazar_o_Registrar (q0, Registro);
    return Registro
  end;
```

El esquema de la función *Reemplazar_o_Registrar* es como sigue:

```
function Reemplazar_o_Registrar (Estado, Registro) =
  begin
    Hijo ← Último_Hijo (Estado);
    if (Tiene_Hijos (Hijo)) then
      Registro ← Reemplazar_o_Registrar (Hijo, Registro);
    if (∃ q ∈ Q : q ∈ Registro ∧ q ≡ Hijo) then
      begin
        Último_Hijo (Estado) ← q;
        Eliminar (Hijo)
      end
    else
      Registro ← Registro ∪ Hijo;
    return Registro
  end;
```

El lazo principal del algoritmo lee las palabras y establece qué parte de cada palabra está en el autómata, es decir, el *Prefijo_Común*, y qué parte no está, es decir, el *Sufijo_Actual*. U paso importante es determinar cuál es el último estado en el camino del prefijo común, que en

a de nuevos estados capaz de reconocer el *Sufijo_Actual*.

función *Prefijo_Común* busca el prefijo más largo de la palabra a insertar que es prefijo guna palabra ya insertada. La función *Añadir_Sufijo* crea una nueva rama que extiende ómata, la cual representa el sufijo no encontrado de la palabra que se va a insertar. La n *Último_Hijo* devuelve una referencia al estado alcanzado por la última transición en orden gráfico que sale del estado argumento. Dado que los datos de entrada están ordenados, transición es la transición añadida más recientemente en el estado argumento, durante la ión de la palabra previa. La función *Tiene_Hijos* es cierta si y sólo si el estado argumento transiciones salientes.

función *Reemplazar_o_Registrar* trabaja efectivamente sobre el último hijo del estado mento. Dicho argumento es el último estado del camino del prefijo común, o bien el estado del autómata en la última llamada de la función principal. Es necesario que el estado mento cambie su última transición en aquellos casos en los que el hijo va a ser reemplazado tro estado equivalente ya registrado. En primer lugar, la función se llama recursivamente isma hasta alcanzar el final del camino de la palabra insertada previamente. Nótese que vez que se encuentra un estado con más de un hijo, siempre se elige el último. La longitud da de las palabras garantiza el fin de la recursividad. Por tanto, al volver de cada llamada iva, se comprueba si ya existe en el registro algún estado equivalente al estado actual. Si el estado actual se reemplaza por el estado equivalente encontrado en el registro. Si no, el o actual se registra como representante de una nueva clase de equivalencia. Es importante car que la función *Reemplazar_o_Registrar* sólo procesa estados pertenecientes al camino palabra insertada previamente, y que esos estados no se vuelven a procesar. □

urante la construcción, los estados del autómata o están en el registro o están en el camino última palabra insertada. Todos los estados del registro son estados que formarán parte del nata mínimo resultante. Así pues, el número de estados durante la construcción es siempre r que el número de estados del autómata resultante más la longitud de la palabra más Por tanto, la complejidad espacial del algoritmo es $\mathcal{O}(n)$, es decir, la cantidad de memoria recisa el algoritmo es proporcional a n, el número final de estados del autómata mínimo. cto al tiempo de ejecución, éste es dependiente de la estructura de datos implementada para ar las búsquedas de estados equivalentes y las inserciones de nuevos estados representantes registro. Utilizando criterios de búsqueda e inserción basados en el número de transiciones tes de los estados, en sus alturas y en sus números de indexación, los cuales se pueden én calcular dinámicamente, la complejidad temporal se puede rebajar hasta $\mathcal{O}(\log n)$.

on este algoritmo incremental, el tiempo de construcción del autómata correspondiente al nario del sistema GALENA se reduce considerablemente: de 29 segundos a 2,5 segundos, a máquina con un procesador Pentium II a 300 MHz bajo sistema operativo Linux. La oración de la información relativa a las etiquetas, lemas y probabilidades consume un o extra aproximado de 4,5 segundos, lo que hace un tiempo de compilación total de 7 dos.

n el mismo trabajo, los autores proponen también un método incremental para la rucción de autómatas finitos acíclicos deterministas mínimos a partir de conjuntos de ras no ordenados [Daciuk *et al.* 2000]. Este método se apoya también en la clonación licación de los estados que se van volviendo conflictivos a medida que aparecen las nuevas ras. Por esta razón, la construcción incremental a partir de datos desordenados es más

2.3 Otros métodos de análisis léxico

Los procesos productivos o derivativos presentes en todos los idiomas constituyen una gran fuen de complicaciones para el análisis morfológico. Debido a ellos, siempre existirán multitud palabras que no estarán incluidas en un diccionario morfológico estático, sin importar lo gran o preciso que éste sea. En otras palabras, se podría decir que el tamaño del lexicón de cualqui lengua es virtualmente infinito. Inevitablemente, éste es un problema al que deben enfrentar los etiquetadores. Tal y como veremos con detalle en los capítulos siguientes, todo etiquetad debe incluir un módulo de tratamiento de palabras desconocidas, independientemente de l mecanismos que tenga integrados para el análisis léxico y para el acceso al diccionario.

Pero efectivamente, muchas de las palabras desconocidas serán formas derivadas a par de una raíz y de una serie de reglas de inflexión. Por ejemplo, es muy probable que la palab `reanalizable` no esté incluida en un diccionario dado, pero parece claro que esta palabra es par del lenguaje, en el sentido de que puede ser utilizada, es comprensible y podría ser fácilmen deducida a partir de la palabra `analizar`[11] y de reglas de derivación tales como la incorporaci de prefijos o la transformación de los verbos en sus respectivos adjetivos de calidad.

Además, no siempre es necesario manejar una información tan particular para cada palab concreta como puede ser su frecuencia o su probabilidad. Fuera de los paradigmas aproximación estocástica al NLP, la mayoría de las aplicaciones utilizan sólo la etiqueta, como mucho la etiqueta y el lema. En esta última sección, por tanto, enumeramos brevemen otros posibles métodos de análisis léxico que han sido desarrollados.

La aproximación quizás más importante, y que ha servido de base para multitud de trabaj relacionados con la construcción automática de analizadores léxicos a partir de la especificaci de las reglas morfológicas de flexión y derivación de una determinada lengua, es el modelo *morfología de dos niveles*[12] [Koskenniemi 1983]. Este modelo se basa en la distinción tradicion que hacen los lingüistas entre, por un lado, el conjunto teórico de morfemas y el orden en el cu pueden ocurrir, y por otro lado, las formas alternativas que presentan dichos morfemas dent del contexto fonológico en el que aparecen. Esto establece la diferencia entre lo que se denomi la *cadena superficial* del lenguaje (por ejemplo, la palabra `quiero`), y su correspondiente *cade léxica* o *teórica* (en este caso, `quero` = `quer` + `o`). Por tanto, `quer` y `quier` se consider alomorfos o formas alternativas del mismo morfema. El modelo de Koskenniemi es de dos nive en el sentido de que cualquier palabra se puede representar mediante una correspondencia direc letra a letra entre su cadena de superficie y la cadena teórica subyacente.

Ejemplo 2.11 Para la palabra inglesa `skies`, podemos asumir que existe una raíz `sky` una terminación de plural `-es`, salvo que, como ocurre en inglés con todos los sustantiv terminados en `y`, la `y` se realiza como una `i`, y por tanto las cadenas teórica y superficial sería respectivamente:

```
s k y - e s
s k i 0 e s
```

En este ejemplo, el `-` inidica el límite de morfemas, y el `0` un caracter nulo o no existente.

[11] Suponiendo que ésta sí está presente en el lexicón.

No es cierto que cualquier `y` se corresponda con una `i`. Esto sólo es válido para el proceso de formación de plural que estamos considerando.

En este caso concreto, la `y` se corresponde sólo con una `i`, y con ninguna otra letra, ni siquiera consigo misma.

delo de morfología de dos niveles incorpora un componente más que permite expresar este e restricciones mediante reglas de transformación. Por ejemplo, esta relación *si y sólo si* a denotarse mediante una regla de doble flecha como la siguiente:

```
i <=> _ -: e: s: ;
```

njunto dado de reglas de este tipo se puede transformar en un traductor de estado finito tunen y Beesley 1992]. Un *traductor de estado finito* no es más que un autómata finito n lugar de caracteres simples, incorpora correspondencias letra a letra como etiquetas de ansiciones, permitiendo transformar la cadena superficial en la cadena teórica, y viceversa.

plo 2.12 La figura 2.10 representa un traductor de estado finito capaz de asociar cada e las formas verbales del verbo `leave` con su lema y su correspondiente etiqueta, es decir: con `leave+VB`, `leaves` con `leave+VBZ` y `left` con `leave+VBD`[13]. □

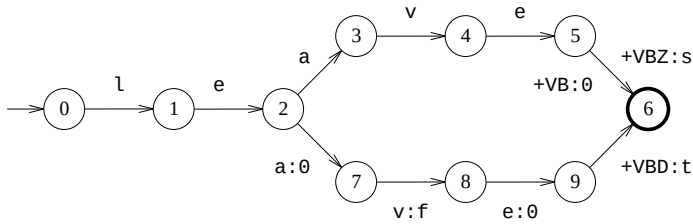


Figura 2.10: Traductor de estado finito para el verbo `leave`

ra cada palabra, existe en el traductor un camino que contiene la forma flexionada de dena superficial y el par lema-etiqueta de la cadena teórica. Dicho camino se puede trar utilizando como clave de entrada cualquiera de las dos cadenas. Esta característica ocesamiento bidireccional de los traductores finitos hace que el modelo de morfología de veles resulte válido tanto para el análisis morfológico, como para la generación de formas. n embargo, para los traductores, la noción de determinismo normalmente tiene sentido considera sólo una de las direcciones de aplicación. Si un traductor presenta este tipo de minismo unidireccional, se dice que es o bien *secuencial por arriba*[14] si es determinista en la ión de la cadena superficial, o bien *secuencial por abajo*[15] si es determinista en la dirección cadena teórica. Por ejemplo, el traductor de la figura 2.10 es secuencial por arriba, pero uencial por abajo, ya que desde el estado 2 con el símbolo de entrada `a` se puede transitar estados diferentes, el 3 y el 7.

notación de las etiquetas es una adaptación del juego de etiquetas utilizado en el corpus BROWN s y Kučera 1982]: `VB` significa verbo en infinitivo, `VBZ` es verbo en tercera persona del singular y `VBD` o en tiempo pasado.

quential upward.

todas las ambigüedades locales en esa dirección se pueden resolver a sí mismas en un núme finito de pasos, se dice que el traductor es *secuenciable*, es decir, se puede construir un traduct secuencial equivalente en la misma dirección [Roche y Schabes 1995, Mohri 1995].

Ejemplo 2.13 La ambigüedad local del traductor de la figura 2.10 se resuelve en el estado Por tanto, se puede incorporar un camino desde el estado inicial hasta el estado 5 que transforn la secuencia `ave` en la cadena vacía, lo cual, tal y como se muestra en la figura 2.11, permi construir un traductor secuencial por abajo, pero sólo en esa dirección, ya que ahora existen d arcos con `a` en la cadena superior partiendo del estado 5.

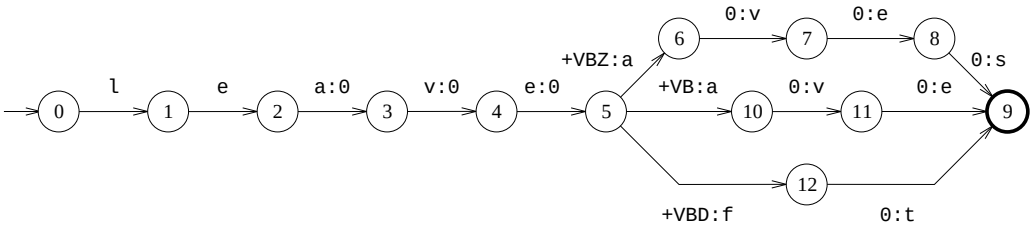


Figura 2.11: Traductor de estado finito secuencial por abajo para el verbo `leave`

Los traductores secuenciales se pueden extender para permitir que emitan desde los estad finales una cadena de salida adicional (*traductores subsecuenciales*) o un número finito p cadenas de salida (*traductores p-subsecuenciales*). Estos últimos son los que permiten realiz el manejo de las ambigüedades léxicas presentes en los lenguajes naturales [Mohri 1995].

Y todos ellos, mediante la incorporación de una información numérica extra en cada estad pueden pasar de ser simples conversores de una cadena en otra[16] a incorporar también funcionalidad de conversión de una cadena en un peso numérico[17]. Típicamente, ese nue dato numérico podría ser una probabilidad logarítmica, de tal manera que la probabilidad emisión de una palabra dada se calcularía simplemente sumando el dato de todos los estad por los que pasa el camino de procesamiento de dicha palabra en el traductor. Ésta es la forn de integrar probabilidades de emisión para las palabras en los traductores finitos [Mohri 199

De entre los múltiples trabajos basados en el modelo de Koskenniemi, destaca la herramien Mmorph [Russell y Petitpierre 1995], un compilador de reglas morfológicas de dos niveles libre distribución, desarrollado en el marco del proyecto Multext[18].

Otros métodos de diseño de analizadores léxicos, que también han sido aplicad con éxito al idioma español, incluyen las aproximaciones basadas en unificaci [Moreno 1991, Moreno y Goñi 1995, González Collar *et al.* 1995] y las basadas en árboles decisión [Triviño 1995, Triviño y Morales 2000].

[16] Traductores *string-to-string*.

[17] Traductores *string-to-weight*.

[18] *Multilingual Text Tools and Corpora* es una iniciativa de la Comisión Europea (bajo los program *Investigación e Ingeniería Lingüística*, *Copernicus*, y *Lenguas Regionales y Minoritarias*), de la Fundaci Científica Nacional de los Estados Unidos, del Fondo Francófono para la Investigación, del CNRS (Centro Nacio francés para la Investigación Científica) y de la Universidad de Provenza. Multext comprende una serie proyectos cuyos objetivos son el desarrollo de estándares y especificaciones para la codificación y el procesamien de textos, y para el desarrollo de herramientas, *corpora* y recursos lingüísticos bajo esos estándares. Los traba han sido realizados también en estrecha colaboración con los proyectos Eagles y Crater, e incluyen herramien

Parte II

Sistemas de etiquetación tradicionales

Capítulo 3

Modelos de Markov ocultos (HMM,s)

Cuando se abordó inicialmente el problema de la etiquetación de textos en lenguaje natur
se comenzó diseñando manualmente reglas de etiquetación que intentaban describir
comportamiento del lenguaje humano. Sin embargo, en el momento en el que aparec
disponibles grandes cantidades de textos electrónicos, muchos de ellos incluso etiquetados, l
aproximaciones estocásticas adquieren gran importancia en el proceso de etiquetación. S
muchas las ventajas de los etiquetadores estocásticos frente a los etiquetadores construid
manualmente. Además del hecho de que evitan la laboriosa construcción manual de las regl
también capturan automáticamente información útil que el hombre podría no apreciar.

En este capítulo estudiaremos un método estocástico comúnmente denominado aproximaci
con modelos de Markov ocultos o HMM,s[1]. A pesar de sus limitaciones, los HMM,s y s
variantes son todavía la técnica más ampliamente utilizada en el proceso de etiquetación d
lenguaje natural, y son generalmente referenciados como una de las técnicas más exitosas pa
dicha tarea. Los procesos de Markov fueron desarrollados inicialmente por Andrei A. Marko
alumno de Chebyshev, y su primera utilización tuvo un objetivo realmente lingüístico: modeliz
las secuencias de letras de las palabras en la literatura rusa [Markov 1913]. Posteriormente, l
modelos de Markov evolucionaron hasta convertirse en una herramienta estadística de propósi
general. La suposición subyacente de las técnicas de etiquetación basadas en HMM,s es q
la tarea de la etiquetación se puede formalizar como un proceso paramétrico aleatorio, cuy
parámetros se pueden estimar de una manera precisa y perfectamente definida.

Comenzaremos con un repaso de la teoría de cadenas de Markov, seguiremos con u
extensión de las ideas mostradas hacia los modelos de Markov ocultos y centraremos nuest
atención en las tres cuestiones fundamentales que surgen a la hora de utilizar un HMM:

1. La evaluación de la probabilidad de una secuencia de observaciones, dado un HM
específico.

2. La determinación de la secuencia de estados más probable, dada una secuencia
observaciones específica.

3. La estimación de los parámetros del modelo para que éste se ajuste a las secuencias
observaciones disponibles.

Una vez que estos tres problemas fundamentales sean resueltos, veremos cómo se pueden aplic
inmediatamente los HMM,s al problema de la etiquetación del lenguaje natural, y estudiarem
con detalle las múltiples variantes de implementación que surgen al hacerlo.

siado la notación, supondremos que existe una correspondencia entre el conjunto de estados
conjunto de números enteros $\{1, 2, \ldots, N\}$, y etiquetaremos cada estado con uno de esos
ros, tal y como se ve en la figura 3.1, donde $N = 5$. Regularmente, transcurrido un espacio
mpo discreto, el sistema cambia de estado (posiblemente volviendo al mismo), de acuerdo
n conjunto de probabilidades de transición asociadas a cada uno de los estados del modelo.
nstantes de tiempo asociados a cada cambio de estado se denotan como $t = 1, 2, \ldots, T$,
stado actual en el instante de tiempo t se denota como q_t. En general, una descripción
bilística completa del sistema requeriría la especificación del estado actual, así como de
los estados precedentes. Sin embargo, las cadenas de Markov presentan dos características
mportantes:

Propiedad del horizonte limitado. Esta propiedad permite truncar la dependencia probabilística del estado actual y considerar, no todos los estados precedentes, sino únicamente un subconjunto finito de ellos. En general, una cadena de Markov de orden n es la que utiliza n estados previos para predecir el siguiente estado. Por ejemplo, para el caso de las cadenas de Markov de tiempo discreto de primer orden tenemos que:

$$P(q_t = j | q_{t-1} = i, q_{t-2} = k, \ldots) = P(q_t = j | q_{t-1} = i). \tag{3.1}$$

Propiedad del tiempo estacionario. Esta propiedad nos permite considerar sólo aquellos procesos en los cuales la parte derecha de (3.1) es independiente del tiempo. Esto nos lleva a una matriz $A = \{a_{ij}\}$ de probabilidades de transición entre estados de la forma

$$a_{ij} = P(q_t = j | q_{t-1} = i) = P(j|i), \qquad 1 \leq i, j \leq N,$$

independientes del tiempo, pero con las restricciones estocásticas estándar:

$$a_{ij} \geq 0, \qquad \forall i, j,$$

$$\sum_{j=1}^{N} a_{ij} = 1, \qquad \forall i.$$

Sin embargo, es necesario especificar también el vector $\pi = \{\pi_i\}$, que almacena la probabilidad que tiene cada uno de los estados de ser el estado inicial[2]:

$$\pi_i = P(q_1 = i), \qquad \pi_i \geq 0, \qquad 1 \leq i \leq N,$$

$$\sum_{i=1}^{N} \pi_i = 1.$$

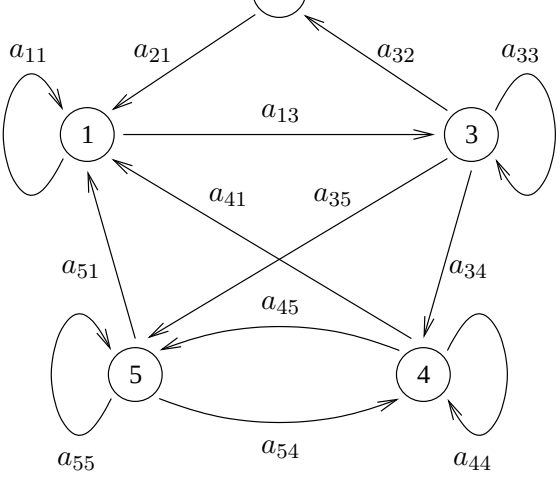


Figura 3.1: Una cadena de Markov de 5 estados con algunas transiciones entre ellos

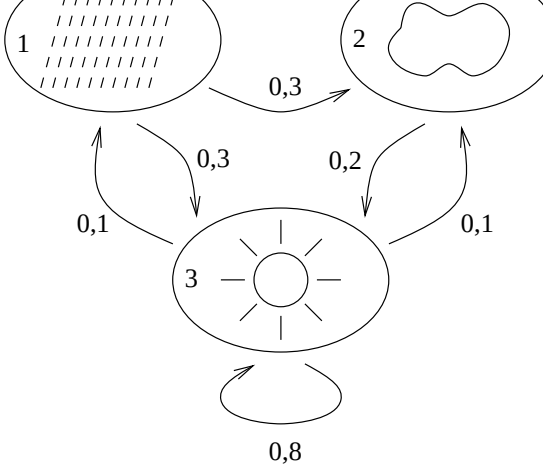


Figura 3.2: Un modelo de Markov para evolución del clima

A un proceso estocástico que satisface estas características se le puede llamar un *modelo Markov observable*, porque su salida es el conjunto de estados por los que pasa en cada instan de tiempo, y cada uno de estos estados se corresponde con un suceso observable.

Ejemplo 3.1 Para remarcar las ideas, supongamos que una vez al día, por ejemplo al mediod se observa el clima, y las posibles observaciones resultan ser las siguientes: (1) precipitación lluvia o nieve, (2) nublado, o (3) soleado. Supongamos también que el clima del día t caracteriza por uno solo de esos tres estados, que la matriz A de las probabilidades de transici entre estados es

$$A = \{a_{ij}\} = \begin{bmatrix} 0,4 & 0,3 & 0,3 \\ 0,2 & 0,6 & 0,2 \\ 0,1 & 0,1 & 0,8 \end{bmatrix}$$

y que los tres estados tienen la misma probabilidad de ser el estado inicial, es decir, $\pi_i =$ $1 \leq i \leq 3$. Tal y como se ve en la figura 3.2, no hemos hecho más que especificar un modelo Markov para describir la evolución del clima.

La probabilidad de observar una determinada secuencia de estados, es decir, la probabilid de que una secuencia finita de T variables aleatorias con dependencia de primer orde $q_1, q_2, \ldots, q_T$, tome unos determinados valores, $o_1, o_2, \ldots, o_T$, con todos los $o_i \in \{1, 2, \ldots, N\}$, sencilla de obtener. Simplemente calculamos el producto de las probabilidades que figuran las aristas del grafo o en la matriz de transiciones:

$$P(o_1, o_2, \ldots, o_T) =$$

$$= P(q_1 = o_1)P(q_2 = o_2|q_1 = o_1)P(q_3 = o_3|q_2 = o_2)\ldots P(q_T = o_T|q_{T-1} = o_{T-1}) =$$

[2]La necesidad de este vector podría evitarse especificando que la cadena de Markov comienza siempre en estado inicial extra, e incluyendo en la matriz A las probabilidades de transición de este nuevo estado: las

plo 3.2 Utilizando el modelo de Markov para la evolución del clima, podemos calcular la bilidad de observar la secuencia de estados 2, 3, 2, 1, como sigue:

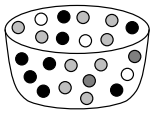
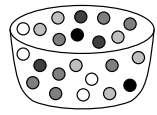
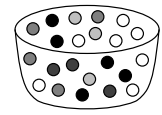
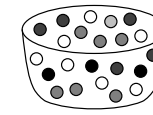
$$, 2, 1) = P(2)P(3|2)P(2|3)P(1|2) = \pi_2 \times a_{23} \times a_{32} \times a_{21} = \frac{1}{3} \times 0,2 \times 0,1 \times 0,2 = 0,00133333.$$

es, por tanto, la probabilidad que corresponde a la secuencia *nublado-soleado-nublado-* □

Extensión a modelos de Markov ocultos

sección anterior hemos considerado modelos de Markov en los cuales cada estado se ponde de manera determinista con un único suceso observable. Es decir, la salida en un o dado no es aleatoria, sino que es siempre la misma. Esta modelización puede resultar siado restrictiva a la hora de ser aplicada a problemas reales. En esta sección extendemos el pto de modelos de Markov de tal manera que es posible incluir aquellos casos en los cuales la vación es una función probabilística del estado. El modelo resultante, denominado modelo arkov oculto, es un modelo doblemente estocástico, ya que uno de los procesos no se puede var directamente (está oculto), sino que se puede observar sólo a través de otro conjunto ocesos estocásticos, los cuales producen la secuencia de observaciones.

plo 3.3 Para ilustrar los conceptos básicos de un modelo de Markov oculto consideremos ema de urnas y bolas de la figura 3.3. El modelo consta de un gran número N de urnas adas dentro de una habitación. Cada urna contiene en su interior un número también e de bolas. Cada bola tiene un único color, y el número de colores distintos para ellas es l proceso físico para obtener las observaciones es como sigue. Una persona se encuentra o de la habitación y, siguiendo un procedimiento totalmente aleatorio, elige una urna inicial, una bola, nos grita en alto su color y devuelve la bola a la urna. Posteriormente elige otra de acuerdo con un proceso de selección aleatorio asociado a la urna actual, y efectúa la cción de una nueva bola. Este paso se repite un determinado número de veces, de manera odo el proceso genera una secuencia finita de colores, la cual constituye la salida observable odelo, quedando oculta para nosotros la secuencia de urnas que se ha seguido.

Urna 1	Urna 2	Urna 3		Urna N
$(color\ 1) = b_{11}$	$P(color\ 1) = b_{21}$	$P(color\ 1) = b_{31}$	$\ldots$	$P(color\ 1) = b_{N1}$
$(color\ 2) = b_{12}$	$P(color\ 2) = b_{22}$	$P(color\ 2) = b_{32}$	$\ldots$	$P(color\ 2) = b_{N2}$
$(color\ 3) = b_{13}$	$P(color\ 3) = b_{23}$	$P(color\ 3) = b_{33}$	$\ldots$	$P(color\ 3) = b_{N3}$
$\vdots$	$\vdots$	$\vdots$		$\vdots$
$(color\ M) = b_{1M}$	$P(color\ M) = b_{2M}$	$P(color\ M) = b_{3M}$	$\ldots$	$P(color\ M) = b_{NM}$

Figura 3.3: Un modelo de Markov oculto de N urnas de bolas y M colores

proceso se modeliza de forma que cada urna se corresponde con un estado del HMM, la

un determinado color de bola no nos dice automáticamente de qué urna viene esa bola.

El ejemplo de las urnas y las bolas puede identificarse de una manera muy intuituva c
el problema de la etiquetación de las palabras de un texto en lenguaje natural. En este ca
concreto, las bolas representan a las palabras, las urnas representan a las distintas etiquet
o categorías gramaticales a las que pertenecen las palabras, y las secuencias de observacion
representan a las frases del texto. Una misma palabra puede estar en urnas distintas, de a
la ambigüedad léxica, y puede estar también varias veces en la misma urna, de ahí que l
probabilidades de las palabras que están dentro de una misma urna puedan ser distintas.

3.3 Elementos de un modelo de Markov oculto

Los ejemplos vistos anteriormente nos proporcionan una idea de lo que son los modelos de Mark
ocultos y de cómo se pueden aplicar a escenarios prácticos reales, tales como la etiquetación
las palabras de un texto en lenguaje natural. Pero antes de tratar nuestro caso particular, vam
a definir formalmente cuáles son los elementos de un modelo de Markov oculto.

Definición 3.1 Un *HMM* se caracteriza por la 5-tupla (Q, V, π, A, B), donde:

1. Q es el conjunto de estados del modelo. Aunque los estados permanecen ocultos, para
mayoría de las aplicaciones prácticas se conocen *a priori*. Por ejemplo, para el caso de
etiquetación de palabras, cada etiqueta del juego de etiquetas utilizado sería un estad
Generalmente los estados están conectados de tal manera que cualquiera de ellos se pue
alcanzar desde cualquier otro en un solo paso, aunque existen muchas otras posibilidad
de interconexión. Los estados se etiquetan como $\{1, 2, \ldots, N\}$, y el estado actual en
instante de tiempo t se denota como q_t. El uso de instantes de tiempo es apropiado, p
ejemplo, en la aplicación de los HMM,s al procesamiento de voz. No obstante, para
caso de la etiquetación de palabras, no hablaremos de los instantes de tiempo, sino de l
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 estad
Por tanto, cada uno de los símbolos individuales que un estado puede emitir se denc
como $\{v_1, v_2, \ldots, v_M\}$. En el caso del modelo de las urnas y las bolas, M es el número
colores distintos y cada $v_k, 1 \leq k \leq M$, es un color distinto. En el caso de la etiquetaci
de palabras, M es el tamaño del diccionario y cada $v_k, 1 \leq k \leq M$, es una palabra distint

3. $\pi = \{\pi_i\}$, es la distribución de probabilidad del estado inicial. Por tanto,

$$\pi_i = P(q_1 = i), \qquad \pi_i \geq 0, \qquad 1 \leq i \leq N,$$

$$\sum_{i=1}^{N} \pi_i = 1.$$

4. $A = \{a_{ij}\}$ es la distribución de probabilidad de las transiciones entre estados, es decir,

Para el caso de un modelo con estados totalmente conexos en un solo paso, tenemos que $a_{ij} > 0$ para todo i, j. Para otro tipo de HMM,s podría existir algún $a_{ij} = 0$.

$B = \{b_j(v_k)\}$ es la distribución de probabilidad de los sucesos observables, es decir,

$$b_j(v_k) = P(o_t = v_k | q_t = j) = P(v_k|j), \quad b_j(v_k) \geq 0, \quad 1 \leq j \leq N,\ 1 \leq k \leq M,\ 1 \leq t \leq T.$$

$$\sum_{k=1}^{M} b_j(v_k) = 1, \qquad \forall j.$$

Este conjunto de probabilidades se conoce también con el nombre de *conjunto de probabilidades de emisión*.

como hemos visto, una descripción estricta de un HMM necesita la especificación de Q y V, junto de estados y el conjunto de los símbolos que forman las secuencias de observación, ctivamente, y la especificación de los tres conjuntos de probabilidades π, A y B. Pero que los dos primeros conjuntos normalmente se conocen *a priori*, y que en todo caso los ltimos elementos del HMM ya incluyen de manera explícita al resto de los parámetros, aremos la notación compacta

$$\mu = (\pi, A, B)$$

argo de las siguientes secciones, y ésta seguirá siendo una representación completa de un . □

ada una especificación de un HMM, podemos simular un proceso estocástico que genere ncias de datos, donde las leyes de producción de dichas secuencias están perfectamente das en el modelo. Sin embargo, es mucho más interesante tomar una secuencia de datos, er que efectivamente ha sido generada por un HMM, y estudiar distintas propiedades sobre ales como su probabilidad, o la secuencia de estados más probable por la que ha pasado. uiente sección se ocupa de este tipo de cuestiones.

Las tres preguntas fundamentales al usar un HMM

en tres preguntas fundamentales que debemos saber responder para poder utilizar los ,s en aplicaciones reales. Estas tres preguntas son las siguientes:

Dada una secuencia de observaciones $O = (o_1, o_2, \ldots, o_T)$ y dado un modelo $\mu = (\pi, A, B)$, ¿cómo calculamos de una manera eficiente $P(O|\mu)$, es decir, la probabilidad de dicha secuencia dado el modelo?

Dada una secuencia de observaciones $O = (o_1, o_2, \ldots, o_T)$ y dado un modelo $\mu = (\pi, A, B)$, ¿cómo elegimos la secuencia de estados $S = (q_1, q_2, \ldots, q_T)$ óptima, es decir, la que mejor *explica* la secuencia de observaciones?

Dada una secuencia de observaciones $O = (o_1, o_2, \ldots, o_T)$, ¿cómo estimamos los parámetros del modelo $\mu = (\pi, A, B)$ para maximizar $P(O|\mu)$?, es decir, ¿cómo podemos encontrar el

que a partir de ella se entrena el HMM. Ésta es la problemática que se aborda en la terce pregunta.

La segunda pregunta trata el problema de descubrir la parte oculta del modelo, es dec trata sobre cómo podemos adivinar qué camino ha seguido la cadena de Markov. Este cami oculto se puede utilizar para clasificar las observaciones. En el caso de la etiquetación de texto en lenguaje natural, dicho camino nos ofrece la secuencia de etiquetas más probable pa las palabras del texto.

La primera pregunta representa el problema de la evaluación de una secuencia observaciones dado el modelo. La resolución de este problema nos proporciona la probabilid de que la secuencia haya sido generada por ese modelo. Un ejemplo práctico puede ser supon la existencia de varios HMM,s compitiendo entre sí. La evaluación de la secuencia en cada u de ellos nos permitirá elegir el modelo que mejor encaja con los datos observados.

A continuación se describen formalmente todos los algoritmos matemáticos que son necesari para responder a estas tres preguntas.

3.4.1 Cálculo de la probabilidad de una observación

Dada una secuencia de observaciones $O = (o_1, o_2, \ldots, o_T)$ y un modelo $\mu = (\pi, A, B)$, querem calcular de una manera eficiente $P(O|\mu)$, es decir, la probabilidad de dicha secuencia dado modelo. La forma más directa de hacerlo es enumerando primero todas las posibles secuenci de estados de longitud T, el número de observaciones. Existen N^T secuencias distint Considerando una de esas secuencias, $S = (q_1, q_2, \ldots, q_T)$, la probabilidad de dicha secuencia estados es

$$P(S|\mu) = \pi_{q_1}\, a_{q_1 q_2}\, a_{q_2 q_3} \ldots a_{q_{T-1} q_T} \tag{3.}$$

y la probabilidad de observar O a través de la secuencia S es

$$P(O|S, \mu) = \prod_{t=1}^{T} P(o_t|q_t, \mu) = b_{q_1}(o_1)\, b_{q_2}(o_2) \ldots b_{q_T}(o_T) \tag{3.}$$

La probabilidad conjunta de O y S, es decir, la probabilidad de que O y S ocurr simultáneamente, es simplemente el producto de (3.2) y (3.3), es decir,

$$P(O, S|\mu) = P(S|\mu)\, P(O|S, \mu)$$

Por tanto, la probabilidad de O dado el modelo se obtiene sumando esta probabilidad conjun para todas las posibles secuencias S:

$$P(O|\mu) = \sum_{S} P(S|\mu)\, P(O|S, \mu).$$

Sin embargo, si calculamos $P(O|\mu)$ de esta forma, necesitamos realizar exactamente $(2T-1)N$ multiplicaciones y $N^T - 1$ sumas, es decir, un total de $2TN^T - 1$ operaciones. Estas cifras son computacionalmente admisibles ni siquiera para valores pequeños de N y T. Por ejemp para $N = 5$ estados y $T = 100$ observaciones, el número de operaciones es del orden de 10 El secreto para evitar esta complejidad tan elevada está en utilizar técnicas de programaci dinámica, con el fin de recordar los resultados parciales, en lugar de recalcularlos. A continuaci

$$\alpha_t(i) = P(o_1, o_2, \ldots, o_t, q_t = i|\mu),$$

ir, la probabilidad conjunta de obtener $o_1, o_2, \ldots, o_t$, la secuencia parcial de observaciones el instante de tiempo t, y de estar en el estado i en ese instante de tiempo t, dado el modelo s valores de $\alpha_t(i)$, para los distintos estados y para los distintos instantes de tiempo, se n obtener iterativamente, y pueden ser utilizados para calcular $P(O|\mu)$ mediante los pasos guiente algoritmo.

ritmo 3.1 Cálculo hacia adelante de la probabilidad de una secuencia de observaciones:

Inicialización:

$$\alpha_1(i) = \pi_i \, b_i(o_1), \qquad 1 \leq i \leq N.$$

Recurrencia:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) \, a_{ij}\right] b_j(o_{t+1}), \qquad t = 1, 2, \ldots, T-1, \qquad 1 \leq j \leq N. \tag{3.4}$$

Terminación:

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

so 1 inicializa las N probabilidades $\alpha_1(i)$ como la probabilidad conjunta de que i sea el o inicial y genere la primera observación o_1. El paso de recurrencia, que es el punto central e algoritmo, se ilustra en la figura 3.4 (a). Esta figura muestra cómo el estado j se puede zar en el instante de tiempo $t+1$ desde los N posibles estados i del instante de tiempo t. cada uno de esos estados, dado que $\alpha_t(i)$ es la probabilidad conjunta de haber observado $\ldots, o_t$, y de estar en el estado i en el instante t, $\alpha_t(i) \, a_{ij}$ será la probabilidad conjunta ber observado $o_1, o_2, \ldots, o_t$, y de haber alcanzado el estado j en el instante $t+1$ desde ado i del instante t. Sumando todos estos productos sobre los N posibles estados i del te t, y multiplicando esa suma por $b_j(o_{t+1})$, la probabilidad de que el estado j emita el lo o_{t+1}, obtenemos $\alpha_{t+1}(j)$, es decir, la probabilidad conjunta de obtener $o_1, o_2, \ldots, o_{t+1}$, uencia parcial de observaciones hasta el instante de tiempo $t+1$, y de estar en el estado se instante de tiempo $t+1$. El cálculo de la ecuación (3.4) se realiza para los N estados un mismo instante de tiempo t, y para todos los instantes de tiempo $t = 1, 2, \ldots, T-1$. mente, el paso 3 obtiene $P(O|\mu)$ sumando el valor de las N variables $\alpha_T(i)$. □

s cálculos globales involucrados en este proceso requieren del orden de N^2T operaciones, como se muestra en el enrejado de la figura 3.4 (b). Más concretamente, se trata de $+ 1)(T-1) + N$ multiplicaciones y $N(N-1)(T-1) + N - 1$ sumas, es decir, un total $T-1)N^2 + 2N - 1$ operaciones. Para $N = 5$ estados y $T = 100$ observaciones, el número de operaciones es aproximadamente 5.000, que comparado con 10^{72} supone un ahorro de

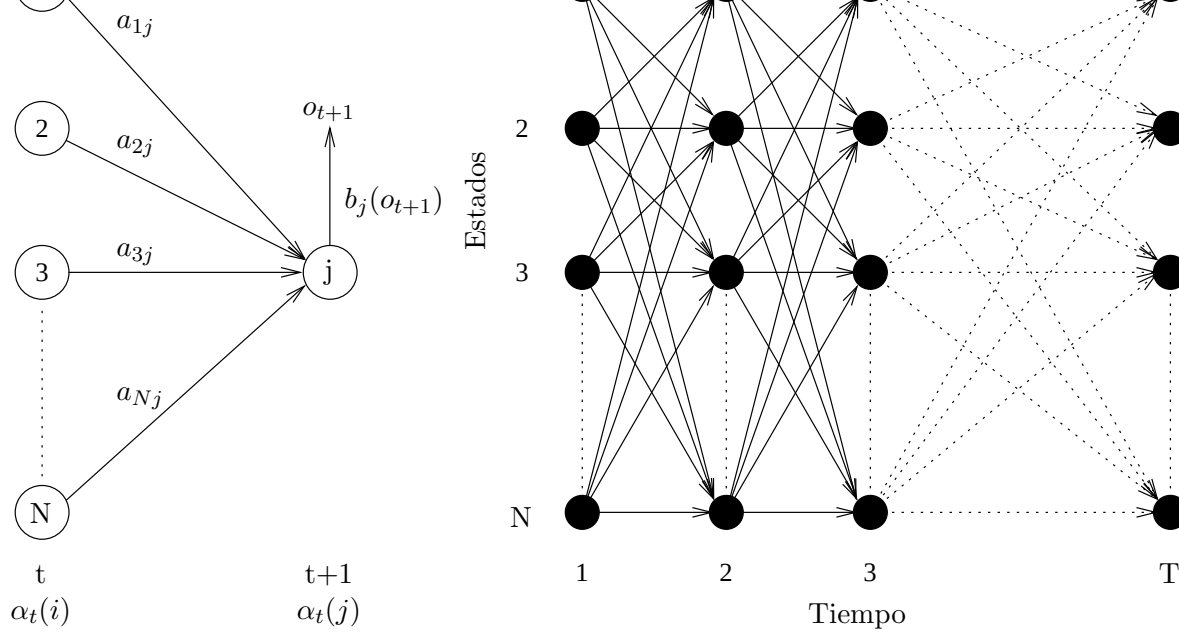


Figura 3.4: (a) Detalle de la secuencia de operaciones necesarias para el cálculo hacia adelan de la variable $\alpha_{t+1}(j)$ y (b) implementación genérica del cálculo hacia adelante de la variab $\alpha_t(i)$ mediante un enrejado de T observaciones y N estados

3.4.1.2 Procedimiento hacia atrás

De manera similar, podemos considerar la variable $\beta_t(i)$ definida como

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, \ldots, o_T | q_t = i, \mu),$$

es decir, la probabilidad de la secuencia de observación parcial desde el instante de tiempo $t+$ hasta el final, dado que el estado en el instante de tiempo t es i y dado el modelo μ. Nuevamen podemos resolver las variables $\beta_t(i)$ y calcular el valor de $P(O|\mu)$ de manera recurrente median los pasos del siguiente algoritmo.

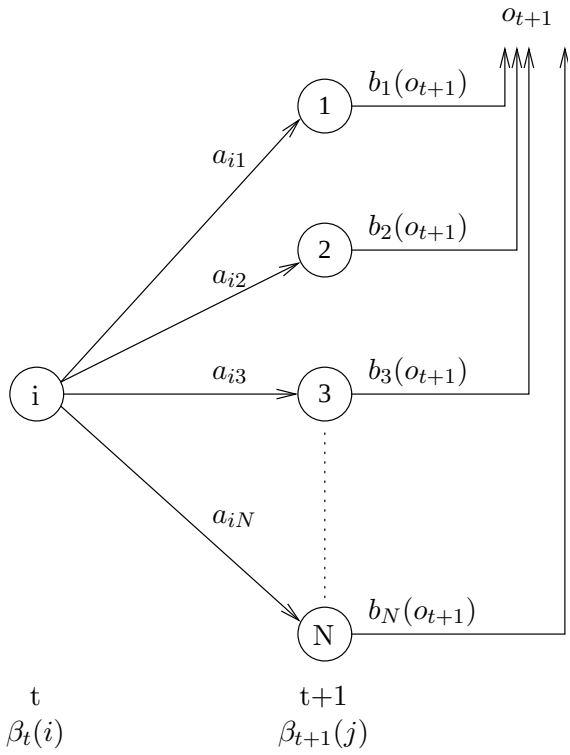
Algoritmo 3.2 Cálculo hacia atrás de la probabilidad de una secuencia de observaciones:

1. Inicialización:
$$\beta_T(i) = 1, \qquad 1 \leq i \leq N.$$

2. Recurrencia:
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, \beta_{t+1}(j)\, b_j(o_{t+1}), \qquad t = T-1, T-2, \ldots, 1, \qquad 1 \leq i \leq N.$$

3. Terminación:
$$P(O|\mu) = \sum_{i=1}^{N} \beta_1(i)\, \pi_i\, b_i(o_1).$$

ene $P(O|\mu)$ sumando el valor de las N variables $\beta_1(i)$ multiplicado por la probabilidad e el estado i sea el estado inicial y por la probabilidad de que emita el primer símbolo de vación o_1. □



a 3.5: Detalle de la secuencia de operaciones necesarias para el cálculo hacia atrás de $\beta_t(i)$

na vez más, el cálculo de las variables $\beta_t(i), 1 \leq t \leq T, 1 \leq i \leq N$, requiere del orden de operaciones y se puede resolver sobre un enrejado similar al de la figura 3.4 (b). Por tanto, dadera razón para haber introducido este procedimiento hacia atrás es que los cálculos crados en él son de vital importancia para resolver las preguntas fundamentales 2 y 3 los HMM,s, es decir, el cálculo de la secuencia de estados óptima y la estimación de los metros del modelo, tal y como veremos a continuación.

Elección de la secuencia de estados más probable

gundo problema ha sido definido vagamente como el problema de encontrar la secuencia ados óptima que mejor *explica* las observaciones. Debido a esta definición informal del ema, podrían existir varias formas de abordarlo, es decir, se podrían considerar diferentes os para esa optimación. Uno de ellos podría ser la elección de los estados que son *dualmente* más probables en cada instante de tiempo. Para implementar este criterio, nos considerar la cantidad $\gamma_t(i)$ definida como

$$P(O|\mu) \qquad \sum_{j=1}^{N} P(q_t = j, O|\mu) \qquad \sum_{j=1}^{N} \alpha_t(j)\,\beta_t(j)$$

donde $\alpha_t(i)$ almacena la probabilidad de la observación parcial $o_1, o_2, \ldots, o_t$, y $\beta_t(i)$ almace la probabilidad de la observación parcial $o_{t+1}, o_{t+2}, \ldots, o_T$, dado el estado i en el instante Utilizando $\gamma_t(i)$ podemos obtener q_t^*, el estado individualmente más probable en el instante como

$$q_t^* = \arg \max_{1 \le i \le N} [\gamma_t(i)], \qquad 1 \le t \le T. \tag{3.}$$

La ecuación (3.7) maximiza el número esperado de estados correctos, eligiendo el estado m probable en cada instante t. Sin embargo, podrían surgir problemas con la secuencia de estad resultante. Por ejemplo, en un HMM con alguna transición entre estados de probabilidad ce ($a_{ij} = 0$ para algún i y algún j), podría ocurrir que esos estados i y j aparecieran contigu en la secuencia óptima, cuando de hecho esa secuencia ni siquiera sería una secuencia váli Esto es debido a que la solución que nos proporciona la ecuación (3.7) determina simplemen el estado más probable en cada instante, sin tener en cuenta la probabilidad de la secuencia estados resultante.

Una posible solución al problema anterior es modificar el criterio de optimalidad pa considerar, por ejemplo, la secuencia que maximiza el número esperado de pares de estad correctos (q_{t-1}, q_t), o de triplas de estados correctos (q_{t-2}, q_{t-1}, q_t), etc. Aunque estos criteri podrían ser razonables para algunas aplicaciones, el criterio más ampliamente utilizado consis en encontrar la mejor secuencia considerando globalmente todos los instantes de tiempo, es dec la secuencia de estados $S = (q_1, q_2, \ldots, q_T)$ que maximiza $P(S|O, \mu)$, lo cual es equivalente maximizar $P(S, O|\mu)$. Existe un procedimiento formal y eficiente, basado también en técnic de programación dinámica, para obtener esa secuencia S. Dicho procedimiento es el algoritm de Viterbi [Viterbi 1967, Forney 1973].

3.4.2.1 Algoritmo de Viterbi

Para encontrar la secuencia de estados más probable, $S = (q_1, q_2, \ldots, q_T)$, dada la observaci $O = (o_1, o_2, \ldots, o_T)$, consideramos la variable $\delta_t(i)$ definida como

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P(q_1, q_2, \ldots, q_{t-1}, q_t = i, o_1, o_2, \ldots, o_t|\mu),$$

es decir, $\delta_t(i)$ almacena la probabilidad del mejor camino que termina en el estado i, tenien en cuenta las t primeras observaciones. Se demuestra fácilmente que

$$\delta_{t+1}(j) = \left[\max_{1 \le i \le N} \delta_t(i)\, a_{ij}\right] b_j(o_{t+1}). \tag{3.}$$

Una vez calculadas las $\delta_t(i)$ para todos los estados y para todos los instantes de tiempo, secuencia de estados se construye realmente hacia atrás a través de una traza que recuerda argumento que maximizó la ecuación (3.8) para cada instante t y para cada estado j. Esta tra se almacena en las correspondientes variables $\psi_t(j)$. La descripción completa del algoritmo como sigue.

Algoritmo 3.3 Cálculo de la secuencia de estados más probable para una secuencia

$$\delta_{t+1}(j) = \left[\max_{1\leq i\leq N} \delta_t(i)\, a_{ij}\right] b_j(o_{t+1}), \qquad t = 1, 2, \ldots, T-1, \qquad 1 \leq j \leq N. \tag{3.9}$$

$$\psi_{t+1}(j) = \arg\max_{1\leq i\leq N} \delta_t(i)\, a_{ij}, \qquad t = 1, 2, \ldots, T-1, \qquad 1 \leq j \leq N.$$

Terminación:

$$q_T^* = \arg\max_{1\leq i\leq N} \delta_T(i).$$

Construcción hacia atrás de la secuencia de estados:

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \qquad t = T-1, T-2, \ldots, 1.$$

oritmo de Viterbi es similar al cálculo hacia adelante de la probabilidad de una observación imos en el algoritmo 3.1. Las únicas diferencias reseñables son que el sumatorio de la ón (3.4) se ha cambiado por la maximización de la ecuación (3.9), y que se ha añadido o final para construir hacia atrás la secuencia de estados. En todo caso, la complejidad goritmo es del orden de N^2T operaciones y se puede resolver también sobre un enrejado r al de la figura 3.4 (b). □

r supuesto, durante los cálculos del algoritmo de Viterbi se podrían obtener empates. o ocurre, la elección del camino se realizaría aleatoriamente. Por otra parte, existen a do aplicaciones prácticas en las cuales se utiliza no sólo la mejor secuencia de estados, sino mejores secuencias. Para todos estos casos, se podría considerar una implementación del tmo de Viterbi que devolviera varias secuencias de estados.

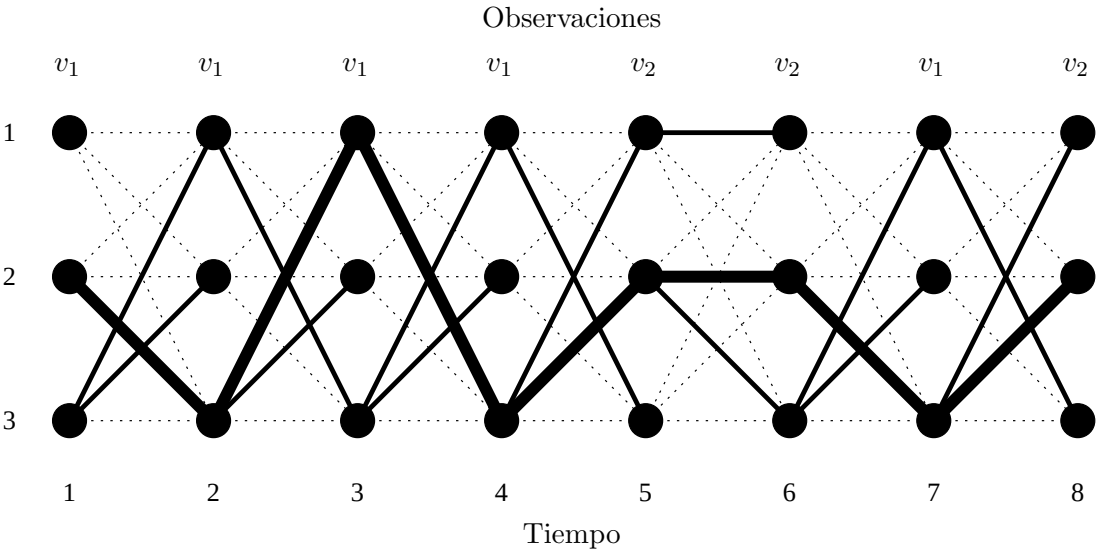


Figura 3.6: Ejemplo de ejecución del algoritmo de Viterbi

$$Q = \{1, 2, 3\}, \qquad V = \{v_1, v_2\}, \qquad \pi = \begin{bmatrix} 0,50 \\ 0,25 \end{bmatrix},$$

$$A = \begin{bmatrix} 0,25 & 0,25 & 0,50 \\ 0 & 0,25 & 0,75 \\ 0,50 & 0,50 & 0 \end{bmatrix} \quad y \quad B = \begin{bmatrix} 0,50 & 0,50 \\ 0,25 & 0,75 \\ 0,75 & 0,25 \end{bmatrix}.$$

Los cálculos para encontrar la secuencia de estados más probable dada la observación

$$O = (v_1, v_1, v_1, v_1, v_2, v_2, v_1, v_2)$$

de longitud $T = 8$, son los siguientes:

$$\begin{aligned}
&\delta_1(1) = \pi_1\, b_1(v_1) = (0,25)(0,50) \\
&\delta_1(2) = \pi_2\, b_2(v_1) = (0,50)(0,25) \\
&\delta_1(3) = \pi_3\, b_3(v_1) = (0,25)(0,75)
\end{aligned}$$

$$\begin{aligned}
&\delta_2(1) = \max\left[\delta_1(1)\, a_{11}, \delta_1(2)\, a_{21}, \underline{\delta_1(3)\, a_{31}}\right] b_1(v_1) = (0,25)\,(0,50)^2\,(0,75) && \psi_2(1) = 3 \\
&\delta_2(2) = \max\left[\delta_1(1)\, a_{12}, \delta_1(2)\, a_{22}, \underline{\delta_1(3)\, a_{32}}\right] b_2(v_1) = (0,25)^2\,(0,50)\,(0,75) && \psi_2(2) = 3 \\
&\delta_2(3) = \max\left[\delta_1(1)\, a_{13}, \underline{\delta_1(2)\, a_{23}}, \delta_1(3)\, a_{33}\right] b_3(v_1) = (0,25)\,(0,50)\,(0,75)^2 && \psi_2(3) = 2
\end{aligned}$$

$$\begin{aligned}
&\delta_3(1) = \max\left[\delta_2(1)\, a_{11}, \delta_2(2)\, a_{21}, \underline{\delta_2(3)\, a_{31}}\right] b_1(v_1) = (0,25)\,(0,50)^3\,(0,75)^2 && \psi_3(1) = 3 \\
&\delta_3(2) = \max\left[\delta_2(1)\, a_{12}, \delta_2(2)\, a_{22}, \underline{\delta_2(3)\, a_{32}}\right] b_2(v_1) = (0,25)^2\,(0,50)^2\,(0,75)^2 && \psi_3(2) = 3 \\
&\delta_3(3) = \max\left[\underline{\delta_2(1)\, a_{13}}, \delta_2(2)\, a_{23}, \delta_2(3)\, a_{33}\right] b_3(v_1) = (0,25)\,(0,50)^3\,(0,75)^2 && \psi_3(3) = 1
\end{aligned}$$

$$\begin{aligned}
&\delta_4(1) = \max\left[\delta_3(1)\, a_{11}, \delta_3(2)\, a_{21}, \underline{\delta_3(3)\, a_{31}}\right] b_1(v_1) = (0,25)\,(0,50)^5\,(0,75)^2 && \psi_4(1) = 3 \\
&\delta_4(2) = \max\left[\delta_3(1)\, a_{12}, \delta_3(2)\, a_{22}, \underline{\delta_3(3)\, a_{32}}\right] b_2(v_1) = (0,25)^2\,(0,50)^4\,(0,75)^2 && \psi_4(2) = 3 \\
&\delta_4(3) = \max\left[\underline{\delta_3(1)\, a_{13}}, \delta_3(2)\, a_{23}, \delta_3(3)\, a_{33}\right] b_3(v_1) = (0,25)\,(0,50)^4\,(0,75)^3 && \psi_4(3) = 1
\end{aligned}$$

$$\begin{aligned}
&\delta_5(1) = \max\left[\delta_4(1)\, a_{11}, \delta_4(2)\, a_{21}, \underline{\delta_4(3)\, a_{31}}\right] b_1(v_2) = (0,25)\,(0,50)^6\,(0,75)^3 && \psi_5(1) = 3 \\
&\delta_5(2) = \max\left[\delta_4(1)\, a_{12}, \delta_4(2)\, a_{22}, \underline{\delta_4(3)\, a_{32}}\right] b_2(v_2) = (0,25)\,(0,50)^5\,(0,75)^4 && \psi_5(2) = 3 \\
&\delta_5(3) = \max\left[\underline{\delta_4(1)\, a_{13}}, \delta_4(2)\, a_{23}, \delta_4(3)\, a_{33}\right] b_3(v_2) = (0,25)^2\,(0,50)^6\,(0,75)^2 && \psi_5(3) = 1
\end{aligned}$$

$$\begin{aligned}
&\delta_6(1) = \max\left[\underline{\delta_5(1)\, a_{11}}, \delta_5(2)\, a_{21}, \delta_5(3)\, a_{31}\right] b_1(v_2) = (0,25)^2\,(0,50)^7\,(0,75)^3 && \psi_6(1) = 1 \\
&\delta_6(2) = \max\left[\delta_5(1)\, a_{12}, \underline{\delta_5(2)\, a_{22}}, \delta_5(3)\, a_{32}\right] b_2(v_2) = (0,25)^2\,(0,50)^5\,(0,75)^5 && \psi_6(2) = 2 \\
&\delta_6(3) = \max\left[\delta_5(1)\, a_{13}, \underline{\delta_5(2)\, a_{23}}, \delta_5(3)\, a_{33}\right] b_3(v_2) = (0,25)^2\,(0,50)^5\,(0,75)^5 && \psi_6(3) = 2
\end{aligned}$$

$$\begin{aligned}
&\delta_7(1) = \max\left[\delta_6(1)\, a_{11}, \delta_6(2)\, a_{21}, \underline{\delta_6(3)\, a_{31}}\right] b_1(v_1) = (0,25)^2\,(0,50)^7\,(0,75)^5 && \psi_7(1) = 3 \\
&\delta_7(2) = \max\left[\delta_6(1)\, a_{12}, \delta_6(2)\, a_{22}, \underline{\delta_6(3)\, a_{32}}\right] b_2(v_1) = (0,25)^3\,(0,50)^6\,(0,75)^5 && \psi_7(2) = 3 \\
&\delta_7(3) = \max\left[\delta_6(1)\, a_{13}, \underline{\delta_6(2)\, a_{23}}, \delta_6(3)\, a_{33}\right] b_3(v_1) = (0,25)^2\,(0,50)^5\,(0,75)^7 && \psi_7(3) = 2
\end{aligned}$$

$$\begin{aligned}
&\delta_8(1) = \max\left[\delta_7(1)\, a_{11}, \delta_7(2)\, a_{21}, \underline{\delta_7(3)\, a_{31}}\right] b_1(v_2) = (0,25)^2\,(0,50)^7\,(0,75)^7 && \psi_8(1) = 3 \\
&\delta_8(2) = \max\left[\delta_7(1)\, a_{12}, \delta_7(2)\, a_{22}, \underline{\delta_7(3)\, a_{32}}\right] b_2(v_2) = (0,25)^2\,(0,50)^6\,(0,75)^8 && \psi_8(2) = 3 \\
&\delta_8(3) = \max\left[\underline{\delta_7(1)\, a_{13}}, \delta_7(2)\, a_{23}, \delta_7(3)\, a_{33}\right] b_3(v_2) = (0,25)^3\,(0,50)^8\,(0,75)^5 && \psi_8(3) = 1
\end{aligned}$$

En cada paso aparecen subrayados los términos máximos. El valor de i en cada uno de es
términos es el valor que se asigna a cada $\psi_t(j)$. La probabilidad máxima para la secuencia
observaciones completa se alcanza en $\delta_8(2)$, lo cual implica que $q_8^* = 2$, y al reconstruir hac
atrás la secuencia de estados obtenemos

$$S = (2, 3, 1, 3, 2, 2, 3, 2)$$

rece unido con una línea continua con el estado 3 del instante 3, ya que $\psi_4(1) = 3$. La ación intuitiva de esta línea es la siguiente: aún no sabemos si el camino más probable á por el estado 1 del instante 4, pero si lo hace, entonces sabemos que pasará también estado 3 del instante 3. Esta *traza* se mantiene hasta el final para todos los estados. Y ltimo, cuando vemos que la probabilidad máxima en el instante 8 correponde al estado 2, struimos el camino hacia atrás desde ese punto para obtener la secuencia de estados más ble, que es la que aparece marcada con una línea continua más gruesa. □

ra el caso que nos ocupa, que es el de la etiquetación de palabras, los cálculos involucrados algoritmo de Viterbi se realizan frase por frase sobre enrejados simplificados como el de la 3.7, donde en cada posición no se consideran todos los estados posibles, es decir, todas quetas del juego de etiquetas utilizado, sino sólo las etiquetas candidatas que proponga el nario para cada palabra.

o obstante, como se puede observar, hemos añadido un estado especial, que denominaremos 0 ó `etiqueta 0`, para marcar el comienzo y el fin de frase. Conceptualmente, el propósito le este nuevo estado es garantizar que el etiquetador simula un sistema que funciona nidamente en el tiempo, sin detenerse, tal y como exige el paradigma de los modelos arkov. En la práctica, una vez que nuestro etiquetador está funcionando, la justificación ial es que si tenemos que etiquetar un conjunto de frases, podemos procesar unas cuantas, er el proceso, apagar el ordenador, encenderlo al día siguiente, arrancar de nuevo el proceso, tar el resto de frases, y el resultado obtenido será el mismo que si las frases hubieran sido tadas todas juntas en bloque.

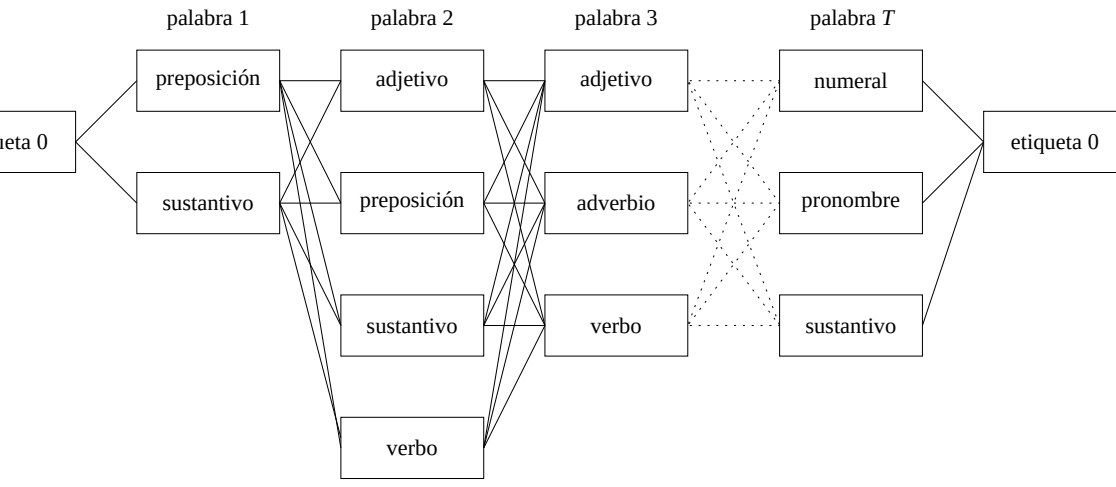


Figura 3.7: Enrejado simplificado para la etiquetación de una frase de T palabras

s experimentos realizados por Merialdo sugieren que no existe una gran diferencia ito entre maximizar la probabilidad de cada etiqueta individualmente y maximizar la bilidad de la secuencia completa, tal y como hace el algoritmo de Viterbi [Merialdo 1994]. ivamente, esto es sencillo de comprender. Con el algoritmo de Viterbi, las transiciones estados o etiquetas son más sensibles, pero si algo va mal, se pueden obtener secuencias arias etiquetas incorrectas. Maximizando etiqueta por etiqueta, esto no ocurre. Un fallo no

que etiquetando de derecha a izquierda. Si revisamos la implementación de los algoritm
de cálculo hacia adelante y hacia atrás de la probabilidad de una secuencia de observacion
dada (algoritmos 3.1 y 3.2), veremos que ambos obtienen la misma probabilidad. Así que,
definitiva, el proceso de etiquetación no es dependiente del sentido elegido. El algoritmo
Viterbi que hemos descrito aquí se basa en el algoritmo hacia adelante y por tanto etiqueta
izquierda a derecha, mientras que el etiquetador de Church, por ejemplo, lo hace en senti
contrario [Church 1988].

Sin embargo, lo que sí es importante es que hasta ahora hemos modelizado el proceso de
etiquetación del lenguaje natural utilizando siempre HMM,s de orden 1, es decir, modelos en l
que la dependencia contextual de una etiqueta se establece solamente en relación a la etique
anterior. A este tipo de modelos se les denomina también modelos de *bigramas* de etiqueta
Si se quiere trabajar con un HMM de orden superior, por ejemplo de orden 2, sería necesar
extender los cálculos del algoritmo de Viterbi para considerar las transiciones de estados des
dos posiciones antes de la etiqueta actual, tal y como podemos ver a continuación.

Algoritmo 3.4 Cálculo de la secuencia de estados más probable para una secuencia
observaciones dada (algoritmo de Viterbi para HMM,s de orden 2):

1. Inicialización:

$$\delta_1(i) = \pi_i \, b_i(o_1), \qquad 1 \leq i \leq N.$$

$$\delta_2(i,j) = \delta_1(i) \, a_{ij} \, b_j(o_2), \qquad 1 \leq i,j \leq N. \tag{3.1}$$

2. Recurrencia:

$$\delta_{t+1}(j,k) = \left[\max_{1 \leq i \leq N} \delta_t(i,j) \, a_{ijk} \right] b_k(o_{t+1}), \qquad t = 2,3,\ldots,T-1, \qquad 1 \leq j,k \leq N.$$

$$\psi_{t+1}(j,k) = \arg \max_{1 \leq i \leq N} \delta_t(i,j) \, a_{ijk}, \qquad t = 2,3,\ldots,T-1, \qquad 1 \leq j,k \leq N.$$

3. Terminación:

$$(q^*_{T-1}, q^*_T) = \arg \max_{1 \leq j,k \leq N} \delta_T(j,k).$$

4. Construcción hacia atrás de la secuencia de estados:

$$q^*_t = \psi_{t+2}(q^*_{t+1}, q^*_{t+2}), \qquad t = T-2, T-3, \ldots, 1.$$

Obsérvese que resulta necesario almacenar tanto $a_{ij} = P(j|i)$, la distribución de probabilid
de las transiciones entre cada posible par de estados, utilizada en la ecuación (3.10), segun
paso de inicialización, como $a_{ijk} = P(k|i,j)$, la distribución de probabilidad de las transicion

tmo. Por ejemplo, en el caso de los HMM,s de orden 2, el conjunto de estados del modelo ine como el producto cartesiano del conjunto de etiquetas, siendo válidas las transiciones un estado (t^i, t^l) y otro (t^m, t^k), donde todas las t^j son etiquetas, sólo cuando $l = m$. En uier caso, esto es lo que se conoce como modelo de *trigramas* de etiquetas, ampliamente nciado y utilizado por la mayoría de los etiquetadores estocásticos.

or supuesto, cabría pensar que el hecho de aumentar el orden de un HMM podría bocar en una mejora de la representación del contexto necesario para la correcta tación de las palabras. Es decir, ¿por qué considerar sólo 2 etiquetas hacia atrás, en de 3, ó 4, ó incluso más? Esto no se lleva a cabo en la práctica porque las unidades formación mayores que el trigrama generalmente modelizan fenómenos lingüísticos muy ejos y que afectan a muy pocos idiomas, resultando por ello suficientemente adecuado el e los trigramas. Por otra parte, el gran problema de aumentar el orden de un HMM es que desorbitadamente el número de parámetros que es necesario estimar para hacer operativo delo, tal y como veremos más adelante.

2 Implementaciones alternativas del algoritmo de Viterbi

que el algoritmo de Viterbi no calcula una probabilidad exacta, sino que construye ecuencia de estados, podemos tomar logaritmos sobre los parámetros del modelo e mentarlo sólo con sumas, sin necesidad de ninguna multiplicación. De esta manera, no e evita el problema de la rápida pérdida de precisión debida a las multiplicaciones de os muy pequeños, sino que además la velocidad de ejecución aumenta ya que las sumas lizan más rápido que las multiplicaciones. En la práctica, es particularmente importante ner de una implementación muy eficiente de este algoritmo, porque es el que realmente ta las palabras que aparecen en los textos, mientras que el proceso de estimación de los metros del modelo, que veremos en la siguiente sección, puede realizarse de manera previa arada, y por tanto su velocidad no es tan crítica. Los pasos del nuevo algoritmo son los ntes.

ritmo 3.5 Cálculo de la secuencia de estados más probable para una secuencia de vaciones dada (algoritmo de Viterbi con logaritmos y sumas):

Preproceso:

$$\begin{array}{ll} \tilde{\pi}_i = \log(\pi_i), & 1 \le i \le N. \\ \tilde{a}_{ij} = \log(a_{ij}), & 1 \le i, j \le N. \\ \tilde{b}_i(o_t) = \log[b_i(o_t)], & 1 \le i \le N,\ 1 \le t \le T. \end{array}$$

Inicialización:

$$\tilde{\delta}_1(i) = \log[\delta_1(i)] = \tilde{\pi}_i + \tilde{b}_i(o_1), \qquad 1 \le i \le N.$$

Recurrencia:

$$\tilde{\delta}_{t+1}(j) = \log[\delta_{t+1}(j)] = \left[\max_{1 \le i \le N}[\tilde{\delta}_t(i) + \tilde{a}_{ij}]\right] + \tilde{b}_j(o_{t+1}), \quad t = 1, 2, \ldots, T-1, \quad 1 \le j \le N.$$

$$\psi_{t+1}(j) = \arg\max_{1 \le i \le N}[\tilde{\delta}_t(i) + \tilde{a}_{ij}], \qquad t = 1, 2, \ldots, T-1, \qquad 1 \le j \le N.$$

4. Construcción hacia atrás de la secuencia de estados:

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \qquad t = T-1, T-2, \ldots, 1.$$

Esta implementación alternativa requiere del orden de N^2T sumas, más los cálculos necesari para el paso de preproceso. No obstante, dado que el paso de preproceso normalmente realiza una sola vez y se almacenan los resultados, su coste es despreciable en la mayoría de l aplicaciones.

Por otra parte, el tiempo de procesamiento del algoritmo de Viterbi también se puede redu introduciendo una búsqueda con corte[3]: cada estado que recibe un valor δ menor que el δ máxi hasta ese instante dividido por un cierto umbral θ queda excluido de los cálculos. Por supuest la incorporación de esta búsqueda con corte ya no garantiza que el algoritmo de Viterbi encuent la secuencia de estados de máxima probabilidad. Sin embargo, para propósitos prácticos y c una buena elección del umbral θ, no existe virtualmente ninguna diferencia de precisión ent los algoritmos con y sin corte. Empíricamente, un valor de $\theta = 1.000$ puede aproximadamen doblar la velocidad de un etiquetador sin afectar a la precisión [Brants 2000].

3.4.3 Estimación de los parámetros del modelo

Dada una secuencia de observaciones O, el tercer problema fundamental relativo a los HMM consiste en determinar los parámetros del modelo que mejor *explican* los datos observad Para el caso que nos ocupa, que es el de la etiquetación, dichas observaciones se correspond con las palabras de un texto en lenguaje natural. En general, existen dos posibles métod de estimación claramente diferenciados, en función de si el texto que observamos ha sido previamente etiquetado o no.

Con el fin de seguir el orden cronológico en el que han ido apareciendo las distintas técnic consideraremos primero el caso en el que el texto que observamos no está etiquetado. E este caso, es posible realizar un proceso de estimación, que denominaremos *no visible* o supervisado, mediante el algoritmo de Baum-Welch. Sin embargo, veremos que un texto etiquetado por sí solo no es suficiente para entrenar el modelo. Es necesario contar también c la ayuda de otro recurso lingüístico en forma de diccionario, que permita obtener un buen pun de inicialización para dicho algoritmo. Posteriormente, a medida que aparecieron disponibles l textos etiquetados, fue surgiendo también otro tipo de proceso de estimación que denominarem *visible* o supervisado.

3.4.3.1 Estimación no supervisada: algoritmo de Baum-Welch

Cuando se dispone de una observación O, formada por un texto que no ha sido previamen etiquetado, este tercer problema de la estimación de los parámetros es el más complejo, ya q no se conoce ningún método analítico definitivo para encontrar un modelo $\mu = (\pi, A, B)$ q maximice $P(O|\mu)$. Sin embargo, podemos elegir un modelo que maximice localmente dic probabilidad mediante un procedimiento iterativo tal como el algoritmo de Baum-Welch, que un caso especial del algoritmo EM[4] [Dempster *et al.* 1977].

[3] *Beam search* o *purging*.

[4] *Expectation-Maximization* (maximización de la esperanza); la parte E del algoritmo de Baum-Welch se cono

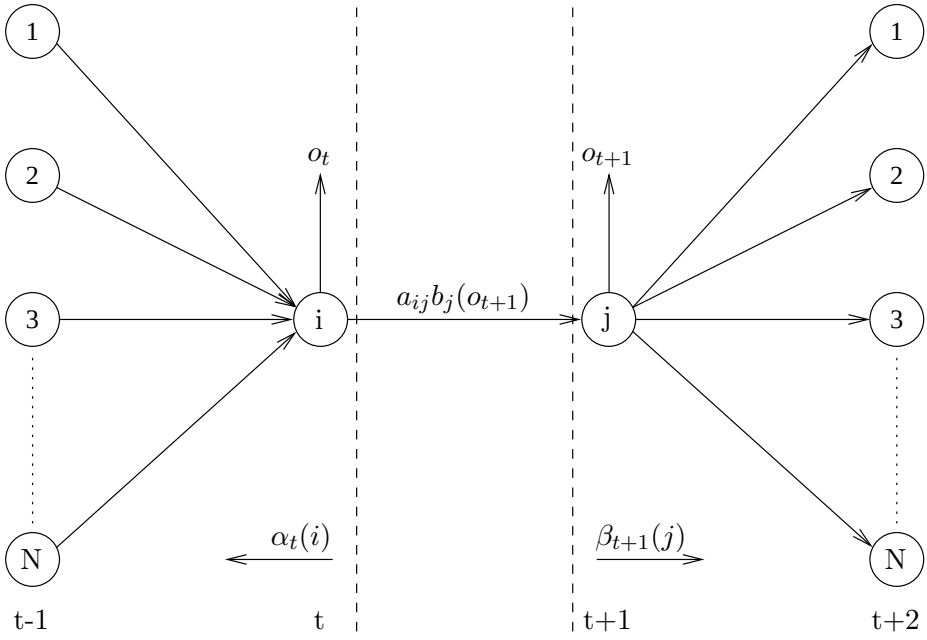
s más probables. Incrementando la probabilidad de esas transiciones y de esos símbolos, uimos un modelo revisado, el cual obtendrá una probabilidad mayor que el modelo or para la secuencia de observaciones dada. Este proceso de maximización, denominado lmente *proceso de entrenamiento*, se repite un cierto número de veces. Finalmente, el tmo se detiene cuando no consigue construir un modelo que mejore la probabilidad de la ncia de observaciones dada.

ra describir formalmente este procedimiento, definimos primero $\xi_t(i,j)$, la probabilidad de en el estado i en el instante t y en el estado j en el instante $t+1$, dada la observación y el modelo, es decir,

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j | O, \mu). \tag{3.11}$$

aminos que satisfacen las condiciones requeridas por la ecuación (3.11) son los que se ran en la figura 3.8. Por tanto, a partir de las definiciones de las variables hacia adelante a atrás, podemos escribir $\xi_t(i,j)$ de la forma

$$) = \frac{P(q_t = i, q_{t+1} = j, O|\mu)}{P(O|\mu)} = \frac{\alpha_t(i)\, a_{ij}\, b_j(o_{t+1})\, \beta_{t+1}(j)}{P(O|\mu)} = \frac{\alpha_t(i)\, a_{ij}\, b_j(o_{t+1})\, \beta_{t+1}(j)}{\sum_{k=1}^{N}\sum_{l=1}^{N} \alpha_t(k)\, a_{kl}\, b_l(o_{t+1})\, \beta_{t+1}(l)}.$$



a 3.8: Detalle de la secuencia de operaciones necesarias para el cálculo de la probabilidad nta de que el sistema esté en el estado i en el instante t y en el estado j en el instante $t+1$

amente, en las ecuaciones (3.5) y (3.6), habíamos definido también $\gamma_t(i)$ como la bilidad de estar en el estado i en el instante t, dada la secuencia de obsevaciones y dado

$$\overline{j=1}$$

Si sumamos $\gamma_t(i)$ a lo largo del tiempo, obtenemos un valor que se puede interpretar como
número esperado de veces que se visita el estado i, o lo que es lo mismo, el número esperado
transiciones hechas desde el estado i, si eliminamos del sumatorio el instante de tiempo $t =$
De manera similar, el sumatorio de $\xi_t(i,j)$ sobre t, también desde $t = 1$ hasta $t = T-1$, se pue
interpretar como el número esperado de transiciones desde el estado i al estado j. Es decir,

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{número esperado de transiciones desde el estado } i \text{ en } O,$$

$$\sum_{t=1}^{T-1} \xi_t(i,j) = \text{número esperado de transiciones desde el estado } i \text{ al estado } j \text{ en } O.$$

Utilizando estas fórmulas, se puede dar un método general para reestimar los parámetros de
HMM.

Algoritmo 3.6 Reestimación de los parámetros de un HMM (algoritmo de Baum-Welch).
conjunto de ecuaciones para la reestimación de π, A y B es el siguiente:

$$\bar{\pi}_i = \text{frecuencia esperada de estar en el estado } i \text{ en el primer instante } = \gamma_1(i),$$

$$\bar{a}_{ij} = \frac{\text{número esperado de transiciones desde el estado } i \text{ al estado } j}{\text{número esperado de transiciones desde el estado } i} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)},$$

$$\bar{b}_j(v_k) = \frac{\text{número esperado de veces en el estado } j \text{ observando el símbolo } v_k}{\text{número esperado de veces en el estado } j} = \frac{\sum_{t=1}^{T} \gamma_t(j)}{\text{tal que } o_t = v_k}$$
$$\sum_{t=1}^{T} \gamma_t(j)$$

Si definimos el modelo actual como $\mu = (\pi, A, B)$ y lo utilizamos para calcular los término
que aparecen en las partes derechas de las tres ecuaciones anteriores, y definimos el mode
reestimado como $\bar{\mu} = (\bar{\pi}, \bar{A}, \bar{B})$ a partir de los valores obtenidos para las partes izquierdas
dichas ecuaciones, está demostrado por Baum que: o bien (1) el modelo inicial μ define
punto crítico para la función de probabilidad, en cuyo caso $\bar{\mu} = \mu$; o bien (2) el modelo $\bar{\mu}$ es m
probable que el modelo μ en el sentido de que $P(O|\bar{\mu}) > P(O|\mu)$, esto es, hemos encontrado
nuevo modelo $\bar{\mu}$ para el cual la probabilidad de generar la secuencia de observaciones es mayo
Así pues, reemplazando μ por $\bar{\mu}$ y repitiendo la reestimación de los parámetros un cierto núme
de veces, podemos mejorar la probabilidad de O en cada iteración, hasta que no se apre

eja, no lineal y con numerosos máximos locales. Por tanto, para encontrar el máximo , una posible aproximación es intentar inicializar el modelo en una región del espacio de metros cercana a ese máximo global. En definitiva, una inicialización aleatoria podría dejar blema de la reestimación de parámetros demasiado abierto y la manera de restringir el ema es elegir unos parámetros adecuados, en lugar de fijarlos aleatoriamente.

gunos autores consideran que en la práctica, una estimación inicial de los parámetros π on números aproximadamente iguales, ligeramente perturbados para evitar los máximos s, pero que respeten las restricciones estocásticas estándar y que sean siempre distintos de suele ser normalmente satisfactoria para que posteriormente el algoritmo de Baum-Welch da por sí solo la regularidad de las etiquetas, es decir, todo lo relativo a los transiciones los estados del modelo. Sin embargo esto no es más que una confusión que parte de la cia general de que cuando un sistema tiene la capacidad de aprender, debe aprenderlo por sí mismo. Esto es efectivamente falso. Cuanta más información se le proporcione mente al sistema, mejor[5]. Es decir, en nuestro caso, se ha demostrado que siempre se obtener mejores resultados cuando se inicializan los parámetros π y A a partir de un etiquetado, por pequeño que sea. Dicho texto podría ser construido manualmente si no pone de ninguno. Incluso repetir unas cuantas veces la estrategia de construir un modelo , etiquetar otra pequeña porción de texto, corregirla manualmente, y utilizarla para estimar evo los parámetros del modelo, ofrecería mejores resultados que la aplicación a ciegas del lo de Baum-Welch. En definitiva, lo que queremos señalar es que la división entre métodos imación visible y no visible no es en nuestro caso tan estricta como la estamos presentando O si se prefiere, podríamos decir que, en sentido estricto, la estimación totalmente no e no existe.

n cualquier caso, cuando no se dispone de textos etiquetados, o cuando estos son muy ños, resulta importante obtener un buen valor inicial para el parámetro B, es decir, para obabilidades de emisión de las palabras, lo cual puede hacerse mediante el uso de un nario. Existen dos métodos generales para realizar esta inicialización: el método de Jelinek étodo de Kupiec. A continuación esbozamos cada uno de ellos:

Método de Jelinek. Si definimos $Q(v_k)$ como el número de etiquetas permitidas para la palabra v_k en el diccionario, $C(v_k)$ como el número de apariciones de la palabra v_k en el corpus de entrenamiento, y $b_j^*(v_k)$ como

$$b_j^*(v_k) = \begin{cases} 0 & \text{si } j \text{ no es una etiqueta permitida para la palabra } v_k \\ \frac{1}{Q(v_k)} & \text{en caso contrario,} \end{cases}$$

r supuesto, esta idea de considerar como modelo más preciso el que más se acerca a los datos disponibles, al extremo, degenera en un tratamiento manual específico para cualquier nuevo corpus, lo cual no es en o realista. Ésta es la razón por la cual se desarrollan modelos abstractos y técnicas de aprendizaje: para que emas se adapten rápidamente a los nuevos datos y sean capaces de manejarlos adecuadamente. Por tanto, e encontrar un equilibrio entre: por un lado, la descripción del modelo (y la dificultad de parametrizarlo), otro, su posibilidad de adaptación (es decir, su aprendizaje) con respecto al volumen de datos. Desde el de vista de la estadística y del aprendizaje, esto se conoce como el *Bias-Variance balance* (balance del sesgo

$$\sum_m b_j(v_m)\,C(v_m)$$

Es decir, el método de Jelinek inicializa las probabilidades de emisión del modelo median el uso de la regla de Bayes, estimando mediante la frecuencia observada la probabilid de aparición de una palabra y suponiendo que todas las etiquetas que aparecen en diccionario para una palabra dada son igualmente probables [Jelinek 1985].

- **Método de Kupiec**. La otra alternativa es agrupar las palabras en clases de ambigüeda de tal manera que todas aquellas palabras que tengan el mismo conjunto de etiquet permitidas en el diccionario pertenezcan a la misma clase o *metapalabra* u_L, donde L un subconjunto de Q, el conjunto de etiquetas utilizado. Es decir, si $L(v_k)$ es el conjun de todas las etiquetas posibles para la palabra v_k, entonces

$$u_L = \{v_k \mid L = L(v_k)\}, \qquad \forall L \subseteq Q.$$

Por ejemplo, la metapalabra $u_{\{\texttt{Scms,P}\}}$ contendrá todas las palabras para las cuales diccionario permite las etiquetas `Scms`, es decir, sustantivo común masculino singul y `P`, es decir, preposición, y ninguna otra etiqueta más. Entonces, a cada una de est metapalabras se le da un tratamiento similar al del método de Jelinek:

$$b_j^*(u_L) = \begin{cases} 0 & \text{si } j \notin L \\ \frac{1}{|L|} & \text{en caso contrario,} \end{cases}$$

donde $|L|$ es el cardinal del conjunto L, y

$$b_j(u_L) = \frac{b_j^*(u_L)\,C(u_L)}{\sum_{L'} b_j^*(u_{L'})\,C(u_{L'})}$$

donde $C(u_L)$ (respectivamente $C(u_{L'})$) es el número de palabras pertenecientes a (respectivamente $u_{L'}$) que aparecen en el corpus de entrenamiento. La implementaci real utilizada por Kupiec es una variante de la presentada aquí, pero se ha expuesto esta manera para hacer más transparente la similitud entre ambos métodos. Por ejemp Kupiec no incluye las 100 palabras más frecuentes dentro de las clases de ambigüedad, si que las trata separadamente como clases de una sola palabra, con el fin de no introdu errores y mejorar así la estimación inicial [Kupiec 1992].

La ventaja del método de Kupiec sobre el de Jelinek es que no necesita afinar una probabilid de emisión diferente para cada palabra. Mediante la definición de estas clases de ambigüeda el número total de parámetros se reduce substancialmente y éstos se pueden estimar de u manera más precisa. Por supuesto, esta ventaja podría convertirse en desventaja si existe suficiente cantidad de material de entrenamiento como para estimar los parámetros palabra p palabra tal y como hace el método de Jelinek.

En el algoritmo de Baum-Welch también es necesario hacer algo para evitar el proble de la pérdida de precisión en punto flotante. Sin embargo, la necesidad de realizar sum dificulta el uso de logaritmos. Una solución bastante común es utilizar unos coeficient

umarlos.

itmo 3.7 Función para la suma de logaritmos. Dados $x = \log(x')$ e $y = \log(y')$, la función a $\log(x' + y')$:

```
function Sumar_Logaritmos (x, y) =
  begin
    if (y − x > log C) then
      return y
    else
      if (x − y > log C) then
        return x
      else
        return min (x, y) + log (exp (x − min (x, y)) + exp (y − min (x, y)))
  end;
```

ta porción de pseudo-código, C representa una constante grande, del orden de 10^{30}. De nanera, lo que estamos haciendo es calcular un factor de escalado apropiado en el momento lizar cada suma, y lo único que resta es tener cuidado con los errores de redondeo. □

2 Estimación supervisada: métodos de suavización

do se dispone de un texto etiquetado, la primera idea que acude a nuestra mente para ar el proceso de estimación de los parámetros del modelo es quizás la de diseñar un sencillo nismo basado en el uso de frecuencias relativas. Pero antes de comentar dicho mecanismo, s a introducir un cambio en la notación que hace referencia a la definición formal de los ,s.

asta aquí, la notación que hemos utilizado estaba inspirada en la excelente presentación Rabiner llevó a cabo en relación con este tipo de procesos estocásticos [Rabiner 1989]. vamente, se trata de una notación genérica sobre HMM,s especialmente orientada a bir con el máximo detalle la implementación de determinados algoritmos, como pueden de Viterbi o el de Baum-Welch. Sin embargo, una vez que hemos centrado la discusión el tema concreto de la etiquetación, algunos aspectos de esa notación no se identifican con ro problema todo lo bien que nos gustaría. Por poner un ejemplo, los estados de un HMM n sido denotados simplemente con los números enteros del conjunto $\{1, 2, \ldots, N\}$. Esto a mucho la expresión de operaciones tales como la indexación de las celdas de la matriz A nsiciones entre estados, pero no nos permite hacer referencia a las etiquetas concretas que án utilizando, si no es a través de la consideración de una función correspondencia entre s conjuntos.

notación que proponemos ahora está más acorde con la que se adopta en la mayoría de ferencias bibliográficas dedicadas específicamente al tema de la etiquetación.

nición 3.2 Básicamente, la nueva notación describe cada uno de los elementos de un HMM sigue:

Considerando la inicial de la palabra inglesa *tag* (etiqueta), denotaremos mediante t^i la

del conjunto de etiquetas utilizado, tal y como ya habíamos esbozado.

2. Anteriormente, habíamos denotado mediante v_k cada uno de los símbolos de observaci correspondientes a la parte *visible* del modelo, y habíamos visto que, en el caso de etiquetación, dichos sucesos observables se corresponden con las palabras. Pues bie de igual manera, considerando la inicial del vocablo inglés *word* (palabra), denotarem ahora mediante w^i la i-ésima palabra del diccionario, y mediante w_i la palabra que ocu la posición i dentro de la frase.

3. Habíamos indicado también que trabajaremos siempre en el nivel de frase, y q consideraremos un estado o etiqueta especial t^0 para marcar el inicio y el final de cada u de las frases. Por tanto, respecto al parámetro π, la distribución de probabilidad del esta inicial, tendremos que $\pi(t^0)$ será igual a 1, y $\pi(t^i)$ será igual a 0, para todo $i = 1, 2, \ldots,$

4. Por lo que respecta al parámetro A, la distribución de probabilidad de las transicion entre estados, basta aclarar que, de acuerdo con la nueva notación, la información q realmente está almacenada en cada una de las celdas de esta matriz es $P(t^j|t^i)$ en el ca de los modelos basados en bigramas, y $P(t^k|t^i\, t^j)$ en el caso de los modelos basados trigramas.

5. Por último, respecto a B, las probabilidades de emisión de las palabras, simplemen indicamos que la información que antes se denotaba mediante $b_j(v_k)$ se escribe aho como $P(w^k|t^j)$.

Para terminar, completamos esta notación con algunos aspectos que nos permitirán avanzar m cómodamente con la exposición. Denotaremos mediante $C(x)$ el número de veces que el suce x aparece en el corpus de entrenamiento. Por ejemplo, $C(t^i, t^j, t^k)$ representa el número de vec que aparece el trigrama $t^i\, t^j\, t^k$. De igual manera, $C(w^k|t^j)$ representa el número de veces q aparece la palabra w^k etiquetada como t^j. Denotaremos también mediante $f(x)$ la frecuenc del suceso x en el corpus de entrenamiento. Y finalmente denotaremos mediante $\hat{p}(x)$ nuest estimación de la probabilidad real $P(x)$.

Nuevo modelo probabilístico. La etiquetación se describe entonces, a través de esta nue notación, como el proceso de encontrar la mejor secuencia de etiquetas $t_{1,n}$ para una frase da de n palabras $w_{1,n}$. Aplicando la regla de Bayes, podemos escribir:

$$\arg\max_{t_{1,n}} P(t_{1,n}|w_{1,n}) = \arg\max_{t_{1,n}} \frac{P(w_{1,n}|t_{1,n})P(t_{1,n})}{P(w_{1,n})} = \arg\max_{t_{1,n}} P(w_{1,n}|t_{1,n})P(t_{1,n}).$$

Ahora, trabajamos sobre esta expresión para conseguir que utilice los parámetros que podem estimar directamente desde el corpus de entrenamiento. Para ello, además de la propiedad d horizonte limitado (3.14), utilizamos dos suposiciones más acerca de las palabras:

1. Las etiquetas no son independientes unas de otras, pero las palabras sí (3.12).

2. La identidad de una palabra depende sólo de su propia etiqueta (3.13).

Así pues, tenemos que:

$$= \left(\prod_{i=1} P(w_i|t_i)\right) \times P(t_n|t_{1,n-1}) \times P(t_{n-1}|t_{1,n-2}) \times \ldots \times P(t_2|t_1) = \quad (3.13)$$

$$= \left(\prod_{i=1}^{n} P(w_i|t_i)\right) \times P(t_n|t_{n-1}) \times P(t_{n-1}|t_{n-2}) \times \ldots \times P(t_2|t_1) = \quad (3.14)$$

$$= \prod_{i=1}^{n} [P(w_i|t_i) \times P(t_i|t_{i-1})].$$

anto, la ecuación final para determinar la secuencia de etiquetas óptima para una frase es

$$\hat{t}_{1,n} = \arg\max_{t_{1,n}} P(t_{1,n}|w_{1,n}) = \arg\max_{t_{1,n}} \prod_{i=1}^{n} [P(w_i|t_i) \times P(t_i|t_{i-1})]$$

os etiquetadores basados en bigramas, y

$$\hat{t}_{1,n} = \arg\max_{t_{1,n}} P(t_{1,n}|w_{1,n}) = \arg\max_{t_{1,n}} \prod_{i=1}^{n} [P(w_i|t_i) \times P(t_i|t_{i-2}, t_{i-1})]$$

os etiquetadores basados en trigramas. Estos cálculos son precisamente los que realiza el tmo de Viterbi, de tal manera que para disponer de un etiquetador totalmente operativo esta especificar de qué manera se pueden obtener los parámetros de funcionamiento del subyacente. La sección anterior presentaba un método para la estimación de dichos metros a partir de textos no etiquetados. Ahora nos ocuparemos del diseño de métodos de ación de parámetros a partir de textos etiquetados.

nemos entonces la idea inicial de diseñar un mecanismo intuitivo para estimar los metros de nuestro modelo. Dado que en este caso partimos de un texto previamente etado, podemos sugerir que el proceso de estimación de las transiciones entre estados podría arse en base a unos sencillos cálculos de frecuencias relativas. En el caso de un modelo o en bigramas, tendríamos que

$$\hat{p}(t^j|t^i) = f(t^j|t^i) = \frac{C(t^i, t^j)}{C(t^i)}.$$

caso de un modelo basado en trigramas, tendríamos que

$$\hat{p}(t^k|t^i\, t^j) = f(t^k|t^i\, t^j) = \frac{C(t^i, t^j, t^k)}{C(t^i, t^j)}.$$

ual manera, para la estimación de las probabilidades de emisión de las palabras, tendríamos

$$\hat{p}(w^k|t^j) = f(w^k|t^j) = \frac{C(w^k|t^j)}{C(t^j)}.$$

aquí, lo que hacemos no es realmente estimar, sino construir un modelo de manera e. Una vez que este modelo está preparado, podemos utilizarlo para etiquetar nuevos mediante la aplicación del algoritmo de Viterbi a cada una de sus frases, y la idea original

n palabras deba ser etiquetada aplicando el algoritmo de Viterbi sobre un enrejado de $373 \times$
estados, sino más bien sobre un enrejado simplificado como el que veíamos en la figura 3.7, don
se consideran sólo las posibles etiquetas de cada palabra. Lo que sí quiere decir es que, en
caso de un modelo basado en bigramas, la matriz A contendría $373 \times 373 = 139.129$ celdas, y s
embargo, en el mayor de los corpus de entrenamiento utilizados en dicho experimento, tan solo
pueden ver 4.037 de esas transiciones (es decir, el $2,90\%$), y las restantes celdas quedarían a cer
En el caso de un modelo basado en trigramas, la matriz A contendría $373 \times 373 \times 373 = 51.895.1$
celdas, sólo se pueden llegar a ver 23.119 (el $0,04\%$), y las restantes quedarían también a cer
Esto es debido al fenómeno de *dispersión de los datos*[7], el cual puede provocar también q
algunas de esas transiciones aparezcan como candidatas a la hora de etiquetar nuevas frases.

Así pues, teniendo en cuenta que los enrejados simplificados ya contemplan sólo
subconjunto reducido de todos los caminos posibles, parece muy arriesgado trabajar c
transiciones nulas, cuyo uso implica multiplicaciones por cero que anularían completamen
todos los caminos que pasen por ellas. Para evitar el problema de las transiciones nulas,
suelen utilizar *métodos de suavización*[8]. Estos métodos permiten que a partir de muestr
pequeñas se puedan estimar unas probabilidades más representativas del comportamiento re
de la población que estamos estudiando.

Una forma de implementar la suavización es mediante técnicas de *interpolación line*
En general, el esquema de suavizado mediante interpolación lineal funciona como sigue.
distribución $f(x)$ observada en un conjunto de E posibles sucesos se modifica añadiendo
valor muy pequeño procedente de una distribución menos específica $q(x)$ a la cual damos
peso o confianza α. Entonces, la distribución de probabilidad que nos interesa se aproxima
la siguiente forma:

$$\hat{p}(x) \approx \frac{f(x) + \alpha\, q(x)}{E + \alpha}.$$

Y si utilizamos el parámetro tradicional de interpolación $\lambda = \frac{\alpha}{E+\alpha}$, entonces también se verifi
la siguiente 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 estad
de un modelo basado en bigramas, esa distribución menos específica ponderada con α
interpolar puede ser la distribución de probabilidad de los *unigramas*, esto es, la distribución
probabilidad de cada etiqueta individualmente[9]. Por tanto, las probabilidades de los bigram
se pueden aproximar mediante

$$\hat{p}(t_i|t_{i-1}) = (1 - \lambda)\, f(t_i|t_{i-1}) + \lambda\, f(t_i)$$

[6]Se trata de un experimento realizado con la versión en español del corpus ITU (*Internatio*
Telecommunications Union CCITT Handbook). Los datos relativos a este experimento se encuentr
en [Graña 2000].

[7]También denominado fenómeno de *sparse data*.

[8]También denominados métodos de *smoothing*.

[9]Esta fórmula viene del hecho de que $P(X|Y) = P(X)$, si X e Y son independientes. Por eso decimos q
$P(X|Y)$ se estima mediante $f(X|Y)$, o en todo caso mediante una combinación lineal de $f(X|Y)$ y $f(X)$,
el bigrama $Y\,X$ aparece en el corpus de entrenamiento, y mediante $f(X)$ si no aparece, asumiendo la hipóte
de independencia para esos sucesos perdidos. Rigurosamente hablando esto no es correcto, pero constituye u

todos los *pesos* λ_i deben ser no negativos y deben satisfacer la restricción $\lambda_1+\lambda_2+\lambda_3=1$.
tinuación, y centrando ya toda nuestra atención en el modelo de trigramas, discutiremos
elegir los pesos λ_i de manera óptima.

izado lineal óptimo. Un modelo de lenguaje que opere de acuerdo con la ecuación (3.15)
ede ver realmente como un HMM. La figura 3.9 muestra un fragmento de este modelo. En
unto es importante aclarar la siguiente cuestión. Existe una variante de representación de
MM,s que lleva asociadas las probabilidades de emisión no a un único estado, sino tanto
ado origen como al estado destino de las transiciones. Es decir, dichas probabilidades
isión están asociadas realmente a los arcos o transiciones, no a los estados individuales.
os autores prefieren enfocar la introducción a los HMM,s comenzando con este tipo de
entación, ya que el paso de generalización desde los HMM,s con probabilidades de emisión
arcos a los HMM,s con probabilidades de emisión en los estados resulta más natural que el
nverso. A pesar de esto, nosotros hemos preferido manejar desde el principio HMM,s con
bilidades de emisión en los estados, porque este es el tipo de HMM,s que efectivamente se
en la práctica para la etiquetación de textos, y por tanto es posible establecer analogías
ambos conceptos desde el primer momento.
obstante, el HMM de la figura 3.9 es un HMM con probabilidades de emisión en los
Ocurre además que en este tipo de HMM,s está permitido el uso de transiciones que no
ningún símbolo, sino que simplemente se utilizan para cambiar de estado. Este tipo de
ciones se denominan *transiciones épsilon* o *transiciones vacías*. En la figura, no aparecen
entados los símbolos de salida, sino simplemente las probabilidades de cada arco. Por
para distinguir las transiciones normales de las transiciones épsilon, hemos representado
últimas mediante líneas discontinuas. Así pues, saliendo del estado más a la izquierda (t^i,t^j)
os tres transiciones vacías que van a los pseudo-estados $s_1(t^i,t^j)$, $s_2(t^i,t^j)$ y $s_3(t^i,t^j)$. Las
bilidades de estas transiciones son λ_1, λ_2 y λ_3, respectivamente. A continuación, saliendo
da uno de los tres pseudo-estados aparecen N transiciones. Cada una de ellas conduce a un
(t^j,t^k), $k=1,2,\ldots,N$, para generar la tercera etiqueta del trigrama. Las probabilidades
as transiciones son $f(t^k)$, $f(t^k|t^j)$ y $f(t^k|t^i\,t^j)$, $k=1,2,\ldots,N$, respectivamente. En la
, hemos etiquetado sólo las transiciones que entran en los estados (t^j,t^1) y (t^j,t^N). Nótese
esde el estado más a la izquierda (t^i,t^j) se puede alcanzar cada uno de estos estados a
de tres caminos posibles, y que por tanto la probabilidad total de llegar a cada estado
de con el cálculo de la ecuación (3.15).
r supuesto, otra forma de ver el problema podría ser considerando el HMM de la figura 3.9
ía como un HMM con probabilidades de emisión en los estados, salvo que ahora es a los
o-estados $s_1(t^i,t^j)$, $s_2(t^i,t^j)$ y $s_3(t^i,t^j)$ a los que no se les permite generar ninguna palabra.
alquier caso, tenemos que las probabilidades de las transiciones normales son conocidas,
ras que las de las transiciones épsilon, λ_1, λ_2 y λ_3, deben ser determinadas mediante algún
dimiento. El HMM completo es desde luego enorme: consta de $4\times N^2$ estados, donde N es
maño del conjunto de etiquetas utilizado. Pero de acuerdo con la ecuación (3.15), para un i
$i\in\{1,2,3\}$, todas las probabilidades λ_i de las transiciones que van desde el estado más a
uierda (t^i,t^j) hasta uno de los estados más a la derecha (t^j,t^k) tienen el mismo valor, sea
ea la combinación de etiquetas que forma el trigrama $t^i\,t^j\,t^k$. Se dice entonces que estas
bilidades toman *valores empatados*[10].

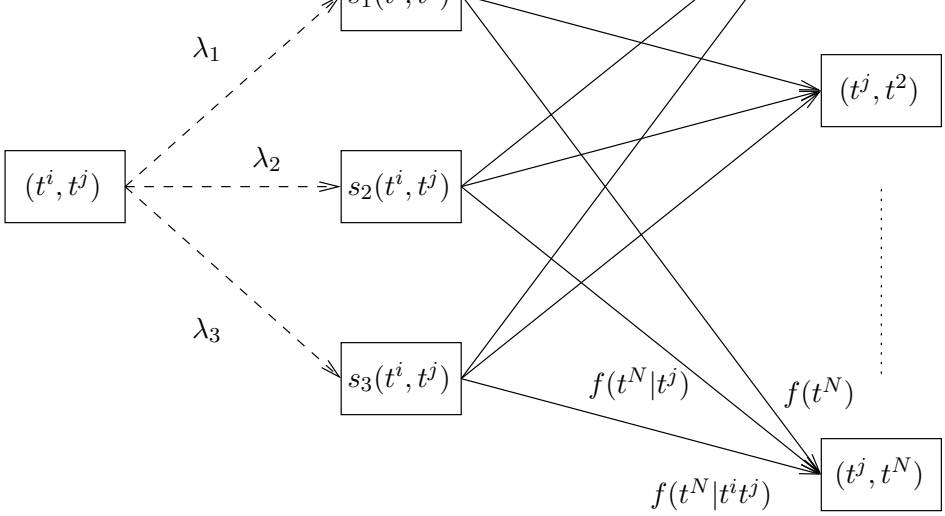


Figura 3.9: Fragmento del suavizado lineal de un HMM basado en trigramas

Pues bien, dado que hemos visto que el modelo de lenguaje al que obedece la ecuación (3.1 es un HMM, se puede utilizar el algoritmo de Baum-Welch para estimar los valores λ_i óptim Además, como una consecuencia favorable de que el algoritmo deba *empatar* o igualar l probabilidades λ_i, independientemente de a qué subparte del HMM pertenezcan, el proce de estimación sólo necesita manejar tres contadores, uno para cada tipo de transición épsilo del número esperado de veces que se cruza por una transición. Para el resto de transiciones, se necesita establecer ningún otro tipo de contador porque de hecho las probabilidades de es transiciones están ya fijadas de antemano. Debido a todo esto, se pueden sugerir ideas práctic que dan lugar a cálculos en este caso más sencillos que los del algoritmo de Baum-Welch, y q por tanto simplifican aún más la obtención de los λ_i óptimos [Jelinek 1997, pp. 66-69].

Pero ahora preferimos centrar nuestro interés en responder a una importante pregunta: ¿q tipo de datos de entrenamiento deberíamos usar para determinar los pesos λ_i? Está claro que pueden ser los mismos datos sobre los que se calculan las frecuencias $f(\ |\ )$, porque en ese caso l estimaciones darían como resultado $\lambda_3 = 1$ y $\lambda_1 = \lambda_2 = 0$. De hecho, $f(t^k|t^i\, t^j)$ es la estimaci de máxima verosimilitud de $P(t^k|t^i\, t^j)$ para los datos de entrenamiento en los que se basa dic frecuencia $f(t^k|t^i\, t^j)$. Sin embargo, mirando de nuevo la figura 3.9, se intuye que, de los tr caminos posibles que conducen a cada uno de los estados más a la derecha, el HMM ha inclui el camino superior para poder generar la tercera etiqueta t^k, incluso aunque el bigrama $t^j\, t^k$ haya aparecido en los datos de entrenamiento. Y de manera similar, mediante ese mismo cami superior y mediante el camino central, se hace posible la tercera etiqueta t^k, incluso aunque exista ninguna ocurrencia del trigrama $t^i\, t^j\, t^k$. Por tanto se concluye que el conjunto total los datos de entrenamiento debe ser dividido en dos porciones:

1. La primera, mucho más grande, denominada *datos de desarrollo*[11], se utiliza para estim las frecuencias relativas $f(\ |\ )$.

upuesto, una vez que se ha realizado este proceso, el modelo obtenido se puede mejorar nando ambas porciones de datos y reestimando las frecuencias $f(\ |\ )$. Esta técnica de zado lineal se denomina *interpolación de borrado*[13].

interpolación de borrado se puede realizar también mediante una eliminación sucesiva da trigrama del corpus de entrenamiento, y una posterior estimación de los λ_i óptimos a del resto de n-gramas del corpus. Conocidos los contadores de frecuencias para unigramas, mas y trigramas, los pesos λ_i se pueden determinar eficientemente a través del siguiente tmo.

ritmo 3.8 Cálculo de los parámetros λ_1, λ_2 y λ_3 de un esquema de interpolación lineal, idas las frecuencias de unigramas, bigramas y trigramas, y conocido N, el tamaño del s de entrenamiento [Brants 2000]:

Inicializar $\lambda_1 = \lambda_2 = \lambda_3 = 0$.

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.

Normalizar los valores λ_1, λ_2 y λ_3.

ando un poco de la notación, con el fin de no complicar en exceso las ecuaciones con la ncia de tantos subíndices, hemos denotado el trigrama $t_{i-2}\,t_{i-1}\,t_i$ mediante $t_1\,t_2\,t_3$, donde anera obvia los números 1, 2 y 3 hacen referencia a la posición de cada etiqueta dentro igrama. Si el denominador de alguna de las expresiones es 0, se define el resultado de xpresión como 0. Restar 1 en este algoritmo es la manera de tener en cuenta los datos no vados. Sin esta resta, el modelo efectivamente sobreestimaría los datos de entrenamiento, ando $\lambda_3 = 1$ y $\lambda_1 = \lambda_2 = 0$ y produciendo peores resultados. □

emos visto que la ecuación (3.15) propone valores constantes para los parámetros de olación λ_1, λ_2 y λ_3. Es cierto que quizás no es una buena idea utilizar siempre los mismos s, pero es cierto también que la consideración de un conjunto de valores λ_i, $i \in \{1, 2, 3\}$, cada posible par de etiquetas eleva muchísimo el número de parámetros del modelo, lo cual o no introduce ninguna mejora en relación con el fenómeno de los datos dispersos, sino mpeora el problema.

o obstante, algunos autores han sugerido que al menos sí sería conveniente una agrupación bigramas en un número moderado de clases y una posterior estimación de un conjunto ores λ_i distinto para cada clase, aunque no queda claro cuál sería un buen criterio para la ción de esas clases.

rants es quizás el único autor que proporciona datos concretos sobre diferentes experimentos rupación. Uno de ellos incluía un conjunto de valores λ_i distinto para cada frecuencia. Otra

ld-out data.

con algunas de las agrupaciones consideradas. Por esta razón, Brants utiliza en su etiquetad interpolación lineal independiente del contexto, es decir, valores constantes para los parámetr λ_i [Brants 2000].

Otros autores, sin embargo, proponen una reducción del número de parámetros a estim mediante la especificación de que ciertos sucesos son absolutamente improbables, es decir, tien probabilidad 0, lo cual permite introducir *ceros estructurales* en el modelo. Efectivamente, hecho de hacer que algunos sucesos no sean posibles, por ejemplo un trigrama formado p tres preposiciones seguidas, añade gran cantidad de estructura al modelo, mejora el rendimien del proceso de entrenamiento, y por tanto facilita en gran medida la labor de estimación parámetros. Pero esto resulta apropiado sólo en algunas circunstancias y no con espacios estados tan grandes.

En cualquier caso, la interpolación lineal no es la única manera de enfrentarse al proble de la poca frecuencia de determinados sucesos en los datos de entrenamiento. Existen otr métodos de estimación que, aunque son también dependientes del número de veces que aparec los sucesos en el corpus de entrenamiento, no se basan en absoluto en frecuencias relativa Como consecuencia, se han sugerido muchas fórmulas que son capaces de asignar probabilidad distintas de cero a los sucesos no observados, permitiendo así hacer una importante distinci entre este tipo de *ceros no observados*, y los *ceros reales* que serían los correspondientes los sucesos efectivamente no posibles. Entre todas ellas, la más conocida es la fórmula Good-Turing[14].

3.4.3.3 Estimación combinada mediante métodos híbridos

Los etiquetadores basados en HMM,s trabajan bien cuando se dispone de un corpus entrenamiento suficientemente grande. A menudo éste no es el caso. Suele ocurrir con frecuenc que queremos etiquetar un texto de un dominio especializado donde las probabilidades de emisi de las palabras son diferentes de las que se pueden observar en los textos de entrenamiento. Otr veces necesitamos etiquetar textos de idiomas para los cuales simplemente no existen *corpo* etiquetados. En estos casos, hemos visto anteriormente que se puede utilizar un procedimien de estimación no visible o no supervisado a partir de una inicialización pseudo-aleatoria de l parámetros del modelo y de la posterior aplicación del algoritmo de Baum-Welch.

Los fundamentos teóricos sugieren que el algoritmo de Baum-Welch debe detenerse cuan no se puede construir un modelo que mejore la probabilidad de la secuencia de entrenamien dada. Sin embargo, ha sido demostrado que, para el caso de la etiquetación, este criterio menudo produce como resultado un sobreentrenamiento del modelo. Este fenómeno ha si estudiado en profundidad por Elworthy, quien entrenó diferentes HMM,s considerando una gr variedad de condiciones de inicialización y de distintos números de iteraciones [Elworthy 199

- En ocasiones, el proceso de entrenamiento se mostraba estable y siempre producía mejor del rendimiento después de cada iteración, demostrando que el criterio probabilístico e apropiado para esos casos.

[14]La idea intuitiva del método de estimación de Good-Turing es que, en lugar de estimar $P(X)$, intentam estimar $P(X|C)$, donde C es una clasificación de sucesos definida *a priori* (en nuestro caso, los sucesos X q aparecen en el corpus de entrenamiento exactamente C veces). De esta manera, es más sencillo estimar $P(X)$ en concreto se puede estimar $P(X|0)$, que es precisamente la probabilidad de los sucesos que no aparecen en corpus. En resumen, el método de Good-Turing corrige la estimación de máxima verosimilitud en base al núm

ción de los HMM,s: cuando se dispone de un diccionario, pero no se dispone de ningún
s etiquetado. Por tanto, en este tipo de situaciones, los cuidados se deben extremar al
no con el fin de no sobreentrenar el modelo. Una forma de conseguir esto es validar dicho
o después de cada iteración sobre un conjunto de datos separado (*held-out data*), y parar
renamiento cuando el rendimiento empieza a decrecer.
worthy también confirmó los resultados encontrados por Merialdo, que demuestran que
dipone inicialmente de un corpus etiquetado, la aplicación del algoritmo de Baum-Welch
degradar el rendimiento ya desde la primera iteración [Merialdo 1994]. Sin embargo, una
aridad interesante es el hecho de que si el texto de entrenamiento y el de validación son
diferentes, unas pocas iteraciones podrían producir mejoras. Además, esto suele ocurrir
ecuencia en la práctica, ya que a menudo tenemos que enfrentarnos con tipos de textos
os cuales no se dispone de *corpora* de entrenamiento etiquetados similares. En resumen:

Si existe un texto de entrenamiento suficientemente grande y similar a los textos con los que se va a trabajar, entonces se debería utilizar un procedimiento de estimacion totalmente visible o supervisado.

Si no existe ningún texto de entrenamiento disponible, o si los textos de entrenamiento y validación son muy diferentes, entonces se debería aplicar el algoritmo de Baum-Welch durante unas pocas iteraciones.

Sólo se debería de aplicar durante un número grande de iteraciones, 10 ó más, cuando no hay ningún tipo de información léxica disponible.

este último caso, no cabe esperar un buen rendimiento. Esto no se debe a ningún defecto goritmo de Baum-Welch, sino al hecho de que dicho algoritmo tan sólo realiza ajustes de rámetros del HMM con el fin de maximizar la probabilidad de los datos de entrenamiento. ambios que realiza para reducir la entropía cruzada de esos datos podrían no estar de do con nuestro objetivo real, que es el de asignar a las palabras etiquetas pertenecientes a njunto predefinido. Por tanto, esta técnica no siempre es capaz de optimizar el rendimiento a tarea particular de la etiquetación.

4 Integración de diccionarios

oblema similar al de las transiciones entre estados ocurre también con las palabras. En imer momento podemos pensar que si estamos hablando de entrenamiento puramente visado, las probabilidades de generación de las palabras son lo único que sí se puede ar perfectamente. Sin embargo, en la práctica, podemos encontrarnos con palabras que arecen en los textos de entrenamiento, pero que quizás sí las tenemos presentes en un nario. Es decir, se trata de palabras para las cuales conocemos sus posibles etiquetas, pero no haber sido vistas durante el entrenamiento, no tienen definidas sus probabilidades de ón. Una vez más, no resulta conveniente dejar a cero esas probabilidades.
or tanto, se hace necesario el uso de métodos que permitan la correcta integración información aportada por el diccionario con los datos proporcionados por el texto de namiento. Existen dos métodos generales para realizar esta integración:

Método *Adding One* de Church. El método *Adding One* o método *de sumar uno*

tiene nada que ver con el orden que siguen en las frases reales. De esta manera, para to
par (w^k, t^j) presente en el diccionario, la probabilidad de emisión se estima mediante

$$\hat{p}(w^k|t^j) = \frac{C(w^k|t^j) + 1}{C(t^j) + K_j}$$

donde K_j es el número de palabras etiquetadas con t^j que aparecen en el diccionar
mientras que para el resto de pares palabra-etiqueta que aparezcan en el corpus
entrenamiento y no en el diccionario la estimación se realiza mediante la misma ecuació
pero sin sumar 1 en el numerador. Intuitivamente, volviendo al ejemplo de las urnas y l
bolas, el método opera como si todas y cada unas de las bolas o palabras del diccionar
fueran colocadas en sus urnas o etiquetas correspondientes una sola vez, como paso prev
a la estimación de las probabilidades de emisión. Con esto se produce el efecto deseado
que las probabilidades de emisión de las palabras del diccionario no queden a cero, inclu
aunque no hayan aparecido en el corpus de entrenamiento.

- **Método de Good-Turing**. Como ya hemos comentado anteriormente, éste es un méto
de estimación que no está basado en frecuencias relativas, pero que también es capaz
asignar probabilidades distintas de cero a los sucesos no observados. En este contex
un suceso posible pero no observado es cualquier par palabra-etiqueta que aparece en
diccionario y no aparece en el corpus de entrenamiento (cero no observado), mientras q
un suceso no posible es cualquier par palabra-etiqueta que no aparece ni en el diccionar
ni en el corpus de entrenamiento (cero real).

Es importante recordar que el uso de un diccionario externo puede ayudar a incrementar
rendimiento del proceso de etiquetación.

3.4.3.5 Tratamiento de palabras desconocidas

Hemos visto anteriormente cómo estimar las probabilidades de emisión para las palabras q
aparecen o bien en el corpus, o bien en nuestro diccionario, o bien en ambos. Pero es seguro q
al intentar etiquetar nuevas frases encontraremos multitud de palabras que no han aparecid
previamente en ninguno de esos recursos. Hemos visto también que un conocimiento *a pri*
de la distribución de etiquetas de una palabra, o al menos de la proporción para la etique
más probable, supone una gran ayuda para la etiquetación. Esto significa que las palabr
desconocidas son el mayor problema de los etiquetadores, y en la práctica la diferencia
rendimiento entre unos y otros puede ser debida a la proporción de palabras desconocidas y
procedimiento de adivinación y tratamiento de las mismas que cada etiquetador tenga integrad

La manera más simple de enfrentarse a este problema es suponer que toda palab
desconocida puede pertenecer a cualquier categoría gramatical, o quizás a una clase de categorí
abiertas, por ejemplo, sustantivos, adjetivos, verbos, etc., pero no preposiciones, ni artículos,
pronombres, que suponemos que pertenecen a otra clase de categorías *cerradas* y que tod
las palabras correspondientes a ellas son conocidas y están ya en el diccionario. Aunque es
aproximación podría ser válida en algunos casos, en general, el no hacer uso de la informaci
léxica de las palabras (prefijos, sufijos, etc.) para proponer un conjunto más limitado de posibl
etiquetas degrada mucho el rendimiento de los etiquetadores. Es por ello que existen numeros
trabajos orientados a explorar este tipo de características, y mejorar así la estimación de l

$$\hat{p}(w^k|t^j) = \frac{1}{Z}\, P(\text{desconocida}|t^j)\; P(\text{mayúscula}|t^j)\; P(\text{sufijo}|t^j)$$

donde Z es una constante de normalización. Este modelo reduce la proporción de error de palabaras desconocidas de más de un 40% a menos de un 20% [Weischedel *et al.* 1993]. Charniak propuso un modelo alternativo que utiliza tanto las raíces como los sufijos [Charniak *et al.* 1993].

La mayor parte del trabajo que se realiza en relación con las palabras desconocidas asume que las propiedades consideradas son independientes. Dicha independencia no siempre es una buena suposición. Por ejemplo, las palabras en mayúsculas tienen más probabilidad de ser desconocidas, y por tanto las propiedades *mayúscula* y *desconocida* del modelo de Weischedel no son realmente independientes. Franz desarrolló un modelo que tiene en cuenta este tipo de dependencias [Franz 1996, Franz 1997].

Actualmente, 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 [Samuelsson 1993]. 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.

Como se puede ver, 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. Por ejemplo:

pero como máximo $m = 10$ caracteres.

- Brants utiliza también una elección independiente del contexto para los pesos suavización de sufijos θ_i, al igual que hacía con los parámetros de interpolación lin de trigramas λ_i. Es decir, en lugar de calcular los θ_i tal y como propuso Samuelss a partir de la desviación estándar de las probabilidades de máxima verosimilitud los sufijos, Brants asigna a todos los θ_i la desviación estándar de las probabilidades máxima verosimilitud no condicionadas de las etiquetas en el corpus de entrenamient

$$\theta_i = \frac{1}{s-1}\sum_{j=1}^{s}(\hat{p}(t^j) - \bar{P})^2$$

para todo $i = m, \ldots, 1$, donde s es el cardinal del conjunto de etiquetas utilizado, la media $\bar{P}$ se calcula como

$$\bar{P} = \frac{1}{s}\sum_{j=1}^{s}\hat{p}(t^j).$$

- Otro grado de libertad del método es el que concierne a la elección de las palabras c corpus de entrenamiento que se utilizan para la extracción de los sufijos. ¿Deberíam utilizar todas las palabras, o algunas son más adecuadas que otras? Dado que l palabras desconocidas son probablemente las más infrecuentes, se puede argument que el uso de los sufijos de las palabras menos frecuentes del corpus constituye u aproximación mejor para las palabras desconocidas que los sufijos de las palabr frecuentes. Brants, por tanto, realiza la extracción de sufijos sólo sobre aquell palabras cuya frecuencia sea menor o igual que un cierto umbral de corte, y fija dic umbral empíricamente en 10 apariciones. Además, mantiene dos pruebas de sufij separadas, en función de si la inicial de la palabra es mayúscula o minúscula.

En general, para las palabras desconocidas, $P(w|t) = 0$ estrictamente hablando. En ca contrario, la palabra no sería desconocida. Pero tal y como hemos dicho, una de las principal habilidades que debe incluir un buen etiquetador es asignar una probabilidad $P(w|t)$ distinta cero para las palabras no presentes en el diccionario. La única restricción que hay que respet es que, para toda etiqueta t del conjunto de etiquetas, debería cumplirse

$$\sum_{w} P(w|t) = 1, \tag{3.1}$$

sobre todas las palabras w: las conocidas y las, en teoría, posiblemente desconocidas. És es la clave del problema, porque implica que deberíamos conocer *a priori* todas las palabr desconocidas. En la práctica, esto no es así, y por tanto no conocemos completamente el conjun de las probabilidades que estamos sumando. Una forma de afrontar el problema es considerar q no todas las palabras desconocidas juegan el mismo papel. Algunas de ellas serán efectivamen palabras nuevas para las cuales queremos probabilidades de emisión distintas de cero, pero otr podrían ser palabras con errores ortográficos para las cuales queremos probabilidades igual cero o, en todo caso, las probabilidades de las correspondientes palabras corregidas.

Así pues, si el conjunto de palabras que van a ser aceptadas como desconocidas es *de algu*

ara ello, se podrían combinar las probabilidades de los sufijos con las probabilidades de
ón de las palabras conocidas, reservando una parte de la masa de probabilidad de éstas
as. Es decir, si conocida la etiqueta t y para toda palabra w del diccionario,

$$S = \sum_{w} P(w|t),$$

$S < 1$, entonces $1 - S$ sería la masa de probabilidad dedicada a las palabras desconocidas,
ida t. $P(\textit{sufijo}|t)$ debería ser una distribución de probabilidad, de tal forma que

$$\sum_{\textit{sufijo}} P(\textit{sufijo}|t) = 1, \tag{3.17}$$

toda etiqueta t del conjunto de etiquetas utilizado. Y los sufijos deberían estar
tamente definidos, de tal manera que la función $\textit{sufijo}(w)$ fuera una aplicación. Es decir,
jo de w debería ser único para cada palabra w. De esta forma, $P(w|t)$ se define para las
ras desconocidas como

$$P(w|t) = (1 - S)\, P(\textit{sufijo}(w)|t),$$

l no sólo es una definición formal, sino también sensible a las propiedades morfológicas de
palabra. Las dificultades, por supuesto, radican en la identificación de los sufijos válidos
as palabras y en la estimación de un buen valor para S.
l y como vimos antes, la implementación que hace Brants del método de Samuelsson no
a las probabilidades de los sufijos con las probabilidades de las palabras conocidas, y por
no respeta la restricción (3.16). Sin embargo, dicho método construye un conjunto de
bilidades de emisión distinto para cada longitud de sufijo, y cada uno de esos conjuntos
a la restricción (3.17). Como se puede apreciar, el enfoque es totalmente distinto, pero el
lo también genera probabilidades directamente utilizables por los modelos de Markov.
n definitiva, el tratamiento de palabras desconocidas[16] representa el punto más débil de
icación de los HMM,s al proceso de etiquetación del lenguaje natural, y no se incluye
almente en el estado del arte de los HMM,s. En esta sección hemos esbozado sólo algunos
métodos de tratamiento de palabras desconocidas que han sido desarrollados e integrados
lmente dentro del marco de los HMM,s.

Variantes de implementación de los HMM,s

redicciones condicionadas sobre una historia o contexto de gran longitud no son siempre una
idea. Por ejemplo, normalmente no existen dependencias sintácticas entre las palabras
parecen antes y después de las comas. Por tanto, el conocimiento de la etiqueta que aparece
de una coma no siempre ayuda a determinar correctamente la que aparece después. De
, ante este tipo de casos, un etiquetador basado en trigramas podría hacer predicciones
s que otro basado en bigramas, debido una vez más al fenómeno de los datos dispersos[17].
cnica de interpolación lineal de las probabilidades de los unigramas, bigramas y trigramas

mbién denominado *word guessing*.
uchas veces, las probabilidades de transición de los trigramas se estiman en base a sucesos de rara aparición,

de conocimiento lingüístico previo. Por ejemplo, Kupiec observó que un HMM de orden
se equivocaba sistemáticamente al etiquetar la secuencia de palabras `the bottom of` con
artículo-adjetivo-preposición. Entonces extendió el modelo con una red especial para es
construcción, con el propósito de que el etiquetador pudiera aprender la imposibilidad de q
una preposición siga a la secuencia artículo-adjetivo [Kupiec 1992]. En general, este méto
selecciona manualmente determinados estados del modelo y aumenta su orden, para casos don
la *memoria* de orden 1 no es suficiente.

Otro método relacionado es el de los Modelos de Markov de memoria variable
VMMM,s[18] [Schütze y Singer 1994, Triviño y Morales 2000]. Los VMMM,s presentan estad
de diferentes *longitudes*, en lugar de los estados de *longitud fija* de los etiquetadores basad
en bigramas o en trigramas. Un VMMM podría transitar desde un estado que recuerda l
dos últimas etiquetas (correspondiente, por tanto, a un trigrama) a un estado que recuerda l
tres últimas etiquetas (correspondiente a un cuatrigrama) y después a otro estado sin memor
(correspondiente a un unigrama). El número de símbolos a recordar para una determina
secuencia se determina durante el proceso de entrenamiento, en base a criterios teóricos. E
contraste con la interpolación lineal, los VMMM,s condicionan la longitud de memoria utiliza
durante la predicción a la secuencia actual, en lugar de considerar una suma ponderada fija pa
todas las secuencias. Los VMMM,s se construyen con estrategias descendentes mediante métod
de división de estados[19]. Una posible alternativa es construirlos con estrategias ascendent
mediante métodos de fusión de modelos[20] [Stolcke y Omohundro 1994, Brants 1998].

Otra aproximación que resulta todavía más potente es la constituida por los model
de Markov jerárquicos no emisores [Ristad y Thomas 1997]. Mediante la introducción
transiciones sin emisiones[21], estos modelos pueden almacenar también dependencias de longit
arbitraria entre los estados.

3.6 Otras aplicaciones de los HMM,s

La teoría matemática relativa a los HMM,s fue desarrollada por Baum y sus colaboradores
finales de los años sesenta y principios de los setenta [Baum *et al.* 1970]. Los HMM,s se aplicar
al procesamiento de la voz o reconocimiento del discurso hablado[22] en los años setenta p
Baker [Baker 1975], y por Jelinek y sus colegas de IBM [Jelinek *et al.* 1975, Jelinek 1976]. F
posteriormente cuando los HMM,s se utilizaron para modelizar otros aspectos de los lengua
humanos, tales como el proceso de etiquetación.

Dentro del contexto del reconocimiento de voz, existen muy buenas referencias sobre l
HMM,s y sus algoritmos [Levinson *et al.* 1983, Charniak 1993, Jelinek 1997]. Particularmen
conocidas son también [Rabiner 1989, Rabiner y Juang 1993]. Todas ellas estudian en deta
tanto los HMM,s continuos, donde la salida es un valor real, como los HMM,s discret
que nosotros hemos considerado aquí. Aunque se centran en la aplicación de los HMM,s
reconocimiento de voz, resulta también muy recomendable su consulta para la obtención

[18] *Variable Memory Markov Models.*
[19] *State splitting.*
[20] *Model merging.*
[21] Transiciones entre estados que no emiten ninguna palabra o que, de manera equivalente, emiten la palab
vacía ϵ.

nuestros HMM,s cuando nos enfrentamos a un nuevo problema? Normalmente, como en o de la etiquetación, es la naturaleza del problema la que determina de alguna manera la ectura del modelo. Para circunstancias en las que no es ese el caso, existen trabajos que an la construcción automática de la estructura de los HMM,s. Dichos trabajos se basan en ncipio de intentar encontrar el HMM más compacto posible que describa adecuadamente tos [Stolcke y Omohundro 1993].

s HMM,s han sido también ampliamente utilizados en aplicaciones bioinformáticas para el is de secuencias de genes [Baldi y Brunak 1998, Durbin *et al.* 1998]. Es cierto que puede er increíble que el manejo de un alfabeto de sólo cuatro símbolos, las bases nitrogenadas DN, entrañe grandes dificultades, pero la bioinformática es un dominio perfectamente ecido, y en el que efectivamente se plantean problemas muy serios que pueden requerir el e modelizaciones complejas.

quizás la aplicación más reciente de los HMM,s dentro del marco del procesamiento de aje natural es su uso directo como motor de búsqueda en sistemas de recuperación de nación [Miller *et al.* 1999]. La idea básica de esta aproximación consiste una vez más en lerar la unión de todas las palabras que aparecen en el corpus como el conjunto de símbolos ida del modelo, e identificar un estado distinto, en este caso no para cada etiqueta, sino cada uno de los posibles mecanismos de generación de palabras en las consultas: términos icos, términos dependientes del documento, palabras que aparecen frecuentemente en nsultas, mecanismos de sinonimia, etc. Posteriormente, se construye un sencillo HMM dual para cada documento, a través del cual se puede calcular la probabilidad de que documento genere las palabras involucradas en la consulta que efectúa el usuario. Esa bilidad indica si el documento es relevante o no, y en función de ella se establece la lista ada de documentos que el sistema muestra al usuario como respuesta a su consulta. Los ados obtenidos con esta técnica son muy prometedores, y refuerzan la perspectiva de que MM,s continúan siendo aplicables incluso en temas de verdadera actualidad como es el de uperación de información.

Capítulo 4

Aprendizaje de etiquetas basado en transformaciones

Tradicionalmente, en lo que se refiere a la etiquetación de textos en lenguaje natural, y tal y cor hemos visto en el capítulo anterior, se han preferido las aproximaciones puramente estocástic frente a las aproximaciones basadas en reglas, debido a su buen rendimiento y sobre todo sus facilidades de entrenamiento automático. Sin embargo, hemos visto también que algun de las hipótesis de funcionamiento de los modelos de Markov no se adaptan del todo bien a l propiedades sintácticas de los lenguajes naturales. Debido a esto, inmediatamente surge la id de utilizar modelos más sofisticados. Podríamos pensar por ejemplo en establecer condicion que relacionen las nuevas etiquetas, no sólo con las etiquetas precedentes, sino también c las palabras precedentes. Podríamos pensar también en utilizar un contexto mayor que el los etiquetadores basados en trigramas. Pero la mayoría de estas aproximaciones no tien cabida dentro de los modelos de Markov, debido a la carga computacional que implican y a gran cantidad de nuevos parámetros que necesitaríamos estimar. Incluso con los etiquetador basados en trigramas hemos visto que es necesario aplicar técnicas de suavización e interpolació ya que la estimación de máxima verosimilitud por sí sola no es lo suficientemente robusta.

Eric Brill presentó un sistema de etiquetación basado en reglas, el cual, a partir de un corp de entrenamiento, infiere automáticamente las reglas de transformación [Brill 1993b], salvan así la principal limitación de este tipo de técnica que es precisamente el problema de cón obtener dichas reglas. El etiquetador de Brill alcanza un rendimiento comparable al de l etiquetadores estocásticos y, a diferencia de éstos, la información lingüística no se captura manera indirecta a través de grandes tablas de probabilidades, sino que se codifica directamen bajo la forma de un pequeño conjunto de reglas no estocásticas muy simples, pero capaces representar interdependencias muy complejas entre palabras y etiquetas. Este capítulo descri las características principales del etiquetador de Brill y de su principio de funcionamiento: paradigma de aprendizaje basado en transformaciones y dirigido por el error[1]. Veremos que es método es capaz de explorar un abanico mayor de propiedades, tanto léxicas como sintáctica de los lenguajes naturales. En particular, se pueden relacionar etiquetas con palabras concreta se puede ampliar el contexto precedente, e incluso se puede utilizar el contexto posterior.

4.1 Arquitectura interna del etiquetador de Brill

El etiquetador de Brill consta de tres partes, que se infieren automáticamente a partir de corpus de entrenamiento: un etiquetador léxico, un etiquetador de palabras desconocidas, y

quetador léxico etiqueta inicialmente cada palabra con su etiqueta más probable, sin tener enta el contexto en el que dicha palabra aparece. Dicha etiqueta más probable se estima amente mediante el estudio del corpus de entrenamiento. A las palabras desconocidas se gna en un primer momento la etiqueta correspondiente a sustantivo propio si la primera es mayúscula, o la correspondiente a sustantivo común en otro caso. Posteriormente, el tador de palabras desconocidas aplica en orden una serie de reglas de transformación s.

se dispone de un diccionario previamente construido, es posible utilizarlo junto con el etiquetador de Brill genera automáticamente. Más adelante veremos cómo se realiza esta ación y en qué circunstancias concretas este proceso ayuda a mejorar el rendimiento del tador.

El etiquetador de palabras desconocidas

quetador de palabras desconocidas opera justo después de que el etiquetador léxico haya tado todas las palabras presentes en el diccionario, y justo antes de que se apliquen las contextuales. Este módulo intenta *adivinar* una etiqueta para una palabra desconocida ación de su sufijo[2], de su prefijo, y de otras propiedades relevantes similares.

ásicamente, cada transformación consta de dos partes: una descripción del contexto de ción, y una regla de reescritura que reemplaza una etiqueta por otra. La plantilla genérica nsformaciones léxicas es la siguiente:

- `x haspref l A`: si los primeros `l` caracteres de la palabra son `x`, se asigna a la palabra desconocida la etiqueta `A`
- `A x fhaspref l B`: si la etiqueta actual de la palabra es `A` y sus primeros `l` caracteres son `x`, se cambia dicha etiqueta por `B`
- `x deletepref l A`: si borrando el prefijo `x` de longitud `l` obtenemos una palabra conocida, se asigna a la palabra desconocida la etiqueta `A`
- `A x fdeletepref l B`: si la etiqueta actual de la palabra es `A` y borrando el prefijo `x` de longitud `l` obtenemos una palabra conocida, se cambia dicha etiqueta por `B`
- `x addpref l A`: si añadiendo el prefijo `x` de longitud `l` obtenemos una palabra conocida, se asigna a la palabra desconocida la etiqueta `A`
- `A x faddpref l B`: si la etiqueta actual de la palabra es `A` y añadiendo el prefijo `x` de longitud `l` obtenemos una palabra conocida, se cambia dicha etiqueta por `B`
- `x hassuf l A`: si los últimos `l` caracteres de la palabra son `x`, se asigna a la palabra desconocida la etiqueta `A`
- `A x fhassuf l B`: si la etiqueta actual de la palabra es `A` y sus últimos `l` caracteres son `x`, se cambia dicha etiqueta por `B`
- `x deletesuf l A`: si borrando el sufijo `x` de longitud `l` obtenemos una palabra conocida, se asigna a la palabra desconocida la etiqueta `A`
- `A x fdeletesuf l B`: si la etiqueta actual de la palabra es `A` y borrando el sufijo `x` de longitud `l` obtenemos una palabra conocida, se cambia dicha etiqueta por `B`

longitud 1 obtenemos una palabra conocida, se cambia dicha etiqueta por B

`w goodright A`: si la palabra aparece inmediatamente a la derecha de la palabra `w`, se asigna a la palabra desconocida la etiqueta `A`

`A w fgoodright B`: si la etiqueta actual de la palabra es `A` y aparece inmediatamente a la derecha de la palabra `w`, se cambia dicha etiqueta por `B`

`w goodleft A`: si la palabra aparece inmediatamente a la izquierda de la palabra `w`, se asigna a la palabra desconocida la etiqueta `A`

`A w fgoodleft B`: si la etiqueta actual de la palabra es `A` y aparece inmediatamente a la izquierda de la palabra `w`, se cambia dicha etiqueta por `B`

`z char A`: si el caracter `z` aparece en la palabra, se asigna a la palabra desconocida la etiqueta `A`

`A z fchar B`: si la etiqueta actual de la palabra es `A` y el caracter `z` aparece en la palabra, se cambia dicha etiqueta por `B`

donde `A` y `B` son variables sobre el conjunto de todas las etiquetas, `x` es cualquier cadena caracteres de longitud 1, 2, 3 o 4, `l` es la longitud de dicha cadena, `w` es cualquier palabra, y es cualquier caracter.

Ejemplo 4.1 A continuación se muestran algunas de las reglas de transformación léxicas m comunes que el etiquetador de Brill encontró para el español:

`rse hassuf 3 V000f0PE1`: si los últimos `3` caracteres de la palabra son `rse`, se asigna a la palabra desconocida la etiqueta `V000f0PE1`, es decir, verbo infinitivo con un pronombre enclítico

`r hassuf 1 V000f0`: si el último caracter de la palabra es `r`, se asigna a la palabra desconocida la etiqueta `V000f0`, es decir, verbo infinitivo

`V000f0 or fhassuf 2 Scms`: si la etiqueta actual de la palabra es `V000f0`, es decir, verbo infinitivo, y sus últimos `2` caracteres son `or`, se cambia dicha etiqueta por la etiqueta `Scms`, es decir, sustantivo común, masculino, singular

`ría deletesuf 3 Vysci0`: si borrando el sufijo `ría` de longitud `3` obtenemos una palabra conocida, se asigna a la palabra desconocida la etiqueta `Vysci0`, es decir, verbo, primera y tercera personas del singular, postpretérito de indicativo

`Scfs r faddsuf 1 V3spi0`: si la etiqueta actual de la palabra es `Scfs`, es decir, sustantivo, común, femenino, singular, y añadiendo el sufijo `r` de longitud `1` obtenemos una palabra conocida, se cambia dicha etiqueta por la etiqueta `V3spi0`, es decir, verbo, tercera persona del singular, presente de indicativo

`el goodright Scms`: si la palabra aparece inmediatamente a la derecha de la palabra `el`, se asigna a la palabra desconocida la etiqueta `Scms`, es decir, sustantivo común, masculino, singular

`Scmp las fgoodright Scfp`: si la etiqueta actual de la palabra es `Scmp`, es decir, sustantivo común, masculino, plural, y aparece inmediatamente a la derecha de la palabra `las`, se cambia dicha etiqueta por la etiqueta `Scfp`, es decir, sustantivo

`w char Ze00`: si el carácter `w` aparece en la palabra, se asigna a la palabra desconocida la etiqueta `Ze00`, es decir, palabra extranjera

eglas fueron generadas por el etiquetador de Brill después de ser entrenado con una porción tto en español procedente del corpus ITU[3]. □

r debajo del formato general de las reglas léxicas propuestas por Brill subyace un estudio stico muy importante. Es por ello que esta plantilla genérica de transformaciones léxicas enta una manera muy elegante de integrar el manejo de palabras desconocidas dentro de erramienta general de etiquetación, y la hace independiente del idioma a tratar. Pero derando el caso concreto del español, se echan de menos fenómenos muy comunes tales el sufijo `mente` para formar adverbios a partir de los adjetivos. En el corpus ITU existe en número de palabras que presentan dicha característica, pero ésta nunca podrá aparecer da en una regla debido a la limitación de 4 caracteres de longitud en los prefijos y sufijos reglas extraídas automáticamente. No obstante, el etiquetador de Brill proporciona al o la posibilidad de añadir manualmente nuevas reglas después del entrenamiento.
tras características relevantes para el tratamiento de palabras desconocidas en modelos de tación basados en reglas fueron estudiadas por Mikheev, quien no sólo amplía el formalismo glas léxicas del etiquetador de Brill, sino que también propone su propio algoritmo para la ción automática de dichas reglas [Mikheev 1997].

El etiquetador contextual

iquetador contextual actúa justo después del etiquetador de palabras desconocidas, ndo en orden una secuencia de reglas contextuales que, al igual que las léxicas, también do previamente inferidas de manera automática a partir del corpus de entrenamiento. La lla genérica de transformaciones contextuales es la siguiente:

- `A B prevtag C`: cambiar la etiqueta `A` por `B` si la palabra anterior aparece etiquetada con la etiqueta `C`
- `A B prev1or2tag C`: cambiar la etiqueta `A` por `B` si una de las dos palabras anteriores aparece etiquetada con la etiqueta `C`
- `A B prev1or2or3tag C`: cambiar la etiqueta `A` por `B` si una de las tres palabras anteriores aparece etiquetada con la etiqueta `C`
- `A B prev2tag C`: cambiar la etiqueta `A` por `B` si la segunda palabra anterior aparece etiquetada con la etiqueta `C`
- `A B nexttag C`: cambiar la etiqueta `A` por `B` si la palabra siguiente aparece etiquetada con la etiqueta `C`
- `A B next1or2tag C`: cambiar la etiqueta `A` por `B` si una de las dos palabras siguientes aparece etiquetada con la etiqueta `C`
- `A B next1or2or3tag C`: cambiar la etiqueta `A` por `B` si una de las tres palabras siguientes aparece etiquetada con la etiqueta `C`
- `A B next2tag C`: cambiar la etiqueta `A` por `B` si la segunda palabra siguiente aparece etiquetada con la etiqueta `C`

etiquetada con la etiqueta `C` y la segunda palabra siguiente con `D`

`A B surroundtag C D`: cambiar la etiqueta `A` por `B` si la palabra anterior aparece etiquetada con la etiqueta `C` y la siguiente con `D`

`A B curwd w`: cambiar la etiqueta `A` por `B` si la palabra actual es `w`

`A B prevwd w`: cambiar la etiqueta `A` por `B` si la palabra anterior es `w`

`A B prev1or2wd w`: cambiar la etiqueta `A` por `B` si una de las dos palabra anteriores es `w`

`A B prev2wd w`: cambiar la etiqueta `A` por `B` si la segunda palabra anterior es `w`

`A B nextwd w`: cambiar la etiqueta `A` por `B` si la palabra siguiente es `w`

`A B next1or2wd w`: cambiar la etiqueta `A` por la etiqueta `B` si una de las dos palabras siguientes es `w`

`A B next2wd w`: cambiar la etiqueta `A` por `B` si la segunda palabra siguiente es `w`

`A B lbigram w x`: cambiar la etiqueta `A` por `B` si las dos palabras anteriores son `w` y `x`

`A B rbigram w x`: cambiar la etiqueta `A` por `B` si las dos palabras siguientes son `w` y `x`

`A B wdand2bfr x w`: cambiar la etiqueta `A` por `B` si la palabra actual es `w` y la segunda palabra anterior es `x`

`A B wdand2aft w x`: cambiar la etiqueta `A` por `B` si la palabra actual es `w` y la segunda palabra siguiente es `x`

`A B wdprevtag C w`: cambiar la etiqueta `A` por `B` si la palabra actual es `w` y la anterior aparece etiquetada con la etiqueta `C`

`A B wdnexttag w C`: cambiar la etiqueta `A` por `B` si la palabra actual es `w` y la siguiente aparece etiquetada con la etiqueta `C`

`A B wdand2tagbfr C w`: cambiar la etiqueta `A` por `B` si la palabra actual es `w` y la segunda palabra anterior aparece etiquetada con la etiqueta `C`

`A B wdand2tagaft w C`: cambiar la etiqueta `A` por `B` si la palabra actual es `w` y la segunda palabra siguiente aparece etiquetada con la etiqueta `C`

donde `A`, `B`, `C` y `D` son variables sobre el conjunto de todas las etiquetas, y `w` y `x` son cualqui palabra. La figura 4.1 resume gráficamente los posibles contextos que se pueden tener en cuen a la hora de aplicar una transformación. La palabra a etiquetar es siempre la de la posición que aparece marcada con un asterisco. Los recuadros indican cuáles son las posiciones relevant de cada esquema contextual. Si en una posición dada aparece un recuadro normal, el esquem considera la etiqueta de la palabra que está en esa posición. Si aparece un recuadro sombread el esquema considera la palabra concreta. Por ejemplo, la regla `A B prevbigram C D` respon al esquema 3, mientras que la regla `A B lbigram w x` utiliza el esquema 15.

Ejemplo 4.2 A continuación se muestran algunas de las reglas de transformación contextual más comunes que el etiquetador de Brill encontró para el español, también a partir de u

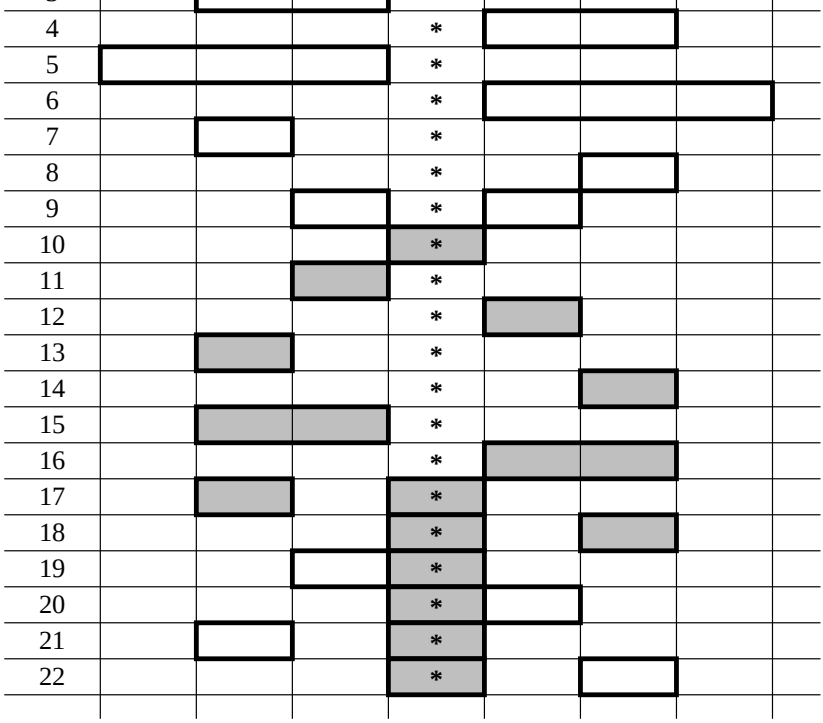


Figura 4.1: Esquemas contextuales de las reglas del etiquetador de Brill

`Afp0 Amp0 prevtag Scmp`: cambiar la etiqueta `Afp0`, es decir, adjetivo, femenino, plural, sin grado, por `Amp0`, es decir, adjetivo, masculino, plural, sin grado, si la palabra anterior aparece etiquetada con la etiqueta `Scmp`, es decir, sustantivo común, masculino, plural

`Scms Ams0 wdprevtag Scms receptor`: cambiar la etiqueta `Scms`, es decir, sustantivo común, masculino, singular, por `Ams0`, es decir, adjetivo, masculino, singular, sin grado, si la palabra actual es `receptor` y la anterior aparece etiquetada con la etiqueta `Scms`

`Scms Ams0 wdand2bfr el transmisor`: cambiar la etiqueta `Scms`, es decir, sustantivo común, masculino, singular, por `Ams0`, es decir, adjetivo, masculino, singular, sin grado, si la palabra actual es `transmisor` y la segunda palabra anterior es `el`

`P Scms nexttag P`: cambiar la etiqueta `P`, es decir, preposición, por `Scms`, es decir, sustantivo común, masculino, singular, si la palabra siguiente aparece etiquetada con la etiqueta `P`

vés de estos ejemplos, se puede apreciar que el formalismo de reglas contextuales del etador de Brill constituye un sencillo pero potente mecanismo, que es capaz de resolver ncordancias de género y número dentro de una misma categoría, las ambigüedades entre tas categorías, como por ejemplo, adjetivo/sustantivo, y hasta puede evitar que aparezcan

El proceso de generación de las reglas, tanto las léxicas en el caso del etiquetador de palabr
desconocidas, como las contextuales en el caso del etiquetador contextual, selecciona el mej
conjunto de transformaciones y determina su orden de aplicación. El algoritmo consta de l
pasos que se describen a continuación. En primer lugar, se toma una porción de texto
etiquetado, se pasa a través de la fase o fases de etiquetación anteriores, se compara la sali
con el texto correctamente etiquetado, y se genera una lista de errores de etiquetación con s
correspondientes contadores. Entonces, para cada error, se determina qué instancia concre
de la plantilla genérica de reglas produce la mayor reducción de errores. Se aplica la regla,
calcula el nuevo conjunto de errores producidos, y se repite el proceso hasta que la reducción
errores cae por debajo de un umbral dado.

La figura 4.2 ilustra gráficamente este procedimiento, que es el que da nombre a la técni
de entrenamiento desarrollada por Brill: aprendizaje basado en transformaciones y dirigido p
el error. El usuario puede especificar los umbrales de error antes del entrenamiento, y pue
también añadir manualmente nuevas reglas de transformación después del mismo.

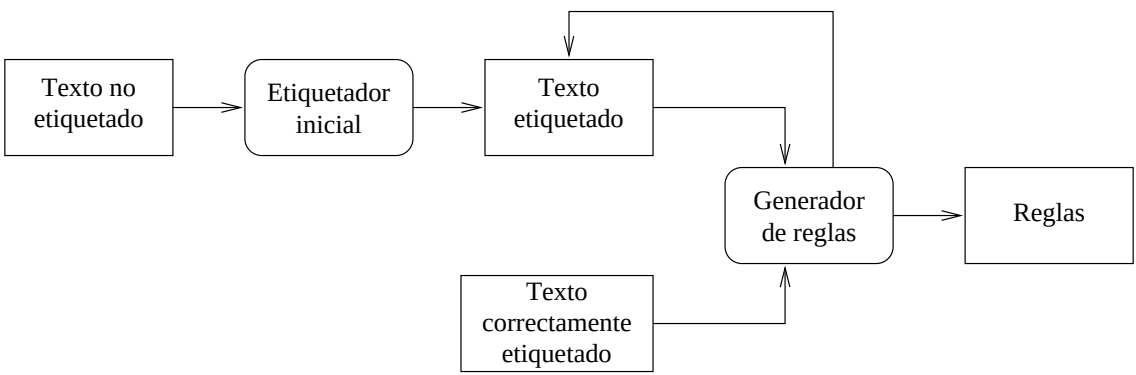


Figura 4.2: Aprendizaje basado en transformaciones y dirigido por el error

A las características anteriormente descritas podríamos añadir también una funcionalid
introducida posteriormente por el propio Brill, la cual permite obtener las k etiquetas m
probables de una palabra[4] [Brill 1994], para ciertas aplicaciones en las que es posible relajar
restricción de una sola etiqueta por palabra. Brill implementa esta nueva funcionalidad mediar
un sencillo cambio en el formato de las reglas:

> la acción *cambiar la etiqueta* `A` *por la etiqueta* `B` se transforma en *añadir la etiqueta* `A` *a la etiqueta* `B` o *añadir la etiqueta* `A` *a la palabra* `w`.

De esta manera, en lugar de reemplazar etiquetas, las reglas de transformación permiten aho
añadir etiquetas alternativas a una palabra. Sin embargo, el problema es que el etiquetador
nos proporciona información sobre la probabilidad de cada etiqueta. Es decir, si consideram
por ejemplo las dos mejores etiquetas para una palabra dada, la única conclusión que podem
extraer es que la que aparece en primer lugar es más probable que la que aparece en segun
lugar, pero bien podría ocurrir tanto que la primera fuera 100 veces más probable que la segund
como que ambas fueran igualmente probables. La cuestión es que este tipo de información podr
ser crucial para algunas aplicaciones, por ejemplo para la construcción de un análisis sintáctic
Los etiquetadores puramente estocásticos sí son capaces de proporcionar estas cifras y adem
lo hacen sin ningún esfuerzo computacional extra.

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

plo 4.3 Si suponemos que VBN y VBD son las etiquetas más probables para las palabras d y shot, respectivamente, el etiquetador léxico podría asignar las siguientes etiquetas[5]:

```
) Chapman/NP killed/VBN John/NP Lennon/NP
) John/NP Lennon/NP was/BEDZ shot/VBD by/BY Chapman/NP
) He/PPS witnessed/VBD Lennon/NP killed/VBN by/BY Chapman/NP
```

que el etiquetador léxico no utiliza ninguna información contextual, muchas palabras n aparecer etiquetadas incorrectamente. Por ejemplo, en (1) la palabra killed ce etiquetada incorrectamente como verbo en participio pasado, y en (2) shot aparece ectamente etiquetada como verbo en tiempo pasado.

na vez obtenida la etiquetación inicial, el etiquetador contextual aplica en orden una ncia de reglas e intenta remediar los errores cometidos. En un etiquetador contextual amos encontrar reglas como las siguientes:

```
N VBD prevtag NP
D VBN nexttag BY
```

mera regla dice: *cambiar la etiqueta* VBN *por* VBD *si la etiqueta previa es* NP. La segunda dice: *cambiar* VBD *por* VBN *si la siguiente etiqueta es* BY. Una vez que aplicamos la primera la palabra killed que aparece en las frases (1) y (3) cambia su etiqueta VBN por VBD, y emos las siguientes etiquetaciones:

```
) Chapman/NP killed/VBD John/NP Lennon/NP
) John/NP Lennon/NP was/BEDZ shot/VBD by/BY Chapman/NP
) He/PPS witnessed/VBD Lennon/NP killed/VBD by/BY Chapman/NP
```

ez que aplicamos la segunda regla, la palabra shot de la frase (5) cambia su etiqueta VBD BN, generando la etiquetación (8), y la palabra killed de la frase (6) vuelve a cambiar su ta VBD otra vez por VBN, y obtenemos la etiquetación (9):

```
) Chapman/NP killed/VBD John/NP Lennon/NP
) John/NP Lennon/NP was/BEDZ shot/VBN by/BY Chapman/NP
) He/PPS witnessed/VBD Lennon/NP killed/VBN by/BY Chapman/NP
```

s visto que una regla nextag necesita mirar un *token* hacia adelante en la frase antes de ser aplicada, y hemos visto también que la aplicación de dos o más reglas puede producir erie de operaciones que no se traducen en ningún cambio neto. Estos dos fenómenos tuyen la causa del no determinismo local del etiquetador de Brill. □

te problema fue abordado por Roche y Schabes, quienes propusieron un sistema que ca las reglas de transformación del etiquetador bajo la forma de un traductor de estado determinista [Roche y Schabes 1995]. El algoritmo de construcción de dicho traductor a de cuatro pasos:

---

notación de las etiquetas es una adaptación del juego de etiquetas utilizado en el corpus BROWN s y Kučera 1982]: VBN significa verbo en participio pasado, VBD es verbo en tiempo pasado, NP es sustantivo

a través de t_2 en un solo paso produce el mismo efecto que procesar cada posición
la cadena de entrada a través de t_1. Este paso se ocupa de casos como el siguien
Supongamos un traductor que implementa la transformación *cambiar* `A` *por* `B` *si una*
las dos etiquetas precedentes es `C`. Este traductor contendrá un arco que transforma
símbolo de entrada `A` en el símbolo de salida `B`, de tal manera que dada la secuencia
entrada `CAA` tenemos que aplicarlo dos veces, en la segunda y en la tercera posicion
para transformarla correctamente en `CBB`. La extensión local es capaz de realizar dic
conversión en un solo paso.

3. En el tercer paso, se genera un único traductor cuya aplicación tiene el mismo efecto q
la aplicación de todos los traductores individuales en secuencia. En general, este traduct
único es no determinista. Cuando necesita recordar un evento tal como `C` *apareció en*
posición i, lo hace lanzando dos caminos distintos: uno en el cual se supone que aparece
una etiqueta que estará afectada por esa `C` precedente, y otro en el que se supone que t
etiqueta no aparecerá.

4. Este tipo de indeterminismo no es eficiente, de ahí que el cuarto paso se ocupe
transformar el traductor no determinista en uno determinista. Esto en general no
posible, ya que los traductores no deterministas pueden recordar eventos de longit
arbitraria y los deterministas no. Sin embargo, Roche y Schabes demuestran que las regl
que aparecen en los etiquetadores basados en transformaciones no generan traductores c
esta propiedad. Por tanto, en la práctica siempre es posible transformar un etiquetad
basado en transformaciones en un traductor de estado finito determinista.

Por tanto, el algoritmo de Brill podría necesitar RKn pasos elementales para etiquetar u
cadena de entrada de n palabras, con R reglas aplicables en un contexto de hasta K *toke*
Con el traductor de estado finito propuesto por Roche y Schabes, para etiquetar una frase
longitud n palabras, se necesitan sólo n pasos, independientemente del número de reglas y de
longitud del contexto que éstas utilizan. Esto significa que el proceso de etiquetación añade a
lectura del texto de entrada una carga computacional que es despreciable en comparación c
tratamientos posteriores tales como los análisis sintáctico y semántico. Con los etiquetador
basados en traductores se pueden llegar a obtener velocidades de etiquetación de varias decen
de miles de palabras por segundo, mientras que con los etiquetadores basados en modelos
Markov esa velocidad puede ser de un orden de magnitud menos. Existen trabajos que estudi
la transformación de modelos de Markov ocultos en traductores de estado finito [Kempe 199
pero en este caso no se puede alcanzar una equivalencia completa ya que los autómatas
pueden simular de una manera exacta los cálculos de punto flotante involucrados en el algoritm
de Viterbi.

4.4 Relación con otros modelos de etiquetación

Se han esbozado ya algunas de las diferencias conceptuales más importantes que existen ent
el etiquetador de Brill y los etiquetadores puramente estocásticos. Esta sección completa
poco más el estudio comparativo de los principios de funcionamiento de éstas aproximacion
y de otras relacionadas. Finalmente, se citan otros posibles campos de aplicación en los que

nismo que etiqueta todas las hojas dominadas por un nodo con la etiqueta de la clase
itaria de ese nodo. Posteriormente, a medida que descendemos por el árbol, reetiquetamos
jas de los nodos hijos, si es que difieren de la etiqueta del nodo padre, en función de las
estas a las cuestiones o decisiones que aparecen en cada nodo. Esta manera de ver los
s de decisión es la que muestra el parecido con el aprendizaje basado en transformaciones,
e ambos paradigmas realizan series de reetiquetados trabajando con subconjuntos de datos
vez más pequeños.

n principio, el aprendizaje basado en transformaciones es más potente que los árboles de
ón [Brill 1995a]. Es decir, existen tareas de clasificación que se pueden resolver con el
dizaje basado en transformaciones, pero no con los árboles de decisión. Sin embargo, no
uy claro si este tipo de potencia extra se utiliza o no en aplicaciones de procesamiento de
aje natural.

principal diferencia entre estos dos modelos es que los datos de entrenamiento se dividen
la nodo de un árbol de decisión, y que se aplica una secuencia de *transformaciones* distintas
cada nodo: la secuencia correspondiente a las decisiones del camino que va desde la raíz
ese nodo. Con el aprendizaje basado en transformaciones, cada transformación de la lista
nsformaciones *aprendidas* se aplica a todo el texto, generando una reescritura cuando el
xto de los datos encaja con el de la regla. Como resultado, si minimizamos en función de
ores de etiquetación cometidos, en lugar de considerar otro tipo de medidas indirectas más
nes en el caso de los árboles de decisión, tales como la entropía, entonces sería relativamente
lo alcanzar el 100% de precisión en cada nodo hoja. Sin embargo, el rendimiento sobre
nuevos sería muy pobre debido a que cada nodo hoja estaría formado por un conjunto de
edades totalmente arbitrarias, que aunque han sido extraídas de los datos de entrenamiento
completamente generales.

rpredentemente, el aprendizaje basado en transformaciones parece ser inmune a este
eno [Ramshaw y Marcus 1994]. Esto se puede explicar parcialmente por el hecho de que
renamiento siempre se realiza sobre todo el conjunto de datos. Pero el precio que hay que
para obtener este tipo de robustez es que el espacio de secuencias de transformaciones
ser grande. Una implementación *naive* del aprendizaje basado en transformaciones sería
nto ineficiente. No obstante, existen maneras inteligentes de realizar las búsquedas en ese
o [Brill 1995b].

Modelos probabilísticos en general

an ventaja de la etiquetación basada en transformaciones es que se pueden establecer
ones sobre un conjunto de propiedades más rico que en el caso de los modelos puramente
sticos. Por ejemplo, se puede utilizar simultáneamente información de los contextos
rdo y derecho, y las palabras concretas, no sólo sus etiquetas, pueden influir en la
tación de las palabras vecinas.

tro punto clave es que las reglas de transformación son más fáciles de entender y de
icar que las probabilidades de transición y de generación de palabras en los etiquetadores
bilísticos. Sin embargo, está claro también que es más difícil prever el efecto que puede
a tener la modificación de una regla dentro de una secuencia de aplicación, ya que el
ortamiento de cada regla depende de la ejecución de las reglas previas y pueden surgir
rosas y complejas interacciones entre ellas.

trata de un método no supervisado[7] y de aplicación completamente automatizada. Desde es
punto de vista, la única diferencia es que los cálculos con números se realizan sólo durante
proceso de aprendizaje, el cual además no presenta problemas de sobreentrenamiento, y una v
que el aprendizaje está hecho, el proceso de etiquetación resulta ser puramente simbólico y
por ello que se puede implementar en una estructura computacional muy eficiente.

Por último, dado que el rendimiento de los etiquetadores basados en transformaciones va
resultar muy similar al de los puramente estocásticos, la decisión final entre utilizar unos u otr
dependerá casi exclusivamente de en qué tipo de sistema va a estar integrado el etiquetador
para qué tipo de aplicaciones se va a utilizar.

Además de al proceso de etiquetación, el aprendizaje basado en transformaciones
ha aplicado también al análisis sintáctico [Brill 1993a, Brill 1993c], al problema de
ligadura de la frase preposicional [Brill y Resnik 1994] y a la eliminación de ambigüedad
semánticas [Dini *et al.* 1998].

[7] En el sentido de que lo que hay presente en los textos de entrenamiento son las etiquetas, pero lo que realmen

Parte III

Reglas contextuales y traductores de estado finito

Capítulo 5

Etiquetación utilizando reglas contextuales

Las técnicas para la etiquetación de textos vistas hasta ahora son las más usad tradicionalmente. Llegados a este punto nos disponemos a estudiar otro método. La raz básicamente es que con los métodos estudiados hasta ahora no se puede mejorar el núme de aciertos, dejando una cota en su precisión de un 96% aproximadamente. Estos resultad además, empeoran utilizando textos de distinto tipo a los usados en el entrenamiento. Ca destacar que la precisión es un factor muy importante, ya que el etiquetado será normalmen un paso de preprocesado, de manera que los errores aquí cometidos se arrastrarán en las fas posteriores.

Con el propósito de mejorar la precisión en el etiquetado se pasó a investigar otras técnic de etiquetación. Una de las ideas fue la de volver a las primeras soluciones, que consistían la utilización de reglas escritas por humanos. Este método no dio muy buenos resultados en momento, con lo cual había que replantearlo. La idea general en la que se basa el replanteamien consiste en la utilización de reglas de menor compromiso para evitar así errores en situacion dudosas. De este modo se obtuvieron una serie de métodos de alta precisión, con el inconvenien de que algunas palabras quedan ambiguas después del etiquetado. Esto es así porque no utilizan reglas de compromiso máximo. A pesar de esto, la mayoría de las palabras tienen u única etiqueta después de pasar por el proceso de etiquetación.

Este método presenta las siguientes ventajas:

- Un grado de aciertos muy alto en la etiquetación del texto. El resultado obtenido superior al conseguido por cualquier otro método. Los creadores del sistema EngC afirman llegar a un 99% de precisión e incluso superarlo en ocasiones [Voutilainen 199 Los creadores del sistema Rtag afirman rondar el 98% [Sánchez *et al.* 1999].

- La precisión, en principio, no depende del tipo de texto, ya que no hay un corpus entrenamiento como con los otros métodos. Al enfrentarse a texto totalmente nuevo (cuanto a temática, estilo, etc.) es sin duda la técnica que obtiene mejores resultados c diferencia.

- Es posible hacer reglas a mayores para un tipo de texto dado, de esta manera se pue aprovechar al máximo la información disponible.

- Se puede acudir a reglas heurísticas que, normalmente, mejoran la precisión

Al utilizar reglas obtenemos un nivel de expresividad alto, de esta manera a la hora de etiquetar una palabra podemos tener en cuenta factores que un etiquetador estadístico no puede utilizar.

Estas técnicas se adecuan muy bien para lenguajes con un tipo de ambigüedades distintas a las del inglés o el español. Oflazer y Tür afirman que este tipo de técnicas son las más adecuadas para lenguajes como el turco y el finlandés [Oflazer y Tür 1996].

n cuanto a las desventajas podemos citar las siguientes:

Aumentan los costes de tiempo ya que las reglas deben ser diseñadas por expertos humanos. Para evitar esto se están empezando a utilizar reglas inducidas automáticamente como ayuda a los expertos [Oflazer y Tür 1996, Samuelsson *et al.* 1996, Màrquez y Padró 1997, Màrquez y Rodríguez 1997], pero no de manera exclusiva, ya que así romperíamos la filosofía de aprovechar el conocimiento humano, y además obtendríamos un etiquetdor similar al de Brill, pero de menos precisión.

Como no se llega a un máximo nivel de compromiso, al final del etiquetado quedan ambigüedades, lo que significa que estos métodos no se pueden usar por si solos cuando se necesita una única etiqueta por palabra.

Estos métodos son los más novedosos, con lo cual los menos estudiados, lo que supone: una falta de estándares, heterogeneidad, etc.

os podemos encontrar varias aproximaciones posibles a la hora de buscar un formalismo ado que refleje esta idea. Una primera aproximación puede ser la de seleccionar y eliminar tas según se cumplan o no una serie de restricciones. El sistema EngCG [Voutilainen 1995] a en esta idea, y también es utilizada en muchos otros etiquetadores [Oflazer y Tür 1996]. segunda aproximación es la de usar reglas que den puntuaciones positivas o negativas distintas etiquetas de una palabra, eliminando, *a posteriori*, aquellas que finalmente de un umbral determinado. El sistema Rtag de la Real Academia Española de la a [Porta 1996, Sánchez *et al.* 1999] utiliza esta idea. También nos podemos encontrar oximaciones que resultan de la combinación de las dos anteriores o incluso otras ideas.

considera más adecuada la primera aproximación, y esto se debe principalmente a:

Se adapta mejor al paradigma de etiquetación, sobre todo a la idea de menor compromiso. Consiste en la eliminación de todas aquellas etiquetas que se consideran incorrectas, sin intervenir en las dudosas. Con la otra aproximación las etiquetas no consideradas se eliminan totalmente.

Es más predecible, de manera que hará mejor su trabajo aunque refinará un menor número de etiquetas. De esta manera el número de errores producidos será menor, factor importante teniendo en cuenta que el etiquetador de reglas formará parte de un sistema mayor. La segunda aproximación sería más adecuada en sistemas independientes, ya que se puede llegar a una etiqueta por palabra, pero perdería precisión.

Evita tener que asignar un umbral de corte y definir los pesos que se sumarán o restarán al valor de cada etiqueta.

las siguientes:

- Potencia y expresividad, para que así los lingüistas puedan considerar el mayor número casos posibles.
- Sintaxis sencilla, para facilitar su implementación y uso.
- Cumplimiento de los estándares en caso de existir, para facilitar su implantación.

La heterogeneidad de los lenguajes de reglas existentes es uno de los grandes problem actuales. No sólo no hay un estándar, sino que los lenguajes son muy distintos unos de otros

Se puede decir que básicamente lo que se usa son reglas del tipo: si se cumple una condici (como, por ejemplo, que la palabra posterior a la actual sea un verbo) se ejecuta una acci (como, por ejemplo, seleccionar la etiqueta sustantivo de la palabra actual). En las condicion de las reglas es donde nos vamos a encontrar con mucha variedad, tanto en los tipos de condici como en la sintaxis utilizada. Sin embargo, en lo que a las acciones se refiere no hay muc variedad, sino que suelen ser: seleccionar una etiqueta posible o borrar una etiqueta de l posibles.

Existen lenguajes sencillos pero poco potentes, como el definido por el departamen de *Computer Engineering and Information Science* de la Univerdidad de Bilkent Turka [Oflazer y Tür 1996]. El problema de este lenguaje es la poca expresividad.

El lenguaje utilizado en el sistema EngCG [Voutilainen 1995], es lo que más se parece un estándar. Sus creadores son los que más han avanzado en este campo y han definido lenguaje que bautizaron como *Constraint Grammars*. Sin embargo este formalismo presen algunos problemas importantes, entre las cuales destacan los siguientes:

- Este lenguaje pretende abordar más problemas que el del etiquetado morfológico. De hec une el etiquetado y el análisis sintáctico, cuando tradicionalmente estos dos problemas tratan en módulos distintos por la gran complejidad que presentan ya por separado. Es proyecto va a abordar una de las partes, la del etiquetado, y las razones de ello se citan continuación:
  - Las herramientas de PLN que han sido implementadas por el grupo COI (COmpiladores y LEnguajes) del Departamento de Computación de la Universid de A Coruña trabajan con módulos independientes, con lo cual hacer un módu conjunto rompería las planificaciones del equipo de trabajo y con mucha segurid acarrearía problemas de integración.
  - Nos podemos encontrar con problemas en los que sólo nos va a interesar trabajar c un etiquetador y no con un analizador sintáctico, como puede ser la lematización.
  - El hecho de abarcar más de un simple etiquetador de palabras hace q potencialmente se ralentice su ejecución, influyendo de forma negativa en los casos los que sólo se necesite etiquetar.
  - El análisis sintáctico ha obtenido sus mejores resultados resolviendo los dos problem por separado.
  - Un punto importante a la hora de desarrollar un sistema, independientemente d tamaño del mismo, es la modularidad, donde se busca mínimo acoplamiento y máxim

- Se admite un abanico de condiciones muy amplio. Esto desemboca en una gran expresividad pero complica la sintaxis, dificultando así su utilización.
- El abarcar más que la etiquetación contribuye a la complejidad del formalismo.

aramente las *Constraint Grammars* son muy potentes. Sin embargo, el añadir potencia
a una disminución de la velocidad del etiquetador, y aumenta la complejidad de manejo
lementación. Puede ser que el añadir una regla más aumente el tiempo de ejecución de
las reglas del etiquetador y, en la práctica, esta nueva regla sólo sirva para añadir un par
os más que van a resolver unas pocas ambigüedades en un texto grande.
r todo lo comentado hasta aquí, quizá sea mejor definir un nuevo lenguaje para el
lismo de reglas, donde se mantenga un compromiso entre potencia, sencillez, eficacia y
dad de ejecución. Este nuevo formalismo es el definido por [Reboredo y Graña 2000], que
a en las *Constraint Grammars*, pero que difiere en los siguientes puntos:

Sintaxis más intuitiva y legible.

Elimina el manejo de términos sintácticos.

Simplifica las condiciones (incluye todos los tipos de condición definidos en las *Constraint Grammars*).

Permite comparar etiquetas con palabras no ambiguas y ambiguas.

LEMMERS: Lenguaje de Etiquetado MEdiante Restricciones Simples.

es el lenguaje para el formalismo de reglas definido por [Reboredo y Graña 2000]. Tras
estudiado y habernos familiarizado con este lenguaje nos dimos cuenta de que la notación
da para definir estas reglas no es muy clara y puede conducir a error. Veámoslo a
uación.
s reglas siguen la clasificación hecha por Samuelsson, Tapanainen y Voutilainen
muelsson *et al.* 1996]. De esta manera nos encontramos con:

Reglas locales, que son de la forma:

Acción (*Categoría de ejecución*) ([**NOT**] *Entero* [**ALGUNO**] *ListaValores*, ...);

siendo:

- *Acción*: acción a ejecutar si se cumplen las condiciones de la regla.
  Hay tres acciones posibles:
  - **BORRA**: elimina una o varias etiquetas dentro de las posibles de la palabra, según qué etiqueta se indique en el apartado de *Categoría de ejecución*. La regla no se ejecuta si dicha etiqueta no existe o es la única para la palabra.
  - **SELECCIONA**: selecciona una o varias etiquetas dentro de las posibles de la palabra.
  - **FUERZA**: fuerza a que la palabra tenga la etiqueta dada aunque dicha etiqueta

- *ListaValores*: es una lista de valores (para más información consultar la sección 5.1 de [Reboredo y Graña 2000]). Una condición se cumplirá si se cumple la comparaci de los valores de la palabra indicada por *Entero* con todos los valores indicados *ListaValores*. Si alguno de estos valores fuera un conjunto, para ese valor sólo mirará que pertenezca al conjunto.

El primer problema con el que nos encontramos es el significado de la coma y los punt suspensivos que van a continuación de la *ListaValores*. No queda claro si representan y-lógico ó un o-lógico. Lo único que sabemos es que los puntos suspensivos indican que puede repetir lo definido más de una vez.

Siguiendo con las reglas, vemos claramente que existen cuatro tipo de comparacion debido a las opciones NOT y ALGUNO. Vamos a verlas a continuación y analizaremos presentan algún problema:

(a) `Entero ListaValores`: se comparan todos los valores de la lista con todos los valor de la palabra indicada por entero. Devolverá verdadero si la palabra es no ambig y los valores encajan o, si la palabra es ambigua y las comparaciones son sólo c literales[1].
Veamos algunos ejemplos[2]:

Caso a)

El	sobre
D	S
P	P
	V

```
BORRA (D) (1 { S } );
```
Esta regla no se ejecutaría porque `sobre` *es una palabra ambigua y la comparación estamos haciendo con etiquetas.*

```
BORRA (D) (1 @sobre);
```
Esta regla si se ejecutaría ya que `sobre` *es una palabra ambigua y la comparación estamos haciendo con un literal.*

Caso b)

El	niño
D	S
P	

```
BORRA (P) (1 { S } );
```
Esta regla si se ejecutaría porque `niño` *no es una palabra ambigua y los valor encajan.*

[1] Valor que se utiliza para hacer referencia a una palabra que aparece en el texto. Se escribe la palabra tal c precedida de @. Existen dos literales especiales, @<@ y @>@, que son principio de oración y final de oraci respectivamente.

[2] Para este y futuros ejemplos suponemos que tenemos un *tag set* reducido formado por las siguientes etiquet D → Determinante. Pron → Pronombre. V → Verbo.

y las condiciones no se cumplen, no se sabe bien qué es lo que devuelve. Podría ser verdadero porque se trata de la negación de la anterior, pero también podría ser falso. Sin embargo nos dice que se devuelve verdadero si la palabra es ambigua.
Veámoslo con un ejemplo:

El	sobre
D	S
P	P

```
BORRA (P) (NOT 1 V);
```

Esta regla podría devolver:
Verdadero, porque la palabra `sobre` *es ambigua independientemente de que se cumpla o no la condición.*
Verdadero, porque la palabra `sobre`, *independientemente de ser ambigua, no tiene* `V` como etiqueta.
Falso, porque a pesar de que la palabra `sobre` *sea ambigua, la etiqueta* `V` *no es una de las candidatas.*

Resumiendo, no queda claro a qué afecta el NOT ni cómo se tratan las ambigüedades.

(c) `Entero ALGUNO ListaValores`: como el primero, salvo que si la palabra es ambigua, se comprueba si los valores con los que se compara están en alguna de las opciones.
Veamos un ejemplo donde:

El	sobre
D	S
P	P
	V

```
BORRA (P) (1 ALGUNO S);
```

Esta regla si se ejecutaría ya que la palabra `sobre` *es ambigua .*

Pero si la palabra en cuestión no fuese ambigua no queda claro qué ocurriría.

(d) `NOT Entero ALGUNO ListaValores`: es la negación exacta de la anterior. Los valores indicados no deben existir en ninguna de las opciones.

Reglas barrera, son de la forma:

Acción (*Categoría de ejecución*) (*Dirección* [**ALGUNO**] $ListaValores_1$
$<$ **BARRERA** $>$ [**ALGUNO**] $ListaValores_2$,...);

siendo:

- *Acción*, *Categoría de ejecución* y *ListaValores* lo mismo que en el apartado anterior.
- *Dirección*: símbolo que indicará la dirección en que funcionará la regla.

Podemos apreciar a simple vista que este tipo de reglas van a presentar los mismos

Acción **SIEMPRE** (*Categoría de ejecución*);

con la restricción de que la acción no podrá ser un `FUERZA`, ya que dejaría todo el tex con la misma etiqueta.

Este tipo de regla habla por sí misma, con lo cual es lo suficientemente clara.

Como conclusión, después de haber analizado con cuidado todos estos puntos, decidim mejorar la sintaxis del tipo de reglas que presentan problemas con el propósito de hacerlas m sencillas, más expresivas y más fáciles de entender.

5.2 Mejoras del formalismo de reglas del lenguaje LEMMERS

Las mejoras que hemos considerado sobre el formalismo de reglas del lenguaje LEMMEI sigue respetando la clasificación de Samuelsson, Tapanainen y Voutilainen indica en [Samuelsson *et al.* 1996]:

1. Reglas locales:
   La alternativa que proponemos básicamente va a influir sobre este tipo de reglas, que co veremos a continuación resultarán más legibles y sencillas.
   Su forma será:

   Acción (*CategoríaEjecución*) ([**NOT**] *Entero Condición ConjuntoEtiquetas*, ...);

   siendo:

   - *Acción*: acción a ejecutar si se cumplen las condiciones de la regla. Hay tres accion posibles, que se corresponden con las indicadas en el apartado anterior.
   - *CategoríaEjecución*: categoría o etiqueta sobre la que se ejecuta la acción. En el ca de ser una categoría, se extiende la regla en n reglas diferentes, siendo n el núme de etiquetas que se pueden formar a partir de dicha categoría. Por ejemplo si n hablan de *sustantivo* y tenemos un conjunto extenso de etiquetas para sustantivo, generarían tantas reglas como etiquetas diferentes existen para sustantivo. Es dec para:

sustantivo	Scms
	Scmp
	Scfs
	Scfp

     generaríamos cuatro reglas diferentes, cada una con una etiqueta.
   - *Entero*: entero que indica la posición de la palabra, respecto de la palabra actual, c la que queremos comparar los valores en una condición.

será verdadera si el conjunto de etiquetas indicado por *Entero* es igual a *ConjuntoEtiquetas*. En caso contrario, la condición de la regla será falsa y por consiguiente ésta no ejecutará su acción.
Si la palabra indicada por *Entero* tiene una sola etiqueta, entonces las acciones `BORRA` y `SELECCIONA` nunca se ejecutarán, independientemente del valor de la condición. Tampoco se ejecutarán si dicha etiqueta no existe.

- **CONTIENE**: simula la operación de inclusión de conjuntos. Es decir, la condición de la regla será verdadera si el conjunto de etiquetas indicado por *Entero* incluye a *ConjuntoEtiquetas*, o lo que es lo mismo si *ConjuntoEtiquetas* está incluido en el conjunto indicado por *Entero*. Cabe destacar que hablamos de inclusión estricta, con lo cual no se permite la igualdad. Si se quiere representar un caso en el que se dé la condición de $\supseteq$ se deben escribir dos reglas:
  Acción (etiqueta) (Entero `ES` *ConjuntoEtiquetas);*
  Acción (etiqueta) (Entero `CONTIENE` *ConjuntoEtiquetas);*

- **PERTENECE**: antes de nada indicar que esta operación únicamente se utilizará cuando queramos que intervengan palabras en la condición de la regla. Simula la operación de pertenencia de conjuntos. Es decir, la condición será verdadera si la palabra indicada por *Entero* es un elemento de *ConjuntoEtiquetas*.

- *ConjuntoEtiquetas*: recalcar que se trata de un *conjunto* de elementos, donde los elementos pueden ser categorías o etiquetas. En el caso de las categorías lo que se hace es expandir dicha categoría en sus etiquetas correspondientes. De manera que el conjunto estará formado por todas las etiquetas que se pueden obtener a partir de dicha categoría, además de lo que ya había si fuera el caso. Por ejemplo, para:

sustantivo	Scms
	Scmp
	Scfs
	Scfp

obtendríamos un conjunto donde por lo menos aparecerían estas cuatro etiquetas.

Ejemplo 5.1 A continuación se muestran algunos casos donde las condiciones de las reglas son verdaderas:

(a)

El	toca	el	bajo
D	V	D	S
⋮	⋮	⋮	

$\Longrightarrow$ `BORRA (D) (3 ES { S } );`

(b)

El	toca	el	bajo
D	V	D	S
⋮	⋮	⋮	P

$\Longrightarrow$ `BORRA (D) (3 ES { S, P } );`

(c)

El	toca	el	bajo
D	V	D	S
.	.	.	.

$\Longrightarrow$ `BORRA (D) (3 CONTIENE { S } );`

Es importante mencionar que se van a permitir construir reglas con condiciones múltipl
simulando un y-lógico. Su forma es:

Acción (*CategoríaEjecución* ([**NOT**] *Entero Condición*$_1$ *ConjuntoEtiquetas*,
[**NOT**] *Entero Condición*$_2$ *ConjuntoEtiquetas*,
...,
[**NOT**] *Entero Condición*$_n$ *ConjuntoEtiquetas*);

Para simular un o-lógico se escriben tantas reglas con condiciones simples como operand
lógicos queremos que participen en la operación. Tiene la siguiente forma:

Acción (*CategoríaEjecución* ([**NOT**] *Entero Condición*$_1$ *ConjuntoEtiquetas*);
Acción (*CategoríaEjecución*) ([**NOT**] *Entero Condición*$_2$ *ConjuntoEtiquetas*);
...;
Acción (*CategoríaEjecución*) ([**NOT**] *Entero Condición*$_n$ *ConjuntoEtiquetas*);

2. Reglas barrera, que son de la forma:

Acción (*Categoría de ejecución*) (*Dirección Condición ConjuntoEtiquetas*$_1$ **BARREI**
Condición ConjuntoEtiquetas$_2$,
...);

siendo:

- *Acción, CategoríaEjecución, Condición, ConjuntoEtiquetas* lo mismo que lo indica
en el apartado anterior.
- *Dirección*: símbolo que indicará la dirección en la que funcionará la regla:

  $+*$ que indica hacia la derecha o adelante.
  $-*$ que indica hacia la izquierda o atrás.

Ejemplo 5.2 Un ejemplo de este tipo de regla podría ser el siguiente:

```
BORRA (S) (-* CONTIENE {V} BARRERA ES {D});
```

que significa, si antes de una palabra hay un *verbo* y entre ese verbo y es palabra no h
ninguna palabra que sea un *determinante*, esa palabra no podrá ser un *sustantivo*.

3. Reglas especiales, exactamente iguales a las vistas en el punto anterior.

tamente.

Reglas Locales, que como vimos son:

Acción (*CategoríaEjecución*) ([**NOT**] *Entero Condición ConjuntoEtiquetas*, ...);

y su esquema general obedece a la siguiente estructura:

- Si sólo intervienen etiquetas:

{BORRA | SELECCIONA | FUERZA} (CategoríaEjecución)
({0 | $n > 0$ | $n < 0$} {ES | CONTIENE} (ConjuntoEtiquetas), ...);

{BORRA | SELECCIONA | FUERZA} (CategoríaEjecución)
(NOT {0 | $n > 0$ | $n < 0$} {ES | CONTIENE} (ConjuntoEtiquetas), ...);

- Si intervienen palabras:

{BORRA | SELECCIONA | FUERZA} (CategoríaEjecución)
({0 | $n > 0$ | $n < 0$} PERTENECE (ConjuntoPalabras), ...);

{BORRA | SELECCIONA | FUERZA} (CategoríaEjecución)
(NOT {0 | $n > 0$ | $n < 0$} PERTENECE (ConjuntoPalabras), ...);

Ejemplo 5.3 Veamos algunos ejemplos de reglas que se podrían aplicar al siguiente enrejado. También se va a reflejar la evolución del enrejado a medida que las reglas se van ejecutando.

Las reglas son:

(a) `BORRA (S) (-1 PERTENECE {Juan});`
La palabra actual no puede ser un *sustantivo* si la anterior es *Juan*.

(b) `SELECCIONA (D) (-1 ES {V}, 1 CONTIENE {S});`
Si la palabra anterior es un *verbo* y la siguiente puede ser un *sustantivo*, entonces la palara actual será una *preposición*.

(c) `SELECCIONA (S) (NOT -1 ES {P}, 0 PERTENECE {vino});`
La palabra actual será un *sustantivo* si es la palabra *vino* y la palabra anterior no es una *preposición*.

Y el enrejado va evolucionando de la siguiente forma:

Primera regla encajada y ejecutada.	Juan S	vino V	a P	probar V	este D Pron S	vino S V
Segunda regla encajada y ejecutada.	Juan S	vino V	a P	probar V	este D	vino S V
Tercera regla encajada y ejecutada.	Juan S	vino V	a P	probar V	este D	vino S

Ejemplo 5.4 Veamos un ejemplo en el que se refleje la utilidad de las reglas que llev un FUERZA.

El	fue	a	cenar
D	V	P	V

- `FUERZA (Pron) (0 ES {D}, NOT 1 ES {S});`
  Si la palabra actual puede ser un *determinante* o un *artículo*, y la siguiente palab no es un *sustantivo*, entonces la palabra actual debe ser forzosamente un *pronomb*

El	fue	a	cenar
Pron	V	P	V

2. Reglas barrera, que son:

Acción (*Categoría de ejecución*) (*Dirección Condición ConjuntoEtiquetas*$_1$ **BARREI**
Condición ConjuntoEtiquetas$_2$,
...);

obedecen al siguiente esquema general:

- Si participan etiquetas en las reglas:
  {BORRA | SELECCIONA | FUERZA} (CategoríaEjecución)
  {+∗,−∗} {ES | CONTIENE} (ConjuntoEtiquetas) < BARRERA >
  {ES | CONTIENE} (ConjuntoEtiquetas), ...);

{BORRA | SELECCIONA | FUERZA} (CategoríaEjecución)
{+∗,−∗} {ES | CONTIENE} (ConjuntoPalabras) < BARRERA >
PERTENECE (ConjuntoPalabras), ...);

- Si participan únicamente palabras:
  {BORRA | SELECCIONA | FUERZA} (CategoríaEjecución)
  {+∗,−∗} PERTENECE (ConjuntoPalabras) < BARRERA >
  PERTENECE (ConjuntoPalabras), ...);

Ejemplo 5.5 Veamos a continuación algunos ejemplos:

Juan	vino	a	probar	el	vino
S	S	P	V	A	S
	V			D	V
				P	

- BORRA (V) (−∗ ES {V} < BARRERA > CONTIENE {S});
  Si antes de una palabra hay un *verbo* y, entre ese verbo y esa palabra no hay ninguna palabra que pueda funcionar como *sustantivo*, esa palabra no será un verbo.

Juan	vino	a	probar	el	vino
S	S	P	V	A	S
	V			D	
				P	

□

Reglas especiales, que como vimos son de la forma:

Acción **SIEMPRE** (*Categoría de ejcución*);

y cuyo esquema de reglas es:

{BORRA | SELECCIONA} SIEMPRE (CategoríaEjecución);

Ejemplo 5.6 Un ejemplo de este tipo de reglas podría ser:
Supongamos que tenemos un diccionario donde aquellas palabras que no son del castellano las etiquetamos como palabraExtranjera. Suponiendo que sabemos *a priori* que el texto está escrito en castellano, podríamos incluir una regla que eliminase estas palabras. Y esta regla sería:

BORRRA SIEMPRE (palabraExtranjera);

□

Capítulo 6

Compilación de reglas contextuales utilizando traductores de estado finito

La ejecución tradicional de las reglas contextuales consiste en aplicar cada una de ellas sob
la cadena de entrada, llevando a cabo, en cada caso posible, los cambios oportunos. Es dec
consiste en un proceso iterativo en el que las reglas se van aplicando a la cadena entrada
forma secuencial. Tras la aplicación de la primera regla sobre la cadena de entrada obtendrem
como salida esa misma cadena, o bien sin ningún cambio, o bien con los cambios oportun
que indique dicha regla. A continuación se aplica la siguiente regla, y así sucesivamente has
que todas hayan sido aplicadas. De esta manera vamos alimentando a cada regla con la sali
de la regla anterior. Este modelo de ejecución es muy sencillo, sin embargo presenta algun
inconvenientes, entre los cuales destacan los siguientes:

- La complejidad temporal de este proceso iterativo es elevada, ya que resulta s
  proporcional al número de reglas contextuales y a la longitud de la cadena de entrada.

- Surge la necesidad de guardar salidas temporales para poder aplicar las siguientes regla

Como solución a este problema proponemos una compilación previa de las reglas. Es
proceso de compilación transformará el conjunto de reglas en una estructura matemáti
denominada *traductor de estado finito*, que es la que permitirá una ejecución más eficien
Las ventajas que obtenemos llevando a cabo la compilación de las reglas de esta manera son l
que se citan a continuación:

- No se lleva a cabo ningún proceso iterativo, sino que de una sola pasada realizamos tod
  los cambios pertinentes sobre la entrada. Es decir, la complejidad temporal no depen
  ahora del número de reglas contextuales, sino que resulta ser lineal respecto sólo a
  longitud de la cadena de entrada.

- No hay salidas intermedias ya que los cambios sobre la entrada se producen a medida q
  se avanza por el traductor.

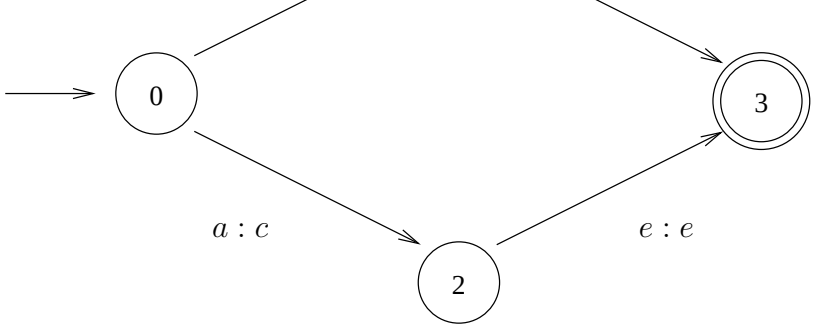


Figura 6.1: T: Ejemplo de traductor de estado finito

Definición formal de un traductor de estado finito

raductor de Estado Finito o *FST*[1] se puede ver como un*Autómata de Estado Finito* o donde los arcos del grafo van a estar etiquetados con un par de símbolos, el símbolo trada y su correspondiente símbolo de salida, en lugar de con un único símbolo de la [Roche y Schabes 1997].

nición 6.1 Un *Traductor de Estado Finito* o *FST* es una 6-túpla $T = (\Sigma_1, \Sigma_2, Q, i, F, E)$:

Σ_1 es un alfabeto finito, denominado *alfabeto de entrada*.

Σ_2 es un alfabeto finito, denominado *alfabeto de salida*.

Q es un conjunto finito de estados.

$i \in Q$ es el estado inicial.

$F \subseteq Q$ es el conjunto de estados finales.

$E \subseteq Q \times \Sigma_1^* \times \Sigma_2^* \times Q$ es el conjunto de transiciones y transformaciones.

□

plo 6.1 La representación gráfica del traductor que se indica a continuación es el que se ra en la figura 6.1:

$$(\{a, b, c, h, e\}, \{a, b, c, h, e\}, \{0, 1, 2, 3\}, 0, \{3\}, \{(0, a, b, 1), (0, a, c, 2), (1, h, h, 3), (2, e, e, 3)\})$$

□

continuación proporcionamos algunas definiciones básicas sobre los *FST*,s:

nición 6.2 Dado un *FST* $T = (\Sigma_1, \Sigma_2, Q, i, F, E)$, un *camino* de T es una secuencia $_i, b_i, q_i))_{i=1,n}$ de transiciones E tal que $q_i = p_{i+1}$ desde $i = 1$ hasta $n - 1$. □

[1] *nite State Transducer.*

Definición 6.4 La *primera proyección* y la *segunda proyección* de un *FST* T $(\Sigma_1, \Sigma_2, Q, i, F, E)$ son dos autómatas de estado finito $p_1(T)$ y $p_2(T)$ tales que:

$$p_1(T) = (\Sigma_1, Q, i, F, E_{p_1}) \text{ donde } E_{p_1} = \{(q, a, q') \mid (q, a, b, q') \in E\}$$
$$p_2(T) = (\Sigma_2, Q, i, F, E_{p_2}) \text{ donde } E_{p_2} = \{(q, b, q') \mid (q, a, b, q') \in E\}$$

Como en el caso de los *FSAs*, los *FSTs* son muy potentes por sus propiedades algorítmic y de cierre.

- *Cierre bajo la Union.* Si T_1 y T_2 son dos *FSTs*, entonces existe un *FST* $(T_1 \cup T_2)$ don $\forall u \in \Sigma^*, (T_1 \cup T_2)(u) = T_1(u) \cup T_2(u)$

- *Cierre bajo la Inversión.* Si $T = (\Sigma_1, \Sigma_2, Q, i, F, E)$ es un *FST*, existe un *FST* T^{-1} tal q $T^{-1}(u) = \{v \in \Sigma^* \mid u \in T(v)\}$. Además, $T^{-1} = (\Sigma_2, \Sigma_1, Q, i, F, E^{-1})$ tal que:

$$(q_1, a, b, q_2) \in E^{-1} \Leftrightarrow (q_1, b, a, q_2) \in E$$

- *Cierre bajo la Composición.* Dados dos *FSTs* $T_1 = (\Sigma_1, \Sigma_2, Q_1, i_1, F_1, E_1)$ y T_2 $(\Sigma_2, \Sigma_3, Q_2, i_2, F_2, E_2)$, entonces existe un traductor $(T_1 \circ T_2)$ tal que $\forall u \in \Sigma_1^*, (T_1 \circ T_2)(u)$ $T_2(T_1(u))$, dando lugar al traductor $T_3 = (\Sigma_1, \Sigma_3, Q_1 \times Q_2, (i_1, i_2), F_1 \times F_2, E_3)$ tal que:

  $E_3 = \{((x_1, x_2), a, b, (y_1, y_2)) \mid \exists c \in \Sigma_2 \cup \{\epsilon\}$ tal que $(x_1, a, c, y_1) \in E_1$ y $(x_2, c, b, y_2) \in E$

  y de manera que:

$$T_3(u) = (T_1 \circ T_2)(u) = T_2(T_1(u)), \forall u \in \Sigma_1^*$$

Definición 6.5 Un traductor de estado finito $T = (\Sigma_1, \Sigma_2, Q, i, F, E)$ es un *traductor sin transiciones* si:

$$E \subseteq Q \times \Sigma_1 \times \Sigma_2 \times Q$$

6.2 Construcción de un traductor a partir de reglas contextuale

A partir de los esquemas de reglas introducidos en el capítulo anterior, es labor de los lingüist proporcionar las reglas contextuales. Una vez que tengamos disponibles las reglas contextuale estaremos en condiciones de construir el traductor de estado finito asociado a cada una de ella No obstante, antes de profundizar en la construcción de los *FST*,s, resulta interesante ver forma de trabajo de nuestro sistema, así como los formatos internos con los que va a operar traductor. Esto permitirá comprender con mayor claridad la forma en la que se construyen l

os procesos de construcción y ejecución de los traductores sean más eficientes, y para que opios traductores en sí mismos sean más compactos, éstos trabajarán internamente con ros enteros como sistema de representación de las palabras y de las etiquetas, en lugar abajar con los caracteres que conforman dichas palabras y etiquetas. De esta manera, ndo de las reglas, hacemos lo siguiente:

Los enteros 0, 1 y -1 están reservados. El 0 representa la cadena vacía o *epsilon*, mientras que el -1 y el 1 representan los puntos de comienzo y final de la frase a etiquetar, respectivamente.

Con todas las palabras que aparecen involucradas en las reglas contextuales creamos un *mini-diccionario*, numerando cada una de ellas con un número negativo entre el -3 y el $-m$, siendo $m-2$ el número de palabras diferentes que aparecen en dicho conjunto de reglas. Se reserva el -2 como comodín, para aquellas palabras que aparezcan en las frases a etiquetar, y no en las reglas.

Comprobamos que todas las etiquetas a las que se hace referencia en las reglas contextuales existen en el *tag set* de nuestro sistema. Construimos entonces un *mini-tag-set* con dichas etiquetas, a las que asignamos un número positivo comprendido entre el 3 y el n, siendo $n-2$ el número total de etiquetas diferentes que aparecen en las reglas. Los números se asignan a las etiquetas según el orden en el que dichas etiquetas aparecen en el *tag set*. Se reserva el 2 como comodín para el resto de etiquetas del *tag set*.

r tanto el mini-diccionario y el mini-tag-set contendrán el *mapping* de todos los símbolos articipan en las reglas que nos servirán para la creación de los *FST*,s. Es decir, finalmente emos algo del estilo:

PR	palabras $\notin$ PR	Inicio de frase	Epsilon	Fin de frase	etiquetas $\notin$ ER	ER
-m, ..., -4, -3	-2	-1	0	1	2	3, 4, ..., n

PR y ER son los conjuntos de palabras y etiquetas que participan en las reglas, ctivamente.

do tengamos todos los traductores creados, uno para cada regla contextual, los naremos para obtener un único traductor. Para poderlo poner en funcionamiento tamos una entrada, que estará formada por una frase y su enrejado correspondiente, donde quetas del enrejado deben estar ordenadas en función de cómo aparecen en el *tag set*. Para darle la entrada al traductor necesitamos hacer un *mapping* de la entrada original a una la equivalente, que entienda el traductor, y que vendrá determinada en función del mini-nario y del mini-tag-set. Una vez que el traductor haya procesado la entrada y dé una , ésta deberá ser igualmente transformada a su nuevo enrejado equivalente. Este proceso eja claramente en la figura 6.2.

Figura 6.2: Proceso de ejecución.

- Para cada palabra de la frase de entrada a etiquetar realizamos la siguiente comprobació si la palabra está en nuestro mini-diccionario, le asignamos el número negati correspondiente; de no ser así, le asignamos el comodín (−2).

- Para cada columna de etiquetas del enrejado procedemos de la siguiente manera. Cogem cada una de las etiquetas y miramos si está en el mini-tag-set. Si es así nos quedamos c el número positivo asociado. Si no es así, le asignamos el comodín (2).

La entrada para el traductor estará delimitada por los caracteres especiales de principio y f de frase, −1 y 1 respectivamente, y estará formada por un número negativo que representa a u palabra y por todas sus etiquetas posibles, que se corresponden con números positivos. Aq merece la pena comentar que el comodín podría aparecer una o más veces, pero para simplific la operación de los traductores, lo tenemos en cuenta una única vez. Esto no va a repercutir el funcionamiento correcto del traductor, y así tendremos el conjunto de etiquetas de la entra del traductor ordenado de menor a mayor. Por ejemplo, tendremos una entrada del estilo:

−1	−5	4	7	9	−2	2	5	7	...	−4	3	6	8	1
Inicio frase	Palabra	Etiquetas			Palabra	Etiquetas			...	Palabra	Etiquetas			Fin frase

Este proceso tendrá que repetirse de forma inversa una vez obtenida la salida del traducto Cabe destacar que hay que guardar la correspondencia de los números con sus etiquetas pa reconstruir el enrejado de salida. Sobre todo, hay que tener cuidado con aquellas etiquetas q se correspondan con los comodines. La forma de actuar consiste en, partiendo de la secuenc de salida del traductor comprobar si ésta ha variado con respecto a la entrada. Para ello, la salida se coge una palabra con sus etiquetas correspondientes y se mira en la secuencia entrada si se han producido cambios. Si es así, se construye el nuevo enrejado en función de l mismos. Si no hay cambios, el enrejado queda tal cual. En caso de haber comodines, como só va a aparecer uno, se comprueba si había o no varios en el enrejado de entrada equivalente. los había, se deberán tener todos en cuenta y escribir sus etiquetas correspondientes.

Ejemplo 6.2 Consideremos el siguiente conjunto de reglas contextuales[3]:

```
SELECCIONA (D) (−1 ES {V}, 0 PERTENECE {el}, 1 CONTIENE {S});
BORRA (S) (−1 ES {D}, 1 PERTENECE {encima,sobre});
FUERZA (P) (−1 ES {S}, 2 ES {S});
SELECCIONA (V) (−1 ES {S},0 PERTENECE {dejar,fue,vino});
```

El mini-diccionario y el mini-tag-set asociados serán:

[3]Para todos los ejemplos consideraremos las siguientes abreviaturas para representar las etiquetas:
Adj → Adjetivo. P → Preposición. S → Sustantivo.

fue	−6	V	6
sobre	−7		
vino	−8		

pongamos que tenemos como enrejado de entrada:

María	vino	para	dejar	el	bajo	sobre	la	mesita	pequeña
S	S	P	V	D	P	S	D	S	Adj
	V			Pron	S				

enrejado equivalente teniendo en cuenta el mini-diccionario y el mini-tag-set sería:

−2	−8	−2	−3	−4	−2	−7	−2	−2	-2
5	5	4	6	3	4	5	3	5	2
	6			2	5				

la entrada para el traductor tendrá el formato indicado a continuación:

$$-1, -2, 5, -8, 5, 6, -2, 4, -3, 6, -4, 2, 3, -2, 4, 5, -7, 5, -2, 3, -2, 5, -2, 2, 1$$

uviéramos como salida:

$$-1, -2, 5, -8, 6, -2, 4, -3, 6, -4, 3, 4, -2, 5, -7, 4, -2, 3, -2, 5, -2, 2, 1$$

amos a cabo el *mapping* inverso. Para la primera palabra vemos que no se han producido os, entonces queda todo tal cual. En cambio para la segunda palabra vemos que se ha cido un cambio, y no participan comodines, entonces nos quedamos con los cambios. Para ma palabra vemos que no se han producido cambios y que nos encontramos con un comodín etiqueta, entonces recuperamos la etiqueta que representa dicho comodín. Los enrejados antes, con y sin *mapping*, serían:

−2	−8	−2	−3	−4	−2	−7	−2	−2	-2
5	6	4	6	3	5	4	3	5	2

María	vino	para	dejar	el	bajo	sobre	la	mesita	pequeña
S	V	P	V	D	S	P	D	S	Adj

□

Construcción de los traductores de estado finito

tinuación vamos a ver los traductores de estado finito que representan los esquemas ales de reglas. No vamos a mostrar todos por el gran número de esquemas de reglas es, pero si indicaremos los más interesantes y significativos.

Reglas locales:
Existen muchas combinaciones posibles para este tipo de reglas, tal y como se puede ver en el capítulo anterior, pero analizaremos las siguientes:

Figura 6.3: *FST* para: `BORRA` (etq) $(0$ `ES` $\{etq_j, ..., etq, ..., etq_k\})$;

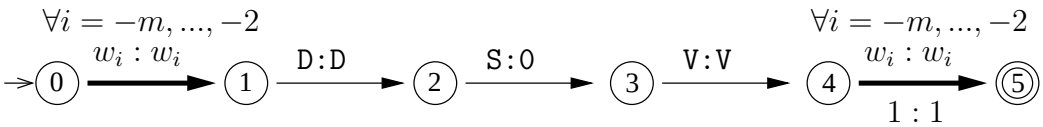


Figura 6.4: *FST* para: `BORRA (S) (0 ES {D,S,V});`

- **BORRA** (etq) $(0$ **ES** $\{etq_j, ..., etq, ..., etq_k\})$**;**
  Esta regla nos dice que la palabra actual no puede tener como etiqueta *etq* si s
  etiquetas posibles son exactamente las indicadas en el conjunto de etiquetas de
  regla $\{etq_j, ..., etq, ..., etq_k\}$.
  El traductor que simula el funcionamiento de esta regla se muestra en la figura 6
  De este traductor caben mencionar los siguientes puntos:
  - En cuanto a la topología de este y de futuros traductores, es necesario aclarar q
    las transiciones que representan palabras se dibujan con una flecha más grue
    para facilitar la lectura. Después de cada palabra nos encontraremos con una
    más transiciones que representan etiquetas, donde por lo menos debe aparec
    una, ya que en el enrejado de entrada no puede aparecer ninguna palabra s
    etiquetas asignadas. También con la intención de facilitar la comprensión
    todos estos traductores, los símbolos que aparecen en las transiciones pued
    hacer referencia a las palabras y etiquetas concretas. No obstante, debe qued
    claro que los traductores reales trabajarán internamente sólo con los núme
    enteros obtenidos a partir del *mapping* correspondiente. De momento, en est
    primeros ejemplos y esquemas generales de traductores, allí donde no aparez
    un número entero concreto, utilizaremos w_i para hacer referencia a las palabr
    y etq_i para hacer referencia a las etiquetas. En el caso de los w_i, el subíndic
    podrá variar desde -2 hasta $-m$, siendo w_{-2} el comodín para palabras. Y
    el caso de los etq_i, el subíndice i podrá variar desde 2 hasta n, siendo etq_2
    comodín para etiquetas.
  - La acción de borrado en el traductor se refleja transformando la etiqueta
    cuestión en un *epsilon*, que se representa con un 0.
  - La condición de igualdad consiste en dibujar una transición por cada etiqueta q
    aparece en el conjunto de etiquetas de la regla. En dicha transición, los símbol
    de entrada y salida están constituidos por la propia etiqueta.
  - La última transición del traductor nos dice que después de la última etique
    forzosamente debe aparece o bien una palabra o bien el fin de frase.

Ejemplo 6.3 Consideremos la siguiente regla:

`BORRA (S) (0 ES {D,S,V});`

Esta regla dice que si las etiquetas posibles para la palabra actual son *determinan*
sustantivo, y *verbo* entonces nunca será *sustantivo*. El traductor de estado fini
correspondiente se muestra en la figura 6.4.

Figura 6.5: *FST* para: `SELECCIONA` (etq) $(0$ `ES` $\{etq_j, ..., etq, ..., etq_k\})$;

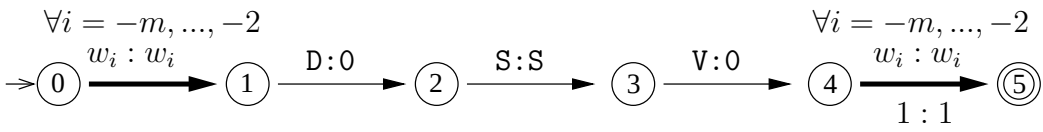


Figura 6.6: *FST* para: `SELECCIONA (S) (0 ES {D,S,V});`

- **SELECCIONA** (etq) $(0$ **ES** $\{etq_j, ..., etq, ..., etq_k\})$**;**
  Esta regla dice que la etiqueta de la palabra actual es *etq* si sus etiquetas posibles son exactamente las indicadas en el conjunto $\{etq_j, ..., etq, ..., etq_k\}$.
  El traductor equivalente se muestra en la figura 6.5. De este traductor, indicamos únicamente que la acción de selección se lleva a cabo transformando las etiquetas que no son susceptibles de ser seleccionadas en *epsilon*, dejando como está la relevante.

  Ejemplo 6.4 Consideremos la siguiente regla:

  `SELECCIONA (S) (0 ES {D,S,V});`

  Esta regla nos dice que la palabra actual es un *sustantivo*, si las etiquetas posibles para ella son *determinante*, *sustantivo* y *verbo*. El traductor que se corresponde con esta regla se muestra en la figura 6.6.

  □

- **FUERZA** (etq) $(0$ **ES** $\{etq_j, ..., etq_k\})$**;**
  Es decir, la etiqueta de la palabra actual se fuerza a *etq* si sus etiquetas posibles son exactamente el conjunto de etiquetas $\{etq_j, ..., etq_k\}$. Destacar que la etiqueta a forzar no es ninguna de las que están *a priori*.
  Su traductor de estado finito correspondiente es el de la figura 6.7. En este traductor, la acción consiste en transformar un *epsilon* en la etiqueta a forzar, poniendo el resto de etiquetas a *epsilon*.

  Ejemplo 6.5 Consideremos la siguiente regla:

  `FUERZA (S) (0 ES {A,D,V});`

  La palabra actual es un *sustantivo* si tiene como etiquetas posibles *artículo*, *determinante* y *verbo*. El traductor equivalente se muestra en la figura 6.8.

  □

- **BORRA** (etq) $(\mathbf{NOT}\ 0\ \mathbf{ES}\ \{etq_j, etq_k..., etq_l\})$**;**
  Es decir, la palabra actual no puede tener como etiqueta *etq* si su conjunto de

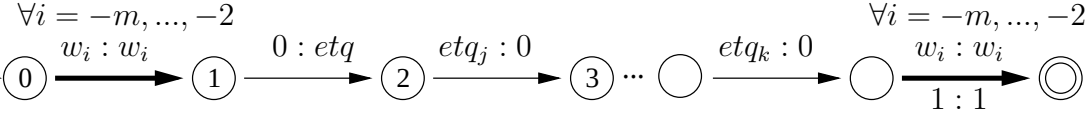


Figura 6.7: *FST* para: `FUERZA` (etq) $(0$ `ES` $\{etq_j, ..., etq_k\})$;

Figura 6.8: *FST* para: `FUERZA (S) (0 ES {A,D,V});`

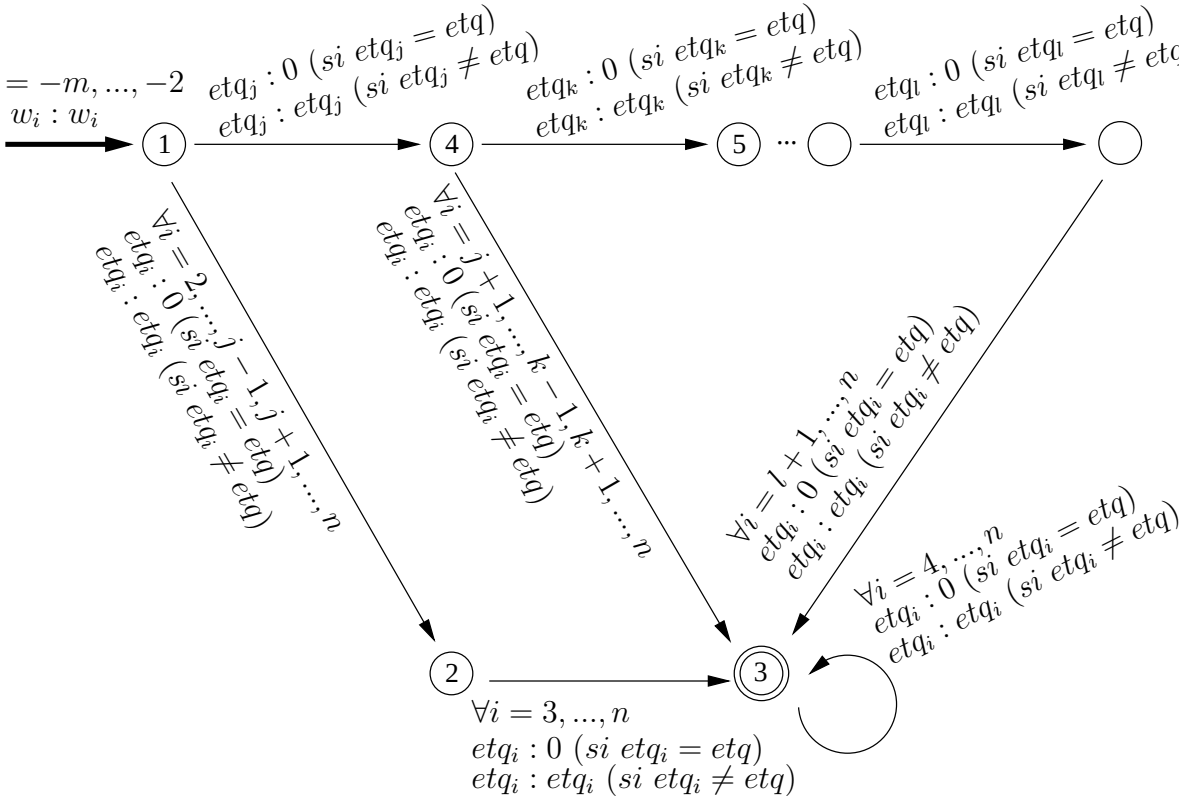


Figura 6.9: *FST* para: `BORRA` (etq) `(NOT 0 ES` $\{etq_j, etq_k..., etq_l\}$`);`

etiquetas posibles no es exactamente el indicado en el conjunto de etiquetas de regla $\{etq_j, etq_k..., etq_l\}$.

El traductor correspondiente es el que se muestra en la figura 6.9. Como podem observar, el NOT complica el diseño del traductor. Lo que se representa básicamen es la no aparición exacta del conjunto de etiquetas de la regla, pero lo que sí pued aparecer son subconjuntos de él con o sin más etiquetas, y superconjuntos de él. I esta manera el camino superior del grafo identifica el conjunto de etiquetas de regla, y muere en un estado que no es final si se cumple la igualdad.

Ejemplo 6.6 Consideremos la siguiente regla:

```
BORRA (V) (NOT 0 ES {Adj});
```

La palabra actual no puede ser un *verbo* si no tiene a *adjetivo* como etiqueta. traductor equivalente se muestra en la figura 6.10.

- **FUERZA** (etq) $(n > 0$ **ES** $\{etq_j, ..., etq_k\})$**;**

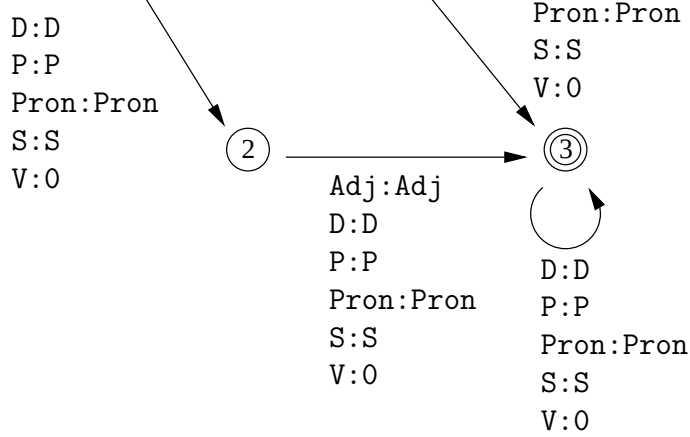


Figura 6.10: *FST* para: `BORRA (V) (NOT 0 ES {Adj});`

que está n posiciones más adelante respecto a la palabra considerada tiene como etiquetas posibles exactamente las indicadas en el conjunto de etiquetas de la regla $\{etq_j, ..., etq_k\}$.
El traductor que se corresponde con este esquema de reglas se encuentra en la figura 6.11. En este caso como la condición se refiere a una palabra distinta de la considerada, tenemos que construir un traductor que pase por todas las palabras pertinentes, dejándolo todo tal y como está, hasta llegar a la palabra afectada por la condición. Al tratarse de un n positivo, entonces la acción de la regla se realiza al principio del traductor y a continuación se pasa por todas las palabras necesarias hasta llegar a la palabra donde se comprueba la condición. Si por el contrario, el número n fuese negativo, el proceso sería inverso, es decir, se le daría la vuelta al traductor de manera que lo primero que se tendría en cuenta sería la condición de la regla, para después pasar por las palabras necesarias hasta llegar a la actual donde se llevaría a cabo la acción.

Ejemplo 6.7 Consideremos la siguiente regla:

`FUERZA (S) (1 ES {Adj});`

La palabra actual es forzosamente un *sustantivo* si la siguiente es un *adjetivo*. El traductor para esta regla se muestra en la figura 6.12.

□

- **BORRA** (etq) (0 **CONTIENE** $\{etq_j, etq_k..., etq_l\}$)**;**
  Es decir, la palabra actual no tendrá etq como etiqueta si entre sus etiquetas posibles nos encontramos las que aparecen en el conjunto de etiquetas de la regla $\{etq_j, etq_k..., etq_l\}$ y alguna más.
  El traductor que simula el comportamiento de este esquema de reglas es el indicado en la figura 6.13. Y de él vale la pena destacar la condición de inclusión, donde se tiene en cuenta la aparición de las etiquetas de la regla y, por lo menos, una más, ya

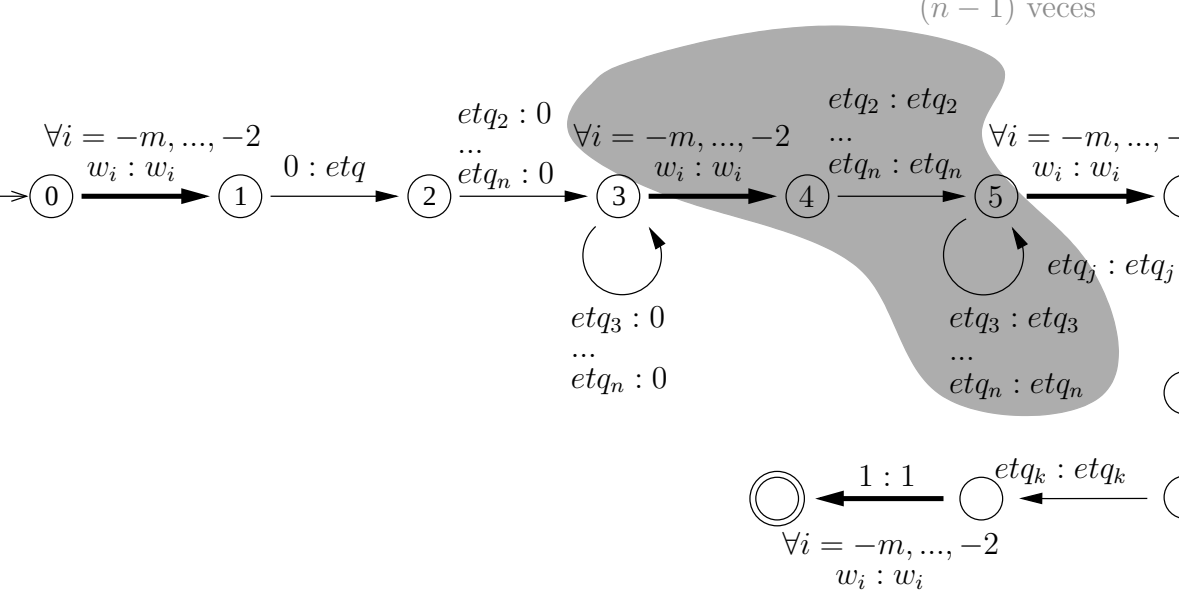


Figura 6.11: *FST* para: `FUERZA` (etq) $(n > 0$ `ES` $\{etq_j, ..., etq_k\})$`;`

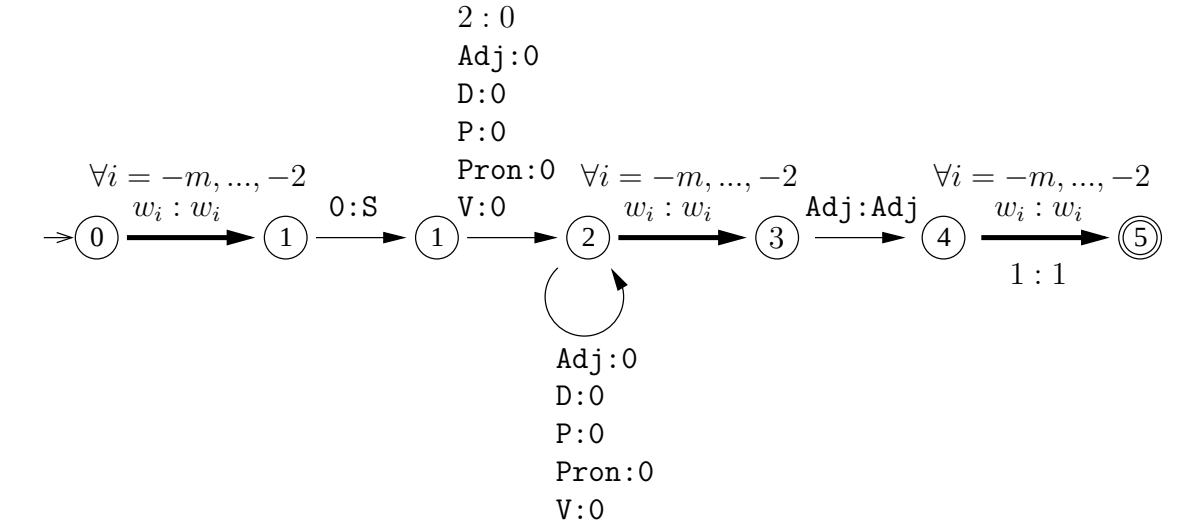


Figura 6.12: *FST* para: `FUERZA (S) (1 ES {Adj});`

```
BORRA (D) (0 CONTIENE {Pron,V});
```

La palabra actual no puede ser un *determinante* si entre sus etiquetas nos encontramos con un *pronombre* y un *verbo* como posibilidades. El traductor de estado finito equivalente es el de la figura 6.14.

□

- **SELECCIONA** (etq_e) (**NOT** $n > 0$ **CONTIENE** $\{etq_j, etq_k, ..., etq_h, etq_l\}$)**;**
  Es decir, la palabra actual tendrá como etiqueta a etq_e siempre y cuando la palabra de n posiciones hacia adelante no tiene entre sus etiquetas el conjunto de etiquetas $(etq_j, etq_k, ..., etq_h, etq_l)$.
  El traductor correspondiente a este esquema de reglas es el que se muestra en la figura 6.15. Como se puede ver la topología de este traductor es algo complicada.

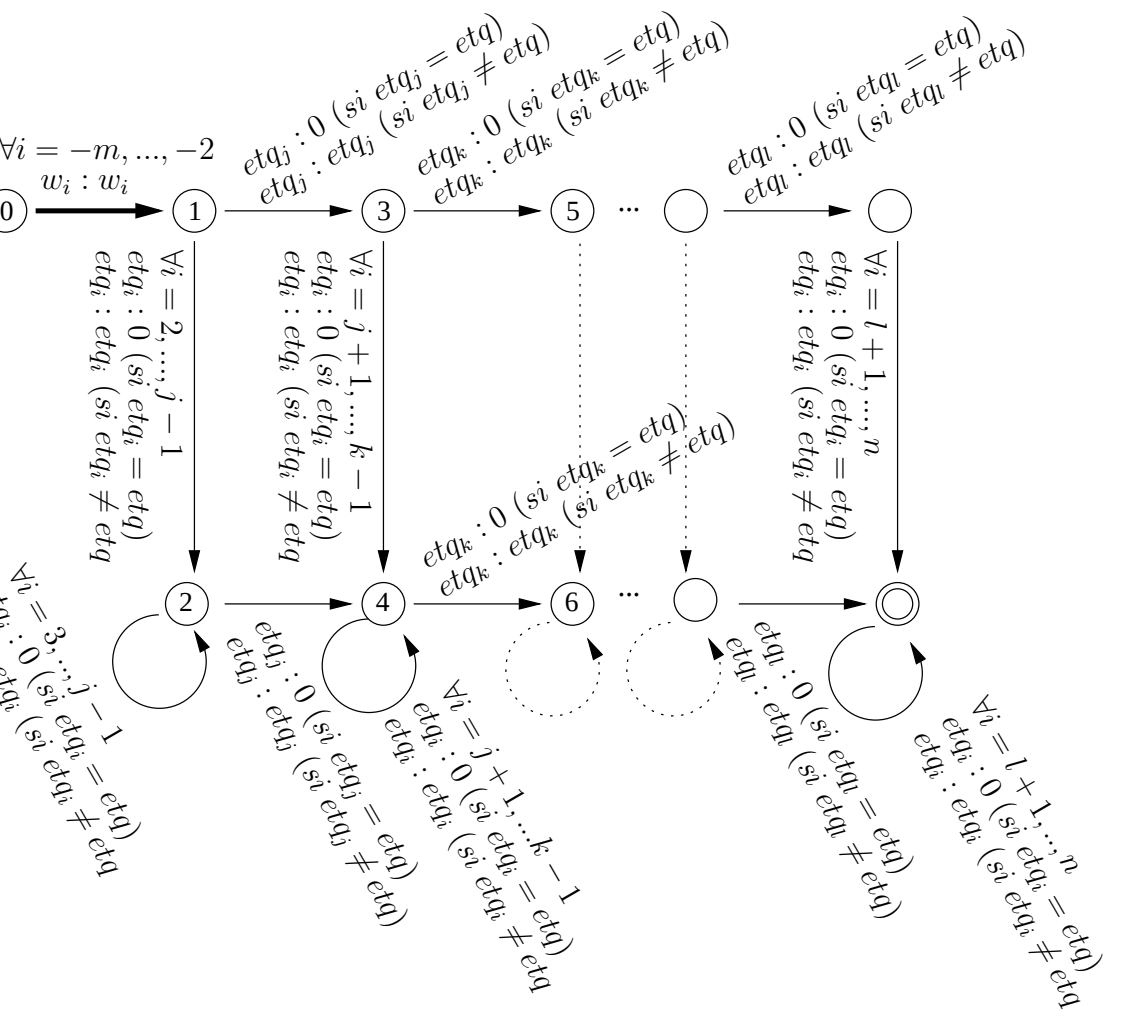


Figura 6.13: Traductor para: BORRA (etq) (0 CONTIENE $\{etq_1, etq_2, ..., etq_n\}$);

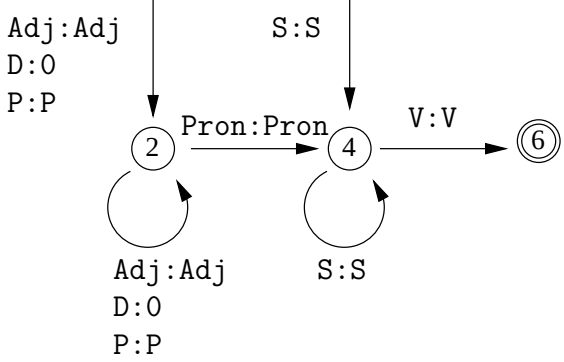


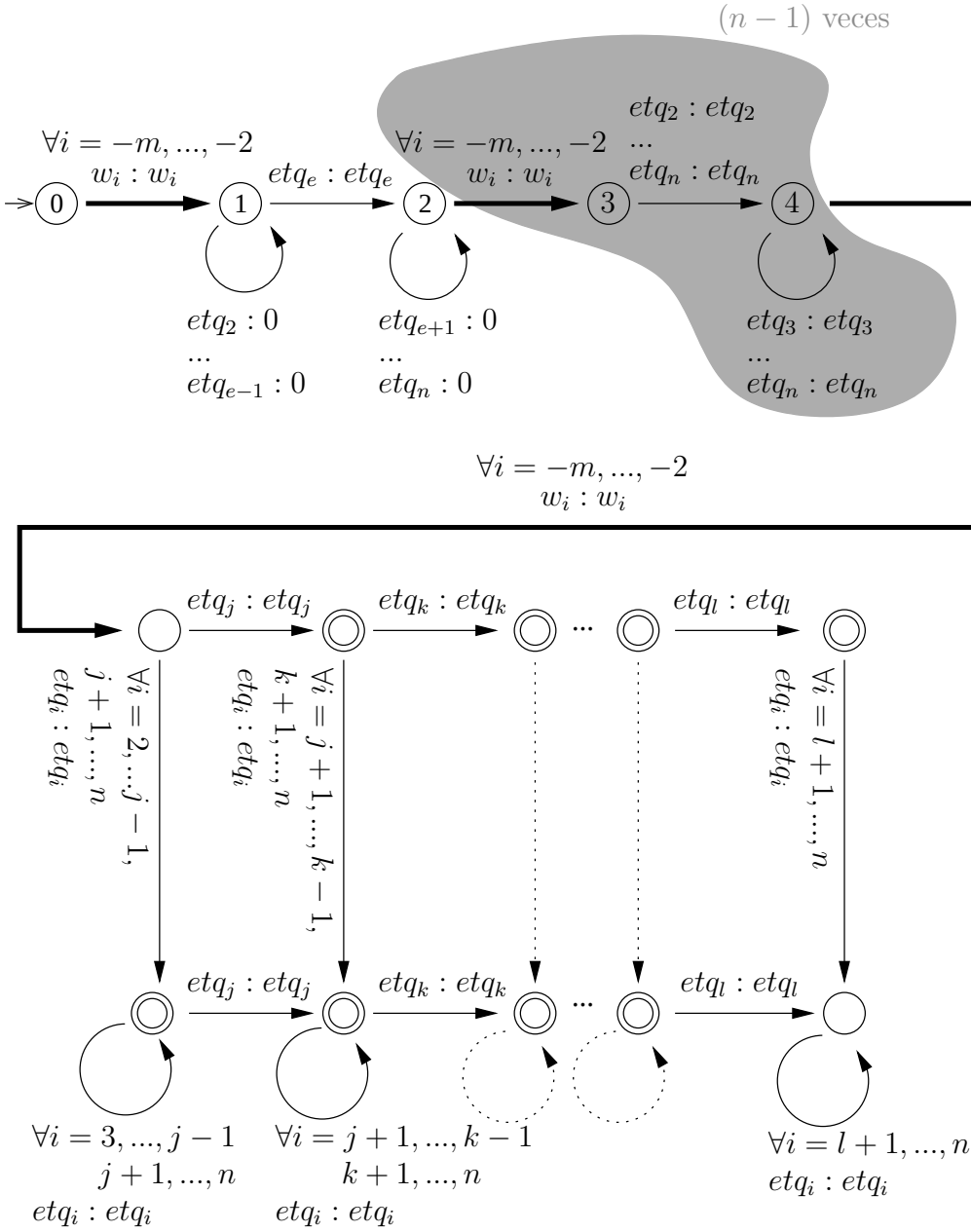
Figura 6.14: Traductor para: `BORRA (D) (0 CONTIENE {Pron,V});`

Esta complejidad principalmente viene determinada por la negación de la continenc de forma que la negación de $A \supset B$ es $A \subseteq B$, siendo A el conjunto de etiquetas la palabra afectada por la condición, y siendo B el conjunto de etiquetas de dic condición en la regla. Analizando un poco la situación, se puede ver que la condici de la regla será falsa únicamente para aquellos conjuntos A que contengan a B ent sus elementos, y en el resto de los casos la condición de la regla será verdadera. decir, la condición se cumple para todos los conjuntos que contengan a algún elemen del conjunto de partes del conjunto B, $P(B)$[4], menos para el propio conjunto B, el que únicamente se permite la igualdad. Veámoslo con un pequeño ejemplo pa aclarar un poco más esta idea. Supongamos que $B = \{1, 2, 3\}$, entonces:

	Verdadero	**Falso**
`CONTIENE` $A \supset B$	$A = \{1, 2, 3, ...\}$	$A = \{\varnothing, ...\}$ $A = \{1, ...\}$ $A = \{2, ...\}$ $A = \{3, ...\}$ $A = \{1, 2, ...\}$ $A = \{1, 3, ...\}$ $A = \{2, 3, ...\}$ $A = \{1, 2, 3\}$
`NOT CONTIENE` $A \subseteq B$	$A = \{\varnothing, ...\}$ $A = \{1, ...\}$ $A = \{2, ...\}$ $A = \{3, ...\}$ $A = \{1, 2, ...\}$ $A = \{1, 3, ...\}$ $A = \{2, 3, ...\}$ $A = \{1, 2, 3\}$	$A = \{1, 2, 3, ...\}$

[4]Sea C un conjunto cualquiera. Se llama *conjunto de partes del conjunto C*, escrito *P(C)*, al conjunto cuy elementos son todos los subconjuntos posibles del conjunto C.
Así, con C={a,b,c} se tiene que:

P(B) = {∅,{a},{b},{c},{a,b},{a,c},{b,c},{a,b,c}}



ura 6.15: *FST* para: `SELECCIONA` (etq_e) (`NOT` $n > 0$ `CONTIENE` $\{etq_j,\ etq_k, ...,\ etq_l\}$);

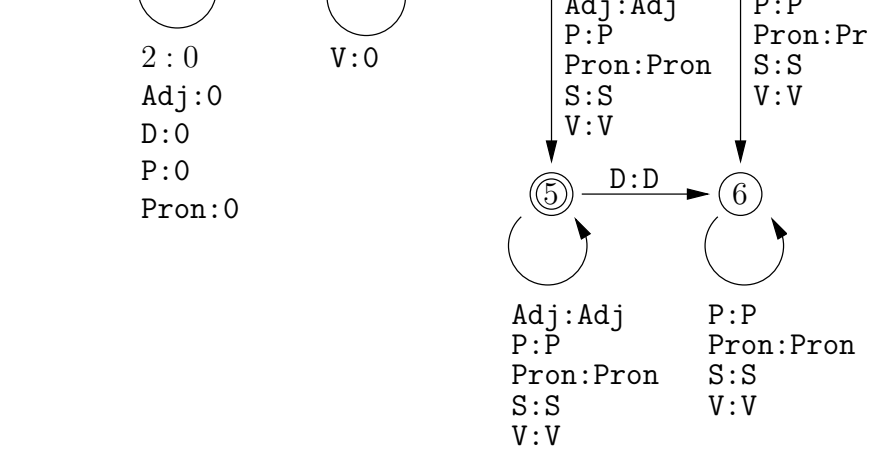


Figura 6.16: Traductor para: `SELECCIONA (S) (NOT 1 CONTIENE {D});`

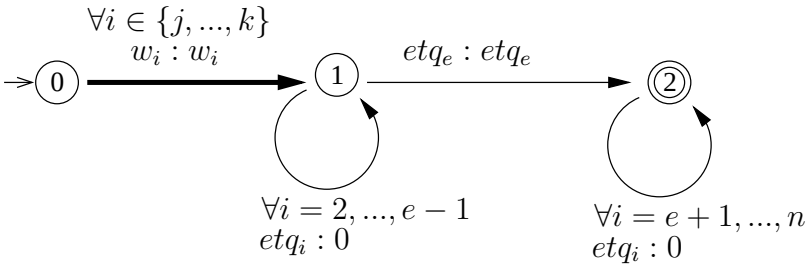


Figura 6.17: *FST* para: `SELECCIONA` (etq_e) (0 `PERTENECE` $\{w_j, ..., w_k\}$);

Ejemplo 6.9 Consideremos la siguiente regla:

```
SELECCIONA (S) (NOT 1 CONTIENE {D});
```

La palabra actual es un *sustantivo* si la siguiente palabra no puede ser
determinante. Esta regla se muestra en la figura 6.16

- **SELECCIONA** (etq_e) (0 **PERTENECE** $\{w_j, ..., w_k\}$);
  Es decir, la palabra actual tiene como etiqueta etq_e si es alguna de las que apare
  en el conjunto de palabras de la regla $\{w_j, ..., w_k\}$.
  El traductor correspondiente se muestra en la figura 6.17. De él solamente indicam
  que la condición de pertenencia para las palabras complica poco el diseño del traduct
  ya que únicamente se van a tener en cuenta aquellas palabras que aparecen en la reg
  en lugar de todas las del mini-diccionario como se venía haciendo hasta ahora, y
  van a ejecutar la acciones adecuadas en las etiquetas correspondientes.

Ejemplo 6.10 Consideremos la siguiente regla:

```
SELECCIONA (V) (0 PERTENECE {es,fue,tiene});
```

Si la palabra actual es alguna del conjunto {es,fue,tiene}, entonces su etiqueta

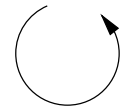


Figura 6.18: *FST* para: `SELECCIONA (V) (0 PERTENECE {es,fue,tiene});`

□

Reglas barrera:
Para este tipo de reglas también nos podemos encontrar con una gran variedad debido a los distintos tipos de condición que permitimos. Sin embargo, únicamente vamos a hacer hincapié en una sola regla. El lector puede fácilmente deducir los traductores de estado finito asociados al resto de las reglas ayudándose de las indicaciones del apartado anterior y de este.

- **SELECCIONA** (etq_e) $(+*$ **ES** $\{etq_j, etq_k, ..., etq_l\}$ $<$ **BARRERA** $>$ **ES** $\{etq_x, etq_y, ..., etq_z\})$**;**
  Es decir, si antes de una palabra nos encontramos con otra cuyas etiquetas se corresponden exactamente con el conjunto de etiquetas $\{etq_j, etq_k, ..., etq_l\}$ y, entre ambas palabras no hay ninguna otra cuyo conjunto de etiquetas sea exactamente $\{etq_x, etq_y, ..., etq_z\}$, entonces esa palabra tendrá a etq_e como etiqueta.
  El traductor de estado finito asociado con este esquema de reglas es el que se muestra en la figura 6.19. El diseño de este traductor es ligeramente complicado, tal y como se puede apreciar en la figura, donde se tienen en cuenta tres caminos alternativos, uno que se corresponde con la primera condición, otro que se corresponde con la segunda condición, y por último otro donde se considerarán el resto de las etiquetas del mini-tag-set. Otro factor que resulta interesante comentar es la necesidad de un lazo para completar la lógica de la regla, ya que se desconoce el número de palabras por la que es necesario navegar para determinar la resolución de las condiciones. Indicamos también que en este caso la acción de la regla se lleva a cabo al principio del traductor debido a que la dirección de ejecución es hacia la derecha $(+*)$. Sin embargo, si elegimos la dirección de ejecución contraria, hacia la izquierda $(-*)$, entonces la acción de la regla se llevará a cabo en las últimas transiciones del traductor.

  Ejemplo 6.11 Consideremos la siguiente regla:

  `SELECCIONA (V) (+* ES {S,P} <BARRERA> ES {D});`

  Si antes de una palabra hay otra que tiene la posibilidad de ser *sustantivo* o *verbo*

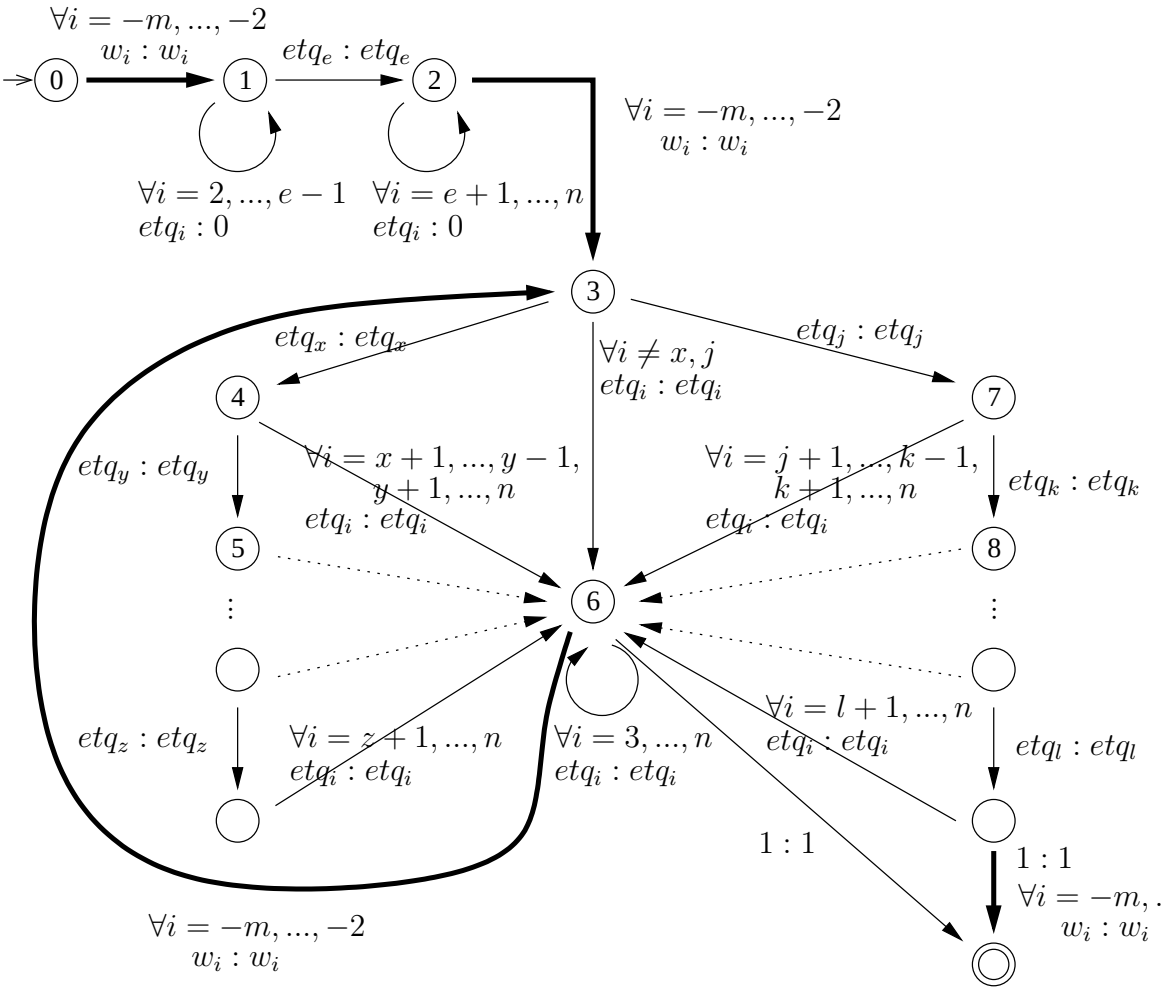


Figura 6.19: *FST* para: `SELECCIONA` (etq_e) $(+*$ `ES` $\{etq_j,\ etq_k,\ ...,\ etq_l\}$ $<$ `BARRERA` $>$ $\{etq_x,\ etq_y,\ ...,\ etq_z\})$`;`

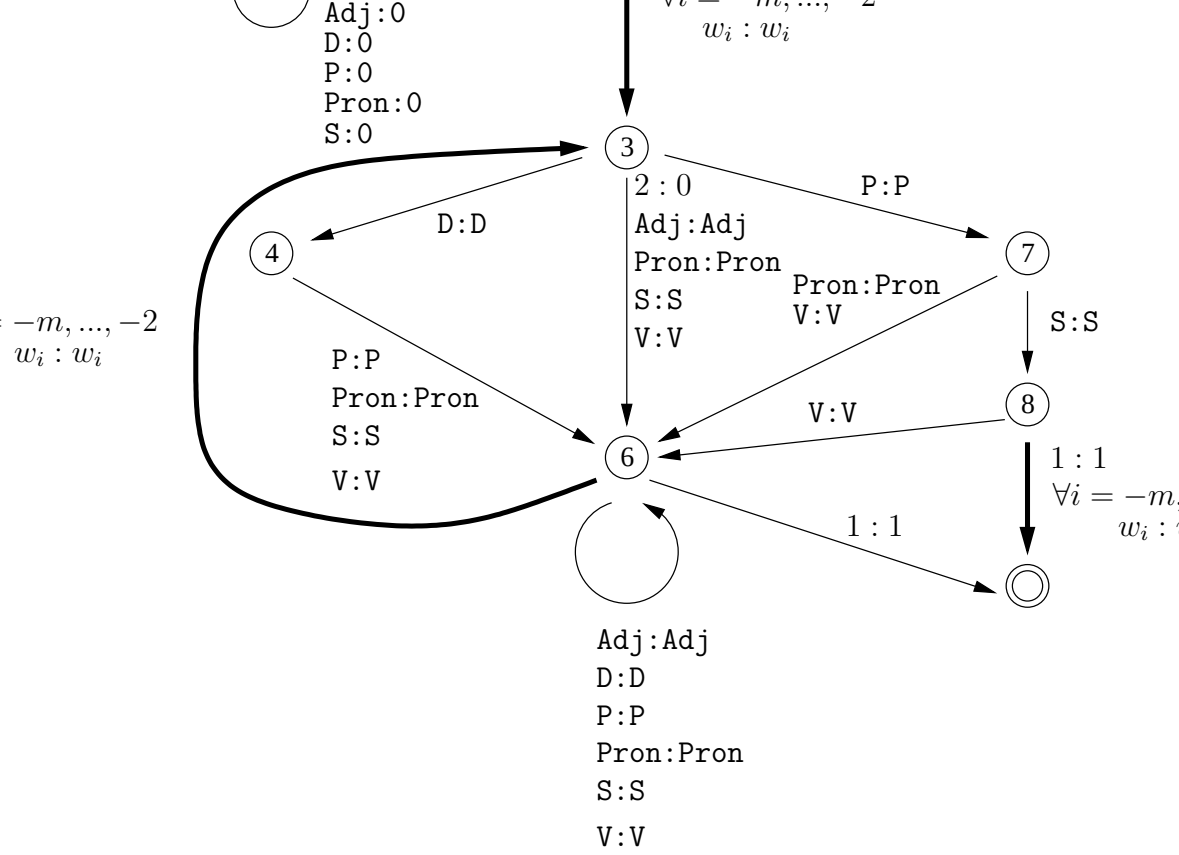


Figura 6.20: *FST* para: `SELECCIONA (V) (+* ES {S,P} <BARRERA> ES {D});`

palabra será un *verbo*. El traductor equivalente se muestra en la figura 6.20.

□

Modo de operación de los traductores de estado finito

dos a este punto, donde ya tenemos los *FST*,s para cada regla, nos queda por crear npilar cada traductor por separado para después combinarlos de manera que al final gamos un único traductor que simula el comportamiento de aplicar todas las reglas de secuencial. Hasta ahora hemos trabajado con las etiquetas y las palabras involucradas reglas con el propósito de facilitar la lectura del documento, sin embargo realmente los ctores van a estar formados por los símbolos que resultan del *mapping* de las reglas, y no palabras y etiquetas en sí, tal y como se introdujo en el apartado 6.2.1. A continuación los los se detallarán de las dos maneras para facilitar su comprensión, pero debe quedar claro n la realidad sólo se construirán los traductores correspondientes utilizando el *mapping* en

con transiciones del traductor a representar, de manera que tendremos tantas líneas en el fiche como transiciones existentes en el traductor. En cada línea se especificarán los siguientes campo estado origen, estado destino, símbolo de entrada y símbolo de salida, todos ellos separados p un espacio en blanco. El estado origen será el primero en aparecer, mientras que el esta destino se especifica al final del fichero. Los símbolos de las transiciones, se van a correspond con números negativos o positivos según hablemos de palabras o etiquetas, y se corresponder con el *mapping* de las palabras y etiquetas que aparecen en el mini-diccionario y en el mini-ta set, respectivamente, según se indicó en el apartado 6.2.1.

Además de los traductores también necesitamos especificar el alfabeto con el que vam a trabajar, y para ello es necesario crear un fichero de símbolos donde se especificarán tod aquellos elementos que participan en las transiciones de los traductores y que hemos referencia en los ficheros de texto. De esta manera el fichero de símbolos tendrá tantas líneas co elementos existan en el mini-diccionario más en el mini-tag-set más los símbolos especiales − −1, 0, 1 y 2. En cada línea se especificarán un número positivo y un símbolo, ambos separad por un espacio en blanco, donde este número positivo será la representación interna del símbo Por último indicar que a cada uno de los ficheros de texto que representen a los traductor se le pondrá extensión `.stxt` y al fichero de símbolos que representa al alfabeto se le pond extensión `.syms`.

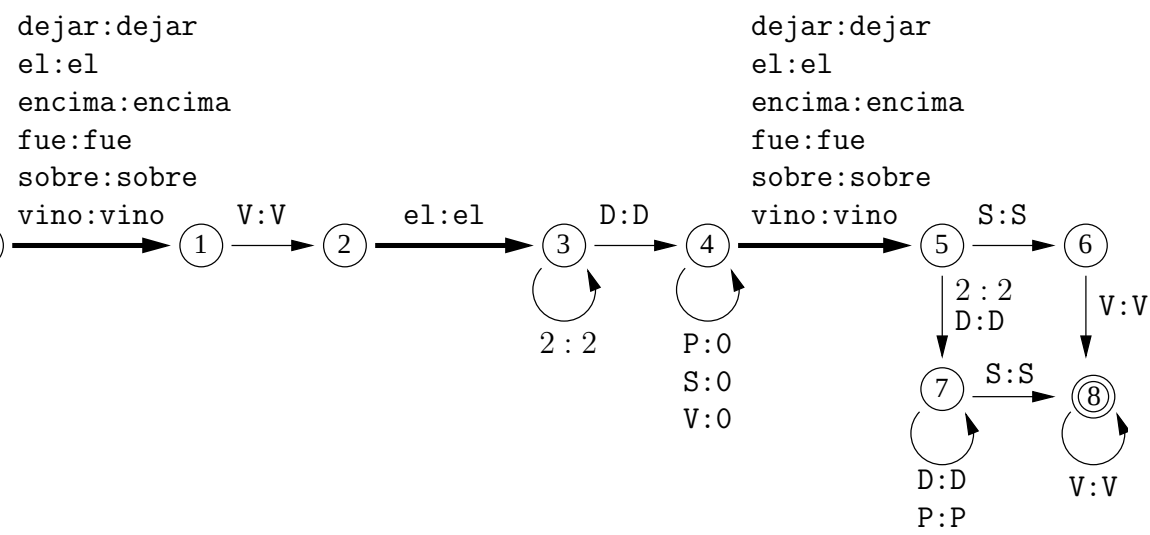
Ejemplo 6.12 Para reflejar todas estas ideas retomemos el ejemplo 6.2 tratado en apartado 6.2.1, donde nos encontrábamos con cuatro reglas. Para simplificar el proble supongamos que sólo tenemos la primera regla, siguiendo tal cual el mini-diccionario y el mi tag-set:

```
SELECCIONA (D) (−1 ES {V}, 0 PERTENECE {el}, 1 CONTIENE {S});
```

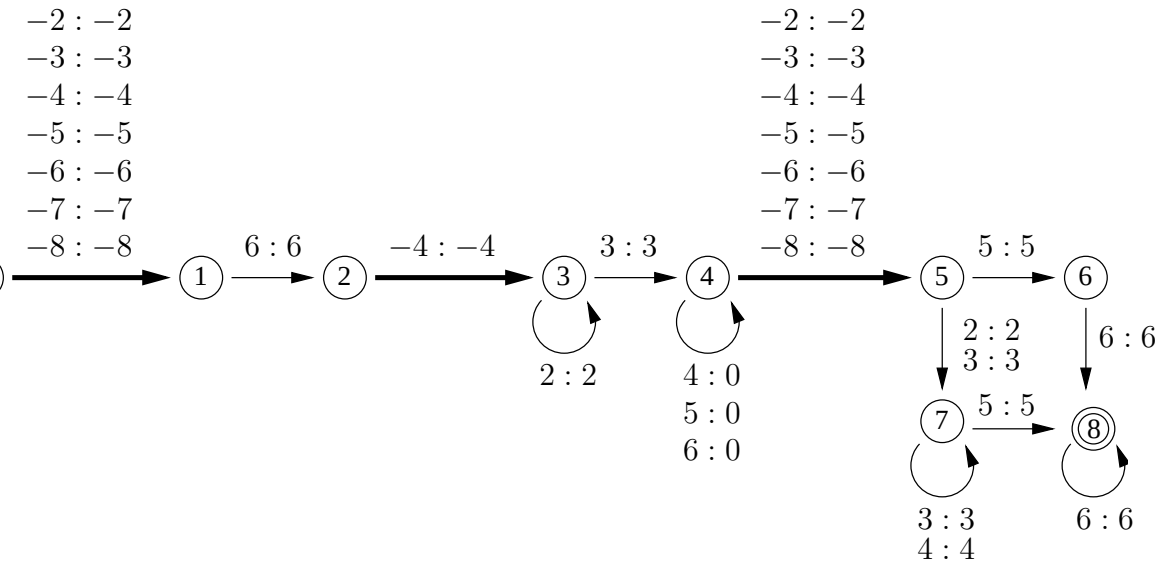
Mini-diccionario		**Mini-tag-set**	
dejar	−3	D	3
el	−4	P	4
encima	−5	S	5
fue	−6	V	6
sobre	−7		
vino	−8		

El traductor de estado finito correspondiente es el que se muestra en la figura 6.21. Utilizan los símbolos que resultan del *mapping* de las palabras y de las etiquetas, el traductor resultante el mismo cambiando únicamente la palabras por números negativos y las etiquetas por núme positivos, tal y como se indica en el mini-diccioario y en el mini-tag-set respectivamente. Es traductor se muestra en la figura 6.22.

A partir de este último traductor, el de la figura 6.22, creamos el fichero de texto que representa. Y a partir del mini-diccionario y del mini-tag-set se crea el fichero de símbol



a 6.21: *FST* de la regla: `SELECCIONA (D) (−1 ES {V}, 0 PERTENECE {el}, TIENE {S});`



a 6.22: *FST* mapeado de la regla: `SELECCIONA (D) (−1 ES {V}, 0 PERTENECE {el}, TIENE {S});`

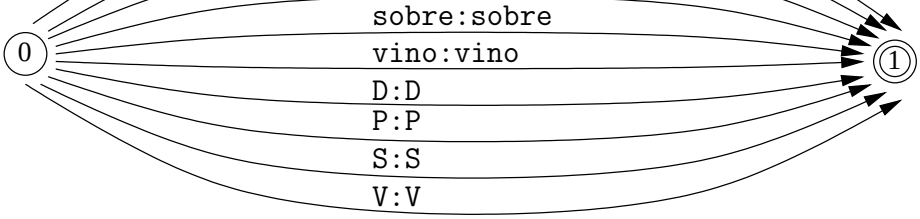


Figura 6.23: *FST* del alfabeto

traductorEjemplo.stxt				alfabeto.syms	
0	1	−3	−3	0	0
0	1	−4	−4	−11	1
0	1	−5	−5	−3	2
0	1	−6	−6	−4	3
0	1	−7	−7	−5	4
0	1	−8	−8	−6	5
1	2	6	6	−7	6
2	3	−4	−4	−8	7
3	4	3	3	1	8
4	4	4	0	3	9
4	4	5	0	4	10
4	4	6	0	5	11
4	5	−3	−3	6	12
4	5	−4	−4		
4	5	−5	−5		
4	5	−6	−6		
4	5	−7	−7		
4	5	−8	−8		
5	6	5	5		
5	7	3	3		
7	7	4	4		
7	8	6	6		
8					

Para el alfabeto también es necesario construir un traductor de estado finito, y éste es el q
se muestra en la figura 6.23. Su traductor real equivalente utilizando los símbolos que result
del *mapping* de las palabras y de las etiquetas es el que se indica en la figura 6.24. De ac
vale la pena comentar que además de los símbolos que participan en las reglas también tenem
que tener en cuenta los símbolos de principio y fin de frase, -1 y 1 respectivamente, los símbol
comodines,-2 para las palabras y 2 para las etiquetas, y el símbolo que representa a *epsilon*, q
como habíamos visto es el 0. El fichero de texto correspondiente se crearía de la misma manei

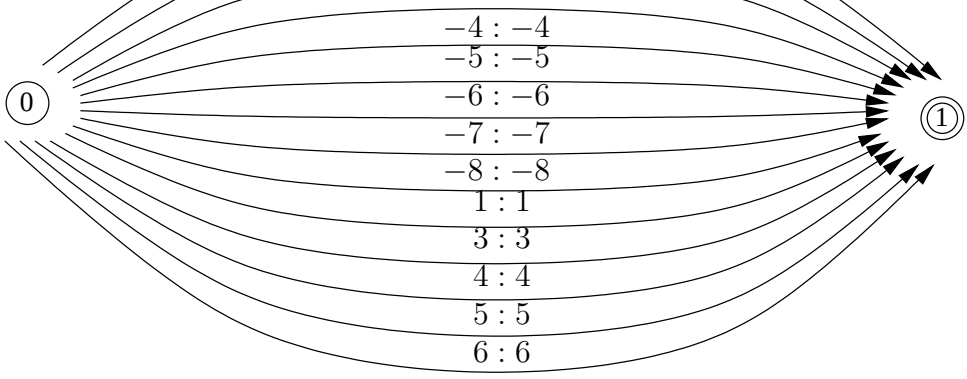


Figura 6.24: *FST* mapeado del alfabeto

Compilación de los traductores de estado finito

vez que tenemos todos los ficheros creados, tanto de traductores como del alfabeto, el nte paso es compilarlo todo.
ro compilamos el alfabeto, y para ello ejecutamos el siguiente comando:

`compila` *nombreAlfabeto*`.stxt`

iendo como salida varios ficheros entre los que se encuentra `alfabeto.fst`, que es el ctor del alfabeto compilado.
tinuación compilamos cada uno de los traductores:

`compila` *nombreTraductor*`.stxt` *nombreTraductorCompilado*

enemos como salida un fichero con extensión `.fst` que contiene el traductor original lado y normalizado. Si a este traductor se le aplica una entrada, obtenemos como salida ma entrada con algunos cambios, que se corresponden con la funcionalidad del traductor.

vez que tenemos todos los traductores compilados y con su lenguaje normalizado, ya nos empezar a trabajar con ellos. La forma en la que podemos trabajar es con cada or separado, pero lo que nos interesa es construir un único traductor que involucre a todos más. Después de analizar las diferentes operaciones que nos ofrece la teoría de traductores nos a la conclusión de que la operación que buscamos es la de composición. Debemos oner todos los traductores de manera que al final obtengamos un único traductor que roporciona como salida, la misma que obtendríamos al aplicar todos los traductores por ado uno detrás de otro. Para esto lo que hacemos es lo siguiente:

`componer` $traductor_1$`.fst` $traductor_2$`.fst ...`

enemos como salida el fichero `composicion.fst` que recoge el traductor compuesto lado y normalizado, disponible para ser utilizado en cualquier momento.

os con un ejemplo muy sencillo que realmente es la operación de composición la operación uscamos.

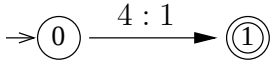


Figura 6.26: Traductor de estado finito T_2

Ejemplo 6.13 Supongamos que tenemos los traductores:

- T_1: que lo que hace es cambiar el número 3 por el número 4 si antes aparece la secuenc de números 1 y 2,

- T_2: que simplemente lo que hace es cambiar siempre el 4 por el 1.

Los traductores T_1 y T_2 se muestran en las figuras 6.25 y 6.26 respectivamente. continuación vamos a llevar a cabo la operación de composición de ambos traductores, de mane que en primer lugar compondremos T_1 con T_2, dando como resultado el traductor $T_2 \circ T_1$ q se muestra en la figura 6.27 que lo que hace es aplicar primero el traductor T_1 y después traductor T_2.
Veamos qué ocurre si introducimos al traductor $T_2 \circ T_1$ la entrada $1, 2, 3, 4, 5$. Obtenemos cor salida la secuencia $1, 2, 1, 1, 5$ con lo cual podemos ver claramente que lo que hace el traduct es lo que realmente buscamos, aplicar primero el traductor T_1 que daría como salida $1, 2, 4, 4$ para aplicar después el traductor T_2 que daría como salida $1, 2, 1, 1, 5$ que es lo que esperábam

La siguiente pregunta que nos podemos hacer es si la composición de T_2 con T_1 nos da o no mismo resultado. Veámoslo a continuación siendo el traductor de estado finito de la figura 6. el que resulta de la composición de T_2 con T_1. Si le aplicamos la misma entrada que al ca anterior, obtendríamos como salida $1, 2, 4, 1, 5$ que como se puede ver claramente es diferen a la salida anterior. De esto deducimos que la composición es dependiente del orden, fact importante que se debe tener en cuenta a la hora de componer los traductores que represent a las reglas.

6.3.3 Funcionamiento de los traductores de estado finito

En este momento ya estamos en condiciones de poder aplicar una entrada al traduct compuesto, de manera que a la salida obtengamos la cadena que resulta de aplicar todas l reglas por separado, con la diferencia de que aquí realizamos la operación en un único pas con todas las ventajas que ello conlleva, tal y como vimos al principio de este tema. Para e debemos seguir los siguientes pasos:

- Construir la entrada: para ello construimos el traductor que representa la cadena entrada y creamos su fichero de texto correspondiente.

- Pasamos la cadena de entrada sobre el traductor compuesto:

  `traduce` *nombreEntrada*`.stxt composicion.fst` *nombreSalida*

- Obtenemos como salida un fichero de texto que contiene la entrada traducida

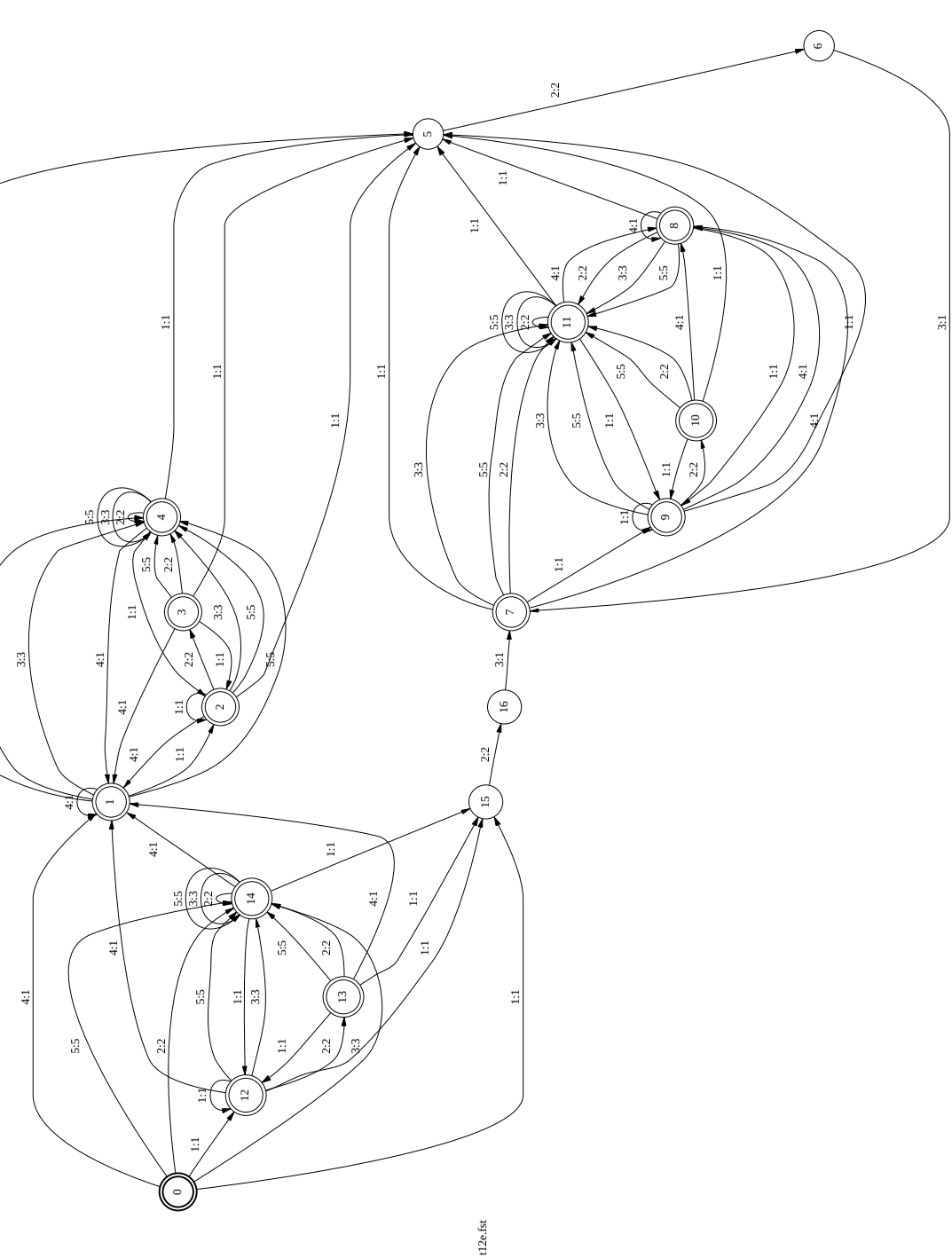


Figura 6.27: Traductor de estado finito $T_2 \circ T_1$

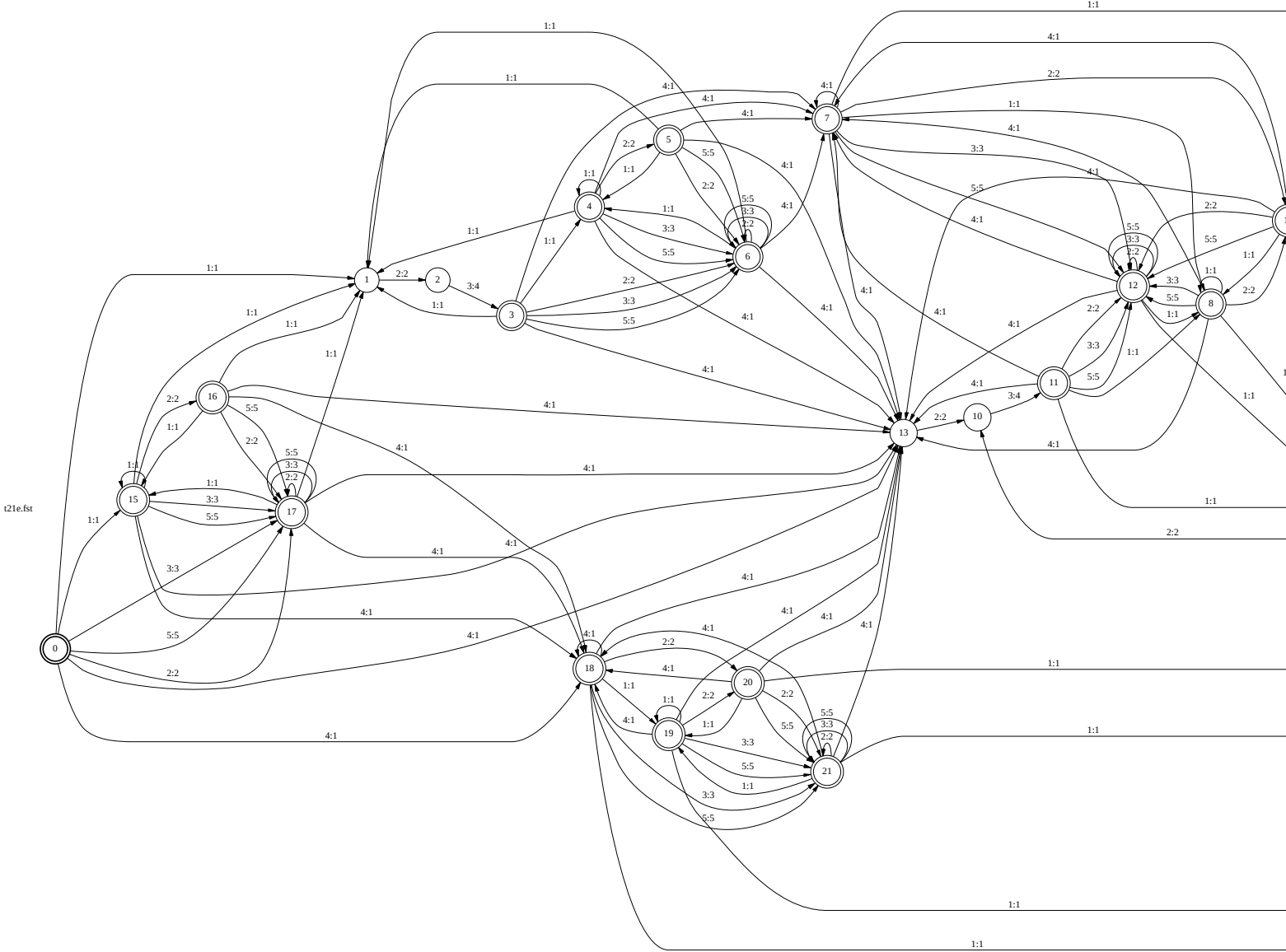


Figura 6.28: Traductor de estado finito $T_1 \circ T_2$

al-purpose finite-state machine software tools implementada por Mehryar Mohri, ndo C. N. Pereira y Michael D. Riley de los laboratorios $AT\&T$ [Mohri 1995, Mohri 1997]. amente lo que aporta esta librería son un conjunto de operaciones sobre autómatas y ctores, que nosotros hemos aprovechado con la intención de conseguir nuestros objetivos. ta manera los comandos introducidos en el apartado anterior son una serie de *scripts* que n a operaciones de esta librería.
os cada uno de estos comandos por separado y de forma más detallada, para estudiar lo ealmente hacen a bajo nivel:

Compilación:
Este comando lo que hace es, en primer lugar, convertir el fichero de texto en la estructura de datos interna utilizada por la librería para poder llevar a cabo las operaciones pertinentes. Para ello se ejecuta el comando:

`fsmcompile -i`$nombreAlfabeto$`.syms -o`$nombreAlfabeto$`.syms`
`-t <`$nombreTraductor$`.stxt>` $nombreTraductorCompilado$`.fst`

A continuación, lo que hace es normalizar el lenguaje del traductor siguiendo los siguientes pasos:
Supongamos que T sea el traductor sobre el que queremos operar y Σ el alfabeto con el que vamos a trabajar, ambos ficheros con extensión `.fst`, entonces:

Paso 1) $a = p_1(T)$
Paso 2) $\Gamma = \Sigma^* - \Sigma^* a \Sigma^*$
Paso 3) $N = \Gamma(T\Gamma)^*$

Desglosemos un poco más en qué consiste cada uno de estos pasos:

- Paso 1: Simplemente hacemos la primera proyección del traductor T con el propósito de quedarnos con los símbolos de entrada de dicho traductor.
  Para ello utilizamos la operación:

  `fsmproject -i T.fst > a.fst`

- Paso 2: Este paso lleva implícito distintas operaciones:
  - *Kleene star* del alfabeto, para permitir que aparezca más de un símbolo:

    `fsmclosure` Σ`.fst >` Σ^*`.fst`
  - Concatenación de Σ^* con a:

    `fsmconcat` Σ^*`.fst a.fst` Σ^*`.fst >` $\Sigma^* a \Sigma^*$`.fst`
  - Diferencia de Σ^* y $\Sigma^* a \Sigma^*$: esta operación únicamente puede llevarse a cabo con autómatas, y nosotros trabajamos con traductores, con lo cual para poder llevarla a cabo primero debemos hacer la primera proyección de ambos sustraendos:

    `fsmproject -i` Σ^*`.fst >` $p_1(\Sigma^*)$`.fsa`[5]
    `fsmproject -i` $\Sigma^* a \Sigma^*$`.fst >` $p_1(\Sigma^* a \Sigma^*)$`.fsa`
    `fsmdifference` $p_1(\Sigma^*)$`.fsa` $p_1(\Sigma^* a \Sigma^*)$`.fsa >` Γ`.fsa`

- Paso 3: Este paso también lleva varias operaciones implícitas, que veremos a continuación:

`fsmclosure T Γ.fst > (TΓ)*.fst`

* Concatenación de Γ con el cierre hecho en el paso anterior:

  `fsmconcat Γ.fsa (TΓ)*.fst > N.fst`

Una vez realizados todos estos pasos el traductor de estado finito resultante, compilado normalizado, está ya en condiciones de operar.

- **Concatenación:**
  Este es el siguiente comando a utilizar una vez compilado todo, y no hace otra co más que la operación de composición, cuya definición formal ha sido introducida en primer apartado del tema. Únicamente debemos destacar de nuevo que esta operaci es dependiente del orden, lo que hay que tener en cuenta antes de llevarla a cabo, pa obtener como salida lo que realmente queremos:

  `fsmcompose` *traductor*$_1$`.fst` *traductor*$_2$`.fst ... > composicion.fst`

- **Traducción:**
  Este es el comando que pone en funcionamiento al traductor. A partir de una entra nos devuelve una salida con los cambios que vienen determinados por la operación d traductor. Para ello utilizamos la operación de composición, de manera que si la cade de entrada pertenece al lenguaje del traductor, obtenemos como salida un traductor con cadena de entrada como símbolos de entrada y con la cadena de salida como símbolos salida. Para quedarnos únicamente con la cadena de salida hacemos la segunda proyecci sobre el traductor obteniendo así el autómata correspondiente que representa a la cade de salida. Si la cadena de entrada no pertenece al lenguaje del traductor, la salida es vac También debemos recordar que la cadena de entrada debe transformarse en un traduct en el que no se produce ninguna traducción, sino que la entrada se mantiene a la salida Indicar que antes de nada lo que se hace es compilar el traductor que representa a cadena de entrada.

  `fsmcompile -i`*nombreAlfabeto*`.syms -o`*nombreAlfabeto*`.syms -t`
  nombreEntrada`.stxt > `*nombreEntrada*`.fst`
  `fsmcompose` *entrada*`.fst composicion.fst > `*salida*`.fst`
  `fsmproject -o` *salida*`.fst >` $p_2($*salida*$)$`.fsa`
  `fsmrmepsilon` *salida*`.fst >` $p_2($*salida*$) - sin - \epsilon$`.fsa`
  `fsmprint -i`*nombreAlfabeto*`.syms -o`*nombreAlfabeto*`.syms`
  $p_2($*salida*$) - sin - \epsilon$`.fsat >` *salida*`.stxt`

6.5 Un ejemplo completo

A continuación, con el propósito de verificar el funcionamiento de todo lo indicado en l apartados anteriores, nos disponemos a mostrar un ejemplo de ejecución completo y detalla paso a paso. Se trata de un prototipo que nos va a mostrar que realmente todas las ideas q hemos desarrollado se pueden aplicar en la práctica[6].

`SELECCIONA (S) (1 ES {V});`
Esta regla nos dice que la etiqueta de la palabra actual será un *sustantivo* si la palabra que le sigue es un *verbo*.

`BORRA (V) (-2 CONTIENE {P,V});`
Esta regla dice que si la segunda palabra anterior a la actual tiene entre sus etiquetas posibles a una *preposición* y a un *verbo*, entonces la palabra actual no podrá ser *verbo*.

`SELECCIONA (P) (-2 PERTENECE {bajo,sobre}, -1 ES {V});`
Esta regla nos dice que la palabra actual será una *preposición* si la palabra anterior es un *verbo* y la segunda palabra anterior es `bajo` o `sobre`.

`FUERZA (D) (1 PERTENECE sobre, NOT 1 ES {P});`
Esta regla dice que la palabra actual será forzosamente un *determinante* si la siguiente palabra es `sobre` y no es una *preposición*.

na vez que tenemos las reglas el siguiente paso sería crear el mini-diccionario y el mini-t que recogen todas las palabras y las etiquetas involucradas en las reglas. Recordamos anto el mini-diccionario como el mini-tag-set también recogen los *mapping* de cada una palabras con números negativos y de cada una de las etiquetas con números positivos, ando el -1 y el 1 para inicio y fin de frase respectivamente, además del -2 y 2 como dín de palabras y etiquetas respectivamente. De esta manera nuestro mini-diccionario y ro mini-tag-set sería el siguiente:

Mini-diccionario		**Mini-tag-set**	
bajo	-3	D	3
sobre	-4	P	4
		S	5
		V	6

tinuación tenemos que construir los traductores de estado finito que representan a las . Veamos entonces cuál es el traductor que se corresponde con cada una de las reglas ores:

`SELECCIONA (S) (1 ES {V});`
Para esta primera regla, el traductor de estado finito resultante se muestra en la figura 6.29 y de él destacamos que nos quedamos con la etiqueta a seleccionar dejando las demás a *epsilon* si la palabra que aparece a continuación tiene únicamente a *verbo* como posible etiqueta. El traductor que resulta de hacer el *mapping* es el que se muestra en la figura 6.30,

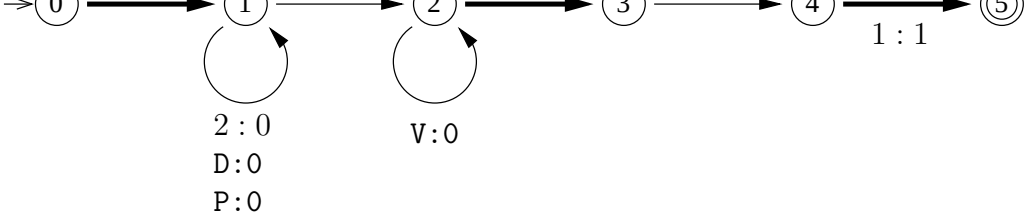


Figura 6.29: *FST* para la regla: `SELECCIONA (S) (1 ES {V});`

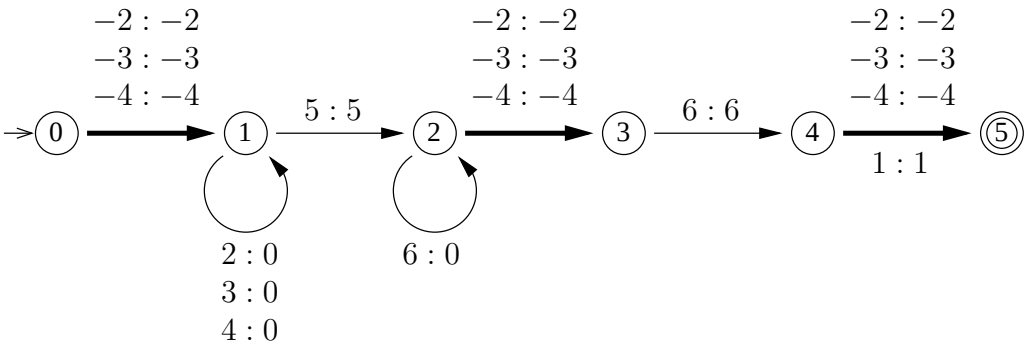


Figura 6.30: *FST* mapeado para la regla: `SELECCIONA (S) (1 ES {V});`

Estado origen	Estado destino	Símbolo entrada	Símbolo salida
0	1	−2	−2
0	1	−3	−3
0	1	−4	−4
1	1	2	0
1	1	3	0
1	1	4	0
1	2	5	5
2	2	6	0
2	3	−2	−2
2	3	−3	−3
2	3	−4	−4
3	4	6	6
4	5	−2	−2
4	5	−3	−3
4	5	−4	−4
4	5	1	1
5			

2. `BORRA (V) (-2 CONTIENE {P,V});`
   Para la segunda regla, los traductores de estado finito sin y con *mapping* son los que
   muestran en las figuras 6.31 y 6.32 respectivamente, y de estos traductores destacam
   que la etiqueta `V` de la palabra actual la transformamos en *epsilon* si la segunda palab

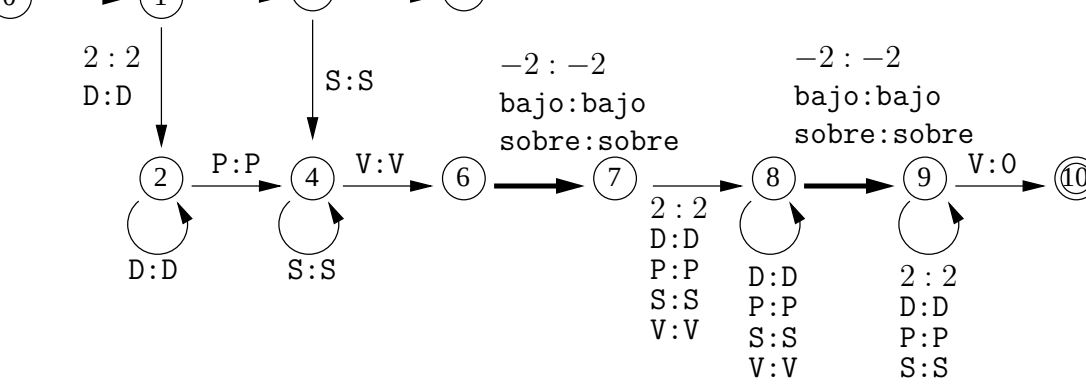


Figura 6.31: *FST* para la regla: `BORRA (V) (-2 CONTIENE {P,V});`

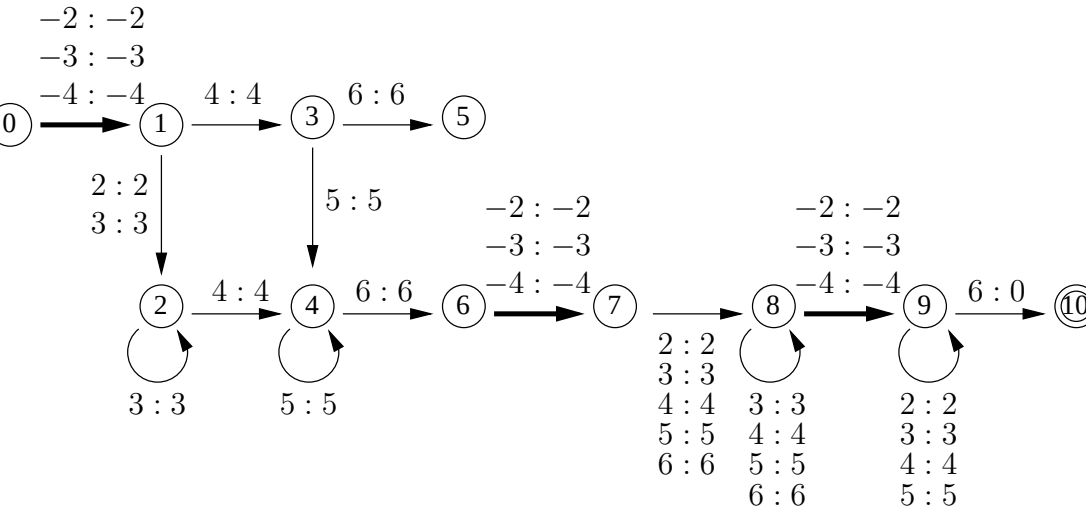


Figura 6.32: *FST* mapeado para la regla: `BORRA (V) (-2 CONTIENE {P,V});`

El fichero de texto se construiría de la misma forma que en el caso anterior.

`SELECCIONA (P) (-2 PERTENECE {bajo,sobre}, -1 ES {V});`
Para la tercera regla, el traductor de estado finito que la representa es el de la figura 6.33, y de este traductor destacamos que de la palabra actual seleccionamos *preposición* dejando el resto de etiquetas a *epsilon* si la palabra anterior es exactamente *verbo* y la segunda palabra anterior es o bien `bajo`, o bien `sobre`. El traductor que resulta de hacer el *mapping* es el que se muestra en la figura 6.34. El fichero de texto se construiría de la misma forma que para la primera regla.

`FUERZA (D) (1 PERTENECE sobre, NOT 1 ES {P});`
Y para la cuarta y última regla, los traductores son los mostrados en las figuras 6.35 y 6.36. De estos traductores destacamos que de *epsilon* traducimos a *determinante* dejando el resto de etiquetas a *epsilon*, ya que hablamos de un `FUERZA`, si la siguiente palabra es `sobre` y no tiene exactamente como etiqueta *preposición*. El fichero de texto se construiría de la

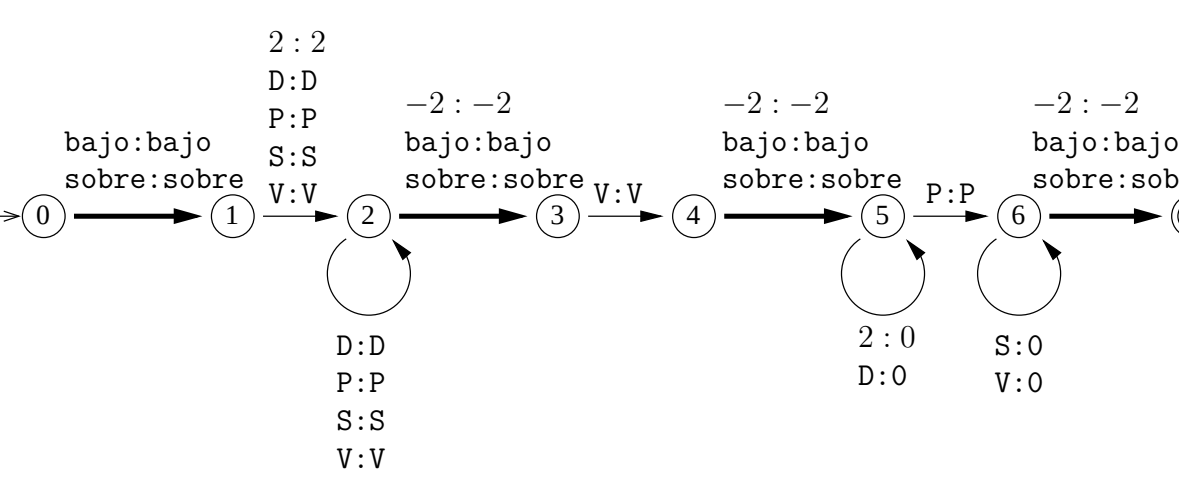


Figura 6.33: *FST* para la regla: `SELECCIONA (P) (-2 PERTENECE {bajo,sobre}, -1 ES {V});`

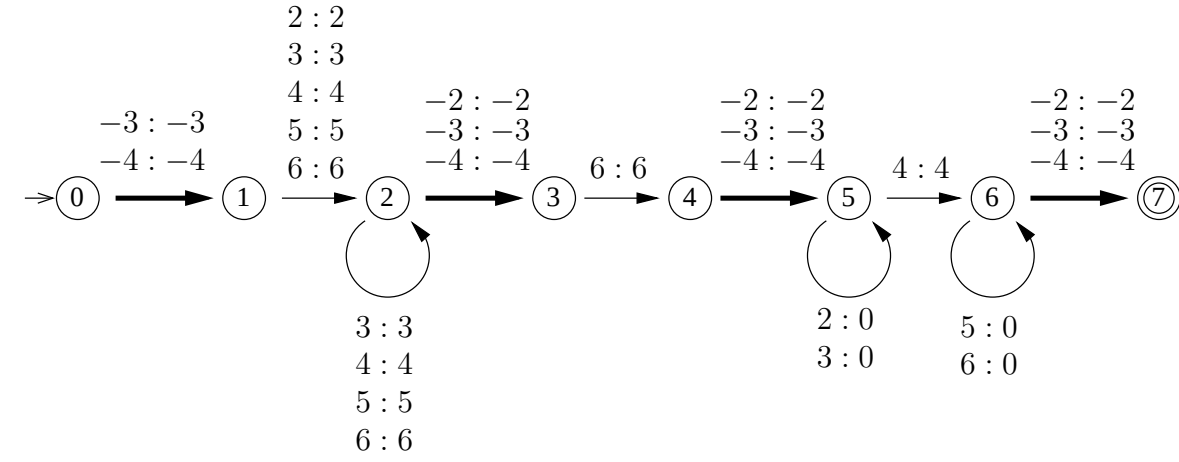
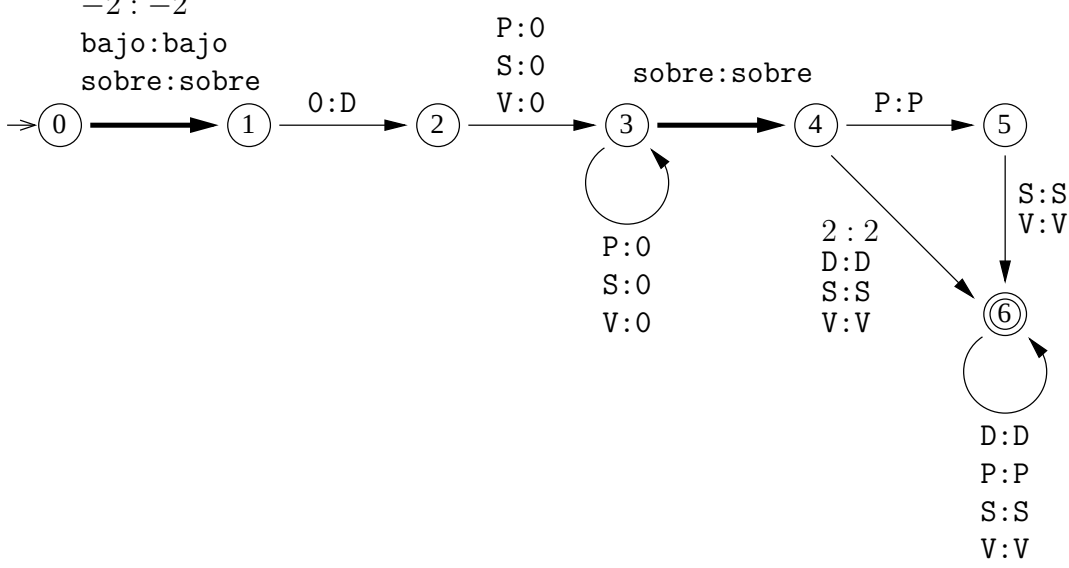
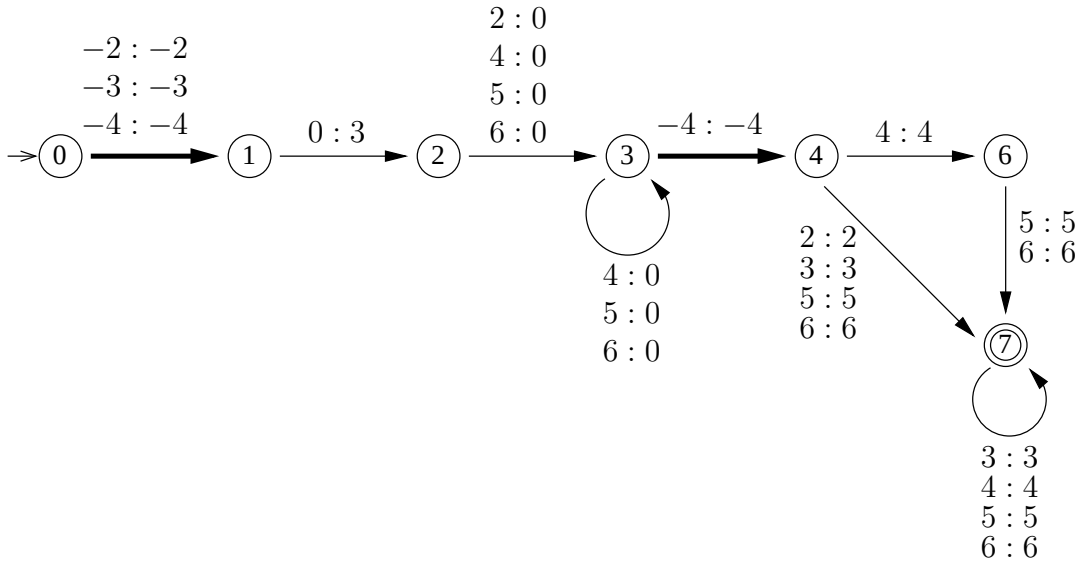


Figura 6.34: *FST* mapeado para la regla: `SELECCIONA (P) (-2 PERTENECE {bajo,sobre -1 ES {V});`



igura 6.35: *FST* para la regla: `FUERZA (D) (1 PERTENECE sobre, NOT 1 ES {P});`



a 6.36: *FST* mapeado para la regla: `FUERZA (D) (1 PERTENECE {sobre}, NOT 1 ES`

estamos en condiciones de poder aplicar una entrada al traductor compuesto. Para este ejemp
la forma de actuar sería la siguiente:

1. Compilación del alfabeto:

```
compilaAlfabeto alfabeto.stxt
```

2. Compilación de los traductores de las reglas:

```
compila regla1.stxt n1

compila regla2.stxt n2

compila regla3.stxt n3

compila regla4.stxt n4
```

3. Composición de los traductores compilados y normalizados:

```
componer n1.fst n2.fst n3.fst n4.fst
```

4. Y ejecución del traductor compilado

```
traduce entrada.stxt composicion.fst salida
```

Para la ejecución del traductor compuesto necesitamos una entrada, y para nuestro ejemp
hemos elegido la siguiente:

El	sobre	está	sobre	la	mesa
Pron	P	V	P	D	S
	S		S	Pron	V
	V		V		

Utilizando el mapeo, la entrada es:

-2	-4	-2	-4	-2	-2
2	4	6	4	2	5
	5		5	3	6
	6		6		

Para poder aplicar la entrada al sistema es necesario crear el traductor de estado finito q
representa este enrejado, y este traductor es el que se muestra en la figura 6.37.

Al aplicar la entrada al traductor, como salida obtenemos el traductor que se muestra en
figura 6.38, y sus enrejados, mapeado y sin mapear, correspondientes son los que se indican
continuación:

-2	-4	-2	-4	-2	-2
3	5	6	4	2	5
				3	

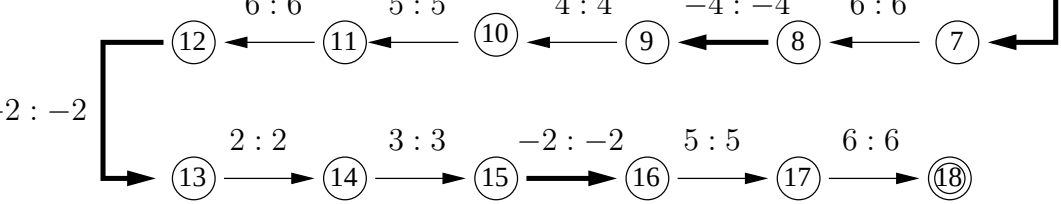


Figura 6.37: *FST* de la entrada

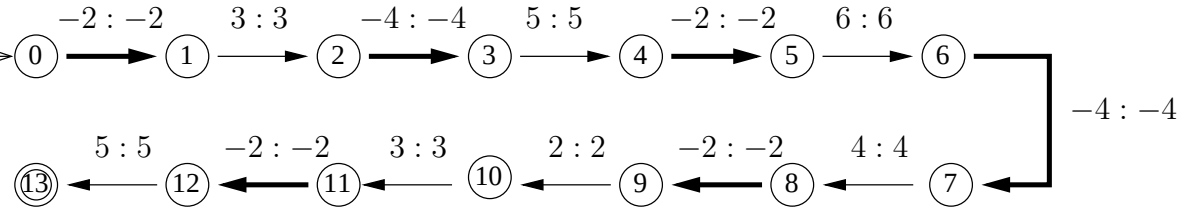


Figura 6.38: *FST* de la salida

El	sobre	está	sobre	la	mesa
D	S	V	P	D	S
				Pron	

omo podemos ver, el traductor ha llevado a cabo algunos cambios sobre la entrada, pero odemos preguntar si estos cambios son realmente los buenos. Para poder corroborar que on, veamos cómo afecta cada una de las reglas a la entrada:

	El	sobre	está	sobre	la	mesa
Enrejado inicial.	Pron	P S V	V	P S V	D Pron	S V
Primera regla encajada y ejecutada.	El Pron	sobre S	está V	sobre P S V	la D Pron	mesa S V
Segunda regla encajada y ejecutada.	El Pron	sobre S	está V	sobre P S V	la D Pron	mesa S
Tercera regla encajada y ejecutada.	El Pron	sobre S	está V	sobre P	la D Pron	mesa S

Efectivamente, vemos que la ejecución del traductor resultante de componer todas las regl
contextuales produce la misma salida que la ejecución individual de cada regla por separad
Por tanto, a través de este caso de estudio, hemos ilustrado de manera práctica las ventaj
de la compilación de reglas contextuales en estructuras más eficientes como son los *FST*,s.
única desventaja que hemos podido detectar ha sido quizás que el tamaño de los traductor
compilados y normalizados no es tan compacto como cabría esperar en un principio. Esto pue
deberse a la propia naturaleza de los traductores, o al coste estructural de la operación
composición, al menos en esta herramienta. El capítulo de conclusiones finales incluye algun
reflexiones más detalladas sobre todos estos aspectos.

Parte IV

Formalización del proceso de segmentación

Capítulo 7

Extensiones del algoritmo de Viterbi

Los diferentes tipos de etiquetadores que existen actualmente asumen que el texto de entra aparece ya correctamente segmentado, es decir, dividido de manera adecuada en *tokens* unidades de información de alto nivel de significado, que identifican perfectamente cada u de los componentes de dicho texto. Esta hipótesis de trabajo no es en absoluto realista debido la naturaleza heterogénea tanto de los textos de aplicación como de las fuentes donde se origina

Así pues, algunas lenguas, como el gallego [Alvarez, Regueira y Montegudo 1986] o español, presentan fenómenos que es necesario tratar antes de realizar la etiquetación. Ent otras tareas, el proceso de segmentación se encarga de identificar unidades de información ta como las frases o las propias palabras. Esta operación puede ser más compleja de lo que pare *a priori*. Por ejemplo, la identificación de las frases se suele realizar considerando ciertas marc de puntuación. Sin embargo, un simple punto puede ser indicativo de fin de frase, pero podr corresponder también al carácter final de una abreviatura.

En el caso de las palabras, la problemática se centra en que el concepto ortográfico de palab no siempre coincide con el concepto lingüístico. Se presentan entonces dos opciones:

1. Las aproximaciones más sencillas consideran igualmente las palabras ortográficas y amplí las etiquetas para representar aquellos fenómenos que sean relevantes. Por ejemplo, palabra `reconocerse` podría etiquetarse como `V000f0PE1`[1] aun cuando está formada p un verbo y un pronombre enclítico, y las palabras de la locución preposicional `a pes de` se etiquetarían respectivamente como `P31`, `P32` y `P33` aun cuando constituyen un úni término. Sin embargo, en idiomas como el gallego, este planteamiento no es viable, ya q su gran complejidad morfológica produciría un crecimiento excesivo del juego de etiqueta Esto complica, entre otras cosas, la creación de los recursos lingüísticos, como por ejemp textos etiquetados, que son necesarios para ajustar los parámetros de funcionamiento los etiquetadores.

2. La solución pasa entonces por no ampliar el juego de etiquetas básico. Como ventaja la complejidad del proceso de etiquetación no se verá afectada por un número elevado etiquetas, la creación de recursos lingüísticos será más sencilla y la información relativa cada término lingüístico se puede expresar de manera más precisa. Por ejemplo, a lo q antes era un simple pronombre enclítico se le pueden atribuir ahora valores de perso número, caso, etc. Como desventaja, se complican las labores del preprocesador, que sólo se verá obligado a identificar las palabras ortográficas, sino que unas veces tendrá q partir una palabra en varias, y otras veces tendrá que juntar varias palabras en una sol

[1] Las etiquetas que aparecen en este capítulo pertenecen a los *tag sets* utilizados en los proyectos GALE (Generador de Analizadores para LEnguajes NAturales) y CORGA (COrpus de Referencia do Galego Actual)

del verbo `tener` con dos pronombres enclíticos, o bien una forma del verbo `tensar` con un solo pronombre. Este fenómeno es muy común en gallego, no sólo con los pronombres enclíticos, sino también con algunas contracciones. Por ejemplo, la palabra `polo` puede ser un sustantivo (en español, pollo), o bien la contracción de la preposición `por` (por) y del artículo `o` (el), o incluso la forma verbal `pos` (pones) con el pronombre enclítico `o` (lo).

te trabajo hemos adoptado la segunda opción, es decir, la de separar y unir (separar, por lo, el verbo de sus pronombres, y unir, por ejemplo, los diferentes constituyentes de una ón). En cualquier caso, la primera opción, la de trabajar al nivel de la palabra ortográfica, taría, después de la fase de etiquetación, una fase de post-procesamiento, cuando se desee ficar los diferentes componentes sintácticos del texto. Dicha fase de post-procesamiento aría las labores análogas a las que involucra nuestro preprocesador.

ra dar una idea de la complejidad de los problemas que se afrontan, vamos a poner algunos típicos que es capaz de resolver el preprocesador:

plo 7.1 Supongamos que tenemos la expresión `polo tanto` (por lo tanto o por el tanto). te caso, estamos ante una locución insegura, es decir, `polo tanto` puede ser una locución, a su vez puede ser un sustantivo, una contracción o un verbo con pronombres enclíticos, `to`, por su parte, puede ser sustantivo o adverbio, si no forma parte de la locución. La entación sería la siguiente[2]:

```
rnativa>
rnativa1>
[Scms polo]

ernativa1>
rnativa2>
[P por]
[Ddms o]

ernativa2>
rnativa3>
[Vpi2s0 pór] [Vpi2s0 poñer]
[Raa3ms o]

ernativa3>
rnativa4>
o&tanto
ernativa4>
ernativa>
```

emplo de aplicación de las 4 diferentes acepciones sería:

Sustantivo+Adverbio: `Coméche-lo polo tanto, que non quedaron nin os osos` (comiste el pollo tanto, que no quedaron ni los huesos).

ta es la salida que nos da el preprocesador y refleja todas las segmentaciones posibles con las que nos

pones tanto tú como él).

- **Locución**: `Estou enfermo, polo tanto quédome na casa` (estoy enfermo, por lo tan me quedo en casa).

Ejemplo 7.2 Un ejemplo de conflicto entre dos posibles descomposiciones de enclíticos aplicab al español sería `ténselo`, que puede ser `tense` (de tensar) más `lo` ó `ten` (de tener) más `se` m `lo`, por lo que la representación correspondiente es:

```
<alternativa>
<alternativa1>
ténse [V2spm0 tensar]
+lo    [Re3sam el]
</alternativa1>
<alternativa2>
tén   [V2spm0 tener]
+se    [Re3yyy se]
+lo    [Re3sam el]
</alternativa2>
</alternativa>
```

7.1 El nuevo etiquetador

Una vez que tengamos la salida del preprocesador, y debido a las segmentaciones ambiguas q hemos descrito anteriormente, el etiquetador debe ser capaz de enfrentarse a flujos de *tokens* distinta longitud. Es decir, no sólo debe ser capaz de decidir qué etiqueta asignar a cada *toke* sino que además debe saber decidir si algunos de ellos constituyen o no una misma entid y asignar, en cada caso, la cantidad de etiquetas adecuada, en función de las alternativas segmentación que le proporciona el preprocesador.

Para llevar a cabo este proceso, se podría considerar la construcción de etiquetador especializados para todas las posibles alternativas, la posterior comparación de s correspondientes salidas, y la selección de la más plausible. No obstante, esta posibilid plantea varios inconvenientes. En primer lugar, sería necesario definir algún criterio objeti de comparación. Si el paradigma de etiquetación que se está utilizando es, por ejemplo, el los modelos de Markov [Brants 2000] ocultos, como es nuestro caso, dicho criterio podría s la comparación de las probabilidades acumuladas normalizadas. Por ejemplo, en la figura 7 llamemos p_i a la probabilidad acumulada del mejor camino (marcado con una línea más grues del enrejado i. Estos valores, es decir p_1, p_2, p_3 y p_4, no son directamente comparable No obstante, si utilizamos probabilidades logarítmicas podemos obtener valores normalizad dividiendo dichas probabilidades por el número de *tokens*. En este caso, $p_1/5$, $p_2/6$, $p_3/7$ $p_4/7$ sí que son comparables. En otros paradigmas, los criterios podrían no ser tan sencill de identificar, y, en cualquier caso, la evaluación individual de cada posible combinación de l

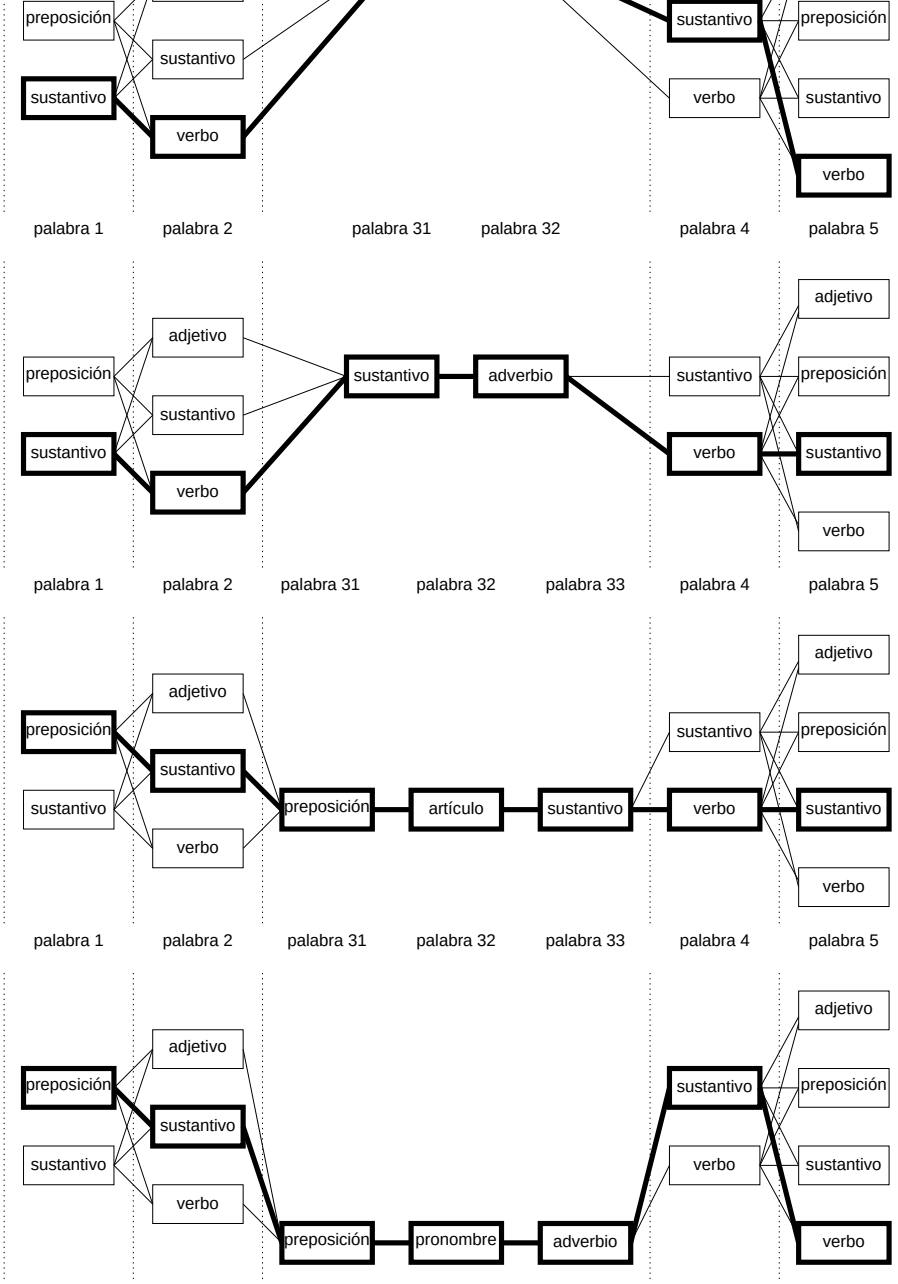


Figura 7.1: Conjunto de diferentes segmentaciones sobre distintos enrejados

ra con dos segmentaciones posibles apareciera en la misma frase de la figura 7.1, entonces amos $4 \times 2 = 8$ secuencias de *tokens* diferentes.

or ello, en nuestro caso, hemos preferido abordar el diseño de una extensión del algoritmo erbi [Viterbi 1967] que, sin pérdida de generalidad y sin necesidad de información extra, es de etiquetar flujos de *tokens* de distinta longitud sobre el mismo enrejado (ver figura 7.2),

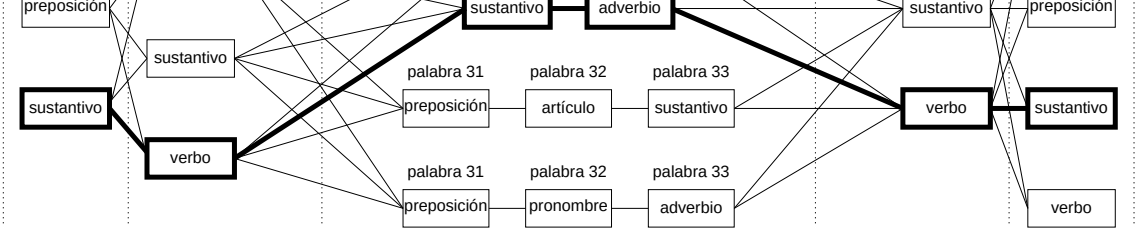


Figura 7.2: Conjunto de diferentes segmentaciones sobre el mismo enrejado

es, por tanto, el paso final del proceso, y su salida es precisamente el texto segmentado desambiguado.

7.2 Extensiones del algoritmo de Viterbi

Como indicamos anteriormente vamos a adaptar el algoritmo de Viterbi para que sea capaz enfrentarse a flujos de *tokens* de distinta longitud. Es decir, a parte de asignar una etiqueta cada *token*, debe ser capaz de decidir si alguno de ellos constituye o no una misma entidad asignar, en cada caso, la cantidad de etiquetas adecuada en función de las diferentes alternativ de segmentación.

7.2.1 Representación de las alternativas de segmentación sobre un *retícul*

Para poder llevar a cabo la modificación del algoritmo de Viterbi, en primer lugar debemos defi una estructura capaz de representar coherentemente las distintas alternativas de segmentaci que nos proporciona el módulo de procesamiento de una frase. Lo primero que nos podem preguntar es si nos sirven los enrejados clásicos. Para ello veamos un ejemplo y estudiemos sob él dicha posibilidad.

Ejemplo 7.3 Supongamos que tenemos la frase: `El sin embargo fue`. El preprocesador n da dos segmentaciones posibles para `sin embargo`: una que se corresponde con un *token*, cuyo caso hablaremos de *conjunción*, y otra que se corresponde con dos *tokens*, en cuyo ca hablaremos de *preposición* y *sustantivo*.

Si a continuación empezamos a construir el enrejado de Viterbi, tal y como se muestra en primer enrejado de la figura 7.3, llegamos a un punto donde no sabemos dónde colocar la etique *conjunción* que etiqueta a `sin embargo` como única entidad. Podríamos colocarla debajo de palabra `sin` o debajo de la palabra `embargo`, supongamos que la colocamos debajo de la prime palabra, tal y como se muestra en el segundo enrejado de la figura. En este punto nos vamos encontrar con otro inconveniente, y es que no se puede establecer un enlace entre la etiqueta `C` la etiqueta `S` de la palabra `embargo`, tal y como se ilustra en el tercer enrejado de la figura. Pa solucionar este inconveniente pensamos en establecer unas marcas de principio y fin de *toke* De esta manera llegamos a construir el enrejado de forma correcta, tal y como se muestra al fi de la figura 7.3. No obstante, como vimos, el enrejado clásico presenta muchas trabas a la ho de representar las segmentaciones ambiguas que nos proporciona el módulo del preprocesado Por este motivo, decidimos que es adecuado pensar en otra estructura más coherente y cómo

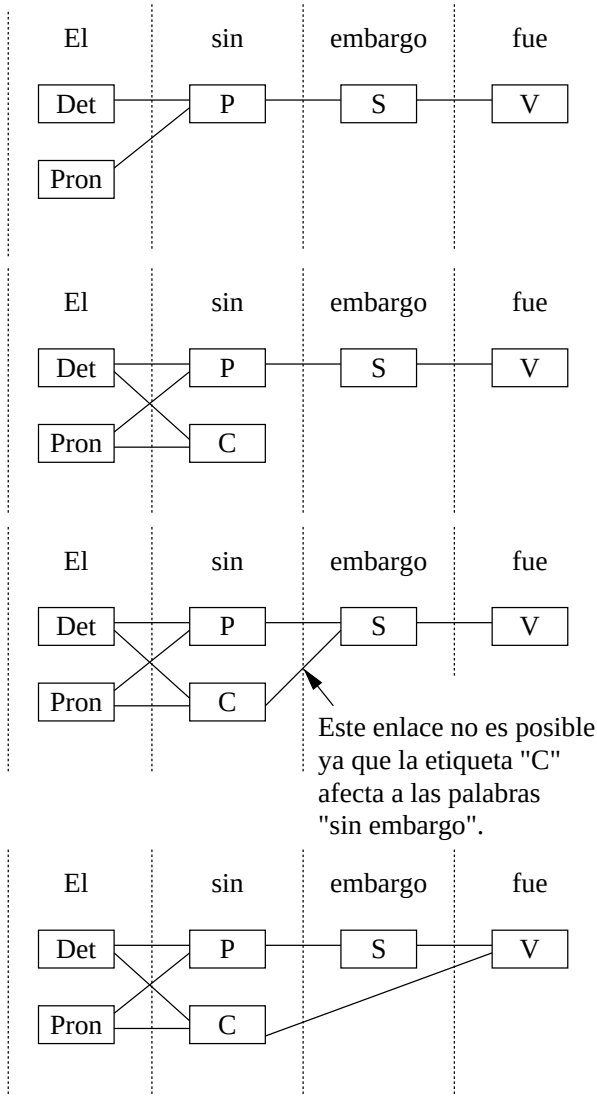


Figura 7.3: Evolución de la construcción del enrejado de Viterbi

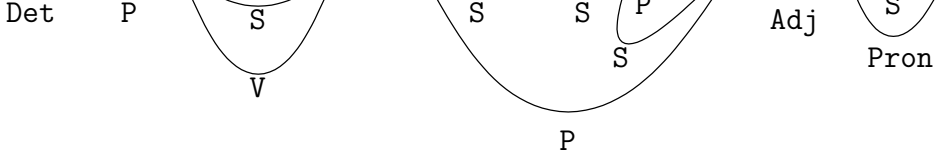


Figura 7.4: Retículo para la frase: `El sin embargo fue a pesar la fruta`

Como se ha indicado anteriormente lo que buscamos es una estructura lo más representati posible para ilustrar nuestro problema, y con la que podamos trabajar de una manera senci y lógica. Esta forma de representación puede venir dada a través de la estructura matemáti conocida como *retículo*. A continuación introducimos la definición formal de *retículo*.

Definición 7.1 Un *retículo* es una relación de orden parcial, donde dados dos elementos, ést van a tener un *ínfimo*, que es el mayor elemento de todos los menores, y un *supremo*, que es menor elemento de todos los mayores.

Ahora, y de nuevo con un ejemplo, mostraremos la forma que tiene un *retículo* y su aplicaci sobre el problema que estamos tratando, que básicamente consiste en numerar los huecos ent las palabras en lugar de las propias palabras.

Ejemplo 7.4 Supongamos que tenemos la frase: `El sin embargo fue a pesar de nue la fruta`. El retículo que representa todas las posibles segmentaciones y etiquetacion correspondientes es el que se muestra en la figura 7.4. Tal y como podemos ver, los hueo entre las palabras que intervienen en la frase se corresponden con los puntos del retículo, y l arcos se corresponden con las posibles etiquetas de los distintos *tokens* que pueden dividir frase. De esta manera, tal y como podemos apreciar en la figura, los caminos por los que puede navegar y los posibles solapamientos entre ellos se representan de una manera más senci y manejable.

7.2.2 Funcionamiento del algoritmo de Viterbi sobre un *retículo*

Llegados a este punto nos disponemos a estudiar, analizar y presentar el algoritmo de Viter modificado que opera sobre *retículos* [Brants 1999]. Como habíamos visto, antes una etique abarcaba exactamente a una palabra. Ahora una etiqueta va a poder abarcar a un núme arbitrario de palabras. Y además, va a cambiar la noción de estado, ya que en un *retícu* la etiqueta del arco que va de un punto a otro representa un estado, mientras que los punt se corresponden con instantes de tiempo o, en nuestro caso, con los huecos que hay entre l palabras. De esta manera, cada arco del retículo se va representar con una tripla (t, t', q), don t se corresponde con el instante de tiempo inicial, t' con el instante de tiempo final y q con estado en cuestión.

Antes de abordar la descripción del algoritmo vamos a introducir las siguientes variable $\Delta_{t,t'}(q)$ es la probabilidad máxima acumulada de llegar al estado q desde el instante de tiempo hasta el instante de tiempo t', es decir, almacena la probabilidad del mejor camino que termi en el estado q empezando en el instante de tiempo t y terminando en el instante t'. Hablan en términos de NLP será la probabilidad máxima acumulada de llegar a la etiqueta q para l

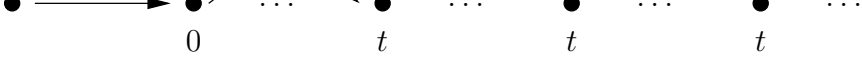


Figura 7.5: Primera extensión del algoritmo de Viterbi: Inicialización

q entre los instantes de tiempo t y t', en NLP es la probabilidad de que la etiqueta q la palabra o secuencia de palabras que hay entre los puntos t y t'.

ritmo 7.1 Primera extensión del algoritmo de Viterbi para que trabaje sobre *retículos*.

Inicialización:
(ver figura 7.5)

$$\Delta_{0,t}(q) = P(q|q_s)\delta_{0,t}(q).$$

Recursión:
(ver figura 7.6)

$$\Delta_{t,t'}(q) = \max_{(t'',t,q')\in Reticulo} \Delta_{t'',t}(q')P(q|q')\delta_{t,t'}(q), \qquad 1 \leq t < T. \tag{7.1}$$

Terminación:
(ver figura 7.7)

$$\max_{Q\in Q^*} P(Q, Reticulo) = \max_{(t,T,q)\in Reticulo} \Delta_{t,T}(q)P(q_e|q).$$

e forma adicional es necesario ir guardando el camino de las triplas que maximizan cada ulador $\Delta_{t,t'}(q)$. Cuando alcancemos el instante de tiempo T, debemos recuperar la mejor del *retículo*:

$$(t_1^m, T, q_1^m) = \arg \max_{(t,T,q)\in Reticulo} \Delta_{t,T}(q)P(q_e|q).$$

endo $t_0^m = T$, recuperamos los argumentos $(t'', t, q) \in Reticulo$ que maximizan la ón 7.1 avanzando hacia atrás en el tiempo:

$$(t_{i+1}^m, t_i^m, q_{i+1}^m) = \arg \max_{(t'',t_i^m,q')\in Reticulo} \Delta_{t'',t_i^m}(q')P(q_i^m|q')\delta_{t_i^m,t_{i-1}^m}(q_i^m),$$

$i \geq 1$, hasta alcanzar $t_k^m = 0$. De manera que, $q_1^m \dots q_k^m$ es el mejor camino de estados nacia atrás. □

que el algoritmo trabaje con la hipótesis de segundo orden, es decir, para que considere la dencia no sólo del estado anterior, sino de los dos estados anteriores, el *retículo* se debe uir de manera que los estados estén formados por dos etiquetas, es decir, cada arco va a asociado dos etiquetas posibles, tal y como se muestra en la figura 7.8. De esta manera, ementos que representan a una arista serán de la forma (t, t', q, q'), y para poder navegar a arista a otra se debe de cumplir la siguiente condición: supongamos que tenemos los ntos (t'', t''', q'', q''') y (t, t', q, q'), y queremos pasar del instante t'' al instante t', en este e podrá transitar, si y sólo si, $q''' = q$ y $t''' = t$. Para hacernos una idea de la forma que a un *retículo* de segundo orden, podemos acudir a la figura 7.9, que se corresponde con el

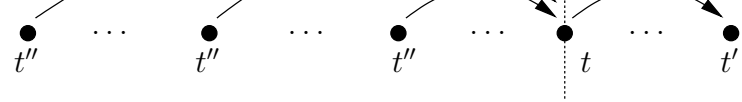


Figura 7.6: Primera extensión del algoritmo de Viterbi: Recursión

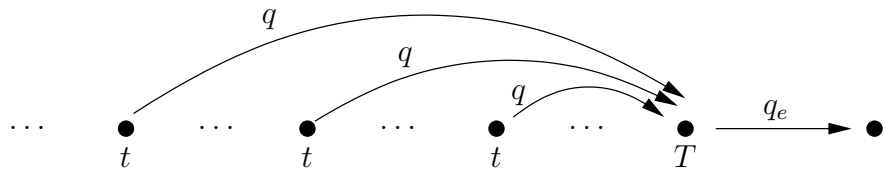


Figura 7.7: Primera extensión del algoritmo de Viterbi: Terminación

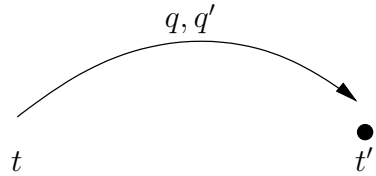


Figura 7.8: Forma de los arcos de los retículos de orden 2

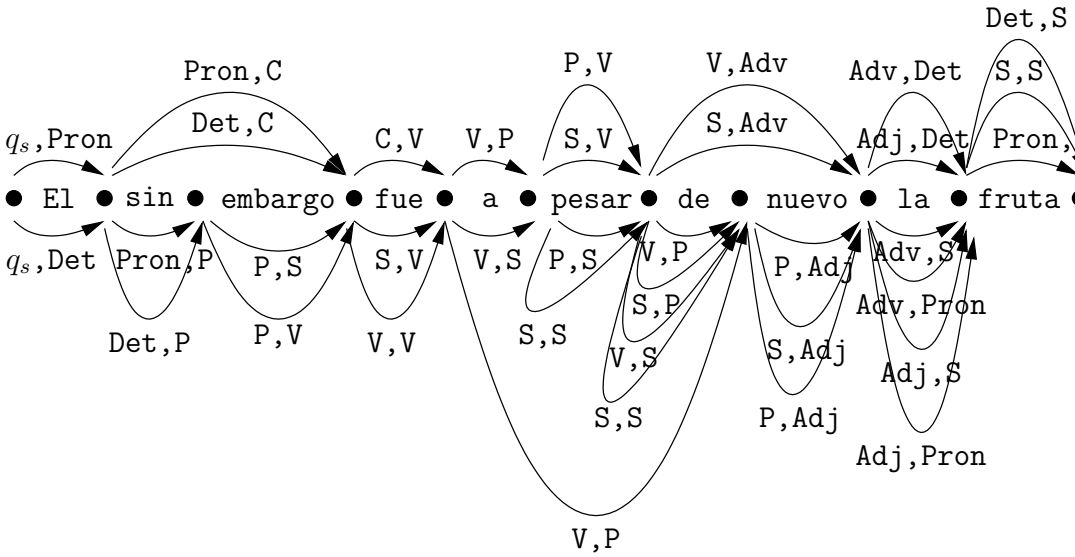


Figura 7.9: Retículo de orden 2 para el ejemplo

etiquetación de textos de manera combinada. Con este fin, debemos adaptar el nuevo tmo de Viterbi modificado para que trabaje con flujos de *tokens* de distinta longitud. Para rabajaremos con la idea de mantener una serie de acumuladores Δ por cada camino de ta longitud que nos conduce a un estado dado. De esta manera, cada uno de los arcos tículo, que como vimos se corresponden con estados, va a poder tener asociado más de un ulador. A continuación presentaremos el algoritmo que tiene en cuenta dicha mejora.

ritmo 7.2 Segunda extensión del algoritmo de Viterbi para que trabaje con flujos de *s* de distinta longitud.

Inicialización:
(ver figura 7.5)

$$\Delta_{0,t,1}(q) = P(q|q_s)\delta_{0,t}(q).$$

Recursión:
(ver figura 7.6)

$$\Delta_{t,t',l}(q) = \max_{(t'',t,q')\in Reticulo} \Delta_{t'',t,l-1}(q')P(q|q')\delta_{t,t'}(q), \qquad 1 \leq t < T. \tag{7.2}$$

Terminación:
(ver figura 7.7)

$$\max_{Q\in Q^*} P(Q, Reticulo) = \max_{l} \frac{\max\limits_{(t,T,q)\in Reticulo} \Delta_{t,T,l}(q)P(q_e|q)}{l}$$

especto a la fase de inicialización destacamos que al instante t llegaremos siempre con ongitud igual a 1. En la fase de recursión, sumamos uno a las longitudes de cada uno de ntadores cuyo estado tiene al instante t' como destino. Y en la fase de terminación nos remos con el mejor de los caminos normalizados. Para ello obtenemos primero los mejores os normalizados de distinta longitud, y después elegimos el mejor.

rma adicional, y al igual que en el algoritmo anterior, necesitamos mantener una traza con ementos del *retículo* que maximizan cada $\Delta_{t,t'}(q)$.
uando se alcance el instante T, obtendremos la longitud del mejor camino del *retículo* de uiente manera:

$$L = \arg\max_{l} \frac{\max\limits_{(t,T,q)\in Reticulo} \Delta_{t,T,l}(q)P(q_e|q)}{l}$$

ontinuación, para poder obtener el mejor último elemento de todos los caminos de longitud uaremos como sigue (ver figura 7.10):

$$(t_1^m, T, q_1^m) = \arg \max_{(t,T,q)\in Reticulo} \Delta_{t,T,L}(q)P(q_e|q).$$

continuación, para recolectar los argumentos (t'', t, q') que maximizan la ecuación 7.2, os $t_0^m = T$ y caminando hacia atrás, hacemos lo siguiente (ver figura 7.11):

$$(t_{i+1}^m, t_i^m, q_{i+1}^m) = \arg \max_{(t'',t^m_i,L-i}(q')P(q_i^m|q')\delta_{t^m_i,t^m_{i+1}}(q_i^m),$$

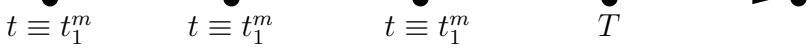


Figura 7.10: Mejor último elemento de todos los caminos de longitud L

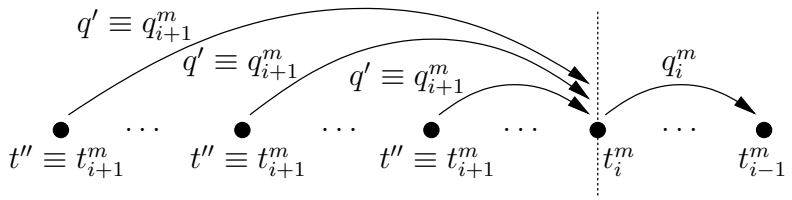


Figura 7.11: Recolección de los elementos que maximizan la ecuación 7.2

para $1 \leq i \leq k$, hasta alcanzar $t_k^m = 0$. Ahora $q_1^m, \ldots, q_k^m$ constituye la mejor secuencia hipótesis (leída al revés).

7.3 Cálculo de la complejidad

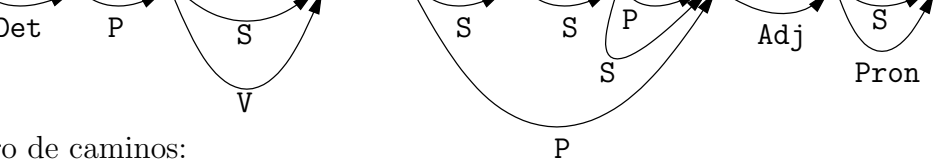
A continuación, con el propósito de hacer una comparativa de complejidades entre el algoritm de Viterbi dinámico y el algoritmo de Viterbi clásico aplicado sobre *retículos*, vamos a realiz un análisis exhaustivo sobre el ejemplo que venimos estudiando, con lo cual, aconsejamos lector que durante el presente estudio tenga en cuenta el *retículo* de la figura 7.4. Antes nada, recordamos que el algoritmo dinámico se ejecuta sobre un *retículo* que contempla tod los caminos de distinta longitud por los que podemos navegar. Sin embargo, para poder ejecut el algoritmo clásico necesitaremos que el *retículo* soporte caminos de una única longitud. I esta forma, el algoritmo se deberá ejecutar sobre distintos *retículos*, obteniendo para cada u de ellos un camino óptimo, que a continuación normalizaremos para después quedarnos con mejor de los caminos normalizados.

Para ambos algoritmos vamos a estudiar la **complejidad espacial** y la **complejida temporal**, siendo para ello necesario el cálculo de los siguientes parámetros: el número caminos posibles dentro del *retículo*, el número de longitudes diferentes que conforman dich caminos, el número de acumuladores necesarios para cada estado del *retículo* y el número operaciones realizadas para la obtención de dichos acumuladores.

Para el estudio de la complejidad espacial calcularemos el número de acumuladores necesari que tenemos que almacenar. Para ello, en el algoritmo clásico tendremos tantos acumulador como arcos existentes, mientras que en el algoritmo dinámico este número se calculará de siguiente manera: cada arco tendrá tantos acumuladores como longitudes distintas tengan l caminos que pueden alcanzar su punto de origen.

En cuanto a la complejidad temporal, calcularemos el número de operaciones que debem llevar a cabo. La forma de calcularlo es común a ambos algoritmos, y no es más que para ca arco, sumar el número de acumuladores de los arcos que pueden alcanzar su punto de origen

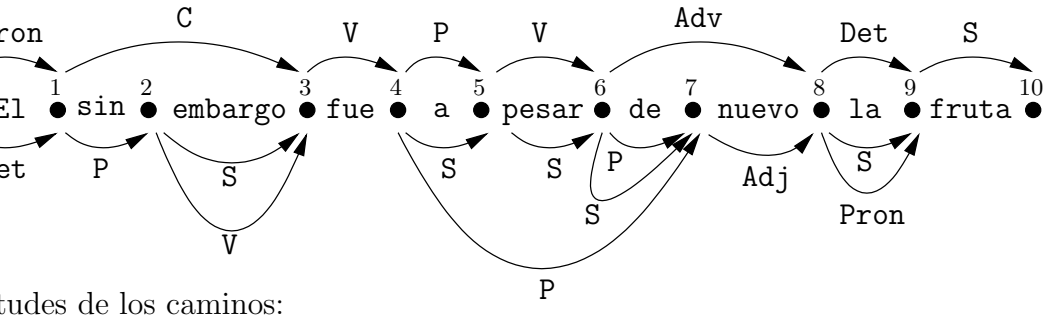
A continuación estudiaremos ambos tipos de complejidad para el algoritmo de Viterbi dinámic



o de caminos:

117	78	39	39	18	9	3	3	1	1

Figura 7.12: Número de caminos para cada instante de tiempo del *retículo*



tudes de los caminos:

1	2	2	3	4	5	4	5	6	7
		3	4	5	6	5	6	7	8
						6	7	8	9
						7	8	9	10

Figura 7.13: Longitudes de los caminos para cada instante de tiempo del *retículo*

es, y lo haremos de la siguiente manera: viendo el *retículo* como un *autómata finito acíclico ninista* lo podemos convertir en un *autómata finito acíclico determinista numerado* tal y se indica en la definición 2.8, de esta manera, obtenemos el número de caminos posibles el punto en el que nos encontramos hasta el instante final. Así, tal y como podemos ver figura 7.12, obtenemos un total de 234 caminos posibles.

continuación calculamos la longitud de los caminos por los que podemos navegar, y para ctuamos de la siguiente forma. En el instante inicial no tenemos ningún camino, con lo cual gitud es 0. Para un instante de tiempo t distinto del inicial, deberemos tener en cuenta los arcos entrantes a dicho instante y quedarnos con todas las longitudes de los instantes del arco, a las que les sumamos uno. Así, para nuestro *retículo*, obtenemos un total de longitudes de caminos diferentes, que son 7, 8, 9 y 10, respectivamente. Dicha situación lemos comprobar sobre el *retículo* de la figura 7.13, donde vamos calculando las longitudes caminos para cada instante de tiempo.

on estas cifras ya podemos calcular el número total de acumuladores que el algoritmo ico necesita, para así determinar la complejidad espacial del algoritmo dinámico. De la a manera, ya estamos en condiciones de calcular el número de operaciones necesarias, os va a determinar la complejidad temporal para dicho algoritmo. La forma en la que se

0	Pron	1	0	1	1
0	Det	1	0	1	1
1	P	2	1	1	2
1	C	3	1	1	2
2	S	3	2	1	1
2	V	3	2	1	1
3	V	4	2,3	2	3
4	P	5	3,4	2	2
4	S	5	3,4	2	2
5	V	6	4,5	2	4
5	S	6	4,5	2	4
4	P	7	3,4	2	2
6	P	7	5,6	2	4
6	S	7	5,6	2	4
6	Adv	8	5,6	2	4
7	Adj	8	4,5,6,7	4	6
8	Det	9	5,6,7,8	4	6
8	Pron	9	5,6,7,8	4	6
8	S	9	5,6,7,8	4	6
9	S	10	6,7,8,9	4	12
				44	73

Como podemos observar, obtenemos un total de 44 acumuladores y de 73 operaciones, cifr que denotan la complejidad espacial y temporal, respectivamente, para el caso del algoritmo Viterbi dinámico.

Una vez estudiada la complejidad para el caso dinámico, nos disponemos a estudiar complejidad para el algoritmo clásico. Como indicamos anteriormente, si utilizamos es aproximación el número de ejecuciones para este algoritmo viene determinado en función las diferentes longitudes que pueden tener los caminos. Cada ejecución nos va a proporcion un camino normalizado, y una vez obtenidos todos estos caminos normalizados elegiremos mejor. Para reflejar este proceso, pensamos en una primera alternativa, la cual consiste en construcción de un retículo para cada longitud de camino existente. De esta manera, empezam a construir el retículo correspondiente para los caminos de longitud 7, tal y como podemos v en la figura 7.14, resultando así un total de 6 caminos posibles. Continuamos ahora con construcción del segundo retículo, donde tenemos en cuenta todos los caminos de longitud Sin embargo, en este punto concluimos que esta idea no es buena, ya que se pueden produ solapamientos entre los caminos que pueden dar lugar, en el mismo retículo, a caminos longitudes diferentes. En concreto, si nos fijamos en la figura 7.15, se puede apreciar que intentar representar en un mismo *retículo* todos los caminos de longitud 8, surgen camin de longitud 7, cuyas trazas se han marcado con los arcos punteados. Esto quiere decir que algoritmo clásico no se puede aplicar, y que es necesario pensar en otra repartición de los posibl caminos en diferentes *retículos* de manera que no haya conflictos de longitud.

La alternativa correcta consiste en construir un árbol de decisión donde los nodos ramificación involucren a los elementos conflictivos, es decir, a las palabras que pueden ten

Figura 7.14: Caminos de longitud 7

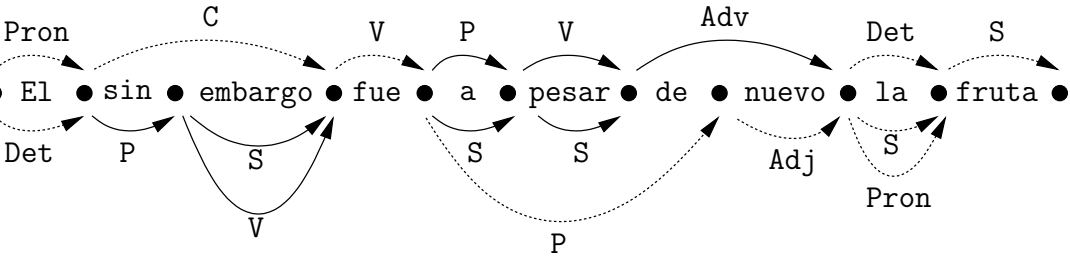


Figura 7.15: Caminos de longitud 8

retículo que venimos estudiando, a través del cual se deduce que son necesarios seis los y no cuatro, para representar todos los caminos sin que se produzcan solapamientos ctivos como los indicados en la primera alternativa. Cabe destacar que nos podemos trar con varios *retículos* que contienen caminos de una misma longitud, pero efectivamente minos involucrados no se pueden representar sobre un mismo *retículo* sin dar lugar a ctos. Veamos a continuación cuáles son estos retículos, así como los acumuladores y ciones involucradas para cada uno de ellos, factores que se calculan de la misma forma ara el caso anterior. En la figura 7.17 tenemos el retículo para los caminos de longitud 7, iendo un total de 6 caminos posibles, 10 acumuladores y 13 operaciones . En la figura 7.18 amos el retículo para caminos de longitud 8, obteniendo para este caso 24 caminos, 13 uladores y 19 operaciones. En la figura 7.19 se encuentra el retículo que refleja los caminos gitud 9, obteniendo un total de 48 caminos posibles, 15 acumuladores y 23 operaciones. evo, en la figura 7.20 nos encontramos con otro retículo para los caminos de longitud 8, nos encontraremos con un total de 12 caminos, 12 acumuladores y 16 operaciones. Otra n la figura 7.21 mostramos los caminos de longitud 9, dando un total de 48 caminos, 15 uladores y 22 operaciones. Y por último, en la figura 7.22 mostramos el retículo para los os de mayor longitud posible, 10, dando un total de 96 caminos, 17 acumuladores y 26 ciones.

on lo cual, para esta aproximación obtendremos un total de 82 acumuladores y operaciones, valores que se corresponden con la complejidad espacial y temporal, ctivamente.

nte las cifras obtenidas podemos ver claramente que la versión dinámica del algoritmo, este caso, resulta mucho mejor, ya que nos encontramos con una complejidad espacial de umuladores frente a 82 que necesita la versión clásica. De igual manera, la complejidad oral también ofrece mejores resultados, nos encontramos con un total de 36 operaciones a las 73 de la versión clásica. Con lo cual, como conclusión podemos determinar que esta n del algoritmo promete mucho, aunque el trabajo al respecto todavía no está cerrado, to en el que profundizaremos en el tema dedicado a las conclusiones.

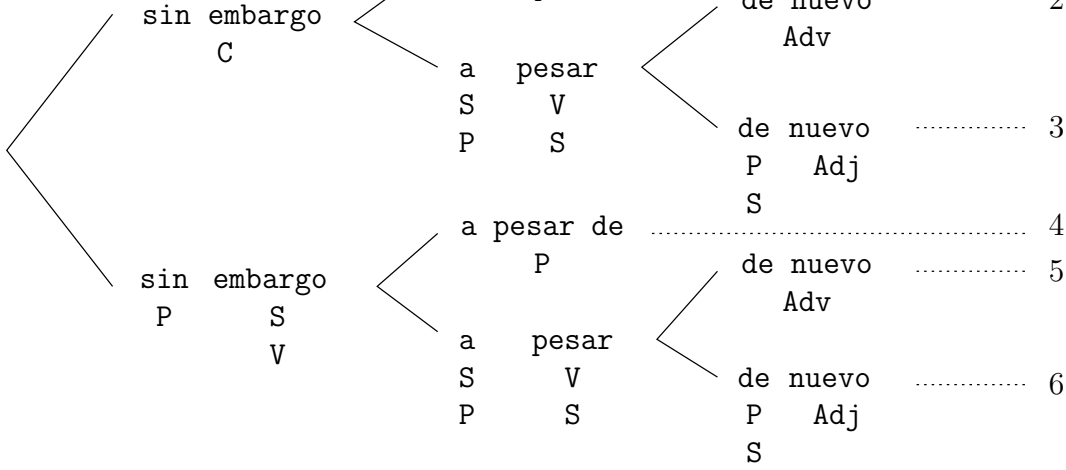


Figura 7.16: Árbol de decisión para las palabras conflictivas de la frase: `El sin embar fue a pesar de nuevo la fruta`

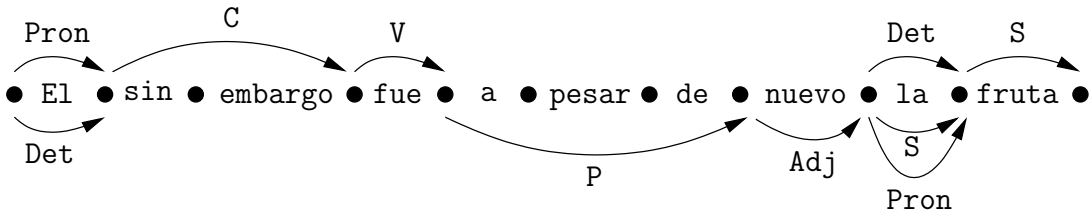


Figura 7.17: Retículo para caminos de longitud 7. Acumuladores = 10 y operaciones = 13

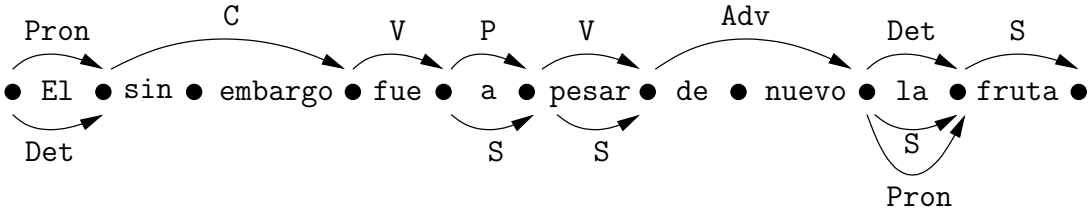


Figura 7.18: Retículo para caminos de longitud 8. Acumuladores = 13 y operaciones = 19

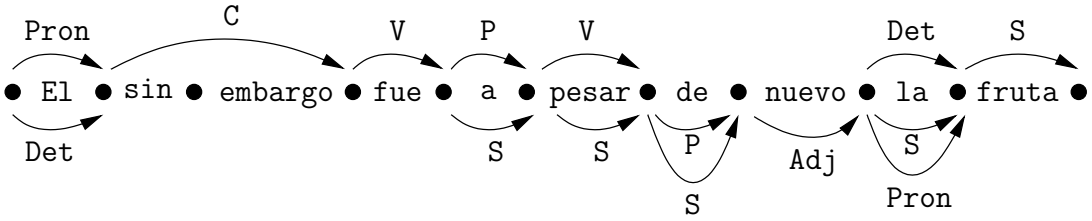
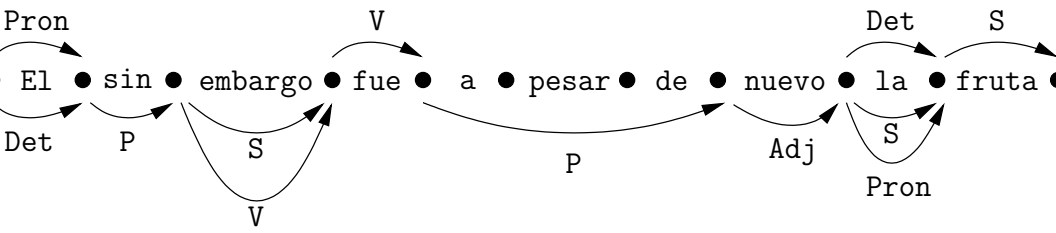
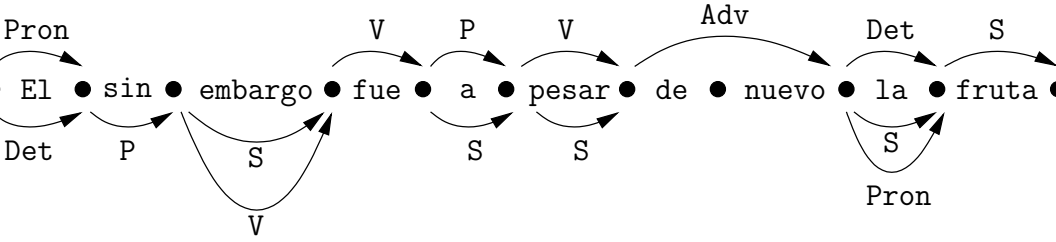


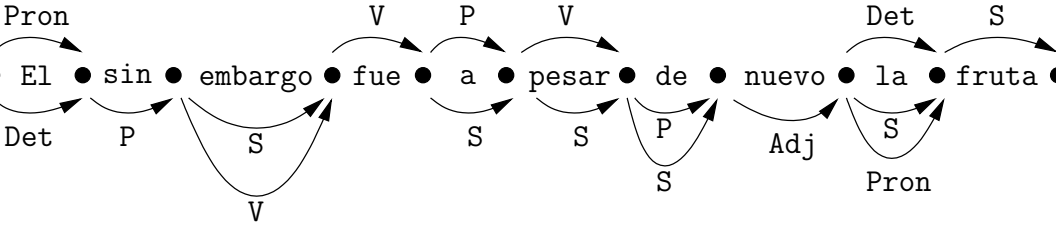
Figura 7.19: Retículo para caminos de longitud 9. Acumuladores = 15 y operaciones = 23



ura 7.20: Retículo para caminos de longitud 8. Acumuladores = 12 y operaciones = 16



ura 7.21: Retículo para caminos de longitud 9. Acumuladores = 15 y operaciones = 22



ura 7.22: Retículo para caminos de longitud 10. Acumuladores = 17 y operaciones = 26

Capítulo 8

Conclusiones y trabajo futuro

Para poder llevar a cabo con garantías el proceso de etiquetación de textos en lenguaje natur
hemos visto que es indispensable tener en cuenta el contexto en el que aparecen cada una de l
palabras del texto. Hemos estudiado diferentes métodos para modelizar dicho contexto, pe
básicamente todos ellos los podemos agrupar en dos clases principales: los sistemas basados
modelos estocásticos, y los sistemas basados en reglas o transformaciones contextuales.

Todos los estudios realizados determinan que el marco probabilístico resulta más adecua
que las aproximaciones simbólicas, constituyendo así la aproximación que mejores rendimient
ofrece actualmente. Además, la adquisición de este tipo de *conocimiento máquina* se reali
de manera automática, y su aplicación resulta sencilla. No obstante, nos encontramos que l
sistemas de etiquetación puramente estocásticos suelen etiquetar erróneamente alrededor del
ó 3% de las palabras de un texto dado. Por tanto, para poder enfrentarnos con garantías a es
pequeño porcentaje de errores, es necesario añadir información de más alto nivel.

La fuente de esta información de más alto nivel puede ser el *conocimiento humano*.
experiencia que un lingüista puede aportar, mediante un pequeño conjunto de reglas contextual
elegidas apropiadamente, puede ser muy preciso en algunos ámbitos, y por lo tanto puede ayud
a mejorar apreciablemente el proceso de etiquetación. No obstante, la adquisición de este ti
de conocimiento y su aplicación exacta pueden resultar complejas.

Por este motivo, se busca el uso combinado de estos dos tipos de conocimiento. Aprovechan
las ventajas que ambos nos ofrecen, se puede abordar el diseño de un sistema de alta precisió
el cual pasa por construir un sistema híbrido que combine las dos estrategias considerad
anteriormente: los modelos estocásticos y los modelos basados en reglas contextuales.

Con el objetivo de mejorar estos resultados de etiquetación gran parte del esfuerzo desarrolla
en este proyecto se ha dedicado al estudio y al análisis de la sintaxis de los sistemas
etiquetación basados en reglas de restricciones, hoy en día considerados los de mayor exponen
en sistemas de etiquetación híbridos. Este proyecto no sólo constituye un intento de mejorar
comprensión de las gramáticas de restricciones, mediante la simplificación de la sintaxis de l
mismas, sino que también se preocupa de abordar la compilación y la ejecución de las reglas
la forma más eficiente posible. A este nivel las aportaciones de este proyecto se centran en l
siguientes puntos:

- Se ha mejorado la sintaxis del formalismo de reglas del lenguaje LEMEI
  [Reboredo y Graña 2000], con el propósito de hacer los esquemas generales de regl
  contextuales más intuitivos y fáciles de entender. De esta manera se consigue u
  aceptación más positiva de las gramáticas de restricciones por parte de los lingüistas.

n lado, y en mejoras de rendimiento por otro. Como conclusiones importantes podemos ar lo siguiente:

La expresividad de las reglas contextuales ha mejorado tanto que prácticamente las reglas hablan por sí mismas, ayudándonos para ello de la teoría de conjuntos. Sin embargo, aclaramos que el trabajo a este nivel no está todavía cerrado ya que son los usuarios finales los que deben tomar las últimas decisiones de diseño. No obstante, el formalismo desarrollado se ha modificado de tal manera que se sigue manteniendo la flexibilidad que presentaban las antiguas reglas de las gramáticas de restricciones.

La compilación y la ejecución de las reglas contextuales resulta eficiente, obteniendo además los mismos resultados que cabrían esperar al ejecutar cada una de las reglas separadas y de forma secuencial. La única desventaja que hemos detectado es que el tamaño de los traductores compilados y normalizados no es tan compacto como esperábamos en un principio. Sin embargo, este pequeño problema puede deberse a alguno de los siguientes motivos:

- La propia naturaleza y estructura de los traductores.
- La operación de composición sobre traductores de estado finito es compleja e involucra un gran coste computacional.
- La forma de representación interna de los traductores que utiliza la herramienta en la que nos hemos apoyado. A esto también se le debe unir la manera en que realiza las operaciones sobre los traductores, que probablemente se lleven a cabo optimizando la complejidad temporal, pero no la espacial. En este sentido resultaría interesante, y lo proponemos como trabajo futuro, el estudio, la evaluación y la comparación con otras herramientas disponibles, o la reflexión sobre la conveniencia de realizar una implementación propia a este nivel.

El prototipo desarrollado podría resultar interesante orientearlo a la detección de errores sistemáticos que producen los etiquetadores. Es decir, a partir de los errores que acostumbran a cometer los etiquetadores, se podría intentar generar automáticamente las reglas contextuales que los corrigen, de manera que al aplicar dichas reglas sobre la salida del etiquetador estos errores serían eliminados.

de finalizar, consideramos interesante recordar que uno de los mayores inconvenientes de stemas de etiquetación es que los resultados obtenidos dependen en exceso del estilo de xtos que se han utilizado para su entrenamiento. Actualmente, se ha demostrado que ma de paliar este problema es a través del diseño de sistemas híbridos. Este tipo de as es el único que permite la incorporación de información capaz de enfrentarse a los s sistemáticos que cometen los etiquetadores tradicionales, y a los fenómenos lingüísticos rrectamente formalizados.

undo objetivo de este proyecto aborda la formalización del proceso de segmentación. Por do, hemos visto que los etiquetadores actuales asumen que el texto de entrada aparece ya tamente segmentado, es decir, dividido de manera adecuada en *tokens*. Esta hipótesis de

Los mayores problemas surgen cuando el módulo preprocesador detecta ambigüedades
el proceso de segmentación. Esta es la situación en la que la solución que se hemos diseña
adquiere su mayor utilidad, y la reflexión que subyace en la idea propuesta es sencilla:

- A este nivel, pensamos que un preprocesador no debería de ir más allá de la simp
  detección y preetiquetación de cada una de las alternativas de segmentación.

- Y, por tanto, la elección de una de ellas debe de hacerse en función del contexto, que
  precisamente lo que estudia el etiquetador.

Así pues, y con este objetivo en mente, hemos diseñado una serie de extensiones sobre
algoritmo de clásico de Viterbi:

- Primeramente, se ha extendido el algoritmo para que pueda trabajar sobre retículos, debi
  a que los enrejados clásicos no son capaces de representar las dependencias solapadas q
  pueden aparecer entre las distintas alternativas de segmentación.

- El algoritmo que trabaja sobre retículos se extendió después para poder seguir trabajan
  con la hipótesis de segundo orden, es decir, la que considera la dependencia no sólo d
  estado anterior, sino de los dos anteriores.

- Finalmente, se incorporó al algoritmo una normalización dinámica basada en la longit
  de los caminos, para evitar que los caminos más cortos se vieran favorecidos sobre los m
  largos.

Posteriormene, se ha demostrado que el algoritmo propuesto es equivalente a la aplicaci
individual del algoritmo clásico sobre las diferentes alternativas de segmentación, y q
proporciona la misma elección que la que realizaría la posterior normalización de dich
alternativas en función de sus longitudes.

Se ha realizado también un estudio intuitivo de la complejidad espacial y temporal d
algoritmo propuesto, y se ha visto que, al menos en este contexto, es comparable a la d
algoritmo clásico y, por supuesto, inferior a la complejidad total que involucran las sucesiv
aplicaciones individuales del algoritmo clásico sobre todas las posibles segmentaciones.

No obstante, el trabajo no está todavía cerrado, y queda pendiente la consideración de l
siguientes aspectos:

- Es cierto que la principal aportación de este nuevo algoritmo se centra en la formalizaci
  del proceso de segmentación. Sin embargo, no se ha podido realizar una evaluaci
  exhaustiva del algoritmo, debido sobre todo a la escasa disponibilidad de textos en l
  que aparezcan un conjunto de segmentaciones ambiguas verdaderamente representativ
  Como ya se ha comentado anteriormente, este tipo de problemas es muy frecuente cuan
  los textos de aplicación están escritos en gallego, y los recursos lingüísticos disponibl
  para este idioma son prácticamente inexistentes. En otras palabras, queda por comprob
  que el algoritmo propuesto efectivamente resuelve bien el problema para el que ha si
  concevido. No obstante, ha quedado diseñado un eficiente marco para tal fin, el cu
  además constituye una aproximación muy elegante a la tarea de segmentación, porq

esos algoritmos al contexto de la etiquetación.

Por último, es necesario comentar que esta segunda fase del trabajo presentado ha sido tratada de manera totalmente aislada. Es decir, no hemos hablado en ningún momento de cómo podrían convivir juntos los dos objetivos que se han cubierto en este proyecto. Precisamente, como parte del trabajo futuro, quedaría también el estudiar la manera de integrar el uso de reglas contextuales sobre retículos.

alquier caso, bajo estas hipótesis de trabajo, y a falta de realizar y comprobar todos las ones anteriormente comentadas, creemos que es posible afirmar que se ha alcanzado un en el cual el proceso de etiquetación no podrá avanzar mucho más allá en un futuro iato.

Vive como si fueras a morir mañana.
Aprende como si fueras a vivir siempre.

Parte V

Apéndices y bibliografía

Apéndice A

Juegos de etiquetas

Este apéndice describe sólo las etiquetas utilizadas en los ejemplos incluidos en el presen
trabajo. Como ya se ha mencionado anteriormente, dichas etiquetas provienen de los *tag s*
utilizados en los proyectos Galena y Corga. Una descripción completa de ambos *tag s*
puede encontrarse a partir de [Vilares *et al.* 1995] y [Vilares *et al.* 1998], respectivamente. M
información sobre estos proyectos está disponible también en `http://coleweb.dc.fi.udc.e`

A.1 Etiquetas Galena

Afp0	Adjetivo femenino plural no aplicable
Amp0	Adjetivo masculino plural no aplicable
Ams0	Adjetivo masculino singular no aplicable
Ncyyp	Numeral cardinal tanto in/determinado plural
P	Preposición
P31	Primer elemento de una preposición de tres elementos
P32	Segundo elemento de una preposición de tres elementos
P33	Tercer elemento de una preposición de tres elementos
Re3sam	Pronombre enclítico acusativo masculino tercera persona del singular
Re3yyy	Pronombre enclítico acusativo o dativo masculino o femenino tercera persona
Scfp	Sustantivo común femenino plural
Scfs	Sustantivo común femenino singular
Scmp	Sustantivo común masculino singular
Scms	Sustantivo común masculino plural
V000f0	Verbo infinitivo
V000f0PE1	Verbo infinitivo con un pronombre enclítico
V2pei0	Verbo pretérito de indicativo segunda persona del plural
V2sei0	Verbo pretérito de indicativo segunda persona del singular
V2spm0	Verbo presente de imperativo segunda persona del singular
V3spi0	Verbo presente de indicativo tercera persona del singular
Vysci0	Verbo condicional de indicativo primera y tercera persona del singular
Vysii0	Verbo imperfecto de indicativo primera y tercera persona del singular
Vysps0	Verbo presente de subjuntivo primera y tercera persona del singular

ms	Pronombre átono acusativo masculino tercera persona del singular
	Sustantivo común masculino singular
0	Verbo presente indicativo segunda persona del singular

Bibliografía

Abney, S. (1996). Part-of-speech tagging and partial parsing. In S. Young and G. Bloothoo (eds.), *Corpus-Based Methods in Language and Speech Processing*, pp. 118-136. Dordrecl Kluwer Academic.

Aho, A.V.; Sethi, R.; Ullman, J.D. (1985). Compilers: principles, techniques and tools. *Addiso Wesley*, Reading, MA.

Alvarez Rosario; Regueira X. L.; Motegudo H. (1Alvarez86986). Gramática Galega. *Edicion Galaxia*.

Bahl, L.R.; Mercer, R.L. (1976). Part-of-speech assignement by a statistical decision algorith In *International Symposium on Information Theory*, Ronneby (Sweden).

Baker, J.K. (1975). Stochastic modeling for automatic speech undestanding. D. Raj Reddy (ed.), *Speech Recognition: Invited papers presented at the 1974 IEI symposium*, pp. 297-307. Academic Press, NY.

Baldi, P.; Brunak, S. (1998). Bioinformatics: the machine learning approach. *The MIT Pre* Cambridge, MA.

Brants, Thorsten (1999) Cascaded Markov Models. Universität des Saarland Computerlinguistik. D-66041 Saarbrücken, Germany.

Baum, L.E.; Petrie, L.; Soules, G.; Weiss, N. (1970). A maximization technique ocurring the statistical analysis of probabilistic functions of Markov chains. *Annals of Mathemati Statistics*, vol. 41, pp. 164-171.

Baum, L.E. (1972). An inequality and associated maximization technique in statistic estimation for probabilistic functions of Markov processes. *Inequalities*, vol. 3, pp. 1-8.

Benello, J.; Mackie, A.W.; Anderson, J.A. (1989). Syntactic category disambiguation with neur networks. *Computer Speech and Language*, vol. 3, pp. 203-217.

Brants, T. (1998). Estimating hidden Markov model topologies. In J. Ginzburg, Z. Khasidashvi C. Vogel, J.-J. Lévy and E. Vallduví (eds.), *The Tbilisi Symposium on Logic, Language a Computation: Selected Papers*, pp. 163-176. *CSLI Publications*, Stanford, CA.

Brants, T. (2000). TnT - A statistical part-of-speech tagger. In *Proceedings of the Sixth Appl Natural Language Processing Conference (ANLP-2000)*, Seattle, WA.

Brill, E. (1993a). Automatic grammar induction and parsing free text: a transformation-bas approach. In *Proceedings of the 31st Annual Meeting of the Association for Computation*

he Third International Workshop on Parsing Technologies, Tilburg/Durbuy (The Netherlands/Belgium).

E. (1994). Some advances in rule-based part of speech tagging. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Seattle, WA.

E. (1995a). Transformation-based error-driven learning and natural language processing: case study in part-of-speech tagging. *Computational Linguistics*, vol. 21, pp. 543-565.

E. (1995b). Unsupervised learning of disambiguation rules for part of speech tagging. In *Proceedings of the Third Workshop on Very Large Corpora*, pp. 1-13.

E.; Magerman, D.M.; Marcus, M.P.; Santorini, B. (1990). Deducing linguistic structure rom the statistics of large corpora. In M. Kaufmann (ed.), *Proceedings of the DARPA peech and Natural Language Workshop*, pp. 275-282, San Mateo, CA.

E.; Resnik, P. (1994). A transformation-based approach to prepositional phrase attachment isambiguation. In *Proceedings of the 15th International Conference on Computational inguistics (COLING-94)*, pp. 1198-1204.

n, P.F.; Della Prieta, S.A.; Della Prieta, V.J.; Mercer, R.L.; Resnik, P.S. (1991). Language nodelling using decision trees. *IBM Research Report*, Yorktown Heights, NY.

, R.; Hammond, K.; Kulyukin, V.; Lytinen, S.; Tomuro, N.; Schoenberg, S. (1997). Question answering form frequently asked files. *AI Magazine*, vol. 18, pp. 57-66.

e, C. (1997). Empirical methods in information extraction. *AI Magazine*, vol. 18, pp. 65-79.

od, J.-P.; Tapanainen, P. (1995). Tagging French - comparing a statistical and a constraint-ased method. In *Proceedings of the 7th Conference of the European Chapter of the ssociation for Computational Linguistics*, pp. 149-156.

iak, E. (1993). Statistical language learning. *The MIT Press*, Cambridge, MA.

iak, E. (1997). Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, p. 598-603.

iak, E.; Hendrickson, C.; Jacobson, N.; Perkowitz, M. (1993). Equations for part-of-peech tagging. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, p. 784-789. Menlo Park, CA.

sky, N. (1957). Syntactic structures. The Hague: Mouton.

h, K.W. (1988). A stochastic parts program and noun phrase parser for unrestricted ext. In *Proceedings of the Second Conference on Applied Natural Language Processing*, p. 136-143.

k, J.; Mihov, S.; Watson, B.W.; Watson, R.E. (2000). Incremental construction of minimal

Computational Linguistics, vol. 14, pp. 31-39.

Dempster, A.P.; Laird, N.M.; Rubin, D.B. (1977). Maximum likelihood from incomplete da via the EM algorithm. *J. Roy. Stat. Soc.*, vol. 39(1), pp. 1-38.

Derouault, A.-M.; Merialdo, B. (1986). Natural language modeling for phoneme-to-te transcription. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. pp. 642-649.

Dini, L.; Di Tomaso, V.; Segond, F. (1998). Error-driven word sense disambiguatic In *Proceedings of the 36th Annual Meeting of the Association for Computatior Linguistics/17th International Conference on Computational Linguistics (COLING-9* pp. 320-324.

Durbin, R.; Eddy, S.; Krogh, A.; Mitchison, G. (1998). Biological sequence analysis: probabilis models of proteins and nucleic acids. *Cambridge University Press*.

Elworthy, D. (1994). Does Baum-Welch re-estimation help taggers? In *Proceedings of the Fou Conference on Applied Natural Language Processing*, pp. 53-58.

Fagan, J.L. (1987). Automatic phrase indexing for document retrieval: an examination syntactic and non-syntactic methods. In *Proceedings of the 10th Annual Internatior ACM/SIGIR Conference on Research and Development in Information Retrieval*.

Forney, G.D. (1973). The Viterbi algorithm. In *Proceedings of the IEEE*, vol. 61 (Marc pp. 268-278.

Foster, G.F. (1991). Statistical lexical disambiguation. *Master's thesis*, School of Comput Science, McGill University.

Francis, W.N.; Kučera, H. (1982). Frequency analysis of English usage. *Houghton Miff Company*, Boston, MA.

Franz, A. (1996). Automatic ambiguity resolution in natural language processing. *Lecture No in Artificial Intelligence*, vol. 1171, Springer Verlag, Berlin.

Franz, A. (1997). Independence assumptions considered harmful. In *Proceedings of the 35 Annual Meeting of the Association for Computational Linguistics/8th Conference of t European Chapter of the Association for Computational Linguistics*, pp. 182-189.

Garside, R.G.; Leech, G.N.; Sampson, G.R. (eds.) (1987). The computational analysis of Englis a corpus-based approach. *Longman*, London.

González Collar, A.L.; Goñi Menoyo, J.M.; González Cristóbal, J.C. (1995). Un analizad morfológico para el castellano basado en chart. En *Actas de la VI Conferencia de Asociación Española para la Inteligencia Artificial (CAEPIA'95)*, Alicante, pp. 343-352.

Graña Gil, J.; Alonso Pardo, M.A.; Valderruten Vidal, A. (1994). Análisis léxico no determinist etiquetación eficiente del lenguaje natural. *Technical Report* 16, Departamento

rown University, Providence, RI.

oft, J.E.; Ullman, J.D. (1979). Introduction to automata theory, languages and omputations. *Addison-Wesley*, Reading, MA.

emin, C.; Klavans, J.L.; Tzoukermann, E. (1997). Expansion of multi-words terms for ndexing and retrieval using morphology and syntax. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics/8th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 24-31.

k, F. (1976). Continuous speech recognition by statistical methods. *IEEE*, vol. 64, p. 532-556.

k, F. (1985). Markov source modeling of text generation. In J.K. Skwirzynski (ed.), *The Impact of Processing Techniques on Communications*, E91 of *NATO ASI series*, p. 569-598. Dordrecht: M. Nijhoff.

k, F. (1997). Statistical methods for speech recognition. *The MIT Press*, Cambridge, MA.

k, F.; Bahl, L.R.; Mercer, R.L. (1975). Design of a linguistic statistical decoder for the ecognition of continuous sppech. *IEEE Transactions on Information Theory*, vol. 21, p. 250-256.

son, F.; Voutilainen, A.; Heikkilä, J.; Anttila, A. (1995). Constraint grammar: a language-ndependent system for parsing unrestricted text. *Mouton de Gruyer*, Berlin.

unen, L.; Beesley, K. (1992). Two-level rule compiler. *Technical Report* ISTL-92-2, Xerox, alo Alto Research Center, CA.

e, A. (1997). Finite states transducers approximating hidden Markov models. n *Proceedings of the 35th Annual Meeting of the Association for Computational inguistics/8th Conference of the European Chapter of the Association for Computational inguistics*, pp. 460-467.

S.; Simmons, R.F. (1963). A computational approach to grammatical coding of English ords. *Journal of the Association for Computing Machinery*, vol. 10, pp. 334-347.

nnimi, K. (1983). Two-level morphology: A general computational model for word-form ecognition and production. *Publications* 11, Department of General Linguistics, University f Helsinki.

c, J. (1992). Robust part-of-speech tagging using a Hidden Markov Model. *Computer peech and Language*, vol. 6, pp. 225-242.

c, J. (1993). MURAX: A robust linguistic approach for question answering using an on-line ncyclopedia. In *Proceedings of the 16th Annual International ACM/SIGIR Conference on esearch and Development in Information Retrieval*, pp. 181-190.

son, S.E.; Rabiner, L.R.; Sondhi, M.M. (1983). An introduction to the application of the heory of probabilistic functions of a Markov process to automatic speech reconigtion. *Bell*

illustrating coupling of tests in chains. In *Proceedings of the Academy of Scienc St. Petersburg*, vol. 7 of VI, pp. 153-162.

Marques, N.C.; Pereira Lopes, G. (1996). A neural network approach to part-of-speech taggin In *Proceedings of the XIII Simpósio Brasileiro de Inteligência Artificial (SBIA-96)*, Ciuriti (Brasil).

Màrquez, L.; Padró, L. (1997). A flexible POS tagger using an automatically acquired langua model. In *Proceedings of joint ACL/EACL-97*, Madrid.

Màrquez, L.; Rodríguez, H. (1997). Automatically acquiring a language model for POS taggi using decision trees. In *Proceedings of the International Conference on Recent Advances Natural Language Processing (RANLP-97)*, Tzigov Chark (Bulgaria).

Marshall, I. (1987). Tag selection using probabilistic methods. In R. Garside, G. Leech a G. Sampson (eds.), *The computational analysis of English: a corpus-based approach*, p 42-65. Longman, London.

McMahon, J.G.; Smith, F.J. (1996). Improving statistical language model performance wi automatically generated word hierarchies. *Computational Linguistics*, vol. 22, pp. 217-24

Merialdo, B. (1994). Tagging English text with a probabilistic model. *Computational Linguisti* vol. 20, pp. 155-171.

Mikheev, A. (1997). Automatic rule induction for unknown-word guessing. *Computatio Linguistics*, vol. 23(3), pp. 405-423.

Miller, D.; Leek, T.; Schwartz, R. (1999). A hidden Markov model information retrieval syste In *Proceedings of the 22nd. Annual International ACM/SIGIR Conference on Research a Development in Information Retrieval*, pp. 214-221.

Mohri, M. (1995). On some applications of finite-state automata theory to natural langua processing. *Natural Language Engineering*, Cambridge University Press, vol. 1(1), pp. 1-2

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computatio Linguistics*, vol. 23(2), pp. 269-311.

Moreno Sandoval, A. (1991). Un modelo computacional basado en la unificación para el anál y la generación de la morfología del español. *Tesis Doctoral*, Departamento de Lingüístic Lenguas Modernas, Lógica y Filosofía de la Ciencia, Universidad Autónoma de Madrid.

Moreno Sandoval, A.; Goñi Menoyo, J.M. (1995). GRAMPAL: A morphological model a processor for Spanish implemented in Prolog. In M. Sessa and M. Alpuente (eds *Proceedings of the Joint Conference on Declarative Programming (GULP-PRODE'9* Marina di Vietri (Italy), pp. 321-331.

Oflazer, K.; Tür, G. (1996). Combining Hand-crafted Rules and Unsupervised Learning Constraint-based Morphological Disambgiguation. *Departament of Computer Engineeri*

Universidad de Barcelona.

er, L. (1989). A tutorial on hidden Markov models and selected applications in speech ecognition. In *Proceedings of the IEEE*, vol. 77, pp. 257-286.

er, L.; Juang, B.H. (1993). Fundamentals of speech recognition. *Prentice-Hall*, Englewood liffs, NJ.

haw, L.A.; Marcus, M.P. (1994). Exploring the statistical derivation of transformational ules sequences for part-of-speech tagging. In *The Balancing Act. Proceedings of the Vorkshop*, Association of Computational Linguistics, pp. 86-95, Morristown, NJ.

redo, J. A. (2000) Diseño e implementación de un entorno de eliminación de ambigúedades éxicas basado en restricciones sintácticas. *Proyecto Fin de Carrere en Ingeniería nformática*.

z, D. (1992). Minimization of acyclic deterministic automata in linear time. *Theoretical Computer Science*, vol. 92(1), pp. 181-189.

d, E.S.; Thomas, R.G. (1997). Hierarchical non-emitting Markov models. In *Proceedings of he 35th Annual Meeting of the Association for Computational Linguistics/8th Conference f the European Chapter of the Association for Computational Linguistics*, pp. 381-385.

e, E.; Schabes, Y. (1995). Deterministic part-of-speech tagging with finite-state transducers. Computational Linguistics, vol. 21(2), pp. 227-253.

e, E.; Schabes, Y. (1997). Finite-state language precessing. *The Massachusetts Institute of Technology*, pp. 14-63.

ll, G.; Petitpierre, D. (1995). MMORPH - The Multext Morphology Program, version 2.3. *MULTEXT deliverable report*.

n, G.; Thorpe, R.W. (1962). An approach to the segmentation problem in speech analysis nd language translation. In *Proceedings of the 1961 International Conference on Machine Translation of Languages and Applied Language Analysis*, vol. 2, pp. 703-724.

elsson, C. (1993). Morphological tagging based entirely on Bayesian inference. In *Proceedings of the 9th Nordic Conference on Computational Linguistics*, Stockholm niversity, Stockholm, Sweden.

elsson, C.; Tapanainen, P.; Voutilainen, A. (1996). Inducing constraint grammars. In *Proceedings of the Third International Colloquium on Grammatical Inference*.

elsson, C.; Voutilainen, A. (1997). Comparing a linguistic and a stochastic tagger. n *Proceedings of the 35th Annual Meeting of the Association for Computational inguistics/8th Conference of the European Chapter of the Association for Computational inguistics*, pp. 246-253.

ez, F;, Porta, J.; Sancho, J.L.; Nieto, A.; Ballester, A.; Fernández, A.; Gómez, J.; Gómez, L.; Raigal, E.; Ruiz, R. (1999). La anotación de los corpus CREA y CORDE. *Revista de la Sociedad Española para el Procesamiento del Lenguaje Natural*, vol.

Proceedings of the 32nd Annual Meeting of the Association for Computational Linguisti pp. 181-187.

Smeaton, A.F. (1992). Progress in the application of natural language processing to informati retrieval tasks. *The Computer Jorunal*, vol. 35, pp. 268-278.

Stolcke, A.; Omohundro, S. (1993). Hidden Markov model induction by Bayesian model mergin In S.J. Hanson, J.D. Cowan and C. Lee Giles (eds.), *Advances in Neural Informati Processing System*, vol. 5, pp. 11-18. Morgan Kaufmann, San Mateo, CA.

Stolcke, A.; Omohundro, S. (1994). Best-first model merging for hidden Markov model inductio *Technical Report* TR-94-003, International Computer Science Institute, University California at Berkeley.

Stolz, W.S.; Tannenbaum, P.H.; Carstensen, F.V. (1965). A stochastic approach to t grammatical coding of English. *Communications of the ACM*, vol. 8, pp. 399-405.

Strzalkowski, T. (1995). Natural language information retrieval. *Information Processing Management*, vol. 31, pp. 397-417.

Tapanainen, P.; Voutilainen, A. (1994). Tagging accurately - Don't guess if you know. *Proceedings of the Fourth Conference on Applied Natural Language Processing*, Stuttgar

Triviño Rodríguez, J.L. (1995). SEAM - Sistema experto para análisis morfológico. *Maste thesis*, Universidad de Málaga.

Triviño Rodríguez, J.L.; Morales Bueno, R. (2000). A Spanish POS tagger with variable memo In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT-200* ACL/SIGPARSE, Trento (Italy), pp. 254-265.

Vilares Ferro, M.; Valderruten Vidal, A.; Graña Gil, J.; Alonso Pardo, M.A. (1995). U approche formelle pour la gènération d'analyseurs de langages naturels. In P. Blache (ed *Proceedings of TALN'95*, Marseille (France), pp. 246-255.

Vilares Ferro, M.; Graña Gil, J.; Cacheda Seijo, F. (1996a). Verification of morphologic analyzers. In *Proceedings of DIALOGUE'96*, Moscow (Russia), pp. 69-72.

Vilares Ferro, M.; Graña Gil, J.; Pan Bermúdez, A. (1996b). Building friendly architectur for tagging. *Revista de la Sociedad Española para el Procesamiento del Lenguaje Natur* vol. 19 (Septiembre), pp. 127-132.

Vilares Ferro, M.; Graña Gil, J.; Alvariño Alvariño, P. (1997). Finite-state morphology a formal verification. In B.K. Boguraev, R. Garigliano, J.I. Tait and A. Kornai (eds.), *Jour of Natural Language Engineering, special issue on Extended Finite State of Langua* vol.2(4) (December), pp.303-304. Cambridge University Press.

Vilares Ferro, M.; Graña Gil, J.; Araujo, T.; Cabrero, D.; Diz, I. (1998). A tagger environme for Galician. In *Proceedings of Workshop on Language Resources for European Minor*

yntactic parser of English. In Fries, Tottie and Schneider (eds.), *Creating and using English anguage corpora*. Rodopi.

ainen, A. (1995). A syntax-based part-of-speech analyser. In *Proceedings of the 7th Conference of teh European Chapter of the Association for Computational Linguistics*, Dublin.

hedel, R.; Meteer, M.; Schwartz, R.; Ramshaw, L.; Palmucci, J. (1993). Coping with mbiguity and unknown through probabilistic models. *Computational Linguistics*, vol. 19, p. 359-382.

l, J.; Daelemans, W. (1999). Recent advances in memory-based part-of-speech tagging. n *Actas del VI Simposio Internacional de Comunicación Social*, Centro de Lingüística plicada, Ministerio de Ciencia y Medio Ambiente, Santiago de Cuba, pp. 590-597.