UN MODELO COMPUTACIONAL BASADO EN LA UNIFICACIÓN PARA EL ANÁLISIS Y GENERACIÓN DE LA MORFOLOGÍA DEL ESPAÑOL

Antonio Moreno Sandoval

Tesis doctoral
Dpto. de Lingüística, Lenguas Modernas, Lógica y Filosofía de la Ciencia
Facultad de Filosofía y Letras
Universidad Autónoma de Madrid
Director: Francisco Marcos Marín
Catedrático de Lingüística General

R.B.C. 54203
BC-TTL-PL-110
UNIVERSIDAD AUTÓNOMA DE MADRID

Trabajo de Investigación presentado por
D. Antonio Moreno Sandoval
ante la Facultad de Filosofia y Letras
de la Universidad Autónoma de Madrid
para la obtención del grado de Doctor,
realizado bajo la dirección del
Dr. D. Francisco Marcos Marín,
Catedrático de Lingüística General

Vº Bº

Fdo: Francisco Marcos Marín

Contenido

Objetivos de la Tesis

La Lingüística Computacional, como disciplina aplicada, aporta a nuestro juicio los siguientes beneficios a la Lingüística, entendida como ciencia general del lenguaje:

- Obliga a formalizar explícitamente los conocimientos del lingüista.
- Sirve de banco de prueba de las hipótesis y teorías lingüísticas.
- Aumenta el número de expectativas laborales de los lingüistas, al tiempo que contribuye a aportar aplicaciones prácticas a la sociedad. (Beneficio social).

Esta disciplina se caracteriza por la interrelación entre especialistas en el mundo de los ordenadores y en las lenguas naturales. La relación lingüista-informático debe estar basada en el reconocimiento mutuo de que tanto uno como el otro son especialistas en su campo, y evitar así las intromisiones en las competencias del otro. Básicamente la tarea del lingüista es dar de forma explícita el modelo que tiene que ser implementado (en este caso una gramática). El informático tiene que decidir cómo debe trasladar ese modelo, a través de programas, al ordenador para que sea capaz de aplicar ese conocimiento a un problema concreto (por ejemplo, analizar e interpretar una oración).

Por supuesto, es útil -y necesario- que ambos tengan conocimientos generales de la otra disciplina, pues contribuirá a una mejor comunicación y comprensión de los problemas. Lo que parece no muy recomendable es que ambas funciones recaigan sobre la misma persona. En primer lugar, por motivos de especialización: es obvio que la riqueza teórica de ambas ciencias requiere bastantes años de esfuerzo para adquirir conocimientos sólidos, lo que hace bastante costoso en tiempo la especialización en ambos campos. En segundo lugar, por motivos prácticos: dos cabezas descubren mejor los problemas que se han pasado por alto que una sola, contemplar los problemas desde perspectivas diferentes ayuda a resolverlos de una forma mucho más consistente. Un ejemplo de esto último nos lo proporcionan los primeros sistemas en lengua natural. Fueron construidos en los años 50 y 60 por especialistas en informática que confiaron en su conocimiento de la lengua como hablantes. Muchos de los problemas que se abordaron de una forma incorrecta (por ejemplo, en traducción automática (Marcos et al. 1989)) se podían haber planteado de una manera más eficiente por un lingüista. En los años 80 se ha visto la incorporación efectiva de lingüistas a los proyectos en lengua natural, y se ha reconocido su papel de expertos: la complejidad de las lenguas naturales, a medida que los proyectos se han ido haciendo más ambiciosos, ha obligado a mejorar los conocimientos sobre ellas. Lo que al principio se planteó como resolución de problemas muy sencillos (por ejemplo, interrogar a una base de datos basándose en las palabras claves y desechando el resto) se ha ido complicando (ahora se pretende interpretar toda la oración). Este reconocimiento de la labor del lingüista, en buena medida, se debe al gran desarrollo que ha experimentado la Lingüística en los últimos 30 años como ciencia específica del lenguaje. Además, ha contribuido el hecho de que bastantes lingüistas se hayan interesado por los ordenadores, aunque sólo sea por aplicaciones muy concretas como el procesamiento de textos o bases de datos.

Por otra parte, con la aparición de los lenguajes de programación *declarativos* así como el desarrollo de lenguajes de usuario orientados a aplicaciones lingüísticas, el acercamiento de los lingüistas y demás especialistas en áreas del conocimiento a la programación es un hecho constatable en los últimos años. Lenguajes como Prolog o LISP, sin ser herramientas específicamente diseñadas para el procesamiento del lenguaje natural, están cerca de lo que un lingüista necesita para implementar su formalización de los fenómenos lingüísticos. Esto ha conducido a que en la actualidad la mayoría de los proyectos en desarrollo y sistemas en uso utilicen dichos lenguajes de programación (o formalismos muy cercanos a ellos) para desarrollar la parte lingüística (es decir, la gramática y el diccionario), y dejar las tareas propias del procesamiento (es decir, el *parser*, el segmentador, la búsqueda en el diccionario, el entorno de usuario, etc.) a lenguajes procedurales que mejoran, en general, la eficiencia del sistema. En esta nueva partición del trabajo, parece que la misión del lingüista es escribir gramáticas y diccionarios y la del programador mejorar el tiempo de procesamiento y el aspecto y la facilidad de uso del sistema (*user-friendly system*).

Implicaciones teóricas de la tesis

Desde el advenimiento de la gramática generativa, uno de los objetivos básicos e ineludibles de cualquier propuesta teórica lingüística es desarrollar un mecanismo formal (es decir, una gramática formal) que genere el conjunto infinito de representaciones que dan cuenta del conjunto infinito de oraciones de cualquier lengua natural. Y el primer punto que hay que definir es cuál es ese mecanismo formal. Haciendo un poco de historia, los primeros trabajos de Chomsky intentaron demostrar que las gramáticas sintagmáticas eran insuficientes para esta labor, y que era necesario, por tanto, un mecanismo formal más poderoso (las transformaciones). Esta propuesta, que fue seguida por la mayoría de la comunidad científica en los años 60, empezó a cuestionarse a finales de los 70 y en la década de los 80 la búsqueda de alternativas a las transformaciones se ha convertido en uno de los temas de moda. En cualquier caso, hay que destacar el cambio radical de postura desde la unanimidad inicial hasta la diversidad actual, claro exponente de que algo está cambiando y de que, tal vez, hay que plantearse el problema desde otro punto de vista. De hecho, las propuestas de las teorías y formalismos basados en la unificación insisten en reducir el poder formal de las gramáticas sobre el principio de que de esta manera se reduce el tiempo de procesamiento en la comunicación humana. Esta tesis pretende colaborar en la discusión presentando una gramática sintagmática independiente del contexto para el tratamiento computacional de los fenómenos morfosintácticos del castellano.

Por otra parte, aunque la polémica sobre la adecuación formal de las gramáticas se haya centrado en gramáticas transformacionales frente a gramáticas sintagmáticas independientes del contexto lo cierto es que para el caso concreto de la morfología teórica las gramáticas dependientes del contexto han dado resultados satisfactorios. Si a esto unimos que desde la tesis de Koskemiemi se ha demostrado que se puede tratar informáticamente la morfología con un modelo de estados finitos se comprenderá por qué consideramos necesario comparar con cierto detenimiento las distintas aproximaciones, aunque no sea el objetivo fundamental establecer una comparación rigurosa.

Como refleja el título, la base teórica de esta tesis descansa sobre los formalismos de unificación y rasgos. El auge experimentado por estos formalismos en los últimos diez años tanto en Lingüística Teórica como, sobre todo, en procesamiento de lenguas naturales, es un indicador de su capacidad para tratar los fenómenos lingüísticos y, más en general, los procesos cognitivos que implican información lingüística.

Por último, nos parece interesante que una un trabajo sobre morfología trate, aunque no sea directamente, de la pertinencia de los tres modelos propuestos por Hockett (*Palabras y Paradigma, Elementos y Colocación, Elementos y Proceso*). Gracias a la nueva perspectiva que nos proporcionan las gramáticas de unificación presentaremos una aproximación distinta al modelo *Elementos y Colocación.*

Sobre la cobertura lingüística

Hemos escogido el campo de la flexión sin entrar en los terrenos (aunque teóricamente más interesantes) de la derivación y composición. Téngase en cuenta que por motivos prácticos (y la Lingüística Computacional es una ciencia aplicada) las implementaciones informáticas deben basarse en conocimientos sólidamente contrastados en la teoría, so pena de invertir más tiempo de lo deseable en elaborar formalmente los fenómenos que se describen. Es ciertamente arriesgado aventurarse en desarrollar aplicaciones informáticas cuando la teoría no ha encontrado todavía soluciones aceptadas unánimemente (y la historia de la lingüística computacional tiene bastantes pruebas de fracasos por ignorar este hecho). En concreto, creemos que no existe en la actualidad ninguna propuesta teórica para los fenómenos derivativos y compositivos que dé cuenta de ellos de forma global y generalizadora (aunque por supuesto hay soluciones parciales). Esto se debe, sin duda, a que su tratamiento implica fuertes interrelaciones con el gran reto de la lingüística (y otras disciplinas "cognitivas" como la lógica, la inteligencia artificial y la psicología cognitiva): formalizar adecuadamente las bases interpretativas del lenguaje. Por tanto, este aspecto se escapa de los objetivos de la tesis.

Debemos señalar que esta tesis se concentra exclusivamente en el procesamiento de la forma gráfica (escrita) de las palabras, sin tener en cuenta la forma fonológica (hablada) de las palabras. Esto es una restricción sobre los datos lingüísticos que se acepta y reconoce en la mayoría de los sistemas de procesamiento del lenguaje natural (salvo, como es obvio, en los de reconocimiento del habla).

Un punto muy importante en nuestros objetivos es conseguir la *declaratividad* o *bidireccionalidad* en nuestra gramática. Es decir, pretendemos que con la misma formalización se pueda tanto analizar como generar palabras del español. Nos proponemos como meta que la solución del problema (es decir, el programa) se parezca en todo lo posible a la especificación del problema (la gramática). La declarativiidad se manifiesta, por tanto, como un requisito teórico y práctico.

Por último y en consecuencia con el carácter predominantemente aplicado de esta tesis, aportamos como prueba documental de nuestra gramática un pequeño programa de demostración (GRAMPAL) con los casos más significativos de la morfología flexiva del español.

Agradecimientos

Tengo que reconocer públicamente la influencia decisiva que ha supuesto para mi formación como lingüista computacional y el desarrollo de la tesis el haber trabajado en los proyectos EUROTRA (traducción automática, CEE) y SILVIA (consulta de bases de datos, IBM Suecia). Quiero manifestar mi agradecimiento sincero a todas aquellas personas que he conocido durante estos años en relación con dichos proyectos y en especial a Francisco Marcos Marín, Pilar Salamanca Fernández, Juan Carlos Moreno Cabrera, Fernando Sánchez León, Peter Lau, Dieter Maas, Ernesto Gil, Jane Brown, Brian White, Consuelo Rodríguez Magro y Luis de Sopeña Pastor. Aunque suene a tópico, la persona que más me ha ayudado ha sido mi mujer, Cristina Olmeda Moreno, que además de soportarme durante las veinticuatro horas del día, ha

escrito el programa de segmentación en C (muchas gracias también por sus consejos a Jesús Sánchez), me ha sugerido mejoras importantes en el programa de demostración GRAMPAL y se ha leído cuidadosamente las distintas versiones de esta tesis.

Por último, un agradecimiento especial al *Centro de Investigación UAM-IBM*, que ha dado todas las facilidades para usar su biblioteca y medios de impresión.

Fundamentos teóricos de la tesis

La primera parte de la tesis se dedicará a exponer las teorías, modelos y problemas que están implicados en nuestra aportación original. Empezaremos con los fundamentos sobre teoría de los lenguajes y gramáticas formales, para pasar a continuación a resumir las características de las gramáticas de unificación y de rasgos. Habrá una pequeña sección dedicada a fundamentos informáticos, que nos servirá para situar algunos conceptos expuestos en nuestro modelo. Por último, dedicamos dos secciones más amplias a presentar las nociones fundamentales de la morfología teórica y su tratamiento computacional.

Nos excusamos por no incluir una breve introducción a la lingüística computacional, pero pensamos que hay magníficos manuales como Tennant(1981), Winograd(1983), Grishman(1986) o Gazdar y Mellish(1989a y b) que hacen del todo innecesaria dicha introducción, en especial porque parecería incompleto intentar resumir en pocas páginas la historia y conceptos de esta disciplina, amén de resultar pretencioso innovar o mejorar las presentaciones de los autores mencionados.

1.0 La jerarquía de Chomsky: el poder formal de las gramáticas

Parece fuera de toda duda que uno de los objetivos de la Lingüística como ciencia del lenguaje es proponer formalizaciones rigurosas de los fenómenos de las lenguas naturales. Para describir las características de una lengua de una manera abstracta y formal necesitamos otra lengua, llamada *metalengua* o *formalismo gramatical.* Las *gramáticas formales* tienen una utilidad evidente en Lingüística Computacional, donde sirven para establecer una caracterización que pueda ser interpretada computacionalmente.

Se puede definir una lengua como un conjunto infinito de oraciones. Cada oración se caracteriza por ser una cadena bien formada con un vocabulario finito de símbolos. "Consideraremos cada oración como una *cadena* o secuencia de cero o más elementos unidos mediante una operación de *concatenación.* Los elementos pueden ser cualquier cosa, en nuestro examen de las lenguas naturales podemos concebir que representan sonidos distintivos (fonemas) u otros elementos lingüísticos (palabras, morfemas), de acuerdo con el sector que estemos estudiando" (Bach 1976:57). El concepto de "cadena bien formada" significa que la forma de la cadena (es decir, la manera en que los símbolos se colocan uno junto al otro) no viola ningún criterio específico de las reglas de formación que contiene la gramática. La noción de gramática formal supone algo más que decir que una gramática describe una lengua. Esto es lo que hace pero de una forma muy definida.

Una gramática formal G está formada por cuatro elementos **< Vn,Vt,R,O >** donde:

- **Vn**: es un conjunto finito de símbolos no terminales. A veces se les conoce como variables. En las aplicaciones lingüísticas se corresponden con las categorías sintácticas. Es su presencia en las reglas lo que permite a una gramática expresar las condiciones de buena formación.
- **Vt**: es un conjunto finito de símbolos terminales. Estos símbolos coinciden más o menos con las palabras de una lengua, si se trata de una gramática de una lengua natural.
- **R**: es un conjunto finito de reglas, también llamadas "producciones." Tienen la forma $\alpha \rightarrow \beta$, donde α y β son cadenas de elementos de Vn y Vt.
- **O**: es el símbolo de partida. O es un elemento de Vn y tiene que aparecer al menos una vez en la parte izquierda de una regla o "producción" (es decir, en lugar de α).

Una gramática G genera una lengua L(G). Existen varios tipos de gramáticas (conocidas también como *gramáticas sintagmáticas* o *de estructura de frase*), dependiendo de la forma de las cadenas α y β en las reglas. El tipo de G determina el

tipo de L(G): la gramática de cierto tipo generan lenguas de un tipo correspondiente. Chomsky estableció una clasificación de tipos de gramáticas que se ha hecho famosa con el nombre de su creador, la *jerarquía de Chomsky*. Antes de ver dicha clasificación, conviene definir dos nociones muy utilizadas en la teoría de los lenguajes formales: *lenguas enumerables recursivamente* y *lenguas recursivas*.

La tarea del lingüista consiste en describir las lenguas humanas, es decir, un número infinito de oraciones, y hacerlo de tal manera que sea capaz de diferenciar las oraciones que son parte de la lengua que ha descrito de las que no pertenecen a esa lengua. Hay esencialmente dos formas de realizar esta tarea. Una consiste en listar todas las oraciones de esa lengua particular, de tal forma que se puede comprobar si una oración particular está incluida en la lista o no. Esta lengua sería, por tanto, *enumerable recursivamente*. Sin embargo, este método presenta serias desventajas. La primera es que conseguir todas las oraciones posibles de una lengua parece una tarea sin fin. Además, aunque consiguiéramos enumerar todas ellas, no habría forma de presentar qué es lo que tienen algunas oraciones en común.

La otra posibilidad está basada en la idea de que los humanos, que después de todo son seres con una memoria limitada, pueden decidir cuando oyen una oración si ésta pertenece o no a su lengua. Esta propiedad se les asigna a los *lenguajes recursivos*: son aquellos conjuntos de oraciones enumerables recursivamente cuyos complementos también son enumerables recursivamente, de tal forma que para conocer si un elemento pertenece a uno de los dos conjuntos (oraciones gramaticales y oraciones agramaticales) podemos enumerar cualquiera de los dos, con la seguridad de que llegaremos a saber en un tiempo finito si un elemento está en un conjunto o no. Esto sugiere que, para cada lengua, existe un número finito de criterios que cumplen cada oración y que los humanos tienen un conocimiento implícito de estos criterios cuando hacen juicios lingüísticos. Los lingüistas tratan de descubrir cuáles son estos criterios y formularlos de una manera explícita en una gramática.

Las gramáticas sintagmáticas en general (como la que hemos definido más arriba) pueden describir cualquier lenguaje enumerable recusivamente. Esto nos sugiere que son demasiado poderosas y que hay que restringir de alguna manera su capacidad generativa[1] para que un programa de ordenador sea capaz de decidir si una oración es gramatical o no, en un tiempo razonable. Desde el punto de vista del procesamiento informático de lenguas naturales (y desde el punto de vista teórico también si queremos explicar la capacidad de los hablantes de entender su lengua) parece necesario introducir algunas restricciones en nuestro formalismo para que "we can be assured that the languages generated will be recursive and, further, that (depending on the constraints) it will be easy to write efficient programs for analyzing these languages" (Grishman 1986:16).

De acuerdo con su *poder generativo débil*, hay cuatro tipos de gramáticas (llamadas *tipo 0, tipo 1, tipo 2* y *tipo 3*), cada una definida por la clase de reglas que contiene. Por cada tipo de gramática, hay una clase de lenguas que sólo pueden ser definidas por gramáticas de ese poder o de un poder superior. Es decir, hay lenguas que puede ser definidas por gramáticas de tipo 0 pero no por las de tipo 1, en cambio

---

[1] La capacidad de predicción de una gramática (es decir, la forma en que se describen explícitamente todas las oraciones de una lengua) puede ser débil o fuerte. Se dice que tienen *capacidad generativa débil* cuando las reglas predicen cómo son las oraciones generadas por una gramática. *Capacidad generativa fuerte* es cuando las reglas además pueden predecir qué estructura subyace a las oraciones que describen.

todas las de tipo 1 pueden ser definidas por las de tipo 0. Dicho de una manera más formal: las lenguas de tipo-i incluyen propiamente a las de tipo-(i + 1), cuando i = 0,1, y 2 (Hopcroft y Ullman, 1979).[2]

La tabla 1 -combinación de las que aparecen en Bach (1976) y en Winograd (1983)- muestra los cuatro tipos de gramáticas y su relación con lenguas y autómatas.

Tipo	Gramáticas	Restricciones a la forma de las reglas	Autómatas	Lenguas
0	Irrestrictas	Ninguna: $\alpha \rightarrow \beta$	Máquinas Turing	Enumerables recursivamente
1	Dependientes del contexto	La parte derecha contiene como mínimo los símbolos de la parte izquierda	Autómatas linealmente finitos	Dependientes del contexto
2	Independientes del contexto	La parte izquierda sólo puede tener un símbolo: $\alpha \rightarrow \beta$...	Autómatas PSD (*Push Down*)	Independientes del contexto
3	Regulares o de estados finitos	La regla sólo puede tener estas dos formas: A → t B, A → t	Autómatas finitos	Regulares

Tabla 1. Jerarquía de Chomsky.

Daremos una breve descripción de las reglas propias de cada tipo, basándonos en Grishman(1986) y empezando por el tipo menos poderoso:

- **Gramáticas regulares o de estados finitos** (tipo 3): pueden tener dos tipos de reglas, dependiendo de si la parte derecha de la regla lleva un símbolo terminal ($A \rightarrow t$) o de si lleva un símbolo terminal seguido de un símbolo no terminal ($A \rightarrow t\ B$). La característica más peculiar de estas gramáticas es que, mientras se está generando una oración (el proceso de pasar de un estado a otro) la única información que necesitamos conocer es el estado en que nos hallamos para escoger la continuación apropiada. Esto significa que no se tiene acceso a la información ya procesada y precisamente esto constituye su punto fuerte y débil a la vez. Por una parte, son el mecanismo más simple de computación de lenguas; por otra, no son capaces de tratar lenguas que sean autoincrustantes, es decir, lenguas que tengan elementos encorchetados del tipo $x,(x),((x))$.... Como está demostrado que las oraciones de las lenguas naturales tienen la posibilidad de aumentarse indefinidamente incorporando oraciones subordinadas, se deduce que las gramáticas regulares no son apropiadas para describir las lenguas naturales. Esto es en teoría, pero en una aplicación práctica es posible que no necesitemos analizar o generar oraciones autoincrustadas, por lo que recurrir a los autómatas de estados finitos es una forma de conseguir eficiencia computacional.

---

[2] Esto no es rigurosamente exacto, como advierten Hopcroft y Ullman: la excepción es la aparición o no de la cadena vacía. El teorema de la jerarquía que ellos presentan es: (a) las lenguas regulares (o conjuntos regulares) están contenidos en las lenguas independientes del contexto; (b) las lenguas independientes del contexto que no contengan la cadena vacía como elemento, están contenidas en las lenguas dependientes del contexto; y (c) las lenguas dependientes del contexto están contenidas en los conjuntos recursivamente enumerables. Para ver las pruebas y las relaciones entre los tipos de gramáticas, lenguas y autómatas consúltese el capítulo 9 de su *Introduction to automata theory, languages, and computation.*

- **Gramáticas independientes del contexto** (tipo 2). Todas las reglas tienen la forma $A \rightarrow x$, donde A es un símbolo no terminal y x es una secuencia de cero o más símbolos tanto terminales como no terminales. Significa que A puede ser sustituido por x en cualquier sitio en que aparezca, independientemente del contexto. Desde el punto de vista computacional, también son fáciles de manejar, a medio camino entre las gramáticas regulares y las gramáticas transformacionales (equivalentes a las máquinas Turing, en muchos casos).

- **Gramáticas dependientes del contexto** (tipo 1). La condición es que la parte derecha de la regla tenga el mismo número de elementos o más que la parte izquierda (p.ej. $AB \rightarrow CDE$). Hay dos notaciones para este tipo de reglas: $A \rightarrow y \ / \ x\_z$, o bien, $xAz \rightarrow xyz$. A pesar de que intuitivamente parece que las reglas de contexto son muy apropiadas para hacer generalizaciones sobre fenómenos lingüísticos, la historia de la Lingüística Computacional proporciona valiosos ejemplos de que es más eficiente hacerlas con gramáticas independientes del contexto aumentadas con ciertas extensiones (Grishman 1986).

- **Gramáticas irrestrictas** (tipo 0). Este tipo de gramáticas no tienen ningún tipo de restricción sobre sus reglas, es decir, pueden ser de cualquier forma (conservando, claro está, la estructura $\alpha \rightarrow \beta$). Estas gramáticas, al igual que las gramáticas transformacionales, pueden definir cualquier lengua recursivamente enumerable y, por lo tanto, pueden analizar tanto estructuras superficiales como profundas (las gramáticas sintagmáticas de menor potencia sólo pueden generar descripciones superficiales). En la práctica, tanto las gramáticas irrestrictas como las transformacionales imponen algún tipo de restricción sobre sus reglas para que el lenguaje definido sea recursivo (es decir, para que el programa sepa reconocer si la oración pertenece o no a la lengua generada).

1.1 Complejidad y decidibilidad

A continuación vamos a exponer dos conceptos básicos de la teoría de los lenguajes formales. Hemos visto hasta ahora que un tipo particular de autómata se corresponde con una clase concreta de lenguas (formales o naturales). El principio de la **complejidad** dice que cuanto más complejo sea el autómata permitido, tanto más complejas serán las lenguas reconocidas por él. Desde este punto de vista, los NDLBA (*nondeterministic linear bounded automata* o autómatas linealmente finitos) son más complejos que los NDPDA (*nonderterministic push down automata* o autómata PSD determinista), y éstos a su vez más complejos que los DPDA (*deterministic push down automata* o autómata PSD no determinista). Esto traducido a los tipos de lenguas implica que las lenguas dependientes del contexto son más complejas que las independientes del contexto (constatación, por otra parte, de una intuición general).

Dentro de los dos tipos de autómatas para lenguas independientes del contexto (los NDPDA y los DPDA), la teoría muestra que el reconocimiento es mucho más fácil y rápido si utilizamos el modelo determinista. De igual forma, la teoría dice que cualquier formalización de las lenguas naturales mediante una gramática independiente del contexto no puede ser determinista, puesto que se reconoce en general que la estructura superficial de las lenguas naturales es ambigua, esto es, que no se puede determinar de antemano. En consecuencia, cuanto más ambicioso, lingüísticamente hablando, pretenda ser un sistema en lengua natural tendrá que utilizar mecanismos no deterministas, aún a costa de una complejidad mayor. A pesar de este presupuesto teórico (pero no en contradicción con él) muchos sistemas utilizan gramáticas determinísticas, reconociendo su inadecuación lingüística. Esto se debe a que dichos

sistemas trabajan con un fragmento muy restringido de una lengua natural, que permite el uso determinista y aumentar así la eficiencia del programa (reduciendo su complejidad).

Por último, hay que señalar que el mecanismo de reconocimiento más simple es el autómata de estados finitos y, por tanto, el más eficiente en teoría. Las propiedades de estos autómatas son bien conocidas (en gran parte debido a su simplicidad) pero en la práctica son menos utilizados que los otros autómatas (concretamente, menos que los *push down*). Su uso más extendido dentro de los sistemas en lengua natural es en la fase léxica (búsqueda de los límites de palabras y entradas en el diccionario). Cuando analicemos el modelo en dos niveles de Koskenniemi volveremos a tratar la simplicidad y eficiencia de los autómatas de estados finitos.

Otra colección de resultados teóricos tiene que ver con cuestiones de **decidibilidad** o resolubilidad (el término inglés es *decidability*) Existen distintas pruebas acerca de determinados problemas, que demuestran que dichos problemas no se pueden resolver, o más precisamente, que no se puede "decidir" sobre esas cuestiones. El caso más conocido el de demostrar que dos lenguas independientes del contexto son iguales (es decir, generan la misma lengua). Se ha demostrado que no existe un algoritmo que pueda determinar si $L(G_1) = L(G_2)$. La conclusión pertinente es que la ausencia de decidibilidad para un problema en particular es un aviso para que no se intente resolver el problema. Esto tiene gran importancia práctica ya que en principio en el diseño de programas no debería aparecer la solución de un problema "indecidible." Pero a pesar de lo que dice la teoría, como ya vimos con el principio de complejidad, en casos particulares se puede hacer caso omiso de sus recomendaciones. Benson (1979) lo expresa muy certeramente en estas líneas:

> But even in cases like the weak equivalence of grammars, two particular grammars under consideration may be sufficiently related and understood that, in that particular case, the weak equivalence can be decided. *The general undecidability simply says that there is no general algorithm which will work in all cases*[3]

Resumiendo, cuanto más compleja y poderosa (en el sentido de poder expresivo) es una gramática menos eficiente y menos se conocen sus propiedades matemáticas. Esto implica que depende de los objetivos que se marque el lingüista teórico o el computacional, puede escoger un tipo u otro de gramática e intentando compensar las ventajas y desventajas de cada tipo: los autómatas finitos son rápidos y sencillos pero son insuficientes para tratar fenómenos de las lenguas naturales; por el contrario, las máquinas de Turing son muy poderosas y pueden tratar cualquier fenómeno, pero no son eficientes. La idea de combinar estas propiedades ha llevado a los investigadores a buscar soluciones intermedias introduciendo cambios en los formalismos:

> Some notational restrictions on grammar formalisms can have the effect of seriously limiting the class of grammars that can be expressed. Conversely, apparently minor notational changes can radically increase the potential mathematical power of the systems characterized. (Gazdar y Mellish 1989:103).

---

[3] El subrayado es nuestro. Cuando Benson habla de la débil equivalencia de dos gramáticas se refiere a una aplicación concreta del problema de la igualdad de dos lenguas independientes del contexto, ya que dicha igualdad puede ser tanto en poder generativo débil como fuerte.

1.2 La adecuación formal de las gramáticas

Una de las cuestiones que más ha interesado a los lingüistas en las últimas décadas es cuál es el tipo de gramática formal más adecuado para describir las lenguas naturales, en el sentido de cuál debe ser su poder expresivo. Hay opiniones para todos los gustos, ya que son muchas las razones que intervienen a la hora de inclinarse por un tipo o por otro. Por otra parte, ya hemos comentado en el apartado anterior que casi no existen tipos puros de gramáticas, pues se han ido modificando según las necesidades particulares. Esto hace que la cuestión se complique un poco más: si la gramática ha sufrido alguna variación, ya sea un restricción de su poder ya sea una ampliación notacional de su poder, no se puede decidir fácilmente si dicha gramática pertenece a un tipo o a otro. Consecuentemente, sus propiedades matemáticas son más difíciles de conocer. Un caso muy conocido son las gramáticas transformacionales: nunca se ha sabido muy bien cuáles eran sus propiedades matemáticas, aunque se han dado pruebas de que eran equivalentes a las máquinas de Turing.

Un hecho que hay que tener en cuenta es que tampoco todos los lingüistas están motivados de la misma forma por las propiedades formales de las gramáticas que escriben. Noam Chomsky, por ejemplo, que empezó desarrollando la teoría de las lenguas formales, ha ido dejando paulatinamente de interesarse por esta cuestión, para concentrarse en otro tipo de adecuación gramatical (la explicación de cómo un niño aprende su lengua, lo que se conoce como *adecuación explicativa*, frente a la *adecuación observativa* o explicación de si las propiedades matemáticas de una gramática se adaptan a las propiedades que manifiestan las lenguas). Por el contrario, Gerald Gazdar está muy interesado esta última adecuación. Y, cómo no, también hay partidarios de no prestar mucha atención a las propiedades formales y dedicarse a conseguir que la caracterización que ellos hacen sirva para interpretar y generar oraciones. Entre estos últimos debemos señalar a Martin Kay:

> Just recently, there has been a partial and grudging retreat from the view that to formalize is to explain. This has not been because of any widespread realization of the essential vacuity of purely formal explanations, but for [...] the failure of the formalists to produce workable criteria on which to distinguish competing theories [...]. The search for sources of constraint to impose on formal grammar has led to an uneasy alliance with the psychologists and a belated rekindling of interest in parsing and other performance issues. [...] My aim in this polemic is not to belittle the value of formalisms. Without them linguistics, like most other scientific enterprises, would be impotent. It is only to discredit them as an ultimate basis for the explanation of the contingent matters. (Kay 1985:252)

Vamos a comentar brevemente los argumentos que se exponen para escoger un tipo u otro. Empezaremos hablando de las gramáticas transformacionales y luego se contrastarán con las sintagmáticas independientes del contexto. Se dedicará un pequeño espacio a explicar por qué las gramáticas contextuales no son muy apreciadas por los lingüistas computacionales y, por último, se hablará también de las propuestas desde el campo informático de tratar las lenguas con autómatas finitos.

1.2.1 Gramáticas transformacionales frente a gramáticas sintagmáticas

Como es bien conocido, en *Estructuras sintácticas* Chomsky fue el primero en declarar que la misión del lingüista no era establecer simplemente clasificaciones taxonómicas de *corpora* lingüísticos, sino escribir gramáticas que den cuenta lo más rigurosa y explícitamente posible de las intuiciones de gramaticalidad de los hablantes. Esto condicionó en gran medida su interés por las propiedades matemáticas de los formalismos gramaticales. Elaboró su conocida jerarquía de tipos de gramáticas y dio pruebas formales acerca de qué tipo de oraciones podían o no generar cada clase de gramática. Con estas pruebas, Chomsky intentó demostrar la inadecuación de ciertas gramáticas basándose en que las lenguas naturales tenían construcciones que sobrepasaban su capacidad generativa. Los dos casos más conocidos son la inadecuación de las gramáticas regulares para tratar las oraciones (o elementos) incrustados y la inadecuación de las gramáticas sintagmáticas (en general) para resolver ambigüedades y relacionar oraciones superficialmente distintas pero con la misma interpretación. En este último caso le indujo a crear otro tipo de gramáticas, dotadas de reglas más poderosas (las transformaciones), que permitiera dar cuenta de las regularidades que se dan más allá del nivel puramente superficial.[4]

Durante la época de la Teoría Estándar las transformaciones tuvieron su máximo auge ya que no sólo servían para distinguir las oraciones gramaticales de las agramaticales sino que también se utilizaron para conectar el nivel de la interpretación fonológica (la estructura superficial) con el nivel de la interpretación semántica (la estructura profunda). En esta época se produjo un cambio radical con respecto al tipo de adecuación necesaria para una gramática: se pasó del interés por las propiedades matemáticas (adecuación "observativa") a concentrarse en cuestiones psicológicas (cómo pueden los hablantes de una lengua relacionar sonidos y significado) y semánticas (adecuación "descriptiva"). En consecuencia, el rigor en la explicitud de la gramática se relajó. Pero al mismo tiempo, se hizo evidente que era necesario restringir el poder de las transformaciones. Varios estudios, entre ellos el de Peters y Ritchie(1973), demostraron que las gramáticas transformacionales de aquella época tenían el mismo poder que las máquinas de Turing, lo que significaba con dichas gramáticas se podía formalizar todo lo que se quisiera formalizar (y fuera formalizable). Los tiempos estaban cambiando de nuevo y Chomsky y sus discípulos empezaron a concentrarse en lo que se conoce como la *adecuación explicativa*: cómo explicar el aprendizaje de la lengua por parte de un niño, sobre la base de los datos que recibe. Esta teoría (fuertemente psicologista) no podía admitir el poder formal de las gramáticas transformacionales existentes, que permitían infinitos análisis del mismo fenómeno.

En la fase actual (*Government and Binding* GB), dominada por la adecuación explicativa, las transformaciones se han reducido al máximo: sólo existe una de hecho, "Muévase-α." El interés ahora está en encontrar un conjunto reducido de principios

[4] Las gramáticas sintagmáticas sólo son capaces de especificar la estructura de una oración, pero no de asignarles una estructura subyacente que la relacione con otras oraciones con interpretación similar. El ejemplo clásico de Chomsky son las oraciones activas y pasivas: en aquella época, consideraba que ambas tenían la misma estructura profunda aunque estructuras superficiales diferentes. Con una gramática sintagmática no es posible dar cuenta de la estructura profunda, por lo que utilizó las transformaciones para conectar ambos niveles de representación.

universales y establecer parámetros de variación entre las lenguas. Wasow(1986) resume certeramente la evolución de la Gramática Generativa:

> With the concentration on learnability and universal grammar, many details in the analyses of particular constructions began to receive less attention. Likewise, concern for explicitness and formalization diminished. Indeed, there seems to be quite a general trade-off between theoretical elegance and attention to empirical detail.

Paralelamente a esta evolución, otros lingüistas (en especial Gazdar y Bresnan, con sus respectivos colaboradores) empezaron a desmarcarse de las propuestas de Chomsky e intentaron volver al rigor empírico y ha interesarse de nuevo por las propiedades formales de las gramáticas. Tanto GPSG (*Generalized Phrase Structure Grammar*) como LFG (*Lexical Functional Grammar*) han eliminado el componente transformacional de sus formalismos. La motivación más clara fue el hecho observado en muchas de las transformaciones de la Teoría Estándar generativista de que tales transformaciones producían eductos estructuralmente idénticos a los generados por los árboles de la base sintagmática: "if deep structures and surface estructures were isomorphic, it was reasoned, then why relate them by means of rules with the power to alter structure?" (Wasow 1986:201).

Pero además de estas consideraciones formales, los partidarios de las gramáticas sintagmáticas exponen también motivaciones psicológicas. Concretamente, recurren al famoso experimento de Marslen-Wilson.[5]

Marslen-Wilson denominó a su experimento *shadowing* (sombra o repetición). Consistía en pedir a un sujeto que repitiera lo que estaba escuchando en una cinta con la menor demora posible. La diferencia entre el estímulo y la repetición era de dos o tres sílabas como media, bajando hasta una sílaba en algunos individuos. El interés del experimento reside en analizar lo que ocurre cuando hay alguna anormalidad o error en el estímulo o en la respuesta del hablante. Marslen-Wilson argumenta que, al menos implícitamente, los lingüistas generativistas aceptan un modelo de *proceso del lenguaje en serie*: es decir, primero se haría un análisis fonológico de la cadena sonora, después se procesarían fonemas para formar palabras y una vez que se hubieran reconocido bastantes palabras, se procesarían éstas sintácticamente. El número de palabras reconocidas debe ser bastante alto, puesto que las transformaciones operan de una forma global, mucho más que local. Por otra parte, en el experimento se comprobó que los errores espontáneos de los hablantes eran sintáctica y semánticamente congruentes con lo que les precedía en un 97% de los casos, tanto si estos fallos se producían al principio de la oración como si lo hacían más tarde.

La conclusión de Marslen-Wilson ataca el "modelo en serie": parece como si cada palabra que procesa el hablante pasa por todos los niveles de descripción y se interpreta en todos esos niveles en función de la información que está disponible en ese punto del proceso de la oración. Es decir, en lugar de una relación jerárquica y estratificada es más natural pensar en un sistema interactivo *on-line* organizado de forma no jerárquica, sin una cadena estricta de niveles informativos. La opinión de Sampson(1983) sobre el experimento de Marslen-Wilson es que el hablante está formulando continuamente "predicciones" acerca de lo que está oyendo y utiliza el *input* para eliminar las suposiciones incorrectas, más que para forzar las correctas.

---

[5] Lo que sigue lo hemos tomado de Sampson(1983): "Context-free parsing and the adequacy of context-free grammars."

El experimento de Marslen-Wilson también tuvo implicaciones sobre la necesidad de tener un nivel profundo de interpretación. Si el papel de la estructura semántica es el de servir de mediadora entre la estructura superficial y la forma semántica o el contenido de la oración, los hablantes del experimento no podrían haber cometido errores influidos por la semántica, como en realidad ocurría. Por lo tanto, si la estructura profunda no es un paso en el proceso de la oración ¿qué justificación puede tener en Lingüística Teórica?

Gazdar usa las ideas de Marslen-Wilson para justificar la propuesta de que no se deben definir las lenguas naturales gramáticas transformacionales, sino con gramáticas sintagmáticas basadas en la estructura superficial.

Para acabar con la breve exposición de la polémica entre gramáticas transformacionales y sintagmáticas mencionaremos las implicaciones que ha tenido en el campo de la Lingüística Computacional.

Durante el auge de las transformaciones (de mediados de los 60 a mediados de los 70) se desarrollaron unos sistemas especiales conocidos por *redes de transición aumentadas* ('augmented transition networks' ATN): "Woods and others developed ATNs essentially as a programming language for writing analyzers where the undoing of transformations could be carried out during processing" (Gazdar y Mellish 1989:95). Estas redes eran básicamente un programa con procedimientos (al estilo de los programas convencionales) y solamente hacían la tarea para la que habían sido diseñados (reconocimiento de oraciones particulares). En la actualidad las ATNs están en completo desuso por haber cambiado radicalmente tanto la teoría lingüística para la que estaban pensadas como el estilo de los formalismos y lenguajes de programación que se utilizan para procesar las lenguas naturales.

Precisamente ese cambio ha impuesto en la actualidad las gramáticas independientes de contexto y los formalismos y lenguajes declarativos. La existencia de varios algoritmos de *parsing* eficientes para las gramáticas independientes del contexto ha contribuido a popularizar este modelo de *parsing* Estos modelos (basados en reglas) asumen que una gramática es un conjunto uniforme de reglas que se centran fundamentalmente en las propiedades estructurales de las representaciones superficiales de las oraciones. Otro hecho que ha contribuido a extender este modelo es el grado de explicitud acerca de tales reglas que existe en la actualidad en teorías como GPSG o LFG. Es decir, estas teorías han desarrollado reglas explícitas y bien definidas que son muy fáciles de trasladar a un programa. No es de extrañar, por tanto, el dominio actual de los modelos basados en gramáticas sintagmáticas independientes del contexto: por una parte, son formalismos relativamente sencillos pero lo suficientemente potentes para las lenguas naturales; por otro lado, existen gramáticas teóricas que proporcionan un buen número de reglas explícitas. Hay un tercer componente, que veremos en el próximo capítulo, que ha influido en el éxito relativo de los sistemas de procesamiento de lenguas naturales en los 80: se trata del mecanismo de *unificación*, como medio de tratar composicionalmente la información lingüística.

1.2.2 Otros modelos computacionales sintagmáticos

Además de las gramáticas independientes del contexto se han propuesto e investigado otros tipos de gramáticas sintagmáticas, pero con un éxito considerablemente menor.

1.2.2.1 Gramáticas dependientes del contexto

La primera motivación para usar reglas de contexto, como señala Gross (1972), fue el tratamiento de los fenómenos de subcategorización y concordancia (algunos ejemplos aparecen ya en *Aspectos* de Chomsky). Pero su uso más extendido ha sido en la fonología y morfología teóricas (véase 4.4, "Localidad, alomorfía y procesos concatenativos y no concatenativos" en la página 38).

A pesar de que en un primer momento la utilización de reglas contextuales parece ofrecer ventajas formales para describir algunos fenómenos --y en consecuencia se deberían utilizar para esos casos-- las gramáticas dependientes del contexto presentan problemas particularmente difíciles, tanto en la teoría como en la práctica.

Los problemas prácticos son fundamentalmente encontrar *parsers* que sean eficientes. Se han descrito algunos *parsers* de este tipo pero todos son especialmente lentos (para referencia sobre ellos consúltese Benson(1979) y Grishman(1986)). Todos ellos se propusieron a finales de los 60 y en la actualidad el autor no tiene conocimiento que exista algún sistema de este tipo funcionando.

Hays(1966) proporciona una pista de por qué es tan difícil construir un *parser* dependiente del contexto: teniendo en cuenta que dicha gramática debe especificar los contextos en los que se puede reescribir un símbolo, "the difficulty that this sensitivity poses is merely that the contextual symbol can cover a longer substring than that covered by the symbol whose rewriting is restricted. The usefulness of length as a control variable is therefore impaired."

Desde el punto de vista teórico, estudios como el de Benson(1970), y sobre todo el influyente de Peters y Ritchie(1973) demostraron que las gramáticas dependientes del contexto no eran más capaces, en cuanto a capacidad generativa débil, que las independientes del contexto. En este último artículo afirmaban que los lingüistas habían usado reglas dependientes del contexto "purely to simplify grammars and to avoid the introduction of ad hoc grammatical categories."[6] Benson (1979) observa que la semántica funcional de los sistemas basados en reglas contextuales parece ser más general de lo que puede ser tratado actualmente con los sistemas existentes, y concluye:

> This suggests that context-sensivity, while theoretically interesting, is not suitable for use in programming language compilers or 'natural' language understanding systems. (Benson 1979; 111).

1.2.2.2 Gramáticas regulares o de estados finitos

En Church(1982) se formuló lo que se conoce como *Finite State Hypothesis*, según la cual todas las lenguas naturales son lenguas regulares, es decir, que pertenecen a la clase de lenguas que se pueden definir por autómatas de estados finitos (o métodos equivalentes como las gramáticas lineales a la derecha o las gramáticas independientes del contexto sin autoincrustamiento -'self-embedding'-)

La motivación básica de la Hipótesis de los Estados Finitos es, al igual que las teorías que apuestan por las transformaciones o por las gramáticas sintagmáticas independientes del contexto, restringir la variedad de gramáticas posibles de las lenguas

---

[6] Un ejemplo de cómo una gramática contextual puede simplificar una gramática sintagmática independiente del contexto lo podemos encontrar en Grishman(1986).

naturales. Como hemos visto en la sección 1.2.1 hay dos motivaciones distintas para tales restricciones:

1. la adecuación explicativa implica que hay explicar cómo los niños adquieren su lengua de manera tan rápida. La explicación de Chomsky es que esto es debido a que 1) el número de gramáticas posibles está fuertemente restringido por los principios de la gramática universal; y 2) porque estos principios y la habilidad para adquirir la gramática son innatos.

2. la adecuación observativa requiere la explicación del proceso de asignación de la estructura oracional, es decir, ¿cómo es posible que los hablantes puedan analizar y generar oraciones tan rápidamente?. La respuesta de los "sintagmáticos" es hay que restringirse a un tipo de gramática que permita dar cuenta del proceso comunicativo en "tiempo real." Gazdar, por ejemplo, propone que esto es posible con gramáticas independientes del contexto.

La propuesta de Ejerhed y Church(1983) es más radical que la de Gazdar:

> [the parser] can be so fast because natural languages belong to a restricted class of languages that can be processed efficiently. What we mean by restricted is regular, which makes parsing in linear time possible; what Gazdar means by restricted is context-free, which makes parsing in cubic time possible. (Ejerhed y Church 1983:412).

Los mismos autores manifiestan que la única forma de explicar la velocidad en el reconocimiento y producción de oraciones es restringiendo la complejidad del procesamiento y por ello proponen el mecanismo más rápido y eficiente conocido: los autómatas de estados finitos.

Esta propuesta, desde el campo del procesamiento de lenguas naturales, ha tenido poca aceptación por parte de los lingüistas teóricos pues, evidentemente, dejan sin explicar muchas cuestiones que interesan a estos últimos. En especial, cómo es posible tratar las oraciones subordinadas de las lenguas naturales. Para más información sobre los modelos basados en estados finitos se pueden consultar Church y Kaplan(1981), Ejerhed(1986) y Ejerhed(1988).

2.0 Gramáticas de unificación y gramáticas de rasgos

En la última década se ha extendido tanto en Lingüística Teórica como en Lingüística Computacional el uso de estructuras de rasgos para representar la información morfológica, sintáctica y semántica. Sin embargo, el uso de rasgos en la descripción lingüística teórica tiene amplia tradición (especialmente en fonología y, de forma limitada, también en sintaxis) durante los años sesenta y setenta.[7] En general, se considera que las categorías están compuestas de conjuntos de pares atributo-valor, donde

> An attribute is a *symbol*, that is, a string of letters. A value is either a symbol or another FD [functional description, i.e. estructura de rasgos]. The equal sign, "=," is used to separate an attribute from its value so that, in $\alpha = \beta$, α is the attribute and β the value. (Kay 1985:256)

Es fundamental que nunca se le pueda asignar más de un valor a cualquier atributo y es bastante habitual que no sea obligatorio asignar un valor a cada atributo y, por tanto, las estructuras de rasgos no necesitan que aparezca la lista completa de rasgos posibles si éstos no tienen un valor asignado para una estructura dada. Por lo general, tampoco hay ningún significado asociado al orden dentro de la estructura de rasgos. Desde un punto de vista matemático, esto significa que las matrices de rasgos son funciones parciales del conjunto de atributos en el conjunto de valores. Como dice Kay(1985) dada una estructura de rasgos *F* que contenga el atributo *a* se puede utilizar la expresión *F(a)* para referirse sin ambigüedad a su valor.

En los últimos años -Shieber (1986), Gazdar y Mellish(1989)- se ha extendido la idea de ver a los rasgos como grafos simples. Se les conoce como gráficos acíclicos dirigidos (*directed acyclic graph, DAG*). El grafo está dirigido porque los arcos tienen direcciones, indicadas por las flechas, y es acíclico porque no contiene ciclos dentro (no es posible ir del nudo a sí mismo siguiendo las flechas). Los arcos están etiquetados con atributos y los nudos tienen valores atómicos.[8]

---

[7] Con *Aspectos*, Chomsky fue el gran precursor de los *símbolos complejos*. Es decir, los formantes estaban compuestos por elementos más pequeños, los rasgos, que estaban especificados binariamente (+ ó -) para una propiedad particular. Se utilizaron especialmente en la gramática transformacional para dar cuenta de las restricciones seleccionales y la subcategorización.

[8] Como señalan Gazdar y Mellish(1989), no es obligatorio que los valores sean atómicos. La sofisticación de la teoría de rasgos actual permite que un valor pueda ser en sí una categoría

Según Reyle y Rohrer(1988), el concepto de gramática de unificación fue desarrollado por Martin Kay a finales de los setenta porque "[Kay] wanted to give a precise definition (and implementation) of the notion of 'feature'." Para ello, quiso combinar los rasgos de distintas categorías complejas, pero permitiendo -además de valores atómicos (típicos de las descripciones lingüísticas y formales hasta esa fecha)- el uso de valores complejos. Para que dos estructuras de rasgos se puedan combinar ("unificar") es necesario que sean *compatibles*. Se definen como *incompatibles* cuando tienen valores diferentes para el mismo atributo. La diferencia entre ambas es crucial, ya que las estructuras que contenga información conflictiva entre sí no podrán unificarse. Formalmente, se define la *unificación* de dos estructuras de rasgos D' y D'' como aquella estructura de rasgos más general D, tal que $D = D' \cup D''$. Este mecanismo es tan poderoso, según Kay, que "given a few extra conventions, it makes it possible to claim that the grammar of a language is nothing more than a single complex FD." En la parte central de la tesis tendremos ocasión de examinar esta afirmación.

A partir de estos conceptos, las estructuras de rasgos y la unificación, se han desarrollado varios formalismos y muchos sistemas de procesamiento de lenguas naturales cuya característica primordial es que tienen unas gramáticas muy compactas y reducidas en cuanto a reglas, a costa de colocar la mayoría de los fenómenos sintácticos en el diccionario.

Shieber(1986) apunta los siguientes requisitos que deben cumplir los formalismos gramaticales de unificación:

- (que estén) basados en la estructura superficial: tienen que proporcionar una caracterización directa del orden superficial real de la cadena de elementos de una oración. Es decir, la interpretación se proyecta directamente de la estructura superficial, sin mediar ningún tipo de transformación.
- informativos: deben asociar la información de los elementos superficiales con algún componente informativo.
- inductivos: deben definir la asociación entre los elementos superficiales y los elementos informativos de una forma recursiva.
- declarativos: tienen que definir la asociación entre cadenas superficiales y elementos informativos en términos de qué asociaciones están permitidas y no de cómo tienen que ser computadas estas asociaciones.
- Los elementos informativos tienen que estar basados en rasgos complejos: las asociaciones entre los rasgos y sus valores tienen que estar bien definidas y organizadas en conjuntos estructurados.

Estos elementos informativos basados en rasgos complejos reciben distintos nombres em la terminología lingüística: estructuras de rasgos (Shieber), estructuras-F (en LFG), haces de rasgos (en GPSG), estructuras funcionales (en FUG) y términos (en DCG).

sintáctica, es decir, que los rasgos pueden tener categorías como valores. Esto permite que se puedan establecer generalizaciones gramaticales de una forma bastante sencilla.

2.1 Distintos formalismos basados en la unificación

Los distintos formalismos que vamos a citar a continuación, según Shieber, pueden considerarse métodos de unificación, aunque no estén basados directamente en ella o en estructuras de rasgos complejos.

Gramática categorial: una gramática categorial utiliza la concatenación de elementos superficiales para formar constituyentes. Los elementos informativos pueden considerarse como categorías complejas formadas por un rasgo *funtor* (con una categoría como valor) y un rasgo *argumento*.

De igual forma las gramáticas de Montague pueden ser consideradas como pares de reglas de combinación de cadenas y reglas de combinación denotativas. Los elementos informativos serían la combinación de un rasgo de categoría compleja y un rasgo cuyo valor es la denotación de la expresión.

GPSG: como utiliza una gramática independiente de contexto, la combinación de elementos superficiales implica únicamente la concatenación (y de esta forma cumple el requisito de estar basado en la estructura superficial). El componente informativo es un sistema restringido de pares atributo-valor, ya sean rasgos con valores simples o complejos.

HPSG: también llamadas gramáticas nucleares (*head grammars*), estas gramáticas están basadas en GSPG pero al contrario que sus predecesoras no tienen restricciones acerca de la independencia contextual.

LFG: las estructuras informativas son un sistema recursivo de atributo-valor que contiene algunos tipos de rasgos e información.

FUG: las estructuras funcionales -las entidades informativas- son un sistema generalizado de pares atributo-valor.

DCG: los términos son las estructuras básicas de información. Se les puede considerar como un tipo de sistema atributo-valor en el cual los rasgos se corresponden a las posiciones argumentales. En particular, un término *f(a,b,c)* está compuesto por un rasgo *funtor* cuyo nombre es *f* y cuya *aridad* o *valencia* es 3, y además los 3 rasgos argumentales con sus respectivos valores *a, b, c* .

Shieber señala que la utilización de rasgos y valores no es extraña en el mundo informático, y que es fácilmente implementable:

> In fact, viewed from a computational perspective, it is not surprising that so many paradigms of linguistic description can be encoded directly with generalized feature/value structures of this sort. Similar structures have been put forward by various computer scientist as general mechanisms for knowledge representation (Ait-Kaci, 1985) and data types (Cardelli, 1984). Thus, we have hardly constrained ourselves at all even though limited to this methodology. (Shieber, 1986: 10)

En resumen, las características que presentan este tipo de enfoque son:

- La asociación de cadenas superficiales con elementos informativos en un sistema de rasgos y valores.

- La construcción inductiva de tales asociaciones mediante la combinación simultánea de subcadenas superficiales y de los elementos informativos asociados. Tal combinación se basa en reglas.

Cuando expliquemos el formalismo de nuestra gramática tendremos ocasión de ver detenidamente estas características.

2.2 El carácter lexicalista de los modelos de unificación

Como comentamos anteriormente, la mayoría de estos formalismos, y especialmente LFG, PATR-II y HPSG, tienden a utilizar un estilo de análisis orientado léxicamente. La mayor parte de la complejidad de estos sistemas reside en el lexicón, reduciendo al máximo la complejidad de las reglas gramaticales. Este tipo de enfoque fue propuesto por primera vez por LFG.

Precisamente la simplicidad de las gramáticas radica en su uso de estructuras léxicas complejas. Pero está claro que nadie desea escribir estructuras de rasgos tan complejas y redundantes para cada entrada léxica. Hay dos soluciones a este problema.

1. Desarrollar unas técnicas que permitan expresar generalizaciones léxicas de tal forma que las entradas léxicas puedan escribirse en una notación compacta. Por ejemplo, en PATR-II se utilizan plantillas y reglas léxicas, en LFG reglas de redundancia léxica, y en HPSG hay transmisión por defecto. A medida que las gramáticas dependen más y más de codificaciones léxicas complejas, estas técnicas adquieren mayor importancia.

2. Ampliar o extender el formalismo para que sea más expresivo. Por ejemplo, se pueden añadir rasgos especiales con un comportamiento complejo para complementar la noción de unificación.

¿ En qué consisten las técnicas que amplían el poder expresivo del formalismo ? Como recoge Shieber(1988), los formalismos pueden dividirse en dos clases principales: aquellos que han sido diseñados para ser *herramientas* lingüísticas, y aquellos que pretenden ser *teorías* lingüísticas. Los objetivos, especialmente en el área de la expresividad del lenguaje, son opuestos. De hecho, estas diferencias se manifiestan en el tipo de extensiones que los formalismos incluyen. Los formalismos de tipo "herramienta" (como PATR-II, FUG, DCG) tienen mecanismos para aumentar el poder expresivo, mientras que los del tipo "teoría" (por ejemplo, LFG, GPSG) tratan de incorporar mecanismos diseñados para tareas muy específicas relacionadas con el tipo de análisis lingüístico que propugnan.

Como ejemplo de mecanismos "expresivos" veremos los que utiliza la gramática funcional de M. Kay (conocida por FUG). Esta gramática incorpora ciertos rasgos especiales (concretamente *cset* y *patterns*) que tienen una interpretación específica: indican el orden de concatenación de las cadenas de subconstituyentes. Otro dispositivo es la incorporación de una notación especial para la alternancia dentro de las estructuras de rasgos. Gracias al uso de llaves ('{ }') se especifica que sólo uno de los rasgos o estructura de rasgos tiene que escogerse. Hay dos tipos de disyunción o alternancia: de rasgos o estructuras de rasgos (disyunción general) y de valores (disyunción de valores). Por último, también utiliza un valor -no un rasgo- con una interpretación especial: el valor ANY ('cualquiera') unifica con todo, al igual que hace una variable. Pero a diferencia de las variables, para que una estructura de rasgos final esté bien formada, no puede estar presente ningún valor ANY. En FUG se entiende

que la oración completa corresponde a una estructura de rasgos (o mejor dicho, estructura funcional). Una desventaja que presentan los sistemas con valores del tipo ANY es que pueden permitir que estructuras incorrectas se conviertan en estructuras bien formadas a través de diversas unificaciones.

Por otra parte, casi todos estos formalismos tienen reglas muy simples de combinación que operan sobre entradas léxicas con estructuras de información muy complejas. En este caso es especialmente importante ser capaz de organizar el diccionario de una manera que elimine las redundancias y que favorezca las generalizaciones.

Shieber señala tres métodos generales bastante utilizados para codificar generalizaciones léxicas: la transmisión o herencia simple (*simple inheritance*), trasmisión o herencia por defecto (*default inheritance*) y transformación de la información léxica.

- **Herencia simple**: las entradas léxicas a menudo comparten mucha estructura en común. Por ejemplo, los tipos de verbos (transitivo, intransitivo,etc.) suelen necesitar complejas descripciones que pueden ser simplificadas gracias al uso de plantillas léxicas que permitan definir las estructuras de rasgos que comparten determinados elementos léxicos y que suponen, por tanto, una abstracción léxica muy útil. Este método presenta la ventaja adicional de que se pueden utilizar distintas plantillas dentro de una jerarquía, de tal forma que unas incluyan a otras. Esto se conoce como "compartir estructura" (*structure-sharing*) y se ha tomado de los sistemas de representación del conocimiento en Inteligencia Artificial. Los conceptos definidos se organizan del más al menos específico. Además una entrada léxica puede "heredar" información de más de una plantilla. Esto se conoce como *herencia o transmisión múltiple.*

- **Herencia por defecto**: este método se utiliza para codificar sobre todo excepciones, es decir, cuando se quiera que una entrada léxica no reciba toda la información asociada dentro de una jerarquía, sino sólo una parte de esa información. Esto se realiza mediante la definición de precedencia de ciertos segmentos de información sobre otros. Pero a veces con este tipo de mecanismos surgen problemas entre las excepciones y la transmisión múltiple debido a que pueden interactuar entre sí. La solución a este problema depende del cuidado que se tenga en asignar precedencias particulares.

- **Transformación de la estructura léxica**: a veces se necesita un mecanismo más poderoso para representar las relaciones sistemáticas entre distintas estructuras de rasgos dentro del lexicón. Un caso muy conocido son las reglas de redundancia léxica de LFG. Estas reglas léxicas se aplican exclusivamente en formas semánticas, y expresan patrones de redundancia dentro de determinadas clases léxicas. Se usa normalmente para relacionar formas semánticas de oraciones activas y pasivas. De nuevo hay que tener muy en cuenta el orden de aplicación de las reglas, pues este tipo de transformaciones arbitrarias dependen en gran medida de la secuencia en que se apliquen.

- **Otras técnicas de organización léxica**: la GPSG utiliza otras técnicas para establecer la información léxica asociada con elementos léxicos particulares de la lengua:

  1. *Convenciones de abreviación,*

  2. *Especificación de rasgos por defecto*: es una variante de la herencia de rasgos por defecto. Consiste en asignar valores por defecto a los rasgos que no reciben un valor por otros medios

3. *Restricciones de coaparición de rasgos*: establecen condiciones sobre las configuraciones admisibles de rasgos.

Para concluir este apartado, qué mejor que las palabras de Shieber:

[...]the fundamental observation that from a broad range of research directions --from varied work within linguistics, artificial intelligence, and computer science-- researchers are converging upon a single approach in which declarative and procedural interpretations of grammars can coexist. (Shieber 1986:67)

3.0 Conceptos informáticos

3.1 *Lenguajes de procedimientos y lenguajes declarativos*

Uno de los conceptos fundamentales que desarrollaremos a lo largo de esta tesis es la conveniencia de usar lenguajes declarativos en los sistemas destinados a procesamiento de lenguas naturales. Empezaremos con una breve exposición de los distintos tipos de lenguajes de computación y a lo largo de todo el desarrollo de la tesis se irán exponiendo los argumentos. Veamos en primer lugar una definición muy sencilla de lenguajes procedurales y lenguajes declarativos:

Lenguajes procedurales: Con un lenguaje procedural se le dice al ordenador cómo hacer lo que uno quiere -paso a paso-. En estos lenguajes hay que decir explícitamente al ordenador el "procedimiento" que tiene que seguir para resolver el problema. La mayoría de los lenguajes de alto nivel utilizados comúnmente pertenecen a este tipo: BASIC, FORTRAN, COBOL, PASCAL, PL/1, C.

Lenguajes declarativos o no procedurales: Se declaran unos "hechos" (*facts*) y unas reglas de inferencia. Con estos datos se le pide al sistema que resuelva una consulta. El sistema sabe cómo obtener la respuesta. El acierto o la existencia de la respuesta en sí misma depende de los datos que proporciona el programador.

Estos lenguajes de programación como por ejemplo, Prolog (**Pro**gramación **Lóg**ica), se basan en la lógica proporcional y en el cálculo de predicados. A la hora de enfrentarse con la programación, el trabajo importante consiste en definir correctamente los datos del problema pues la máquina efectúa el trabajo mecánico de tratamiento.

Lenguajes mixtos: en ocasiones se ha considerado la existencia de lenguajes mixtos, que combinan los procedimientos con la capacidad de inferencia. El representante más conocido es LISP.

En el caso concreto del procesamiento de lenguas naturales, si tenemos formalizado nuestro conocimiento lingüístico a través de reglas, éstas pueden ser codificadas directamente en Prolog y el mismo programa, mediante su motor de inferencia, se encarga de hacer el *parsing*. Los otros lenguajes de programación, en cambio, requieren que se les indique paso a paso lo que tienen que hacer. No es

extraño, por tanto, que la mayoría de los sistemas en lengua natural de los últimos años se escriban en Prolog[9] o en LISP.

La razón de ello es, simplemente, porque es más elegante y más sencillo (Grishman 1986). Tenemos un lenguaje de programación con una estructura de control simple que permite analizar lenguas independientes del contexto sin requerir que tengamos que escribir un procedimiento de *parsing*. Pero posiblemente la mayor ventaja que nos proporciona Prolog es que es muy fácil modificar y añadir reglas a la gramática. Con otros lenguajes los cambios importantes suponen tener que reescribir de nuevo la gramática y los procedimientos. Un inconveniente actual de los lenguajes declarativos es que son mucho más lentos que los convencionales (i.e., procedurales). En los proyectos a gran escala se plantea la posibilidad de desarrollar el prototipo en Prolog (sobre todo, por su flexibilidad ante los cambios), para luego codificar el producto final en un lenguaje más rápido.

La tarea del programador se plantea, por lo tanto, de un modo distinto según el tipo de lenguaje que vaya a usar. Hay ciertas técnicas de programación que se corresponden más típicamente con los lenguajes procedurales que con los declarativos, y viceversa. Uno de los ejemplos más típicos de estas diferencias técnicas es el uso de la **iteración** o la **recursividad**. Se entiende por iteración la actuación repetida de un proceso hasta que se cumpla la condición **explícita**, que se hace operativa durante la ejecución de un programa. Esta es una técnica que se codifica fácilmente en lenguajes de procedimientos haciendo uso de los típicos bucles (*loops*) con instrucciones como *do...while*, *if...else*, etc. En cambio, un procedimiento es recursivo cuando es capaz de referirse a sí mismo. Es decir, un procedimiento recursivo puede llamarse a sí mismo cada vez que la condición no esté satisfecha y el ciclo básico de computación no se ha completado, hasta obtener un valor explícito que completa el proceso. La recursión es un proceso **implícito** que es característico de los lenguajes declarativos. La recursión es una técnica más abstracta que la iteración y su uso debe limitarse a los casos necesarios porque consume grandes cantidades de memoria (en Prolog, por ejemplo, supone hacer una copia de esa parte del programa en cada repetición). Desde el punto de vista del lingüista, en cambio, es más natural usar la recursión que la iteración pues se aproxima más a la forma en la que se concibe el lenguaje y las gramáticas. En la sección dedicada al formalismo gramatical se verá una típica definición recursiva cuando se explique el predicado `member`.

Otra de las diferencias más importantes entre lenguajes de procedimientos y lenguajes simbólicos es las distintas maneras en las que el ordenador los entiende. Por lo general, los lenguajes del tipo de PASCAL, C, COBOL, etc, se **compilan**. Esto quiere decir que existe un traductor, conocido normalmente por *compilador*, que reconoce las instrucciones del programa y las **traduce** al lenguaje de la máquina, a simple código binario. El código resultante es el que se usa para ejecutar el programa, mientras que el código fuente no tiene ninguna función en la ejecución, ni tampoco el compilador una vez que ha cumplido su misión. Si se desea hacer una modificación en el programa, hay

---

[9] No es casual que Prolog se adapte tan bien a las características de las lenguas naturales, como comentan Pereira y Shieber (1987):

> Almost from its origin, the development of logic programming has been closely tied to the search for computational formalisms for expressing syntactic analyses of natural-language sentences. One of the main purposes in developing Prolog was to create a language in which phrase-structure and semantic-interpretation rules for a natural-language question-answering system could be easily expressed. (Pereira y Shieber 1987:2)

que repetir el proceso de **compilación**. Por el contrario, los lenguajes simbólicos y en especial Prolog son lenguajes **interpretados**. A diferencia de un compilador, un *intérprete* no se limita a hacer una traducción de las instrucciones sino que tiene un papel muy activo ya que es el encargado de tomar las cláusulas del programa e ir aplicando sobre ellas las reglas que están contenidas en su **motor de inferencia**, enviando entonces instrucciones al ordenador en su propio lenguaje. Esto quiere decir que en un programa Prolog, por ejemplo, no se consigue un código que se pueda ejecutar independientemente, sino que tanto el programa fuente como el intérprete deben estar activos durante el proceso. Implica también que el programador puede interaccionar con el programa durante la "ejecución" y modificarlo incluso, añadiendo o quitando ciertas cláusulas. Si contar con un intérprete es una ventaja que evita al programador el trabajo de escribir explícitamente cosas que el lenguaje puede inferir, la presencia de este *intermediario* es, en cambio, un inconveniente cuando de eficiencia se trata. De ahí que en los últimos tiempos estén apareciendo nuevas versiones de lenguajes declarativos como Prolog, que incorporan la gran ventaja de poder ser compilados. Es decir, una vez que el intérprete ha evaluado el programa, lo traduce al código máquina y lo guarda en un fichero que es ejecutable por sí mismo. El programa de demostración que se incluye con esta tesis está implementado en uno de estos lenguajes, concretamente en Turbo-Prolog de la casa Borland. Esta es una forma de conseguir aunar las ventajas de unos lenguajes con la rapidez de los otros.

3.1.1 Prolog

Desde sus comienzos alrededor de 1970 muchos programadores han elegido este lenguaje para aplicaciones de computación simbólica como por ejemplo, bases de datos relacionales, lógica matemática, resolución de problemas abstractos, procesamiento de lenguas naturales y algunas áreas de inteligencia artificial.

Muchas personas encuentran que la tarea de escribir un programa en Prolog no se parece a la especificación de un algoritmo de la forma en que se hace en los lenguajes de programación convencionales. Un programador de Prolog debe buscar qué relaciones formales y objetos aparecen en el problema que trata y cuáles son las relaciones verdaderas que conducen a una solución. Es decir, Prolog es tanto un lenguaje descriptivo como prescriptivo: hay que describir los hechos conocidos y las relaciones en un problema, tanto como prescribir la secuencia de pasos que tiene que ejecutar el ordenador para resolverlo. En este sentido, Prolog también es procedural, con la diferencia de qué no depende totalmente de la información explícita en el programa. La forma real en que un ordenador realiza la computación se especifica parcialmente por la semántica declarativa lógica de Prolog, parcialmente por los nuevos hechos que Prolog puede inferir de los hechos dados, y sólo parcialmente por la información explícita proporcionada por el programa. En definitiva, la mayor diferencia de Prolog con los lenguajes procedurales es que está dotado de un **motor de inferencia**, es decir, un conjunto de reglas que le permiten hacer deducciones de una forma predeterminada a partir de los datos.

Prolog proporciona una estructura de datos, llamada "términos" (*term*). Un programa en Prolog consta de un conjunto de cláusulas (*clauses*), donde cada una es o bien un hecho (*fact*) acerca de una información dada o una regla acerca de cómo puede inferirse una solución partiendo de los hechos dados (reglas de inferencia).

Programar en Prolog consiste en:

1. Declarar algunos hechos acerca de objetos y sus relaciones.

2. Definir reglas sobre objetos y sus relaciones.

3. Hacer preguntas acerca de objetos y sus relaciones.

Prolog es un lenguaje conversacional o interactivo, es decir, el usuario puede mantener un cierto tipo de conversación con el ordenador. Una vez que ha introducido los hechos y las reglas necesarias para resolver el problema, puede formular preguntas apropiadas y Prolog buscará las respuestas y las mostrará. Describiremos brevemente el funcionamiento de Prolog y sus conceptos fundamentales.

Prolog es un lenguaje simbólico lo que quiere decir que no maneja un conjunto de instrucciones preestablecido, sino un conjunto de **símbolos** propuestos por el programador. Las nociones básicas son las de **átomo** y **predicado**:

- **átomos**: un átomo es simplemente una cadena de caracteres elegidos por el usuario para representar un objeto específico o una relación. Estas cadenas tienen que cumplir unas restricciones muy sencillas como la de no comenzar por un número o por un carácter de subrayado (*underscore*). Por ejemplo, "29marzo" o "_sintagma" no son átomos admitidos. Tampoco pueden contener ciertos caracteres como los guiones ("harley-davison" no es un átomo). Una cuestión muy importante es que si el primer carácter de la cadena es una letra mayúscula, se entiende que la cadena representa a una **variable** dentro del programa. Prolog también puede manejar **números enteros**, pero esta característica no se usa demasiado en procesamiento de lenguas naturales.

- **predicados**: son simplemente átomos que dan nombre a una relación entre un número determinado de argumentos (que pueden ser átomos a su vez o estructuras más complejas). Por ejemplo en:

```
palabra(raíz,terminación).
```

  `palabra` es el nombre del predicado, mientras que `raíz` y `desinencia` son los argumentos.

Cuando todos los argumentos de un predicado están representados por **constantes** decimos que ese predicado es un **hecho** (*fact*). Para construir un programa en Prolog es imprescindible declarar un cierto número de hechos que puedan ser tomados como punto de partida (*verdades ciertas*) por el sistema de inferencia y de reglas. Normalmente, todos los hechos que contiene un programa se agrupan y forman la *base de datos*. Por ejemplo, el grupo de predicados:

```
raíz(niñ).
terminación(o).
terminación(a).
terminación(os).
```

podría constituir la base de datos de un pequeño programa. En los sistemas de procesamiento de lenguas naturales esta base de datos contiene generalmente el diccionario o lexicón.

Sin embargo, no siempre todos los argumentos de un predicado tienen un valor constante, sino que muy frecuentemente estos argumentos están representados por **variables**. Se conoce como **término** a cualquiera de los tres tipos de elementos que Prolog puede manejar: *constantes*, *variables* y *estructuras*. Para averiguar los valores de esas variables tenemos que proporcionarle al programa un camino que le permita

deducirlos. La forma de hacerlo es establecer una serie de reglas que permitan inferir las soluciones a partir de los hechos. La siguiente regla establece que llamamos palabra a la asociación de una raíz con una terminación. Para ello, tienen que cumplirse dos condiciones: el primer argumento tiene que estar declarado como raíz en la base de datos y el segundo como terminación:

```
palabra(Raíz,Terminación) :-
     raíz(Raíz),
     terminación(Terminación).
```

De tal forma que si el usuario desea conocer cuántas palabras se pueden componer con la pequeña base de datos que dábamos como ejemplo, puede formular la siguiente pregunta:

```
palabra(Raíz,Terminación) ?
```

donde Raíz y Terminación son variables. Prolog tomará el predicado e intentará buscar valores a las variables usando la información que le proporciona la regla. El predicado que encabeza la regla y el que pone en marcha su funcionamiento se conoce como la *cabeza de la cláusula* (la parte izquierda de la regla) mientras que los predicados que contiene y cuyas condiciones deben cumplirse se denominan el *cuerpo de la regla* (la parte derecha de la regla). Empieza a comprobar por el primer predicado de los que componen el cuerpo de la regla:

```
raíz(Raíz).
```

y encuentra el primer hecho de la base de datos raíz(niñ) con lo cual la variable Raíz queda ligada al valor niñ. A continuación intenta satisfacer las condiciones del segundo predicado, va a la base de datos y encuentra terminación(o), liga o al valor de la variable Terminación de forma que el predicado palabra está en condiciones de retornar al usuario la primera solución:

```
palabra(niñ,o).
```

Sin embargo, todavía quedan otras posibles soluciones del problema. El usuario puede solicitarlas (normalmente escribiendo ';') y Prolog retomará el predicado y la cláusula e intentará encontrar otro conjunto de variables que satisfagan las condiciones. De esta manera nos proporcionará todo el conjunto de soluciones con los datos disponibles (en nuestro ejemplo, falta terminación(as) para producir palabra(niñ,as)):

```
palabra(niñ,a).
palabra(niñ,os).
```

Puede ocurrir que una de las hipótesis que ha formulado Prolog resulte equivocada. En ese caso Prolog vuelve hacia atrás, desandando el camino e intenta otra nueva hipótesis. A este proceso se le conoce como *backtracking* o retrotrazado.

Esta es la forma en que Prolog resuelve los problemas. Existen pequeñas diferencias notacionales entre distintas versiones de Prolog, pero todos coinciden en este método. Existen también ciertas extensiones de la notación original que se han ideado precisamente para facilitar la escritura de gramáticas, como la notación DCG (*Definite Clause Grammar*). Esta notación incluye el uso del operador '-->' como un infijo. Si el Prolog que estamos usando posee un intérprete de DCG, cuando encuentre una regla como la siguiente:

```
oración --> sintagma_nominal, sintagma_verbal.
```

Lo traduce a cláusulas ordinarias de Prolog de la siguiente manera:

```
oración(S0,S) :-
        sintagma_nominal(S0,S1),
        sintagma_verbal(S1,S).
```

donde cada átomo ha sido traducido a un predicado que toma dos argumentos: el primero para la parte de la cadena de entrada que consumimos y el segundo para lo que resta de la cadena. (Se pueden consultar más detalles sobre la relación entre DCG y Prolog ordinario en Clocksin y Mellish, 1981 y en Gazdar y Mellish, 1989). La versión de Prolog que hemos usado para codificar GRAMPAL carecía de un intérprete DCG por lo cual no hemos adoptado esta reducción notacional a la hora de escribir las reglas.

La potencia y la flexibilidad de Prolog reside en que además de manejar átomos y estructuras tan sencillas como las que hemos ido viendo hasta ahora, puede manejar también estructuras de datos más complejas. Por ejemplo, un predicado puede tomar como argumento a otro predicado con sus propios argumentos dentro de él.

```
oración(sn(det,nombre),sv(verbo,sn)).
```

Otra de las estructuras que se utilizan ampliamente para la representación simbólica son las listas. Una lista es una secuencia ordenada de elementos de longitud no determinada. La notación más común es encerrar toda la lista entre corchetes y separar entre sí los elementos mediante comas. [1,2,3,4] es una lista de números. Para manejar una lista se suele dividir ésta en dos partes, una que contiene sólo el primer elemento de la lista y que se denomina *cabeza* y otra que contiene todos los elementos restantes y que se llama *cola*. Las listas son muy útiles para la representación simbólica de conjuntos y han sido uno de los elementos claves en nuestra implementación. En la sección dedicada al formalismo se puede ver más detalladamente el uso que se ha hecho de ellas.

3.1.2 Lenguajes orientados al usuario

Hasta el momento hemos hablado de lenguajes procedurales y lenguajes declarativos basándonos en los lenguajes que podríamos llamar clásicos. Sin embargo, no todas las posibilidades se reducen a esos lenguajes más conocidos. Existe la tendencia entre algunos miembros de la comunidad informática a la creación de nuevos lenguajes de propósito más restringido que los que hemos visto hasta ahora: lenguajes enfocados al tratamiento de problemas concretos. Se pueden crear lenguajes de programación que puedan ser usados de manera muy específica por un matemático, un ingeniero o un lingüista. Tal lenguaje diferiría de los lenguajes convencionales de programación, como Fortran, Cobol o C, por ser naturales para aquellos profesionales que no tienen experiencia en informática pero que necesitan resolver problemas particulares de una manera eficiente, segura e intuitiva. Incluso el tiempo y el coste total del proceso de depuración se reduciría mejorando las especificaciones originales. Este lenguaje debería ser fácil de comprender por los usuarios para que éstos pudieran escribir e interaccionar con programas sin ayuda de programadores.

Klerer (1987) los denomina lenguajes de computación "orientados al usuario" (*user-oriented*). Más que diseñar lenguajes universales que puedan ser aplicados en todos los contextos concebibles, el enfoque orientado al usuario reclama lenguajes para aplicaciones especializadas. Estos lenguajes orientados deben permitir notaciones y sintaxis que están ampliamente extendidas en la representación convencional de la disciplina específica. La sintaxis es flexible, permitiendo formas alternativas y estructuras de control del ordenador. La utilidad de la notación se mide a través de su aceptación psicológica por parte del usuario. La ambigüedad se resuelve basándose en el contexto, el modelo cognitivo del usuario y una paráfrasis de la la interpretación que el sistema hizo del programa del usuario. Las reglas de programación son escasas, las estructuras del formato son poderosas e intentan evitar los errores.

Es una creencia ampliamente aceptada el que comunicarse con ordenadores utilizando la metodología de programación convencional es una tarea cara, poco eficiente y no muy fiable. Es bastante desalentador, además, que para tratar problemas difíciles y extensos se necesiten programadores profesionales, de tal forma que no es nada fácil para aquellas personas que no son informáticos utilizar los lenguajes de programación de una manera avanzada. Al mismo tiempo, muchos informáticos sienten que las metodologías actuales de programación se están quedando anticuadas.

El punto de vista de Klerer es que los lenguajes que se utilizan para comunicarse con un ordenador deberían permitir al usuario que es un experto en su campo de aplicación, pero que no sea necesariamente un experimentado informático, resolver un problema aplicado de una manera eficiente y económica, y que sea comprensible (es decir, que esté documentada por sí misma) para cualquiera. El objetivo de los lenguajes orientados al usuario es que el programa (la descripción de la solución del problema) se asemeje lo más posible a la especificación del problema. Si conseguimos esto, será posible mejorar ciertos aspectos del proceso de programación. Por ejemplo, el proceso de depurado se convertiría en la comprobación de las especificaciones; y las especificaciones serían en sí mismas la documentación.

El proceso de programación tradicional se considera como un conjunto de procedimientos formales. En cambio los nuevos lenguajes orientados al usuario deberían ser sustancialmente "naturales" al dominio concreto de aplicación. Klerer utiliza el término "natural" en el sentido de que la diferencia entre la estructura-notación de un programa y la estructura de la especificación del problema es mínima. Por ejemplo, en el campo de la ingeniería y de las ciencias los manuales están

escritos en un lenguaje técnico y en notación matemática no lineal. Por lo tanto, existe una gran diferencia entre la especificación del problema expresada en una lengua técnica y su solución programada utilizando un lenguaje de programación convencional.

Por otra parte, los sistemas convencionales han estado dominados por unos datos introducidos desde el teclado y expresados de una forma lineal. Por lo tanto, la estructura de entrada y salida de los datos ha estado regulada por los enfoques convencionales. En la actualidad, sin embargo, es posible utilizar otros tipos de mecanismos de entrada de datos al ordenador, así como una salida mucho más flexible.

Lo que buscamos es un lenguaje de especificación que reduzca el proceso de transformación a un programa ejecutable y simultáneamente utilice terminología lingüística y notación que minimice el grado de entrenamiento específico del usuario así como la propensión al error. La meta de los lenguajes orientados al usuario es conseguir tal grado de facilidad de lectura (*readability*) que el programa esté documentado por sí mismo y sea fácil de comprender y modificar por otros.

Este tipo de enfoque encuentra inapropiado el diseño de un lenguaje universal que pueda ser usado en todas las aplicaciones concebibles, porque sería tan complejo como incontrolable en su uso real. Por lo tanto, los lenguajes específicos tienen que estar diseñados para áreas de aplicación específicas; sólo de esta forma el diseño de lenguajes orientados al usuario podrá tener éxito.

Otra diferencia esencial entre los lenguajes convencionales y los orientados al usuario radica en la interpretación de la naturaleza del problema del lenguaje de computación. Desde un punto de vista convencional programar, en su forma más pura, es un proceso de

Hay que tener en cuenta además que el diseño de los lenguajes convencionales ha mantenido una dependencia histórica del soporte físico (*hardware*) disponible, pero en la actualidad la potencia y flexibilidad del *hardware* han aumentado vertiginosamente.

En nuestra opinión, y dado el gran desarrollo que se prevé para las técnicas que aplican el procesamiento de lenguas naturales sería muy deseable conseguir un lenguaje de programación específicamente orientado a la lingüística.

Las gramáticas independientes del contexto puede dar cuenta de casi todos los fenómenos lingüísticos utilizando una buena combinación de rasgos. Pero es evidente que si se dispusiera de una herramienta propiamente enfocada a aplicaciones lingüísticas, se podría reflejar de una manera más sencilla e intuitiva la descripción (y solución) de los problemas, con el beneficio adicional de que la comunidad lingüística en general podría enterarse sin necesidad de tener conocimiento de lenguajes de programación particulares.

Tal lenguaje de programación enfocado a la lingüística (al estilo del creado por Klerer para las ciencias y la ingeniería) debería contar con la posibilidad de escribir reglas en una notación bidimensional (por ejemplo, los corchetes [..X..[Y]], o los árboles sintácticos) y utilizar el contexto en los casos apropiados, donde su uso mejora el planteamiento permitido por las reglas independientes del contexto. No nos cabe la menor duda de que este salto cualitativo en el campo informático pondría a disposición de los lingüistas computacionales una herramienta valiosa, y todo ello permitiría un acercamiento considerable entre la Lingüística teórica y la computacional, con el innegable beneficio para ambas disciplinas. Para que el trabajo en lingüística computacional dé los frutos apetecidos es necesario que lingüistas e informáticos

trabajen conjuntamente aplicando de forma simultánea sus conocimientos a la resolución de los problemas.

Por lo tanto, podemos trazar un pequeño esquema de la evolución histórica de los lenguajes de programación aplicados al proceso de las lenguas naturales:

En el período más temprano y por motivos fundamentalmente técnicos se intentaba codificar directamente en lenguaje máquina, posteriormente empezaron a usarse lenguajes de programación de alto nivel, pero de propósito general, y enseguida se difundió también el uso de los lenguajes simbólicos y declarativos como LISP y Prolog. Es posible que el futuro se oriente hacia el uso de lenguajes específicos de la aplicación.

4.0 Algunos aspectos de Morfología Teórica

Las secciones de este capítulo tratarán de la definición de conceptos y problemas teóricos que presenta la Morfología Flexiva. La intención es ayudar a situar, mediante la comparación con los problemas y conceptos específicos de la Morfología Computacional (que se exponen en el siguiente capítulo), las aproximaciones y soluciones respectivas. Aunque el enfoque de la tesis se centra en la aplicación computacional de la morfología, no implica que rechacemos las cuestiones teóricas: al contrario, creemos que algunas de nuestras conclusiones están en concordancia con las nuevas corrientes teóricas, lo que supone un argumento recíproco de que las propuestas tienen constatación empírica y teórica.

4.1 Breve historia: presentación de escuelas y su evolución.

Los gramáticos griegos y latinos, influidos por sus lenguas extraordinariamente ricas en fenómemos flexivos, dedicaban una buena parte de sus gramáticas (la *analogía*) a clasificar las palabras en paradigmas, en detrimento de los procesos no flexivos y, más concretamente, de la formación de palabras.

Es bien conocido que el término "morfología" no existía en la gramática clásica y que fue introducido en la lingüística en el siglo XIX. Algunos apuntan que el término lo inventó Goethe para aplicarlo a la biología y que posteriormente, debido a la influencia que ejercía la biología evolucionista sobre los filólogos comparatistas, se adaptó para cubrir tanto la flexión como la derivación (Lyons 1979). Las gramáticas tradicionales generalmente distinguían entre el "estudio de las formas" y la teoría de "poner en relación" las palabras para dar lugar a las oraciones.

Esta distinción fue evitada y criticada por Saussure, considerándola ilusoria:

> Una declinación no es ni una lista de formas, ni una serie de abstracciones lógicas, sino una combinación de ambas cosas: formas y funciones son solidarias, y es difícil, por no decir imposible, separarlas. Lingüísticamente, la morfología no tiene objeto real y autónomo; no puede constituir una disciplina distinta de la sintaxis. (Saussure 1922; 1980: 187).

Seguidores de Saussure como Bally o Hjelmslev continuaron oponiéndose al papel autónomo de la morfología por considerar que las relaciones entre los elementos

gramaticales que intervenían en el interior de las palabras no diferían sustancialmente de las relaciones entre unidades mayores[10] .

La lingüística americana, en cambio, tuvo otro punto de partida. Sapir sí distinguía entre fenómenos morfológicos dentro de las palabras y fenómenos sintácticos en las oraciones, aunque generalizó ambos fenómenos bajo la noción de *proceso gramatical.* Para Sapir todas las lenguas son equivalentes en cuanto a posibilidades de expresión, independientemente de si utilizan medios sintácticos o morfológicos. Bloomfield tenía un punto de vista parecido basándose en el concepto de *construcción.* Aquellas construcciones en las que ninguno de sus constituyentes es una forma ligada son construcciones sintácticas. Aquellas otras en las que hay alguna forma ligada son construcciones morfológicas. Bloomfield considera, por tanto, que estas últimas deben agruparse en un componente separado.

Z. S. Harris, el maestro de Chomsky, expuso en *Methods in Structural Linguistics* su método basado en la sustitución para obtener la clasificación de los morfemas (del que hablaremos más detenidamente en otro apartado). Lo que nos interesa destacar aquí es que para este lingüista "the syntactic and morphological results are obtained by the same procedure, so that no distinction is drawn between them" (citado de Schultink (1988:3)).

Este enfoque se adoptó en el primer modelo generativo (*Sintactic Structures*), donde no existe un componente morfológico autónomo. Los fenómenos como la afijación se tratan en el componente sintáctico, mientras que los morfofonológicos son asignados al componente fonológico de la gramática. La aparición de *The Grammar of English Nominalizations* (1960) de R. Lees supuso el primer estudio morfológico generativo, donde se pretendía establecer una sistematización de los procesos derivativos utilizando transformaciones. En el marco presentado por Chomsky (1957), las palabras complejas no tenían cabida en el lexicón (compuesto por palabras simples). Lees propuso que tanto los compuestos como los derivados se formaran mediante reglas en el componente transformacional. Es decir, un compuesto como *maestresala* tendría la siguiente estructura profunda oracional[11]

#maestre pres ser *de la sala#*

a la que se aplicarían sucesivamente distintas transformaciones de borrado hasta llegar al compuesto con su forma superficial. Aunque tal propuesta fue cuestionada por su complejidad y por problemas de elisión de material léxico y de excesivo poder de las transformaciones requeridas, supuso el reconocimiento del valor fundamental del componente léxico. Así, en *Aspectos* se reserva un lugar dentro de la base para el lexicón (una colección de entradas léxicas compuestas de rasgos complejos sintácticos, semánticos y fonológicos). Este fue el primer paso hacia un componente morfológico autónomo.

En la segunda mitad de los 60 se dieron otros dos pasos importantes en esa misma dirección. El primero lo constituyó el trabajo de Chomsky y Halle, *The Sound Pattern of English* (1968), donde se manifestaba la necesidad de reglas de reajuste: estas

---

[10] Para una visión más detallada de la relación entre morfología y sintaxis dentro del estructuralismo europeo se puede consultar Marcos Marín (1980; 1985:160 y ss.)

[11] Ejemplo tomado de Scalise (1984, 1987:24).

reglas adaptan las estructuras generadas por el componente sintáctico en una forma apropiada para ser aducto del componente fonológico (tendremos ocasión de analizar con más detalle tales reglas). Esto supuso concretamente la "separación" de las reglas morfofonológicas del componente fonológico.

El otro avance significativo fue la demarcación de los límites entre la sintaxis y la morfología, gracias al famoso artículo de Chomsky, "Remarks on Nominalization" (1970) que inauguró la Morfología Léxica. En él se establecía que todos los procesos morfológicos regulares y productivos debían permanecer en el componente transformacional; por el contrario, todo lo irregular e idiosincrásico debía estar en el lexicón (todo ello ilustrado por las diferencias entre los "nombradores de gerundio" y los "nombradores derivados" del inglés, respectivamente). La intención de Chomsky fue reducir el excesivo poder de las transformaciones mediante la asignación de una estructura más rica al componente de base (gracias a la "teoría de la $\bar{X}$" y a las reglas de formación de palabras en el lexicón). Scalise (1984) considera que la "Hipótesis lexicalista ampliada" -como se la llamó- marcó un hito importante en la evolución hacia la hipótesis *modular* de la gramática y además "creó el espacio teórico necesario para su componente morfológico autónomo, posibilidad que quedaba explícitamente excluida en los primeros trabajos de la gramática generativo-transformacional" (Scalise 1984;1987: 35).

Se abrió en los años 70 una nueva etapa para la morfología gracias al artículo de M. Halle "Prolegómenos a una teoría de la formación de palabras" (1973). En él se propuso por primera vez un componente morfológico autónomo dentro del marco generativo, basado en que todos los fenómenos morfológicos deben recogerse en el lexicón y en que existen unas reglas específicas para la formación de palabras. Además proporcionó una explicación a una diferencia fundamental entre la sintaxis y la morfología: la noción clave en morfología es el concepto de "palabra posible pero inexistente," frente a la noción sintáctica de "oración posible / oración imposible" (o gramatical / agramatical). El modelo de Halle, aunque de naturaleza "programática" como se ha señalado repetidas veces, sirvió para que los lingüistas generativistas volvieran a investigar en morfología, disciplina olvidada desde la época estructuralista.

La tesis de Siegel, *Topics in English Morphology* (1974) y el artículo de Jackendoff "Morphological and Semantic Regularities in the lexicon" (1975) sirvieron de prolegómenos al trabajo fundamental y fundacional de la Morfología generativa: *Word Formation in Generative Grammar*, de Aronoff (1976). Las principales innovaciones de Aronoff fueron:

1. El modelo está basado en la palabra y no en el morfema.
2. Perfecciona las reglas morfológicas, desarrollando las Reglas de Formación de Palabras (RPF).
3. Formula una serie de restricciones sobre dichas reglas con el objeto de reducir su poder.
4. Explicita un conjunto de Reglas de Reajuste cuya misión es la de reajustar el educto de las RFP.

Además insistió en la idea avanzada por Halle de que la misión de la morfología es dar cuenta del hecho de que un hablante que oiga una palabra compleja por primera vez la reconoce como palabra de su lengua y tiene alguna idea acerca de su estructura y significado, es decir, es capaz de reconocer "palabras posibles pero inexistentes" en su vocabulario personal. Pero la consecuencia más importante del trabajo de Aronoff fue la de unificar el dominio de la morfología dentro del componente léxico. Para ello tuvo

que "tomar" algunos fenómenos que estaban "asignados" al componente sintáctico (como la composición y la derivación) y otros al componente fonológico (la alomorfía y la elisión). Los primeros los trató con las Reglas de Formación de Palabras; para los segundos utilizó las Reglas de Reajuste. De esta forma, el componente léxico queda integrado por el diccionario (palabras y temas) y el subcomponente morfológico (reglas derivativas, de composición y flexivas). Chomsky (1981) define así la situación de la morfología dentro del marco de *Goverment and Binding* (GB):

> The lexicon specifies the abstract morpho-phonological structure of each lexical item and its syntactic features, including its categorial features and its contextual features. (Chomsky 1981:5)

A partir de Aronoff (1976), gran cantidad de estudios se han dedicado a la relación entre las reglas sintácticas y morfológicas en el sentido de que ambas comparten muchos factores como la condición de núcleo, el filtrado de rasgos, la subcategorización, la adyacencia, la $\bar{x}$, etc. Destacamos las propuestas de Selkirk (1982) de aplicar los principios sintácticos de la $\bar{x}$ a la palabra o a la hipótesis de Anderson (1982) de que las reglas morfológicas no operan exclusivamente en el componente léxico sino en varios componentes de la gramática. Otros como Pesetsky (1985) y Sproat (1985) se cuestionan la necesidad de un componente morfológico autónomo ya que las condiciones de buena formación de varios aspectos de la estructura de la palabra se aplican en varios componentes de la gramática[12] . Hay que hacer notar que esta hipótesis es bastante similar a la planteada por Saussure a principios de este siglo. Es innegable que la relación entre morfología y sintaxis es muy estrecha y compleja.

Por último, hay que hacer referencia a la morfología dentro otras nuevas teorías. Por ejemplo, Moortgat (1988) presenta una forma de tratar fenómenos morfológicos complejos (como las paradojas de encorchetado -'bracketing paradoxes'- donde la representación prosódico-fonológica está en contradicción con la representación morfológica) con el cálculo de Lambek. El cálculo categorial permite caracterizar independientemente la prosodia de la sintaxis y proporcionar al mismo tiempo una interpretación semántica directa de las estructuras prosódicas. Esta característica (derivada de la "completitud estructural" del cálculo de Lambek) resuelve algunos problemas que se dan entre distintos niveles de representación utilizando gramáticas sintagmáticas. Veamos, por ejemplo, cómo se representaría un compuesto sintético como "surcoreano":

```
'sur'   'corea'    'no'
 N/N       N       N\A
```

Y por último, mencionaremos brevísimamente el *status* de la morfología en la gramática de dependencias. Según Tesnière, las relaciones morfológicas están impregnadas de la idea general de "jerarquía" o "dependencia" de unos elementos con respecto a otros:

---

[12] Nuestra tesis se une a esta serie de trabajos teóricos, aportando un argumento práctico: es posible y rentable computacionalmente tratar la morfología con reglas gramaticales, similares a las de la sintaxis. En nuestro caso, esto supone un replanteamiento de la distribución de los componentes lingüísticos que propone la escuela generativista: para nosotros el componente léxico sólo incluye las entradas de diccionario, con su información gramatical y contextual; en cambio, las reglas morfológicas se incorporan al componente sintáctico (la gramática, en nuestra terminología).

El concepto de clase de palabra del que parte L. Tesnière no es una mera suma de lexemas y morfemas, sino que, además de recibir respecto a las cuatro categorías mayores [...] un significado categorial, la relación entre los elementos integrantes de las palabras es de determinación jerárquica. Por ejemplo, en *niños* el lexema es determinado por el morfema y no a la inversa. (Báez 1988:16).

Lamentablemente no tenemos mucho espacio para extendernos en otras aproximaciones teóricas a la morfología. Esta breve introducción sólo pretende "enmarcar" teóricamente los conceptos que se van a utilizar.

4.2 Los formantes morfológicos

Es una práctica habitual en los manuales tradicionales definir los *morfemas* como las unidades mínimas de análisis gramatical dotadas de significación. Los criterios generales para segmentar las palabras se basan en la distribución y en su aspecto fonológico (u ortográfico). El método distribucional, establecido por los estructuralistas, se basa en la equivalencia gramatical: dos formas o más son gramaticalmente afines si comparten algún factor en común. El método que elaboró Harris consistía en establecer tablas (por ejemplo, con las formas conjugadas de los verbos) y extraer de ellas los componentes o factores distribucionales de las palabras. La mencionada tabla de formas verbales conjugadas podría tener la siguiente estructura abstracta:

Infinitivo	Gerundio	Participio
A X	A Y	A Z
B X	B Y	B Z
C X	C Y	C Z
.	.	.
.	.	.
.	.	.

Donde A,B,X,Z... son representaciones abstractas de unidades mínimas de forma y significado (es decir, morfemas): no se pueden agrupar dichas entidades lingüísticas en otras proporciones menores en las que se pueda establecer la relación forma-significado (los fonemas son exclusivamente forma).

El mismo test contrastivo que se aplicaba a los fonemas en la época estructuralista se aplicaba también a los morfemas. Se estudiaba las distintas apariciones de un mismo morfema y se establecían los distintos *morfos* (los segmentos fonológicos resultantes de la partición de la palabra). Cuando existe más de un morfo por morfema, que aparecen en distintos entornos, se les denomina *alomorfos*. Los alomorfos siempre están en distribución complementaria: cuando ocurre uno en un contexto determinado no puede aparecer otro distinto en el mismo contexto. En caso contrario, se trata de morfos que pertenecen a morfemas distintos.

El problema principal que presenta el morfema como elemento básico de la morfología es que en muchas lenguas (el castellano, por ejemplo) hay palabras que no pueden segmentarse en partes, si no es arbitrariamente, a pesar de que estas palabras

pertenecen a las mismas clases gramaticales a las que pertenecen otras que sí son segmentables. Es el caso de los plurales irregulares del español como *virus* o *lunes*, o los adjetivos gradados (*mejor, peor, mayor*). Esta es la razón por la que los estructuralistas distinguieron entre *morfema* (elemento de "forma") y *morfo* (realización "substancial" a nivel fonológico u ortográfico). Para aludir al morfema generalmente se utiliza uno de sus morfos entre llaves (por ejemplo, el morfema de plural {s}); los morfos se representan así /s/, /es/. La distinción entre morfema y morfo, entendiendo al morfema como unidad distribucional abstracta, permite describir la distribución de palabras gramaticalmente equivalentes de una forma similar. Por ejemplo, la tercera persona del singular del pretérito indefinido de un verbo regular y de otro irregular:

temió {tem} + {ió}
quiso {quer} + {ió}

A pesar de esta "abstracción" del concepto de morfema, es innegable que en las lenguas flexivas (como el latín) es imposible segmentar de una manera inequívoca las palabras en morfos. Es decir, no se puede decidir qué segmento de la palabra corresponde al morfema de 'caso' y cuál otro al de 'número'; o en un verbo, cuál es la vocal temática, el morfema de 'tiempo', y el de 'aspecto'. Muchas veces se ha recurrido al *morfema cero [Ø]*, pero no deja de ser un 'recurso' sin constatación fonológica real, es decir, no se puede afirmar que está en distribución complementaria con otros alomorfos por la sencilla razón de no se puede contrastar con los datos la distribución de [Ø].

Esta teoría ha recibido muchas críticas, sobre todo por el paralelo que establece entre la fonología y la morfología. En las teorías posteriores el morfema ha pasado a ser una unidad de la sintaxis o del componente léxico, y ya no es simplemente una clase de alomorfos en distribución complementaria. Aunque se haya criticado hasta la saciedad el enfoque 'taxonomista' de los estructuralistas, no se puede dejar de reconocer que la primera tarea de un morfólogo es la descripción de los morfemas de una lengua. Es de alabar el empeño de los estructuralista por desarrollar una metodología organizada y procedimientos formales que permitieran identificar las unidades gramaticales con el máximo rigor y objetividad. De hecho, muchas descripciones estructuralistas conservan un valor permanente y genuino, como comprobaremos en la parte descriptiva de esta tesis.

La morfología generativa partió del concepto de "morfema," por ejemplo, en el modelo de Halle (1973) el primero de los subcomponentes del lexicón era la "lista de morfemas." Éstos eran las unidades básicas y cada uno se representaba como una secuencia de segmentos fonológicos y una etiqueta sintáctica:[13]

Nombre: $[casa]_N$
Verbo: $[discutir]_V$
Sufijo: $[-dad]_{Suf}$

La monografía de Aronoff (1976) supuso un cambio radical en la morfología: la base del lexicón la constituyen las palabras, en lugar de los morfemas.[14] Aronoff

[13] Ejemplo tomado de Scalise (1984; 1987:40).

[14] Antes de Aronoff, otros autores como Chomsky (1965) -donde se rechaza el morfema como

reconoce que *existen* morfemas (que son unidades por debajo del nivel de la palabra), pero que éstos *no tienen significado autónomo* fuera de las palabras en que aparecen. Es decir, para Aronoff los morfemas han dejado de ser la "unidad mínima con significado propio" y dicho papel debe asignarse a un nivel superior: el nivel de la palabra.

El argumento fundamental para sostener esta hipótesis es que existen morfemas sin un significado constante. El ejemplo que da es el de los nombres ingleses que presentan el esquema *X + berry*. En unos cuantos casos (*cranberry, boysenberry, huckleberry*), X no aparece aisladamente o en otras palabras. En otros casos (*strawberry, blackberry, gooseberry*), aunque X existe como palabra aislada, el significado de *X + berry* no es composicional. Análogamente, analiza la estructura interna de algunos verbos de origen latino (tanto en inglés como en español; tomamos los ejemplos de la adaptación española de Scalise (1984; 1987:53-54)) formados por temas y prefijos ligados con esquemas como *X + ferir, X + mitir, X + ducir...*, donde *X* puede ser algún prefijo de éstos: *re, in, con, trans....* La conclusión es que ni el prefijo ni la base tienen un significado constante y, por tanto, no se puede analizar el significado de estos verbos en función de las dos partes constituyentes.[15]

En el modelo de Aronoff (la morfología léxica) hay cuatro tipos de formantes léxicos *palabras simples, temas, afijos* y *morfemas flexivos*. Antes de pasar a exponer las características de estos formantes, hay que aclarar la diferencia entre *palabra* y *tema*. La Hipótesis de la Palabra-Base (HPB) de Aronoff supone que "todo proceso regular de formación de palabras toma la palabra como base." Algunos lingüistas han criticado esta hipótesis argumentando que en las lenguas flexivas (como el latín o el alemán) si a una palabra flexionada le quitas el morfema flexivo el resultado es una forma trabada (o "tema"). Está claro, por tanto, que la aplicación de las RFP a palabras o a temas (o a ambos) está sujeta a las variantes tipológicas (hablaremos de esto más adelante), y que, en cualquier caso, se debe considerar que el Diccionario lo componen tanto "palabras" como "temas"[16] . La diferencia básica entre ambos es que las primeros son *formas libres* y los segundos *formas trabadas* o *ligadas*.

Los otros dos tipos de formantes comparten una característica que los distingue de los componentes del Diccionario: los *afijos* y *morfemas flexivos* "se introducen mediante reglas [ya sean derivativas o flexivas] de modo que no existe ninguna

---

base de la morfología flexiva-, Chomsky (1970) o Postal (1969) habían aportado argumentos para considerar la palabra como "signo mínimo" de la sintaxis.

[15] Se podría presentar como contraargumento que *-berry* o *-ferir, -mitir, -ducir* no son verdaderos morfemas, simplemente porque "no tienen significado propio." De hecho, las descripciones estructuralistas no los suelen incluir en sus listas de morfemas. Creemos que Aronoff forzó demasiado los argumentos en su interés por justificar la *palabra* como unidad mínima con significado propio: es cierto que los morfemas no tienen significado autónomo fuera de las palabras, pero también es cierto que en las lenguas flexivas el significado de las palabras está, en la mayoría de los casos, determinado composicionalmente por los morfemas que las forman. En resumen, la *autonomía de significado* de las palabras no implica la *ausencia de significado* en unidades discretas menores y, por otra parte, que el significado de las palabras pueda deducirse de las partes menores que las forman no implica tampoco que todas las palabras tengan que dividirse necesariamente en morfemas.

[16] Sin embargo, tanto Aronoff como Scalise consideran que es agramatical la secuencia "tema + flexión," siguiendo la *Hipótesis de la Palabra-Base*. Nosotros no estamos de acuerdo con esta postura, por las razones argumentadas para las lenguas flexivas. Por ejemplo, cualquier forma verbal del castellano se compone de una "raíz" (o tema) y "desinencias" (o morfemas flexivos). No tiene sentido, a nuestro entender, postular en castellano que los morfemas flexivos verbales se adjuntan a palabras, como ocurre en inglés.

diferencia entre la representación de [estos formantes] y las reglas que los introducen" (Scalise 1984;1987:94). Para Aronoff, concretamente, los afijos y los morfemas flexivos son *reglas* y, por lo tanto, se representan en subcomponentes diferentes. Esta afirmación tiene implicaciones notacionales (formantes y reglas se representan de la misma manera) y al mismo tiempo teóricas (se les considera "reglas" en lugar de elementos formantes autónomos, y tienen que añadirse a otras unidades específicas).

Queremos anticipar que nuestro enfoque teórico no coincide con el marco que propone Aronoff y el resto de morfólogos generativistas. Al contrario, consideramos que tanto las *palabras* y los *temas* como los *afijos* y *morfemas flexivos* son los formantes del diccionario, y que determinadas reglas gramaticales (sin entrar en si son específicamente morfológicas o sintácticas) se encargan de flexionar las palabras. En gran medida nuestra aproximación está más cerca de las posturas estructuralistas, pero sin el uso del morfema ∅ y reelaborando la noción de *concatenación* de morfemas desde una perspectiva formal basada en la *unificación* de rasgos.

En este capítulo hemos definido los conceptos básicos de la Morfología teórica en sus distintas épocas: *morfema, morfo* y *alomorfo* en el estructuralismo; *palabra, tema, afijos* y *morfemas flexivos* en la gramática generativa. En el resto de nuestra exposición se entremezclarán dichos conceptos, porque, aunque pertenezcan a marcos teóricos distintos, no se contradicen. Así por ejemplo, con el término "morfemas" nos podemos referir tanto a "temas" como a "sufijos," o podemos hablar de "alomorfía" en el "morfema flexivo" de plural. Cuando presentemos los problemas de la Morfología Computacional, definiremos nuevos conceptos sobre los formantes.

4.3 Distintos enfoques morfológicos

En el capítulo introductorio se hizo un breve repaso de la historia de la morfología en relación con los otros componentes lingüísticos. Ahora veremos la evolución que han experimentado los distintos enfoques sobre la estructura de la palabra.

Nuestra exposición tiene un punto de partida ineludible: el importante artículo de Hockett, "Two models of grammatical descriptions"(1954), en el que se distinguían tres enfoques generales sobre la estructura lingüística:

- **Palabras y Paradigma** [*Word and Paradigm*], (PP).
- **Elementos y Colocación** [*Item and Arrangement*], (EC).
- **Elementos y Proceso** [*Item and Process*], (EP).

Relacionaremos cada enfoque con las distintas fases de la evolución del tratamiento de la Morfología en las escuelas lingüísticas. Empezaremos por el modelo PP.

El enfoque *Palabras y Paradigma* se corresponde con el modelo tradicional. Como dice Matthews (1974; 1980:70) "para los gramáticos clásicos del mundo greco-romano el problema de la palabra era fundamentalmente un problema de clasificación." Las palabras se clasificaban por categorías y por paradigmas (las gramáticas de corte tradicional como el *Esbozo* siguen conservando esa estructura). La palabra es la unidad central y los términos gramaticales (el caso, el aspecto, la voz...) son los elementos mínimos de la Sintaxis. El procedimiento se basa en el

establecimiento de clases paradigmáticas sobre lexemas modélicos (por ejemplo, *amar* para la primera conjugación, etc.) a través de los contrastes que presentan entre sí (así, la segunda y tercera conjugaciones del español aunque tienen paradigmas muy similares se distinguen en algunos tiempos, lo que justifica la clasificación separada de verbos acabados en -ER: y -IR). Las irregularidades aparecen tratadas aparte, en paradigmas especiales.

Una característica del modelo *Palabras y Paradigma* que supone una ventaja sobre los otros modelos es que "los morfemas no están organizados secuencialmente, sino que son propiedades de la palabra como un todo; no sorprende que puedan de hecho localizarse en ciertas 'circunstancias' (la identificación de la persona y número en griego [por ejemplo] siempre se realiza por la terminación), ni sorprende que esto no ocurra en otros casos" (Matthews (1974; 1980:155). Es decir, dentro de este modelo la determinación de los formantes de la palabra se reduce a raíces o lexemas y a terminaciones o accidentes (también conocidos como *exponentes*). Así, en el caso problemático del latín (y de las lenguas básicamente flexivas), donde no es posible segmentar las palabras en morfemas de una manera coherente, el modelo PP lo resuelve identificando las propiedades morfosintácticas con una terminación determinada. Como dice Matthews (1974) con el modelo basado en morfemas creamos un problema de explicación aparentemente gratuito y señala además que para las lenguas flexivas (y el español se puede considerar una de ellas según él) este modelo resulta el más apropiado:

> Una posibilidad más atractiva podría ser hacer una descripción sin tener que recurrir a procesos: en lugar de derivar la forma léxica por operaciones sucesivas, podríamos decir, en cambio, que su estructura presenta un número de posiciones afijas sucesivas, cada una de las cuales está en blanco o cubierta por afijos elegidos de un modo semejante. (Matthews 1974; 1980:157).

La complejidad de las relaciones de los afijos ("accidentes" o "exponentes" de las propiedades morfosintácticas) entre sí o con los lexemas, estará en función de los contrastes dentro del paradigma. En la práctica, la mayoría de los exponentes se ajustan a un cierto modelo: en unas lenguas predomina la sufijación, en otras la prefijación; en los nombres del español, primero se une el sufijo de género (cuando existe) y luego el de número.

El uso de esta técnica permite predecir cualquier forma nueva por *analogía* con un paradigma u otro. De hecho, éste ha sido el método tradicional de aprendizaje de lenguas, y se sigue utilizando con innegable provecho. Quizás la desventaja más seria que presenta este enfoque es su falta de criterios "explicativos": en muchos casos se limita a dar una lista de las formas de un paradigma, con la división entre "temas" y "terminaciones." Cuando hablemos de la Morfología Computacional, mostraremos que PP es el enfoque subyacente en el tratamiento informático de la morfología de algunos sistemas.

Del modelo *Elementos y Colocación* ya hemos hablado cuando expusimos el método distribucional. Fue el enfoque predominante en la Lingüística durante los años 40 y 50. Su aportación fundamental fue que se podían segmentar las palabras en unos constructos abstractos (los morfemas). Un concepto básico era que dichas unidades se relacionaban entre sí meramente por *sucesión* (un morfema detrás del otro). En esencia, el análisis morfológico de una lengua se reduce a:[17]

---

[17] Tomado de Matthews (1970; 1975:102).

1. La especificación del inventario de morfemas.
2. La especificación de las secuencias en las que estos morfemas pueden aparecer.
3. La especificación del morfo o alomorfos que pueden realizar cada morfema.

Este enfoque recibió duras críticas a causa de su inadecuación para tratar problemas como los que presentan las lenguas flexivas o el conocido caso de los pasados irregulares del inglés (como *sank, drank, sang*, etc), donde no se puede segmentar la palabra siguiendo el procedimiento distribucional.

Nos queda hablar de *Elementos y Proceso*, el modelo que goza de más partidarios desde los comienzos de la gramática generativa. Los orígenes de este enfoque se encuentran en la búsqueda de la superación de los problemas intratables por *Elementos y Colocación* (en la versión estructuralista). Por lo tanto, se parte de la negación de la existencia de unidades *discretas* en morfología. Es importante señalar que el enfoque tradicional de *Palabras y Paradigma* tampoco reconocía las unidades discretas, y que la característica esencial del modelo siguiente (EC) fue precisamente la postulación de tales unidades. Por tanto, una de las cuestiones básicas sobre las que se fundamentan los distintos enfoques es el reconocimiento o no de "señales discretas o aisladas."

¿ En qué consiste el modelo EP ? Como su propio nombre indica, consta de *elementos* (los morfemas léxicos), que tendrían desde el principio una *representación básica* sobre la que actuarían un número más pequeño de morfemas (flexivos, derivativos) que tienen la capacidad de modificar los morfemas léxicos. A esto último se denomina *procesos morfológicos*, que afectan a la raíz lexemática provocando un cambio que da lugar a la forma superficial[18] (ya sea fónica o gráfica). Veamos un ejemplo: la forma verbal del castellano *vengo* es el resultado de un proceso flexivo sobre la raíz *ven* que da lugar a la forma superficial de la primera persona del singular del presente de indicativo del verbo *venir*. Análogamente, un proceso derivativo (el sufijo agentivo *-ero*) sobre el lexema *libro* generará la palabra *librero*[19] .

Este enfoque tiene un carácter esencialmente dinámico que contrasta con el carácter "segmental" del modelo EC. Una ventaja sobre este último es que *Elementos y Proceso* es más general pues permite describir tanto estructuras discretas (palabras que se segmentan claramente en morfemas) como estructuras no discretas, utilizando los mismos criterios. Por otra parte, no es extraño que EP sea el enfoque utilizado por la morfología generativa, debido a que la idea de "proceso" se adapta muy bien al concepto de *transformación* o regla que "transforma" una estructura abstracta profunda en una cadena superficial.

En el marco actual de la Morfología Generativa los fenómenos morfológicos se recogen dentro del componente léxico. Los *elementos* del diccionario son las "palabras" y los "temas"; los *procesos morfológicos* son principalmente de dos tipos: las Reglas de Formación de Palabras (que a su vez se dividen en Reglas Derivativas y Reglas de Compuestos); y las Reglas Flexivas. Los procesos de formación de palabras (las RFP) se caracterizan por ser opcionales y poder aplicarse de forma repetida. Los procesos flexivos son obligatorios y sólo se aplican una vez.

---

[18] Además, en algunos casos se pueden necesitar *procesos morfofonológicos* (las Reglas de Reajuste de la morfología generativa) que se ocupan de los últimos detalles fonológicos (elisión de vocales, etc.).

[19] Después del proceso de reajuste fonológico que elimina la vocal "-o" de *libro*.

La teoría morfológica actual se ha desarrollado especialmente en los últimos quince años, bajo el modelo de *Elementos y Proceso*, y se ha conseguido una sistematización de los fenómenos morfológicos comparable a la alcanzada en los enfoques *Palabras y Paradigma* y *Elementos y Colocación* durante los períodos clásico y estructuralista, respectivamente. Sin embargo, como expresamos al principio del capítulo, la relación entre la morfología y la sintaxis es una cuestión abierta que se reformula continuamente y que es tema fundamental (directa o indirectamente) de cualquier trabajo en Morfología.

4.4 Localidad, alomorfía y procesos concatenativos y no concatenativos

En esta sección hablaremos de los problemas fundamentales que presenta el tratamiento de los fenómenos de la estructura interna de la palabra.

Empezaremos exponiendo el marco actual de la morfología generativa. El principio básico que gobierna todos los procesos o fenómenos morfológicos es la *localidad*. El "contexto" o "entorno" aparece en todas las reglas morfológicas. Así lo expresa explícitamente McCarthy en su principio de "Prohibición de Transformación Morfológica":

> Todas las reglas morfológicas tienen la forma A → B / X, donde A, B y X son cadenas de elementos (eventualmente nulos)[20]

Dejando aparte el peculiar nombre de dicho principio (que se formuló para distinguir las reglas morfológicas de las reglas transformacionales, como argumento en defensa de la autonomía del componente morfológico), creemos que expresa un hecho indiscutible: las reglas morfológicas son sensibles al contexto. Daremos nuestra propia definición de *localidad* en morfología:

> Cualquier regla flexiva, derivativa o de composición implica que para que se lleve a cabo un determinado proceso sobre una base es necesario que se cumplan ciertas condiciones en los elementos que se encuentran en el entorno más próximo.

Para empezar, todas las reglas morfológicas son sensibles a la categoría sintáctica de la base. Las reglas del plural y género sólo se aplican a las categorías nominales; las llamadas "desinencias" sólo se unen a bases verbales; cada sufijo y prefijo tiene una base específica a la que adjuntarse.

Nos parece oportuno hacer una aclaración en este punto. Las reglas fonológicas también son reglas sensibles al contexto. En los ejemplos apuntados alguien se podría preguntar si no son en realidad reglas fonológicas en lugar de reglas morfológicas. La diferencia básica entre ambos tipos de reglas reside precisamente en el elemento o elementos que forman el contexto de la regla: si el contexto es un segmento fonológico, la regla será fonológica; por el contrario, si el contexto esta formado por elementos morfológicos, se trata de una regla morfológica. Podemos ver esta diferencia en la conversión de las consonantes dentales en [s] si les sigue el sufijo *ión*. Como en otros contextos fonológicos similares del castellano (*-iente, -iano*) no se produce el

[20] Tomado de Scalise (1984; 1987:190).

mencionado cambio (*vert-ir* → *vert-iente*; *Mozart* → *morzart-iano*), podemos concluir que se trata de una regla morfológica. Este distinción fue establecida por Aronoff refiriéndose específicamente a las Reglas Alomórficas (R Al), que provocan cambios fonológicos (como el que hemos visto de la conversión de dentales en [s]) pero sólo a *ciertos morfemas que están en el contexto inmediato de otros morfemas determinados.* En esta misma sección volveremos a hablar de las Reglas Alomórficas.

Volviendo al problema de la representación del contexto lingüístico, hasta ahora hemos presentado una notación: las "reglas sensibles al contexto" propuestas por Chomsky como base de las gramáticas de tipo 1 en su jerarquía[21] . Desde hace unos años en morfología (y en sintaxis) se utiliza una notación esencialmente equivalente a las reglas contextuales. Es lo que llamaremos *notación entre corchetes.* La notación entre corchetes tiene la ventaja de describir claramente la estructura de los constituyentes mostrando al mismo tiempo el entorno en el que aparecen. Por eso decimos que son "esencialmente equivalentes" a las reglas contextuales, ya que permiten dar cuenta de las restricciones del contexto. Esta notación se utiliza mucho en la sintaxis generativa cuando se explican los procesos relacionados con el "movimiento de α" y el "principio de subyacencia" y demás procesos donde la noción de "localidad" tiene aplicación.

En morfología este tipo de notación puede aparecer en dos variantes. La más usual presenta esta forma:

$$[[[\ ]_X\ +\text{Suf}]_Y\ +\text{Suf}]_Z$$

donde *X, Y* y *Z* son categorías sintácticas que pueden coincidir o no entre sí. Por ejemplo, *formalizar* se podía describir como **[forma] + al] + izar]**, donde X,Y y Z se instanciarían N, Adj y V respectivamente. Si en la regla modelo de arriba sustituimos el segundo *Suf* por *Flex* (morfema flexivo), entonces Z siempre será la misma categoría de Y, porque los morfemas flexivos no cambian la categoría de la palabra.

En realidad, esta notación es una abreviación de

$$[X]_\alpha \rightarrow [[X]_\alpha\ +\text{Suf}]_\beta$$

que a su vez es equivalente a

---

[21] Una forma alternativa de regla de contexto es

$$XAY \rightarrow XBY$$

donde X e Y son el contexto. Esta notación tiene la desventaja de que impone una restricción más estrecha sobre las gramáticas permitidas que la primera forma A → B / X ___ Y. Por ejemplo, una regla de la forma *A B* → *B A* no se puede reescribir con la segunda notación. En cambio, con la notación que utilizamos se puede definir cualquier lengua que se defina con la notación *XAY* → *XBY* (Grishman 1986). Debido a esto, se prefiere normalmente la notación que utilizamos.

$$Suf \rightarrow Suf \ / \ X_{[+\alpha]} \ \_\_$$

Las ventajas de la notación "entre corchetes comprimida"[22] son evidentes: primero, es más sencilla y elegante por cuanto no repite elementos en ambas partes de la regla; segundo, refleja en una sola representación lineal toda la estructura de los elementos constituyentes, frente a las otras dos que necesitan aplicarse varias veces y no proporcionan la estructura general. La notación entre corchetes comprimida tiene características similares a la notación arbórea, a la que ha sustituido en muchos casos por motivos de mejor "visualización"[23] .

Una vez visto distintos procedimientos para representar las restricciones contextuales, nos concentraremos de nuevo en la noción de localidad. Dentro del marco de la morfología generativa se han formulado algunos principios que son *condiciones* sobre la aplicación de las reglas morfológicas y que implican esta noción. Concretamente, se han propuesto dos condiciones que se excluyen entre sí y que han provocado una polémica acerca de cuál es la más adecuada. El punto de partida se encuentra en Aronoff cuando propuso la "hipótesis de la contigüidad lineal": las Reglas de Formación de Palabras son sensibles únicamente a los rasgos asociados con el morfema adyacente al morfema adjuntado. Esto implicaba que los sufijos sólo consideraban los rasgos del último morfema de la base a la hora de adjuntarse. Análogamente, los prefijos sólo "veían" los rasgos del primer morfema de la base. Se demostró que la hipótesis "lineal" no era la adecuada pues se daban casos que no cumplían esta condición.

En su lugar Siegel propuso la "Condición de Adyacencia," que se basa en el concepto de *dominio cíclico*: cuando se aplica una RFP se crea un nudo cíclico (β en el ejemplo)

$$[\ [X]_{\alpha}\ +Afijo]_{\beta}$$

La *Condición de Adyacencia* establece que dada una estructura como

$$[\ [\ [X]_{\alpha}\ +Afijo]_{\beta}\ +Afijo]_{\gamma}$$

en la que α, β y γ son nudos cíclicos dentro de la palabra, el afijo (γ) tiene acceso a la información contenida en el nudo β pero no a la información contenida en el nudo α.

La otra propuesta alternativa es la *Condición de Átomo*, expuesta por primera vez en Williams (1981). Se basa, a su vez, en el concepto de *núcleo*: es el elemento

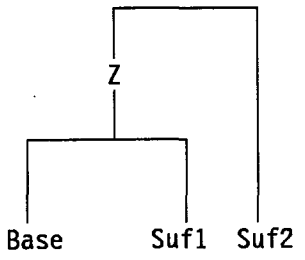
---

[22] Algunos autores la denominan simplemente "representación etiquetada," en oposición a "representación arbórea."

[23] En el marco clásico de la Gramática Generativa las relaciones entre las transformaciones y el componente de la base se expresaban mejor con estructuras arbóreas. En el marco GB, las complejas transformaciones han sido sustituidas por reglas muy simples ("Muévase-α"), a las que se añaden filtros y condiciones de buena formación que se aplican tanto a la estructura-P como a la estructura-S. Como ahora se proporciona la misma estructura en todos los niveles, se prefiere la representación entre corchetes ya que es la que refleja mejor esta nueva concepción.

capital de la construcción el cual "filtra" sus rasgos hacia el nudo cíclico superior, de tal forma que tanto el núcleo como el constituyente superior están asociados por un conjunto idéntico de rasgos (aunque el constituyente superior cuenta con algunos rasgos relevantes filtrados de los elementos no nucleares). El rasgo fundamental que filtra el núcleo a su constituyente superior es la categoría sintáctica. Por consiguiente, en la prefijación el núcleo es la base, pero en la sufijación el núcleo es el sufijo (compárese *re* + *elaborar*, donde la categoría de la palabra derivada es la misma que la de la base, frente a *breve* + *dad*, en la que la categoría sintáctica la impone siempre el sufijo). Esta noción está estrechamente emparentada con la "Teoría de la $\bar{X}$" y con la "Teoría de la Rección" en sintaxis.[24] .

La *Condición de Átomo* establece que el afijo que se adjunta sólo es capaz de "ver" los rasgos filtrados por el núcleo al nudo cíclico superior:



Suf1 y Z comparten los mismos rasgos porque son "núcleo" y "constituyente superior" respectivamente; Suf2 tiene acceso sólo a la información disponible en Z (de esa forma, tiene acceso a Suf1).

Básicamente, la "Condición de Adyacencia" y la "Condición de Átomo" hacen las mismas predicciones. La primera permite hacer referencia a condiciones en morfemas específicos; la segunda establece los rasgos a los que un afijo es sensible. A la hora de evaluar ambas propuestas se han presentado argumentos a favor de una y otra. Sin entrar en la polémica teórica, avanzaremos por nuestra parte la observación de que computacionalmente es más "manejable" la Condición de Átomo por cuanto trata con rasgos que se filtran a un nudo superior o cabeza. Esto se puede integrar fácil y directamente en un sistema que incorpore un mecanismo de unificación de rasgos, como veremos en el apartado correspondiente. Acabaremos esta parte con las palabras de Scalise (1984; 1987:201):

> Podemos concluir que tanto C Ad [Condición de Adyacencia] como C At [Condición de Átomo] definen hechos interesantes pero que ninguna de las dos condiciones logra dar cuenta de la totalidad de los fenómenos observados. En cualquier caso, lo que es importante es que las RPF están sujetas a algún principio de *localidad* y, por ello, que hay que *excluir del campo de la morfología variables esenciales o reglas de larga distancia*[25] .

---

[24] Concretamente:

> The central notion of Government Theory is the relation between the head of a construction and the categories dependent on it. (Chomsky 1981:5).

[25] La cursiva es nuestra. Hay que tener en cuenta que en el marco GB (Chomsky 1981, 1986a,

4.4.1 La alomorfía

La especificación del morfo o alomorfos que pueden realizar cada morfema era una de las tareas principales del morfólogo en la época estructuralista (bajo el enfoque *Elementos y Colocación*). Efectivamente, según este modelo se establecen inventarios completos con los morfemas y sus alomorfos, especificando la distribución de cada uno de ellos, es decir, describiendo su contexto de aparición. Tendremos ocasión de ver este enfoque aplicado a la descripción y tratamiento de la morfología del español en la parte central de esta tesis.

En contraste, la descripción de la alomorfía pasa a un lugar secundario dentro del marco de la morfología generativa, que se centra en el establecimiento de la Reglas de Formación de Palabras. Aronoff propuso que las Reglas de Alomorfía (R Al) se incluyeran dentro de las Reglas de Reajuste (RR). Recordemos que en el modelo de Aronoff (que se considera el marco estándar de la morfología dentro de la teoría generativa) las RR se encargan de ajustar fonológicamente el educto de las RFP. Es decir, realizan la misión de lo que en el estructuralismo se llamaban las "reglas morfofonológicas." Aronoff define las R Al en los siguientes términos:

> La regla de alomorfía ajusta la forma de un morfema o clase de morfema determinado en el contexto inmediato de otro morfema o clase de morfema[26] .

Dentro de este marco, las R Al tienen autonomía propia: se distinguen del componente fonológico pues se las considera reglas morfológicas (ver la aclaración en la 4.4, "Localidad, alomorfía y procesos concatenativos y no concatenativos" en la página 38); por otro lado, al pertenecer al conjunto de Reglas de Reajuste, no se consideran Reglas de Formación de Palabras. Esta reglas presentan además una ventaja: permiten asumir una forma constante para cada afijo, puesto que se puede postular una forma básica y dar cuenta de las variantes mediante las Reglas de Alomorfía.

En el modelo clásico de *Palabras y Paradigma* no se puede hablar propiamente del concepto de *alomorfía*. Las alternancias de formas se especificaban directamente como elementos dentro del paradigma. Pero aunque el concepto de morfema no existe como tal, en cambio está la idea de *lexema* al que pertenecen las diferentes variantes de la raíz. Por lo tanto, en una versión actualizada del enfoque PP los alomorfos corresponderían a esas variantes del *lexema*.

La alomorfía es uno de los problemas más serios a la hora de tratar computacionalmente los fenómenos morfológicos: primero hay que tenerla en cuenta cuando se está en la fase de segmentación de la cadena de caracteres (es decir, la palabra); posteriormente, en el proceso de análisis de los distintos formantes es necesaria para impedir formaciones incorrectas; por último, tiene un papel decisivo en

1986b, 1988) hay tres teorías relacionadas con la localidad: la "Condición de Subyacencia" (dentro de la "Teoría de los límites") especifica que la única transformación ("Muévase-α") sólo puede atravesar un nudo cíclico; la "Teoría del ligamiento," que se encarga de establecer en la estructura-s la referencia de los elementos nominales vacíos o llenos (anáforas, pronombres y expresiones referenciales); y la "Teoría del Control," que define las relaciones de correferencia entre los sujetos PRO de las cláusulas de infinitivo y otros SSNN de la oración principal. Además, hay otras nociones como "Mando-c" o "rección" que tienen que ver con la localidad, una de las cuestiones fundamentales de la teoría lingüística de la última década. Se explica así el interés que han suscitado estos problemas paralelamente en morfología.

[26] Citado de la traducción española de Scalise (1984; 1987:72).

la generación de cadenas terminales (de nuevo, palabras) ya que se debe escoger el alomorfo adecuado y "encadenarlo" correctamente a los otros elementos formantes de la palabra "gráfica." Hablaremos de todos estos problemas con mucho más detalle en el capítulo dedicado a 5.0, "Tratamiento computacional de la morfología" en la página 53.

Distinguiremos tres tipos básicos de alomorfía:

1. **Alomorfía en el tema**
2. **Alomorfía en los afijos**
3. **Doble alomorfía: en el tema y en algún afijo**

La *alomorfía en el tema* se produce cuando un lexema tiene distintas variantes. Es el caso típico de las lenguas flexivas. Por ejemplo, los paradigmas irregulares de los verbos de las lenguas romances. En las lenguas semíticas este procedimiento es explotado sistemáticamente. Por ejemplo, el llamado "plural fracto" del árabe consiste en un cambio vocálico:

```
'bab'(puerta)  /  'abuab'(puertas)
```

La *alomorfía en los afijos* es el caso más extendido de alomorfía en las lenguas del mundo. En la mayoría de las lenguas flexivas (y en otras que no lo son especialmente como el inglés) es muy común encontrar ejemplos de sufijos, prefijos y morfemas flexivos que presentan varias variantes en alternancia, dependiendo de las características morfológicas o morfofonológicas de las bases a las que se unen. Por ejemplo, la formación del plural de los nombres en alemán presenta ocho posibilidades[27] :

1	-	sin modificación	der Lehrer - die Lehrer
2	¨	con inflexión vocálica	der Mantel - die Mäntel
3	-e	con la terminación -e	der Tag - die Tage
4	¨ e	inflexión vocálica y -e	der Platz - die Plätze
5	-er	con la terminación -er	das Kind - die Kinder
6	¨ er	inflexión vocálica y -er	das Haus - die Häuser
7	-(e)n	con la terminación -(e)n	der Student - die Studenten; die Stunde - die Stunden
8	-s	con la terminación -s	der Krimi - die Krimis

Tabla 2. Plural en alemán

Como muestra el ejemplo, el morfema de plural en unos casos es una terminación que se adjunta a la raíz --(3),(5),(7) y (8)--; en otros es una modificación vocálica de la raíz pero sin adjunción de otro elemento --(2)--; también existe un caso sin marca formal de plural --(1)--. Por último, en (4) y (6) vemos que la alomorfía

[27] Tomado de Luscher y Schäpers (1981): *Gramática del alemán contemporáneo*, München: Max Hueber, pág. 109.

afecta tanto a la terminación como a la raíz ("Haus" → "Häuser"). Esto es lo que llamamos *doble alomorfía en la raíz y en un afijo*. Cuando expongamos nuestra descripción de los paradigmas irregulares de los verbos españoles veremos más ejemplos de doble alomorfía, pero esta vez teniendo en cuenta que los cambios ortográficos se consideran variantes alomórficas desde el punto de vista computacional.[28]

Trataremos a continuación de cómo se da cuenta de la alomorfía por medio de reglas. Llevamos repitiendo desde el principio de esta sección que todas las reglas morfológicas necesitan representar el entorno en que se producen los procesos. Pero para conseguir eso no es suficiente especificar el contexto (ya sea mediante "/" o con un estructura etiquetada con corchetes). Necesitamos utilizar símbolos complejos que contengan información en forma de *rasgos*. Hemos hablado extensamente sobre la utilidad de los rasgos para hacer descripciones lingüísticas, simplemente recordaremos que "la necesidad de rasgos abstractos ha sido siempre reconocida y tales rasgos se usan, en la actualidad, tanto en morfología como en fonología" (Scalise 1984; 1987:64).

(Por razones de una mejor visualización, en este caso concreto utilizaremos la notación de reglas contextuales, aunque bien se podría utilizar la notación entre corchetes). La combinación de rasgos y contexto permite expresar formalmente fenómemos como, por ejemplo, la alomorfía del sufijo *-mento* en español (datos tomados de Soledad Varela, responsable de la adaptación española de Scalise (1984;1987:65)).

Los nombres derivados deverbales con este sufijo presentan dos realizaciones alomórficas, *[-mento], [-miento]*, dependiendo de si la base derivativa verbal tiene el rasgo **[+heredado]** o **[-heredado]**. Es decir, las voces latinas que han sobrevivido en español (marcados con el rasgo [+heredado]) llevan *-mento*: *aditamento, juramento, suplemento*; las palabras de creación romance ([-heredado]), en cambio, llevan *-miento*: *acuartelamiento, ofrecimiento, llamamiento*. Todo ello se puede reflejar con la siguiente regla:[29]

$$/\text{mento}/ \rightarrow [\text{mento}] \;/\; V_{[+\text{heredado}]} \_\_$$

$$/\text{mento}/ \rightarrow [\text{miento}] \;/\; V_{[-\text{heredado}]} \_\_$$

Además de nuestra propia clasificación de la alomorfía, las alternancias alomórficas se pueden clasificar como ***recurrentes*** y ***no recurrentes*** (Matthews 1974;1980:101). La alomorfía *recurrente* es cuando los alomorfos presentan una alomorfía regular, es decir, que bajo determinados contextos siempre se dan los mismos alomorfos. Es el caso más habitual y se da especialmente en las lenguas aglutinantes. Por su parte, la alomorfía *no recurrente* se caracteriza por una alternancia que no aparece en otras series de alomorfos. Por ejemplo, la variación fonética del verbo inglés *catch, caught* no se encuentra en otros verbos irregulares ingleses.

---

[28] Por ejemplo, en *ciñ - endo*, donde la raiz regular *ceñ-* presenta un cierre vocálico y la desinencia de gerundio tiene una reducción vocálica (*iendo → endo*).

[29] En el ejemplo se indican los ajustes fonológicos pertinentes con la vocal temática del verbo, que se producirían una vez aplicada la regla.

Otra posible clasificación es alomorfía **condicionada morfológicamente** frente a alormofía **condicionada fonológicamente**. Un ejemplo de la segunda es el plural del castellano que se rije por unas reglas de contexto vocálico o consonántico. El ejemplo que presentamos de la alomorfía en la raíz provocada por el sufijo *-ión* es un caso de alomorfía morfológicamente (o gramaticalmente) condicionada: no está implicado un fenómeno fonológico general del castellano.

4.4.2 Procesos concatenativos y procesos no concatenativos

Una de las críticas que recibió el modelo estructuralista de *Elementos y Colocación* era la adjunción de morfemas se basaba únicamente en la *concatenación*. Hay muchos casos en los que una palabra no se forma simplemente por la unión de un morfema tras otro. Por ejemplo, la palabra *reestructuración*[30] no se puede describir como *re+estructura+ción* ya que el prefijo *re-* no puede adjuntarse a nombres y el sufijo *-ción*, a su vez, no puede adjuntarse a nombres. Por lo tanto, la derivación tiene que efectuarse en dos pasos:

1. *estructurar* → *reestructurar*
2. *reestructurar* → *reestructuración*

Nótese que el orden de adjunción primero del prefijo y luego del sufijo no se sigue siempre en español. Por ejemplo, en derivados de la forma:

$$in + X_V + able$$

la estructura interna es la contraria a la que presenta *reestructuración*:

$$[[re + [estructura-]_V\ ]_V + ción]_N$$

$$[in- [[olvid-]_V + able]_{Adj}\ ]_{Adj}$$

De cualquier forma, nos parece que este argumento contra la concatenación no es válido: demuestra que la concatenación de morfemas sin más ($X + Y + Z$) es una representación incorrecta, pero lo que se propone sigue siendo una concatenación lineal de dos elementos en dos pasos[31] . A nuestro juicio, este caso pertenece a lo que Lieber (1988) considera sistemas de formación *configuracionales*: "Given the asumption of hierarchical structure in addition to linear order, we might better call such systems *configurational*, rather than concatenative."

Durante la última década se ha discutido mucho sobre procesos de formación de palabras *concatenativos* y *no concatenativos* por lo que nos parece imprescindible en este momento de la exposición recurrir a ejemplos de cada tipo.

---

[30] El ejemplo procede de Corbin (1976) "Peut-on faire l'hypothèse d'une dérivation en morphologie?," citado en Scalise (1984 1987:48).

[31] Nótese, por otra parte, que el concepto de concatenación es fundamental tanto en los formalismos de unificación como en Lingüística Computacional: en ambos ámbitos es la operación básica para unir cadenas superficiales (si es que utilizamos una gramática independiente del contexto).

Por procesos concatenativos generalmente se entiende la *afijación* y por procesos no concatenativos aquellos que implican cambios fonológicos en la estructura de la palabra (variaciones consonánticas o vocálicas, procesos en la raíz y los paradigmas). Analizaremos ambos procesos por separado.

4.4.2.1 Procesos concatenativos

Adoptaremos la posición más restrictiva acerca de la afijación: los afijos no se funden con la raíz y están claramente delimitados (en contraste con los morfemas flexivos que no se pueden aislar nítidamente). Matthews (1974;1980:134) proporciona una definición más formal:

> Dos características definen a la afijación. En primer lugar, el derivando (la forma que resulta una vez aplicado el proceso u operación) consta del operando (la forma a la que se aplica) mas un nuevo formativo que ha sido añadido o 'afijado' al primero. [...] En segundo lugar, este formativo adicional (el afijo) es una constante invariable, independiente de cualquier operando concreto.

Los procesos de afijación se dividen en *prefijación, sufijación* e *infijación*, según el lugar donde se añada el afijo. Los casos más habituales son la sufijación (tendencia dominante en la familia indo-europea) y la prefijación (característica de muchas lenguas amerindias). Encontrar ejemplos de infijación no es tan fácil. Matthews (1974) nos proporciona uno del latín: la formación de los temas perfectivos a partir de los perfectivos en verbos como *rumpo* o *relinquo*. Los temas de participio se obtienen mediante la adición del infijo *-t-* y perdiendo la nasal de los temas imperfectivos (normalmente los temas de presente): *rup-t-*, *relic-t-*.

Por lo tanto, se puede hacer una distinción entre los procesos afijativos que no modifican la estructura interna de la raíz (prefijación y sufijación) y los que la rompen (infijación). En este sentido, los infijos comparten esa características con muchos procesos no concatenativos.

4.4.2.2 Procesos no concatenativos

La peculiaridad básica de estos procesos es que el elemento que se adjunta no es una constante aislable fácilmente, pero sí posee una forma básica que está determinada total o parcialmente por la misma forma del operando (el elemento que "recibe" el proceso). Es el caso típico de las variaciones consonánticas o vocálicas dentro de la palabra, también llamada *modificación interna* (Moreno Cabrera 1987). Hay tres tipos de variación (*mutation* en la terminología inglesa) vocálica:

- **apofonía**: es la alternancia vocálica. Por ejemplo, los cambios en algunas formas del plural de nombres en inglés (*man* → *men*; *foot* → *feet*) o también en los verbos irregulares (*shoot* → *shot*, *find* → *found*). Este fenómeno se conoce en alemán con el nombre de "Ablaut": es la oposición entre las formas verbales por tener vocales distintas *geben* → *gab* ("dar" / (yo) "di" y también (él,ella) "dio"). La apofonía es muy productiva en lenguas semíticas como el árabe o el hebreo donde las raíces están formadas por "esqueletos" consonánticos (ya mencionamos el caso del "plural fracto" del árabe: *kitaab* (libro) / *kutub* (libros)).

- **metafonía**: es la modificación de un elemento de la raíz, normalmente una vocal, por influencia de un sufijo. Un caso típico es el "Umlaut" del alemán. Cuando presentamos la formación de plural del alemán, vimos que varias alternancias alomórficas consistían en la inflexión vocálica de un elemento de la raíz al unirse el sufijo de plural (*-e* o *er* según los casos). También podemos dar algún ejemplo en

español: el sufijo derivativo de nombres *-ía* cuando se añade a sustantivos derivados en *-dor* suele "cerrar" la vocal (*o* → *u*). Por ejemplo, *pagador* → *pagaduría, hablador* → *habladuría, contaduría*, etc.[32]

- **armonía vocálica**: es el caso contrario de la apofonía y la metafonía, la raíz afecta a la vocal del sufijo, es decir, las vocales de los sufijos están totalmente determinadas (en "armonía") por las vocales de la palabras a las que se adjuntan. La armonía vocálica es una característica propia de las lenguas aglutinantes como el turco o el húngaro.

El cambio consonántico más conocido es la **reduplicación**, que consiste en la repetición de una consonante (y a veces tambien de una vocal epentética de ayuda). En latín existe un grupo de perfectos que presentan reduplicación de la raíz (Matthews 1974; 1980:138):

cu-curr-i ('yo corrí') raíz: *curr-*
mo-mord-i ('yo mordí') raíz: *mord-*
fe-fell-i ('yo engañé') raíz: *mord-*

Moreno Cabrera (1987) proporciona varios casos de variación consonántica en irlandés. El proceso, en esencia, es el mismo: el cambio de la consonante inicial de la palabra dependiendo de determinados fenómenos morfosintácticos. Si es un nombre depende de la función que éste realice en la oración (los nombres propios en genitivo y los sustantivos precedidos de posesivos -es decir, cuando se implica la función de "posesión"- aspiran la consonante inicial). Los verbos también aspiran la consonante inicial cuando están conjugados en pasado, imperfecto y condicional.

Otro tipo de procesos no concatenativos es la *modificación interna suprasegmental* que agrupa los procesos de cambios tonales y acentuales al emitir las palabras que sirven para oponer lexemas. La **modificación** o **variación tonal** es pertinente, por ejemplo, en chino pekinés, donde dependiendo de si la entonación es ascendente o descendente una cadena fónica puede significar una palabra u otra. Pero un caso en el que el cambio tonal implica una distinción morfológica (y no léxica, como en el chino) es la oposición entre el "cuasi-pasado" y el "perfecto" en lumasaaba, una lengua bantú del África Oriental (Matthews 1974; 1980:143). El **cambio acentual** es un proceso que en inglés opone sustantivos como *tránsport* que se derivan de verbos (*transpórt*[33]).

Acabaremos esta clasificación de procesos no concatenativos con la **modificación externa**. Según la definición de Moreno Cabrera (1987:93) "consiste en la utilización de apéndices que se unen a la palabra pudiendo afectar internamente a ésta, es decir, su composición fonética. Estos apéndices se "funden" con la palabra sin que a veces pueda distinguirse con claridad dónde acaba la raíz y dónde empieza el apéndice." Estos procesos son típicos de las lenguas flexivas y contrastan con la afijación de las lenguas aglutinantes. Hemos comentados en varias ocasiones que los morfemas flexivos del latín (o del español en menor medida) no se pueden aislar, entre otras razones porque

---

[32] El ejemplo está inspirado en la entrada *-ía* del *Diccionario de Dudas* de M. Seco, aunque corrige un error en la descripción por parte del académico: él incluye también a los nombres acabados en *-tor* cuando lo regular es que todos estos derivados no cambien la *o* por *u* (*auditoría, rectoría, gestoría, cantoría, pretoría...*).

[33] El acento es puramente ilustrativo. Por supuesto, en inglés no hay distinción gráfica entre el verbo y el nombre.

no siempre las terminaciones de caso, número, etc. son siempre las mismas (no hay modo de establecer alomorfos). En la parte descriptiva de la tesis utilizaremos el concepto paradigmático de *terminación* para agrupar toda la información flexiva de los verbos castellanos, pero reelaborado desde el enfoque de las gramáticas de unificación (donde el tratamiento composicional de la información exige que los elementos que se unifican tienen que estar claramente segmentados y su información especificada explícitamente).

Hay que hacer constar que existen algunos procesos que presentan complicaciones. Son los **morfemas discontinuos** como por ejemplo el sufijo verbal del español *-ecer* que se adjunta con mayor productividad a bases adjetivas con la forma *en + X* (*en-car-ecer, en-torp-ecer*). De alguna forma hay que explicar que el sufijo tiene acceso a la primera posición de la palabra para ver si tiene o no incorporado el prefijo *en-*. Un análisis posible de este fenómeno de discontinuidad es postular una estructura similar a la presentada en el caso de *reestructuración* (tomado de Scalise 1984;1987:67):

$$[[en + [X]_{Adj}] + ecer]$$

Esta propuesta no contradice la noción de localidad, ya que no recurre a reglas de larga distancia. Un caso un poco más complicado son las estructuras triconsonánticas de las palabras árabes. Algunos lingüistas (Matthews 1974;1980:143) consideran que las vocales que se intercalan son "infijos partidos" (*katab*, raíz:*k-t-b* + infijo partido: *a-a*). Este tipo de problemas han recibido bastante atención en la morfología computacional desde la aparación del modelo en dos niveles de Koskenniemi. Precisamente una de las críticas que ha recibido es que no puede dar cuenta satisfactoriamente de estos problemas.

Es el momento de recapitular sobre todos los procesos en conjunto. La afijación (entendida como los procesos que no afectan a la constitución fonética de palabra a la que se adjuntan) contituye un proceso bastante regular y sistemático. En cambio, los procesos de modificación interna, que son los que nos interesan especialmente en relación con la morfología verbal del español, son bastante más complejos. En la parte de descripción de la tesis veremos cómo es posible dar cuenta de tales fenómenos desde una perspectiva basada en la especificación de los contextos morfológicos mediante el uso de rasgos, pero utilizando simplemente una gramática independiente del contexto.

4.5 Tipología morfológica de las lenguas

La clasificación de las lenguas por sus características gramaticales y morfológicas sigue siendo una de las más extendidas y aceptadas, a pesar de ser la primera etapa en la evolución de la investigación tipológica (Marcos Marín 1990). El punto de partida lo encontramos en los hermanos Schlegel y en Humboldt y llega hasta el siglo XX, con Jespersen y Skalicka. La ordenación de las lenguas se basa no en sus relaciones de parentesco o derivación de otra anterior (el famoso árbol genealógico de los comparatistas del siglo pasado), sino en sus características morfológicas.

Antes de pasar a ver la clasificación hay que hacer notar que aunque se presenten ejemplos de cada tipo, esto no implica que dichas lenguas no tengan

características de otros tipos. Dicho en palabras de Lyons: "no hay tipos puros"[34] . En todos los casos, es una cuestión de grado la de pertenecer más a un tipo que a otro, en función de las características idiosincrásicas de cada lengua. Además, una lengua puede comportarse morfológicamente de una manera con una clase de palabras y de otra completamente distinta con otras categorías.

Los tipos son cuatro y se oponen, en primer lugar, por ser lenguas *analíticas* o *sintéticas*:

- *Analíticas*

  1. **Aislantes**: se entiende por tal aquellas lenguas en las que todas las palabras son invariables. Los ejemplos más comunes son el chino y el vietnamita. Los morfemas son unifuncionales, pero no los aglutinan en torno a una forma léxica radical, sino que los distribuye por la oración. El grado medio de "aislamiento" de una lengua normalmente se expresa como la proporción entre el número de morfemas y el número de palabras (Lyons 1979). Cuanto menor es la proporción entre ambas, más aislante es la lengua. El inglés se suele considerar una lengua bastante analítica por cuanto tiene un gran número de palabras de morfema único.

- *Sintéticas*

  1. **Flexivas**: Las dos características fundamentales de estas lenguas son, por un lado, la dificultad de segmentar neta y coherentemente las palabras en morfos y, por otra parte, la falta de correspondencia entre aquellos segmentos de la palabra que pudiéramos reconocer y los morfemas, es decir, que las terminaciones de caso, número, etc. no son siempre las mismas. El checo Skalicka las agrupó en un único rasgo: las terminaciones de estas lenguas son *polifuncionales* (Marcos Marín 1990). Y añadió tres rasgos más: 1) las palabras llevan siempre una terminación con información sintáctica y semántica; 2) la formación de palabras admite la movilidad de sus elementos; y 3) el orden de palabras en la oración es relativamente libre. El ejemplo más conocido es el latín.

  2. **Aglutinantes**: se caracterizan por que sus palabras se componen generalmente de una secuencia de morfos y cada uno de ellos representa un único morfema. Por ejemplo, en húngaro y en turco sólo existe un sufijo de plural, que se añade a todo nombre para obtener su correspodiente plural (Moreno Cabrera 1987). Esto implica que los morfemas son fácilmente reconocibles y segmentables y se da una correspondencia estricta entre morfema y morfo (en contraste total con las lenguas flexivas). De todas formas, el grado en que las

---

[34] Esta idea ha sido desarrollada por el checo Skalicka como la tesis central de su *kostrukt-orientierte typoligie*:

> Toda lengua es una mezcla de diferentes tipos. Las lenguas particulares se distinguen entre sí según lo grande que sea el porcentaje que presenten de los diversos tipos.

Citado de Moreno Cabrera (1987:111). Este mismo autor dice:

> Con Skalicka asistimos a un sustancial cambio metodológico de una tipología clasificatoria, dominante en el siglo XIX [...], a una tipología de extremos, típica del siglo XX. [...] En la actualidad, las tipologías lingüísticas [...] se enfocan desde el punto de vista *generalizador*, que se opone al punto de vista *individualizador*.

lenguas emplean un único morfo para realizar un morfema varía considerablemente, aunque es fácil encontrar casos en muchas lenguas.

3. **Incorporantes** o **polisintéticas**: estas lenguas, como el esquimal, se caracterizan por que todas las relaciones de la oración pueden expresarse por adjunciones hechas en una sola raíz, pero sin poder cambiarse ninguna parte de la oración. Estas adjunciones o argumentos del predicado no son necesariamente palabras. Dicho de una forma un poco ruda, las oraciones se componen de una sola palabra. Muchas lenguas indoamericanas pertenecen a este tipo

Sin desestimar a los otros dos tipos, las lenguas flexivas y las aglutinantes son las que más nos interesan en relación con los procesos concatenativos y no concatenativos que hemos visto en el apartado anterior. Podemos resumir la oposición entre ambos tipos de lenguas en el siguiente cuadro, basado en (Moreno Cabrera 1987):

	LENGUAS FLEXIVAS	LENGUAS AGLUTINANTES
oposición morfológica	modificación interna	afijación
oposición morfofonológica	metafonía y apofonía	armonía vocálica

La pertinencia de incluir la clasificación tipológica tiene que ver con la idea expresada por varios modelos computacionales de que su propuesta es de carácter universal.

Cuando tratemos en la próxima sección de las distintas aproximaciones al tratamiento computacional de la morfología, se hablará de la polémica acerca de la mayor o menor adecuación de los enfoques a cada tipo de lenguas. Es decir, de la misma manera que el modelo paradigmático está basado en las lenguas flexivas y es sin duda el más apropiado para ellas, la morfología en dos niveles de Koskenniemi está influida desde sus orígenes por el tratamiento de los fenómenos flexivos en las lenguas aglutinantes.

4.6 Características propias de la flexión

Aunque la historia de la Lingüística nos muestra que la relación entre morfología y sintaxis ha fluctuado desde la completa inclusión de la primera dentro de la segunda hasta la autonomía de ambas, es evidente que los elementos flexivos tienen un comportamiento fundamentalmente sintáctico al ser elementos de la concordancia. De acuerdo con este hecho, nosotros entendemos que la manera más adecuada de tratar teóricamente la flexión es dentro del componente sintáctico. Desde el punto de vista computacional también propondremos que es más eficiente incluirla dentro del mismo nivel que los fenómenos sintácticos y usando incluso el mismo tipo de reglas.

4.6.1 Tipos de flexión

Durante siglos se ha pensado que la flexión --en el sentido tradicional de terminaciones agrupadas en paradigmas-- era un "universal" lingüístico. Con la llegada de la lingüística moderna (que supuso un interés por todo tipo de lenguas y la pérdida del carácter modélico de las lenguas clásicas) no se puede afirmar ya que la morfología flexiva sea una parte necesaria, ni siquiera real, de la gramática de todas las lenguas. Por ejemplo, en las lenguas aislantes como el chino o el vietnamita, donde la mayoría de las palabras constan de un único morfema y una única sílaba, no se puede hablar de paradigmas. Tampoco se puede mantener la clasificación paradigmática en las lenguas aglutinantes, pues en la mayoría de los casos sus morfemas flexivos son únicos. Se impone, por tanto, utilizar otro concepto de flexión distinto del enfoque tradicional. Nos puede servir el de Matthews (1974):

> Por flexión de una palabra, categoría o lo que sea, nos referiremos al proceso completo o a cualquier parte del mismo por medio del cual se derive una forma léxica.

Si queremos especificar más esta definición podemos recurrir a la teoría actual de la morfología generativa, donde las reglas flexivas se caracterizan por adjuntarse detrás de la base (ya sea palabra o tema) y por no cambiar la categoría de la base. Además, son reglas obligarias y se aplican después las reglas de formación de palabras (en los casos pertinentes). En cualquier caso, es importante destacar que la "palabra flexionada" es ya un elemento terminal en la estructura superficial, pues ha recibido todos los procesos del componente léxico[35] . En otras palabras, son *formas libres* que pueden aparecer en la representación superficial sin necesidad de recibir otra regla morfológica. Nótese que esta definición es similar a la de "palabra" de Aronoff, como vimos en la sección 4.2, "Los formantes morfológicos" en la página 32, que se opone a "tema" o *forma trabada*. Esta distinción nos parece muy significativa: por una parte, es evidente que sólo las formas libres pueden aparecer en la estructura superficial.

Según la forma en que se adjunten los morfemas flexivos tendremos distintos tipos de flexión (Scalise 1984):

1. **Flexión por "adición"**: es el caso típico de las lenguas aglutinantes. Los morfemas flexivos se unen a la base, que es una forma libre. En inglés, la forma verbal *write* es una palabra que puede aparecer libremente en los contextos pertinentes (es decir, teniendo en cuenta la concordancia con el sujeto y el tiempo de la oración con respecto al contexto temporal del discurso). Si queremos formar la tercera persona del singular del presente basta con añadir *-s* a la palabra.

2. **Flexión por "sustitución"**: propia de las lenguas flexivas. Los morfemas se adjuntan a formas trabadas ("temas"), que no pueden aparecer como tales en ninguna representación superficial. En latín, en alemán, en español --por ejemplo-- las formas verbales se conjugan mediante la "sustitución" de una terminación por otra.

---

[35] Para ser más exactos, dentro del marco teórico de Aronoff, después de las reglas flexivas tienen que aplicarse las reglas de reajuste (entre ellas, las reglas de alomorfía) y, posteriormente, una convención para la inserción de lindes. Se llega así al componente fonológico, que es donde se materializa fonéticamente la oración.

Con esta distinción superamos el problema de la concepción tradicional de la flexión (basada exclusivamente en la flexión por sustitución). Por otra parte, no podemos olvidar --como ya se comentó en la sección de tipología morfológica de las lenguas-- que en muchas lenguas se pueden dar al mismo tiempo ambos tipos de flexión, aplicados a distintas clases de palabras. En español tenemos varios ejemplos: la flexión del número en los nombres y adjetivos se puede considerar del tipo por "adición" (los morfemas de plural se añaden sin más a cualquier forma libre en singular). La conjugación verbal, en cambio, es un caso de flexión por "sustitución."

El español proporciona un ejemplo atípico de "sustitución" en las formas nominales que están marcadas morfológicamente para el género: los morfemas de masculino y femenino (*-o, -a*) se adjuntan a formas ligadas (*niñ-, bonit-*). Este caso presenta una mezcla de las características de la "sustitución" (los morfemas flexivos se adjuntan a raíces o temas) y de la "adición" (los morfemas son aislables de la raíz de manera inequívoca).

En el tratamiento computacional de la flexión en español tendremos muy en cuenta las diferencias entre las **formas libres** y las **formas trabadas** o **ligadas.**

5.0 Tratamiento computacional de la morfología

La tarea básica que tiene que realizar cualquier *procesador* morfológico es la de analizar y/o generar las palabras (en forma gráfica, no fonológica) de una lengua. Normalmente, el procesamiento morfológico se divide en dos partes:

1. Segmentación de la palabra en morfemas, mediante la identificación de raíces y terminaciones en el diccionario. A esta tarea se la conoce como **lematización.**

2. **Recuperación de la información léxica,** que se consigue a partir de los morfemas segmentados.

Para que un procesador morfológico se considere completo debe permitir la relación con los componentes sintáctico y semántico, aunque este requisito depende en gran medida de la aplicación y del tratamiento de los fenómenos pertinentes en cada componente. Por ejemplo, el reconocimiento de frases idiomáticas o de compuestos *multiwords* tiene implicaciones sintácticas y semánticas, de igual forma que el tratamiento de la derivación está condicionado por el enfoque que reciba en el componente interpretativo.

Por otra parte, los investigadores en lo que podríamos denominar Morfología Computacional han creado nuevos términos y conceptos, sobre todo en el dominio de la representación léxica y la búsqueda en el diccionario. Una característica de la Morfología Computacional es que se ha dedicado fundamentalmente al análisis, dejando de lado en muchas ocasiones los problemas de síntesis o generación de palabras (en línea, por otra parte, con la evolución de los sistemas generales de procesamiento de lenguas naturales). Igualmente, se ha tratado más extensamente la flexión que la derivación y composición.

Hay que señalar, por último, que la morfología tiene un papel importante en muchas aplicaciones de la Lingüística Computacional, por ejemplo, en lexicología y lexicografía, en el análisis estilístico y estadístico de textos, en sistemas de recuperación de información y de consulta de bases de datos, y en traducción automática. Los procesadores morfológicos se pueden dividir en dos clases según el tipo de aplicación en que se integren:

- Módulos morfológicos integrados dentro de sistemas con otros componentes (sintácticos, semánticos) y que realizan un procesamiento completo de algún fragmento de lenguas naturales.

- Analizadores/generadores morfológicos concebidos para dar cuenta de problemas específicos (normalmente en aplicaciones lexicográficas) y que, por tanto, no

constituyen un componente dentro de un sistema de procesamiento de lenguas naturales[36]

5.1 Problemática de la Morfología computacional

Antes de ver detenidamente las características de los "procesadores" morfológicos es necesario hablar de los problemas específicos que presenta el tratamiento de la morfología por ordenador.

Lo primero que hay que tener en cuenta es que el procesamiento morfológico presenta los mismos problemas básicos que el tratamiento computacional de las lenguas naturales, a saber:

1. carencia de una descripción lingüística exhaustiva y consistente de los fenómenos --en este caso-- morfológicos[37]

2. la infinidad de fluctuaciones que en el uso de su lengua hacen los hablantes, y en concreto en el uso de las palabra (por ejemplo, en la flexión de los extranjerismos o en la conjugación verbal: *andé, anduve*). Esta cuestión es de suma importancia dependiendo de la aplicación concreta: si se trata de un sistema de recuperación de información textual o de traducción automática de texto escrito es de suponer que las fluctuaciones se ajustarán a la norma escrita; pero si se trata de una interfaz entre un usuario y la máquina, puede ser pertinente recoger algunas "desviaciones" de la norma culta, pero que sean habituales en la lengua cotidiana. Creemos que la descripción será mejor cuanto más amplio sea el fragmento de la lengua que se describa. En este sentido, nosotros buscaremos siempre un formalismo lo bastante expresivo que permita tratar el uso "fuera de la norma."

3. las limitaciones informáticas, tanto en el "soporte físico" (es decir, las características de la máquina: su capacidad de almacenamiento y su rapidez de procesamiento, fundamentalmente) como en el "soporte lógico" (especialmente, los lenguajes de programación y los compiladores de gramáticas formales). Es evidente, como lo demuestra la historia de la informática, que los ordenadores no son un medio "natural" para reproducir el lenguaje y que, a medida que las computadoras y la ciencia de la Computación han ido evolucionando, se han conseguido mejores resultados. En concreto, el procesamiento léxico requiere una gran capacidad de almacenamiento (si queremos que el diccionario tenga una cobertura suficiente).

---

[36] Schraeder y Willée (1989) llaman a estas dos clases *Stand-by-Verfahren* y *Stand-alone-Verfahren*, respectivamente.

[37] Es bien conocida esta queja de los lingüistas computacionales, que en buena medida es su obligación remediar. De todas formas, debemos tener en cuenta que en todas las ciencias, ya sean teóricas o aplicadas, la falta de acuerdo acerca de problemas concretos es una realidad, y que, en última instancia, la meta científica es explicar y formalizar los fenómenos que no lo han sido todavía.

5.1.1 La palabra como cadena de caracteres

Se ha hablado de que la morfología computacional ha desarrollado su propio dominio de conceptos, motivado parcialmente por cuestiones lingüísticas y en parte por motivos informáticos. Lo primero que hay que tener en cuenta es que el concepto de "palabra" está muy bien delimitado: *palabra* es una *cadena de caracteres* entre dos *espacios en blanco*. (No tendremos en cuenta aquí que esos caracteres son para el ordenador en realidad una secuencia de "0" y "1"; en terminología informática se dice que esto es "transparente" al usuario, es decir, que el usuario no percibe el proceso de conversión de caracteres a código binario y que, por tanto, a todos los efectos el carácter que aparece en el monitor o en la impresora es un carácter también para el ordenador).

De esta primera definición podemos extraer varias consecuencias. La primera es que la "palabra fonológica" no cuenta en el procesamiento del lenguaje natural, salvo en los sistemas de reconocimiento del habla, pero el educto de estos sistemas siempre es una palabra "gráfica," luego partiremos del supuesto de que el aducto de cualquier sistema PLN es una secuencia de caracteres y blancos[38] . Como resultado, muchos de los criterios de la Morfología teórica no se pueden aplicar, pues tienen su fundamento en la "palabra fonológica." Por ejemplo, a la hora de explicar las irregularidades del paradigma verbal español no se considera como tal el cambio *le-er* → *le-yendo, le-yó...*, ya que "algunas aparentes anomalías obedecen a principios generales fonológicos del sistema español y no constituyen, por tanto, irregularidad" (R.A.E 1973;1989:274)[39] . Análogamente, ningún lingüista diría que los siguientes pares presentan irregularidades y que son, por tanto, alomorfos:

sac-o	saqu-e	/sako/	/sake/
pag-o	pagu-e	/pago/	/page/
mez-o	mec-e	/meθo/	/meθe/

Desde el punto de vista computacional, siguiendo el criterio "gráfico," estas son irregularidades en el paradigma que hay que tener en cuenta[40] . Pero este criterio

---

[38] De nuevo, por motivos de simplificación, no tendremos en cuenta que un texto digitalizado puede contener mucha más información como marcas de página, capítulo, negrita, itálica, etc.. La mayoría de los sistemas de traducción automática y de recuperación de información textual tienen un primer nivel de procesamiento que consiste en descodificar esta información textual y entregar al sistema únicamente caracteres y blancos.

[39] El *Esbozo* es muy explícito en este sentido: el criterio es siempre fonológico, nunca ortográfico:

> [...] en la clasificación de las irregularidades no entran para nada las variaciones simplemente ortográficas [...]. *Nazco* no es una variante de *nace* por la presencia de *z* (tanto la *z* de *nazco* como la *c* de *nace* en este caso representan el fonema /θ/), sino por la *c*, representación aquí del fonema /k/. (R.A.E 1973;1989:274).

Bello, antes de la aparición de la Fonología como disciplina de la Lingüística contemporánea, ya afirmaba que "para calificar a un verbo de regular o irregular no debe atenderse a las letras con que se escribe sino a los sonidos con que se pronuncia. Como conjugamos con el oído, no con la vista, no hay ninguna irregularidad en las variaciones de letras que son necesarias para que no se alteren los sonidos" (Bello 1984:172).

[40] Casajuana y Rodríguez (1986) exponen en su "Clasificación de los verbos castellanos para un diccionario por ordenador":

> [...] en esta clasificación ha sido necesario tener en cuenta algunas consideraciones,

gráfico no sólo afecta a las consonantes, sino también a las "vocales acentuadas" y a las "vocales sin acentuar." Para el ordenador *no es lo mismo* el carácter **o** que el carácter **ó**. Cada uno se corresponde con un código binario distinto: por ejemplo, en código ASCII la *o* es el carácter 111 y la *ó* es el carácter 162. Esto tiene la siguiente consecuencia. Una palabra como *camión* desde este punto de vista tiene dos alomorfos gráficos, uno para el singular y otro para el plural:

		Código ASCII					
camión	[camión]	99	97	109	105	**111**	110
camion	[camion -es]	99	97	109	105	**162**	110

Como explican acertadamente Barton, Berwick y Ristad (1987), los cambios en la forma gráfica de las palabras son en realidad la amalgama superficial de dos hechos distintos:

- los cambios fonológicos
- las convenciones ortográficas

Y aunque la fonología y la ortografía no tienen el mismo estatus en la Lingüística, lo cierto es que las diferencias no son relevantes para el tratamiento computacional: "in any case, it is surface spelling that is input to program that analyzes text" (Barton, Berwick y Ristad 1987:117).

Podemos concluir que el tratamiento de la alomorfía por ordenador se rige por el *Criterio Gráfico*:

Criterio Gráfico

Se considerará variante alomórfica a toda cadena que presente alguna variación en sus caracteres.

Hay que hacer notar que si bien este criterio aumenta considerablemente el tamaño del diccionario, por otra parte es explícito e inequívoco y reduce algunas irregularidades que se fundamentan en el criterio fonológico. Analizaremos detenidamente esto en la sección de Flexión Verbal. Esta complejidad añadida al procesamiento morfológico por ordenador ha sido señalada en numerosas ocasiones y casi todos los modelos han desarrollado mecanismos para manejar apropiadamente esta "explosión" de combinaciones entre las distintas variantes de morfemas contiguos.

puramente ortográficas, que normalmente no se contemplan en las publicaciones sobre el tema, y que son de gran importancia a la hora de generar mecánicamente todas las formas flexivas.

5.1.2 Lematización

Dijimos al principio de esta sección que la tarea fundamental de todo procesador morfológico es analizar (o generar) las palabras, es decir, reconocer la estructura interna de las cadenas que son presentadas como aducto (o generar la forma superficial correcta, es decir, el educto final del sistema). Para llevar a cabo esta tarea es imprescindible realizar una descripción exhaustiva de los morfemas flexivos (que constituyen una clase cerrada, pues no puede aumentar ni disminuir por tener una función gramatical) y de las características que deben tener los morfemas léxicos (clase abierta a nuevas incorporaciones) para que sus distintas variantes se correspondan con las variantes apropiadas de los morfemas flexivos. Es decir, hay que proporcionar una descripción de los formantes morfológicos y establecer unas reglas o esquemas de combinación de dichos formantes (en algunos sistemas, como el modelo en dos niveles, también se necesita un conjunto de reglas de cambio ortográfico). Pero una vez hecha la tarea "lingüística" es necesario poner a funcionar todo ese conocimiento de una manera eficiente.

Uno de los problemas más importantes para conseguir esta eficiencia computacional es la *segmentación* de la cadena en posibles morfemas, mediante la búsqueda en el diccionario. A todo el proceso lo denominamos *lematización*. El peso computacional de este proceso descansa en la "búsqueda en el diccionario," y en este aspecto es en donde se diferencian bastante los distintos modelos. Presentaremos primero la problemática general y luego veremos las soluciones propuestas.

Uno de los problemas clásicos de la Lingüística Computacional es el reconocimiento de las palabras, es decir, de cadenas arbitrarias de caracteres, y sus propiedades gramaticales y semánticas. Es una de las tareas primeras a las que cualquier sistema tiene que enfrentarse: se han dado infinidad de propuestas y todavía no es posible llegar a un acuerdo de cuál es la mejor. Lo que parece estar claro es que hay un momento central en todo proceso de reconocimiento de palabras: es el momento en que las cadenas de caracteres se contrastan con las que hay en una lista, que además contiene las propiedades gramaticales y semánticas asociadas con cada *item* léxico. A esta lista la denominaremos **diccionario**. Ya hemos señalado que para que este diccionario sea manejable en tamaño y podamos aprovechar las ventajas que nos proporcionan las regularidades en el lexicón, dicho diccionario tiene que componerse de elementos más elementales que las palabras o formas flexionadas, es decir, necesitamos morfemas.

Presentaremos a continuación dos estrategias generales de búsqueda en el diccionario, una basada en un procedimiento recursivo y otra en un procedimiento iterativo. La primera estrategia se usa principalmente con lenguajes de programación simbólicos o declarativos (como LISP y Prolog), mientras que la segunda se suele codificar en algún lenguaje de tipo procedural (lenguaje C, Pascal, etc.).

Una estrategia simple de búsqueda en el diccionario es la siguiente (entendiéndose que se pueden buscar tanto palabras enteras como morfemas). Tenemos un procedimiento recursivo que empieza a examinar la primera entrada del diccionario y la contrasta con el primer carácter de la palabra que estamos buscando. Si tiene éxito, continúa con el resto de los caracteres, por turno, hasta que reconozca toda la cadena. Lo normal es que falle y tenga que volver a iniciar el proceso con la siguiente entrada, y así hasta que llegue al final del diccionario (teniendo en cuenta que es posible que la cadena no se encuentre en el diccionario; en ese caso debe contestar que la palabra no se encontró). El proceso se muestra en la figura 1; cada vez que se cumple un ciclo, es decir, se vuelve a empezar por el paso (1), se toma la siguiente

entrada de diccionario, ya que puede haber varias soluciones (por eso, la primera instrucción es comprobar que se han mirado ya todas las entradas)

```
CAD = cadena de caracteres          DIC = entrada de diccionario
CABCAD = cabeza de CAD              CABDIC = cabeza de DIC
COLCAD = cola de CAD                COLDIC = cola de DIC


1) Compruébese si hay alguna DIC en la lista; si no quedan,
            respóndase: "La cadena no existe en el diccionario"

2) Tómese CAD y la primera DIC en la lista

    3) compruébese la igualdad CABCAD = CABDIC; si falla, váyase a (1)

        4) tómese COLCAD y compruébese CABCAD = CABDIC;
                                        si falla, váyase a (1);
                                        si tiene éxito, repítase (4).

            5) si hay éxito, entonces CAD = DIC

6) Vuélvase al paso (1)
```

Figura 1. Algoritmo de búsqueda recursiva en el diccionario

Cabeza y *cola* son dos términos de programación que se utilizan comúnmente en lenguajes simbólicos cuando se procesan *listas de elementos*. Se suele definir las "listas" como secuencias de símbolos en las cuales el orden es relevante. Una lista es una estructura de datos binaria, cuyo primer argumento (la "cabeza") contiene el primer elemento de la misma y cuyo segundo argumento (la "cola") define, recursivamente, el resto de la lista. Es decir, la *cola* de una lista siempre es otra lista; en cambio, la *cabeza* de la lista es siempre un elemento. Imaginémonos que la palabra *casa* es un objeto compuesto (o estructura de datos) que consta de cuatro elementos o caracteres: **c, a, s, a**. Según la definición que hemos dado[41]

```
la cabeza de [c,a,s,a]   es   c
la cola de [c,a,s,a]     es   [a,s,a]
```

En nuestro algoritmo de búsqueda, CABCAD correspondería a *c* y COLCAD a *[a,s,a]*. Como la definición de lista es recursiva, *[a,s,a]* a su vez se compone de cabeza (en este caso CABCAD = *a*) y cola (es decir, COLCAD = *[s,a]*), y así hasta llegar al último carácter (cuando una lista tiene sólo un elemento, entonces dicho elemento es la cabeza y la cola es la *lista vacía* representada como *[]*). Debido a la recursividad de esta estructura de datos, necesitamos un algoritmo recursivo que procese la

[41] Estamos utilizando una representación de lista que consiste en incluir todos los elementos dentro de corchetes y separados cada uno por comas.

comprobación de equivalencia entre CAD y DIC. Por ello, tenemos en primer lugar la comprobación de lista vacía (es decir, no hay más entradas de diccionario que confrontar con la cadena), que hace de condición de terminación: cuando se llega a ese punto, se para la búsqueda. En segundo lugar, tenemos las instrucciones propias de la búsqueda: se contrasta siempre la cabeza de la CAD con la cabeza de la entrada de diccionario (que es otra lista de caracteres); una vez que se ha efectuado la comprobación se toma el primer elemento de la cola (que se ha convertido por tanto en CABCAD) y se hace el mismo proceso hasta llegar al último elemento de la lista. Es habitual representar gráficamente la estructura de una lista con un gráfico arbóreo, como el de la figura 2.

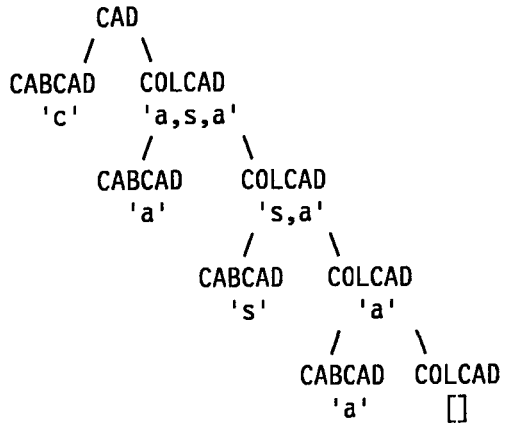


Figura 2. Árbol que representa la búsqueda recursiva

Un procedimiento alternativo de búsqueda es utilizar un "reconocedor" de estados finitos, es decir, un tipo particular de autómata de estados finitos que decida si la cadena de caracteres pertenece o no al diccionario. Para realizarlo necesitamos diseñar una red de transición de estados finitos (el tipo más sencillo de red de transición). Dicha red se compone de los siguientes elementos:

- un conjunto de nodos iniciales,
- un conjunto de nodos finales,
- un conjunto de nodos intermedios, y
- un conjunto de arcos que hay que atravesar para ir de un nodo a otro. Estos arcos se definen por una etiqueta.

La figura 3 representa esquemáticamente una red que reconoce las palabras *sol, casa* y *cama.*

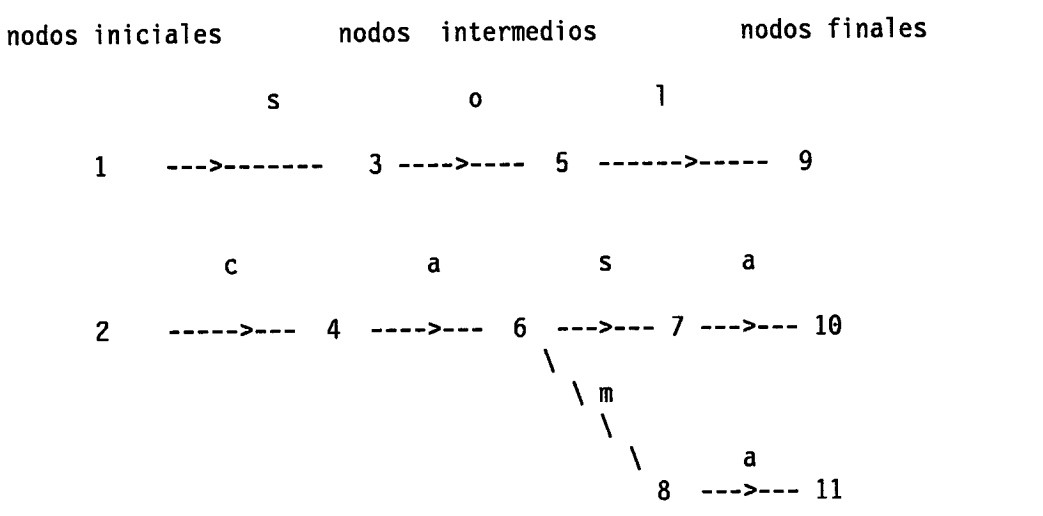


Figura 3. Red de transición de estados finitos

El autómata funciona de la siguiente manera: el aducto es una cadena de caracteres (p.ej. "casa") que se coloca en una cinta (*tape*) donde hay algún mecanismo que señale el estado en que nos encontramos y, por tanto, los caracteres que quedan por procesar. (Esta es la forma tradicional de explicar el funcionamiento de un autómata, en realidad la "cinta" y el "puntero" son posiciones en la memoria del ordenador). Una forma de representar un estado cualquiera de la red es mediante una secuencia de dos elementos: el número del arco en que nos encontramos y el resto del aducto (los caracteres que tienen que ser reconocidos). En nuestra red, el estado que resulta después de haber reconocido la primera letra de "casa" es

(4, a s a)

Una palabra o morfema serán reconocidos como elementos del diccionario si existe un camino (*path*) que vaya desde un estado inicial a otro final. En cambio, si no se puede llegar hasta el final de la cadena (imaginémonos una palabra inexistente como "casamón") o si se ha podido llegar al final de la cadena pero no nos hallamos en un estado final (por ejemplo, se ha escrito por error la palabra incompleta "cas"), entonces el reconocedor nos avisará de que dicha cadena no existe en el diccionario.

De esta forma, podemos escribirnos todo el diccionario definiendo para cada palabra o morfema su correspondiente camino en la red. Pero en cuanto tengamos algún nodo del que partan dos o más arcos necesitamos un procedimiento que nos permita buscar a través de todas las alternativas posibles. Gazdar y Mellish (1989) proponen un algoritmo general de búsqueda que se basa en la idea de crear un recipiente ("pool" en terminología informática) donde almacenar los distintos estados alternativos. Cuando se llega a un estado con varios arcos de continuación se escoge un estado, se calculan los estados válidos para el siguiente paso (desde dicho estado) y se añaden al "recipiente." Luego se escoge otro y se hace la misma operación, borrando siempre del "pool" el estado que se pone vigente. Se repite el mismo proceso hasta que se vacía el recipiente; si en algún momento se ha alcanzado un estado final, se anuncia su éxito. El algoritmo se muestra en la figura 4.

Como advierten los autores, la descripción está incompleta pues no especifica la estrategia para decidir qué alternativa se escoge en cada paso. Ésta puede ser o bien

```
(1)  créese el conjunto de estados iniciales alternativos (el "pool" inicial)

(2)  repítase el siguiente proceso:

     (2.1) selecciónese un estado (borrándolo del "pool")

     (2.2) SI es un estado final, ENTONCES párese y anúnciese 'éxito'
           EN OTRO CASO,

           (2.2.1) calcúlense los siguiente estados válidos desde este
                   estado

           (2.2.2) añádanse dichos estados al "pool"

(3) SI el "pool" está vacío, párese y anúnciese 'fallo'
```

Figura 4. Algoritmo general de búsqueda para redes de estados finitos.. Tomado de Gazdar y Mellish (1989:48)

"búsqueda en profundidad" (*depth-first*) o bien "búsqueda a lo ancho" (*breadth-first*). La primera implica que las últimas opciones añadidas al "pool" tiene preferencia, hasta que se agote el camino. Por el contrario, la búsqueda a lo ancho prueba antes los caminos más cortos.

Si comparamos los dos algoritmos en términos de eficiencia computacional obtenemos que el segundo (el de estados finitos) es bastante más rápido que el recursivo: una vez que ha encontrado el camino correcto (los primeros caracteres) gasta menos tiempo en cada paso. Por el contrario, la estrategia que contrasta recursivamente cada carácter de la cadena con las entradas de diccionario, deshace en infinidad de ocasiones el camino recorrido (*backtracking*) y no aprovecha los resultados parciales que han tenido éxito (cosa que no ocurre con el reconocedor de estados finitos, pues por definición no puede volver atrás).

En cambio, si lo vemos desde el punto de vista de la elegancia científica, que se sustenta sobre los criterios que simplicidad y generalidad, tenemos que dar preferencia al primer método que es declarativo, frente al estrictamente procedural.

5.1.3 Extracción de la información contenida en la palabra

Una vez reconocidos y segmentados los formantes de la palabra, hay que obtener la información (morfosintáctica, aunque según los sistemas también se puede extraer del diccionario la información semántica asociada a la entrada léxica), para completar así el proceso de análisis morfológico.

Dos son los procedimientos más usados. Uno consiste en tomar directamente la información asociada a ese morfema y asignársela a la palabra que se está construyendo. Esta suele ser la estrategia utilizada por las aplicaciones léxicas o los sistemas que hacen uso de autómatas de estados finitos. La idea es que una vez que se ha reconocido la forma léxica base su flexión está "dirigida" por reglas (ya sean las *clases de continuación* del modelo de estados finitos, o bien las reglas de generación del paradigma, en los sistemas paradigmáticos).

La segunda estrategia consiste en utilizar reglas sintagmáticas para procesar la información contenida en los morfemas. Esto implica que los morfemas son los elementos terminales de la gramática y la flexión es un fenómeno sintáctico y no léxico. Hay ciertas diferencias en la manera en que las reglas morfosintácticas recuperan la información de los morfemas, dependiendo si el peso del proceso se carga en la regla en sí o simplemente se deja en manos del mecanismo de transmisión de rasgos.

Veremos ejemplos de ambos procedimientos al hablar de los procesadores morfológicos. Una cuestión que hay que tener en cuenta es que las dos fases del procesamiento morfológico se invierten cuando se trata de generar palabras: hay que dar primero la información para luego escoger la variante alomórfica adecuada. Es decir, primero debemos proporcionar al sistema qué tipo de forma flexionada queremos obtener (por ejemplo, primera persona del singular del presente de subjuntivo del verbo X). Una vez que el sistema distribuye la parte de información pertinente a cada formante de la palabra, escoge la variante apropiada. La manera de hacer todo el proceso de la síntesis de palabras es muy diferente de un modelo a otro. Precisamente lo que más se ha criticado de los sistemas desarrollados hasta la fecha es que muchos no son capaces de utilizar la misma gramática para el análisis y para la generación. Esto se debe básicamente a que la formalización no es declarativa.

5.1.4 Procesos concatenativos y no concatenativos. La alomorfía

Como vimos en la sección 4.4, "Localidad, alomorfía y procesos concatenativos y no concatenativos" en la página 38, los procesos concatenativos se caracterizaban por no tener influencia en la estructura interna de la palabra o raíz a la que se aplicaban. Es el caso de la afijación (ya sea prefijación o sufijación). Estos procesos no presentan ningún problema a la hora de su tratamiento computacional (al igual que ocurría con su explicación teórica). Simplemente se trata de unir dos o más cadenas superficiales (aquí el concepto "concatenación" tiene un sentido de lo más literal[42]). El problema surge con los procesos que implican un cambio en la estructura interna de las raíces (o cadenas). Por ejemplo, la mayoría de las irregularidades del paradigma verbal español se deben a cambios vocálicos o consonánticos, o ambos. Estos cambios son provocados cuándo un determinado sufijo verbal se adjunta a una raíz verbal susceptible a dicho cambio. Veamos un ejemplo. En verbos como "a c e r t a r" (raíz regular "a c e r t") el acento en la forma flexionada recae en la penúltima o en la última sílaba, según los casos[43] . Cuando el acento cae en la última (es decir, forzosamente en la desinencia) no se produce ningún cambio en la raíz verbal:

```
acert -é    acert -ó    acert -arán    acert -ad ...
```

Pero cuando el acento se realiza en la penúltima sílaba, nos encontramos con dos posibilidades, dependiendo de si la desinencia es de una o más sílabas. En el primer caso, el acento cae en la raíz diptongando la vocal. En el segundo caso, el acento lo lleva la desinencia:

---

[42] Algunos lenguajes tienen instrucciones o predicados especiales para realizar esta tarea. Concretamente, el predicado *concat(X,Y,Z).* de PROLOG une dos cadenas, X e Y, en otra tercera, Z. En LISP existe la función (CONCAT .. ..) que hace lo mismo.

[43] Exceptuando la primera persona del plural del imperfecto del subjuntivo (*acertáramos*) y del condicional (*acertaríamos*).

```
(1)  aciert -o      aciert -as      aciert -an ...
(2)  acert -amos    acert -ais      acert -aron ...
```

Vemos, por tanto, que la adjunción de un morfema flexivo provoca en unos casos la modificación interna de la raíz. Esta alteración interna crea entonces una variante alomórfica, y la tarea que se presenta al lingüista es describir bajo que condiciones se produce esta modificación, de tal forma que las variantes alomórficas estén siempre identificadas con los morfemas flexivos que han provocado dicha alomorfía.

Evidentemente, las irregularidades verbales no se pueden explicar únicamente mediante procesos sincrónicos sino que en bastantes casos se deben a procesos diacrónicos (participios irregulares, evolución de verbos como *andar, estar*, pretéritos fuertes, verbos con varios paradigmas como *ser, ir*...). Lo habitual es tratar a los fenómenos sincrónicos como procesos regulares y a los segundos como formas lexicalizadas. También podemos considerar como procesos regulares los cambios ortográficos obligatorios como:

```
z --> c   delante de 'e' o 'i'
c --> qu  delante de 'e' o 'i'
g --> gu  delante de 'e' o 'i'
```

Pero volvamos a la problemática de los procesos no concatenativos. Un caso mucho más complejo muestran las lenguas semíticas. Las palabras del árabe presentan una estructura triconsonántica, a la que se suele considerar la raíz. Esta raíz puede recibir distintos tipos de procesos que modifican (o mejor, "rellenan") el esqueleto triconsonántico. Entre otros, pueden ser que una vocal sea larga o corta, que una consonante sea simple o geminada, o que se repita parte del material de una sílaba en la siguiente. Pero la relevancia de estos procesos reside en que cada uno de ellos implica un matiz semántico al verbo básico (por ejemplo, causatividad, reflexividad, etc.) que hay que reflejar al analizar o generar una palabra. Por eso, como señala Kay (1987), "our problem will therefore involve designing and abstract device capable of combining components of these three kinds into a single sequence."

La modificación interna junto con las convenciones ortográficas son los principales causantes de la alomorfía, tanto de la raíz como de los morfemas flexivos. Uno de los puntos donde se diferencian los distintos formalismos es precisamente el tratamiento de la variación alomórfica. En el apartado correspondiente de la "Introducción a la morfología flexiva" (concretamente en la sección 4.4) explicamos cómo se podía tratar la alomorfía con reglas de contexto. En la actualidad no se dispone de ningún lenguaje de programación o formalismo compilable que permita hacer uso de reglas contextuales o la notación entre corchetes, en el sentido que estamos acostumbrados a utilizar en la morfología teórica.

Básicamente hay dos procedimientos extendidos. En el modelo de gramáticas independientes del contexto se utilizan los rasgos y la unificación. El modelo en dos niveles de Koskenniemi utiliza reglas que ponen en relación la forma superficial con la léxica.

5.2 *Visión histórica de los procesadores morfológicos*

En los primeros tiempos de la morfología computacional la mayor parte del trabajo estaba en gran medida condicionado por los detalles de programación (lenguajes y máquinas) y se tenía muy poco en cuenta los criterios lingüísticos. Tanto es así que sólo ciertas partes del vocabulario se cubrían por la morfología, y los fenómenos morfológicos complejos se ignoraban, en general. Varios autores coinciden en señalar que esto se debió a la influencia del inglés:

> Such heavy emphasis on syntax on the one hand and almost total neglect of morpholgy on the other follows from the idiosyncracies of English. And due to the dominant role English has in the computational linguistic community, this somewhat unbalanced view permeates the computational linguistic literature. (Jäppinen y Ylilammi 1986:257).

Efectivamente, la flexión en inglés es bastante simple (sobre todo las reglas básicas) en comparación con las lenguas flexivas o aglutinantes. Como además no hay muchos fenómenos de fusión o no concatenativos y se considera que es una lengua bastante analítica (gran número de palabras de morfema único) no es extraño que en un principio se pensara que era más práctico colocar todas las formas plenamente flexionadas en el diccionario, ya con toda su información, para empezar por el análisis sintáctico. A este método se le denomina *full listing hypothesis* (FLH), es decir, el diccionario no tiene raíces ni morfemas, sino formas completas. No existe, por lo tanto, ningún tipo de procesamiento morfológico dentro de este enfoque.

Fueron precisamente los investigadores que trabajaban con lenguas con una flexión rica y productiva los que han desarrollado procesadores morfológicos pensando sobre todo en la economía de computación. Nótese, por ejemplo, que un verbo del español puede tener más de 80 formas conjugadas (utilizando sólo los tiempos más usuales), pero en finés puede tener más de un millar. Se comprende así por qué los lingüistas computacionales escandinavos han revolucionado esta disciplina. Actualmente todo el mundo está de acuerdo que el procesamiento morfológico es necesario, incluso en lenguas tan aislantes como el inglés. Con ello se consigue ampliar el diccionario con el menor coste posible de entradas y, además, los procesadores morfológicos han demostrado que lo hacen eficientemente.

En un artículo reciente Schraeder y Willée (1989) calculan que hay algo más de 25 sistemas de procesamiento morfológico (implementados y documentados) aunque hay que tener en cuenta que se centran en los sistemas para el alemán. Por poner un ejemplo, sólo citan un sistema para el español, cuando existen unos cuantos como veremos más adelante. Podemos afirmar, sin mucho riesgo de equivocarnos, que en la actualidad cualquier sistema de procesamiento del lenguaje natural de moderadas pretensiones tiene un componente morfológico. Por esta razón, sólo nos dedicaremos a analizar los modelos más influyentes, con referencia explícita a los sistemas para el español.

5.3 Distintos modelos de procesadores morfológicos

La clasificación de los procesadores que vamos a presentar está ordenada por los modelos de Hockett, lo que no quiere decir que los autores de los procesadores se hayan adherido explícitamente a la corriente que les asignamos. En realidad, casi ningún procesador morfológico ha escogido un modelo teórico como referencia para probar su pertinencia. Como reconoce Karlsson (1989):

> Very little work has, in fact, been done on direct computational testing of autonomously established linguistic models.

En la exposición no hemos pretendido ser exhaustivos sino presentar los ejemplos más representativos y mostrar algunos casos particulares del español.

5.3.1 Procesadores basados en el modelo Palabra y Paradigma

5.3.1.1 *El procesador de Hellberg*

Uno de los modelos más tempranos de análisis morfológico fue el diseñado por Steffan Hellberg (Hellberg 1972, 1978, citado de Karlsson 1989) para la morfología del sueco. El sistema pertenece al tipo paradigmático y tenía entre sus objetivos conseguir una cobertura total del sistema morfológico con todas sus variaciones, limitaciones y extensiones. La descripción completa contiene 235 paradigmas y el lexicón 8.600 lemas. El sistema no hacía uso de reglas de reescritura en el sentido clásico. Cada paradigma estaba numerado y contenía llamadas a ciertas subrutinas también numeradas que se ejecutaban de acuerdo con las características de la entrada. Por ejemplo, el paradigma 101 (que se muestra en la figura 5) comprobaba en primer lugar la longitud de la desinencia.

```
101 flick/a     nn utr
LO ≠ ---> 92                          flicka(s)
   ---> e ---> 96                     flickan(s)
                                      flickor(s)
> 5                                   flickorna(s)
                                      flick(s)-
SH ---> 11                            flicke-
 ---> (10)
```

Figura 5. Ejemplo tomado de Karlsson 1989

Si la cola era mayor que 5 se llamaba a las subrutinas para compuestos y derivados, mientras que si era menor se llamaba a las subrutinas para finales flexivos. La raíz se recuperaba del lexicón. Las rutinas especifican segmentos aceptables así como llamadas a otras subrutinas, hasta que se completa el análisis de la cadena.

Karlsson desarrolló un analizador morfológico para el finés siguiendo estas mismas líneas (Karlsson 1985). La idea fundamental de su modelo paradigmático era que el análisis morfológico estuviera guiado por un léxico con raíces concretas de nivel fonológico derivadas mediante patrones de un diccionario ordinario de formas básicas.

5.3.1.2 *El analizador morfosintáctico de textos en español desarrollado en Pisa*

El analizador morfosintáctico que desarrollaron Rati, Saba, Catarsi y Cappelli en el Instituto di Linguistica Computazionale de Pisa ejemplifica las características de los procesadores morfológicos de primera generación. Se desarrolló durante los últimos años de la década de los 70 y principios de los 80[44] .

En primer lugar, se trata de un procesador morfológico orientado a la lexicografía automatizada más que al procesamiento sintáctico y semántico. Esto explica, en gran medida, que sólo trate el análisis de las palabras y no sea capaz de generarlas (el sistema no está pensado para la generación sino únicamente para el reconocimiento de formas). En palabras de sus autores, "el sistema puesto a punto permite obtener semi-automáticamente la lematización y el análisis morfológico del 80 % de las ocurrencias de un texto" (Ratti *et al.*, pág. 9).

Por tratarse de una aplicación lexicográfica, su aplicación fundamental se encamina al análisis textual[45] que, posteriormente, puede utilizarse para estudios de frecuencia de palabras, su posición y compatibilidad con respecto a otras, así como la productividad de afijos y cambios de significado. Se puede encuadrar por tanto en la línea de trabajos como los de Josse de Kock, en la Universidad de Lovaina.

El analizador está organizado en tres fases:

1. un Pre-procesador
2. un Procesador morfológico
3. un Procesador morfosintáctico.

El *Pre-procesador* tiene como misión aislar y analizar las formas "sinsemánticas" (es decir, las que pertenecen a categorías cerradas como los pronombres, preposiciones, conjunciones, etc.) y los sintagmas más frecuentes. Su método consiste en buscar esas palabras en dos listas (una para las formas sinsemánticas y otra para los sintagmas más frecuentes) y compararla con la forma del texto que se está examinando. Si hay éxito en la comparación, se le asigna la categoría y lema correspondiente. Con este preprocesamiento ellos calculan que se reconoce aproximadamente un 50% de las ocurrencias del texto. Téngase en cuenta que de esta manera se "reconocen" los pronombres, artículos y cuantificadores como formas flexionadas directamente en el diccionario pero no se "analizan" como formas con raíz y morfema flexivo.

El *Procesador morfológico* toma las formas que no han sido analizadas en la fase anterior, es decir, los nombres, verbos y adjetivos. Un algoritmo de segmentación basándose en listas de afijos y sufijos segmenta la palabra en todas sus particiones posibles. Asume entonces que lo que no es ni prefijo ni sufijo tiene que ser raíz y

---

[44] El documento en que nos basamos es una publicación sin fecha (calculamos que de los primeros años del 80) impresa en Pisa por *Giardini Editori e Stampatori.* Agradecemos al profesor Marcos Marín que nos la haya facilitado.

[45]

> Un texto analizado con este sistema se presenta por lo tanto como un conjunto de datos organizados en modo homogéneo y preparados para sucesivas investigaciones de tipo formal y estadístico... (Ratti *et al*, pág 9).

comprueba que se halle en el diccionario. Consulta la entrada apropiada y encuentra en ella qué prefijos y sufijos admite, así como un código del paradigma flexivo que tienen que usar y de esta forma se extrae el análisis y la lematización. Si no puede encontrar ninguna raíz que coincida, aplica unas reglas de "transcripción morfografémicas" que dan cuenta de fenómenos como la desaparición del acento gráfico en los plurales de algunos nombres, diptongación en la raíz de los verbos, etc. El sistema retorna todos los posibles análisis y dispone de un *procesador morfosintáctico* muy sencillo que mediante reglas basadas en las homografías más corrientes del español resuelve la mayoría de las ambigüedades.

5.3.1.3 *Procesador morfológico de IBM para el español*

Otro de los analizadores que podemos encuadrar en este marco teórico es el desarrollado por el grupo de lexicografía computacional del Centro de Investigación UAM-IBM. Se diseñó en un principio como componente de un corrector de textos y también para ser usado dentro de un diccionario de sinónimos. Los autores reconocen expresamente que el fundamento teórico de su sistema es el modelo Palabras y Paradigma:

> La filosofía de nuestra clasificación morfológica en ordenador sigue básicamente estos principios que favorecen la simplificación y el máximo aprovechamiento de los recursos tanto informáticos como lingüísticos. En toda ella subyace el concepto fundamental de *analogía* que nos permite predecir una determinada forma **por analogía** con otra que ocupa su mismo lugar en el paradigma modelo. (Rodríguez Magro *et al* 1990: 493).

El sistema sigue muy fielmente la idea clásica de paradigma. Para los nombres se construyen distintos modelos de acuerdo con el número de formas que tienen (dos o cuatro), su última letra y cualquier otra irregularidad que puedan presentar. Los adjetivos tienen una clasificación paradigmática bastante similar a la de los nombres, pero con algún paradigma extra para la apocopación (*buen/bueno*). Un paradigma típico sería el de nombres masculinos con plural en 's':

```
*Añadir s, sólo masculino ('abecé, ácido')
 @901@                    SPMN
  ms
  mp       s
```

Para los verbos se construyen también modelos estructurales que toman en cuenta todos los posibles casos distintos del sistema verbal español. Por ejemplo, una forma en blanco dentro del paradigma indica que el verbo es defectivo. También pueden incluirse dobles formas como las del imperfecto de subjuntivo. El diccionario contiene la *forma básica* de cada palabra, esto es, el segmento más largo a partir del cual pueden derivarse todas las formas añadiendo sufijos. Estas formas básicas no siempre coinciden con la noción de *raíz*: para la palabra 'examen', por ejemplo, la forma básica es 'ex' que constituye el segmento invariable, ya que el plural es 'exámenes' (Rodríguez Magro *et al* 1990: 495). Cada una de estas formas tiene asociado un código que indica el número del paradigma al que pertenece. La documentación de que disponemos no explica mucho acerca del proceso mediante el cual se llega desde la forma flexionada que aparece en el texto a la forma básica del diccionario, se menciona sólo que se hace mediante el uso de reglas. En los casos de irregularidades muy fuertes, en los que no existe prácticamente ninguna relación formal entre las cadenas superficiales, se recurre

a la supleción; por ejemplo, para las formas 'iba' y 'voy'. Como la cantidad de paradigmas que hay que manejar es muy grande el sistema dispone de métodos para abreviar las formas y para compactar los diccionarios que quedan reducidos a un tamaño manejable. El sistema lleva funcionando ya algún tiempo con resultados satisfactorios, a pesar de que desde el punto de vista estrictamente lingüístico tenga algunas soluciones que parecen forzadas como la reducción de algunas raíces a segmentos mucho menores o la existencia de paradigmas diferentes para formas que gramaticalmente se consideran analógicas.

5.3.1.4 *El modelo paradigmático de Calder*

El sistema más reciente y más elaborado basado en el modelo de Palabras y Paradigma es el de Jonathan Calder (Calder 1989). Este modelo se basa en la existencia de paradigmas y de *ecuaciones de cadena* para especificar cómo se combinan los morfemas. Una especificación de cadena (*string specification*) es una secuencia de variables del conjunto de variables V y de constantes, ligadas mediante el operador '+'. Por ejemplo,

```
w+a+l+k+s           es una cadena
W+s                 es una especificación de cadena
w+a+l+k+s = W+s     es una ecuación de cadena
```

La operación que determina la asignación de valores a variables de forma que cumplan la ecuación se denomina *unificación de cadenas.* Una entrada léxica S:P asocia una cadena S que no contiene variables a un conjunto P de propiedades gramaticales. Una regla léxica es un trío formado por un nombre, propiedades de la cadena de entrada y propiedades de la cadena de salida. A continuación se muestra la estructura general de las reglas y un ejemplo concreto de regla (la de la tercera persona del singular del presente de los verbos ingleses):

```
        <Name, IS:IP, OS:OP>

lexical_rule(3sg,
     [verb,base] → [verb,finite,3sg])
```

El nombre de la reglas es '3sg' y se aplica sobre la forma base de los verbos (la cadena y las propiedades de entrada) y da como resultado un verbo en forma finita con los rasgos de la tercera persona del singular.

Un paradigma está formado por un nombre que lo identifica; una cadena y una serie de propiedades que controlan la aplicación del paradigma; un conjunto de reglas léxicas que producen las distintas formas de ese paradigma; y una cadena de salida. El primero de los dos paradigmas que se muestran es el correspondiente a la conjugación de los verbos regulares en inglés, mientras que el segundo muestra un caso particular del plural de las palabras terminadas en 'o'.

```
table(verb,          Verb:[verb,base,Past = Verb+ed],

[base    3sg     non3sg      past_participle      past      passive      progressive]
 Verb    Verb+s  Verb        Past                 Past      Past         Verb+ing]    )


table(piano,
            S:[noun,singular,
            S = {piano,piccolo,....}],

   [singular, plural]
   [S       , S+s   ])
```

Figura 6. Ejemplos de paradigmas según Calder

La naturaleza declarativa del sistema implica que no se encuentra ligado a una interpretación computacional concreta. Por lo tanto, suponemos que la lematización se lleva a cabo por procedimientos similares a los que usan otros sistemas, aunque el autor no se refiere a ellos. Las consideraciones de mera eficiencia pueden hablar en favor de una realización por medio de una máquina de estados finitos, sin embargo Calder demuestra que ciertas construcciones exhiben un patrón independiente del contexto del que difícilmente puede dar cuenta una aproximación de estados finitos. Una de las aportaciones más importantes del modelo de Calder es precisamente una formalización rigurosa que le sirve para poder demostrar la existencia de este tipo de problemas.

En resumen, el tratamiento informático de la morfología siguiendo el modelo teórico PP se divide en los siguientes pasos:

1. reconocimiento de la raíz mediante rutinas que contrastan la cadena con el diccionario,
2. una vez encontrada una raíz candidata, se utiliza el código o la información que indica a qué paradigma pertenece y se generan las formas completas del paradigma,
3. finalmente, para el reconocimiento se contrasta si la cadena que estamos analizando coincide con alguna de las formas del paradigma.

Algunas características notables de estos sistemas son:

- no necesitan usar símbolos abstractos (como morfemas ∅ o morfofonemas del nivel léxico) puesto que se basan en la representación superficial,
- capturan las nociones lingüísticas de paradigma (o grupo de palabras con las mismas particularidades morfológicas) y la de analogía, que han demostrado su utilidad en terrenos aplicados como la enseñanza de lenguas,
- el diccionario contiene una sola entrada para cada lexema, pero en cambio hay un número muy considerable de reglas que está en función del número de paradigmas,
- son muy flexibles porque permiten dar cuenta de cualquier tipo de irregularidad,

- por último, son bastante eficientes computacionalmente y permiten el uso de las mismas subrutinas dando cuenta de similitudes entre distintos paradigmas.

5.3.2 Modelos basados en Elementos y Proceso

Agruparemos en este epígrafe a la mayoría de los modelos que utilizan autómatas de estados finitos, pues todos ellos implican que durante el análisis de una palabra se siguen unos caminos que implican distintos procesos sobre la forma básica (Karlsson 1989).

5.3.2.1 La Gramática Morfografémica de Kaplan y Kay

Uno de los sistemas más tempranos e influyentes de los basados en esta aproximación es el *Morphographemic Transducer* de Kaplan y Kay. Teóricamente su importancia reside en la posibilidad de traducir las reglas fonológicas (en el sentido de la gramática generativa, es decir, reglas morfofonológicas) a un *transducer* de estados finitos con dos cintas muy similar a un autómata de estados finitos simple, excepto por el número de cintas. De esta forma se introducía la idea de que muchas de las alternancias morfofonológicas son simples en el sentido de que se pueden formalizar con máquinas abstractas muy sencillas (los autómatas de estados finitos pertenecen al tipo más restringido de la jerarquía de los lenguajes formales). El *Morphographemic Transducer* supone la existencia de un nivel de representación léxica y otro nivel de cadenas superficiales. Suponiendo que tuviéramos la representación léxica de un verbo como *flY* y que el *transducer* recibe la cadena superficial *flies*. El *transducer* tomará el primer par de caracteres *f:f* en el estado 0. Este par no provoca ninguna modificación en el estado, lo mismo que el segundo par *l:l*. En el estado 0 hay una correspondencia permitida entre la representación léxica *Y* y la superficial *i* que produce una transición hacia el estado 4. Desde este estado hay un retroceso al estado 0 si el siguiente par es *e:e*. Finalmente, el par *s:s* no cambia la situación. La cadena de entrada se ha consumido y nos encontramos en un estado de los designados como finales, por lo tanto, el *transducer* acepta la cadena *flies*. Posteriormente unas rutinas recuperan la representación morfológica apropiada del diccionario. Una de las ideas interesantes es que el diccionario no es un elemento estático sino que es el que verdaderamente dirige al *transducer* evaluando la admisibilidad de las parejas de caracteres.

Al menos teóricamente existe la posibilidad de colocar todos estos *transducers* en cascada y construir un enorme autómata con cientos o miles de estados. Esta idea ha tenido varios desarrollos posteriores. Entre ellos un autómata para la morfología del español desarrollado por los laboratorios AT&T (Tzoukermann y Liberman 1990) que consta de unas 55.000 mil entradas básicas (cada una de ellas un pequeño autómata) y una adición de 3.000 arcos más para dar cuenta de las posibilidades flexivas y derivativas del español. Los autores comentan que se requieren unos 2.500 arcos para representar la alomorfía de la raíz. A pesar de lo engorroso del procedimiento, parece que es capaz de procesar varios cientos de palabras por segundo en una SUN 4.

Existen ya en España varios sistemas de análisis morfológico que utilizan una red de estados finitos. Concretamente para el castellano se ha desarrollado MARS (*Morphological Analysis for Retrieval Support*) de la empresa SIEMENS (Meya 1986) y también AM de Intersoftware (Martí 1986a, 1986b). Ambos sistemas prácticos se usan para recuperar información de una base de datos: el analizador morfológico se utiliza para "la identificación de formas, su análisis y la asociación de información a cada una de ellas" (Meya 1986).

5.3.2.2 *Morfología en dos niveles de K. Koskenniemi*

Pero la evolución más conocida de la "gramática morfografémica" de Kaplan y Kay es sin duda la morfología en dos niveles de Koskenniemi (*Two-level Morphology*). Koskenniemi tomó de estos autores la idea de relacionar cadenas superficiales y léxicas mediante un autómata y, por otra parte, tomó el concepto de pequeños lexicones relacionados entre sí del trabajo de Kartunnen y sus ayudantes en TEXFIN (un sistema de análisis morfológico del finés por ordenador).

Las dos características principales del modelo en dos niveles son:

1. Es tanto una teoría como un formalismo para la descripción de todos los fenómenos morfológicos (flexión, derivación y composición).

2. Puede ser implementado en programas para analizar y generar palabras.

El formalismo está pensado para proporcionar un modelo simple pero general que permita describir la estructura de las palabras en cualquier lengua. Esta propiedad ("language independency") la presenta el autor como una de las novedades de su propuesta. Dicho formalismo tiene dos componentes principales:

- *El lexicón o diccionario*: se divide en varios sublexicones para raíces y distintos tipos de afijos. Por medio de un mecanismo de enlace se definen las secuencias o combinaciones posibles de elementos de distintos sublexicones. Estas secuencias se denominan "clases o esquemas de continuación" (*continuation classes*). Por otro lado, hay que distinguir entre las *representaciones léxicas* de las palabras, que son las que aparecen en el lexicón, de las *representaciones superficiales*, que son simplemente las formas escritas.

- *Las reglas*: este componente define las relaciones que hay entre las representaciones léxicas y superficiales. Este componente es necesario debido a que en muchas ocasiones ambas representaciones para un mismo morfema difieren (pensemos por ejemplo en los distintos alomorfos que puede puede tener la raíz de un verbo irregular en castellano). La diferencia con las reglas de la fonología/morfología generativa reside en que las reglas en dos niveles se encargan exclusivamente de definir las condiciones en las que una correspondencia es correcta o incorrecta.

La importancia práctica de este modelo, como insiste repetidamente Koskenniemi, se basa en que la teoría puede ser implementada mediante un programa eficiente, fundamentalmente porque las reglas en dos niveles se pueden codificar como un autómata de estados finitos, lo que las convierte en rápidas y simples.

Otra de las innovaciones de este modelo es su *bidireccionalidad*. Es decir, una vez establecidas las reglas y los subdiccionarios para una lengua cualquiera, el programa es capaz de reconocer y producir palabras de esa lengua. Koskenniemi explica la novedad de su modelo "because it had long been an accepted position in computational linguistics that morphological analyzers are distinct from synthesizers, and that each language needs a morphological program tailored specially for it" (Koskenniemi 1985a:2). Recogemos a continuación el esquema del modelo en dos niveles, donde las flechas continuas representan el proceso de reconocimiento y las flechas discontinuas el proceso de generación de palabras.

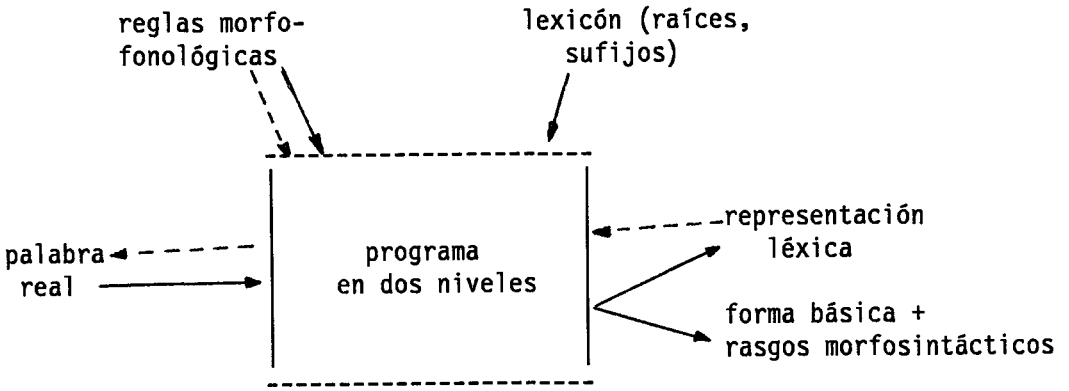


Figura 7. Tomado de Koskenniemi (1985a)

El lexicón en dos niveles está compuesto de sublexicones. El mayor de ellos contiene las raíces, mientras que los restantes contienen distintos tipos de afijos. Las clases de continuación se usan para definir la estructura morfotáctica de las palabras y consisten en un puntero al nombre del sublexicón que contiene los afijos permitidos. La figura que sigue muestra una definición rudimentaria de las clases de continuación para el finés (para todos los ejemplos que aparecen del finés seguimos (Koskenniemi 1985a)):

```
CONTINUATIONS
        ( /N = Numb # )
        ( C = Case )
        ( P = Poss Clitics # )
        ( K = Clitics # )
        ( /V = Tense )
        ( Pe = Person )
        ( Root = Root )
        ( # = # )
```

Los símbolos "/N," "C," etc. son los nombres de las clases de continuación y "Num," "Case," "Poss," etc. son nombres de sublexicones. El símbolo "#" quiere decir que la forma puede terminar ahí y por lo tanto, los dos sublexicones anteriores ("/N" y "C") son opcionales. Algunos de los sublexicones correspondientes serían:

```
LEXICON  Case
     ssA          P        "INE";
     A            P        "PTV"
LEXICON  Poss
    /ni           K        "SG1";
    /si           K        "SG2";
LEXICON  Tense
    0             Pe       "PRES ACT";
    I             Pe       "PAST ACT";
    vA            /N       "DVMA"
    .             .          .
    .             .          .
    .             .          .
```

Cada entrada en un sublexicón tiene tres componentes: la representación léxica ("/ni," "/si"), el nombre de una clase de continuación ("K"), e información referente a la entrada ("SG1" e "SG2") que puede ser más o menos compleja de acuerdo con las necesidades posteriores del sistema.

El componente de reglas describe las distintas formas superficiales que pueden adoptar las raíces y los distintos sufijos. La función de las reglas es hacer la comparación entre las representaciones léxicas y las superficiales. Por ejemplo, la palabra finesa *laseja* ('algunos vasos') tiene las siguientes representaciones:

```
      representación léxica:   l a s i I A
representación superficial:    l a s e j a
```

La representación léxica se compone de la raíz "lasi," el morfema de plural "I" y la desinencia de partitivo "A." "I" se usa para representar el morfema de plural porque tiene algunos efectos sobre las vocales precedentes y se comporta de una forma distinta a una "i" normal. La "A" mayúscula del partitivo representa un archifonema que aparece como "a" o "ä" dependiendo de la armonía vocálica de la raíz precedente.

Las reglas comparan estas dos representaciones. Segmentos léxicos invariantes se realizan como sí mismos, luego la correspondencia entre los tres primeros pares es trivial. Los tres restantes son ejemplos de variaciones morfofonológicas.

1. La "i" final de la raíz se realiza normalmente como sí misma excepto si va seguida del plural "I" en cuyo caso se realiza como "e"
2. El plural oblicuo "I" se realiza normalmente como "i," pero si se encuentra entre dos vocales se convierte en "j"
3. El archifonema se realiza "A" se realiza como "a" si la raíz contiene vocales con armonía vocálica hacia atrás ("a," "o" o "u"), en otro caso se realiza como "ä"

La tarea del formalismo de reglas en dos niveles es expresar condiciones necesarias y suficientes para las distintas realizaciones alternativas de un carácter. No se trata de transformar los caracteres léxicos en realizaciones sino de expresar

restricciones a la ocurrencia de un determinado par de caracteres. Las dos secuencias que hemos visto se pueden representar como un conjunto de pares:

```
(l,l)  (a,a)  (s,s)  (i,e)  (I,j)  (A,a)
```

La siguiente regla establece que (i,e) solamente puede ocurrir delante del plural "I":

```
i
    <=>   ----  I
e             =
```

El contexto expresado a la derecha es puramente gramatical aunque en otras reglas puede tratarse de un contexto fonológico. Requiere la presencia del morfema de plural, con independencia del carácter superficial en que se realice (el signo " = " indica precisamente esto). La parte de contexto de las reglas permite usar ciertas convenciones de símbolos como V para representar cualquier vocal. Las reglas no están ordenadas por lo que se deben considerar todas simultáneamente. Todas las condiciones de las reglas deben satisfacerse.

Como hemos dicho, la finalidad de las reglas es aceptar o rechazar pares de caracteres. Aceptar o rechazar caracteres es exactamente lo que hace un autómata de estados finitos. Las reglas en dos niveles están basadas en *expresiones regulares* que se corresponden muy de cerca con los autómatas de estados finitos. Una secuencia de estados corresponde al reconocimiento del contexto a la izquierda y otra a la verificación del contexto a la derecha. Se usa una representación tabular en lugar de la gráfica, porque es más sencilla de mantener y de codificar. Los estados se numeran del 1 en adelante y las columnas se reservan para los distintos pares de caracteres. Daremos como ejemplo la regla acerca de (i,e). El conjunto de pares posibles se puede desglosar en cuatro partes relevantes para esta alternancia: (i,e), *i* realizado de cualquier otra manera (i,=); el plural *I* cualquiera que sea su realización (I,=), y cualquier otro par distinto de los anteriores (=,=). Cada una de estos pares tendrá una representación en la tabla del autómata:

	i e	i =	I =	= =
1:	2	3	1	1
2:	0	0	1	0
3:	2	3	0	1

El estado 1 es el inicial y el autómata permanece en él hasta que encuentra alguno de los pares que provocan la transición como pueden ser (i,e) o (i,i). El autómata va al estado 2 cuando encuentra un par (i,e). Desde ese estado sólo puede continuar si lo que se encuentra es el plural "I." El estado 3 sirve para garantizar que otras realizaciones de "i" no estén seguidas de "I" y es también un estado final, puesto

que el estado 2 no lo era (requería la presencia del plural "I," luego la palabra no podía acabar ahí).

La transformación de las reglas en autómatas se puede hacer mediante un compilador de reglas. Recientemente se ha conseguido una versión de este compilador que funciona en ordenadores personales (PC-KIMMO).

Koskenniemi desarrolló la primera versión del modelo en Pascal y ha sido convertida posteriormente a otros lenguajes y adaptada para funcionar en distintos entornos. Existe una versión que puede funcionar en ordenadores AT y manejar unas 3.000 entradas léxicas.

El sistema de Koskenniemi presenta varias ventajas importantes como son su bidireccionalidad; la economía con la que puede tratar los fenómenos morfografémicos y morfofonológicos regulares (almacenando una sola entrada léxica y haciendo uso de reglas válidas para un grupo bastante grande de cambios); y la eficiencia que le proporciona su implementación por medio de dispositivos tan sencillos como los autómatas. Sin embargo, su eficacia no es tan manifiesta cuando hay que manejar fenómenos no concatenativos y en ocasiones tiene que recurrir finalmente a la supleción. Por tanto, aunque su autor insiste que es un modelo universal para el reconocimiento y síntesis de palabras, creemos que es mucho más apropiado y eficaz para el tratamiento de la afijación y los cambios morfofonológicos y gráficos en las fronteras de morfemas, propios de las lenguas aglutinantes. En cambio para las lenguas donde las fronteras entre morfemas no están definidas nítidamente y muchos morfemas tienen modificación interna, como las lenguas semíticas o las romances, se han propuesto ampliaciones del formalismo que se alejan del modelo original.

En España, un grupo de la Facultad de Informática de San Sebastián ha construido un analizador del euskera basado en el modelo de dos niveles (Aguirre *et al* 1989). Su primera aplicación ha sido un corrector ortográfico del euskera.

5.3.3 Modelos basados en Elementos y Colocación

La mayoría de los procesadores que incluimos en este apartado no reconocen explícitamente su adscripción a ningún modelo morfológico particular. Creemos que pueden encontrarse más próximos a este modelo que a los anteriores porque utilizan la *supleción* o *suplencia* como medio para representar la alomorfía (por contraste con la *analogía* del modelo paradigmático, y la *forma básica* del modelo de procesos). Es decir, cada variante alomórfica se representa en el diccionario por medio de una entrada. Esto implica, indudablemente, un mayor número de entradas en estos procesadores que en los otros (una de las razones, por otra parte, que inducen a utilizar los modelos paradigmáticos y de estados finitos para aplicaciones léxicas). Por tanto, la tarea fundamental es asignar correctamente cada alomorfo con el correspondiente que le sigue o precede. Normalmente esto se hace con reglas y con la comprobación de rasgos de los alomorfos. Debido a esto se las suele conocer como gramáticas morfosintácticas de rasgos.

La gran dificultad de este modelo es establecer declarativamente las condiciones de combinación de los alomorfos. Trost (1990:371) manifiesta:

> Conventional morphosyntactic grammars do not allow to describe non-concatenative parts of morphology declaratively.

En iguales términos se expresa Koskenniemi (1985a), insistiendo en que los procesadores convencionales de reglas no son capaces de utilizar la misma gramática para el análisis y la generación (véase cita en la sección anterior). Ambas afirmaciones, sin dejar de ser históricamente ciertas, creemos que están equivocadas y la intención de nuestra tesis es demostrarlo. Por principio no hay ninguna limitación que impida que las formalizaciones de los fenómenos morfológicos se puedan hacer con gramáticas morfosintácticas de rasgos. Es cierto que la experiencia ha demostrado que no es una tarea fácil, pero también hay que tener en cuenta que la mayoría de los procesadores morfológicos de este tipo se han concentrado exclusivamente en el reconocimiento de formas. Es decir, los sistemas de análisis operan, en principio, sobre la suposición de que las palabras mal formadas no van a aparecer (salvo errores ortográficos) en los textos escritos o en las preguntas de un usuario a un sistema de consulta de bases de datos. Por lo tanto, sólo necesitan que, por medio de rasgos, se identifiquen los alomorfos de un mismo morfema. Así, por ejemplo, en la Gramática del español para *User Specialty Languages* (USL) de Luis de Sopeña los alomorfos del plural se identifican por un valor numérico (tomado de Sopeña 1982):

```
ES:     NLEV=12,   ALEV=32
 S:     NLEV=12,   ALEV=32
```

La regla de plural de los nombres simplemente comprueba que el constructo FINMOT (el nombre dado a los sufijos flexivos en el formalismo USL, ver Zoeppritz (1984)) se une al constructo NOMEN (el nombre en singular, con su rasgo de género ya asignado) para formar otro NOMEN con el valor +PL (plural):

```
NOMEN<+FLE,... +PL, -SG> <---' NOMEN< -FLE,...> FINMOT<NLEV=12>
```

Esta regla establece simplemente que cualquiera de los dos alomorfos de plural puede unirse a un nombre que no esté flexionado (-FLE). Pero no especifica las condiciones de buena formación, de tal forma que permite analizar como palabras correctas tanto *casas* como *casaes* (y también *camiónes, camións, camiones* y *camions*, al incluir dos alomorfos para la raíz).

Aunque esto pueda parecer inadmisible desde un punto de vista teórico, es bastante eficiente a la hora de analizar palabras en una aplicación concreta. Hay que tener en cuenta que la mayoría de estos sistemas están orientados hacia la sintaxis y no están especialmente motivados por invertir tiempo en construir formalizaciones declarativas de la morfología: basta con que sepan reconocer *todas* las posibles palabras que aparezcan en el sublenguaje de aplicación. De hecho, puede considerarse como un signo de robustez ante los errores el que permita reconocer palabras no gramaticales.

En el mismo sentido se manifiesta Hallebeck (1989) al hablar de las reglas morfológicas incluidas en la gramática formal del español del proyecto ASATE de la Universidad de Nimega:

> La gramática pretende ofrecer una descripción del español escrito actual tanto en el aspecto sintáctico como el morfológico y lexical. Va destinada al análisis de realizaciones concretas del lenguaje; es decir, al reconocimiento de la estructura lexical, morfológica y sintáctica de frases. No es pues una gramática productiva.

> Esto trae como consecuencia que las reglas de la gramática pueden ser más liberales porque no están sujetas a la condición de que sólo se produzcan manifestaciones gramaticales o aceptables. (Hallebeck 1989: 104).

Por supuesto, esta relajación en las condiciones de gramaticalidad no es privativa de los sistemas del español. Por ejemplo, Whitelock(1988) reconoce que su gramática morfosintáctica categorial para el japonés basada en la unificación sobregenera debido a la ausencia tanto de tratamiento para la subcategorización verbal como porque "morphografemic phenomena are not treated with any degree of generality." En este caso, los objetivos de la gramática eran investigar las posibilidades de caracterizar la morfosintaxis del japonés combinando las técnicas de las gramáticas categoriales y los formalismos de unificación. En este tipo de trabajos lo que importa es comprobar las posibilidades generales del enfoque, en lugar de entrar en detalles. En resumen:

> In a computational context, *overgeneration* by the grammar is not necessarily a problem if the goal is to *recognize* or *understand* weel-formed language. On the other hand, it could prove fatal if the goal is to *produce* well-formed language. (Gazdar y Mellish 1989:109)

A pesar de que este planteamiento sea bastante habitual en las gramáticas morfosintácticas existen otras aproximaciones. El proyecto EUROTRA de la Comunidad Económica Europea, por ejemplo, tiene como objetivo final lograr la traducción automática entre todas las nueve lenguas oficiales de los países comunitarios. Esto supone trabajar con 72 pares de lenguas y conseguir para cada una de ellas tanto análisis como generación. Por lo tanto se planteó desde el primer momento la necesidad de conseguir gramáticas reversibles en todos los niveles incluido el morfológico. Esto suponía preocuparse de establecer explícitamente las condiciones de buena formación gramatical de tal manera que sólo sea posible analizar o producir las que sean gramaticales.

EUROTRA utiliza una gramática independiente del contexto implementada en un formalismo basado en la unificación para combinar el procesamiento morfológico y sintáctico (Lau y Perschke 1987). Una de las razones que influyeron en la decisión de no escoger el modelo en dos niveles de Koskenniemi era la dificultad de conseguir una coherencia entre los diccionarios de las distintas lenguas, teniendo en cuenta que el conocimiento fonológico de la propia lengua es fundamental en la codificación y además evitar puntos teóricos controvertidos como las fronteras de morfema, los archifonemas y los caracteres vacíos. Nótese que las escuelas lingüísticas predominantes en cada país pueden influir a la hora de aplicar estos criterios. Por lo tanto, se decidió separar en dos fases el análisis morfológico: en la primera se tratan los aspectos morfografémicos, es decir, la normalización de caracteres y , sobre todo, la alomorfía y en la segunda se tratan los fenómenos morfosintácticos. En el primer nivel se establecen qué combinaciones de letras forman las posibles cadenas básicas de una lengua determinada.[46]

El segundo nivel combinaba los distintos morfemas resultantes mediante rasgos y transmitía la información al nivel superior de acuerdo con las convenciones de transmisión de rasgos.

El autor estuvo encargado de la realización del módulo morfológico del español en su primera versión (Moreno Sandoval 1989). Esta tesis tuvo su origen en aquel

---

[46] En la actualidad este nivel está subsumido por un segmentador.

trabajo y, aunque se ha desarrollado fuera del formalismo de EUROTRA incorporando un mecanismo especial de concatenación, ha estado guiada por la idea certeramente expresada por Lau y Pershcke acerca de que la declaratividad es necesaria y posible para dar cuenta de la alomorfía dentro de un modelo independiente del contexto.

5.3.4 Modelos mixtos de unificación y dos niveles

En los últimos años se han propuesto sistemas que combinan una morfología en dos niveles para el tratamiento de los problemas morfografémicos con una gramática basada en rasgos para el tratamiento de la información morfosintáctica. Esto es la conclusión a la que han llegado algunos investigadores después de haber contrastado las experiencias con los dos modelos.

5.3.4.1 El reconocedor morfológico de Bear

John Bear fue el primero en proponer un procesador morfológico que "when parsing a word, uses two sets of rules: rules describing the syntax of words, and rules describing facts about orthography" (Bear 1986:272). Su trabajo parte de la base de que en el programa se pueden separar el reconocimiento de los morfemas (la lematización) de la recuperación de la información morfosintáctica. Para ello, hay que utilizar reglas ortográficas y reglas sintácticas diferentes. Las reglas ortográficas son muy parecidas a las de Koskenniemi, aunque con un formalismo particular. Sin embargo, Bear reconoce que el sistema de reglas de que disponía en ese momento (1986) sobregenera para el inglés:

> Although they produce all the right answers and allow multiple forms for words like `[travel+er] <-> ([traveller] or [traveler])`, which is certainly a positive result, they also allow multiple forms for words which do not allow them. For instance they generate both `[referred]` and `[refered]`. As mention earlier, this problem will be tolerated for the time being. (Bear 1986:274).

Las reglas sintácticas se implementaron en un intérprete de reglas del formalismo PATR pero con la posibilidad de admitir la disyunción en el mecanismo de unificación. El tratamiento de la información es similar al de otros formalismos basados en rasgos.

5.3.4.2 El modelo computacional de descripción léxica de Ritchie,Black, Pulman and Russell

Como ejemplo de diccionario por ordenador que integra un procesador morfológico veremos el modelo presentado por Ritchie, Pulman, Black y Russell (1987). Estos autores proponen un marco general (en el que se incluyen varios formalismos para dar cuenta de procesos ortográficos, morfológicos y de redundancia léxica) que permita establecer generalizaciones lingüísticas que puedan ser fácilmente interpretadas por ordenador. Básicamente, el sistema se compone de:

1. Un **lexicón**, cuyas entradas léxicas son morfemas, cada una de las cuales contiene varios campos donde se describen sus propiedades fonológicas, sintácticas y semánticas, así como su forma gráfica a modo de clave de cita.

2. Varios conjuntos de **reglas**:

   - ***reglas léxicas***, que expresan relaciones entre entradas o entre campos léxicos. Tienen una interpretación procedural que "expande" un conjunto pequeño de

entradas básicas en otro mayor con más categorías (algo así como desarrollar la familia léxica a partir de la palabra madre).

- ***reglas morfológicas***, que expresan las relaciones entre caracteres y morfemas dentro de una palabra. Tienen también una interpretación procedural que permite que una cadena de caracteres sea descompuesta en morfemas para proporcionar el análisis estructural de la palabra.

Nos concentraremos aquí exclusivamente en las reglas morfológicas, aunque queremos destacar la utilidad del conjunto de reglas léxicas que permiten extraer y ampliar información de una forma generalizada mediante un preprocesamiento (o compilación) que añade información adicional a las entradas. Las reglas morfológicas son de dos tipos:

- **Reglas ortográficas** (*Spelling Rules*): establecen las relaciones entre *formas superficiales* y las *formas léxicas* (las formas canónicas de las entradas léxicas). Están basadas en el formalismo de Koskenniemi ("Two-Level Morphology") y se pueden usar, mediante una implementación procedural, para segmentar cadenas de caracteres (la palabra gráfica) en morfemas, teniendo en cuenta las variaciones ortográficas.

- **Reglas de formación de palabras** (*Word Grammar Rules*): estas reglas describen las posibles estructuras internas de las palabras, usando un formalismo basado en rasgos al estilo de Gazdar *et al* (1985). Estas reglas permiten analizar una secuencia de morfemas y presentarla en forma de árbol estructural.

Por tanto, el procesamiento morfológico en este sistema se realiza en dos etapas. Una primera fase *morfográfica* (los autores utilizan frecuentemente el concepto "morphographemic rules" como sinónimo de "spelling rules") que se encarga de contrastar la forma superficial de la palabra con los morfemas del diccionario. Como el formalismo es básicamente el mismo que el de Koskenniemi, nos remitimos a la sección correspondiente.

La segunda fase es propiamente el análisis morfológico de la palabra. Para los autores esta parte constituye una "gramática de la palabra" (*Word Grammar*), donde se trata tanto la morfología flexiva como la derivativa una vez abstraídos los detalles gráficos particulares. En concreto, las reglas describen:

- las secuencias posibles y permitidas de morfemas,
- las concatenaciones válidas, y
- la categoría sintáctica resultante de la palabra formada por varios morfemas.

Esta "gramática de la palabra" está basada en el concepto de rasgos y valores, de tal forma que cualquier constituyente (morfema o palabra) se representa como un conjunto de rasgos y valores. Los autores siguen en gran medida el modelo GPSG de rasgos sintácticos, aunque el sistema permite utilizar cualquier otro modelo basado en rasgos. Como condición imprescindible, todos los rasgos deben ser declarados explícitamente. Los rasgos, dependiendo de los valores que puedan recibir, son de dos tipos:

- rasgos de valor "atómico" (*atomic-valued features*). Por ejemplo, el rasgo N puede tener los valores "+" y "-."
- rasgos cuyo valor es una categoría (*category-valued features*).

Todas las reglas tienen una forma típica, propia de las gramáticas de unificación de rasgos, a saber:

```
madre --> hijo1, hijo2, ..., hijoN
```

donde *madre, hijo1, hijo2,* etc. son categorías compuestas de rasgos. Las reglas pueden tener uno o más hijos y contener variables, que se instancian durante el *parsing* y permiten "copiar" valores de rasgos de los hijos a la madre y viceversa. Las variables se utilizan sobre todo para transmitir rasgos (es decir, sus valores) entre los componentes del árbol de análisis. Pero además cuentan con otro mecanismo de "passing features," son las llamadas **convenciones de trasmisión de rasgos** (*feature-passing conventions*) "which can be thought as rules for passing information UP the analysis tree (from terminal morphemes to the final word), or for passing information DOWN the analysis tree (from word to constituent morphemes)." Estas convenciones están tomadas de la GPSG y adaptadas a las generalizaciones morfológicas propuestas por Selkirk (1982). Son tres y no entraremos en detalles, que se pueden encontrar en las fuentes citadas.

Por último, señalar que el analizador morfológico busca todas las estructuras posibles de la palabra dada, de acuerdo con las reglas de la gramática, y devuelve dichas estructuras de constituyentes con la categoría resultante del análisis. En cuanto a las reglas morfológicas, sólo hay cuatro reglas para el inglés, que se encargan de analizar respectivamente la prefijación, la sufijación nominal (incluyendo derivación y flexión), la sufijación verbal sobre base verbal y la sufijación verbal sobre base nominal o adjetiva (es decir, las formas verbales derivadas de raíces adjetivas o nominales). Esta economía descriptiva y su generalidad se basa en la concepción lexicalista de la gramática (como vimos, una característica de los enfoques basados en la unificación de rasgos) que concentra la especificación de información no en las reglas gramaticales sino en las convenciones de filtrado y transmisión de rasgos:

> [...] these highly economical grammar rules are made possible by the assumption that the various feature-passing conventions (and feature defaults) will ensure that features are correct. (Ritchie *et al* 1987:300)

El funcionamiento general del sistema está dividido en tres fases:

1. Aplicación de las reglas morfográficas para segmentar la palabra.
2. Expansión de los rasgos de cada entrada de la segmentación mediante las reglas léxicas.
3. Aplicación de la gramática de la palabra para producir el análisis morfológico y léxico de la palabra dada.

La primera etapa del análisis ya ha sido explicada al hablar de las reglas ortográficas (*Spelling rules*) y cuya misión es segmentar la palabra, proporcionando tantos entradas como particiones sean posibles; en inglés, por ejemplo, una *-s* en final de palabra puede ser un morfema de plural o de tercera persona del singular. Como el procesamiento de la información morfológica no se hace propiamente hasta la fase de la gramática de la palabra, en este estadio del análisis sólo se obtienen las distintas combinaciones posibles de morfemas para la palabra dada, y el análisis posterior eliminará las incorrectas. Nótese que es necesario proporcionar todas las segmentaciones posibles de una palabra para dar cuenta de las homografías: *reports* es tanto "él informa" como "informes"; en este caso la segmentación tiene que dar, al

menos[47] , cuatro entradas de diccionario (verbo, nombre, sufijo plural, sufijo de tercera persona singular).

La segunda fase del análisis es consecuencia de las características propias de este sistema. Las entradas básicas del diccionario tienen sólo la información imprescindible (la categoría, subcategorización, tipo de sufijo, etc.) y las reglas léxicas se encargan de añadir nuevos rasgos (en su terminología, "completar las entradas hechas por el lingüista"). Por ejemplo, añadir el rasgo LAT a aquellas raíces de origen latino, especificar el nivel (BAR 0) para las raíces y (BAR -1) para los afijos, etc. Con esta "precompilación" del diccionario obtenemos entradas más detalladas de los morfemas segmentados (pero sólo de esos morfemas y no de todas las entradas del diccionario[48]) y pasamos a la tercera fase.

Con la aplicación de la gramática de la palabra (*Word Grammar*) conseguimos la estructura interna de la palabra y su información descompuesta en rasgos. Este proceso se divide siempre en dos partes, que se aplican en este orden:

1. aplicación de las reglas morfológicas, y
2. aplicación de las convenciones de transmisión de rasgos.

Por ejemplo la regla para sufijos (*SUFFIXING rule*) tiene la siguiente forma (tiene el formalismo típico de los sistemas basados en LISP):

```
(SUFFIXING
   ((BAR 0) (N +)) ->
            ((BAR 0)), ((N +) (FIX SUF)) )
```

Que se puede leer así: "un nombre o adjetivo ((BAR 0) (N +)) puede construirse de una raíz nominal o adjetiva (BAR 0) seguida de un sufijo nominal ((N +) (FIX SUF))." Esta regla sirve tanto para la derivación como para la flexión nominal del inglés (téngase en cuenta que la flexión del inglés es muy sencilla, por ejemplo, no incluyen el rasgo de género para la concordancia interna dentro del SN). Una vez aplicada la regla sobre una secuencia de raíz y sufijo, el nodo madre resultante tiene una información muy pobre (nivel 0, N +) que se amplia al aplicarse las convenciones de transmisión de rasgos.

Queremos insistir en el valor práctico del sistema de Ritchie *et al.*, pues combina por una parte las ventajas del modelo de estados finitos con las de las gramáticas independientes del contexto: el acceso al diccionario (lematización) parece ser más efectivo si utilizamos el tipo de gramáticas más sencillo (las de estados finitos), mientras que para la recuperación de la información es muy adecuado el uso de rasgos

---

[47] Además podría segmentar también como *re* (prefijo) + *port* (palabra base) + *s* (sufijo flexivo). Finalmente, las reglas de la gramática descartarían esta posibilidad pues el prefijo *re* no se puede aplicar ni a *port* ni a *port* + *s*.

[48] Uno se puede preguntar por qué no tener la precompilación de todo el diccionario. Simplemente por cuestión de espacio, pero lo cierto es que eso ahorraría tiempo de procesamiento. Los autores no explican porqué escogieron esta opción.

(Gazdar y Mellish 1989[49]) y su implementación en un sistema de reglas independientes del contexto. Es, sin duda, la propuesta más generalizadora de todas las que conocemos, incluso en su aplicación pues pretende ser tanto un sistema aislado (diccionario) como un componente léxico integrado en un sistema general de procesamiento de lenguas naturales. Desafortunadamente no conocemos su funcionamiento real, pero es de esperar que funcione eficientemente.

Existen otros sistemas mixtos que intentan combinar la morfología en dos niveles con gramáticas basadas en rasgos. Concretamente Trost(1990) describe su modelo para el alemán que divide el tratamiento de la morfología en dos: para los fenómenos morfofonológicos y los procesos no concatenativos utiliza un modelo de dos niveles; para los procesos concatenativos (es decir, la afijación) utiliza una gramática basada en la unificación de rasgos. La novedad de su propuesta es tratar con el modelo de Koskenniemi la morfología no concatenativa (concretamente el fenómeno de modificación interna conocido como "Umlaut"). Para ello necesita restringir la aplicabilidad de las reglas de dos niveles, dotándolas de filtros en la forma de estructuras de rasgos.

Una crítica frecuente al modelo de Koskenniemi es que, por estar basado en consideraciones fonológicas y ortográficas (cuyas reglas son por definición aplicables a todas las palabras de una lengua)[50] no es fácil especificar las modificaciones vocálicas o consonánticas provocadas por morfemas flexivos particulares. En el caso del Umlaut alemán, las transformaciones *a* → *ä*, *au* → *äu*, *o* → *ö* y *u* → *ü* se producen bajo determinadas circunstancias flexivas que difieren según las categorías:

- en los nombres pueden ser:
  - marca de plural en sí misma: *Mutter, Mütter*, o
  - ir en combinación con una terminación de plural explícita: *Mann, Männer*
- en los adjetivos es marca de grado comparativo, en combinación con un final: *groß, größer, am größten*
- en los verbos que siguen la conjugación fuerte marca todo el subjuntivo II y las segunda y terceras personas del presente de indicativo,

además en la derivación también aparece en combinación con algunos sufijos derivativos: *klagen, kläglich*. En contraste con la flexión, cuando aparece en la derivación no aporta ninguna información a la forma derivada. Según Trost, debido a que el Umlaut implica asignación de información morfosintáctica (plural, grado comparativo, determinados tiempos y personas verbales) no es apropiado, desde un punto de vista lingüístico, que un componente morfofonológico trate información que no le compete. Esta última crítica al modelo de dos niveles ya se había formulado antes:

---

[49] Porque facilita el acceso composicional al tratamiento del significado (Gazdar y Mellish 1989:101).

[50] Trost(1990:3) dice:

> [...] the original view of Koskenniemi [is] that morphonological rules are to be applied over the whole lexicon regardless of morphosyntactic considerations.

[..] the problems of treating morpho-syntax in the lexicon, which in reality is what happens in Koskenniemi's original model where the lexical entries for root morphemes are marked for "continuation classes" (references to sub-lexicons which determine the legal combinations of morphemes). (Lau y Perschke 1987:20)

Los filtros que propone Trost(1990) son estructuras de rasgos que permiten (o impiden, según los casos) que una determinada regla se aplique si el filtro unifica con la estructura de rasgos del morfema al que se aplica la sustitución vocálica. Es decir, en el caso del Umlaut, la regla se aplicará cuando una raíz que esté marcada para llevar el Umlaut se concatene con el afijo que lo permita.

5.3.5 Comparación de los tres modelos

En los apartados anteriores se han expuesto en líneas generales las características de los sistemas relacionándolos con el enfoque teórico más o menos implícitamente supuesto. Al mismo tiempo que los presentábamos se fueron haciendo comparaciones pertinentes en cada caso. Queremos ahora resumir el tratamiento concreto de cada modelo a tres cuestiones: la *sobregeneración*, la *adecuación formal en morfología* y la *eficiencia computacional.*

5.3.5.1 La sobregeneración

El hecho de que un sistema particular sobregenere, es decir, permita más formas de las gramaticales, no es achacable al modelo subyacente: es simplemente una cuestión de establecer una formalización adecuada y rigurosa. En lo que se diferencia cada modelo es en las técnicas de que dispone para formalizar declarativamente sus descripciones.

- *Palabras y Paradigma* se rige por el concepto de *analogía* que le indica cómo establecer paradigmas. Su preocupación descriptiva consiste en clasificar paradigmáticamente las palabras de una lengua. En consecuencia, si se encuentran formas especiales que no se pueden tratar con ninguno de los paradigmas existentes hay que crear un nuevo paradigma, aunque sólo sea para un caso. Esto implica que suele haber muchas reglas redundantes. La sobregeneración en este modelo se produce si la forma base no está bien adscrita a un paradigma determinado.

- *Elementos y Proceso* basa su descripción en reglas fonológicas y ortográficas. Como su aplicación es general sobre todo el diccionario, hay que controlar que se apliquen sólo en los casos gramaticales. Ya hemos visto que una solución es incorporar filtros de rasgos, pero esto supone un cambio bastante radical en la filosofía del modelo de dos niveles. La solución habitual para los fenómenos no concatenativos es utilizar la supleción (añadir una forma especial aparte de la forma básica) (Kay 1987).

- *Elementos y Colocación.* Los sistemas que basan su tratamiento en la concatenación de morfemas utilizan por lo general reglas y estructuras con rasgos. La misión fundamental es definir todo el inventario de alomorfos y afinar la descripción utilizando el menor número posible de rasgos para no complicar el proceso de comprobación de valores entre los segmentos que se concatenan. Es necesario para ello que el inventario de rasgos sea lo suficientemente expresivo para que permita dar cuenta de todas las asociaciones posibles entre alomorfos y al

mismo tiempo controlar la sobregeneración con restricciones de aparición de rasgos.

Como conclusión de todo ello, podemos decir que lo fundamental es establecer una clasificación rigurosa de los fenómenos de alomorfía. De hecho, cualquier descripción que reúna este requisito puede ser utilizada por cualquier modelo, adaptándola a sus técnicas particulares.

5.3.5.2 *La eficiencia computacional*

Hay que hacer una observación sobre las aparentes "incompatibilidades" que surgen entre la descripción lingüística y su tratamiento computacional. Normalmente, en los formalismos lingüísticos se insiste en el carácter declarativo para definir las regularidades de una lengua y en reducir el poder expresivo del formalismo, mientras que en las implementaciones computacionales se opta por técnicas "poderosas" que permitan una gran flexibilidad y generalidad. Debido a esto, se suelen adoptar algoritmos procedurales, con los que se consigue, en líneas generales, mayor eficiencia.

Es interesante, sin embargo, que tengamos presente que tanto los formalismos (o mejor dicho, formalizaciones)[51] como sus implementaciones pueden ser o bien declarativos o bien procedurales. En ese sentido existen las siguientes posibilidades:

1. Formalización procedural (o no explícitamente declarativa) e implementación procedural
2. Formalización procedural e implementación declarativa
3. Formalización declarativa e implementación procedural
4. Formalización declarativa e implementación declarativa

La posibilidad (2) es un contrasentido: toda implementación declarativa implica una formalización declarativa. Obviamente no existen casos de este tipo. Lo más habitual es encontrar sistemas de tipo (1), en el que se combina una formalización no explícitamente declarativa (es decir, que no se restringen a las formas gramaticales y su aplicación tiene un orden definido, normalmente ascendente por tratarse de sistemas de análisis y no de generación) con una implementación procedural. Koskenniemi provocó un gran revuelo al presentar por primera vez un sistema del tipo (3): su formalismo permite dar cuenta de los fenómenos morfológicos independientemente del orden de ejecución que se requiera, y además con la implementación procedural más sencilla (un autómata de estados finitos). Los sistemas de gramáticas basadas en rasgos y en unificación en contraste con el modelo de Koskenniemi presentan un tratamiento declarativo de los fenómenos morfosintácticos, que los hacen más atractivos desde el punto de vista lingüístico. Los sistemas mixtos dos niveles-gramática de rasgos son una evolución hacia el tipo (4) aprovechando las ventajas de la implementación procedural para el tratamiento de la lematización. Nuestra tesis pretende ser un ejemplo del último tipo. Alguien se puede preguntar qué sentido tiene utilizar una implementación declarativa cuando se puede mejorar la eficiencia con procedimientos. Simplemente por motivos de claridad y sencillez: la formalización (es decir, la gramática y el diccionario) es más fácil de trasladar a un lenguaje de programación

[51] Los formalismos no son declarativos en sí mismos, puesto que son meras herramientas lingüísticas. Son las formalizaciones concretas las que deben mostrar esa cualidad.

declarativo como Prolog que implementarla en un lenguaje procedural. Por otra parte, la implementación declarativa es explicativa por sí misma y permite a cualquier persona con conocimientos de lingüística comprender el programa fuente, sin necesidad de experiencia o conocimientos específicos de programación. Como vimos en la sección 3.1.2 existe la tendencia en teoría de la computación a utilizar lenguajes de programación orientados a disciplinas o problemas concretos, en lugar de adaptar el planteamiento de problemas que ya tienen una formalización dentro de su propia disciplina a una reformulación en un lenguaje de computación de propósito general.

5.3.5.3 La adecuación formal en morfología

Calder(1989) expresa certeramente un punto de vista bastante extendido en la actualidad acerca del tratamiento computacional de la morfología

> A common assumption in linguistics is that the phonological, morphological and orthographic statements are most appropiately phrased in a fundamentally procedural way [...]. Morphological analysis under the rubric of finite-state morphology (Koskenniemi 1983) has arguably tended to support the view that morphological alternation is best described by stating procedures for the destructive alteration of orthographic units. At the very least, it appears to have led to the view that morphological descriptions should be restricted to those with an immediate interpretation in terms of the operations of finite-state transducers.

Por el contrario su formalismo intenta representar declarativamente las relaciones morfológicas dentro del enfoque Palabras y Paradigmas y una gramática independiente del contexto. Compartimos igualmente la afirmación de Calder acerca de la inadecuación de las gramáticas de estados finitos:

> As Gazdar(1985) notes, it is not certain that morphological phenomena in natural language are best characterized by finite-state devices [...] the formal power of our framework might be interpreted as a virtue rather than a vice and future work should also look at introducing (at least) context-free devices into our computational interpretation of morphology. Unsurprisingly, this is an area for further research. (Calder 1989:64)

Siguiendo en la misma línea, creemos que las gramáticas de estados finitos sólo son apropiadas para dar cuenta de los procesos regulares (por algo son conocidas también como gramáticas regulares) como los cambios ortográficos y los fonológicos. Para todos los cambios en los que intervengan procesos morfológicos no concatenativos su efectividad ha resultado nula porque se necesita obligatoriamente una entrada extra (Kay 1987). Renunciar a la idea de mantener una forma única en el lexicón para cada entrada léxica nos parece una claudicación teórica muy importante: el modelo teórico de Elementos y Proceso lo puede explicar perfectamente con reglas de alomorfía, que evidentemente tienen el poder expresivo de las reglas de contexto. La falta de expresividad de las gramáticas de estados finitos obliga a recurrir a la suplencia, invalidando precisamente su característica más atractiva: la de tener una forma base única en el diccionario, de la que se derivan todas las formas flexionadas. Se demuestra una vez más, que los fenómenos de las lenguas naturales no se pueden tratar con el poder expresivo de las gramáticas de estados finitos, aunque sean el mecanismo procedural más eficiente.

Morfología del español basada en la unificación

Nuestro modelo se basa en los siguientes presupuestos teóricos:

1. es una gramática basada en la unificación de rasgos, como elementos portadores de información,
2. es una gramática independiente del contexto y basada en el orden superficial real de los elementos gráficos o cadena de caracteres,
3. se concentra exclusivamente en el tratamiento morfosintáctico y no realiza ningún tipo de interpretación semántica o léxica (es decir, la composición y la derivación no reciben un tratamiento específico). En este sentido, la Flexión no se considera dentro del módulo morfológico-léxico de la Gramática, y se integra por tanto en la Sintaxis,
4. utiliza el modelo morfológico de *Elementos y Colocación*, siguiendo el principio de composicionalidad para decidir la segmentación de la palabra.
5. es una gramática declarativa en el sentido de que define las asociaciones permitidas de cadenas superficiales e información gramatical. Debido a esta característica es una gramática completamente reversible: sirve tanto para analizar como generar morfológicamente cualquier palabra del español,
6. tiene un enfoque marcadamente lexicalista, es decir, la complejidad recae sobre el diccionario y no sobre las reglas de la gramática.

De todos estos puntos, con sus ventajas e inconvenientes, ya se ha hablado en las secciones anteriores. Ha llegado el momento de mostrarlos explícitamente. Nuestra exposición se dividirá en tres partes: en la primera explicaremos detalladamente el componente léxico del procesador (es decir, cómo se codifican las entradas del diccionario). La segunda parte se dedica a la descripción de las reglas gramaticales, que combinan los elementos léxicos para producir un análisis morfológico o generar una forma. En el apéndice se documentará el programa de demostración así como un pequeño manual de operación.

6.0 El diccionario

Cualquier sistema de procesamiento de lenguas naturales está compuesto básicamente por un **diccionario** (lugar en el que se localizan los elementos terminales o cadenas superficiales, con toda su información asociada, generalmente en forma de rasgos) y por una **gramática** (que contiene las reglas que combinan los elementos terminales y su información). Para escribir tanto un diccionario como una gramática que puedan ser tratados por el ordenador es necesaria una descripción rigurosa y formalizada de los fenómenos lingüísticos. En nuestro caso, estableceremos primero clasificaciones, que nos servirán posteriormente para codificar la descripción en forma de rasgos. Una vez hecho esto, la transcripción a un lenguaje de programación es bastante directa.

Nuestra clasificación es una aproximación nueva al modelo de *Elementos y Colocación* en el sentido de que es una clasificación basada en la distribución de los morfemas (o mejor dicho alomorfos). En consecuencia, se compone de:

- inventario de alomorfos y morfemas, y
- especificación de las secuencias o contextos de aparición de los alomorfos.

Veremos que tanto el inventario como la especificación contextual se codifica por medio de rasgos, de manera que los lexemas se representan mediante el rasgo **unidad léxica**, los alomorfos por el rasgo **cadena** y los contextos de aparición a través de una combinación de rasgos especiales.

Las novedades con respecto a las clasificaciones estructuralistas son:

1. no se acepta la presencia de elementos vacíos o morfemas $\emptyset$, y
2. la "segmentabilidad" de los morfemas está condicionada por el tratamiento composicional de la información que trasmiten, es decir, cada morfema aporta a la información total de la palabra una subparte claramente diferenciada del resto de los formantes.

De esta manera se superan ciertas inadecuaciones descriptivas de las clasificaciones estructuralistas (fuertemente criticadas por los partidarios de *Elementos y Proceso*). En concreto, ambas están relacionadas con el establecimiento de las unidades discretas que forman una palabra: la morfología estructuralista hacía particiones que no se correspondían con la cadena superficial en aquellos casos donde no se distingue claramente a qué segmento fonológico o gráfico corresponde un determinado morfema. En nuestra propuesta, los morfemas (como elementos terminales del diccionario) forman una unidad discreta tanto en forma superficial como en su información asociada. Por ejemplo, la información gramatical del verbo (persona, número, tiempo, modo...) en español no puede ser aislada nítidamente en varios segmentos superficiales, por lo tanto, dicha información se agrupa en un único formante, que por su parte sí

puede aislarse del formante con información léxica (la raíz verbal). Así se consigue que la información de las dos subpartes de la palabra se unifique composicionalmente.

Nótese que esta aproximación es idéntica a la propuesta por los antiguos gramáticos, que sólo distinguían entre raíces y terminaciones. Hemos intentado, por tanto, aprovechar la solución que da para estos casos el modelo de *Palabras y Paradigmas* pero sin asumirla en aquellos lugares donde la distinción entre los morfemas es nítida (por ejemplo, en la flexión nominal).[52] Esta reelaboración dentro del modelo de *Elementos y Colocación* se fundamenta a nuestro entender desde los presupuestos de los gramáticas de unificación, donde sólo se reconocen como cadenas las que se manifiestan superficialmente y donde el tratamiento composicional de la información tiene un peso decisivo[53] . En otras palabras, hemos actualizado en cierta medida la relación estructuralista entre *forma y significado* a la versión "unificacionista" de *cadena superficial e información.*

Por otra parte, el procesamiento morfológico requiere una descripción basada en la *palabra gráfica* o cadena de caracteres. Esta característica es fundamental para entender nuestra clasificación: es **formal** por cuanto que se ajusta y se basa en la *forma gráfica* de las palabras[54] . En consecuencia, para establecer las variantes alomórficas seguiremos el *Principio Gráfico* (definido en la sección 5.1.1, "La palabra como cadena de caracteres" en la página 55). Nuestra clasificación contrasta, por tanto, con las clasificaciones lingüísticas habituales, basadas en principios fonológico-históricos. A nuestro entender, las diferencias se resumen en tres:

- En primer lugar, la clasificación formal necesita aumentar su inventario de alomorfos con cadenas que desde la óptica fonológica se encuadran perfectamente en los paradigmas existentes, y que son fruto de los cambios ortográficos que ocurren en las palabras del español escrito. Por ejemplo, los nombres que cambian de raíz en plural (*león, leon - es, lápiz, lápic - es*) o casos como el verbo *cocer*, con cuatro raíces en lugar de dos (*coc - er, coz - amos, cuec - e, cuez - o*)

- Por otra parte, las clasificaciones fonológico-históricas se basan, como es obvio, en procesos fonológicos (cambios vocálicos, consonánticos, etc.) mientras que las clasificaciones formales se basan en la distribución de las formas dentro de la conjugación. Por ejemplo, los cambios vocálicos *e --> ie, o --> ue* (*acertar, sonar*) están agrupados en tipos distintos en la clasificación de la R.A.E (tipos B y C, respectivamente); mientras que en nuestra clasificación aparecen dentro del mismo

---

52 Por otra parte, el modelo PP no reconocía que los morfemas fueran unidades discretas con significado, al igual que el modelo *Elementos y Proceso* tampoco lo hace, lo que contrasta con nuestro planteamiento.

53 Recuérdese que el modelo *Elementos y Proceso* es incompatible con un formalismo de unificación porque el primero implica una "transformación" desde la estructura profunda a la superficial (por ejemplo, la modificación interna o diptongación de una raíz es un proceso originado por la afijación de un morfema flexivo que implica un cambio en la forma superficial). Por su parte, un requisito de las gramáticas de unificación es que estén basadas en "el orden superficial real de la cadena de elementos" (Shieber 1986;1989:6). El modelo *Elementos y Colocación* permite la *concatenación* de los formantes morfológicos superficiales.

54 De aquí en adelante, cuando utilicemos el término *formal* nos estaremos refiriendo al sentido expuesto y no a otras posibles interpretaciones del término ("gramática formal" = "gramática formalizada").

modelo,[55] pues comparten la misma distribución formal (diptongación en las formas fuertes de la flexión).

- Todos los autores que han trabajado en la clasificación de las irregularidades verbales coinciden en señalar lo difícil que es conseguir una descripción exhaustiva de los verbos españoles. Bello se conforma con exponer su tipología "sin entrar en pormenores embarazosos para los principiantes: conjugando éstos cierto número de verbos de cada clase, según el respectivo modelo, no habrán menester más para familiarizarse con la conjugación de todos ellos" (Bello 1984:192). Otros autores, como Alcina y Blecua, son más explícitos: "cada verbo tiene una o más alternancias con lo que dificulta una clasificación exhaustiva" (Alcina y Blecua 1980:769). En otras ocasiones, la dificultad consiste en establecer cuál es la raíz y cuáles son los morfemas flexivos. Las clasificaciones fonológicas suelen recurrir entonces a la evolución histórica[56] . Esta explicación, aunque perfectamente válida desde el punto de vista teórico, no ayuda mucho a una disciplina aplicada como la Lingüística Computacional, donde hay que proporcionar descripciones que sean posibles de codificar en el ordenador. Para ello se requiere, por una parte, criterios explícitos que permitan tomar una decisión y, por otro lado, que se puedan aplicar en todos los casos. Creemos que los criterios formales cumplen ambos requisitos y, por tanto, son útiles para la "modelización" y el posterior tratamiento informático.

Antes de continuar, queremos expresar nuestro reconocimiento a la influencia que ha recibido nuestra clasificación de algunos trabajos anteriores. Por una parte, los capítulos dedicados a la Morfología del español en el *Esbozo* de la R.A.E., son consulta imprescindible y fundamento del resto de las clasificaciones fonológico-históricas, cuyo origen (especialmente en la parte dedicada al verbo) está en el trabajo de Menéndez Pidal (*Manual de gramática histórica española*). Por otra parte, la labor investigadora del grupo de lexicografía computacional del Centro de Investigación UAM-IBM. Ha sido un grupo pionero en nuestro país y sus publicaciones me han servido de punto de referencia a la hora de construir mi propia clasificación, fundamentalmente gracias a la exhaustividad alcanzada en el tratamiento del léxico del castellano.

Por último, queremos reivindicar el trabajo de Andrés Bello como la primera clasificación formal de los paradigmas irregulares del verbo castellano. Aunque el gramático americano señala que "para reconocer a un verbo regular o irregular no debe atenderse a las letras con que se escribe sino a los sonidos con que se pronuncia" creemos que su clasificación se anticipa en gran medida a las descripciones formales (y también a las fonológicas). Bello expone muy claramente que, a la hora de establecer una clasificación de las irregularidades, hay que basarse en la *forma* o *estructura material*, ya sea fónica o gráfica, y dejar de lado el *significado* o su origen histórico:

> [...] relativamente a la conjugación no miramos como compuestos sino a los verbos en cuyo infinitivo aparece el del simple sin la menor alteración [...].

---

[55] Puede extrañar que utilicemos los conceptos de *modelo, paradigma, tipo* dentro del marco de *Elementos y Colocación*. Hay que tener en cuenta, sin embargo, que tal concepto no es privativo del modelo *PP* ya que para establecer los inventarios de morfemas y alomorfos se necesita recurrir a la elaboración de tablas donde se contrasten las formas. De hecho, presentaremos la clasificación con los modelos que nos han servido para establecer los variantes alomórficas.

[56] Dice el *Esbozo*, en la nota 4 de la pág. 256:

> Acaso lo mejor... es explicar históricamente la formación del futuro y del condicional, limitándonos a describir el tema, sin entrar en más pormenores.

Prescindiremos pues del significado, y sólo atenderemos a la estructura material. Así, en lo que atañe al mecanismo de la conjugación [...] *convertir* no es compuesto de *verter*, y por el contrario, *impedir* lo es de *pedir*. (Bello 1984:173)

Efectivamente, su clasificación, a diferencia de las fonológico-históricas[57] , consiste en trece "clases de verbos irregulares," que se establecen en función de grupos de formas afines u *órdenes*. Precisamente este concepto de grupos de formas afines dentro del paradigma (lo que nosotros llamamos "distribución de las formas") es, a nuestro entender, la clave de las clasificaciones formales:

Cuando una forma experimenta una alteración radical, casi siempre sucede que hay otras formas que la experimentan del mismo modo [...]. Alterada la raíz en una de sus formas pertenecientes a cualquiera de estos órdenes los verbos que son irregulares en él experimentan una alteración igual en las otras formas del mismo, y tienen por consiguiente una raíz peculiar e irregular en todas ellas.

La anticipación de Andrés Bello en este punto, como en tantos otros, es una prueba más de la calidad imperecedera de su gramática.

6.1 La Flexión Verbal

Las mayoría de las clasificaciones morfológicas suelen distinguir cuatro formantes en el verbo español:

Raíz	Vocal temática	Tiempo-Aspecto	Persona-Número
am	*- a -*	*- ba -*	*- s -*

Pero esta descripción no nos parece la más apropiada para el tratamiento informático, tanto por motivos teóricos como por razones prácticas.

En primer lugar, no se ha llegado a un acuerdo sobre cómo segmentar el verbo en sus respectivos formantes morfológicos. Algunos autores (Marcos Marín 1980), por ejemplo, agrupan la raíz con la vocal temática (formando el *lexema verbal*); para otros, la vocal temática es una *característica* (según la definición de la R.A.E., características son los morfemas flexivos de tiempo y modo) que agrupada con la raíz forman un *tema* (*Esbozo* 1989:249). Pero donde se dan mayores discrepancias es a la hora de establecer la segmentación de determinadas formas verbales: debido a que muchas formas no presentan los cuatro formantes hay que recurrir a los morfemas ∅, y cada cuál los asigna a su gusto (aprovechando la "intangibilidad" del morfema vacío). Por ejemplo, la primera persona del singular del Presente de Indicativo *cant - o* sólo manifiesta un único formante flexivo. Como el morfema de persona y número de esta forma verbal siempre es ∅ (en estas clasificaciones), el otro morfema vacío lo asignan algunos autores a la vocal temática (p. ej. Alcina y Blecua 1980) y otros al morfema de

[57] Recordemos que estas clasificaciones, como por ejemplo la de la Real Academia, las irregularidades se agrupan por cambios fonológicos y no por paradigmas.

tiempo-aspecto (Stockwell, Bowen y Martin 1965, Marcos Marín 1980). En cualquier caso, el paradigma queda un poco desequilibrado frente a las otras formas del Presente:

Raíz	Vocal temática	Tiempo-Aspecto	Persona-Número
cant	∅	- o -	∅
cant	- o -	∅	∅
cant	- a -	∅	- s
cant	- a -	∅	∅

Esta clasificación se complica mucho más si empezamos a tener en cuenta las irregularidades o variaciones de los morfemas flexivos. Como reconocen Alcina y Blecua todavía no se dispone de una descripción definitiva basada en estos cuatro formantes:

> Han sido problemas fundamentales que todavía impiden llegar a una definición concluyente, la abundancia de formas que toman las realizaciones en el discurso... y la falta de precisión en el concepto de categorías verbales que afectan a la expresión de esta clase de palabras, y la imprecisión y problematismo en la segmentación de los morfemas que expresan. (Alcina y Blecua 1980:732-734).

Precisamente basándonos en la dificultad de segmentar nítidamente qué porción del segmento fonológico corresponde a qué morfema ya sea vocal temática, tiempo-aspecto o persona-número[58] , creemos que es preferible distinguir solamente entre **raíz verbal** y **morfema flexivo verbal** (donde se realizaría toda la información referente al tiempo, modo, aspecto, persona, número, y finitud o infinitud).

Otro argumento (a la vez teórico y práctico) en favor de esta clasificación es la cuestión de la pertinencia del morfema vacío en una descripción de este tipo. Ya comentamos que desde el punto de vista teórico es difícil justificar un constructo que no tiene manifestación concreta (capítulo 4.2, "Los formantes morfológicos" en la página 32). Si nos situamos dentro de un marco de gramáticas de unificación, es todavía menos justificable la presencia de un morfema ∅. ¿Qué información se puede transmitir desde un constructo vacío al nodo superior? Sería necesario un mecanismo especial de instanciación de rasgos, lo cual no es una solución muy elegante. Otra posibilidad sería crear entradas de diccionario con cadenas vacías, pero habría que escribir infinidad de ellas... Ritchie *et al.* (1987) comentan al respecto de introducir morfemas con la forma superficial vacía y rasgos sintácticos asignados a dichas cadenas vacías que "the complications this would introduce into the morphographemic segmentation and word-grammar parsing are regarding as wholly unacceptable computationally."

Otro problema que nos presentaría el morfema ∅ es cómo analizarlo por el ordenador. La palabra escrita no tiene blancos entre sus caracteres que permitan

---

[58] Nótese que a diferencia de los morfemas flexivos nominales, en los que se distinguen fonológicamente género y número.

decidir cuándo hay un morfema vacío (imaginémonos algo así como *cant_o_*; el signo _ equivale a un blanco). En resumen, en una teoría o formalismo basado en la superficie no se justifica el uso de elementos superficiales vacíos, al igual que LFG o GPSG rechazan las *huellas* de elementos que se mueven y las categorías vacías, en la sintaxis.[59]

Hay más argumentos prácticos (es decir, informáticos) para querer hacer la segmentación lo más sencillamente posible: si tenemos un regla de flexión verbal que deba combinar tres o cuatro elementos (raíz, vocal temática...) consumirá ésta más tiempo que si tiene que buscar la combinación adecuada sólo entre dos elementos (raíz y morfema flexivo). Hay que tener en cuenta además que ya de por sí es costosa la comprobación entre morfemas diferentes pero con la misma cadena, para invertir un tiempo extra en controlar el poder de combinación de estas unidades en la concatenación de varios formantes.

En resumen, al segmentar el verbo en dos únicos formantes conseguimos simplificar la descripción y disminuir el tiempo de procesamiento, frente a la opción de cuatro formantes verbales, al tiempo que damos un tratamiento composicional a la información gramatical de las terminaciones verbales. Es decir, las *desinencias* son junto con las *raíces verbales* las unidades discretas de la morfología verbal.

6.1.1 Clasificación formal del verbo

La forma habitual de clasificar los verbos del español es mediante paradigmas que recogen la *conjugación* (o flexión) de un verbo modelo, esto es, "la serie entera de las formas verbales con una raíz común, es decir, todas las formas de un verbo determinado" (*Esbozo* 1973; 1989:249). En nuestra clasificación, *raíz común* corresponde a **unidad léxica**, término que tomamos del formalismo utilizado por el proyecto EUROTRA (origen de esta tesis), y que se ajusta mejor, creemos, al concepto para designar a una entidad léxica del diccionario. Ya comentamos, por otra parte, que con este rasgo se identifica el *lexema* y con el rasgo **cadena** se identifica a sus alomorfos.

A la hora de elaborar una clasificación que sirva de modelo para procesar la información morfológica de los verbos, necesitamos una codificación que conecte cada raíz con su morfema flexivo correspondiente para cada forma de la conjugación. Esta codificación debe servir tanto para los verbos regulares como para aquellos que presentan alguna diferencia con los paradigmas regulares. Es bien sabido que existen dos tipos de variación alomórfica, según se produzca en la raíz o en el morfema flexivo (también es posible que aparezcan ambas a un mismo tiempo, como veremos). Es mucho más habitual que la alomorfía se produzca en la raíz, aunque no es tan excepcional que haya irregularidades en el morfema flexivo[60] . A pesar de esta evidente complejidad, es obligado recordar la famosa cita de Bello:

---

[59] No es imposible conseguir que un ordenador "inserte" blancos dentro de las palabras, pero evidentemente es un trabajo extra que resta eficiencia al programa.

[60] Como señala el *Esbozo* ambos tipos de alternencia tienen un origen histórico diferente. Las irregularidades en la terminación flexiva son heredadas del latín, mientras que las múltiples variaciones de la raíz son un "fenómeno lingüístico secundario, debido a las leyes fonéticas romances que han actuado sobre el sistema entero de la lengua" (nota 1, pág. 250). En cualquier caso, para nuestra descripción formal no afecta esta diferencia histórica.

> Yo dudo que alguna de las lenguas romances sea tan regular, por decirlo así, en las irregularidades de sus verbos, como el castellano. (Nota XII, pág. 191).

Si bien es cierto que la mayoría de las irregularidades verbales se explican por la evolución fonética, que es de por sí un artífice de la "regularización" de fenómenos (aunque con el ineludible período de transición de unas formas a otras), no es menos cierto que las convenciones ortográficas son absolutamente regulares (perdónese la obviedad). Esto significa que todas las irregularidades producto de cambios gráficos ("z" → "c," "c" → "qu," etc.) son predecibles.

Por otra parte, la descripción tiene que estar enfocada al tipo de formalismo que se va a utilizar en la Gramática de la Palabra (recordemos que se trata de una gramática basada en la unificación de rasgos). Es decir, las reglas fonológicas propuestas por el *Esbozo* no se ajustan al modelo formal de la unificación[61] . Como explicamos en la sección 2.0, "Gramáticas de unificación y gramáticas de rasgos" en la página 13, este enfoque se caracteriza por asociar cadenas superficiales con elementos informativos mediante un sistema de rasgos y valores, y por la combinación a través de reglas de varias cadenas superficiales (y sus elementos informativos), dando como resultado la concatenación de las cadenas y la unificación de la información asociada a ellas.

De esta forma, vamos a asociar cada cadena superficial con dos rasgos específicos que informarán de las posibilidades de combinación con otras cadenas superficiales. Estos rasgos son:

- **tipo_raíz** identifica la raíz verbal y la terminación correspondiente a cada forma, mediante un código numérico, y
- **tipo_des** distingue entre varios alomorfos del morfema flexivo (desinencia) gracias a un conjunto de valores distintos.

La combinación de ambos rasgos permite decidir cuál es la raíz y la terminación correctas para cada caso, como veremos detenidamente en la sección de la Gramática.

6.1.1.1 El rasgo tipo_raíz

Como avanzamos más arriba, necesitamos una codificación que nos permita identificar cualquier forma de la conjugación, de manera que podamos referirnos a ella sin ambigüedad, para marcar qué alomorfo de la raíz y qué terminación deben combinarse. Para ello hemos elaborado la siguiente **matriz** o **tabla de las formas verbales**. En la vertical se colocan los valores del rasgo **tiempo_modo**, y en la horizontal los del rasgo **persona_número**.

---

[61] Aunque, en cambio, se pueden adaptar directamente a un modelo basado en un autómata de estados finitos. Martin Kay reconoce que las reglas fonológicas le inspiraron para hacer su "morphological transducer."

Tabla 3. Matriz de la conjugación.							
	sg_1	**sg_2**	**sg_3**	**pl_1**	**pl_2**	**pl_3**	**no**
pres_ind	11	12	13	14	15	16	
impf_ind	21	22	23	24	25	26	
pret_ind	31	32	33	34	35	36	
futuro	41	42	43	44	45	46	
pres_subj	51	52	53	54	55	56	
imp_subj	61	62	63	64	65	66	
cond	71	72	73	74	75	76	
imper		82			85		
infin							00
ger							90
part							99

Los valores de estos dos rasgos son:

- **persona_número** =

  sg_1 (primera persona del singular)

  sg_2 (segunda persona del singular)

  sg_3 (tercera persona del singular)

  pl_1 (primera persona del plural)

  pl_2 (segunda persona del plural)

  pl_3 (tercera persona del plural)

  no (forma no personal)

- **tiempo_modo** =

  pres_ind (presente del indicativo)

  impf_ind (imperfecto del indicativo)

  pret_ind (pretérito perfecto simple del indicativo o pretérito indefinido)

  futuro (futuro del indicativo)

  pres_subj (presente del subjuntivo)

  imp_subj (imperfecto del subjuntivo)

  cond (condicional)

  imper (imperativo)

  infin (infinitivo)

  ger (gerundio)

part (participio)

No se ha incluido en la tabla el futuro del subjuntivo por ser un tiempo que "hoy solo se usa, aunque poco, en la lengua literaria y en algunas frases hechas conservadas en el habla coloquial" (*Esbozo* 1989: 482); pero sobre todo por no aumentar innecesariamente la matriz con más códigos, aunque perfectamente se le podría asignar una serie. Las formas compuestas no tienen código numérico porque se construyen con una regla a partir de las formas flexionadas del verbo *haber* y el participio del verbo principal. Más detalles sobre estos rasgos y sus posibles valores se encontrarán en la sección donde se explica el formalismo y las reglas de GRAMPAL.

Hemos escogido un código numérico para los valores del rasgo **tipo_raíz**, porque necesitamos distinguir entre 48 posibilidades y utilizar valores alfabéticos podría resultar muy engorroso (teniendo en cuenta, además que **tipo_raíz** es un rasgo *compuesto*, es decir, que consiste en una lista más o menos larga de valores). Imaginémonos, por ejemplo, que para la primera persona del singular del presente del indicativo establecemos un valor como *sg_1_pres_ind*...En cambio, hemos optado por una combinación numérica que no es difícil de memorizar[62] :

```
la decena (1X) para el presente de indicativo
la veintena (2X) para el imperfecto del indicativo
la treintena (3X) para el pretérito simple del indicativo.
...
el uno (X1) para la primera persona del singular
el dos (X2) para la segunda persona del singular
el tres (X3) para la tercera persona del singular
el cuatro (X4) para la primera persona del plural
...
```

Así, el 65 es la segunda persona del plural del imperfecto del subjuntivo, el 82 la segunda del singular del imperativo, etc. La finalidad de todo esto es que cuando queramos expresar la distribución de los dos alomorfos de la raíz de *acertar* podamos asociar la cadena *aciert* con los códigos correspondientes de la tabla para las formas fuertes de la conjugación; y *acert* con el resto:

```
cadena = aciert     tipo_raíz = 11,12,13,16,51,52,53,56,82

cadena = acert      tipo_raíz = 14,15,21,22,...,54,55,...,85,00,90,91
```

Hemos reservado el código **100** para las raíces únicas (la de los verbos regulares), con el fin de ahorrarnos escribir los 47 códigos de la conjugación completa:

```
cadena = am         tipo_raíz = 100
cadena = tem        tipo_raíz = 100
cadena = part       tipo_raíz = 100
```

---

[62] Otro argumento, en este caso exclusivamente práctico, es que los ordenadores trabajan más rápidamente con códigos numéricos que con códigos alfabéticos.

A su vez, cada desinencia lleva el código correspondiente a la forma verbal en que aparece, y de esa manera la concatenación con la raíz apropiada se realiza sin ningún tipo de ambigüedad, a pesar de que existan varias cadenas superficiales idénticas (para distintas formas). Cada desinencia lleva también la información gramatical expresada mediante rasgos:

```
cadena = a   tipo_raíz = 13   pers_núm = sg_3   tiemp_mod = pres_ind

cadena = a   tipo_raíz = 51   pers_núm = sg_1   tiemp_mod = pres_subj

cadena = a   tipo_raíz = 53   pers_núm = sg_3   tiemp_mod = pres_subj
```

6.1.1.2 El rasgo tipo_des

Las irregularidades de la terminación flexiva no son tan abundantes, pero sobre todo no se dan en todas las formas de la conjugación, como ocurre con la irregularidad en el radical. Esto nos permite hacer una codificación mucho más sencilla, recurriendo a valores alfabéticos:

- **tipo_des** =

  reg (desinencias regulares)

  pres (variante alomórfica para el presente de ind.)

  pret1 (variante alomorfica para el pret. de ind.)

  pret2 (variante alomorfica para el pret. de ind.)

  fut_cond (variante alomorfica para el futuro y el condicional)

  imp_subj (variante alomórfica para imperfecto del subj.)

  imper (variante alomórfica para el imperativo)

  infin (variante alomórfica para el infinitivo)

  ger (variante alomórfica para el gerundio)

  part1 (variante alomórfica para el participio)

  part2 (variante alomórfica para el participio)

  defect (código para las formas defectivas)

¿ Cómo se asigna este rasgo ? El valor de **tipo_des** está en función de la desinencia que se utilice en cada forma verbal. Existe una desinencia *regular* para cada forma de la conjugación, tomada de los paradigmas regulares de las tres conjugaciones. Estas desinencias llevan todas el valor **reg**, además de los otros rasgos mencionados.

Así por ejemplo, las tres desinencias mostradas más arriba se completarán de la siguiente manera[63] :

```
cadena = a  tipo_raíz = 13  pers_núm = sg_3  tiemp_mod = pres_ind  tipo_des = reg

cadena = a  tipo_raíz = 51  pers_núm = sg_1  tiemp_mod = pres_subj  tipo_des = reg

cadena = a  tipo_raíz = 53  pers_núm = sg_3  tiemp_mod = pres_subj  tipo_des = reg
```

Todas las desinencias que no coincidan con las regulares son las variantes alomórficas, y cada una lleva como valor un código alfanumérico que identifica el tiempo a que corresponde dicha entrada:

```
cadena = e  tipo_raíz = 31  pers_núm = sg_1  tiemp_mod = pret_ind  tipo_des = pret1

cadena = endo  tipo_raíz = 90  pers_núm = no  tiemp_mod = ger  tipo_des = ger
```

La primera entrada es la variante alomórfica para la primera persona del singular de los pretéritos fuertes (p.ej. *dij -e*, frente al alomorfo "regular" *-í* en *part -í*). La segunda es la variante del morfema de gerundio en *riñ - endo*. De esta forma se establece una codificación distinta para cada variante alomórfica, que permite asignar la variante adecuada a cada raíz, como explicaremos al hablar de la gramática.

A la hora de establecer las irregularidades de la desinencia hemos seguido los siguientes criterios siempre que la segmentación entre raíz y terminación no esté muy clara:

- *Elegancia descriptiva* : siempre que se pueda se evitará la "lexicalización" de formas. En todo diccionario computacional aparecen algunas entradas con toda su información flexiva incorporada, debido a que su irregularidad es tan específica que no es rentable tener raíces y desinencias *ad hoc* (por ejemplo, *he, sé, fui* etc.). Como explicamos al hablar de la historia de los procesadores morfológicos, el gran avance teórico se dio cuando se pasó del simple listado de formas flexionadas al conjunto de raíces y morfemas flexivos que se pueden combinar mediante reglas, reduciendo así el espacio dedicado a cada unidad léxica. Por lo tanto, sólo lexicalizaremos aquellas formas que presenten irregularidades únicas con respecto a los paradigmas y, por el contrario, crearemos entradas en aquellos casos donde la variante de la desinencia pueda usarse para varios verbos (aunque éstos sean un número reducido)[64] .

---

[63] La lista completa de las desinencias regulares se verá más adelante dentro de este mismo capítulo, en la sección 6.1.2.

[64] El concepto de *lexicalización* es contrario a los objetivos de esta tesis. Si incluimos las formas flexionadas en el diccionario, ¿para qué necesitamos un procesador morfológico? Por las características de este estudio, primaremos el principio de *elegancia descriptiva* sobre los otros, aunque queremos señalar que la lexicalización es más eficiente en algunos casos.

- *Economía descriptiva* : es importante que haya el menor número posible de entradas, tanto de raíces como de terminaciones flexivas, pues de esta manera se aumenta la velocidad del procesamiento (las reglas de la gramática tienen que probar menos). Este principio parece estar en contradicción con el anterior, pero no es así. Simplemente limita y previene el riesgo de querer hacerlo todo mediante reglas y cargar innecesariamente el diccionario (en el caso extremo, siempre se puede encontrar una segmentación de cualquier forma excepcional, p. ej. *s - é* (de *saber*), a costa de crear una variante única de la raíz y de la desinencia. Si se quiere, todo se puede regularizar. Debido a que el formalismo de unificación es muy potente (en el sentido de que permite tratar cualquier segmentación que se proponga) es necesario restringir su aplicación para obtener mayor eficiencia. Por lo tanto, debemos conseguir la codificación más económica y sencilla que nos permita generar y/o analizar las palabras del castellano. Por eso, en algunas ocasiones, tomaremos decisiones que tal vez no sean las más adecuadas desde el punto de vista teórico, pero que son las más rentables para el procesamiento.

- *Limitar al máximo el número de variantes alomórficas de las desinencias.* Este requisito último está en conjunción con la economía descriptiva: si a la hora de establecer una segmentación hay que escoger entre aumentar el número de raíces o el de desinencias, es preferible aumentar las primeras. La razón es que nuestra regla para flexionar verbos prueba para cada raíz todas las desinencias posibles[65] . Mientras que el número de entradas para las raíces verbales es abierto (por definición, pertenecen al grupo de las categorías léxicas), las desinencias (al igual que el resto de las categorías gramaticales) son un inventario cerrado, y es necesario que este inventario sea lo más reducido posible. Una vez cerrada la lista de desinencias, la ampliación del diccionario siempre será con nuevas entradas léxicas.

A continuación veremos ejemplos de cómo se aplican estos principios. En lugar de explicar uno a uno los distintos valores del rasgo **tipo_des**, los agruparemos según el tipo de irregularidad que presenten. Estas pueden ser de dos tipos:

1. Irregularidades puras
2. Irregularidades meramente ortográficas

6.1.1.3 Irregularidades puras

Dentro de este apartado situamos los llamados *pretéritos fuertes*, los futuros y condicionales irregulares, y los participios irregulares. En las clasificaciones fonológicas estas irregularidades son consideradas aparte, teniendo en cuenta su carácter excepcional y los pocos verbos a los que se extienden. Nosotros incluiremos además las irregularidades en el gerundio y en el imperfecto del subjuntivo, así como los verbos defectivos.

Los escasos verbos españoles que evolucionaron de los perfectos fuertes latinos presentan una variante alomórfica en la desinencia de la primera y tercera del singular, a la que nosotros distinguimos de la "regular" por medio del valor **pret1**:

---

[65] Una forma como *viniera* tiene dos análisis morfológicos posibles, uno con primera persona y otro como tercera del imperfecto del subjuntivo. Por tal motivo, hay que permitir que se prueben todas combinaciones susceptibles de acabar en un análisis con éxito. Este es un principio básico para una gramática declarativa.

```
cadena = e  tipo_raíz = 31  pers_núm = sg_1  tiemp_mod = pret_ind tipo_des = pret1

cadena = o  tipo_raíz = 33  pers_núm = sg_3  tiemp_mod = pret_ind  tipo_des = pret1
```

Los participios irregulares del español comparten con los pretéritos fuertes la característica de que el acento cae en la última sílaba de la raíz, de tal manera que desaparece la secuencia *id*[66] de la desinencia, quedando únicamente *-o*[67] . Esta variante del morfema de participio se distingue por el valor **part1**:

```
cadena = o  tipo_raíz = 99  pers_núm = no  tiemp_mod = part  tipo_des = part1
```

De esta forma, la codificación de los participios irregulares se hará como *hech - o, dich - o, impres - o, frit - o*, etc.. Alguien podría preferir lexicalizar estos participios, ya que obligan a incluir una variante de la raíz exclusivamente para una única forma. Nosotros, por motivos de "elegancia descriptiva," preferimos tratarlos mediante la regla de flexión verbal, puesto que, aunque constituyen un repertorio limitado, el mismo alomorfo de la desinencia puede usarse para todos ellos.

Sobre el tratamiento de los verbos que tienen doble participio, uno fuerte y otro débil, por ejemplo, *impreso, imprimido*, véase la explicación al final de la sección 6.1.5.

Como dice el *Esbozo*, "un reducido número de verbos sincopan la *-e-* y la *-i-* de las terminaciones *-er -ir* de infinitivo cuando este entra en la formación del futuro y del condicional" (R.A.E. 1973;1989:308). En la clasificación académica se dan tres posibilidades:

1. verbos que "interponen una *d* entre la última consonante de la raíz y la *r* del infinitivo," p.ej. *pondré, pondría, vendré, vendría,*
2. verbos que sincopan la vocal sin interposición de consonante: *habré, habría, querré, querría,*
3. verbos que pierden la consonante de la raíz y modifican la vocal radical, como *haré, haría, diré, diría*

Este caso nos permite aplicar nuestro principio tercero de segmentación (limitar el número de alomorfos de la desinencia). Parece claro que desde una perspectiva puramente formal los verbos del segundo y tercer caso presentan una segmentación tal que:

---

[66] Los participios irregulares sólo se dan en la segunda y tercera conjugación.

[67] La R.A.E. considera, en cambio, que hay varios sufijos irregulares: *-cho* en *he - cho, di - cho, -so* en *pre - so, impre - so*, etc.. Aplicando el principio tercero de segmentación, es preferible tener una única desinencia (*-o*) para los participios irregulares que media docena.

RAÍZ + -re, -ría ...

hab - ré	hab - ría	quer - ré	quer - ría
ha - ré	ha - ría	di - ré	di - ría

Por tanto, necesitamos una entrada especial para cada variante alomórfica, a la que asignamos el valor **fut_cond** para distinguirla de la variante regular (que lleva el rasgo **tipo_des = reg**):

```
cadena = ré  tipo_raíz = 41  pers_núm = sg_1  tiemp_mod = futuro  tipo_des = fut_cond
cadena = rás  tipo_raíz = 42  pers_núm = sg_2  tiemp_mod = futuro  tipo_des = fut_cond
⋮
cadena = ría  tipo_raíz = 71  pers_núm = sg_1  tiemp_mod = cond  tipo_des = fut_cond
cadena = rías  tipo_raíz = 72  pers_núm = sg_2  tiemp_mod = cond  tipo_des = fut_cond
⋮
```

Ahora bien, ¿ cómo segmentar los futuros y condicionales con una *-d-* epentética? Ya comentamos anteriormente la dificultad de deslindar claramente la raíz y la terminación en algunos tiempos, y citamos concretamenta la nota 4 de la pág. 256 del *Esbozo* donde los académicos evitan pronunciarse en favor o bien de *ven - dré* o bien de *vend - ré*. Nosotros, en cambio, preferimos la segunda segmentación pues al tener ya entradas para las variantes *-ré*, etc. podemos aprovecharlas, evitando así nuevas desinencias en *-dré, -dría* (escasamente productivas, por otra parte).

Dentro de las irregularidades puras de difícil segmentación tenemos también el gerundio, las terceras personas del pretérito de indicativo y todas las personas del imperfecto de subjuntivo (derivadas precisamente del tema de perfecto). La R.A.E.[68] y otras clasificaciones fonológicas consideran que en estas formas no se da una irregularidad en el morfema flexivo, sino simplemente una alternancia fonológica en la vocal *i*. Es decir, en unos casos aparece la consonante *y* (*ca - yendo*), y en otros casos simplemente no se escribe (*ciñ - endo*). En el primer caso "la imposibilidad silábica /lei.ó/ o /le.ió/, lo mismo que en el caso de plural de *rey* /ˈréy/: /ˈréi.es/ o /ˈré.ies/, explica fonológicamente las formas *le - yó, re - yes*, con consonante *y*." (R.A.E. 1973;1989:174). El segundo caso "la *i* semiconsonante se funde con los sonidos palatales *ll, ñ, ch, j*, cuando le preceden como final de lexema." (Alcina y Blecua 1980:762). Según el *Esbozo*, "se explica así [...] la conciencia lingüística de que en *ñ* y *ll* hay embebido o latente un sonido [j]" (nota 9, pág. 275).

Creemos que desde un punto de vista teórico, estas explicaciones no consideran las irregularidades verbales como un asunto estrictamente morfológico (aunque las explican dentro de la Morfología) sino que la base para decidir si existe una irregularidad es meramente fonológica. Creemos que es más adecuado decir que el

---

[68] Para todas estas irregularidades, véanse los siguientes apartados del *Esbozo*: 2.12.1.f, 2.12.3.A y 2.12.12, y las notas 9, 12, 13, 15 y 18 del capítulo 2.12 (Conjugación irregular).

sufijo de gerundio se manifiesta como *endo* en los contextos morfológicos donde la raíz verbal acaba en *ñ* o *ll* (en lugar de mantener que no hay irregularidad porque el fonema vocálico se ha "fundido" con la consonante final de la raíz, puesto que aunque sea indiscutiblemente cierto por evolución histórica, sincrónicamente el fonema /ñ/ está plenamente consolidado y contrasta con las combinaciones de /ni/ en palabras del castellano actual[69]).

En cualquier caso, para el tramiento informático debemos considerar alomorfos todas aquellas variaciones gráficas en las cadenas de caracteres (nuestro *Principio Gráfico*), de tal forma que en la cadena c i ñ e n d o es pertinente asignar *ciñ* a la raíz verbal y *endo* a la desinencia. De esta manera, distinguiremos los siguientes alomorfos con sus correspondientes valores **ger, pret2** y **imp_subj** basándonos en el paradigma de *ceñir*:

```
ciñ - endo, ciñ - ó, ciñ - eron, ciñ - era, ciñ - ese,
ciñ - eras, ciñ - eses...  ciñ - eran, ciñ - esen.
```

```
cadena = endo  tipo_raíz = 90  pers_núm = no  tiemp_mod = ger  tipo_des = ger

cadena = ó  tipo_raíz = 33  pers_núm = sg_3  tiemp_mod = pret_ind  tipo_des = pret2

cadena = eron  tipo_raíz = 36  pers_núm = pl_3  tiemp_mod = pret_ind  tipo_des = pret2

cadena = era  tipo_raíz = 61  pers_núm = sg_1  tiemp_mod = imp_subj  tipo_des = imp_subj

cadena = ese  tipo_raíz = 61  pers_núm = sg_1  tiemp_mod = imp_subj  tipo_des = imp_subj

cadena = eras  tipo_raíz = 62  pers_núm = sg_1  tiemp_mod = imp_subj  tipo_des = imp_subj

cadena = eses  tipo_raíz = 62  pers_núm = sg_1  tiemp_mod = imp_subj  tipo_des = imp_subj
⋮
```

Es importante señalar que la mayoría de estas variantes alomórficas se utilizan en varios tipos de verbos. Por ejemplo, la tercera persona del plural de los pretéritos fuertes y todas las formas del imperfecto del subjuntivo de los verbos cuya raíz acabe en *-j*: *dij - eron, traj - eron, conduj - eron, dij - era, traj - era, conduj - era*, etc.. Más significativo es que todos los verbos acabados en *-eír* utilizan las mismas desinencias que *ceñir*: *ri - endo, ri - ó, ri - eron, ri - era....* En este caso la Academia reconoce que se trata de una "verdadera irregularidad"[70] .

---

[69] Dice al respecto Navarro Tomás en su *Manual de pronunciación española*:

> Algunos tratados muy corrientes han extendido ... el error de considerar equivalentes el sonido de la *ñ* y el de *n* + *y*, lo cual hace confundir en la pronunciación formas tan distintas como, por ejemplo, *Miño, minio, uñón* y *unión*. (Navarro Tomás 1980:132)

[70] Esta supresión de la *i* no silábica del tema no está condicionada, como en el caso de los verbos en *-eñir*, por razones fonológicas de alcance general y debe considerarse como una verdadera irregularidad. (R.A.E. 1973;1989:277-278).

Con esta enumeración queremos insistir en que estos alomorfos son productivos (y necesarios) dentro del paradigma verbal español. Consideremos ahora si son estrictamente necesarios alomorfos como *yendo, yó, yeron, yera, yese....*

Hay varios grupos de verbos que podrían necesitar estos alomorfos: los acabados en *-uir*, en *-oír*, en *-eer*, y *caer* y sus derivados. En todos ellos, el resultado sería una raíz "regular" (la base radical del infinitivo) más una desinencia "irregular" (desinencia que llevaría un rasgo distinto de **tipo_des = reg**, por ejemplo, **tipo_des = ger2**):

```
hu - yendo    hu - yó    hu - yeron    hu - yera    hu - yese

o  - yendo    o  - yó    o  - yeron    o  - yera    o  - yese

le - yendo    le - yó    le - yeron    le - yera    le - yese

ca - yendo    ca - yó    ca - yeron    ca - yera    ca - yese
```

Esto, en principio, parece una posibilidad de mantener la "regularidad" de la raíz, y por lo tanto suprimir la irregularidad en estas formas. Pero no es así: al tener desinencias irregulares necesitamos mantener la distinción distribucional en el paradigma entre las formas que utilizan las desinencias regulares y las irregulares, y esto sólo es posible especificando qué formas llevan qué desinencias. Conclusión: habría que crear dos entradas para la misma cadena-raíz verbal, una con los códigos numéricos de las formas que llevan desinencias regulares (**tipo_des = reg**)

```
cadena = le  tipo_raíz = 00,11,12...21...31...41...51...71...82,85,99  tipo_des = reg
```

y otra con los códigos de las formas con desinencias irregulares

```
cadena = le  tipo_raíz = 33,36,61,62,63,64,65,66,90  tipo_des = ger,pret2,imp_subj
```

No es posible, como pudiera parecer, agrupar todos los valores de **tipo_des** en una sola entrada porque no habría forma de saber si 33 lleva una desinencia *reg* o *pret2*, y nuestra gramática, por defecto, siempre generará y analizará las dos (tanto la forma correcta como la incorrecta. Véase 6.1.5, "¿ Cómo codificar un verbo particular ?" en la página 170).

Esta solución presenta varios inconvenientes. En primer lugar, contraviene nuestro principio de limitar el número de variantes de las terminaciones, aumentando además el número de raíces verbales. En segundo lugar, los verbos en *-uír* y *-oír* tienen varias formas del paradigma con un alomorfo de la raíz acabado en *-y*:

```
huy - o  huy - es  huy - e  huy - en  huy - a  huy - as....

         oy - es   oy - e   oy - en   oy - e (imperativo)
```

En este caso la partición silábica es /o.yes/, pero nadie asignaría la *y* a la desinencia. Para estos verbos es muy productivo utilizar, por lo tanto, por una parte la raíz que ya existe para unas formas y por otra parte las desinencias que ya existen para otros

verbos. Es decir, sin añadir ninguna entrada nueva al diccionario podemos procesar también

```
huy - ó  huy - eron  huy - endo  huy - era  huy - ese ....

oy  - ó  oy  - eron  oy  - endo  oy  - era  oy  - ese ....
```

y esta es la solución que hemos adoptado.

Sólo nos quedan, por tanto, dos pequeños grupos para los que podría ser rentable crear desinencias especiales. Los acabados en *-eer* son trece (*leer, releer, esleer, peer, creer, acreer, malcreer, descreer, sobreseer, poseer, desposeer, proveer, desproveer*), y los derivados de *caer* son *decaer* y *recaer*. Pensamos que no es nada productivo introducir 16 nuevas desinencias (1 en el gerundio, 2 en el pretérito y 12 en el imperfecto del subjuntivo) para tratar estos 16 verbos. Luego, aplicando nuestro principio tercero de segmentación, aumentaremos las raíces de estos verbos[71] :

```
ley - ó  ley - eron  huy - endo  huy - era  huy - ese ....

cay - ó  cay - eron  cay - endo  cay - era  cay - ese ....
```

en lugar de crear las desinencias *yó, yeron*, etc.

Los verbos defectivos son aquellos que presentan un paradigma incompleto. Este fenómeno puede afectar tanto a verbos regulares como irregulares, de la misma forma que no hay un modelo único de verbo defectivo, aunque la mayoría sólo se emplean en infinitivo y participio. Nuestro sistema de descripción permite especificar para cada caso las formas que se emplean. El procedimiento es el mismo que para el resto de los paradigmas: se asignan los códigos pertinentes a la raíz o raíces y en una de ellas se incluye el valor **defect** (véase 6.1.5, ¿ Cómo codificar un verbo particular ?)

```
cadena = conciern  tipo_raíz = 13,16,53,56    tipo_des = reg

cadena = concern   tipo_raíz = 00,23,26,90,99  tipo_des = reg, defect
```

---

[71] El cambio de 16 desinencias por 16 nuevas raíces es mucho más productivo de lo que pudiera parecer a simple vista. Recordemos que nuestra gramática comprueba para cada raíz todas las terminaciones posibles. Por ejemplo, para generar la tercera del singular del imperfecto del subjuntivo (forma 63) de un verbo cualquiera, la regla contrasta todas las desinencias que lleven **tipo_raíz = 63**. Cuantas más haya, más tendrá que probar para esa forma concreta y para todos los verbos. La diferencia con respecto a las raíces es que la ampliación de entradas sólo afecta a la comprobación de ese verbo en concreto. Por eso, se procesa más rápidamente un verbo con menos variantes de la raíz. Resumiendo, el aumento de desinencias afecta a la velocidad de procesamiento general y el aumento de raíces afecta especialmente al tiempo de procesamiento de ese verbo, de ahí que es preferible el segundo caso.

Por último, hay un reducido número de variantes alomórficas muy particulares que se explicarán en 6.1.4, "Paradigmas irregulares especiales" en la página 161.

6.1.1.4 *Irregularidades meramente ortográficas.*

El resto de las desinencias irregulares son simplemente variantes alomórficas fruto de un cambio ortográfico. Es el caso de los verbos vocálicos acabados en *-oír, -eír, -aer, -eer.* Dichos verbos tienen que llevar tilde en la *i* de la desinencia, por tratarse de palabras con hiato (*Esbozo* apartado 1.8.3.A). En consecuencia, la forma *o - ímos* lleva acento y, por tanto, es distinta de la desinencia regular *-imos.* Dado que es un fenómeno que se produce con regularidad, no "merece" ser tratado como una verdadera irregularidad excepcional del tipo *sé, di, haz* (que son formas lexicalizadas). A pesar de que el número de verbos afectados es escaso, en esta ocasión nos compensa añadir una desinencia, porque de esa forma evitamos la lexicalización (nuestro *principio de elegancia descriptiva*) y reflejamos la regularidad del cambio gráfico.

He aquí la lista de tales irregularidades gráficas, así como el valor que se les asigna:

```
cadena = ímos     tipo_raíz = 14   tipo_des = pres   (o - ímos, re - ímos)

cadena = íste     tipo_raíz = 32   tipo_des = pret2  (ca - íste, le - íste)

cadena = ímos     tipo_raíz = 34   tipo_des = pret2  (ca - ímos, le - ímos)

cadena = ísteis   tipo_raíz = 35   tipo_des = pret2  (o - ísteis, le - ísteis)


cadena = íd    tipo_raíz = 85   tipo_des = imper  (o - íd, re - íd)

cadena = ír    tipo_raíz = 00   tipo_des = infin  (o - ír, re - ír)

cadena = ído   tipo_raíz = 99   tipo_des = part2  (o - ído, ca - ído)
```

6.1.1.5 *La combinación de ambos rasgos*

El rasgo **tipo_des** aparece tanto en la raíz como en la terminación, de igual manera que ocurre con **tipo_raíz.** Necesitamos estas "etiquetas" en ambas cadenas porque el mecanismo de concatenación tiene que comprobar que dichas cadenas son compatibles para sus rasgos de *tipo_raíz* y *tipo_des*[72] . Es decir, para que se puedan "concatenar" una raíz y una desinencia es necesario que ambas compartan el mismo valor en sus rasgos *tipo_raíz* y *tipo_des*[73] . Imaginémonos que queremos generar o analizar la tercera persona del pretérito simple de indicativo del verbo *pedir.* En nuestro diccionario tenemos dos entradas para dicho verbo:

---

[72] Dedicamos un apartado completo al mecanismo de concatenación en el capítulo dedicado a la gramática (sección 7.1.3, "El mecanismo de concatenación morfológica" en la página 203).

[73] Para que el mecanismo de concatenación acepte la combinación es necesario también que pertenezcan a la misma conjugación, es decir, tener el valor idéntico para el rasgo **conjugación.**

```
cadena = ped  tipo_raíz = 00,14,15,...   tipo_des = reg

cadena = pid  tipo_raíz = 11,12... 33... tipo_des = reg
```

Esta codificación se debe entender como que ambas cadenas están en distribución complementaria y que llevan una desinencia regular. Si miramos ahora las desinencias posibles para la forma 33 (tercera del sing. del pretérito) encontramos[74] :

```
cadena = ió   tipo_raíz = 33   tipo_des = reg

cadena = o    tipo_raíz = 33   tipo_des = pret1
```

De las varias combinaciones posibles sólo tendrá éxito *pid* + *ió*, pues coinciden en los valores (**tipo_raíz = 33, tipo_des = reg**); mientras que *ped* + *ió* falla porque no existe el valor 33 en el *tipo_raíz* de *ped*. Análogamente, fracasa la combinación *pid* + *o* (para la tercera del singular del pretérito, se entiende) porque cada elemento tiene diferentes valores para el rasgo **tipo_des** (*reg* y *pret1*, respectivamente).

A continuación damos una lista de combinaciones posibles entre la raíz y la desinencia. Cuando mencionamos *raíz regular* queremos decir verbos con raíz única (código 100, los tradicionales verbos regulares):

		tipo_raíz	tipo_des
1)	raíz regular + des. regular: am- o	100	reg
2)	raíz irregular + des. regular: pid -ió	33	reg
3)	raíz irreg. + des. irreg.: pus -e	31	pret1

6.1.2 Paradigmas regulares

Los tres paradigmas regulares del español forman la base del sistema flexivo verbal. Se caracterizan por presentar una única raíz y las desinencias sirven de punto de referencia para marcar las irregularidades o variaciones en el morfema flexivo dentro de cada conjugación. En nuestra descripción, todos los verbos que se ajusten a estos patrones llevarán la siguiente codificación en su raíz:

- **tipo_raíz = 100**
- **tipo_des = reg**

Damos a continuación la lista completa de todas las cadenas regulares de las tres conjugaciones, con sus rasgos asignados. Esto es un ejemplo de cómo pueden ser las entradas de diccionario en un formalismo de unificación de rasgos.

---

[74] No mostramos la desinencia de la primera conjugación (*-ó*, en *am- ó*) ni la variante alomórfica **pret2** (*cay- ó*) porque su presencia no añade nuevos datos al ejemplo.

```
cadena = o  pers_núm = sg_1   tiemp_mod = pres_ind   conjugación = cjg1 cjg2 cjg3
                                        tipo_raíz = 11  tipo_des = reg

cadena = as  pers_núm = sg_2   tiemp_mod = pres_ind   conjugación = cjg1
                                        tipo_raíz = 12  tipo_des = reg

cadena = es  pers_núm = sg_2   tiemp_mod = pres_ind   conjugación = cjg2 cjg3
                                        tipo_raíz = 12  tipo_des = reg

cadena = a  pers_núm = sg_3   tiemp_mod = pres_ind   conjugación = cjg1
                                        tipo_raíz = 13  tipo_des = reg

cadena = e  pers_núm = sg_3   tiemp_mod = pres_ind   conjugación = cjg2 cjg3
                                        tipo_raíz = 13  tipo_des = reg

cadena = amos  pers_núm = pl_1   tiemp_mod = pres_ind   conjugación = cjg1
                                        tipo_raíz = 14  tipo_des = reg

cadena = emos  pers_núm = pl_1   tiemp_mod = pres_ind   conjugación = cjg2
                                        tipo_raíz = 14  tipo_des = reg

cadena = imos  pers_núm = pl_1   tiemp_mod = pres_ind   conjugación = cjg3
                                        tipo_raíz = 14  tipo_des = reg

cadena = áis  pers_núm = pl_2   tiemp_mod = pres_ind   conjugación = cjg1
                                        tipo_raíz = 15  tipo_des = reg

cadena = éis  pers_núm = pl_2   tiemp_mod = pres_ind   conjugación = cjg2
                                        tipo_raíz = 15  tipo_des = reg

cadena = ís  pers_núm = pl_2   tiemp_mod = pres_ind   conjugación = cjg3
                                        tipo_raíz = 15  tipo_des = reg

cadena = an  pers_núm = pl_3   tiemp_mod = pres_ind   conjugación = cjg1
                                        tipo_raíz = 16  tipo_des = reg

cadena = en  pers_núm = pl_3   tiemp_mod = pres_ind   conjugación = cjg2 cjg3
                                        tipo_raíz = 16  tipo_des = reg

cadena = aba  pers_núm = sg_1   tiemp_mod = impf_ind   conjugación = cjg1
                                        tipo_raíz = 21  tipo_des = reg

cadena = ía  pers_núm = sg_1   tiemp_mod = impf_ind   conjugación = cjg2 cjg3
                                        tipo_raíz = 21  tipo_des = reg

cadena = abas  pers_núm = sg_2   tiemp_mod = impf_ind   conjugación = cjg1
                                        tipo_raíz = 22  tipo_des = reg

cadena = ías  pers_núm = sg_2   tiemp_mod = impf_ind   conjugación = cjg2 cjg3
                                        tipo_raíz = 22  tipo_des = reg

cadena = aba  pers_núm = sg_3   tiemp_mod = impf_ind   conjugación = cjg1
                                        tipo_raíz = 23  tipo_des = reg
```

```
cadena = ía  pers_núm = sg_3   tiemp_mod = impf_ind   conjugación = cjg2 cjg3
                                    tipo_raíz = 23  tipo_des = reg

cadena = ábamos  pers_núm = pl_1   tiemp_mod = impf_ind   conjugación = cjg1
                                    tipo_raíz = 24  tipo_des = reg

cadena = íamos  pers_núm = pl_1   tiemp_mod = impf_ind   conjugación = cjg2 cjg3
                                    tipo_raíz = 24  tipo_des = reg

cadena = abais  pers_núm = pl_2   tiemp_mod = impf_ind   conjugación = cjg1
                                    tipo_raíz = 25  tipo_des = reg

cadena = íais  pers_núm = pl_2   tiemp_mod = impf_ind   conjugación = cjg2 cjg3
                                    tipo_raíz = 25  tipo_des = reg

cadena = aban  pers_núm = pl_3   tiemp_mod = impf_ind   conjugación = cjg1
                                    tipo_raíz = 26  tipo_des = reg

cadena = ían  pers_núm = pl_3   tiemp_mod = impf_ind   conjugación = cjg2 cjg3
                                    tipo_raíz = 26  tipo_des = reg

cadena = é  pers_núm = sg_1   tiemp_mod = pret_ind   conjugación = cjg1
                                    tipo_raíz = 31  tipo_des = reg

cadena = í  pers_núm = sg_1   tiemp_mod = pret_ind   conjugación = cjg2 cjg3
                                    tipo_raíz = 31  tipo_des = reg

cadena = aste  pers_núm = sg_2   tiemp_mod = pret_ind   conjugación = cjg1
                                    tipo_raíz = 32  tipo_des = reg

cadena = iste  pers_núm = sg_2   tiemp_mod = pret_ind   conjugación = cjg2 cjg3
                                    tipo_raíz = 32  tipo_des = reg

cadena = ó  pers_núm = sg_3   tiemp_mod = pret_ind   conjugación = cjg1
                                    tipo_raíz = 33  tipo_des = reg

cadena = ió  pers_núm = sg_3   tiemp_mod = pret_ind   conjugación = cjg2 cjg3
                                    tipo_raíz = 33  tipo_des = reg

cadena = amos  pers_núm = pl_1   tiemp_mod = pret_ind   conjugación = cjg1
                                    tipo_raíz = 34  tipo_des = reg

cadena = imos  pers_núm = pl_1   tiemp_mod = pret_ind   conjugación = cjg2 cjg3
                                    tipo_raíz = 34  tipo_des = reg

cadena = asteis  pers_núm = pl_2   tiemp_mod = pret_ind   conjugación = cjg1
                                    tipo_raíz = 35  tipo_des = reg

cadena = isteis  pers_núm = pl_2   tiemp_mod = pret_ind   conjugación = cjg2 cjg3
                                    tipo_raíz = 35  tipo_des = reg

cadena = aron  pers_núm = pl_3   tiemp_mod = pret_ind   conjugación = cjg1
                                    tipo_raíz = 36  tipo_des = reg

cadena = ieron  pers_núm = pl_3   tiemp_mod = pret_ind   conjugación = cjg2 cjg3
```

```
                                        tipo_raíz = 36  tipo_des = reg

cadena = aré  pers_núm = sg_1   tiemp_mod = futuro   conjugación = cjg1
                                        tipo_raíz = 41  tipo_des = reg

cadena = eré  pers_núm = sg_1   tiemp_mod = futuro   conjugación = cjg2
                                        tipo_raíz = 41  tipo_des = reg

cadena = iré  pers_núm = sg_1   tiemp_mod = futuro   conjugación = cjg3
                                        tipo_raíz = 41  tipo_des = reg

cadena = arás  pers_núm = sg_2   tiemp_mod = futuro   conjugación = cjg1
                                        tipo_raíz = 42  tipo_des = reg

cadena = erás  pers_núm = sg_2   tiemp_mod = futuro   conjugación = cjg2
                                        tipo_raíz = 42  tipo_des = reg

cadena = irás  pers_núm = sg_2   tiemp_mod = futuro   conjugación = cjg3
                                        tipo_raíz = 42  tipo_des = reg

cadena = ará  pers_núm = sg_3   tiemp_mod = futuro   conjugación = cjg1
                                        tipo_raíz = 43  tipo_des = reg

cadena = erá  pers_núm = sg_3   tiemp_mod = futuro   conjugación = cjg2
                                        tipo_raíz = 43  tipo_des = reg

cadena = irá  pers_núm = sg_3   tiemp_mod = futuro   conjugación = cjg3
                                        tipo_raíz = 43  tipo_des = reg

cadena = aremos  pers_núm = pl_1   tiemp_mod = futuro   conjugación = cjg1
                                        tipo_raíz = 44  tipo_des = reg

cadena = eremos  pers_núm = pl_1   tiemp_mod = futuro   conjugación = cjg2
                                        tipo_raíz = 44  tipo_des = reg

cadena = iremos  pers_núm = pl_1   tiemp_mod = futuro   conjugación = cjg3
                                        tipo_raíz = 44  tipo_des = reg

cadena = aréis  pers_núm = pl_2   tiemp_mod = futuro   conjugación = cjg1
                                        tipo_raíz = 45  tipo_des = reg

cadena = eréis  pers_núm = pl_2   tiemp_mod = futuro   conjugación = cjg2
                                        tipo_raíz = 45  tipo_des = reg

cadena = iréis  pers_núm = pl_2   tiemp_mod = futuro   conjugación = cjg3
                                        tipo_raíz = 45  tipo_des = reg

cadena = arán  pers_núm = pl_3   tiemp_mod = futuro   conjugación = cjg1
                                        tipo_raíz = 46  tipo_des = reg

cadena = erán  pers_núm = pl_3   tiemp_mod = futuro   conjugación = cjg2
                                        tipo_raíz = 46  tipo_des = reg

cadena = irán  pers_núm = pl_3   tiemp_mod = futuro   conjugación = cjg3
                                        tipo_raíz = 46  tipo_des = reg
```

```
cadena = e  pers_núm = sg_1   tiemp_mod = pres_subj   conjugación = cjg1
                                      tipo_raíz = 51  tipo_des = reg

cadena = a  pers_núm = sg_1   tiemp_mod = pres_subj   conjugación = cjg2 cjg3
                                      tipo_raíz = 51  tipo_des = reg

cadena = es  pers_núm = sg_2   tiemp_mod = pres_subj   conjugación = cjg1
                                      tipo_raíz = 52  tipo_des = reg

cadena = as  pers_núm = sg_2   tiemp_mod = pres_subj   conjugación = cjg2 cjg3
                                      tipo_raíz = 52  tipo_des = reg

cadena = e  pers_núm = sg_3   tiemp_mod = pres_subj   conjugación = cjg1
                                      tipo_raíz = 53  tipo_des = reg

cadena = a  pers_núm = sg_3   tiemp_mod = pres_subj   conjugación = cjg2 cjg3
                                      tipo_raíz = 53  tipo_des = reg

cadena = emos  pers_núm = pl_1   tiemp_mod = pres_subj   conjugación = cjg1
                                      tipo_raíz = 54  tipo_des = reg

cadena = amos  pers_núm = pl_1   tiemp_mod = pres_subj   conjugación = cjg2 cjg3
                                      tipo_raíz = 54  tipo_des = reg

cadena = éis  pers_núm = pl_2   tiemp_mod = pres_subj   conjugación = cjg1
                                      tipo_raíz = 55  tipo_des = reg

cadena = áis  pers_núm = pl_2   tiemp_mod = pres_subj   conjugación = cjg2 cjg3
                                      tipo_raíz = 55  tipo_des = reg

cadena = en  pers_núm = pl_3   tiemp_mod = pres_subj   conjugación = cjg1
                                      tipo_raíz = 56  tipo_des = reg

cadena = an  pers_núm = pl_3   tiemp_mod = pres_subj   conjugación = cjg2 cjg3
                                      tipo_raíz = 56  tipo_des = reg

cadena = ara  pers_núm = sg_1   tiemp_mod = imp_subj   conjugación = cjg1
                                      tipo_raíz = 61  tipo_des = reg

cadena = ase  pers_núm = sg_1   tiemp_mod = imp_subj   conjugación = cjg1
                                      tipo_raíz = 61  tipo_des = reg

cadena = iera  pers_núm = sg_1   tiemp_mod = imp_subj   conjugación = cjg2 cjg3
                                      tipo_raíz = 61  tipo_des = reg

cadena = iese  pers_núm = sg_1   tiemp_mod = imp_subj   conjugación = cjg2 cjg3
                                      tipo_raíz = 61  tipo_des = reg

cadena = aras  pers_núm = sg_2   tiemp_mod = imp_subj   conjugación = cjg1
                                      tipo_raíz = 62  tipo_des = reg

cadena = ases  pers_núm = sg_2   tiemp_mod = imp_subj   conjugación = cjg1
                                      tipo_raíz = 62  tipo_des = reg
```

```
cadena = ieras  pers_núm = sg_2   tiemp_mod = imp_subj   conjugación = cjg2 cjg3
                                      tipo_raíz = 62  tipo_des = reg

cadena = ieses  pers_núm = sg_2   tiemp_mod = imp_subj   conjugación = cjg2 cjg3
                                      tipo_raíz = 62  tipo_des = reg

cadena = ara  pers_núm = sg_3   tiemp_mod = imp_subj   conjugación = cjg1
                                      tipo_raíz = 63  tipo_des = reg

cadena = ase  pers_núm = sg_3   tiemp_mod = imp_subj   conjugación = cjg1
                                      tipo_raíz = 63  tipo_des = reg

cadena = iera  pers_núm = sg_3   tiemp_mod = imp_subj   conjugación = cjg2 cjg3
                                      tipo_raíz = 63  tipo_des = reg

cadena = iese  pers_núm = sg_3   tiemp_mod = imp_subj   conjugación = cjg2 cjg3
                                      tipo_raíz = 63  tipo_des = reg

cadena = áramos  pers_núm = pl_1   tiemp_mod = imp_subj   conjugación = cjg1
                                      tipo_raíz = 64  tipo_des = reg

cadena = ásemos  pers_núm = pl_1   tiemp_mod = imp_subj   conjugación = cjg1
                                      tipo_raíz = 64  tipo_des = reg

cadena = iéramos  pers_núm = pl_1   tiemp_mod = imp_subj   conjugación = cjg2 cjg3
                                      tipo_raíz = 64  tipo_des = reg

cadena = iésemos  pers_núm = pl_1   tiemp_mod = imp_subj   conjugación = cjg2 cjg3
                                      tipo_raíz = 64  tipo_des = reg

cadena = arais  pers_núm = pl_2   tiemp_mod = imp_subj   conjugación = cjg1
                                      tipo_raíz = 65  tipo_des = reg

cadena = aseis  pers_núm = pl_2   tiemp_mod = imp_subj   conjugación = cjg1
                                      tipo_raíz = 65  tipo_des = reg

cadena = ierais  pers_núm = pl_2   tiemp_mod = imp_subj   conjugación = cjg2 cjg3
                                      tipo_raíz = 65  tipo_des = reg

cadena = ieseis  pers_núm = pl_2   tiemp_mod = imp_subj   conjugación = cjg2 cjg3
                                      tipo_raíz = 65  tipo_des = reg

cadena = aran  pers_núm = pl_3   tiemp_mod = imp_subj   conjugación = cjg1
                                      tipo_raíz = 66  tipo_des = reg

cadena = asen  pers_núm = pl_3   tiemp_mod = imp_subj   conjugación = cjg1
                                      tipo_raíz = 66  tipo_des = reg

cadena = ieran  pers_núm = pl_3   tiemp_mod = imp_subj   conjugación = cjg2 cjg3
                                      tipo_raíz = 66  tipo_des = reg

cadena = iesen  pers_núm = pl_3   tiemp_mod = imp_subj   conjugación = cjg2 cjg3
                                      tipo_raíz = 66  tipo_des = reg

cadena = aría  pers_núm = sg_1   tiemp_mod = cond  conjugación = cjg1
```

```
                                        tipo_raíz = 71  tipo_des = reg

cadena = ería  pers_núm = sg_1   tiemp_mod = cond  conjugación = cjg2
                                        tipo_raíz = 71  tipo_des = reg

cadena = iría  pers_núm = sg_1   tiemp_mod = cond  conjugación = cjg3
                                        tipo_raíz = 71  tipo_des = reg

cadena = arías  pers_núm = sg_2   tiemp_mod = cond  conjugación = cjg1
                                        tipo_raíz = 72  tipo_des = reg

cadena = erías  pers_núm = sg_2   tiemp_mod = cond  conjugación = cjg2
                                        tipo_raíz = 72  tipo_des = reg

cadena = irías  pers_núm = sg_2   tiemp_mod = cond  conjugación = cjg3
                                        tipo_raíz = 72  tipo_des = reg

cadena = aría  pers_núm = sg_3   tiemp_mod = cond  conjugación = cjg1
                                        tipo_raíz = 73  tipo_des = reg

cadena = ería  pers_núm = sg_3   tiemp_mod = cond  conjugación = cjg2
                                        tipo_raíz = 73  tipo_des = reg

cadena = iría  pers_núm = sg_3   tiemp_mod = cond  conjugación = cjg3
                                        tipo_raíz = 73  tipo_des = reg

cadena = aríamos  pers_núm = pl_1   tiemp_mod = cond  conjugación = cjg1
                                        tipo_raíz = 74  tipo_des = reg

cadena = eríamos  pers_núm = pl_1   tiemp_mod = cond  conjugación = cjg2
                                        tipo_raíz = 74  tipo_des = reg

cadena = iríamos  pers_núm = pl_1   tiemp_mod = cond  conjugación = cjg3
                                        tipo_raíz = 74  tipo_des = reg

cadena = aríais  pers_núm = pl_2   tiemp_mod = cond  conjugación = cjg1
                                        tipo_raíz = 75  tipo_des = reg

cadena = eríais  pers_núm = pl_2   tiemp_mod = cond  conjugación = cjg2
                                        tipo_raíz = 75  tipo_des = reg

cadena = iríais  pers_núm = pl_2   tiemp_mod = cond  conjugación = cjg3
                                        tipo_raíz = 75  tipo_des = reg

cadena = arían  pers_núm = pl_3   tiemp_mod = cond  conjugación = cjg1
                                        tipo_raíz = 76  tipo_des = reg

cadena = erían  pers_núm = pl_3   tiemp_mod = cond  conjugación = cjg2
                                        tipo_raíz = 76  tipo_des = reg

cadena = irían  pers_núm = pl_3   tiemp_mod = cond  conjugación = cjg3
                                        tipo_raíz = 76  tipo_des = reg

cadena = a  pers_núm = sg_2   tiemp_mod = imper   conjugación = cjg1
                                        tipo_raíz = 82  tipo_des = reg
```

```
cadena = e  pers_núm = sg_2   tiemp_mod = imper   conjugación = cjg2 cjg3
                                        tipo_raíz = 82  tipo_des = reg

cadena = ad  pers_núm = pl_2   tiemp_mod = imper   conjugación = cjg1
                                        tipo_raíz = 85  tipo_des = reg

cadena = ed  pers_núm = pl_2   tiemp_mod = imper   conjugación = cjg2
                                        tipo_raíz = 85  tipo_des = reg

cadena = id  pers_núm = pl_2   tiemp_mod = imper   conjugación = cjg3
                                        tipo_raíz = 85  tipo_des = reg

cadena = ar pers_núm = no   tiemp_mod = infin   conjugación = cjg1
                                        tipo_raíz = 00  tipo_des = reg

cadena = er pers_núm = no   tiemp_mod = infin   conjugación = cjg2
                                        tipo_raíz = 00  tipo_des = reg

cadena = ir pers_núm = no   tiemp_mod = infin   conjugación = cjg3
                                        tipo_raíz = 00  tipo_des = reg

cadena = ando pers_núm = no   tiemp_mod = ger   conjugación = cjg1
                                        tipo_raíz = 90  tipo_des = reg

cadena = iendo pers_núm = no   tiemp_mod = ger   conjugación = cjg2 cjg3
                                        tipo_raíz = 90  tipo_des = reg

cadena = ado pers_núm = no   tiemp_mod = part   conjugación = cjg1
                                        tipo_raíz = 91  tipo_des = reg

cadena = ido pers_núm = no   tiemp_mod = part   conjugación = cjg2 cjg3
                                        tipo_raíz = 91  tipo_des = reg
```

6.1.3 Paradigmas irregulares

A continuación describimos 13 paradigmas o modelos irregulares de conjugación. La clave para incluir un verbo o grupos de verbos en un modelo determinado es la distribución de las distintas raíces en la conjugación (es decir, que el número de raíces resultante y los códigos del rasgo **tipo_raíz** de cada una de las raíces sean los mismos para cada verbo del grupo). Como explicamos anteriormente, los argumentos fonológicos no se tendrán en cuenta. Esto significa que varios grupos de verbos aparecerán clasificados en el mismo paradigma aunque sus irregularidades fonológicas no tengan nada en común. Para evitar el desconcierto que esto pudiera producir, estableceremos *subgrupos*, recogiendo las peculiaridades específicas como, por ejemplo, cambios vocálicos, consonánticos, participios irregulares, desinencias irregulares, etc.

Para ayudar a identificar la distribución formal de las raíces, daremos un esquema gráfico al principio de cada modelo. En dicho esquema utilizaremos la siguiente convención para distinguir unas raíces de otras:

```
111 :  primera raíz  (la raíz "regular")
222 :  segunda raíz
333 :  tercera raíz
444 :  cuarta raíz
...
```

Unos breves comentarios antes de pasar a los paradigmas. En primer lugar, recordar que este tipo de clasificación ya fue adelantada en el siglo pasado por Andrés Bello. Cualquiera puede observar un innegable paralelismo entre las "clases de verbos" del insigne gramático americano y nuestros modelos. La razón es obvia pues ambos se basan en el concepto de que determinados cambios se dan siempre en determinadas formas (los famosos *órdenes* de Bello). La diferencia más notable entre ambas es que la clasificación de Bello no tiene en cuenta los cambios ortográficos y nosotros sí, por razones suficientemente explicadas.

El segundo comentario tiene que ver con la finalidad de la clasificación que sigue. Los paradigmas verbales que presentamos tienen como único fin servir de guía en la elaboración del diccionario para GRAMPAL, y como demostración de que el formalismo puede describir y tratar todos los tipos de irregularidad de los verbos castellanos. Sin embargo, es importante destacar que el método de codificación propuesto aquí es útil y eficiente sin necesidad de establecer paradigmas y "agrupaciones" de verbos. Por lo tanto, no pretendemos agotar todos los casos posibles de irregularidades, con la seguridad de que nuestro método puede codificar cualquiera. Se trata únicamente de rellenar el esquema y asignar de acuerdo con él el código o códigos correspondientes a cada raíz.

Por otra parte, tampoco debe entenderse que seguimos el modelo paradigmático en el tratamiento de la morfología, simplemente porque en ningún momento proponemos una forma como base y las otras se derivan de ella. Para nosotros, todas las variantes alomórficas de la raíz están al mismo nivel y cada una sirve para formar sólo determinadas formas flexionadas. Para establecer el inventario de alomorfos tuvimos que elaborar tablas de las conjugaciones y nos parece pertinente y útil mostrar las similitudes distribucionales que se dan entre verbos que habitualmente se clasifican en paradigmas diferentes.

Incluimos a continuación el inventario de todas las desinencias irregulares, para que sirva de referencia en las descripciones:

```
cadena = ímos   pers_núm = pl_1   tiemp_mod = pres_ind   conjugacion = cjg3
                                     tipo_raíz = 14   tipo_des = pres

cadena = e   pers_núm = sg_1     tiemp_mod = pret_ind   conjugacion = cjg2 cjg3
                                     tipo_raíz = 31   tipo_des = pret1

cadena = í   pers_núm = sg_1     tiemp_mod = pret_ind   conjugacion = cjg2 cjg3
                                     tipo_raíz = 31  tipo_des = pret2

cadena = iste   pers_núm = sg_2   tiemp_mod = pret_ind   conjugacion = cjg2 cjg3
                                     tipo_raíz = 32  tipo_des = pret1

cadena = íste   pers_núm = sg_2   tiemp_mod = pret_ind   conjugacion = cjg2 cjg3
                                     tipo_raíz = 32   tipo_des = pret2

cadena = o   pers_núm = sg_3   tiemp_mod = pret_ind   conjugacion = cjg2 cjg3
                                     tipo_raíz = 33   tipo_des = pret1

cadena = ó   pers_núm = sg_3   tiemp_mod = pret_ind   conjugacion = cjg2 cjg3
                                     tipo_raíz = 33   tipo_des = pret2

cadena = imos   pers_núm = pl_1   tiemp_mod = pret_ind   conjugacion = cjg2 cjg3
                                     tipo_raíz = 34   tipo_des = pret1

cadena = ímos   pers_núm = pl_1   tiemp_mod = pret_ind   conjugacion = cjg2 cjg3
                                     tipo_raíz = 34   tipo_des = pret2

cadena = isteis   pers_núm = pl_2   tiemp_mod = pret_ind   conjugacion = cjg2 cjg3
                                     tipo_raíz = 35   tipo_des = pret1

cadena = ísteis   pers_núm = pl_2   tiemp_mod = pret_ind   conjugacion = cjg2 cjg3
                                     tipo_raíz = 35   tipo_des = pret2
```

```
cadena = ieron   pers_núm = pl_3   tiemp_mod = pret_ind   conjugacion = cjg2 cjg3
                                     tipo_raíz = 36   tipo_des = pret1

cadena = eron   pers_núm = pl_3   tiemp_mod = pret_ind   conjugacion = cjg2 cjg3
                                     tipo_raíz = 36   tipo_des = pret2

cadena = ré    pers_núm = sg_1   tiemp_mod = futuro   conjugacion = cjg2 cjg3
                                     tipo_raíz = 42   tipo_des = fut_cond

cadena = rás   pers_núm = sg_2   tiemp_mod = futuro   conjugacion = cjg2 cjg3
                                     tipo_raíz = 42   tipo_des = fut_cond

cadena = rá   pers_núm = sg_3   tiemp_mod = futuro   conjugacion = cjg2 cjg3
                                     tipo_raíz = 43   tipo_des = fut_cond

cadena = remos   pers_núm = pl_1   tiemp_mod = futuro   conjugacion = cjg2 cjg3
                                     tipo_raíz = 44   tipo_des = fut_cond

cadena = réis   pers_núm = pl_2   tiemp_mod = futuro   conjugacion = cjg2 cjg3
                                     tipo_raíz = 45   tipo_des = fut_cond

cadena = rán   pers_núm = pl_3   tiemp_mod = futuro   conjugacion = cjg2 cjg3
                                     tipo_raíz = 46   tipo_des = fut_cond

cadena = era   pers_núm = sg_1    tiemp_mod = imp_subj   conjugacion = cjg2 cjg3
                                     tipo_raíz = 61   tipo_des = imp_subj

cadena = ese   pers_núm = sg_1    tiemp_mod = imp_subj   conjugacion = cjg2 cjg3
                                     tipo_raíz = 61   tipo_des = imp_subj

cadena = eras   pers_núm = sg_2   tiemp_mod = imp_subj   conjugacion = cjg2 cjg3
                                     tipo_raíz = 62   tipo_des = imp_subj

cadena = eses   pers_núm = sg_2   tiemp_mod = imp_subj   conjugacion = cjg2 cjg3
                                     tipo_raíz = 62   tipo_des = imp_subj

cadena = era   pers_núm = sg_3   tiemp_mod = imp_subj   conjugacion = cjg2 cjg3
                                     tipo_raíz = 63   tipo_des = imp_subj

cadena = ese   pers_núm = sg_3   tiemp_mod = imp_subj   conjugacion = cjg2 cjg3
                                     tipo_raíz = 63   tipo_des = imp_subj

cadena = éramos   pers_núm = pl_1   tiemp_mod = imp_subj   conjugacion = cjg2 cjg3
                                     tipo_raíz = 64   tipo_des = imp_subj

cadena = ésemos   pers_núm = pl_1   tiemp_mod = imp_subj   conjugacion = cjg2 cjg3
                                     tipo_raíz = 64   tipo_des = imp_subj

cadena = erais   pers_núm = pl_2   tiemp_mod = imp_subj   conjugacion = cjg2 cjg3
                                     tipo_raíz = 65   tipo_des = imp_subj

cadena = eseis   pers_núm = pl_2   tiemp_mod = imp_subj   conjugacion = cjg2 cjg3
                                     tipo_raíz = 65   tipo_des = imp_subj

cadena = eran   pers_núm = pl_3   tiemp_mod = imp_subj   conjugacion = cjg2 cjg3
```

```
                                        tipo_raíz = 66   tipo_des = imp_subj

cadena = esen   pers_núm = pl_3   tiemp_mod = imp_subj   conjugacion = cjg2 cjg3
                                        tipo_raíz = 66   tipo_des = imp_subj

cadena = ría   pers_núm = sg_1    tiemp_mod = cond   conjugacion = cjg2 cjg3
                                        tipo_raíz = 71   tipo_des = fut_cond

cadena = rías   pers_núm = sg_2   tiemp_mod = cond   conjugacion = cjg2 cjg3
                                        tipo_raíz = 72   tipo_des = fut_cond

cadena = ría   pers_núm = sg_3   tiemp_mod = cond   conjugacion = cjg2 cjg3
                                        tipo_raíz = 73   tipo_des = fut_cond

cadena = ríamos   pers_núm = pl_1   tiemp_mod = cond   conjugacion = cjg2 cjg3
                                        tipo_raíz = 74   tipo_des = fut_cond

cadena = ríais   pers_núm = pl_2   tiemp_mod = cond   conjugacion = cjg2 cjg3
                                        tipo_raíz = 75   tipo_des = fut_cond

cadena = rían   pers_núm = pl_3   tiemp_mod = cond   conjugacion = cjg2 cjg3
                                        tipo_raíz = 76   tipo_des = fut_cond

cadena = íd   pers_núm = pl_2   tiemp_mod = imper   conjugacion = cjg2 cjg3
                                        tipo_raíz = 85   tipo_des = imper

cadena = ír,   pers_núm = no    tiemp_mod = infin   conjugacion = cjg2 cjg3
                                        tipo_raíz = 0   tipo_des = infin

cadena = o,   pers_núm = no    tiemp_mod = part   conjugacion = cjg2 cjg3
                                        tipo_raíz = 99   tipo_des = part1

cadena = ído,   pers_núm = no    tiemp_mod = part   conjugacion = cjg2 cjg3
                                        tipo_raíz = 99   tipo_des = part2
```

Modelo 1

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	222	222	222	111	111	222	
impf_ind	111	111	111	111	111	111	
pret_ind	111	111	111	111	111	111	
futuro	111	111	111	111	111	111	
pres_subj	222	222	222	111	111	222	
imp_subj	111	111	111	111	111	111	
cond	111	111	111	111	111	111	
imper		222			111		
infin							111
ger							111
part							111

Tabla 4. Tipo 1a: ACERTAR, CONTAR, MOLER, ADQUIRIR. Irregularidad en las formas fuertes (diptongación)							
	sg_1	**sg_2**	**sg_3**	**pl_1**	**pl_2**	**pl_3**	**no**
pres_ind	aciert - o	aciert - as	aciert - a	acert - amos	acert - áis	aciert - an	
impf_ind	acert - aba	acert - abas	acert - aba	acert - ábamos	acert - abais	acert - aban	
pret_ind	acert - é	acert - aste	acert - ó	acert - amos	acert - asteis	acert - aron	
futuro	acert - aré	acert - arás	acert - ará	acert - aremos	acert - aréis	acert - arán	
pres_subj	aciert - e	aciert - es	aciert - e	acert - emos	acert - éis	aciert - en	
imp_subj	acert - ara, ase	acert - aras, ases	acert - ara, ase	acert - áramos, ásemos	acert - arais, aseis	acert - aran, asen	
cond	acert - aría	acert - arías	acert - aría	acert - aríamos	acert - aríais	acert - arían	
imper		aciert - a			acert - ad		
infin							acert - ar
ger							acert - ando
part							acert - ado

- **tipo_raíz**
  1. ***acert***: 00 14 15 21 22 23 24 25 26 31 32 33 34 35 36 41 42 43 44 45 46 54 55 61 62 63 64 65 66 71 72 73 74 75 76 85 90 99
  2. ***aciert***: 11 12 13 16 51 52 53 56 82
- **tipo_des**
  1. ***reg***: para todas las formas

Tabla 5. Tipo 1b: OLER, ERRAR. Irregularidad en las formas fuertes (cambios ortográficos)							
	sg_1	**sg_2**	**sg_3**	**pl_1**	**pl_2**	**pl_3**	**no**
pres_ind	huel - o	huel - es	huel - e	ol - emos	ol - éis	huel - en	
impf_ind	ol - ía	ol - ías	ol - ía	ol - íamos	ol - íais	ol - ían	
pret_ind	ol - í	ol - iste	ol - ió	ol - imos	ol - isteis	ol - ieron	
futuro	ol - eré	ol - erás	ol - erá	ol - eremos	ol - eréis	ol - erán	
pres_subj	huel - a	huel - as	huel - a	ol - amos	ol - áis	huel - an	
imp_subj	ol - iera, iese	ol - ieras, ieses	ol - iera, iese	ol - iéramos, iésemos	ol - ierais, ieseis	ol - ieran, iesen	
cond	ol - ería	ol - erías	ol - ería	ol - eríamos	ol - eríais	ol - erían	
imper		huel - e			ol - ed		
infin							ol - er
ger							ol - iendo
part							ol - ido

- **tipo_raíz**
  1. ***ol***: 00 14 15 21 22 23 24 25 26 31 32 33 34 35 36 41 42 43 44 45 46 54 55 61 62 63 64 65 66 71 72 73 74 75 76 85 90 99
  2. ***huel***: 11 12 13 16 51 52 53 56 82
- **tipo_des**
  1. ***reg***: para todas las formas

Tabla 6. Tipo 1c: DESVIAR, ACTUAR, AULLAR. Irregularidad en las formas fuertes (acentuación de la vocal débil)							
	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	actú - o	actú - as	actú - a	actu - amos	actu - áis	actú - an	
impf_ind	actu - aba	actu - abas	actu - aba	actu - ábamos	actu - abais	actu - aban	
pret_ind	actu - é	actu - aste	actu - ó	actu - amos	actu- asteis	actu - aron	
futuro	actu - aré	actu - arás	actu - ará	actu - aremos	actu - aréis	actu - arán	
pres_subj	actú - e	actú - es	actú - e	actu - emos	actu - éis	actú - en	
imp_subj	actu - ara, ase	actu - aras, ases	actu - ara, ase	actu - áramos, ásemos	actu - arais, aseis	actu - aran, asen	
cond	actu - aría	actu - arías	actu - aría	actu - aríamos	actu - aríais	actu - arían	
imper		actú - a			actu - ad		
infin							actu - ar
ger							actu - ando
part							actu - ado

- **tipo_raíz**
  1. ***actu***: 00 14 15 21 22 23 24 25 26 31 32 33 34 35 36 41 42 43 44 45 46 54 55 61 62 63 64 65 66 71 72 73 74 75 76 85 90 99
  2. ***actú***: 11 12 13 16 51 52 53 56 82
- **tipo_des**
  1. ***reg***: para todas las formas

Dentro de este subtipo se da un curioso contraste. Las clasificaciones fonológicas suelen dedicar un apartado especial a los verbos acabados en *-iar, -uar* y a los que llevan en la raíz los grupos vocálicos *-ai-, -au-*. A estos verbos se les considera regulares aunque con problemas de delimitación vocálica (R.A.E. 1973;1989: 329 y ss.). Las diferencias básicas entre estos dos grupos de verbos son:

- acabados en *-iar, -uar*:
  1. vocal débil átona (diptongo creciente): *cambiar* (/kám.bio/), *averiguar* (/a.be.rí.guo/)
  2. vocal débil tónica (hiato creciente): *desviar* (/des.bí.o/), *actuar* (/ak.tú.o/)
- con *ai, au* en el radical:
  1. diptongo decreciente: *bailar* (/bái.lo/), *causar* (/káu.so)
  2. hiato decreciente: *aullar* (/a.ú.llo/)

Desde la perpectiva formal únicamente son regulares (es decir, tienen raíz única) los verbos que se articulan como diptongos (i.e. los de vocal débil átona), con las excepciones de aquellos que presentan además un cambio gráfico (p. ej. *averigüe*, ver tipo 2 en nuestra clasificación). En cambio, los verbos con hiato tienen siempre más de un alomorfo de la raíz, pues llevan acento gráfico en las vocales débiles (y algunos experimentan, además, cambios gráficos en las consonantes, p. ej. *enraiz- ar, enraíz- o, enraíc- e, enraic- emos*, que lo incluimos en nuestro tipo 3).

Otros verbos que pertenecen a este paradigma son[75] :

acertar	*acordar*	*acostar*	*acrecentar*	*alentar*
aliquebrar	*apacentar*	*apernar*	*apostar*	*apretar*
aprobar	*arrendar*	*ascender*	*aserrar*	*asonar*
aspaventar	*atender*	*atravesar*	*atronar*	*aventar*
calentar	*cernir*	*cerrar*	*cimentar*	*coarrendar*
colar	*comprobar*	*concertar*	*concordar*	*condescender*
condoler	*confesar*	*conmover*	*contender*	*consolar*
contar	*costar*	*defender*	*degollar*	*demoler*
demostrar	*desacertar*	*desalentar*	*desapretar*	*desatender*
descender	*descerrar*	*descollar*	*desconcertar*	*descontar*
desdentar	*desenterrar*	*desenvolver*	*desgobernar*	*deshelar*
desherbar	*desmembrar*	*despertar*	*despernar*	*despoblar*
desterrar	*devolver*	*discernir*	*disolver*	*disonar*
distender	*doler*	*encender*	*encerrar*	*encomendar*
encontrar	*encordar*	*enhestar*	*enmendar*	*ensangrentar*
entender	*enterrar*	*entrecerrar*	*envolver*	*entrepernar*
escarmentar	*extender*	*gobernar*	*hacendar*	*heder*
helar	*hendir*	*herbar*	*herrar*	*incensar*
infernar	*llover*	*malsonar*	*manifestar*	*mentar*
merendar	*moler*	*morder*	*mostrar*	*nevar*
oler	*pensar*	*perder*	*perniquebrar*	*promover*
quebrar	*reapretar*	*recalentar*	*recomendar*	*recordar*
recostar	*regimentar*	*remendar*	*remorder*	*remover*
renovar	*repensar*	*repoblar*	*reprobar*	*resollar*
resolver	*resonar*	*resquebrar*	*retemblar*	*reventar*
revolver	*rodar*	*salpimentar*	*sarmentar*	*sembrar*
sentar	*serrar*	*sobreentender*	*solar*	*soldar*
soñar	*subarrendar*	*temblar*	*tentar*	*tender*
trascender	*ventar*	*verter*	*volar*	*volver*

[75] Sólo incluimos los verbos de uso cotidiano. Así por ejemplo evitaremos verbos como *herventar, jimenzar*, etc.

Modelo 2

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	111	111	111	111	111	111	
impf_ind	111	111	111	111	111	111	
pret_ind	222	111	111	111	111	111	
futuro	111	111	111	111	111	111	
pres_subj	222	222	222	222	222	222	
imp_subj	111	111	111	111	111	111	
cond	111	111	111	111	111	111	
imper		111			111		
infin							111
ger							111
part							111

Tabla 7. Tipo 2a: CAZAR. Cambios gráficos en consonantes ($z \rightarrow c$)							
	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	caz - o	caz - as	caz - a	caz - amos	caz - áis	caz - an	
impf_ind	caz - aba	caz - abas	caz - aba	caz - ábamos	caz - abais	caz - aban	
pret_ind	cac - é	caz - aste	caz - ó	caz - amos	caz - asteis	caz - aron	
futuro	caz - aré	caz - arás	caz - ará	caz - aremos	caz - aréis	caz - arán	
pres_subj	cac - e	cac - es	cac - e	cac - emos	cac - éis	cac - en	
imp_subj	caz - ara, ase	caz - aras, ases	caz - ara, ase	caz - áramos, ásemos	caz - arais, aseis	caz - aran, asen	
cond	caz - aría	caz - arías	caz - aría	caz - aríamos	caz - aríais	caz - arían	
imper		caz - a			caz - ad		
infin							caz - ar
ger							caz - ando
part							caz - ado

- **tipo_raíz**
  1. *caz*: 00 11 12 13 14 15 16 21 22 23 24 25 26 32 33 34 35 36 41 42 43 44 45 46 61 62 63 64 65 66 71 72 73 74 75 76 82 85 90 99
  2. *cac*: 31 51 52 53 54 55 56
- **tipo_des**
  1. *reg*: para todas las formas

Tabla 8. Tipo 2b: **FABRICAR**. Cambios gráficos en consonantes (*c* → *qu*)							
	sg_1	**sg_2**	**sg_3**	**pl_1**	**pl_2**	**pl_3**	**no**
pres_ind	fabric - o	fabric - as	fabric - a	fabric - amos	fabric - áis	fabric - an	
impf_ind	fabric - aba	fabric - abas	fabric - aba	fabric - ábamos	fabric - abais	fabric - aban	
pret_ind	fabriqu - é	fabric - aste	fabric - ó	fabric - amos	fabric - asteis	fabric - aron	
futuro	fabric - aré	fabric - arás	fabric - ará	fabric - aremos	fabric - aréis	fabric - arán	
pres_subj	fabriqu - e	fabriqu - es	fabriqu - e	fabriqu - emos	fabriqu - éis	fabriqu - en	
imp_subj	fabric - ara, ase	fabric - aras, ases	fabric - ara, ase	fabric - áramos, ásemos	fabric - arais, aseis	fabric - aran, asen	
cond	fabric - aría	fabric - arías	fabric - aría	fabric - aríamos	fabric - aríais	fabric - arían	
imper		fabric - a			fabric - ad		
infin							fabric - ar
ger							fabric - ando
part							fabric - ado

- **tipo_raíz**
  1. ***fabric***: 00 11 12 13 14 15 16 21 22 23 24 25 26 32 33 34 35 36 41 42 43 44 45 46 61 62 63 64 65 66 71 72 73 74 75 76 82 85 90 99
  2. ***fabriqu***: 31 51 52 53 54 55 56
- **tipo_des**
  1. ***reg***: para todas las formas

Tabla 9. Tipo 2c: PAGAR. Cambios gráficos en consonantes (*g* → *gu*)							
	sg_1	**sg_2**	**sg_3**	**pl_1**	**pl_2**	**pl_3**	**no**
pres_ind	pag - o	pag - as	pag - a	pag - amos	pag - áis	pag - an	
impf_ind	pag - aba	pag - abas	pag - aba	pag - ábamos	pag - abais	pag - aban	
pret_ind	pagu - é	pag - aste	pag - ó	pag - amos	pag - asteis	pag - aron	
futuro	pag - aré	pag - arás	pag - ará	pag - aremos	pag - aréis	pag - arán	
pres_subj	pagu - e	pagu - es	pagu - e	pagu - emos	pagu - éis	pagu - en	
imp_subj	pag - ara, ase	pag - aras, ases	pag - ara, ase	pag - áramos, ásemos	pag - arais, aseis	pag - aran, asen	
cond	pag - aría	pag - arías	pag - aría	pag - aríamos	pag - aríais	pag - arían	
imper		pag - a			pag - ad		
infin							pag - ar
ger							pag - ando
part							pag - ado

- **tipo_raíz**
  1. ***pag***: 00 11 12 13 14 15 16 21 22 23 24 25 26 32 33 34 35 36 41 42 43 44 45 46 61 62 63 64 65 66 71 72 73 74 75 76 82 85 90 99
  2. ***pagu***: 31 51 52 53 54 55 56
- **tipo_des**
  1. ***reg***: para todas las formas

Tabla 10. Tipo 2d: **AVERIGUAR.** Cambios gráficos en vocal (*g* → *gü*)							
	sg_1	**sg_2**	**sg_3**	**pl_1**	**pl_2**	**pl_3**	**no**
pres_ind	averigu - o	averigu - as	averigu - a	averigu - amos	averigu - áis	averigu - an	
impf_ind	averigu - aba	averigu - abas	averigu - aba	averigu - ábamos	averigu - abais	averigu - aban	
pret_ind	averigü - é	averigu - aste	averigu - ó	averigu - amos	averigu - asteis	averigu - aron	
futuro	averigu - aré	averigu - arás	averigu - ará	averigu - aremos	averigu - aréis	averigu - arán	
pres_subj	averigü - e	averigü - es	averigü - e	averigü - emos	averigü - éis	averigü - en	
imp_subj	averigu - ara, ase	averigu - aras, ases	averigu - ara, ase	averigu - áramos, ásemos	averigu - arais, aseis	averigu - aran, asen	
cond	averigu - aría	averigu - arías	averigu - aría	averigu - aríamos	averigu - aríais	averigu - arían	
imper		averigu - a			averigu - ad		
infin							averigu - ar
ger							averigu - ando
part							averigu - ado

- **tipo_raíz**
  1. ***averigu***: 00 11 12 13 14 15 16 21 22 23 24 25 26 32 33 34 35 36 41 42 43 44 45 46 61 62 63 64 65 66 71 72 73 74 75 76 82 85 90 99
  2. ***averigü***: 31 51 52 53 54 55 56
- **tipo_des**
  1. ***reg***: para todas las formas

Otros verbos que pertenecen a este paradigma son:

abalanzar	*abanicar*	*abdicar*	*abocar*	*abarcar*
abrazar	*abroncar*	*acercar*	*acezar*	*achacar*
achicar	*acidificar*	*acorazar*	*actualizar*	*adamascar*
adelgazar	*aderezar*	*adjudicar*	*adverbializar*	*afianzar*
afincar	*africanizar*	*agilizar*	*agonizar*	*agudizar*
aguzar	*ahincar*	*ahorcar*	*ahuecar*	*alambicar*

alborozar
alfabetizar
alunizar
americanizar
amplificar
antipatizar
aplacar
apriscar
armonizar
atemorizar
atizar
automatizar
azuzar
becar
caducar
calzar
caramelizar
catequizar
centuplicar
chocar
clasificar
colectivizar
comunicar
contraatacar
cotizar
cruzar
damnificar
delegar
desacralizar
desautorizar
descalzar
descolocar
desembarazar
desempacar
desestancar
deshumanizar
desnacionalizar
despersonalizar
destacar
dislocar
dogmatizar
dulcificar
ejemplarizar
embarcar
emboscar
empacar
encarnizar
endulzar
enfoscar
enjuagar
ensalzar
entronizar
esbozar
especializar
estabilizar

alcanzar
allegar
alzar
amerizar
analizar
apagar
aplazar
arabizar
arrancar
atenazar
atomizar
autorizar
balizar
bonificar
cagar
canalizar
carbonizar
cauterizar
cercar
churrascar
claudicar
colocar
conculcar
contraindicar
cristalizar
cuadruplicar
danzar
demarcar
desamortizar
desbancar
descapitalizar
descortezar
desembarcar
desenfocar
desintoxicar
deslavazar
desnaturalizar
despiezar
diagnosticar
divagar
domesticar
economizar
ejemplificar
embarrancar
embozar
empalagar
encauzar
energizar
enfrascar
enlazar
entrechocar
enzarzar
escandalizar
especificar
estancar

alcoholizar
almohazar
amagar
amordazar
anarquizar
apalancar
aplicar
arborizar
arriscar
aterrizar
atracar
avanzar
barnizar
bostezar
calcar
canonizar
caricaturizar
cazar
certificar
cicatrizar
climatizar
colonizar
confraternizar
convocar
cristianizar
cualificar
dedicar
democratizar
desaplicar
desbocar
descentralizar
descuartizar
desembarrancar
desenlazar
desfalcar
deslizar
desnucar
desplazar
dializar
diversificar
dosificar
edificar
electrificar
embaucar
embragar
empalizar
encharcar
enfatizar
engarzar
enmarcar
entrelazar
equivocar
escenificar
esperanzar
estandardizar

alegar
alocar
amenazar
amortizar
anatematizar
aparcar
apocar
arcaizar
atacar
aterrorizar
atrancar
avezar
bautizar
brincar
calcificar
capitalizar
cascar
cegar
chamuscar
civilizar
cocar
comarcar
contabilizar
confiscar
criticar
cuantificar
defecar
densificar
desatascar
descalcificar
descocar
desecar
desembocar
desenroscar
desflecar
desmenuzar
desorganizar
despotricar
disecar
divinizar
dragar
editorializar
electrizar
embocar
embrazar
emporcar
encolerizar
enfervorizar
enjaezar
enriscar
entresacar
erizar
esclavizar
espiritualizar
estatificar

alegorizar
altercar
amenizar
amostazar
anegar
apelmazar
apologizar
aristocratizar
atascar
atezar
autentificar
axiomatizar
beatificar
buscar
calificar
caracterizar
castellanizar
centralizar
chascar
clarificar
codificar
comercializar
contemporizar
cosificar
crucificar
cubicar
deificar
derrocar
desatrancar
descalificar
descodificar
deseducar
desembragar
desestabilizar
desguazar
desmitificar
despedazar
desratizar
disfrazar
doblegar
dramatizar
educar
embarazar
emborrascar
embriagar
encabezar
enderezar
enfocar
enjalbegar
enroscar
entroncar
erradicar
españolizar
esquematizar
esterilizar

estigmatizar
estilizar
etimologizar
estomagar
estragar
estratificar
estucar
eternizar
exorcizar
explicar
exteriorizar
extranjerizar
europeizar
evangelizar
evocar
fabricar
falsificar
familiarizar
fecundizar
fertilizar
finalizar
fincar
fiscalizar
flexibilizar
fluidificar
formalizar
fornicar
fortificar
fraternizar
fructificar
galvanizar
garantizar
gargarizar
gasificar
generalizar
gentilizar
germanizar
glorificar
gongorizar
gozar
gratificar
halagar
hechizar
helenizar
higienizar
hincar
hiperbolizar
hipnotizar
hipotecar
hispanizar
hocicar
holgazar
homogeneizar
horrorizar
hospitalizar
hostilizar
hozar
humanizar
idealizar
identificar
impermeabilizar
imprecar
impurificar
incomunicar
inculcar
indagar
indemnizar
independizar
indicar
individualizar
indizar
industrializar
inmortalizar
inmovilizar
inmunizar
insensibilizar
insonorizar
intelectualizar
intensificar
internacionaliza
intoxicar
intranquilizar
intrincar
inutilizar
invocar
ionizar
ironizar
islamizar
italianizar
izar
jerarquizar
justificar
labializar
laicizar
lanzar
lateralizar
latinizar
lazar
legalizar
legar
lenificar
lexicalizar
liberalizar
liofilizar
lizar
llagar
llegar
localizar
lubricar
lubrificar
magnetizar
mancar
manducar
marcar
mariscar
martirizar
mascar
masticar
materializar
maternizar
matizar
maximizar
mazar
mecanizar
mediatizar
medicar
memorizar
mercantilizar
mercar
mercerizar
metaforizar
metalizar
metatizar
metodizar
militarizar
mineralizar
minimizar
mitificar
mistificar
mixtificar
modernizar
modificar
monopolizar
momificar
moralizar
mordiscar
mortificar
motorizar
movilizar
municipalizar
nacionalizar
narcotizar
nasalizar
naturalizar
naufragar
neutralizar
notificar
normalizar
obcecar
obstaculizar
oficializar
ofuscar
oliscar
opacar
optimizar
organizar
pacificar
paganizar
pagar
palatalizar
panificar
paralizar
particularizar
pasteurizar
patentizar
pecar
pellizcar
penalizar
perjudicar
personalizar
personificar
pescar
petrificar
picar
pinzar
pizcar
plagar
planificar
plantificar
platicar
pluralizar
perennizar
poetizar
polarizar
polemizar
politizar
pontificar
popularizar
pormenorizar
potabilizar
practicar
preconizar
predicar
presurizar
prevaricar
profesionalizar
profetizar
profundizar
propagar
pronosticar
prosificar
protagonizar
protocolizar
provocar
publicar
pulverizar
puntualizar
punzar
purificar
quintuplicar
racionalizar
radicalizar
radicar
rarificar
rascar
ratificar
realizar
realzar
rebozar
rebuscar
recalcar
rectificar
reedificar
reeducar
reemplazar
refrescar
regularizar
relegar
remolcar
remozar
reorganizar
repescar
repicar
replicar
resecar
responsabilizar
retocar
retozar
retrancar
revalorizar
revindicar
revocar
rezagar
rezar
ridiculizar
rivalizar
rizar
romanizar
roncar
roscar
rozar
ruborizar
rubricar
sacar
sacralizar
sacrificar
salificar
salpicar
santificar
saponificar
satirizar
secar
secularizar
sensibilizar
señalizar

septuplicar	sextuplicar	significar	silabizar	silogizar
simbolizar	simpatizar	simplificar	sincronizar	singularizar
sindicar	sintetizar	sintonizar	sistematizar	socializar
sofisticar	sofocar	solazar	solemnizar	solidarizar
solidificar	sollozar	sonorizar	sonsacar	suavizar
sufragar	suplicar	surcar	tabicar	tapizar
teatralizar	temporalizar	temporizar	teologizar	teorizar
testificar	tipificar	tiranizar	tocar	tonificar
totalizar	trabucar	traficar	tragar	tranquilizar
trapazar	traumatizar	trazar	trenzar	trincar
triplicar	triscar	trompicar	tronzar	trucar
truncar	ubicar	unificar	universalizar	urbanizar
utilizar	vagar	vaporizar	velarizar	verificar
versificar	vigorizar	vindicar	visualizar	vitrificar
vivificar	vocalizar	volatilizar	vulcanizar	vulgarizar

Modelo 3

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	222	222	222	111	111	222	
impf_ind	111	111	111	111	111	111	
pret_ind	333	111	111	111	111	111	
futuro	111	111	111	111	111	111	
pres_subj	444	444	444	333	333	444	
imp_subj	111	111	111	111	111	111	
cond	111	111	111	111	111	111	
imper		222			111		
infin							111
ger							111
part							111

Tabla 11. Tipo 3a: EMPEZAR, ENRAIZAR. Diptongación o cambio acentual y cambio gráfico en consonante (*-ezar, -aizar*)							
	sg_1	**sg_2**	**sg_3**	**pl_1**	**pl_2**	**pl_3**	**no**
pres_ind	empiez - o	empiez - as	empiez - a	empez - amos	empez - áis	empiez - an	
impf_ind	empez - aba	empez - abas	empez - aba	empez - ábamos	empez - abais	empez - aban	
pret_ind	empec - é	empez - aste	empez - ó	empez - amos	empez - asteis	empez - aron	
futuro	empez - aré	empez - arás	empez - ará	empez - aremos	empez - aréis	empez - arán	
pres_subj	empiec - e	empiec - es	empiec - e	empec - emos	empec - éis	empiec - en	
imp_subj	empez - ara, ase	empez - aras, ases	empez - ara, ase	empez - áramos, ásemos	empez - arais, aseis	empez - aran, asen	
cond	empez - aría	empez - arías	empez - aría	empez - aríamos	empez - aríais	empez - arían	
imper		empiez - a			empez - ad		
infin							empez - ar
ger							empez - ando
part							empez - ado

- **tipo_raíz**
  1. ***empez***: 00 14 15 21 22 23 24 25 26 32 33 34 35 36 41 42 43 44 45 46 61 62 63 64 65 66 71 72 73 74 75 76 85 90 99
  2. ***empiez***: 11 12 13 16 82
  3. ***empec***: 31 54 55
  4. ***empiec***: 51 52 53 56
- **tipo_des**
  1. ***reg***: para todas las formas

Tabla 12. Tipo 3b: NEGAR ROGAR COLGAR JUGAR. Diptongación y cambio gráfico en consonante y vocal (*-egar, -ogar, -olgar, -ugar*)							
	sg_1	**sg_2**	**sg_3**	**pl_1**	**pl_2**	**pl_3**	**no**
pres_ind	cuelg - o	cuelg - as	cuelg - a	colg - amos	colg - áis	cuelg - an	
impf_ind	colg - aba	colg - abas	colg - aba	colg - ábamos	colg - abais	colg - aban	
pret_ind	colgu - é	colg - aste	colg - ó	colg - amos	colg - asteis	colg - aron	
futuro	colg - aré	colg - arás	colg - ará	colg - aremos	colg - aréis	colg - arán	
pres_subj	cuelgu - e	cuelgu - es	cuelgu - e	colgu - emos	colgu - éis	cuelgu - en	
imp_subj	colg - ara, ase	colg - aras, ases	colg - ara, ase	colg - áramos, ásemos	colg - arais, aseis	colg - aran, asen	
cond	colg - aría	colg - arias	colg - aría	colg - aríamos	colg - aríais	colg - arían	
imper		cuelg - a			colg - ad		
infin							colg - ar
ger							colg - ando
part							colg - ado

- **tipo_raíz**
  1. ***colg***: 00 14 15 21 22 23 24 25 26 32 33 34 35 36 41 42 43 44 45 46 61 62 63 64 65 66 71 72 73 74 75 76 85 90 99
  2. ***cuelg***: 11 12 13 16 82
  3. ***colgu***: 31 54 55
  4. ***cuelgu***: 51 52 53 56
- **tipo_des**
  1. ***reg***: para todas las formas

Tabla 13. Tipo 3c: COCER. Diptongación y cambio gráfico en consonante (*-ocer*)							
	sg_1	**sg_2**	**sg_3**	**pl_1**	**pl_2**	**pl_3**	**no**
pres_ind	cuez - o	cuec - es	cuec - e	coc - emos	coc - éis	cuec - en	
impf_ind	coc - ía	coc - ías	coc - ía	coc - íamos	coc - íais	coc - ían	
pret_ind	coc - í	coc - iste	coc - ió	coc - imos	coc - isteis	coc - ieron	
futuro	coc - eré	coc - erás	coc - erá	coc - eremos	coc - eréis	coc - erán	
pres_subj	cuez - a	cuez - as	cuez - a	coz - amos	coz - áis	cuez - an	
imp_subj	coc - iera, iese	coc - ieras, ieses	coc - iera, iese	coc - íeramos, íesemos	coc - ierais, ieseis	coc - ieran, iesen	
cond	coc - ería	coc - erías	coc - ería	coc - eríamos	coc - eríais	coc - erían	
imper		cuec - e			coc - ed		
infin							coc - er
ger							coc - iendo
part							coc - ido

- **tipo_raíz**
  1. *coc*: 00 14 15 21 22 23 24 25 26 31 32 33 34 35 36 41 42 43 44 45 46 61 62 63 64 65 66 71 72 73 74 75 76 85 90 99
  2. *cuec*: 12 13 16 82
  3. *coz*: 54 55
  4. *cuez*: 11 51 52 53 56
- **tipo_des**
  1. *reg*: para todas las formas

En este grupo hay que hacer varios comentarios. En primer lugar, destacar la inclusión del verbo *jugar* dentro del subtipo B. *Jugar* es un verbo irregular aislado en las clasificaciones fonológicas: el único caso de diptongación *u* → *ue*, porque no tiene compuestos, ya que no lo son *conjugar, enjugar* ni *desjugar*, los tres regulares por ser de distintas procedencias.

En segundo lugar la presencia de *cocer* en un subgrupo diferente a *empezar*, cuando ambos tienen las mismas irregularidades (diptongación y cambio de *c* por *z*), se justifica porque no comparten exactamente la misma distribución de formas. *Cocer* no cambia la raíz básica (*coc-*) en la primera persona del singular del pretérito simple de indicativo (a diferencia de *empezar*, que usa *empec-*). Igualmente, en la primera del singular del presente de indicativo (código 11), *empezar* comparte raíz con las otras

formas fuertes del presente de indicativo y el imperativo (*empiez-*), mientras que *cocer* lo hace con las del presente de subjuntivo (*cuez-*). Evidentemente, esta ligera variación distribucional se debe al hecho de que pertenecen a distintas conjugaciones, con lo que la presencia de la *-e* en primera posición de la desinencia varía. De cualquier forma, podemos considerar estos dos grupos de verbos dentro del mismo modelo.

Otros verbos de este paradigma son:

almorzar	*avergonzar*	*cegar*	*colgar*	*comenzar*
denegar	*desasosegar*	*desplegar*	*empezar*	*esforzar*
forzar	*fregar*	*holgar*	*negar*	*plegar*
reforzar	*refregar*	*regar*	*renegar*	*replegar*
restregar	*retorcer*	*revolcar*	*rogar*	*segar*
sosegar	*torcer*	*trasegar*	*trastocar*	*trocar*
tropezar	*volcar*			

Modelo 4

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	222	111	111	111	111	111	
impf_ind	111	111	111	111	111	111	
pret_ind	111	111	111	111	111	111	
futuro	111	111	111	111	111	111	
pres_subj	222	222	222	222	222	222	
imp_subj	111	111	111	111	111	111	
cond	111	111	111	111	111	111	
imper		111			111		
infin							111
ger							111
part							111

Tabla 14. Tipo 4a: OFRECER CONOCER NACER LUCIR. Consonante epentética "z" (*-ecer, -ocer, -acer, -cir*)							
	sg_1	**sg_2**	**sg_3**	**pl_1**	**pl_2**	**pl_3**	**no**
pres_ind	conozc - o	conoc - es	conoc - e	conoc - emos	conoc - éis	conoc - en	
impf_ind	conoc - ía	conoc - ías	conoc - ía	conoc - íamos	conoc - íais	conoc - ían	
pret_ind	conoc - í	conoc - iste	conoc - ió	conoc - imos	conoc - isteis	conoc - ieron	
futuro	conoc - eré	conoc - erás	conoc - erá	conoc - eremos	conoc - eréis	conoc - erán	
pres_subj	conozc - a	conozc - as	conozc - a	conozc - amos	conozc - áis	conozc - an	
imp_subj	conoc - iera, iese	conoc - ieras, ieses	conoc - iera, iese	conoc - ieramos, iésemos	conoc - ierais, ieseis	conoc - ieran, iesen	
cond	conoc - ería	conoc - erías	conoc - ería	conoc - eríamos	conoc - eríais	conoc - erían	
imper		conoc - e			conoc - ed		
infin							conoc - er
ger							conoc - iendo
part							conoc - ido

- **tipo_raíz**
  1. ***conoc***: 00 12 13 14 15 16 21 22 23 24 25 26 31 32 33 34 35 36 41 42 43 44 45 46 61 62 63 64 65 66 71 72 73 74 75 76 82 85 90 99
  2. ***conozc***: 11 51 52 53 54 55 56
- **tipo_des**
  1. ***reg***: para todas las formas

Tabla 15. Tipo 4b: ASIR. Consonante epentética "g" (*-asir*)							
	sg_1	**sg_2**	**sg_3**	**pl_1**	**pl_2**	**pl_3**	**no**
pres_ind	asg - o	as - es	as - e	as - imos	as - ís	as - en	
impf_ind	as - ía	as - ías	as - ía	as - íamos	as - íais	as - ían	
pret_ind	as - í	as - iste	as - ió	as - imos	as - isteis	as - ieron	
futuro	as - iré	as - irás	as - irá	as - iremos	as - iréis	as - irán	
pres_subj	asg - a	asg - as	asg - a	asg - amos	asg - áis	asg - an	
imp_subj	as - iera, iese	as - ieras, ieses	as - iera, iese	as - ieramos, iesemos	as - ierais, ieseis	as - ieran, iesen	
cond	as - iría	as - irías	as - iría	as - iríamos	as - iríais	as - irían	
imper		as - e			as - id		
infin							as - ir
ger							as - iendo
part							as - ido

- **tipo_raíz**
  1. ***as***: 00 12 13 14 15 16 21 22 23 24 25 26 31 32 33 34 35 36 41 42 43 44 45 46 61 62 63 64 65 66 71 72 73 74 75 76 82 85 90 99
  2. ***asg***: 11 51 52 53 54 55 56
- **tipo_des**
  1. ***reg***: para todas las formas

Otros verbos que pertenecen a este paradigma son

aborrecer	*acrecer*	*adolecer*	*adormecer*	*agradecer*
amortecer	*aparecer*	*apetecer*	*balbucir*	*compadecer*
comparecer	*complacer*	*conocer*	*convalecer*	*crecer*
decrecer	*desabastecer*	*desadormecer*	*desaparecer*	*desconocer*
desentumecer	*desfallecer*	*deslucir*	*desmerecer*	*desobedecer*
desvanecer	*embrutecer*	*empalidecer*	*empequeñecer*	*emplastecer*
emprobecer	*enaltecer*	*enardecer*	*encalvecer*	*encallecer*
encanecer	*encarecer*	*engrandecer*	*enlanguidecer*	*enloquecer*

enmohecer	enmudecer	ennegrecer	ennoblecer	enorgullecer
enrarecer	enriquecer	enrojecer	enronquecer	ensoberbecerse
ensombrecerse	ensordecer	entenebrecer	enternecer	entontecer
entorpecer	entrelucir	entristecer	entumecer	envanecer
envejecer	enverdecer	envilecer	escarnecer	esclarecer
establecer	estremecer	fallecer	favorecer	fenecer
fortalecer	fosforecer	guarecer	guarnecer	humedecer
languidecer	lucir	merecer	mohecer	nacer
obedecer	obscurecer	ofrecer	orgullecer	oscurecer
pacer	padecer	palidecer	parecer	perecer
permanecer	pertenecer	podrecer	prevalecer	reaparecer
reblandecer	reconocer	recrudecer	reflorecer	rejuvenecer
relentecer	relucir	renacer	resplandecer	restablecer
reverdecer	robustecer	traslucir	verdecer	

Modelo 5

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	222	555	555	111	111	555	
impf_ind	111	111	111	111	111	111	
pret_ind	333	333	333	333	333	333	
futuro	444	444	444	444	444	444	
pres_subj	222	222	222	222	222	222	
imp_subj	333	333	333	333	333	333	
cond	444	444	444	444	444	444	
imper		666			111		
infin							111
ger							111/333/555
part							111/777

Tabla 16. Tipo 5a: TENER. Todo tipo de modificaciones en la raíz y la desinencia							
	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	teng - o	tien - es	tien - e	ten - emos	ten - éis	tien - en	
impf_ind	ten - ía	ten - ías	ten - ía	ten - íamos	ten - íais	ten - ían	
pret_ind	tuv - e	tuv - iste	tuv - o	tuv - imos	tuv - isteis	tuv - ieron	
futuro	tend - ré	tend - rás	tend - rá	tend - remos	tend - réis	tend - rán	
pres_subj	teng - a	teng - as	teng - a	teng - amos	teng - áis	teng - an	
imp_subj	tuv - iera, iese	tuv - ieras, ieses	tuv - iera, iese	tuv - ieramos, iesemos	tuv - ierais, ieseis	tuv - ieran, iesen	
cond	tend - ría	tend - rías	tend - ría	tend - ríamos	tend - ríais	tend - rían	
imper		ten			ten - ed		
infin							ten - er
ger							ten - iendo
part							ten - ido

- **tipo_raíz**
  1. ***ten***: 00 14 15 21 22 23 24 25 26 85 90 99
  2. ***teng***: 11 51 52 53 54 55 56
  3. ***tuv***: 31 32 33 34 35 36 61 62 63 64 65 66
  4. ***tend***: 41 42 43 44 45 46 71 72 73 74 75 76
  5. ***tien***: 12 13 16
  6. ***ten***: 82
- **tipo_des**
  1. ***reg***: 00 11 12 13 14 15 16 21 22 23 24 25 26 32 34 35 36 51 52 53 54 55 56 61 62 63 64 65 66 85 90 99
  2. ***pret1***: 31 33
  3. ***fut_cond***: 41 42 43 44 45 46 71 72 73 74 75 76

Tabla 17. Tipo 5b: VENIR. Todo tipo de modificaciones en la raíz y la desinencia							
	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	veng - o	vien - es	vien - e	ven - imos	ven - ís	vien - en	
impf_ind	ven - ía	ven - ías	ven - ía	ven - íamos	ven - íais	ven - ían	
pret_ind	vin - e	vin - iste	vin - o	vin - imos	vin - isteis	vin - ieron	
futuro	vend - ré	vend - rás	vend - rá	vend - remos	vend - réis	vend - rán	
pres_subj	veng - a	veng - as	veng - a	veng - amos	veng - áis	veng - an	
imp_subj	vin - iera, iese	vin - ieras, ieses	vin - iera, iese	vin - iéramos, iésemos	vin - ierais, ieseis	vin - ieran, iesen	
cond	vend - ría	vend - rías	vend - ría	vend - ríamos	vend - ríais	vend - rían	
imper		ven			ven - id		
infin							ven - ir
ger							vin - iendo
part							ven - ido

- **tipo_raíz**
  1. ***ven***: 00 14 15 21 22 23 24 25 26 85 99
  2. ***veng***: 11 51 52 53 54 55 56
  3. ***vin***: 31 32 33 34 35 36 61 62 63 64 65 66 90
  4. ***vend***: 41 42 43 44 45 46 71 72 73 74 75 76
  5. ***vien***: 12 13 16
  6. ***ven***: 82
- **tipo_des**
  1. ***reg***: 00 11 12 13 14 15 16 21 22 23 24 25 26 32 34 35 36 51 52 53 54 55 56 61 62 63 64 65 66 85 90 99
  2. ***pret1***: 31 33
  3. ***fut_cond***: 41 42 43 44 45 46 71 72 73 74 75 76

Tabla 18. Tipo 5c: DECIR. Todo tipo de modificaciones en la raíz y la desinencia							
	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	dig - o	dic - es	dic - e	dec - imos	dec - ís	dic - en	
impf_ind	dec - ía	dec - ías	dec - ía	dec - íamos	dec - íais	dec - ían	
pret_ind	dij - e	dij - iste	dij - o	dij - imos	dij - isteis	dij - eron	
futuro	di - ré	di - rás	di - rá	di - remos	di - réis	di - rán	
pres_subj	dig - a	dig - as	dig - a	dig - amos	dig - áis	dig - an	
imp_subj	dij - era, ese	dij - eras, eses	dij - era, ese	dij - éramos, ésemos	dij - erais, eseis	dij - eran, esen	
cond	di - ría	di - rías	di - ría	di - ríamos	di - ríais	di - rían	
imper		di			dec - id		
infin							dec - ir
ger							dic - iendo
part							dich - o

- **tipo_raíz**
  1. ***dec***: 00 14 15 21 22 23 24 25 26 85
  2. ***dig***: 11 51 52 53 54 55 56
  3. ***dij***: 31 32 33 34 35 36 61 62 63 64 65 66
  4. ***di***: 41 42 43 44 45 46 71 72 73 74 75 76
  5. ***dic***: 12 13 16 90
  6. ***di***: 82
  7. ***dich***: 99
- **tipo_des**
  1. ***reg***: 00 11 12 13 14 15 16 21 22 23 24 25 26 32 34 35 36 51 52 53 54 55 56 85 90
  2. ***pret1***: 31 33
  3. ***pret2***: 36
  4. ***fut_cond***: 41 42 43 44 45 46 71 72 73 74 75 76
  5. ***imp_subj***: 61 62 63 64 65 66
  6. ***part1***: 99

Este subgrupo **5c** tiene una variante especial: la desinencia de la tercera del plural del pret. de indicativo (36) es igual a la de los verbos de los modelos 8, 9, 10 y 12: *cayeron, huyeron, rieron* y *leyeron*. También presenta un participio irregular.

Tabla 19. Tipo 5d: PONER. Todo tipo de modificaciones en la raíz y la desinencia

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	pong - o	pon - es	pon - e	pon - emos	pon - éis	pon - en	
impf_ind	pon - ía	pon - ías	pon - ía	pon - íamos	pon - íais	pon - ían	
pret_ind	pus - e	pus - iste	pus - o	pus - imos	pus - isteis	pus - ieron	
futuro	pond - ré	pond - rás	pond - rá	pond - remos	pond - réis	pond - rán	
pres_subj	pong - a	pong - as	pong - a	pong - amos	pong - áis	pong - an	
imp_subj	pus - iera, iese	pus - ieras, ieses	pus - iera, iese	pus - ieramos, iesemos	pus - ierais, ieseis	pus - ieran, iesen	
cond	pond - ría	pond - rías	pond - ría	pond - ríamos	pond - ríais	pond - rían	
imper		pon			pon - ed		
infin							pon - er
ger							pon - iendo
part							puest - o

- **tipo_raíz**
  1. ***pon***: 00 12 13 14 15 16 21 22 23 24 25 26 85 90
  2. ***pong***: 11 51 52 53 54 55 56
  3. ***pus***: 31 32 33 34 35 36 61 62 63 64 65 66
  4. ***pond***: 41 42 43 44 45 46 71 72 73 74 75 76
  5. ***pon***: 82
  6. ***puest***: 99
- **tipo_des**
  1. ***reg***: 00 11 12 13 14 15 16 21 22 23 24 25 26 32 34 35 36 51 52 53 54 55 56 61 62 63 64 65 66 90
  2. ***pret1***: 31 33
  3. ***fut_cond***: 41 42 43 44 45 46 71 72 73 74 75 76
  4. ***part1***: 99

Variantes del subgrupo **5d**:

- 12 13 y 16 llevan la raíz regular (como HACER),
- raíz especial para el participio

Tabla 20. Tipo 5e: HACER. Todo tipo de modificaciones en la raíz y la desinencia							
	sg_1	**sg_2**	**sg_3**	**pl_1**	**pl_2**	**pl_3**	**no**
pres_ind	hag - o	hac - es	hac - e	hac - emos	hac - éis	hac - en	
impf_ind	hac - ía	hac - ías	hac - ía	hac - íamos	hac - íais	hac - ían	
pret_ind	hic - e	hic - iste	hiz - o	hic - imos	hic - isteis	hic - ieron	
futuro	ha - ré	ha - rás	ha - rá	ha - remos	ha - réis	ha - rán	
pres_subj	hag - a	hag - as	hag - a	hag - amos	hag - áis	hag - an	
imp_subj	hic - iera, iese	hic - ieras, ieses	hic - iera, iese	hic - íeramos, íesemos	hic - ierais, ieseis	hic - ieran, iesen	
cond	ha - ría	ha - rías	ha - ría	ha - ríamos	ha - ríais	ha - rían	
imper		haz			hac - ed		
infin							hac - er
ger							hac - iendo
part							hech - o

- **tipo_raíz**
  1. ***hac***: 00 12 13 14 15 16 21 22 23 24 25 26 85 90
  2. ***hag***: 11 51 52 53 54 55 56
  3. ***hic***: 31 32 34 35 36 61 62 63 64 65 66
  4. ***ha***: 41 42 43 44 45 46 71 72 73 74 75 76
  5. ***hiz***: 33
  6. ***haz***: 82
  7. ***hech***: 99
- **tipo_des**
  1. ***reg***: 00 11 12 13 14 15 16 21 22 23 24 25 26 32 34 35 36 51 52 53 54 55 56 61 62 63 64 65 66 90
  2. ***pret1***: 31 33
  3. ***fut_cond***: 41 42 43 44 45 46 71 72 73 74 75 76
  4. ***part1***: 99

Variantes:

- 12 13 y 16 están dentro de la raíz regular (como PONER),
- 33 está aislado,
- raíz especial para el participio

Los verbos que se ajustan a este modelo son:

abstenerse	*antedecir*	*anteponer*	*atenerse*	*avenir*
bendecir	*componer*	*contener*	*contradecir*	*contrahacer*
contraponer	*contravenir*	*convenir*	*decir*	*deponer*
desavenirse	*descomponer*	*desconvenir*	*desdecir*	*deshacer*
detener	*disconvenir*	*disponer*	*detener*	*devenir*
entretener	*exponer*	*hacer*	*imponer*	*interponer*
indisponer	*interdecir*	*intervenir*	*maldecir*	*mantener*
obtener	*oponer*	*poner*	*posponer*	*predecir*
predisponer	*preponer*	*presuponer*	*prevenir*	*proponer*
provenir	*recomponer*	*reconvenir*	*rehacer*	*retener*
reponer	*revenir*	*sobreponer*	*sobrevenir*	*sostener*
subvenir	*superponer*	*supervenir*	*suponer*	*tener*
transponer	*trasponer*	*venir*	*yuxtaponer*	

Modelo 6

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	222	222	222	111	111	222	
impf_ind	111	111	111	111	111	111	
pret_ind	333	333	333	333	333	333	
futuro	111	111	111	111	111	111	
pres_subj	222	222	222	111	111	222	
imp_subj	333	333	333	333	333	333	
cond	111	111	111	111	111	111	
imper		222			111		
infin							111
ger							111/333
part							111

Tabla 21. Tipo 6: QUERER PODER. Cambios vocálicos y consonánticos en las formas fuertes (presentes, pret. de ind., imperf. del subj, e imperativo)							
	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	quier - o	quier - es	quier - e	quer - emos	quer - éis	quier - en	
impf_ind	quer - ía	quer - ías	quer - ía	quer - íamos	quer - íais	quer - ían	
pret_ind	quis - e	quis - iste	quis - o	quis - imos	quis - isteis	quis - ieron	
futuro	quer - ré	quer - rás	quer - rá	quer - remos	quer - réis	quer - rán	
pres_subj	quier - a	quier - as	quier - a	quer - amos	quer - áis	quier - an	
imp_subj	quis - iera, iese	quis - ieras, ieses	quis - iera, iese	quis - iéramos, iésemos	quis - ierais, ieseis	quis - ieran, iesen	
cond	quer - ría	quer - rías	quer - ría	quer - ríamos	quer - ríais	quer - rían	
imper		quier - e			quer - ed		
infin							quer - er
ger							quer - iendo
part							quer - ido

- **tipo_raíz**
  1. ***quer***: 00 14 15 21 22 23 24 25 26 41 42 43 44 45 46 54 55 71 72 73 74 75 76 85 90 99
  2. ***quier***: 11 12 13 16 51 52 53 56 82
  3. ***quis***: 31 32 33 34 35 36 61 62 63 64 65 66
- **tipo_des**
  1. ***reg***: 00 11 12 13 14 15 16 21 22 23 24 25 26 32 34 35 36 51 52 53 54 55 56 61 62 63 64 65 66 90 99
  2. ***pret1***: 31 33
  3. ***fut_cond***: 41 42 43 44 45 46 71 72 73 74 75 76

Las variantes son:

- en los derivados de QUERER, la diptongación *e* → *ie*,
- en PODER, la diptongación *o* → *ue*,
- además, PODER tiene una variante radical en el gerundio, que es la misma que la del pretérido fuerte (*pud - iendo*)

Verbos de este modelo son:

bienquerer *desquerer* *malquerer* *querer* *poder*

Modelo 7

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	222	111	111	111	111	111	
impf_ind	111	111	111	111	111	111	
pret_ind	333	333	333	333	333	333	
futuro	111	111	111	111	111	111	
pres_subj	222	222	222	222	222	222	
imp_subj	333	333	333	333	333	333	
cond	111	111	111	111	111	111	
imper		111			111		
infin							111
ger							111
part							111

Tabla 22. Tipo 7a: CONDUCIR. Cambios consonánticos en las formas fuertes (presente del subj., pret. de ind., imperf. del subj, e imperativo)

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	conduzc - o	conduc - es	conduc - e	conduc - imos	conduc - ís	conduc - en	
impf_ind	conduc - ía	conduc - ías	conduc - ía	conduc - íamos	conduc - íais	conduc - ían	
pret_ind	conduj - e	conduj - iste	conduj - o	conduj - imos	conduj - isteis	conduj - eron	
futuro	conduc - iré	conduc - irás	conduc - irá	conduc iremos	conduc - iréis	conduc - irán	
pres_subj	conduzc - a	conduzc - as	conduzc - a	conduzc - amos	conduzc - áis	conduzc - an	
imp_subj	conduj - era, ese	conduj - eras, eses	conduj - era, ese	conduj éramos, ésemos	conduj - erais, eseis	conduj - eran, esen	
cond	conduc - iría	conduc - irías	conduc - iría	conduc - iríamos	conduc - iríais	conduc - irían	
imper		conduc - e			conduc - id		
infin							conduc - ir
ger							conduc - iendo
part							conduc - ido

- **tipo_raíz**
  1. ***conduc***: 00 12 13 14 15 16 21 22 23 24 25 26 41 42 43 44 45 46 71 72 73 74 75 76 82 85 90 99
  2. ***conduzc***: 11 51 52 53 54 55 56
  3. ***conduj***: 31 32 33 34 35 36 61 62 63 64 65 66
- **tipo_des**
  1. ***reg***: 00 11 12 13 14 15 16 21 22 23 24 25 26 32 34 35 41 42 43 44 45 46 71 72 73 74 75 76 90 99
  2. ***pret1***: 31 33
  3. ***pret2***: 36
  4. ***imp_subj***: 61 62 63 64 65 66

Tabla 23. Tipo 7b: TRAER. Cambios consonánticos y vocálicos en las formas fuertes (presente del subj., pret. de ind., imperf. del subj, e imperativo)

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	traig - o	tra - es	tra - e	tra - emos	tra - éis	tra - en	
impf_ind	tra - ía	tra - ías	tra - ía	tra - íamos	tra - íais	tra - ían	
pret_ind	traj - e	traj - iste	traj - o	traj - imos	traj - isteis	traj - eron	
futuro	tra - eré	tra - erás	tra - erá	tra - eremos	tra - eréis	tra - erán	
pres_subj	traig - a	traig - as	traig - a	traig - amos	traig - áis	traig - an	
imp_subj	traj - era, ese	traj - eras, eses	traj - era, ese	traj - eramos, esemos	traj - erais, eseis	traj - eran, esen	
cond	tra - ería	tra - erías	tra - ería	tra - eríamos	tra - eríais	tra - erían	
imper		tra - e			tra - ed		
infin							tra - er
ger							tray - endo
part							tra - ído

- **tipo_raíz**
  1. ***tra***: 00 12 13 14 15 16 21 22 23 24 25 26 41 42 43 44 45 46 71 72 73 74 75 76 82 85 99
  2. ***traig***: 11 51 52 53 54 55 56
  3. ***traj***: 31 32 33 34 35 36 61 62 63 64 65 66
  4. ***tray***: 90

- **tipo_des**

  1. ***reg***: 00 11 12 13 14 15 16 21 22 23 24 25 26 32 34 35 41 42 43 44 45 46 71 72 73 74 75 76
  2. ***pret1***: 31 33
  3. ***pret2***: 36
  4. ***imp_subj***: 61 62 63 64 65 66
  5. ***ger***: 90
  6. ***part2***: 99

TRAER y sus derivados presentan la misma distribución que los verbos en -UCIR, con la única excepción del gerundio. Como existe el alomorfo *-endo* para el gerundio de verbos como CAER, LEER, REÑIR, etc., creamos una entrada especial para la variante de la raíz (ver explicación en 6.1.1.3, Irregularidades puras). Otra posibilidad es lexicalizar la forma, pero de todas maneras no evitaríamos incluir una entrada de diccionario especial.

Tabla 24. Tipo 7c: SABER. Cambios consonánticos y vocálicos en las formas fuertes (presente del subj., pret. de ind., imperf. del subj, e imperativo)

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	sé	sab - es	sab - e	sab - emos	sab - éis	sab - en	
impf_ind	sab - ía	sab - ías	sab - ía	sab - íamos	sab - íais	sab - ían	
pret_ind	sup - e	sup - iste	sup - o	sup - imos	sup - isteis	sup - ieron	
futuro	sab - ré	sab - rás	sab - rá	sab - remos	sab - réis	sab - rán	
pres_subj	sep - a	sep - as	sep - a	sep - amos	sep - áis	sep - an	
imp_subj	sup - iera, iese	sup - ieras, ieses	sup - iera, iese	sup - iéramos, iésemos	sup - ierais, ieseis	sup - ieran, iesen	
cond	sab - ría	sab - rías	sab - ría	sab - ríamos	sab - ríais	sab - rían	
imper		sab - e			sab - ed		
infin							sab - er
ger							sab - iendo
part							sab - ido

- **tipo_raíz**

  1. ***sab***: 00 12 13 14 15 16 21 22 23 24 25 26 41 42 43 44 45 46 71 72 73 74 75 76 82 85 90
  2. ***sep***: 51 52 53 54 55 56

3. ***sup***: 31 32 33 34 35 36 61 62 63 64 65 66
4. ***sé***: 11

- **tipo_des**
  1. ***reg***: 00 12 13 14 15 16 21 22 23 24 25 26 32 34 35 36 82 85 90 99
  2. ***pret1***: 31 33
  3. ***fut_cond***: 41 42 43 44 45 46 71 72 73 74 75 76

SABER y su derivado *resaber* tiene como particularidad la lexicalización de la primera persona del singular del presente de indicativo. Por lo demás, presenta la misma distribución que los verbos en -UCIR.

Resumiendo, los verbos que se incluyen en este paradigma son:

abstraer	*aducir*	*bajotraer*	*balbucir*	*bistraer*
conducir	*contraer*	*distraer*	*deducir*	*dentrotraer*
desatraer	*deslucir*	*educir*	*enlucir*	*entrelucir*
extraer	*inducir*	*introducir*	*lucir*	*maltraer*
prelucir	*producir*	*reconducir*	*reducir*	*relucir*
reproducir	*resaber*	*retraducir*	*retrotraer*	*saber*
seducir	*substraer*	*sustraer*	*traducir*	*traer*

Modelo 8

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	222	222	222	111	111	222	
impf_ind	111	111	111	111	111	111	
pret_ind	111	111	222	111	111	222	
futuro	111	111	111	111	111	111	
pres_subj	222	222	222	222	222	222	
imp_subj	222	222	222	222	222	222	
cond	111	111	111	111	111	111	
imper		222			111		
infin							111
ger							222
part							111

Tabla 25. Tipo 8a: HUIR. Cambios consonánticos y vocálicos en las formas fuertes (presente del subj., pret. de ind., imperf. del subj, e imperativo)							
	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	huy - o	huy - es	huy - e	hu - imos	hu - ís	huy - en	
impf_ind	hu - ía	hu - ías	hu - ía	hu - íamos	hu - íais	hu - ían	
pret_ind	hu - í	hu - iste	huy - ó	hu - imos	hu - isteis	huy - eron	
futuro	hu - iré	hu - irás	hu - irá	hu - iremos	hu - iréis	hu - irán	
pres_subj	huy - a	huy - as	huy - a	huy - amos	huy - áis	huy - an	
imp_subj	huy - era, ese	huy - eras, eses	huy - era, ese	huy - éramos, ésemos	huy - erais, eseis	huy - eran, esen	
cond	hu - iría	hu - irías	hu - iría	hu - iríamos	hu - iríais	hu - irían	
imper		huy - e			hu - íd		
infin							hu - ir
ger							huy - endo
part							hu - ido

- **tipo_raíz**
  1. ***hu***: 00 14 15 21 22 23 24 25 26 31 32 34 35 41 42 43 44 45 46 71 72 73 74 75 76 85 99
  2. ***huy***: 11 12 13 16 33 36 51 52 53 54 55 56 61 62 63 64 65 66 82 90
- **tipo_des**
  1. ***reg***: 00 12 13 14 15 16 21 22 23 24 25 26 31 32 34 35 41 42 43 44 45 46 51 52 53 54 55 56 71 72 73 74 75 76 82 85 99
  2. ***pret2***: 33 36
  3. ***imp_subj***: 61 62 63 64 65 66
  4. ***ger***: 90

El rasgo destacado de este modelo es que hemos considerado la consonante *y* en las formas 33 36 y 90 como parte integrante de la raíz, en lugar de tratarla como un alomorfo de la desinencia. Esta elección se justifica por el hecho de ser *huy-* una variante del radical en otras formas y, paralelamente, por la existencia de desinencias en *-eron, -era, -ese, -endo*, etc. para otros tipos de verbos (por ejemplo, *traj- eron, dij- era, cay- endo*). Sería menos rentable crear nuevas desinencias para *-yó, -yeron, -yera, -yendo*, cuando de cualquier manera tenemos las otras y se pueden usar sin ningún coste adicional. Para otros argumentos a favor de esta solución, véasen los *Criterios de segmentación* en la sección 6.1.1.2

Tabla 26. Tipo 8b: CEÑIR. Cambio vocálico en las formas fuertes (presentes, pret. de ind., imperf. del subj, e imperativo)							
	sg_1	**sg_2**	**sg_3**	**pl_1**	**pl_2**	**pl_3**	**no**
pres_ind	ciñ - o	ciñ - es	ciñ - e	ceñ - imos	ceñ - ís	ciñ - en	
impf_ind	ceñ - ía	ceñ - ías	ceñ - ía	ceñ - íamos	ceñ - íais	ceñ - ían	
pret_ind	ceñ - í	ceñ - iste	ciñ - ó	ceñ - imos	ceñ - isteis	ciñ - eron	
futuro	ceñ - iré	ceñ - irás	ceñ - irá	ceñ - iremos	ceñ - iréis	ceñ - irán	
pres_subj	ciñ - a	ciñ - as	ciñ - a	ciñ - amos	ciñ - áis	ciñ - an	
imp_subj	ciñ - era, ese	ciñ - eras, eses	ciñ - era, ese	ciñ - éramos, ésemos	ciñ - erais, eseis	ciñ - eran, esen	
cond	ceñ - iría	ceñ - irías	ceñ - iría	ceñ - iríamos	ceñ- iríais	ceñ - irían	
imper		ciñ - e			ceñ - id		
infin							ceñ - ir
ger							ciñ - endo
part							ceñ - ido

- **tipo_raíz**
  1. ***ceñ***: 00 14 15 21 22 23 24 25 26 31 32 34 35 41 42 43 44 45 46 71 72 73 74 75 76 85 99
  2. ***ciñ***: 11 12 13 16 33 36 51 52 53 54 55 56 61 62 63 64 65 66 82 90
- **tipo_des**
  1. ***reg***: 00 11 12 13 14 15 16 21 22 23 24 25 26 31 32 34 35 41 42 43 44 45 46 51 52 53 54 55 56 71 72 73 74 75 76 82 85 99
  2. ***pret2***: 33 36
  3. ***imp_subj***: 61 62 63 64 65 66
  4. ***ger***: 90

El subgrupo de verbos en *-ÑIR* presenta exactamente la misma distribución de raíces y desinencias que los verbos en *-UIR*, lo que nos aporta un argumento, desde el punto de vista formal, de que la *y* de *huy-ó* puede tratarse como una consonante que se ha añadido a la raíz, en lugar de un alomorfo de la desinencia. La equivalencia distribucional en dos grupos de verbos es una prueba de bastante peso en una clasificación de este tipo a la hora de elegir entre una segmentación u otra.

Tabla 27. Tipo 8c: PEDIR. Cambio vocálico en las formas fuertes (presentes, pret. de ind., imperf. del subj, e imperativo)							
	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	pid - o	pid - es	pid - e	ped - imos	ped - ís	pid - en	
impf_ind	ped - ía	ped - ías	ped - ía	ped - íamos	ped - íais	ped - ían	
pret_ind	ped - í	ped - iste	pid - ió	ped - imos	ped - isteis	ped - ieron	
futuro	ped - iré	ped - irás	ped - irá	ped - iremos	ped - iréis	ped - irán	
pres_subj	pid - a	pid - as	pid - a	pid - amos	pid - áis	pid - an	
imp_subj	pid - iera, iese	pid - ieras, ieses	pid - iera, iese	pid - iéramos, iésemos	pid - ierais, ieseis	pid - ieran, iesen	
cond	ped - iría	ped - irías	ped - iría	ped - iríamos	ped - iríais	ped - irían	
imper		pid - e			ped - id		
infin							ped - ir
ger							pid - endo
part							ped - ido

- **tipo_raíz**
  1. ***ped***: 00 14 15 21 22 23 24 25 26 31 32 34 35 41 42 43 44 45 46 71 72 73 74 75 76 82 85 99
  2. ***pid***: 11 12 13 16 33 36 51 52 53 54 55 56 61 62 63 64 65 66 82 90
- **tipo_des**
  1. ***reg***: todas las formas

Este subgrupo se caracteriza por no tener ninguna desinencia irregular. A pesar de este fuerte contraste con los verbos en -UIR, -NIR los consideramos dentro del mismo modelo porque la distribución de las irregularidades radicales es exactamente la misma.

La lista de verbos del modelo 8 es la siguiente:

afluir	*argüir*	*atribuir*	*ceñir*	*circuir*
colegir	*comedir*	*competir*	*concebir*	*concluir*
confluir	*conseguir*	*constituir*	*constreñir*	*construir*
contribuir	*corregir*	*derretir*	*derruir*	*desceñir*
descomedir	*desmedir*	*despedir*	*desteñir*	*destituir*
destruir	*desvestir*	*diluir*	*disminuir*	*distribuir*
elegir	*embestir*	*estatuir*	*estreñir*	*excluir*
expedir	*fluir*	*fruir*	*gemir*	*gruir*
henchir	*heñir*	*huir*	*imbuir*	*impedir*
influir	*incluir*	*inmiscuir*	*instituir*	*instruir*
intuir	*investir*	*irruir*	*medir*	*obstruir*
pedir	*perseguir*	*proseguir*	*prostituir*	*recluir*
recolegir	*reconstruir*	*reelegir*	*reexpedir*	*refluir*
regir	*rehuir*	*remedir*	*rendir*	*reñir*
repetir	*reseguir*	*restituir*	*reteñir*	*retribuir*
revestir	*seguir*	*sobrevestir*	*subseguir*	*sustituir*
teñir	*travestir*	*vestir*		

Modelo 9

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	222	222	222	111	111	222	
impf_ind	111	111	111	111	111	111	
pret_ind	111	111	333	111	111	333	
futuro	111	111	111	111	111	111	
pres_subj	222	222	222	333	333	222	
imp_subj	333	333	333	333	333	333	
cond	111	111	111	111	111	111	
imper		222			111		
infin							111
ger							333
part							111

Tabla 28. Tipo 9a: SENTIR ADVERTIR DORMIR MORIR. Cambios vocálicos en las formas fuertes (diptongación y cerramiento vocálico, esquemas *-e-ir, -o-ir*)							
	sg_1	**sg_2**	**sg_3**	**pl_1**	**pl_2**	**pl_3**	**no**
pres_ind	sient - o	sient - es	sient - e	sent - imos	sent - ís	sient - en	
impf_ind	sent - ía	sent - ías	sent - ía	sent - íamos	sent - íais	sent - ían	
pret_ind	sent - í	sent - iste	sint - ió	sent - imos	sent - isteis	sint - ieron	
futuro	sent - iré	sent - irás	sent - irá	sent - iremos	sent - iréis	sent - irán	
pres_subj	sient - a	sient - as	sient - a	sint - amos	sint - áis	sient - an	
imp_subj	sint - iera, iese	sint - ieras, ieses	sint - iera, iese	sint - iéramos, iésemos	sint - ierais, ieseis	sint - ieran, iesen	
cond	sent - iría	sent - irías	sent - iría	sent - iríamos	sent - iríais	sent - irían	
imper		sient - e			sent - id		
infin							sent - ir
ger							sint - iendo
part							sent - ido

- **tipo_raíz**
  1. ***sent***: 00 14 15 21 22 23 24 25 26 31 32 34 35 41 42 43 44 45 46 71 72 73 74 75 76 85 99
  2. ***sient***: 11 12 13 16 51 52 53 56 82
  3. ***sint***: 33 36 54 55 61 62 63 64 65 66 90
- **tipo_des**
  1. ***reg***: todas las formas

Este subgrupo tampoco tiene ninguna desinencia irregular, como los del grupo PEDIR, a los que se parecen bastante salvo en que los verbos en *-e-ir, -o-ir* tienen un alomorfo más de la raíz. Son exclusivamente verbos de la tercera conjugación y los cambios vocálicos son:

- verbos con el patrón -E-IR:
  - diptongación *e* → *ie*
  - cerramiento vocálico *e* → *i*
- verbos con el patrón -O-IR:
  - diptongación *o* → *ue*
  - cerramiento vocálico *o* → *u*

Tabla 29. Tipo 9b: REÍR. Cambios vocálicos en las formas fuertes (cerramiento vocálico, *-eír*)							
	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	rí - o	rí - es	rí - e	re - ímos	re - ís	rí - en	
impf_ind	re - ía	re - ías	re - ía	re - íamos	re - íais	re - ían	
pret_ind	re - í	re - íste	ri - ó	re - ímos	re - ísteis	ri - eron	
futuro	re - iré	re - irás	re - irá	re - iremos	re - iréis	re - irán	
pres_subj	rí - a	rí - as	rí - a	ri - amos	ri - áis	rí - an	
imp_subj	ri - era, ese	ri - eras, eses	ri - era, ese	ri - éramos, ésemos	ri - erais, eseis	ri - eran, esen	
cond	re - iría	re - irías	re - iría	re - iríamos	re - iríais	re - irían	
imper		rí - e			re - íd		
infin							re - ír
ger							ri - endo
part							re - ído

- **tipo_raíz**
  1. *re*: 00 14 15 21 22 23 24 25 26 31 32 34 35 41 42 43 44 45 46 71 72 73 74 75 76 85 99
  2. *rí*: 11 12 13 16 51 52 53 56 82
  3. *ri*: 33 36 54 55 61 62 63 64 65 66 90
- **tipo_des**
  1. *reg*: 00 12 13 14 15 16 21 22 23 24 25 26 31 35 41 42 43 44 45 46 51 52 53 54 55 56 71 72 73 74 75 76 82 85 99
  2. *pret2*: 32 33 34 35 36
  3. *imp_subj*: 61 62 63 64 65 66
  4. *ger*: 90

Los verbos en *-EÍR* son muy irregulares en cuanto a variantes de la terminación flexiva, dándose casos verdaderamente excepcionales como las variantes de las formas 32 (*-íste*), 34 (*-ímos*), 35 (*-ísteis*) o la del infinitivo (*-ír*). Están producidas por cambios ortográficos, al igual que los verbos en -OÍR (Tipo 10a, pág. 154).

Nótese además que estas formas aportan un argumento adicional para considerar *-ó* y *-eron* como alomorfos de la desinencia. Mantener que la segmentación de las terceras personas del pretérito es *r - ió, r - ieron* obligaría a añadir una entrada radical para estas formas. Nuestra segmentación, en cambio, permite utilizar las

entradas ya existentes para otras formas del paradigma (la raíz *ri* y las desinencias *ó* y *eron*), además de preservar el paralelismo distribucional de las desinencias con otros verbos de otros paradigmas (verbos en *-ÑIR, -UIR, -OÍR, -EER*). Por otra parte, la R.A.E. considera que estos verbos han perdido la *i* del diptongo de la desinencia y que la raíz es *ri-*, con lo que su segmentación coincide con la nuestra.

Otros verbos de este paradigma son:

adherir	*adormir*	*advertir*	*arrepentir*	*asentir*
conferir	*consentir*	*controvertir*	*convertir*	*desadvertir*
desconsentir	*desleír*	*desmentir*	*diferir*	*digerir*
disentir	*divertir*	*dormir*	*engreír*	*entremorir*
erguir	*freír*	*herir*	*inferir*	*ingerir*
injerir	*invertir*	*malherir*	*mentir*	*morir*
pervertir	*preferir*	*proferir*	*presentir*	*referir*
refreír	*reír*	*requerir*	*resentir*	*revertir*
sentir	*sofreír*	*sonreír*	*subvertir*	*sugerir*
transferir	*zaherir*			

Modelo 10

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	222	111/333	111/333	111	111	111/333	
impf_ind	111	111	111	111	111	111	
pret_ind	111	111	333	111	111	333	
futuro	111	111	111	111	111	111	
pres_subj	222	222	222	222	222	222	
imp_subj	333	333	333	333	333	333	
cond	111	111	111	111	111	111	
imper		111/333			111		
infin							111
ger							333
part							111

Tabla 30. Tipo 10a: OÍR. Varias irregularidades (*o* → *oig*, *o* → *oy*, verbos en *-oír*)							
	sg_1	**sg_2**	**sg_3**	**pl_1**	**pl_2**	**pl_3**	**no**
pres_ind	oig - o	oy - es	oy - e	o - ímos	o - ís	oy - en	
impf_ind	o - ía	o - ías	o - ía	o - íamos	o - íais	o - ían	
pret_ind	o - í	o - íste	oy - ó	o - ímos	o - ísteis	oy - eron	
futuro	o - iré	o - irás	o - irá	o - iremos	o - iréis	o - irán	
pres_subj	oig - a	oig - as	oig - a	oig - amos	oig - áis	oig - an	
imp_subj	oy - era, ese	oy - eras, eses	oy - era, ese	oy - éramos, ésemos	oy - erais, eseis	oy - eran, esen	
cond	o - iría	o - irías	o - iría	o - iríamos	o - iríais	o - irían	
imper		oy - e			o - íd		
infin							o - ír
ger							oy - endo
part							o - ído

- **tipo_raíz**
  1. *o*: 00 14 15 21 22 23 24 25 26 31 32 34 35 41 42 43 44 45 46 71 72 73 74 75 76 85 99
  2. *oig*: 11 51 52 53 54 55 56
  3. *oy*: 12 13 16 33 36 61 62 63 64 65 66 82 90
- **tipo_des**
  1. *reg*: 11 12 13 15 16 21 22 23 24 25 26 31 41 42 43 44 45 46 51 52 53 54 55 56 71 72 73 74 75 76 82
  2. *imp_subj*: 61 62 63 64 65 66
  3. *pret2*: 32 33 34 35 36
  4. *pres*: 14
  5. *imper*: 85
  6. *infin*: 00
  7. *ger*: 90
  8. *part2*: 99

Los verbos en *-OÍR* presentan todas las irregularidades de las desinencias que son puramente ortográficas (por la norma que obliga a acentuar las vocales *i, u* cuando son hiatos). Algunas de ellas las hemos visto también en los verbos en -EÍR (Tipo 9b, pág. 152)

Por otra parte, al igual que los verbos en *-UIR*, existe una variante de la raíz con la *y* (*oy - es, oy - e, oy - en*). Es un argumento que hemos utilizado para no crear nuevas desinencias en *-yó, -yeron*, etc.

Tabla 31. Tipo 10b: SEGUIR. Varias irregularidades (cambio vocálico y cambio gráfico *g* → *gu*, verbos en *-eguir*)

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	sig - o	sigu - es	sigu - e	segu - imos	segu - ís	sigu - en	
impf_ind	segu - ía	segu - ías	segu - ía	segu - íamos	segu - íais	segu - ían	
pret_ind	segu - í	segu - iste	sigu - ió	segu - imos	segu - isteis	sigu - ieron	
futuro	segu - iré	segu - irás	segu - irá	segu - iremos	segu - iréis	segu - irán	
pres_subj	sig - a	sig - as	sig - a	sig - amos	sig - áis	sig - an	
imp_subj	sigu - iera, iese	sigu - ieras, ieses	sigu - iera, iese	sigu - iéramos, iésemos	sigu - ierais, ieseis	sigu - ieran, iesen	
cond	segu - iría	segu - irías	segu - iría	segu - iríamos	segu - iríais	segu - irían	
imper		sigu - e			segu - id		
infin							segu - ir
ger							sigu - iendo
part							segu - ido

- **tipo_raíz**
  1. ***segu***: 00 14 15 21 22 23 24 25 26 31 32 34 35 41 42 43 44 45 46 71 72 73 74 75 76 85 99
  2. ***sig***: 11 51 52 53 54 55 56
  3. ***sigu***: 12 13 16 33 36 61 62 63 64 65 66 82 90
- **tipo_des**
  1. ***reg***: para todas las formas

SEGUIR podría incluirse en el modelo de PEDIR (cambio vocálico *e* → *i*) si no fuera por la peculiaridad gráfica de *sig - o, a, as,...*, que obliga a tener otra variante de la raíz. En la distribución de sus raíces es exactamente igual a la de OÍR, por lo que lo incluimos en este modelo.

Tabla 32. Tipo 10c: CAER. Varias irregularidades (*a* → *aig*, *a* → *ay*)							
	sg_1	**sg_2**	**sg_3**	**pl_1**	**pl_2**	**pl_3**	**no**
pres_ind	caig - o	ca - es	ca - e	ca - emos	ca - ís	ca - en	
impf_ind	ca - ía	ca - ías	ca - ía	ca - íamos	ca - íais	ca - ían	
pret_ind	ca - í	ca - íste	cay - ó	ca - ímos	ca - ísteis	cay - eron	
futuro	ca - iré	ca - irás	ca - irá	ca - iremos	ca - iréis	ca - irán	
pres_subj	caig - a	caig - as	caig - a	caig - amos	caig - áis	caig - an	
imp_subj	cay - era, ese	cay - eras, eses	cay - era, ese	cay - éramos, ésemos	cay - erais, eseis	cay - eran, esen	
cond	ca - iría	ca - irías	ca - iría	ca - iríamos	ca - iríais	ca - irían	
imper		ca - e			ca - ed		
infin							ca - er
ger							cay - endo
part							ca - ído

- **tipo_raíz**
  1. ***ca***: 00 12 13 14 15 16 21 22 23 24 25 26 31 32 34 35 41 42 43 44 45 46 71 72 73 74 75 76 82 85 99
  2. ***caig***: 11 51 52 53 54 55 56
  3. ***cay***: 33 36 61 62 63 64 65 66 90
- **tipo_des**
  1. ***reg***: 00 11 12 13 14 15 16 21 22 23 24 25 26 31 41 42 43 44 45 46 51 52 53 54 55 56 71 72 73 74 75 76 82 85
  2. ***imp_subj***: 61 62 63 64 65 66
  3. ***pret2***: 32 33 34 35 36
  4. ***ger***: 90
  5. ***part2***: 99

CAER comparte con OÍR las raíces de presente en *-ig* (*caigo, oigo, caiga, oiga*) además de muchas de las irregularidades ortográficas de la desinencias (*caíste, oíste, caímos, oímos, caído, oído*). Por razones suficientemente explicadas, hemos tratado análogamente la segmentación de las terceras personas del pretérito, todas las del imperfecto del subjuntivo y el gerundio: *cayó, oyó, cayeron, oyeron, cayera, oyera,... cayendo, oyendo.*

Damos a continuación la lista de verbos de este modelo:

caer	*conseguir*	*decaer*	*desoír*	*entreoír*
oír	*perseguir*	*proseguir*	*recaer*	*seguir*
reseguir	*subseguir*	*trasoír*		

Modelo 11

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	222	111	111	111	111	111	
impf_ind	111	111	111	111	111	111	
pret_ind	111	111	222	111	111	222	
futuro	333	333	333	333	333	333	
pres_subj	222	222	222	222	222	222	
imp_subj	111	111	111	111	111	111	
cond	333	333	333	333	333	333	
imper		111			111		
infin							111
ger							111
part							111

Tabla 33. Tipo 11: VALER SALIR. Cambios consonánticos (*al→ alg, al → ald,* verbos en *-aler, alir*)

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	valg - o	val - es	val - e	val - emos	val - éis	val - en	
impf_ind	val - ía	val - ías	val - ía	val - íamos	val - íais	val - ían	
pret_ind	val - í	val - iste	val - ió	val - imos	val - isteis	val - ieron	
futuro	vald - ré	vald - rás	vald - rá	vald - remos	vald - réis	vald - rán	
pres_subj	valg - a	valg - as	valg - a	valg - amos	valg - áis	valg - an	
imp_subj	val - iera, iese	val - ieras, ieses	val - iera, iese	val - iéramos, iésemos	val - ierais, ieseis	val - ieran, iesen	
cond	vald - ría	vald - rías	vald - ría	vald - ríamos	vald - ríais	vald - rían	
imper		val - e			val - ed		
infin							val - er
ger							val - iendo
part							val - ido

- **tipo_raíz**

1. ***val***: 00 12 13 14 15 16 21 22 23 24 25 26 31 32 33 34 35 36 61 62 63 64 65 66 82 85 90 99
2. ***valg***: 11 51 52 53 54 55 56
3. ***vald***: 41 42 43 44 45 46 71 72 73 74 75 76

- **tipo_des**
  1. ***reg***: 00 11 12 13 15 16 21 22 23 24 25 26 31 41 42 43 44 45 46 51 52 53 54 55 56 61 62 63 64 65 66 71 72 73 74 75 76 82 85
  2. ***fut_cond***: 41 42 43 44 45 46 71 72 73 74 75 76

Este paradigma no exhibe ninguna irregularidad especial. Está compuesto por los derivados de VALER y SALIR:

equivaler *prevaler* *resalir* *salir* *sobresalir* *valer*

Modelo 12

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	111	111	111	111	111	111	
impf_ind	111	111	111	111	111	111	
pret_ind	111	111	222	111	111	222	
futuro	111	111	111	111	111	111	
pres_subj	111	111	111	111	111	111	
imp_subj	222	222	222	222	222	222	
cond	111	111	111	111	111	111	
imper		111			111		
infin							111
ger							222
part							111

Tabla 34. Tipo 12: LEER, CREER, POSEER. Irregularidades especiales en las desinencias (verbos en *-eer*)							
	sg_1	**sg_2**	**sg_3**	**pl_1**	**pl_2**	**pl_3**	**no**
pres_ind	le - o	le - es	le - e	le - emos	le - éis	le - en	
impf_ind	le - ía	le - ías	le - ía	le - íamos	le - íais	le - ían	
pret_ind	le - í	le - íste	ley - ó	le - ímos	le - ísteis	ley - eron	
futuro	le - eré	le - erás	le - erá	le - eremos	le - eréis	le - erán	
pres_subj	le - a	le - as	le - a	le - amos	le - áis	le - an	
imp_subj	ley - era, ese	ley - eras, eses	ley - era, ese	ley - éramos, ésemos	ley - erais, eseis	ley - eran, esen	
cond	le - ería	le - erías	le - ería	le - eríamos	le - eríais	le - erían	
imper		le - e			le - ed		
infin							le - er
ger							ley - endo
part							le - ído

- **tipo_raíz**
  1. *le*: 00 11 12 13 14 15 16 21 22 23 24 25 26 31 32 34 35 41 42 43 44 45 46 51 52 53 54 55 56 71 72 73 74 75 76 82 85 99
  2. *ley*: 33 36 61 62 63 64 65 66 90
- **tipo_des**
  1. *reg*: 00 11 12 13 14 15 16 21 22 23 24 25 26 31 41 42 43 44 45 46 51 52 53 54 55 56 71 72 73 74 75 76 82 85
  2. *imp_subj*: 61 62 63 64 65 66
  3. *pret2*: 32 33 34 35 36
  4. *ger*: 90
  5. *part2*: 99

Como explicamos al principio de este capítulo (sección 6.1.1.3), los trece verbos en *-eer* tienen en nuestra clasificación dos variantes de la raíz, en analogía con CAER, para evitar nuevas desinencias *-yó, -yeran, -yendo...* Esta decisión se ha tomado teniendo en consideración que, aunque creáramos dichas desinencias, necesitaríamos dos entradas para la raíz.

Los verbos de este paradigma son:

acreer	*creer*	*descreer*	*desposeer*	*desproveer*
esleer	*leer*	*malcreer*	*peer*	*poseer*
proveer	*releer*	*sobreseer*		

Modelo 13

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	111	111	111	111	111	111	
impf_ind	111	111	111	111	111	111	
pret_ind	111	111	222	111	111	222	
futuro	111	111	111	111	111	111	
pres_subj	111	111	111	111	111	111	
imp_subj	222	222	222	222	222	222	
cond	111	111	111	111	111	111	
imper		111			111		
infin							111
ger							222
part							111

Tabla 35. Tipo 13: MULLIR EMPELLER ATAÑER GRUÑIR. Irregularidades especiales en las desinencias (verbos en *-añer, -añir, -iñir, -uñir, -ullir, -eller*)

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	gruñ - o	gruñ - es	gruñ - e	gruñ - imos	gruñ - ís	gruñ - en	
impf_ind	gruñ - ía	gruñ - ías	gruñ - ía	gruñ - íamos	gruñ - íais	gruñ - ían	
pret_ind	gruñ - í	gruñ - iste	gruñ - ó	gruñ - imos	gruñ - isteis	gruñ - eron	
futuro	gruñ - iré	gruñ - irás	gruñ - irá	gruñ - iremos	gruñ - iréis	gruñ - irán	
pres_subj	gruñ - a	gruñ - as	gruñ - a	gruñ - amos	gruñ - áis	gruñ - an	
imp_subj	gruñ - era, ese	gruñ- eras, eses	gruñ - era, ese	gruñ - éramos, ésemos	gruñ - erais, eseis	gruñ - eran, esen	
cond	gruñ - iría	gruñ - irías	gruñ - iría	gruñ - iríamos	gruñ - iríais	gruñ - irían	
imper		gruñ - e			gruñ - id		
infin							gruñ - ir
ger							gruñ - endo
part							gruñ - ido

- **tipo_raíz**

  1. ***gruñ***: 00 11 12 13 14 15 16 21 22 23 24 25 26 31 32 34 35 41 42 43 44 45 46 51 52 53 54 55 56 71 72 73 74 75 76 82 85 99

  2. ***gruñ***: 33 36 61 62 63 64 65 66 90

- **tipo_des**

  1. ***reg***: 00 11 12 13 14 15 16 21 22 23 24 25 26 31 32 34 35 41 42 43 44 45 46 51 52 53 54 55 56 71 72 73 74 75 76 82 85
  2. ***imp_subj***: 61 62 63 64 65 66
  3. ***pret2***: 33 36
  4. ***ger***: 90

Este modelo es especial porque no es irregular por su raíz sino por sus terminaciones. Estos verbos presentan variaciones con respecto a las cadenas regulares de la desinencia en las terceras personas del pretérito de indicativo, todas las personas del imperfecto del subjuntivo y el gerundio. Nótese además que la distribución de las raíces es la misma que la de los verbos del modelo 12 (-EER), lo que permitiría también agruparlos con ellos.

Verbos que se ajustan a este modelo son:

atañer	*bruñir*	*complañir*	*constriñir*	*desmullir*
empeller	*empuñir*	*engullir*	*engurruñir*	*escabullir*
frañer	*gañir*	*gruñir*	*mullir*	*muñir*
plañer	*plañir*	*rebullir*	*regruñir*	*remullir*
restriñir	*retiñir*	*sarpullir*	*tañer*	*tullir*
uñir	*zambullir*			

6.1.4 Paradigmas irregulares especiales

Hay algunos verbos sueltos que no pueden incluirse en ningún modelo de los mencionados, debido a sus peculiaridades. Estos verbos aparecen en todas las clasificaciones tratados aparte. La mayoría de sus irregularidades son tan particulares que hemos decidido lexicalizar algunas de sus formas para evitar sobrecargar excesivamente el diccionario con desinencias y entradas muy pocos productivas. Estas irregularidades excepcionales son:

- los verbos *ser* e *ir* tienen variantes radicales completamente distintas, resultado de la evolución histórica. Concretamente las formas del imperfecto del indicativo (*era, eras, iba, ibas...*) se apartan claramente de la regularidad.
- *estar* y *dar* en algunas formas de los presentes llevan acento gráfico en sus terminaciones, hecho aislado en toda la conjugación irregular. Además, por ser uno bisílabo y el otro monosílabo tampoco coinciden sus irregularidades en las mismas formas. Es por tanto un caso claro para lexicalizar.
- el presente del indicativo de los verbos *ser* y *haber* también presentan una irregularidad especial (*soy, eres... he, has...*). Igualmente, las primeras personas del presente de indicativo de los verbos *dar, estar* e *ir* son casos aislados en la conjugación. Todas estas formas también se lexicalizan.
- algunas formas debido a su carácter monosilábico reciben un tratamiento gráfico distinto al "regular," por ejemplo, *fui, fue, sé, di, dio, dais, deis, dé, vais, vi, vio* (en lugar de *fuí, fué, se, dí, dió, dáis, déis, de, váis, ví, vió*). Se lexicalizan.

- la evolución histórica de algunos verbos españoles ha conducido a una situación anómala dentro de las conjugaciones: por ejemplo, el verbo *ir* ha heredado sus formas del presente de indicativo del verbo latino *vadere*, por lo que usa las terminaciones de la primera conjugación (*v- as, v- a, v- amos...*); los verbos *andar* y *estar* han creado por analogía con *haber* pretéritos fuertes e imperfectos que llevan terminaciones de la segunda conjugación (*hub- e, anduv- e, estuv- e, hub- iera, anduv- iera, estuv- iera...*). Análogamente, las formas de pretérito de indicativo e imperfecto del subjuntivo de *dar* evolucionaron del latín *dare* (donde ya era irregular: *dedi*) hacia las desinencias propias de los verbos de la segunda y tercera. En consecuencia, vamos a asignar a estas raíces (*v-, anduv-, estuv-* y *d-*) las terminaciones correspondientes a las conjugaciones mencionadas.

Tabla 36. SER

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	soy	eres	es	somos	sois	son	
impf_ind	era	eras	era	éramos	erais	eran	
pret_ind	fui	fu - iste	fue	fu - imos	fu - isteis	fu - eron	
futuro	s - eré	s - erás	s - erá	s - eremos	s - eréis	s - erán	
pres_subj	se - a	se - as	se - a	se - amos	se - áis	se - an	
imp_subj	fu - era, ese	fu - eras, eses	fu - era, ese	fu - éramos, ésemos	fu - erais, eseis	fu - eran, esen	
cond	s - ería	s - erías	s - ería	s - eríamos	s - eríais	s - erían	
imper		sé			s - ed		
infin							s - er
ger							s - iendo
part							s - ido

- **tipo_raíz**
  1. *s*: 00 41 42 43 44 45 46 71 72 73 74 75 76 85 90 99
  2. *fu*: 32 34 35 36 61 62 63 64 65 66
  3. *se*: 51 52 53 54 55 56
- **tipo_des**
  1. *reg*: 00 32 34 35 41 42 43 44 45 46 51 52 53 54 55 56 71 72 73 74 75 76 85 90 99
  2. *imp_subj*: 61 62 63 64 65 66
  3. *pret2*: 36
- **formas lexicalizadas:** ***soy, eres, es, somos, sois, son, era, eras, era, éramos, erais, eran, fui, fue*** y ***sé***

La característica principal de SER es la diferente procedencia de las raíces que forman su conjugación. Esto nos obliga a lexicalizar una parte de las formas (todo el imperfecto y el presente de indicativo). Nuestra codificación aprovecha todas las desinencias que existen ya en otros verbos para evitar más lexicalizaciones. Por ejemplo, en el pretérito del indicativo y el imperfecto de subjuntivo tomamos como raíz *fu-* y utilizamos las desinencias apropiadas de la segunda conjugación (*fui* y *fue* se lexicalizan pues sus posibles desinencias *-i, -e* no se corresponden con ninguna de las establecidas para la segunda (*hub-e, hub-o, tem-í, tem-ió*).

Tabla 37. IR

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	voy	v - as	v - a	v - amos	vais	v - an	
impf_ind	iba	ibas	iba	íbamos	ibais	iban	
pret_ind	fui	fu - iste	fue	fu - imos	fu - isteis	fu - eron	
futuro	i - ré	i - rás	i - rá	i - remos	i - réis	i - rán	
pres_subj	vay - a	vay - as	vay - a	vay - amos	vay - áis	vay - an	
imp_subj	fu - era, ese	fu - eras, eses	fu - era, ese	fu - éramos, ésemos	fu - erais, eseis	fu - eran, esen	
cond	i - ría	i - rías	i - ría	i - ríamos	i - ríais	i - rían	
imper		ve			id		
infin							ir
ger							yendo
part							ido

- **tipo_raíz**
  1. *i*: 41 42 43 44 45 46 71 72 73 74 75 76
  2. *fu*: 32 34 35 36 61 62 63 64 65 66
  3. *vay*: 51 52 53 54 55 56
  4. *v*: 12 13 14 16 (conjugación 1)
- **tipo_des**
  1. *reg*: 32 34 35 51 52 53 54 55 56 (conjugación 3)
  2. *reg*: 12 13 14 16 (conjugación 1)
  3. *fut_cond*: 41 42 43 44 45 46 71 72 73 74 75 76
  4. *imp_subj*: 61 62 63 64 65 66
  5. *pret2*: 36

- **formas lexicalizadas:** ***voy, vais, iba, ibas, iba, íbamos, ibais, iban, fui, fue, ve, id, ir, yendo*** e ***ido***

El verbo IR es el más irregular de todos los verbos castellanos. Por evolución histórica, el español "sustituyó las formas flexivas de sus dos presentes y el imperativo singular por las procedentes del verbo latino *vadere*, y su perfecto simple y las formas del subjuntivo afines del mismo por *fui ... fuese ... fuera ...* del verbo *ser*"(R.A.E. 1973;1989:305). Nosotros tratamos el presente del indicativo como una raíz *v-* y las desinencias regulares de la primera conjugación; el presente de subjuntivo tiene una raíz particular (*vay-*) y las desinencias regulares de la tercera conjugación; por último, tratamos las formas *fui, fuese*, etc. como se explicó en las correspondientes del verbo SER.

Las formas del imperfecto del indicativo se lexicalizan, al igual que las de SER, porque sus terminaciones flexivas son únicas en toda la conjugación española. Igualmente, las formas no personales tienen que ser lexicalizadas porque, en este caso, coinciden con las desinencias del infinitivo y participio (no sería rentable hacer una partición forzada del tipo *i - r, y - endo, id - o*, por ejemplo, pues complica tanto las raíces verbales como las desinencias).

Hemos establecido una segmentación particular para las formas del futuro y del imperfecto del subjuntivo *i - ré, i - ría*, aprovechando que existen desinencias en *-re ... -ría* .... En estos casos, aunque no exista ninguna justificación teórica, es pertinente para evitar tener otras doce formas lexicalizadas (dado que esta vez contamos con desinencias similares para otros verbos).

Tabla 38. HABER

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	he	has	ha	hemos	hab - éis	han	
impf_ind	hab - ía	hab - ías	hab - ía	hab - íamos	hab - íais	hab - ían	
pret_ind	hub - e	hub - iste	hub - o	hub - imos	hub - isteis	hub - ieron	
futuro	hab - ré	hab - rás	hab - rá	hab - remos	hab - réis	hab - rán	
pres_subj	hay - a	hay - as	hay - a	hay - amos	hay - áis	hay - an	
imp_subj	hub - iera, iese	hub - ieras, ieses	hub - iera, iese	hub - iéramos, iésemos	hub - ierais, ieseis	hub - ieran, iesen	
cond	hab - ría	hab - rías	hab - ría	hab - ríamos	hab - ríais	hab - rían	
imper		he			hab - ed		
infin							hab - er
ger							hab - iendo
part							hab - ido

- **tipo_raíz**
  1. ***hab***: 00 15 21 22 23 24 25 26 41 42 43 44 45 46 71 72 73 74 75 76 85 90 99
  2. ***hay***: 51 52 53 54 55 56
  3. ***hub***: 31 32 33 34 35 36 61 62 63 64 65 66
- **tipo_des**
  1. ***reg***: 00 12 13 14 15 16 21 22 23 24 25 26 32 34 35 36 51 52 53 54 55 56 61 62 63 64 65 66 90 99
  2. ***pret1***: 31 33
  3. ***fut_cond***: 41 42 43 44 45 46 71 72 73 74 75 76
- **formas lexicalizadas**: ***he, has, ha, hemos, han*** y ***he*** (82)

HABER es un verbo que distribucionalmente tiene mucho parecido con los verbos del modelo 5 (PONER, HACER): comparten las mismas irregularidades en la raíz y desinencia en todos los tiempos menos en el presente del indicativo. La primera del singular de HABER (*he*) tiene que ser lexicalizada porque su terminación no guarda ninguna relación con la desinencia de esa persona (*-o*), rompiendo así la simetría de las raíces irregulares (el grupo primero de formas afines de Bello: si cambia la raíz de la primera persona del singular, afecta también a todas las raíces de las formas de presente del subjuntivo). Por otra parte, el resto de las formas del presente del indicativo (excepto la segunda del plural, *hab - emos*) también se han lexicalizado.

Tabla 39. DAR

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	doy	d - as	d - a	d - amos	dais	d - an	
impf_ind	d - aba	d - abas	d - aba	d - ábamos	d - abais	d - aban	
pret_ind	di	d - iste	dio	d - imos	d - isteis	d - ieron	
futuro	d - aré	d - arás	d - ará	d - aremos	d - aréis	d - arán	
pres_subj	dé	d - es	dé	d - emos	deis	d - en	
imp_subj	d - iera, iese	d - ieras, ieses	d - iera, iese	d - iéramos, iésemos	d - ierais, ieseis	d - ieran, iesen	
cond	d - aría	d - arías	d - aría	d - aríamos	d - aríais	d - arían	
imper		d - a			d - ad		
infin							d - ar
ger							d - ando
part							d - ado

- **tipo_raíz**

1. *d*: 00 12 13 14 16 21 22 23 24 25 26 41 42 43 44 45 46 52 54 56 71 72 73 74 75 76 82 85 90 99 (conjugación 1)

2. *d*: 32 34 35 36 61 62 63 64 65 66 (conjugación 2)

- **tipo_des**

1. *reg*: 00 12 13 14 16 21 22 23 24 25 26 41 42 43 44 45 46 52 54 56 71 72 73 74 75 76 82 85 90 99 (conjugación 1)

2. *reg*: 32 34 35 36 61 62 63 64 65 66 (conjugación 2)

- **formas lexicalizadas**: ***doy, dais, di, dio, dé*** (51), ***dé*** (53) y ***deis***.

La peculiaridad de DAR, ESTAR y ANDAR es que aunque la mayoría de las formas toman desinencias de la primera conjugación, también presentan formas con desinencias propias de los verbos de la segunda y tercera (en el pretérito del indicativo y en el imperfecto del subjuntivo). Por lo tanto, utilizamos las desinencias "regulares" de la segunda-tercera conjugación para estos tiempos. Este tratamiento, además de estar fundado teóricamente[76] nos permite dar cuenta de estos fenómenos excepcionales sin recurrir ni a la lexicalización ni al establecimiento de nuevas variantes desinenciales para estas formas (ambas posibilidades son contrarias a nuestros criterios de segmentación y aumentan innecesariamente el diccionario). Como resultado obtenemos un tratamiento más económico y regular al tiempo que nos ajustamos a los datos lingüísticos.

[76] Tanto por motivos de evolución histórica (dichas formas evolucionaron así por analogía con las correspondientes en otros paradigmas) como por razones sincrónicas (las desinencias actuales coinciden con las desinencias regulares de los verbos de la segunda y tercera conjugación).

Tabla 40. ESTAR							
	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	estoy	estás	está	est - amos	est - áis	están	
impf_ind	est - aba	est - abas	est - aba	est - ábamos	est - abais	est - aban	
pret_ind	estuv - e	estuv - iste	estuv - o	estuv - imos	estuv - isteis	estuv - ieron	
futuro	est - aré	est - arás	est - ará	est - aremos	est - aréis	est - arán	
pres_subj	esté	estés	esté	est - emos	est - éis	estén	
imp_subj	estuv - iera, iese	estuv - ieras, ieses	estuv - iera, iese	estuv - iéramos, iésemos	estuv - ierais, ieseis	estuv - ieran, iesen	
cond	est - aría	est - arías	est - aría	est - aríamos	est - aríais	est - arían	
imper		está			est - ad		
infin							est - ar
ger							est - ando
part							est - ado

- **tipo_raíz**
  1. *est*: 00 14 15 21 22 23 24 25 26 41 42 43 44 45 46 54 55 71 72 73 74 75 76 85 90 99 (conjugación 1)
  2. *estuv*: 31 32 33 34 35 36 61 62 63 64 65 66 (conjugación 2)
- **tipo_des**
  1. *reg*: 00 14 15 21 22 23 24 25 26 41 42 43 44 45 46 54 55 71 72 73 74 75 76 85 90 99
  2. *reg*: 32 34 35 36 61 62 63 64 65 66 (conjugación 2)
  3. *pret1*: 31 33
- **formas lexicalizadas**: ***estoy, estás, está, están, esté, estés, esté, estén*** y ***está*** (82).

ESTAR presenta una irregularidad excepcional en la conjugación irregular del español: las desinencias de los presentes y segunda del singular del imperativo están acentuadas. Lo mismo ocurre con DAR, sólo que este verbo por ser monosilábico no acentúa algunas formas y por ello podemos utilizar las desinencias regulares, y viceversa. Véase el contraste:

```
d - as      estás
d - a       está
d - es      estés
dais        est - áis
deis        est - éis
```

Por otra parte, llevan las desinencias de la segunda y tercera conjugación para los tiempos pretérito del indicativo e imperfecto del subjuntivo, como explicamos en DAR.

Tabla 41. ANDAR							
	sg_1	**sg_2**	**sg_3**	**pl_1**	**pl_2**	**pl_3**	**no**
pres_ind	and - o	and - as	and - a	and - amos	and - áis	and - an	
impf_ind	and - aba	and - abas	and - aba	and - ábamos	and - abais	and - aban	
pret_ind	anduv - e	anduv - iste	anduv - o	anduv - imos	anduv - isteis	anduv - ieron	
futuro	and - aré	and - arás	and - ará	and - aremos	and - aréis	and - arán	
pres_subj	and - e	and - es	end - e	and - emos	and - éis	and - en	
imp_subj	anduv - iera, iese	anduv - ieras, ieses	anduv - iera, iese	anduv - iéramos, iésemos	anduv - ierais, ieseis	anduv - ieran, iesen	
cond	and - aría	and - arías	and - aría	and - aríamos	and - aríais	and - arían	
imper		and - a			and - ad		
infin							and - ar
ger							and - ando
part							and - ado

- **tipo_raíz**
  1. ***and***: 00 11 12 13 14 15 16 21 22 23 24 25 26 32 34 35 36 41 42 43 44 45 46 71 72 73 74 75 76 82 85 90 99 (conjugación 1)
  2. ***anduv***: 31 32 33 34 35 36 61 62 63 64 65 66 (conjugación 2)
- **tipo_des**
  1. ***reg***: 00 11 12 13 14 15 16 21 22 23 24 25 26 41 42 43 44 45 46 51 52 53 54 55 56 71 72 73 74 75 76 82 85 90 99 (conjugación 1)
  2. ***reg***: 32 34 35 36 61 62 63 64 65 66 (conjugación 2)
  3. ***pret1***: 31 33 (conjugación 2)

ANDAR exhibe una distribución de formas muy parecidas a ESTAR, con la diferencia de que no tiene ninguna forma lexicalizada. El pretérito del indicativo y el imperfecto del subjuntivo se conjugan con las desinencias de la segunda, como se explicó en DAR (pág. 165)

Tabla 42. VER

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	ve - o	v - es	v - e	v - emos	v - éis	v - en	
impf_ind	ve - ía	ve - ías	ve - ía	ve - íamos	ve - íais	ve - ían	
pret_ind	vi	v - iste	vio	v - imos	v - isteis	v - ieron	
futuro	v - eré	v - erás	v - erá	v - eremos	v - eréis	v - erán	
pres_subj	ve - a	ve - as	ve - a	ve - amos	ve - áis	ve - an	
imp_subj	v - iera, iese	v - ieras, ieses	v - iera, iese	v - íeramos, íesemos	v - ierais, ieseis	v - ieran, iesen	
cond	v - ería	v - erías	v - ería	v - eríamos	v - eríais	v - erían	
imper		v - e			v - ed		
infin							v - er
ger							v - iendo
part							vist - o

- **tipo_raíz**
  1. *v*: 00 12 13 14 15 16 32 34 35 36 41 42 43 44 45 46 61 62 63 64 65 66 71 72 73 74 75 76 82 85 90
  2. *ve*: 11 21 22 23 24 25 26 51 52 53 54 55 56
  3. *vist*: 99
- **tipo_des**
  1. *reg*: para todas las formas
- **formas lexicalizadas**: *vi* y *vio*

Por razones formales consideramos la raíz principal de VER a *v* frente a *ve*. Las clasificaciones fonológicas consideran, en cambio, que en *v - er, v - es, v - e...* se ha producido una "reducción silábica o contracción" de la raíz regular que se manifiesta en *ve - o, ve - ía, ve - a....* Por otra parte, necesitamos lexicalizar *vi, vio* por la misma razón que *di, dio*. Se conjugan de la misma manera los derivados *antever, entrever, prever* y *rever*.

6.1.5 ¿ Cómo codificar un verbo particular ?

Ya mencionamos que nuestro sistema de codificación permite describir la conjugación de cualquier verbo castellano. Es tan simple como rellenar adecuadamente la matriz, de la forma que hemos mostrado en los ejemplos. En los modelos que hemos visto, los verbos se han agrupado por similitudes distribucionales. Veremos a continuación cómo tratar un verbo que no se adapta a los modelos presentados.

En primer lugar, hablaremos de los verbos defectivos. Aunque parezca un grupo compacto (por el hecho de que se les agrupa a todos bajo el mismo término) casi cada verbo es un modelo aislado. Para empezar, puede afectar lo mismo a los verbos regulares (es decir, con una sola raíz como *abolir*) que a los irregulares (*concern - ir, conciern - e*). Además, las formas incompletas (las que no aparecen en la conjugación) no siempre son las mismas: unos verbos sólo se usan en participio (*aguerrido*); la mayoría sólo presentan las terceras personas (es el caso de los verbos llamados "atmosféricos" o "unipersonales," y también de *concernir, acontecer, acaecer...*); algunos de la tercera conjugación, como *abolir*, sólo se emplean en aquellas formas que "la terminación es *i* o principia por *i*" (Bello 1980:193); en otro casos, están excluidos los tiempos perfectivos (*acostumbrar, soler*). Si además añadimos que las distintas clasificaciones no se ponen de acuerdo a la hora de considerar si una forma concreta deber ser considerada defectiva o no[77] , se comprenderá nuestra afirmación de que deben tratarse como casos aislados, cada uno con su codificación particular. Daremos a continuación dos ejemplos, uno con un verbo de raíz única(*abolir*) y otro con doble raíz (*concernir*) y con distinta distribución de las formas.

[77] Compárese por ejemplo el capítulo dedicado a los verbos defectivos en las gramáticas de Bello y de la R.A.E, o los artículos concretos en el *Diccionario de Dudas* de M. Seco)

Tabla 43. ABOLIR							
	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	NO	NO	NO	abol - imos	abol - ís	NO	
impf_ind	abol - ía	abol - ías	abol - ía	abol - íamos	abol - íais	abol - ían	
pret_ind	abol - í	abol - iste	abol - ió	abol - imos	abol - isteis	abol - ieron	
futuro	abol - iré	abol - irás	abol - irá	abol - iremos	abol - iréis	abol - irán	
pres_subj	NO	NO	NO	NO	NO	NO	
imp_subj	abol - iera, iese	abol - ieras, ieses	abol - iera, iese	abol - iéramos, iésemos	abol - ierais, ieseis	abol - ieran, iesen	
cond	abol - iría	abol - irías	abol - iría	abol - iríamos	abol - iríais	abol - irían	
imper		NO			abol - id		
infin							abol - ir
ger							abol - iendo
part							abol - ido

- **tipo_raíz**
  1. ***abol***: 00 14 15 21 22 23 24 25 26 31 32 34 35 36 41 42 43 44 45 46 61 62 63 64 65 66 71 72 73 74 75 76 85 90 99
- **tipo_des**
  1. ***reg***: 00 14 15 21 22 23 24 25 26 31 32 34 35 36 41 42 43 44 45 46 61 62 63 64 65 66 71 72 73 74 75 76 85 90 99
  2. ***defect***: 11 12 13 16 51 52 53 54 55 56 82

Tabla 44. CONCERNIR							
	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	NO	NO	concier - e	NO	NO	conciern - en	
impf_ind	NO	NO	concern - ía	NO	NO	concern - ían	
pret_ind	NO	NO	concern - ió	NO	NO	concern - ieron	
futuro	NO	NO	NO	NO	NO	NO	
pres_subj	NO	NO	concier - a	NO	NO	conciern - an	
imp_subj	NO	NO	concern - iera, iese	NO	NO	concern - ieran, iesen	
cond	NO	NO	NO	NO	NO	NO	
imper		NO			NO		
infin							concern - ir
ger							concern - iendo
part							NO

- **tipo_raíz**
  1. ***concern***: 00 23 26 63 66 90
  2. ***conciern***: 13 16 53 56
- **tipo_des**
  1. ***reg***: 00 13 16 23 26 53 56 63 66 90
  2. ***defect***: el resto

Para los verbos como ACOSTUMBRAR o SOLER, que no se usan en los tiempos perfectivos (**he acostumbrado, *había solido*, según el *Esbozo*) o como CONCERNIR, que tampoco tiene formas compuestas, la solución es tratar como forma defectiva el participio. De esta manera, los tiempos compuestos no se pueden formar. *Acostumbrado* se incluiría en el diccionario como adjetivo.

Un ejemplo de verbo "particular" es *placer* y, en general, cualquier verbo que manifieste vacilaciones o alternancias a la hora de usar una forma concreta de la conjugación. Presenta una alternacia peculiar en las formas 33 (*plac - ió; plug - o*), 36 (*plac - ieron; plugu - ieron*) y 63 (*plac - iera, - iese; plugu - iera, iese*).[78]

Cuando nos encontramos con casos como estos, podemos optar por dos soluciones:

[78] La Academia propone también la alternancia en la forma 53 entre *plazca* y *pluga*.

1. Escoger una distribución como único modelo permitido por el sistema. Por ejemplo, decidirse por el uso más corriente y evitar la serie de alternancias menos frecuentes. En nuestro ejemplo, habría que suprimir las formas irregulares *plugo, pluguieron, pluguiera.* Esta opción es evidentemente la más sencilla y es muy útil si se conoce que en una aplicación determinada no se va a hacer uso de esas formas. Pero simplifica excesivamente las cosas y, sobre todo, no da cuenta de todas las formas posibles.

2. Intentar recoger todas las posibilidades. En el ejemplo, habría que permitir la convivencia de las formas regulares (*plació*) y las irregulares (*plugo*).

Nuestro sistema permite las dos opciones. Si escogemos la primera, simplemente tenemos que seguir el modelo 4a (CONOCER) y obtendríamos dos raíces: *plac-* y *plazc-*. Para la opción más completa, rellenamos la matriz de la forma siguiente:

Tabla 45. PLACER

	sg_1	sg_2	sg_3	pl_1	pl_2	pl_3	no
pres_ind	plazc - o	plac - es	plac - e	plac - emos	plac - éis	plac - en	
impf_ind	plac - ía	plac - ías	plac - ía	plac - íamos	plac - íais	plac - ían	
pret_ind	plac - í	plac - iste	plac - ió, plug - o	plac - imos	plac - isteis	plac - ieron, plugu - ieron	
futuro	plac - eré	plac - erás	plac - erá	plac - eremos	plac - eréis	plac - erán	
pres_subj	plazc - a	plazc - as	plazc - a	plazc - amos	plazc - áis	plazc - an	
imp_subj	plac - iera, iese	plac - ieras, ieses	plac - iera, iese, plugu -iera, iese	plac - íeramos, íesemos	plac - ierais, ieseis	plac - ieran, iesen	
cond	plac - ería	plac - erías	plac - ería	plac - eríamos	plac - eríais	plac - erían	
imper		plac - e			plac - ed		
infin							plac - er
ger							plac - iendo
part							plac - ido

- **tipo_raíz**

  1. ***plac***: 00 12 13 14 15 16 21 22 23 24 25 26 31 32 33 34 35 36 41 42 43 44 45 46 61 62 63 64 65 66 71 72 73 74 75 76 82 85 90 99

  2. ***plazc***: 11 51 52 53 54 55 56

  3. ***plug***: 33

4. ***plugu***: 36 63

- **tipo_des**

1. ***reg***: para todas las formas
2. ***pret1***: 33

Lo primero que observamos en esta codificación es que las formas 33 36 y 63 aparecen dos veces como valores del rasgo **tipo_raíz**, es decir, aparecen en cadenas distintas. Concretamente, esto significa que la cadena *plac* es la raíz de la forma 33, y que la cadena *plug* también es la raíz de la forma 33. ¿Cómo asignaremos la terminación correcta a cada raíz? Con el rasgo **tipo_des** de la siguiente manera:

plació

```
cadena = plac  tipo_raíz = ...33...   tipo_des = reg

cadena = ió  tipo_raíz = 33   tipo_des = reg
```

plugo

```
cadena = plug  tipo_raíz = 33   tipo_des = pret1

cadena = o   tipo_raíz = 33   tipo_des = pret1
```

PLACER, por su idiosincrasia, nos proporciona la oportunidad de comprobar el poder de nuestro sistema para generar o analizar casos excepcionalmente irregulares. En el capítulo dedicado a la Gramática veremos cómo ambas entradas pasan por la misma (y única) regla de la flexión verbal. Otros verbos que pueden ser tratados de esta manera son los derivados como *complacer, aplacer, desplacer* y *displacer* o los verbo RAER y ROER, que presentan las alternancias:

RAER: *raigo/rayo raiga/raya ... raigan/rayan*

ROER: *roo/roigo/royo roa/roiga/roya ...*

Hay algunos verbos regulares que tienen sin embargo un participio irregular[79] (*abrir, abierto*), o incluso conviven el participio fuerte con el débil, de creación romance (*imprimir, impreso, imprimido*). En ambos casos, se necesitan dos raíces distintas: una para las formas regulares y otra para el participio:

[79] Recuérdese que los participios fuertes de los verbos irregulares son tratados como una raíz especial dentro de sus conjugaciones.

```
                        ABRIR

cadena = abr   tipo_raíz = 00,11,12... 90    tipo_des = reg

cadena = abiert  tipo_raíz = 99    tipo_des = part1


                        IMPRIMIR

cadena = imprim   tipo_raíz = 100    tipo_des = reg

cadena = impres  tipo_raíz = 99    tipo_des = part1
```

Nótese que en el segundo caso (el de las formas alternantes) la formar regular (*imprim - ido*) se incluye dentro del código 100, mientras que para los participios irregulares únicos (*abiert - o*) se necesita escribir en la cadena regular todos los códigos de la conjugación menos el del participio (99).

Por último, hablaremos de algunos casos excepcionales provocados por la grafía, como por ejemplo *delinquir*, un verbo completamente regular desde una óptica fonológica. Presenta la irregularidad gráfica *qu* → *c* en la primera del singular del presente del indicativo y en todas las personas del presente de subjuntivo: *delinco, delinca, delincas ... delincamos*), y dicha distribución de irregularidades no se da en ninguno de los modelos de nuestra clasificación. Como no hay otros verbos acabados en -QUIR, aparte de *chasquir* y *muquir*, no creemos pertinente establecer un modelo especial. Posiblemente existan más casos aislados con este tipo de irregularidades gráficas que no puedan incluirse en los modelos de irregularidad que hemos visto pero, de cualquier forma, pueden codificarse perfectamente con el método descrito.

6.2 La Flexión Nominal

La flexión nominal del español presenta notables diferencias con la flexión verbal. En primer lugar, es posible una segmentación nítida en varios formantes debido a que se distinguen fonológica y gráficamente:

niñ - o - s *verde - s* *lápic - es* *leon - a* *president - e - s*

Esto simplifica bastante la descripción en comparación con las irregularidades verbales, y sobre todo no se presta ni a la vacilación ni a la toma de decisiones más o menos arbitrarias. Debido a esto último, en esta ocasión no seguiremos el modelo paradigmático. No hay ninguna duda de que en la palabra *n i ñ a s* la *a* corresponde al morfema de género femenino y la *s* al de número plural. La clasificación paradigmática agruparía en *as* ambas informaciones, pero a costa de la redundancia que supone tener un morfema *a* con información de género femenino y número singular (para los casos como *niñ - a*) y otro morfema *s* con la información de número plural (para las palabras

de género inherente como *mano - s*) En los formalismos basados en la unificación tiene mucho peso el tratamiento composicional de la información. Es decir, el significado del constructo resultante es la suma de la información de las partes que lo forman. Si podemos distinguir nítidamente de qué elemento proviene cada pieza informativa (rasgos), debemos mantener esa distinción en lugar de agrupar varios elementos. Llevando esto último a un caso extremo, desde un punto de vista no composicional no es necesario el procesamiento morfológico: ¿para qué necesitamos dividir la palabra en partes con significado autónomo, si podemos tener toda la información en la misma palabra? Evidentemente, si rechazamos el tratamiento composicional de la información desde los morfemas, poco tendremos que hacer en un diccionario[80] . Este es uno de los argumentos a favor de una morfología basada en la unificación: los morfemas son los primeros elementos autónomos con información y esta información puede ser tratada apropiadamente mediante un mecanismo que combine la información de varios de esos elementos de forma composicional y la transmita a un constructo superior.

Una segunda característica es que nos encontramos con un reducidísimo inventario de morfemas flexivos y variantes alomórficas. Las desinencias verbales necesitan casi 150 entradas para registrar todas sus variantes. En cambio, sólo hay 5 entradas para los morfemas flexivos nominales. Evidentemente, esto favorece también a la descripción.

En tercer lugar, hay que señalar que la flexión nominal es un proceso morfológico que afecta a varias categorías sintácticas por igual. Estas categorías, pueden ser abiertas (es decir, que no están constituidas por un inventario cerrado de unidades léxicas y, por lo tanto, se pueden ampliar indefinidamente) como el nombre o el adjetivo; o bien pueden ser categorías cerradas como los pronombres o los cuantificadores[81] . Si enfocamos el diccionario del español desde un punto de vista "productivo," la flexión nominal tiene mucho más peso que la verbal, pues abarca cuantitativamente un número bastante mayor de entradas léxicas. En ese sentido, la relativa simplicidad de la flexión nominal del español beneficia al tiempo general del procesamiento morfológico[82] .

Otra diferencia: la mayoría de las irregularidades de la raíz (es decir, cuando hay más de un alomorfo radical) tienen que ver con cambios ortográficos, como la pérdida del acento gráfico (*camión, camion - es*) o el cambio de *z* por *c* (*pez, pec - es*). Recordemos que la mayoría de las irregularidades radicales de los verbos se deben a cambios fonológicos. Por otra parte, no se puede hablar de irregularidad en los morfemas flexivos nominales (en el sentido de que no hay un modelo regular de flexión, como ocurre con las conjugaciones verbales, donde necesitábamos un rasgo especial **tipo_des** para dar cuenta de las variantes). El morfema de plural, por ejemplo, se realiza como *-s* o como *-es* según los contextos morfofonológicos, pero sin que cualquiera de los dos se "sienta" como más regular que el otro.

En relación con la regularidad y la composicionalidad está el concepto de lexicalización. Una forma se lexicaliza cuando su estructura formal no se adapta a los

---

[80] Sin embargo, la aproximación paradigmática podría ser eficiente en ciertas aplicaciones, donde se valore más la rapidez que la elegancia teórica.

[81] Estamos asumiendo un punto de vista sincrónico. Por supuesto, los pronombres y cuantificadores también evolucionan con el paso del tiempo.

[82] Compárese por ejemplo la complejidad de los nominales del alemán, con tres géneros y varios casos.

morfemas existentes (y por lo tanto sería necesario crear nuevas entradas alomórficas para esa forma particular). Aunque se puede hablar de casos más o menos especiales en la flexión nominal (como las palabras invariables del tipo *virus, matemáticas, alicates...*) no se los debe considerar propiamente como casos excepcionales que necesiten ser lexicalizados. Es más, una diferencia notoria en nuestra codificación es que no tenemos ningún nominal lexicalizado y todos se tratan por medio de reglas.

A pesar de las diferencias señaladas, el método de descripción es básicamente el mismo. Una entrada de diccionario es un "haz o estructura de rasgos" compuesto por:

- Una *cadena superficial* de caracteres.
- Un conjunto de rasgos con información gramatical.
- Un conjunto de rasgos con información para la concatenación de las cadenas superficiales.

La información gramatical de los nominales es su *género* y su *número* (además de la *categoría*). Los rasgos para la concatenación se encargan de distinguir qué alomorfo de género y número es el que se tiene que adjuntar a una raíz nominal dada. Para ello contamos con **tipo_gen** y **tipo_num**, que (análogamente a **tipo_raíz** y **tipo_des**) pueden llevar varios valores, y se asignan tanto a la raíz como a los morfemas flexivos.

Por último señalar en esta introducción que, en analogía con la flexión verbal, daremos una clasificación paradigmática de distintos modelos de flexión nominal. Sin embargo, queremos insistir en que tales paradigmas se muestran como ayuda práctica (de la misma forma que se presentaban los modelos verbales) para codificar entradas de diccionario. El método basado en rasgos que exponemos permite dar cuenta de cualquier forma nominal del castellano.

6.2.1 El género

Entendemos por formas nominales aquellas que tienen rasgos de género y número. Aunque lo habitual es que estos rasgos se manifiesten por medio de formantes, no necesariamente esto es siempre así. Es más, en el caso del rasgo **género** lo más corriente es que no aparezca marca formal o, lo que es lo mismo, morfema flexivo de género. Es decir, desde un punto estrictamente formal se pueden dar dos tipos básicos de asignación del rasgo de **género**:

1. formas nominales sin morfema flexivo de género (palabras de *género inherente*), y
2. formas nominales con morfema flexivo de género

Esta distinción formal marca un diferencia fundamental: las primeras llevan la información morfosintáctica de género (y de número también) en la entrada léxica; las segundas reciben esa información de los morfemas flexivos. Las primeras pueden funcionar sintácticamente sin necesidad de otros morfemas (es lo que se conoce en Morfología Generativa como *palabras*). Las segundas no pueden aparecer aisladamente y necesitan "completarse" con los morfemas flexivos (son las *raíces*). Esta diferencia crucial de tener o no función sintáctica independiente nosotros lo expresaremos mediante el rasgo **nivel**, con dos valores:

- **nivel** = **nivel_0** para las ***palabras*** como *mano, sol, león*
- **nivel** = **nivel_-1** para las ***raíces*** como *niñ-, leon-*.

Su significado, siguiendo a Selkirk (1982), es: sólo las categorías que tengan nivel 0 o superior pueden manifestarse superficialmente (son el punto de contacto entre la morfología y la sintaxis); el **nivel -1** representa las categorías terminales del diccionario, que no pueden aparecer en la superficie de forma aislada.

El rasgo **nivel** también se asigna a los verbos (y a las demás categorías léxicas). Todas las raíces verbales llevan el valor **nivel_-1** y las formas lexicalizadas el valor **nivel_0**. Las raíces no poseen toda la información necesaria para funcionar en el nivel sintáctico y necesitan ser procesadas previamente por el componente morfológico. Por el contrario, las formas lexicalizadas aparecen ya con toda la información que les permite ser elementos terminales de la sintaxis[83] .

Por lo tanto, el diccionario estará compuesto por:

- **Palabras**: elementos léxicos con información gramatical suficiente y capaces de aparecer como elementos terminales en la superficie,
- **Raíces**: elementos léxicos con información gramatical insuficiente para aparecer aisladamente y que necesitan concatenarse con elementos flexivos, y
- **Morfemas flexivos**: elementos con información gramatical pero que necesitan combinarse con elementos léxicos.

Una vez aclarada esta importante distinción, continuaremos con la exposición sobre el género. Esta información gramatical se especifica mediante un único rasgo (**género**) con varios valores. Dicho rasgo se coloca en las *palabras*, en el morfema de género y también en las *raíces*, pero en este caso por su ausencia (i.e. con el valor **no**):

```
cadena = sol  nivel = nivel_0  unidad_léxica = sol    género = mas

cadena = mano   nivel = nivel_0  unidad_léxica = mano    género = fem

cadena = niñ   nivel = nivel_-1  unidad_léxica = niño    género = no

cadena = bonit  nivel = nivel_-1  unidad_léxica = bonito    género = no

cadena = o   género = mas

cadena = e   género = mas

cadena = a   género = fem
```

Hay cuatro valores posibles para **género**:

- **mas**: para el masculino (en *palabras* y morfemas flexivos),
- **fem**: para el femenino (en *palabras* y morfemas flexivos)

---

[83] En nuestra gramática, por razones que explicaremos en su momento, incluso las formas con *nivel_0* tienen que pasar por las reglas morfológicas antes de continuar su procesamiento sintáctico.

- **mas/fem**: para ambos géneros (sólo en *palabras*)
- **no**: para palabras sin género (sólo en *raíces*)

Los valores **mas** y **fem** se asignan según el género gramatical de la palabra, sin atender a que ésta pueda designar indistintamente a varón o hembra[84] , como por ejemplo algunos femeninos en *-a* (*la víctima*) y algunos masculinos en *-a* (*los parias*) y en *-o* (*un vejestorio*). También se deben tratar así los nombres epicenos (*el gorila* es **mas**culino y *la ballena* es **fem**enino), y, en general, cualquier nombre que exija una concordancia gramatical determinada independientemente del sexo de la persona o animal al que se esté refiriendo (*el personaje, las autoridades...*). El valor **no** se asigna a las raíces para mantener la coherencia e insistir en que la información de género "proviene" del morfema flexivo. Por último, el valor **mas/fem** se aplica a aquellas *palabras* cuyo género se determina sólo por la concordancia gramatical, es decir, su forma aislada no permite decidir si es masculino o femenino. Es el caso de los nombres como *artista, periodista, testigo*, etc. y, sobre todo, de la mayoría de los adjetivos acabados en *-e* como *grande, verde*. Para estos casos, en el nivel morfológico no tenemos información suficiente para decidir cuál sea el género concreto de esa palabra. Necesitamos, por tanto, que esta información se asigne en el componente sintáctico, cuando se compruebe la concordancia interna dentro del sintagma nominal o la concordancia externa con un pronombre:

el / la testigo

la casa grande / el baúl grande

el / la artista inquietante

ella / él es lingüista

El tratamiento de estos casos depende de la aplicación posterior del análisis morfológico y de la forma en que las reglas sintácticas hacen esta comprobación. Hay varias soluciones posibles.

La primera es proporcionar dos análisis, uno con el valor **mas** y otro con el valor **fem**. La regla que formaría el SN se encargaría de elegir el análisis adecuado y rechazar el otro. Esta opción es la menos eficiente pues, por regla general, para resolver una ambigüedad se debe evitar duplicar los análisis en los niveles inferiores con el fin de impedir la expansión sucesiva.

Si buscamos la eficiencia lo mejor es dejar sin instanciar el valor del rasgo **género**. En algunas versiones de Prolog y en varios formalismos de unificación se utiliza para este fin el símbolo _ (subrayado o *underscore*, también conocido como *variable*

---

[84] Recordando la definición de la Academia:

> Decimos que un nombre es femenino o masculino cuando las formas respectivamente femeninas o masculinas del artículo y de algunos pronombres, caracterizadas las primeras por el morfema de género *-a*, y las segundas por el morfema de género *-o*, *-e* o por ningún morfema, se agrupan directamente con el sustantivo en construcción atributiva o aluden a él fuera de esta construcción. (R.A.E. 1973;1989:173).

anónima). Esto significa que el valor de ese rasgo no se conoce por el momento, pero que se asignará en el transcurso del procesamiento.[85] Esto es útil porque la ambigüedad se resuelve en el momento en que se dispone de la información necesaria (en este caso, el género de los especificadores y complementos del nombre), y mientras tanto no ha provocado sobregeneración.

Sin embargo, nosotros hemos escogido el valor **mas/fem** por motivos de la aplicación. En el programa que hemos escrito como demostración de nuestra gramática no se puede utilizar la *variable anónima* en la base de datos que funciona de diccionario (es un requerimiento de la versión de Prolog de que disponemos). Por otra parte, pensamos que es más "visual" que cuando se muestra el análisis de una palabra se proporcione **género** = **mas/fem** en lugar de **género** = _ .

En cualquier caso, es una cuestión convencional usar esos u otros valores. Lo importante es que con ellos se marca la diferencia de palabras con género conocido durante el proceso morfológico y las que necesitan la ayuda contextual de otros elementos oracionales para determinar su género.

6.2.1.1 *El rasgo tipo_gen*

Debido a que existen tres alomorfos del morfema de género (dos para el masculino, *-o, -e*; uno para el femenino, *-a*) necesitamos un rasgo especial para que el mecanismo de concatenación pueda escoger el adecuado a cada caso (al igual que tenemos **tipo_raíz** y **tipo_des** para el verbo). El rasgo **tipo_gen** tiene cuatro valores:

- **inh**: es el valor para las palabras con género inherente, es decir, que no llevan morfema de género.
- **mas1**: es el valor para las raíces que llevan *-o* como morfema de masculino
- **mas2**: es el valor para las raíces que llevan *-e* como morfema de masculino
- **fem**: es el valor para las raíces que llevan el morfema femenino *-a*.

La combinación de estos valores permite concatenar adecuadamente, por ejemplo, *niñ* con *o* (y no con *e*, **niñ- e*) y *president* con *e* (y no con *o*, **president-o*). Análogamente al uso de **tipo_des**, **tipo_gen** aparece el en morfema léxico (palabra o raíz) y en el flexivo. Si es una raíz (**nivel** -1) puede ser un rasgo complejo (con más de un valor en alternancia); en las palabras (**nivel_0**) y morfemas flexivos sólo puede tener un valor:

---

[85] El símbolo _ también significa en Prolog que no se tenga en cuenta el valor de ese rasgo o que se instancie con cualquier valor.

```
PALABRA
cadena = mano   nivel = nivel_0   gen = mas   tipo_gen = inh

RAÍCES
cadena = niñ   nivel = nivel_-1   gen = no   tipo_gen = mas1  fem

cadena = president   nivel = nivel_-1   gen = no   tipo_gen = mas2  fem


MORFEMAS DE GENERO
cadena = o   gen = mas   tipo_gen = mas1

cadena = e   gen = mas   tipo_gen = mas2

cadena = a   gen = fem   tipo_gen = fem
```

El mecanismo de concatenación comprueba que los valores de **tipo_gen** son idénticos para la raíz (una palabra nunca se une a un morfema de género; por eso se le asigna el valor **inh**) y el morfema de género, para concatenar seguidamente las cadenas superficiales. Con los datos del último ejemplo tendríamos las siguientes concatenaciones:

```
cadena = niñ  tipo_gen = mas1    +    cadena = o   tipo_gen = mas1 ==> cadena = niño

cadena = niñ  tipo_gen = fem     +    cadena = a   tipo_gen = fem  ==> cadena = niña

cadena = president  tipo_gen = mas2    +    cadena = e   tipo_gen = mas2  ==> cadena = presidente

cadena = president  tipo_gen = fem     +    cadena = a   tipo_gen = fem   ==> cadena = presidenta
```

Hay un caso en el que utilizamos **tipo_gen** con los valores **inh, fem**: en aquellas raíces donde el formante masculino no se manifiesta aunque sí el femenino, como por ejemplo *investigador, investigador - a, español, español - a.* La entrada para estos casos especiales cambia ligeramente con respecto a las raíces normales. El rasgo **género** tiene el valor **mas** en lugar de **no**. Es necesario para que la forma en masculino singular pueda llevar su información de género. En realidad, estos casos son tratados como un híbrido entre palabra y raíz. La razón de esto es evitar tener dos entradas con la misma cadena[86] . En otras palabras, se puede considerar que la entrada:

---

[86] Otra posibilidad sería considerarlas como *palabras* que pueden concatenarse también a morfemas de género (nuestra definición de *palabras* indica que éstas sólo pueden recibir el morfema de plural). Pero esto nos obligaría a tener dos reglas más en nuestra gramática, lo que aumentaría considerablemente el tiempo general de procesamiento. Preferimos, por tanto, "forzar" que la forma de masculino singular pase por la regla para *palabras* y las otras formas por sus correspondientes reglas para *raíces*. Conseguimos, además, una única entrada de diccionario en lugar de tener dos (*español* como palabra; *español* como raíz).

```
cadena = español   nivel = nivel_-1   gen = mas    tipo_gen = inh  fem
```

es la unificación de estas dos entradas:

```
cadena = español   nivel = nivel_0   gen = mas    tipo_gen = inh
cadena = español   nivel = nivel_-1  gen = no     tipo_gen = fem
```

En la explicación de la gramática se verá cómo se procesa este tipo de nominales.

6.2.2 El número

Al igual que la información de género, el número puede manifestarse como un formante morfológico específico o incluirse en la información de la entrada de diccionario. Análogamente, el segundo caso se da en las palabras (recordemos, son los elementos léxicos que llevan toda la información sintáctica necesaria para aparecer aisladamente en la superficie); las raíces necesitan concatenarse con morfemas flexivos.

Debido a que no existe un morfema de singular propiamente dicho,[87] esta información tiene que asignarse de alguna forma a las palabras resultantes de la combinación de la raíz y el morfema de género, como *niñ-o, niñ-a*. En los formalismos de unificación normalmente hay dos caminos: uno es por medio de reglas gramaticales (*adquisición*); el otro es por herencia léxica. La adquisición consiste en que al aplicar una determinada regla (en nuestro caso, la regla que une la raíz con el morfema de género) el rasgo pertinente se instancie en el constructo resultante (la palabra). La otra solución es que dicho rasgo se incluya en la entrada léxica y se transmita (en terminología de unificación, "heredar un rasgo") al constructo superior. Cualquiera de las dos soluciones es igual de eficiente y no supone ningún aumento extra para el diccionario. Debido al enfoque predominantemente lexicalista que hemos dado a nuestra gramática, preferimos la herencia del rasgo. Hay que tener en cuenta, además, que los nominales de género inherente (p. ej. *casa* o *lápiz*, de **nivel_0**) llevan necesariamente ese rasgo en su entrada (de igual forma que llevan el rasgo de género también). Por lo tanto, el rasgo **número** = **sg** siempre se instancia por herencia desde el componente nominal, y no desde el morfema flexivo ni tampoco por la regla.

Si bien la información de **número** = **singular** sólo puede aparecer en las categorías léxicas, la información de **número** = **plural** normalmente se trasmite por el morfema de plural. Hay unos cuantos casos excepcionales como *alicates, afueras, tijeras, ambages, víveres* etc. donde se da "la ausencia o el casi desuso del singular" (R.A.E. 1973;1989:186). Son los tradicionales *pluralia tantum*. En las entradas de estas palabras tenemos que especificar **número** = **pl**, pues de otra manera no se podría trasmitir la información, a no ser que se quisiera crear una entrada "irreal" de singular como *afuera, ambage, vívere* para que luego se combinara con el morfema flexivo. Esto

[87] Por motivos de composicionalidad, consideramos que los morfemas de género *o, e, a* sólo transmiten la información de género, de la misma forma que *s, es* sólo llevan información de número.

último nos parece artificial y costoso pues construye un plural a partir de un singular inexistente.

Paralelamente a la distribución de valores que vimos para el género, tenemos también un valor **no** para el rasgo **número**. El rasgo **número** = **no** aparece en todas las raíces surgidas por la discrepancia entre la forma gráfica del singular y del plural, por ejemplo *pez, pec - es.* En este caso el raso se asigna a la variante que se usa en el plural. Como esta raíz sólo se va a encontrar en combinación con el morfema de plural, no tiene sentido que especifiquemos **sg** o **pl** ya que en ningún caso se va a heredar de la raíz (como en *gat - o*) sino del morfema.

Queda por explicar un valor para **número**. El valor **sg/pl** se emplea para los casos de indistinción o sincretismo de singular y plural (*lunes, virus, crisis...*). Al igual que hacíamos con el rasgo **género** = **mas/fem**, dejamos que "el sincretismo se resuelva en la secuencia sintáctica con el auxilio de las formas de número diferenciadas de que están dotados la mayor parte de los pronombres y adjetivos" (R.A.E. 1973;1989:181)[88] .

6.2.2.1 *El rasgo tipo_plu*

El morfema de plural se realiza por medio de dos alomorfos que se diferencian por el valor del rasgo **tipo_plu** (rasgo para el mecanismo de concatenación):

```
cadena = s     número = plural     tipo_plu = plu1

cadena = es    número = plural     tipo_plu = plu2
```

De nuevo, este rasgo **tipo_plu** sirve para escoger la variante apropiada y se incluye también en la entrada de la palabra o raíz nominal:

```
cadena = luna   nivel = nivel_0   gen = fem   num = sg  tipo_gen = inh  tipo_plu = plu1

cadena = sol    nivel = nivel_0   gen = mas   num = sg  tipo_gen = inh   tipo_plu = plu2

cadena = gat    nivel = nivel_-1  gen = no    num = sg  tipo_gen = mas1 fem   tipo_plu = plu1
```

Con esta codificación se da cuenta de *luna -s, sol -es, gat -o -s* y *gat -a -s.*

Hay un tercer valor posible para **tipo_plu**: es **noplu**. Se asigna a aquellas palabras que no pueden ser unidas a ninguno de los alomorfos de plural, como por ejemplo las palabras que son invariables con respecto al número (*virus, lunes*)[89] o como las palabras que sólo se utilizan en plural (*alicates, afueras*). También se utiliza para aquellas palabras (como *león, inglés, pez*) cuyos plurales no se pueden formar a partir de las cadenas del singular debido a que se produce un cambio gráfico: *leon - es, ingles - es,*

---

[88] De igual forma, aquí se podría utilizar la *variable anónima* _, pero por las mismas razones que en el caso de **género** = **mas/fem** preferimos **número** = **sg/pl**.

[89] Este uso del valor **noplu** equivale al morfema ∅ de muchas clasificaciones, pero no debe confundirse con él porque **noplu** tiene más usos, como demuestran los ejemplos.

pec - es. Esto provoca inevitablemente dos entradas, una para cada cadena, y con rasgos muy distintos:

```
cadena = pez  nivel = nivel_0  gen = mas  num = sg  tipo_gen = inh  tipo_plu = noplu

cadena = pec  nivel = nivel_-1  gen = mas  num = no  tipo_gen = inh  tipo_plu = plu2
```

Resumiendo, los valores posibles de **número** son:

1. **sg**: singular (en palabras y raíces)
2. **pl**: plural (en palabras *pluralia tantum* y morfemas de plural)
3. **sg/pl**: singular/plural (en palabras invariables)
4. **no**: no es pertinente ni singular ni plural (en raíces para plurales con cambios gráficos con respecto a la forma de singular)

Y los valores para **tipo_plu**:

1. **plu1**: plurales en *-s*,
2. **plu2**: plurales en *-es*,
3. **noplu**: ninguna marca de plural.

6.2.3 Paradigmas nominales

Los paradigmas que mostramos a continuación, al igual que los paradigmas del verbo, tienen como objetivo ayudar a establecer las cadenas superficiales de una unidad léxica y la información asignada a cada cadena (en otras palabras, las entradas del diccionario). No hay que asociarlos, por tanto, con el concepto de *paradigma* del modelo PP. Organizaremos la clasificación siguiendo el número de formas flexionadas y la asignación de rasgos. De nuevo, esta clasificación se articula en torno a la forma gráfica, sin tener en cuenta los procesos morfofonológicos.

6.2.3.1 Paradigma nominal 1

Este modelo incluye todas las unidades léxicas (para las distintas categorías nominales: nombre, adjetivo, artículo, pronombre, etc.) que presenten cuatro formas flexionadas: **masculino singular, femenino singular, masculino plural** y **femenino plural.** Esto traducido a nuestro sistema de rasgos es:

```
nivel = nivel_0  unidad_léx = X  categoría = Y   género = mas  número = sg

nivel = nivel_0  unidad_léx = X  categoría = Y   género = fem  número = sg

nivel = nivel_0  unidad_léx = X  categoría = Y   género = mas  número = pl

nivel = nivel_0  unidad_léx = X  categoría = Y   género = fem  número = pl
```

Dentro de este paradigma hay varios subgrupos, dependiendo del número de entradas de diccionario que necesite la unidad léxica.

El primer subgrupo lo forman aquellos lexemas que sólo tienen un alomorfo, como *niñ- o, a, os, as; president- e, a, es, as; bonit- o, a, os, as; segund- o, a, os, as.* El modelo de entrada de diccionario es como sigue[90] :

```
cadena = C  nivel = nivel_-1  unidad_léx = X   categoría = Y
   género = no  número = sg  tipo_gen = mas1 | mas2  fem   tipo_plu = plu1
```

y los ejemplos de arriba (a partir de ahora damos los rasgos en su forma abreviada)

```
cadena = niñ  niv = nivel_-1  lex = niño  cat = n  gen = no  num = sg
              tipo_gen = mas1 fem  tipo_plu = plu1

cadena = president  niv = nivel_-1  lex = presidente  cat = n  gen = no  num = sg
              tipo_gen = mas2 fem  tipo_plu = plu1

cadena = bonit  niv = nivel_-1  lex = bonito  cat = adj  gen = no  num = sg
              tipo_gen = mas1 fem  tipo_plu = plu1

cadena = segund  niv = nivel_-1  lex = segundo  cat = ord  gen = no  num = sg
              tipo_gen = mas1 fem  tipo_plu = plu1
```

El segundo subgrupo está formado por unidades léxicas cuya forma de masculino singular coincide con la cadena superficial que se utiliza como raíz para las otras formas, como es el caso de los nominales acabados en *-or* (*doctor, -a, -es, -as*; *español, -a, -es, -as*). Para estos casos se utiliza el siguiente modelo de entrada:

```
cadena = C  niv = nivel_-1  lex = X  cat = Y  gen = mas  num = sg
              tipo_gen = inh fem  tipo_plu = plu2
```

Por lo tanto, la entrada para *doctor* será

---

[90] La barra que separa **mas1** de **mas2** significa que estos dos valores están en alternancia.

```
cadena = doctor  niv = nivel_-1  lex = doctor  cat = n  gen = mas  num = sg
                 tipo_gen = inh fem  tipo_plu = plu2
```

Un tercer grupo dentro de este paradigma lo forman los nominales que tienen dos cadenas alternantes, debido a un cambio gráfico. En este grupo se incluyen todos los nominales que cambian la vocal acentuada en plural (*león, inglés, holgazán, mallorquín...*). Este grupo está muy relacionado con el anterior (el masculino singular no lleva tampoco morfema de género) pero, a diferencia de los acabados en *-or, -ol*, etc., su cadena superficial no puede ser utilizada para las otras formas flexionadas (**león - a, -es, -as*). Esto nos obliga a tratar dicha forma como una *palabra* (**nivel_0**) que no puede llevar morfema de plural (**tipo_plu = noplu**). La otra cadena, en cambio, es una *raíz* (**nivel_-1**) con los rasgos **mas, tipo_gen = inh** y **plu2** para la forma masculino plural, y el rasgo **tipo_gen = fem** para las formas femeninas. El rasgo **num = sg** lo "hereda" exclusivamente la forma de femenino singular[91] :

```
cadena = C  niv = nivel_0  lex = X  cat = Y  gen = mas  num = sg
            tipo_gen = inh  tipo_plu = noplu

cadena = C  niv = nivel_-1  lex = X  cat = Y  gen = mas  num = sg
            tipo_gen = inh fem  tipo_plu = plu2


cadena = león  niv = nivel_0  lex = león  cat = n  gen = mas  num = sg
            tipo_gen = inh  tipo_plu = noplu

cadena = leon  niv = nivel_-1  lex = león  cat = n  gen = mas  num = sg
            tipo_gen = inh fem  tipo_plu = plu2
```

Como mencionamos al explicar los valores de **tipo_gen**, la entrada para la cadena *leon* ciertamente es un híbrido entre *raíz* y *palabra*. Nos hemos permitido esta excepción en la codificación para no tener dos entradas con la misma cadena (en el caso de *leon* una sería para la forma de masculino plural, y la otra para las formas de femenino singular y plural). De esta manera, una única entrada puede ser utilizada por dos reglas, pero sin interferir entre sí.

Un caso aún más excepcional presentan los nominales con cuatro formas cuya forma base acaba en *-z*, como *andaluz, andaluz -a, andaluc -es, andaluz -a -s*. En esta ocasión tenemos también dos cadenas superficiales para la lematización, que codificamos de la siguiente manera:

---

[91] El valor de **num = pl** para la forma de femenino plural se transmite desde el morfema de plural (*leon -a -s*). Para más detalles, véase la explicación de la gramática.

```
cadena = C  niv = nivel_-1  lex = X  cat = Y  gen = mas  num = sg
            tipo_gen = inh fem  tipo_plu = noplu

cadena = C  niv = nivel_-1  lex = X  cat = Y  gen = mas  num = no
            tipo_gen = inh  tipo_plu = plu2


cadena = andaluz  niv = nivel_-1  lex = andaluz  cat = n  gen = mas  num = sg
            tipo_gen = inh fem  tipo_plu = noplu

cadena = andaluc  niv = nivel_-1  lex = andaluz  cat = Y  gen = mas  num = no
            tipo_gen = inh  tipo_plu = plu2
```

Nuestro formalismo permite dar cuenta también de otros casos muy particulares, con la cantidad mínima imprescindible de entradas. Por ejemplo, los casos en los que el femenino es distinto del masculino. Son bastante especiales y, aunque de número reducido, tienen un uso muy frecuente. Hay que distinguir dos grupos: los que utilizan una terminación productiva que se añade o se permuta por parte de la raíz masculina, y los que utilizan palabras completamente diferentes:

1. terminación
   - **-esa**: *príncipe, duque, abad... princesa, duquesa, abadesa*
   - **-isa**: *poeta, profeta, sacerdote... poetisa, profetisa, sacerdotisa*
   - **-ina**: *rey, héroe, gallo, jabalí... reina, heroína, gallina, jabalina*
   - **-triz**: *emperador, actor... emperatriz, actriz*
2. palabras diferentes: *toro / vaca, padre / madre...*

La estrategia en estos casos es "agrupar" las distintas cadenas mediante el rasgo de **unidad_léxica**. Así, por ejemplo, podríamos expresar conceptualmente que *toro* es la forma masculina y *vaca* es la forma femenina de una misma unidad léxica. Por supuesto, no estamos sugiriendo que ésa sea la mejor solución, simplemente que con el formalismo que utilizamos se puede establecer esa codificación. La otra posibilidad es tratar *toro* y *vaca* como entidades distintas, de género inherente.

Parece, en cambio, que para las palabras que tienen una parte del radical en común es más intuitivo agruparlas dentro de la misma **unidad léxica**. Para ello, distinguimos las dos cadenas de la siguiente manera:

```
cadena = C  niv = nivel_0  lex = X  cat = Y  gen = mas  num = sg
            tipo_gen = inh  tipo_plu = plu2

cadena = C  niv = nivel_0  lex = X  cat = Y  gen = fem  num = sg
            tipo_gen = inh  tipo_plu = plu1


cadena = rey  niv = nivel_0  lex = rey  cat = n  gen = mas  num = sg
            tipo_gen = inh  tipo_plu = plu2

cadena = reina  niv = nivel_0  lex = rey  cat = n  gen = fem  num = sg
            tipo_gen = inh  tipo_plu = plu1
```

Otra particularidad que presentan algunos nominales del español es la de tener dos formas alternantes[92] :

este / esto	esta	estos	estas
buen / bueno	buena	buenos	buenas
actor	actriz / actora	actores	actrices / actoras
jabalí	jabalina	jabalíes / jabalís	jabalinas

y ésta es una codificación posible[93] en nuestro formalismo:

```
cadena = este  niv = nivel_0  lex = este  cat = dem  gen = mas  num = sg
            tipo_gen = inh  tipo_plu = noplu

cadena = est  niv = nivel_-1  lex = este  cat = dem  gen = no  num = sg
            tipo_gen = mas1 fem  tipo_plu = plu1
```

---

[92] En algunos casos, una de las formas es bastante vulgar, como *jabalís*. En el caso de la alternancia *actriz / actora*, *actriz* corresponde al sentido de persona que representa un papel en cine o teatro, mientras que por *actora* se entiende demandante o acusador ante los tribunales (Seco 1986). Depende de la aplicación el elegir las dos variantes, o sólo la que más convenga.

[93] Decimos una codificación posible porque hay distintas opciones. En los ejemplo tanto *reina* como *jabalina* aparecen tratadas como *palabras*, pero también es posible codificarlas como *raíces*: *rein - a, jabalín - a.* El tratamiento como *palabra* es más rápido.

```
cadena = buen  niv = nivel_0  lex = bueno  cat = adj  gen = mas  num = sg
              tipo_gen = inh  tipo_plu = noplu

cadena = buen  niv = nivel_-1  lex = bueno  cat = adj  gen = no  num = sg
              tipo_gen = mas1 fem  tipo_plu = plu1


cadena = actor  niv = nivel_0  lex = actor  cat = n  gen = mas  num = sg
              tipo_gen = inh fem  tipo_plu = plu2

cadena = actriz  niv = nivel_0  lex = actor  cat = n  gen = fem  num = sg
              tipo_gen = fem  tipo_plu = noplu

cadena = actric  niv = nivel_-1  lex = actor  cat = n  gen = fem  num = no
              tipo_gen = inh  tipo_plu = plu2


cadena = jabalí  niv = nivel_0  lex = jabalí  cat = n  gen = mas  num = sg
              tipo_gen = inh  tipo_plu = plu1 plu2

cadena = jabalina  niv = nivel_0  lex = jabalí  cat = n  gen = fem  num = sg
              tipo_gen = inh  tipo_plu = plu1
```

6.2.3.2 *Paradigma nominal 2*

Aquí incluimos todos las unidades léxicas nominales que presentan dos formas, una para el singular y otra para el plural. La información de género es inherente a la unidad léxica, pues no se transmite desde ningún morfema específico. El morfema de número **plural** es el único formante morfológico que admiten. Este paradigma es el más extendido en el léxico del castellano, sobre todo en los nombres (donde se aplica a más del 80%). El caso más típico es el de una única entrada:

```
cadena = C  niv = nivel_0  lex = X  cat = Y  gen = mas  num = sg  tipo_gen = inh  tipo_plu = P

cadena = C  niv = nivel_0  lex = X  cat = Y  gen = fem  num = sg  tipo_gen = inh  tipo_plu = P


cadena = sol  niv = nivel_0  lex = sol  cat = n  gen = mas  num = sg
              tipo_gen = inh  tipo_plu = plu2

cadena = luna  niv = nivel_0  lex = luna  cat = n  gen = fem  num = sg
              tipo_gen = inh  tipo_plu = plu1
```

Aunque también es muy habitual el modelo de dos entradas, debido al cambio gráfico en la cadena de plural (*pez, pec -es; ilusión, ilusion -es*):

```
cadena = C  niv = nivel_0  lex = X  cat = Y  gen = G  num = sg  tipo_gen = inh  tipo_plu = noplu

cadena = C  niv = nivel_-1  lex = X  cat = Y  gen = G  num = no  tipo_gen = inh  tipo_plu = plu2


cadena = pez  niv = nivel_0  lex = pez  cat = n  gen = mas  num = sg
             tipo_gen = inh  tipo_plu = noplu

cadena = pec  niv = nivel_-1  lex = pez  cat = n  gen = mas  num = no
             tipo_gen = inh  tipo_plu = plu2

cadena = ilusión  niv = nivel_0  lex = ilusión  cat = n  gen = fem  num = sg
             tipo_gen = inh  tipo_plu = noplu

cadena = ilusion  niv = nivel_-1  lex = ilusión  cat = n  gen = fem  num = no
             tipo_gen = inh  tipo_plu = plu2
```

Dentro de este esquema nos encontramos con todos los nominales de género inherente acabados en *-z* y en *-n* o *-s* con vocal anterior acentuada. También se incluyen aquí los cambios de acentuación como *régimen, regímen -es; examen, exámen - es; carácter, caracter -es*:

```
cadena = examen  niv = nivel_0  lex = examen  cat = n  gen = mas  num = sg
             tipo_gen = inh  tipo_plu = noplu

cadena = exámen  niv = nivel_-1  lex = examen  cat = n  gen = mas  num = no
             tipo_gen = inh  tipo_plu = plu2

cadena = régimen  niv = nivel_0  lex = régimen  cat = n  gen = mas  num = sg
             tipo_gen = inh  tipo_plu = noplu

cadena = regimen  niv = nivel_-1  lex = régimen  cat = n  gen = mas  num = no
             tipo_gen = inh  tipo_plu = plu2
```

Normalmente casi todos los extranjerismos son de este tipo, es decir, género inherente y una forma para el singular y otra para el plural. Algunos se adaptan perfectamente a la grafía española, como *estándar, estándar -es*, con lo cual sólo necesitamos una sola entrada siguiendo el esquema de *sol*. Es bastante corriente, sin embargo, que las formas de plural difieran gráficamente como por ejemplo *lord, lor -es; hipérbaton, hipérbato - s; jersey, jerséi - s*. En estos casos escribimos dos entradas:

```
cadena = lord  niv = nivel_0  lex = lord  cat = n  gen = mas  num = sg
              tipo_gen = inh  tipo_plu = noplu

cadena = lor  niv = nivel_-1  lex = lord  cat = n  gen = mas  num = no
              tipo_gen = inh  tipo_plu = plu2

cadena = jersey  niv = nivel_0  lex = jersey  cat = n  gen = mas  num = sg
              tipo_gen = inh  tipo_plu = noplu

cadena = jerséi  niv = nivel_-1  lex = jerséi  cat = n  gen = mas  num = no
              tipo_gen = inh  tipo_plu = plu1

cadena = hipérbaton  niv = nivel_0  lex = hiperbáton  cat = n  gen = mas  num = sg
              tipo_gen = inh  tipo_plu = noplu

cadena = hipérbato  niv = nivel_-1  lex = hiperbáton  cat = n  gen = mas  num = no
              tipo_gen = inh  tipo_plu = plu1
```

Por otra parte, los nominales acabados en *-á, -í, -ú* suelen presentar una vacilación en sus formas de plural (*sofá -s, sofá -es; maniquí -s, maniquí -es; bambú -s, bambú -es*). El uso culto prefiere normalmente el alomorfo *-es* (**plu2**) mientras que en el habla coloquial se utiliza *-s* (**plu1**) (R.A.E. 1973; 1989:184). Nuestro formalismo permite dar cuenta de ambas variantes con una sóla entrada gracias a la posibilidad de usar valores en disyunción para el rasgo **tipo_plu**:

```
cadena = maniquí  niv = nivel_0  lex = maniquí  cat = n  gen = mas/fem  num = sg
              tipo_gen = inh  tipo_plu = plu1 plu2
```

también se pueden tratar las tres formas de plural alternantes de *maravedí, maravedí -s, maravedí -es, maravedís -es*:

```
cadena = maravedí  niv = nivel_0  lex = maravedí  cat = n  gen = mas  num = sg
              tipo_gen = inh  tipo_plu = plu1 plu2

cadena = maravedís  niv = nivel_-1  lex = maravedí  cat = n  gen = mas  num = no
              tipo_gen = inh  tipo_plu = plu2
```

Dentro de este modelo de dos formas hay un subgrupo bastante numerosos constituido por palabras nominales con género morfológicamente indeterminado. Nos estamos refiriendo a los nombres como *artista* pero también a los llamados *adjetivos invariables* (R.A.E. 1973;1989:191-2):

- los que terminan en *-a* en singular: *persa, malva, agrícola,*
- los acabados en *-í, -ú* en singular: *alfonsí, hindú,*
- la mayor parte de los acabados en *-e*: *agradable, independiente, breve*
- la mayor parte de los acabados en consonante: *audaz, elemental, azul, joven.*

Estas palabras generalmente sólo presentan un cadena superficial (con la excepción de los acabados en *-í, -ú* y en *-z*) y su entrada de diccionario se distingue por llevar **gen = mas/fem**:

```
cadena = artista  niv = nivel_0  lex = artista  cat = n  gen = mas/fem  num = sg
             tipo_gen = inh  tipo_plu = plu1

cadena = grande  niv = nivel_0  lex = grande  cat = adj  gen = mas/fem  num = sg
             tipo_gen = inh  tipo_plu = plu1
```

6.2.3.3 *Paradigma nominal 3*

Por último nos encontramos con las palabras que presentan una única forma, ya sea porque utilizan la misma para el singular y el plural, ya sea porque carecen de singular o plural. De cualquier manera, este modelo de flexión nominal es mucho menos corriente que los dos anteriores y formado casi exclusivamente por nombres.

El primer grupo lo constituyen las palabras de número indeterminado como *virus* y *crisis*. Sus rasgos distintivos son **num = sg/pl** y **tipo_plu = noplu**. Además, son de género inherente, es decir, llevan el rasgo de género en la entrada de diccionario:

```
cadena = virus  niv = nivel_0  lex = virus  cat = n  gen = mas  num = sg/pl
             tipo_gen = inh  tipo_plu = noplu

cadena = crisis  niv = nivel_0  lex = crisis  cat = n  gen = fem  num = sg/pl
             tipo_gen = inh  tipo_plu = noplu
```

El segundo grupo está compuesto por los nombres conocidos en la gramática tradicional como *pluralia* y *singularia tantum*, es decir, los nombres empleados exclusivamente o casi exclusivamente en plural (*bienes, maitines, alforjas, matemáticas*) o en singular (*estrés, informática*). En estos casos la entrada lleva los rasgos de **género** y **número** instanciados, además de **tipo_gen = inh** (para evitar que se concatene con un morfema de género) y **tipo_plu = noplu** (para impedir la unión con un morfema de número). En cierta medida, se puede decir que la palabra está "lexicalizada":

```
cadena = bienes  niv = nivel_0  lex = bienes  cat = n  gen = mas  num = pl
             tipo_gen = inh  tipo_plu = noplu

cadena = matemáticas  niv = nivel_0  lex = matemáticas  cat = n  gen = fem  num = pl
             tipo_gen = inh  tipo_plu = noplu

cadena = estrés  niv = nivel_0  lex = estrés  cat = n  gen = mas  num = sg
             tipo_gen = inh  tipo_plu = noplu

cadena = informática  niv = nivel_0  lex = informática  cat = n  gen = fem  num = sg
             tipo_gen = inh  tipo_plu = noplu
```

Hay un grupo verdaderamente muy reducido de adjetivos con género y número indeterminados: *chico / chica / chicos / chicas / rubiales, mochales* (R.A.E. 1973;1989:194)[94] . Aquí la entrada como valores de **género** y **número** lleva respectivamente **mas/fem** y **sg/pl**:

```
cadena = rubiales  niv = nivel_0  lex = rubiales  cat = adj  gen = mas/fem  num = sg/pl
                tipo_gen = inh  tipo_plu = noplu
```

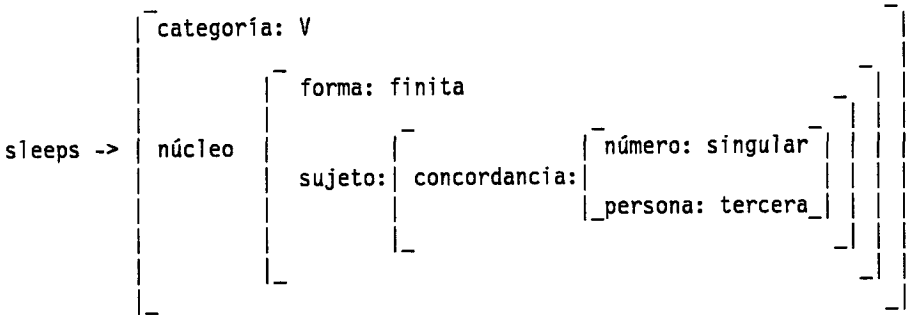
[94] Hemos encontrado solamente *vivales, frescales, finolis* y *tiquismiquis*. También se pueden incluir aquí los compuestos que se utilizan para oficios o tareas que pueden ser desempeñados por ambos sexos como *un / una / unos / unas lavacoches* o compuestos despectivos como *un / una pelagatos*.

7.0 La gramática

En nuestra exposición vamos a proceder en primer lugar a la definición del formalismo. En concreto, hablaremos de los rasgos utilizados y sus valores posibles; seguidamente expondremos una regla modelo de la gramática, que servirá de base para explicar los mecanismos de concatenación y unificación. Por último, analizaremos cada una de las reglas de la gramática, en su versión de unificación de términos.

7.1 *El formalismo*

Como es conocido, los formalismos de unificación se caracterizan por la asociación de cadenas superficiales con elementos informativos o rasgos. Estos dos componentes de cualquier estructura lingüística habitualmente se representan por separado aunque con alguna marca que especifique la relación entre ellos. Es decir, las cadenas o bien en una estructura arbórea (por ejemplo en Johnson(1988:5)) o bien como cadenas sin más, que se asocian a una estructura de rasgos. Mostramos un ejemplo tomado de Shieber(1986;1989:27):



Esta estructura lingüística (propia del formalismo PATR-II) equivaldría[95] en nuestro formalismo a:

[95] Téngase en cuenta que el ejemplo incorpora una información sintáctico-funcional que nosotros no pretendemos en ningún momento, pero mantenemos dicha información para dar una idea general de la relación que existe con otros formalismos de unificación

```
palabra(cadena = sleeps, categoría = v,
        núcleo(forma = finita, sujeto(concordancia(número = singular,persona = tercera))))
```

Es decir, utilizando una estructura propia de un formalismo basado en Prolog, la cadena se incluye como un rasgo (o argumento) más del *objeto* resultante. Como hemos ido avanzado, el diccionario contiene las asociaciones primitivas entre cadenas terminales y sus estructuras de rasgos:

```
raíz_nom(cadena = niñ, unidad_léx = niño, ... )

morf_gen(cadena = a, género = fem,...)
```

Recordemos que las estructuras de rasgos pueden ser de dos tipos:

1. pueden ser estructuras *atómicas* (i.e. un valor constante único, como **singular**), o
2. estructuras *complejas* (i.e. con atributos y valores, ordenados en una estructura jerárquica).

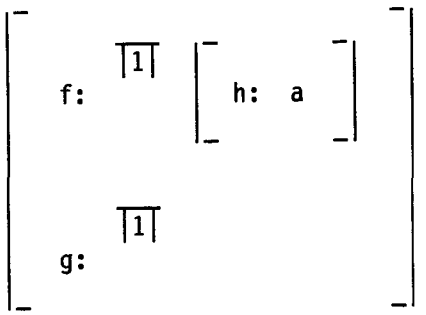
El ejemplo de *sleeps* muestra los dos tipos. Por una parte, **categoría = v** es un rasgo *atómico*, por otra, **núcleo(forma = finita, sujeto(concordancia(número = singular, persona = tercera)))** es un rasgo *complejo*.

Una particularidad de los formalismos de unificación es la utilización de convenciones acerca de rasgos, como el *valor compartido* o como *disyunción de valores*. El uso de estas convenciones, que podríamos denominar de mejora notacional[96] , permiten ampliar de manera general la expresividad del formalismo. Por ejemplo, cuando existe un valor que es compartido por dos o más atributos éste "se escribe sólo una vez y a su izquierda se añade un recuadro coindexado que lo identifica; los otros rasgos que comparten este mismo valor tiene en su lugar únicamente el recuadro coindexado" (Shieber 1986;1989:13-14)[97] :

---

[96] La lingüística recurre habitualmente a este tipo de convenciones para evitar descripciones engorrosas. Recuérdese especialmente que Peters y Ritchie (1973) demostraron que las reglas dependientes del contexto no es más que una simplificación notacional de las reglas independientes del contexto. De igual forma, el uso de ( ) y { } para indicar la opcionalidad y la alternancia, respectivamente, está muy extendido entre la comunidad lingüística.

[97] Esta notación se desarrolló para distinguir entre rasgos con un único valor (valor compartido) y rasgos con valores distintos pero semejantes. Para más detalles, consúltese Shieber(1986).



La *disyunción* o *alternancia* en los valores (o también en las estructuras de rasgos) se introdujo en algunos formalismos (FUG, por ejemplo) mediante la notación entre llaves ({}). Con ella se indica que la unificación sólo debe aplicarse a uno de los valores del conjunto. Por ejemplo, para expresar que la palabra *artista* puede llevar ambos géneros, y que en función de las condiciones de concordancia se escoja el apropiado, podríamos utilizar una alternancia de valores:

```
género = {masculino femenino}
```

Hay otros signos gráficos (como | o ;) que se suelen utilizar para marcar la disyunción. Con ello conseguimos especificar en una sola regla condiciones de aplicación que de otra forma se expresarían en varias reglas por separado. En un formalismo sin disyunción, las restricciones que imponen los rasgos deben cumplirse todas simultáneamente para que se aplique la regla (es decir, operan en *conjunción*). Con la disyunción podemos expresar varias reglas en una, pues se indica que para un determinado rasgo se utilice cualquiera (pero sólo uno) de los valores especificados. Naturalmente, esto implica que el concepto de unificación de rasgos debe ser ampliado y que nuestro formalismo ha aumentado su poder expresivo.

Efectivamente, todas las entradas de diccionario que hemos visto hasta ahora (menos las formas lexicalizadas) contienen algún rasgo con *disyunción de valores*. Concretamente, **tipo_raíz, tipo_des, tipo_gen** y **conjugación** pueden llevar como valor una lista de uno o más elementos:

```
tipo_raíz = 11 51 52 53 54...

tipo_des = pret1 part1...

tipo_gen = mas1 fem.

tipo_plu = plu1 plu2.

conjugación = cjg1 cjg2 cjg3
```

No hemos utilizado ninguna convención gráfica especial para señalar este tipo de disyunción, simplemente se codifica con un blanco entre valores distintos. Esta notación simplifica las entradas de diccionario en el sentido de que agrupa en una sola

todas aquellas que comparten los mismos pares atributo-valor y se diferencian únicamente en el valor de **tipo_raíz, tipo_des, tipo_plu** o **tipo_gen**. En este sentido, se puede decir que

```
raíz(cadena = pong, nivel = -1, lex = poner, tipo_raíz = 11 51 52 53 54 55 56, tipo_des = reg)
```

es una reducción notacional de

```
raíz(cadena = pong, nivel = -1, lex = poner, tipo_raíz = 11, tipo_des = reg)
raíz(cadena = pong, nivel = -1, lex = poner, tipo_raíz = 51, tipo_des = reg)
raíz(cadena = pong, nivel = -1, lex = poner, tipo_raíz = 52, tipo_des = reg)
raíz(cadena = pong, nivel = -1, lex = poner, tipo_raíz = 53, tipo_des = reg)
raíz(cadena = pong, nivel = -1, lex = poner, tipo_raíz = 54, tipo_des = reg)
raíz(cadena = pong, nivel = -1, lex = poner, tipo_raíz = 55, tipo_des = reg)
raíz(cadena = pong, nivel = -1, lex = poner, tipo_raíz = 56 tipo_des = reg)
```

Antes de proseguir, creemos necesario citar un comentario de Shieber acerca del poder expresivo de los formalismos:

> El poder expresivo tiene un valor opuesto según lo que se pretenda. Si para la elaboración de una herramienta descriptiva o computacional se busca el máximo poder expresivo, cuando se trata de establecer una teoría lingüística universal un gran poder expresivo es un inconveniente. Así, objetivos diferentes pueden conllevar elecciones diferentes en el momento de diseñar un formalismo [...] algunas de las teorías que presentamos (de manera especial, GPSG y LFG) se interesan especialmente en los universales lingüísticos; para ellas el poder expresivo tiene que ser limitado, no promovido. (Shieber 1986;1989:6)

A nuestro juicio, estas palabras expresan certeramente por qué la disyunción de valores no está disponible en la mayoría de las teorías lingüísticas. El concepto de *restrictividad* (es decir, el hecho de que la Gramática Universal sea lo suficientemente restrictiva en cuanto a las opciones que permita, sobre la base psicológica de que el niño desarrolla una gramática partiendo de unos datos muy limitados) está aceptado en la mayoría de las teorías actuales. Aunque, como señala Shieber(1988), las restricciones a la Gramática Universal pueden ser de naturaleza formal o teórica[98] , la mayor parte del peso de la restrictividad recae en el formalismo. Se explica así que se prefiera no utilizar la disyunción de valores, mecanismo formal que implica una complejidad mayor que el uso de valores únicos y un aumento del poder expresivo.

---

[98] Es decir, propias de una teoría particular, como por ejemplo la teoría universal del control y la complementación en LFG.

Por otra parte, los formalismos orientados al procesamiento del lenguaje natural, como FUG o el nuestro, necesitan ser suficientemente expresivos. En primer lugar, porque ya están restringidos por las limitaciones que imponen los ordenadores como es el hecho de que la información tenga que expresarse en forma lineal y el proceso sea generalmente secuencial (tanto de *hardware* como de *software*) y, por lo tanto, necesitan de alguna forma "compensar" las limitaciones para mejorar la eficiencia. En este sentido, los esfuerzos se centran en reducir el tamaño de las gramáticas y los diccionarios mediante notaciones como la que mostramos más arriba. En segundo lugar, estos formalismos deben permitir a los lingüistas computacionales escribir gramáticas y diccionarios de la manera más cómoda posible. De igual manera, las notaciones como la disyunción permiten escribir reglas y entradas más compactas. Hay que tener en cuenta que los lingüistas teóricos rara vez se enfrentan a la tarea de construir gramáticas "enteras," sino que normalmente se dedican a elaborar partes significativas y dejar el desarrollo completo simplemente esbozado. En el ejemplo de la entradas de diccionario para la cadena *pong-*, con valores únicos, un lingüista teórico normalmente presentaría una como modelo y las demás las explicaría por analogía. Un lingüista computacional no puede hacer eso (a no ser que quiera desarrollar una gramática incompleta). Está obligado a codificar *explícitamente* todas las entradas y reglas necesarias.

A pesar de todo lo que hemos dicho sobre aumentar el poder expresivo en los formalismos orientados al procesamiento de lenguas naturales, hay que tener en cuenta que todo aumento formal siempre implica un aumento en el programa. Por tanto, hay que sopesar las ventajas y desventajas de toda extensión formal. Por ejemplo, en nuestro formalismo no nos interesa extender la posibilidad de usar la disyunción de valores a otros rasgos de los mencionados, como **número = sg pl** o **género = mas fem**, debido a que esto nos obligaría a complicar notablemente nuestro programa, en su estado actual[99] .

En consecuencia, aplicaremos dos restricciones al uso de la disyunción de valores, de tal forma que sólo pueda ser utilizada para los cuatro rasgos mencionados. Las restricciones son:

1. los rasgos complejos con disyunción de valores sólo se pueden usar en el mecanismo de concatenación morfológica, y
2. la información de estos rasgos no se transmite al nodo superior, por lo tanto, no participa del proceso general de unificación.

Estas restricciones nos permiten distinguir entre dos tipos de rasgos en nuestro formalismo:

- **rasgos con información gramatical** (sintáctica)
  - son rasgos **atómicos** y de **valor único**
  - se utilizan en el proceso de **unificación**
- **rasgos con información contextual** (morfológica)

---

[99] Esto no quiere decir que si cambiamos los objetivos del programa o, simplemente, si se codifica para ser utilizado en ordenadores más potentes no sea práctico extender la disyunción. Como hemos mencionado, un formalismo orientado al procesamiento de lenguas naturales está obligado a ser eficiente computacionalmente.

- son rasgos **complejos** y con **disyunción de valores**
- se utilizan en el proceso de **concatenación** morfológica.

Normalmente los formalismos y gramáticas basados en rasgos imponen una restricción general que es que cada estructura de rasgos debe contener un conjunto finito de atributos y valores. Esto se conoce con el término de *declaración de rasgos.* Según Johnson(1988:23), "some restriction of this kind is motivated by psychological considerations: surely with finite resources only finite structures can be represented, hence a psychologically realistic theory should admit only finite structures."

En la figura de la página 200 damos la declaración de atributos y sus valores permitidos por nuestra gramática. En la exposición distinguiremos entre rasgos gramaticales y contextuales y también entre rasgos de valor único y rasgos con disyunción de valores. Simplemente a efectos expositivos de esta declaración utilizaremos la coma para separar **valores únicos**:

```
rasgo = valor1, valor2...    p.ej.  gen = mas, fem, no, mas/fem
```

y el espacio en blanco para separar **valores en disyunción**:

```
rasgo = valor1 valor2...    p.ej.  tipo_gen = mas1 mas2 fem inh
```

Hasta el momento hemos utilizados varios términos para referirnos al mismo rasgo (p. ej., **número, num; unidad_léx, lex**, etc.). La declaración de rasgos servirá también a partir de ahora para unificar nuestra terminología.

7.1.1 Una nota sobre la codificación del diccionario

El poder expresivo del formalismo es excesivo para codificar algunas entradas de diccionario, en la forma que se expusieron en la parte descriptiva. La expresividad es interesante porque permite tratar con una misma entrada las variantes gráficas como *jabalís, jabalíes*:

```
nom(cadena = jabalí, nivel = nivel_0,... ,tipo_gen = inh, tipo_plu = plu1 plu2)
```

El rasgo **tipo_plu** tiene dos valores en disyunción, de tal manera que con la misma entrada se puede generar y analizar un plural con *-s* (plu1) y otro con *-es* (plu2).

Este recurso notacional, en cambio, sobregenera o sobreanaliza palabras si la codificación del diccionario incluye rasgos incompatibles. Por ejemplo, supongamos que queremos escribir una entrada de diccionario para el demostrativo *este* tal que

```
nom(cadena = est, nivel = nivel_-1, ... , tipo_gen = mas2 mas1 fem, tipo_plu = plu1)
```

```
/*  Declaración de rasgos  */

/*  RASGOS GRAMATICALES:   */

nivel = nivel_0, nivel_-1

lex = CUALQUIER VALOR

cat = nombre, adj, art, dem, pro_intg, pro_rel, ordin

pers_num = sg_1, sg_2, sg_3, pl_1, pl_2, pl_3, no

tiemp_mod = pres_ind, impf_ind, pret_ind, pres_subj, imp_subj, futuro, cond,
            pret_perf, pret_plus, pret_ante, pret_perf_subj, pret_plus_subj, futuro_perf, cond_perf,
            imper, infin, ger, part

formav = fin, infin

gen =  mas, fem, mas/fem, no

num =  sg, pl, sg/pl, no


/*  RASGOS CONTEXTUALES:   */

conjugacion = cjg1 cjg2 cjg3

tipo_raiz = 00 11 12 13 14 15 16 21 22 23 24 25 26 31 32 33 34 35 36
               41 42 43 44 45 46 51 52 53 54 55 56 61 62 63 64 65 66
               71 72 73 74 75 76 82 85 90 91 100

tipo_des =  reg pres pret1 pret2 pret3 fut_cond imp_subj imp_subj2 imper
            infin ger part1 part2 defect

tipo_gen =  mas1 mas2 fem inh

tipo_plu = plu1 plu2 noplu
```

Figura 8. Declaración de atributos y valores.

Con esta codificación obtendríamos *este, esto, esta* pero también **estes**, *estos, estas.* Evidentemente, hay una forma incorrecta que la gramática no puede impedir, pues la entrada léxica le dice que con la raíz *est-* puede aparecer el morfema de género masculino *-e*, independientemente de si lleva o no un morfema de plural.

Para resolver este problema de excesivo poder formal, la solución más sencilla es desdoblar la entrada en dos:[100]

[100] Sobre el ejemplo concreto, nosotros preferimos codificar la forma *este* como si fuera una

```
nom(cadena = est, nivel = nivel_-1, ... ,tipo_gen = mas2, tipo_plu = noplu)

nom(cadena = est, nivel = nivel_-1, ... , tipo_gen = mas1 fem, tipo_plu = plu1)
```

Esta codificación tiene el inconveniente de ser menos económica y elegante que la primera pero es la única posibilidad de evitar formas incorrectas. En principio, para el reconocimiento de cadenas no supondría mayor problema el tener una entrada única, pues se parte del supuesto de que las palabras que se analizan son correctas y existentes. En ese caso, nadie preguntaría el análisis de la forma *estes*. Puede ser eficiente incluso en algunas aplicaciones donde se valore la flexibilidad y tolerancia a los errores del usuario (por ejemplo, aceptar la forma irregular pero correcta *anduvo* junto con la forma regular incorrecta, *andó*). Pero a la hora de generar sólo se deben admitir las formas correctas (y en ese caso *estes* sería generada como una de las formas masculinas del plural). La restrictividad es un requisito teórico fundamental. Como señala Shieber(1988) "todas estas teorías [LFG, GSPG y GB] definen la gramaticalidad de las oraciones en términos de restricciones para la buena formación simultánea, y no de procesos para el reconocimiento." Aunque también debe ser un requisito de los sistemas de procesamiento del lenguaje natural con pretensiones "lingüísticas."

En realidad, las **restricciones de aparición de valores incompatibles en disyunción,**[101] dentro de nuestro formalismo, pueden formularse de manera muy sencilla: son incompatibles aquellos valores cuyas cadenas asociadas estén en distribución complementaria y alguna de las formas generadas no sea gramatical; y son compatibles cuando las variantes generadas sean correctas. Siguiendo este criterio son incompatibles:

- **mas1** y **mas2,**
- **noplu** y cualquiera de los otros dos valores, **plu1** o **plu2,**
- **reg** y cualquier otro valor para **tipo_des,**
- **pret1** y **pret2** son incompatibles entre sí,
- **part1** y **part2** también son incompatibles.

Aunque la lista de restricciones de aparición es corta y fácil de recordar, no sería difícil incluir algún tipo de comprobación automática en el programa para evitar que estos valores se codificaran juntos por equivocación en las entradas de diccionario.

entrada del tipo *camión* (una forma aislada en singular). Véase el comentario en la sección 6.2.3.1, Paradigma nominal 1.

[101] No hay que confundir este concepto con las *restricciones de coaparición de rasgos* en la teoría GPSG, pues si bien es cierto que estas últimas establecen condiciones de aceptabilidad entre las estructuras de rasgos, su cometido fundamental es abreviar la información léxica asociada con las palabras concretas de una lengua. Nuestras restricciones no son una técnica para escribir de forma abreviada las entradas de diccionario (Shieber 1986) sino restricciones al poder expresivo del formalismo.

7.1.2 Una regla de la gramática.

Hemos visto hasta el momento cómo codificar las entradas del diccionario y las características de los elementos que portan la información (los rasgos). Analizaremos a continuación de un modo abstracto el funcionamiento de las reglas gramaticales, que combinan la información de las estructuras terminales (entradas de diccionario) para crear estructuras lingüísticas abstractas o no-terminales. En nuestra gramática, dichas estructuras no terminales se acaban en el nivel de la palabra, pero eso no implica que no puedan llegar hasta el nivel oracional con la conveniente extensión de la gramática con reglas únicamente de unificación.

Es conveniente recordar antes de empezar que un formalismo de unificación se basa fundamentalmente en dos operaciones:

- la **concatenación** como única operación que puede combinar cadenas,
- la **unificación** como única operación que combina información

Consecuentemente nuestra regla modelo tiene que permitir esas dos operaciones. Debido a las particularidades del proceso morfológico (y de nuestra formalización) la operación de concatenación es más compleja que en una regla sintáctica, por lo que necesita un mecanismo especial, que una regla sintáctica no requiere.

Una regla se compone de un *constructo superior* (llamado también *nodo madre*), que es la estructura lingüística resultante de la unificación de uno o más *constructos inferiores* (o *nodos hijos*)[102] . Es, por tanto, típicamente una regla independiente del contexto:

```
palabra(nivel = nivel_0, lex = UL, cat = C, ... , cadena = CAD) -->

        raíz(cadena = CADRAIZ, lex = UL, cat = C, ... , tipo_raíz = A B C...),

        morfema(cadena = CADMORF, ... , tipo_raíz = A).
```

Siguiendo la convención usual en Prolog, los elementos que empiezan por mayúsculas son variables y los que empiezan por minúsculas son constantes. El símbolo --> separa la parte derecha de la regla (los nudos hijos) de la parte izquierda (el nudo madre). Cada nudo está encabezado por el término que identifica al constructo (p.ej. *palabra, raíz, morfema*) En el ejemplo hemos utilizado contructos hipotéticos en la parte derecha de la regla. Es decir, *raíz* y *morfema* no existen como tales, sino *raiz_verb, desinencia, morf_gen....* La intención es mostrar una regla en abstracto: se puede generalizar que una regla de la gramática toma una raíz y un morfema para formar una palabra. Los nodos son estructuras de rasgos encerradas

---

[102] Se puede cuestionar por qué un sólo nodo hijo puede "unificarse" consigo mismo. La idea es que para que una estructura de rasgos se convierta en otra con distinta información es necesario que pase a través de una regla. Así por ejemplo un nombre propio puede convertirse en SN y recibir la información propia de un SN gracias al paso por la regla. Nuestra gramática tiene dos casos donde hay un único nodo hijo, las lexicalizaciones verbales y los nominales con género inherente en singular. Se explica porque los elementos terminales, es decir, las entradas de diccionario necesitan "consolidarse" como elementos no terminales, es decir, palabras.

entre paréntesis, separados unos nodos de otros por comas, al igual que los rasgos dentro de los nodos. Los puntos suspensivos dentro de ellos indican un número indeterminado de rasgos con información gramatical (varía según sea un nodo nominal o verbal). Por el momento no nos interesa entrar en más detalles sobre dicha información. Simplemente destacaremos que cualquier constructo resultante de la gramática será una **palabra** con cuatro rasgos imprescindibles (**nivel_0, unidad_léxica, categoría** y **cadena**) además de la mencionada información gramatical. De igual manera, se puede generalizar que cualquier constructo hijo tiene rasgos de concatenación (en nuestro ejemplo, utilizamos paradigmáticamente el rasgo **tipo_raíz**, pero bien pudiera ser cualquier otro). Dichos rasgos también varían si se trata de un constructo nominal o verbal. El punto al final del paréntesis de cierre del último nodo hijo sirve para indicar el final de la regla[103] .

7.1.3 El mecanismo de concatenación morfológica

Este mecanismo es una extensión notacional de la operación de *concatenación* propia de los formalismos de unificación, que permite dar cuenta de las restricciones del contexto. Esta extensión constituye una peculiaridad del formalismo que utilizamos, precisamente por tratarse de un formalismo orientado a la morfología. Es decir, las reglas de combinación usuales en otros formalismos de unificación (Shieber 1986;1989:22-23) parten de las palabras, como cadenas básicas, para concatenar cadenas mayores (sintagmas y oraciones). Dicha operación combinativa se compone de dos partes:

1. la forma en que se concatenan las cadenas,
2. las características que deben presentar las estructuras de rasgos que se combinan

Como señala Shieber(1986), la primera parte se describe utilizando una regla independiente del contexto que especifique "que la cadena asociada con el primero [constituyente] es la concatenación de las cadenas asociadas con el segundo y el tercero [constituyentes]." La segunda parte impone unas condiciones para que se produzca dicha concatenación. Estas condiciones se expresan "mediante identidades entre las subpartes de las estructuras de rasgos asociadas." Para ver cómo funciona esta operación en una regla sintáctica mostraremos cómo se podría escribir una regla en nuestro formalismo para construir un sintagma nominal a partir de un artículo y un nombre, teniendo como condiciones de combinación la concordancia de género y número.

```
sn(nivel = nivel_1, ... , num = N, gen = G, cadena = ART_NOM) -->

  art(nivel = nivel_0, ..., num = N, gen = G, cadena = ART),

  nombre(nivel = nivel_0, ..., num = N, gen = G, cadena = NOM).
```

La primera parte de la operación está expresada mediante:

---

[103] Esto es propio de los formalismos basados en Prolog, donde toda cláusula debe acabar con un punto.

```
sn(cadena = ART_NOM) --> art(cadena = ART), nombre(cadena = NOM).
```

Las condiciones de aplicación de esta regla está especificadas por los rasgos que aparecen en los elementos de la derecha de la regla. Es decir, para que un *artículo* y un *nombre* se combinen formando un *sintagma nominal* es necesario que ambos sean de **nivel_0** y su valor de **num** y **gen** sea coincidente, lo que se expresa con las variables N y G. Esto es lo que se entiende por identidades entre subpartes de las estructuras de rasgos. Por supuesto, esta regla es muy simple y se puede especificar más añadiendo nuevas comprobaciones de rasgos.

Una característica habitual de la operación de combinación de cadenas en sintaxis es que los rasgos que han servido para seleccionar las cadenas se transmiten al constructo resultante (en nuestro caso, son los rasgos de **gen** y **num** del SN). Por el contrario, como característica intrínseca, la información que se utiliza para concatenar morfemas en nuestro formalismo no es información gramatical (sintáctica) y, por tanto, no se transmite al nodo superior. Es decir, **tipo_raiz, conjugación**, etc. no aportan información susceptible de ser usada en el nivel sintáctico, pues (que el autor sepa) ningún verbo está subcategorizado sintácticamente por pertenecer a una o a otra conjugación, por ejemplo.

En principio, que la información no se utilice en el nivel superior no es un argumento para desarrollar una extensión de la operación de concatenación. Si los rasgos utilizados para codificar la aparición de cada forma fueran rasgos con valor único, no habría ningún problema en usar la combinación de cadenas a la manera que hemos mostrado más arriba. Sin embargo, el sistema que nosotros hemos ideado se basa en la disyunción de valores y se hace necesario, por tanto, ampliar el formalismo. No hay que confundir valores en disyunción con la asignación de más de un valor a un atributo. Esto último no está admitido en ningún sistema basado en rasgos por ser contradictorio. La disyunción quiere decir que uno y sólo uno de los valores puede ser usado en la regla que se esté considerando. Es un mecanismo que está admitido en otros formalismos como FUG (Kay 1985) y Bear (1986).

Los rasgos de concatenación pueden tener varios valores en disyunción. Para expresar este concepto utilizamos el de *lista* en Prolog. Una *lista* es simplemente una secuencia ordenada de elementos que puede tener una longitud variable[104] . Los elementos de una lista en Prolog pueden ser desde átomos hasta otros términos con listas incluidas, pero en nuestro caso sólo nos interesan los elementos más simples, i.e. los átomos. Los valores en disyunción de un rasgo de concatenación van a ser expresados como listas de átomos (cada valor un átomo). Los elementos de una lista se escriben separados por comas y encerrados entre corchetes. La versión de Prolog que utilizamos exige que los átomos alfanuméricos estén rodeados por comillas dobles, aunque no así los átomos numéricos. Por lo tanto, los rasgos de concatenación que

[104] Hay una propiedad muy interesante de las listas de elementos: al ser de longitud variable no hay que "predecir" por adelantado de cuántos elementos va a constar la lista. Por ejemplo, no es necesario especificar cuántas formas conjugadas de un verbo utilizan determinada raíz (lo cual, como hemos visto es muy útil, pues hay infinidad de combinaciones para el rasgo **tipo_raíz**. Se puede decir que las listas pueden representar cualquier clase de estructura simbólica y son muy utilizadas en procesamiento del lenguaje natural e inteligencia artificial.

hemos descrito en las secciones de "Flexión Nominal" y "Flexión verbal" se codifican de la siguiente manera en nuestra gramática[105] :

```
tipo_raiz([0,11,12,13,14,15,16,51,52,53,56,82,90]).

tipo_des(["pret1","fut_cond","part1"]).

conjugacion(["cjg2","cjg3"]).

tipo_gen(["mas1","fem"]).

tipo_plu(["plu1","plu2"]).
```

el funtor es el atributo del rasgo y los valores son los átomos de la lista. De esta manera sencilla y eficiente para el ordenador, damos cuenta de los rasgos con valores en disyunción. Hace falta también establecer la relación que asocie los valores en disyunción de un rasgo en un constructo con el valor único de ese mismo rasgo en el otro constructo, de tal forma que se compruebe que ambos constructos poseen el mismo valor para el mismo rasgo. En otras palabras, hay que establecer la identidad entre las subpartes, como hemos visto en el ejemplo de la regla para SN. Lo mejor es verlo con un caso concreto.

El sistema de codificación para identificar las variantes alomórficas que se encadenan presenta la particularidad de que para un mismo rasgo una de las dos cadenas *siempre* tiene valores en disyunción y la otra un valor único:

```
raiz_verb(... conjugacion("cjg2"),tipo_raiz([11,51,52,53,54,55,56])...)

desinencia(... conjugacion(["cjg2","cjg3"]),tipo_raiz(53)...)
```

el ejemplo nos muestra que no necesariamente el rasgo complejo se manifiesta siempre en el mismo constructo. De hecho, podemos comprobar que la **raiz verbal** tiene un rasgo simple para la conjugación y un rasgo complejo para identificar las formas (**tipo_raiz**); en cambio, la **desinencia** puede tener valores en disyunción para la conjugación[106] y un valor único para identificar a qué forma de la conjugación pertenece esa desinencia.

El mecanismo de reconocimiento o de identidad del mismo valor en los dos constructos que se concatenan se basa en la noción de **pertenencia**: se busca si el valor único está incluido en la lista de valores en disyunción. Si hay éxito es que ambas cadenas comparten el mismo valor para el mismo rasgo y, por tanto, satisfacen esa condición. El paso siguiente es comprobar si hay más condiciones; en caso de que una

---

[105] Anticipamos que la gramática está escrita directamente en Turbo-Prolog y que no utilizamos acentos para escribir las constantes.

[106] Recordemos que esto es una forma de simplificación notacional para evitarnos escribir dos entradas de diccionario exactamente iguales menos para el valor de **conjugación**

no se cumpla, se detiene el proceso. La forma de comprobar en Prolog la *pertenencia a una lista* es mediante un predicado **member** definido recursivamente de la siguiente forma:

```
member(X,[X|_]).
```

X es miembro de la lista si X es igual que la cabeza de la lista

o bien,

```
member(X,[_|Y]) :- member(X,Y).
```

X es miembro de la lista si X es un miembro de la cola de la lista

De esta forma, si queremos comprobar la pertenencia de el valor de la conjugación en una *raíz verbal* a la lista de valores de la conjugación de una *desinencia* podemos hacer la siguiente asignación: al valor único de la raíz se le asigna la variable *C2*, por ejemplo, y a la lista de valores las variables *C1* (para la *cabeza* de la lista) y *C3]* (para la *cola* de la lista). El siguiente paso es comprobar si *C2* es miembro de la *cabeza* o de la *cola*, mediante un predicado *member*:

```
member(C2,[C1|C3])
```

En el ejemplo mostrado arriba, la instanciación de valores para el rasgo **conjugacion** sería:

```
member("cjg2",["cjg2"|"cjg3"]).
```

que devolvería el mensaje de que *"cjg2" está incluido en la lista de valores en disyunción.* En realidad, lo que devuelve es que "cjg2" es idéntico al átomo que aparece en la cabeza de la lista. A efectos de nuestro formalismo, que un elemento aislado esté incluido en una lista de elementos es equivalente a decir que el valor único de un rasgo es *idéntico* o coincide con uno de los valores en disyunción del mismo rasgo en otro constructo. Siguiendo con los ejemplos de arriba, la asignación de valores para el rasgo **tipo_raiz** sería:

```
member(53,[11|51,52,53,54,55,56]).
```

lo que produciría el mismo resultado que con la comprobación de pertenencia anterior, sólo que esta vez el elemento aislado (el valor único **53**) está incluido en la *cola* de la lista. De esta manera, comprobaríamos que ambos constructos comparten idénticos valores para los rasgos de concatenación y, por lo tanto, cumplen las condiciones para que se concatenen las cadenas.

Una vez vistas las dos partes del mecanismo de concatenación de morfemas por separado, mostraremos la forma completa de dicho mecanismo utilizando la abstracción del modelo de regla gramatical expuesto en la sección anterior, en su codificación en Prolog. Todas las reglas de nuestra gramática tienen una extensión que asigna variables a los rasgos de concatenación de los constructos con los que opera el mecanismo de concatenación. El predicado Prolog que realiza toda la operación es **concatena** (repetimos aquí la regla abstracta con pares atributo-valor para ayudar a la comparación con su equivalente en Prolog, en la que los rasgos corresponden a las posiciones de los argumentos y en cada posición aparece únicamente el valor):

```
/* Regla modelo con rasgos   */

palabra(nivel = nivel_0, lex = UL, cat = C, ... , cadena = CAD) -->

       raíz(cadena = CADRAIZ, lex = UL, cat = C, ... , tipo_raíz = A B C...),

       morfema(cadena = CADMORF, ... , tipo_raíz = A).



/* Regla modelo con términos */

palabra(nivel_0,UL,Cat, ... ,FormaFlex):-

     raiz(CadRaiz,UL,Cat, ... ,Tipo_raiz1, ...),

     morfema(CadMorf, ... ,Tipo_raiz2, ...),

     concatena(...
               Tipo_raiz1,Tipo_raiz2,
               ...,
               CadRaiz,CadMorf,FormaFlex).
```

Una vez asignadas las variables a los rasgos especiales de concatenación (en el ejemplo de la regla modelo, **Tipo_raiz1, Tipo_raiz2, CadRaiz, CadMorf** y **FormaFlex**) el predicado **concatena** realiza la comprobación con los valores reales de los rasgos y las cadenas. Dicho predicado se define de manera abstracta:

```
concatena(... ,tipo_raiz([R1|R3]),tipo_raiz(R2), ... ,
               CadRaiz,CadMorf,FormaFlex):-

              concat(CadRaiz,CadMorf,FormaFlex),
                   ... ,
              member(R2,[R1|R3]),
                   ...  .
```

Nuestra intención al mostrar la "definición abstracta" del predicado (y de la regla, en los ejemplos anteriores) es insistir en los puntos esenciales, que se dan en todos los casos pero con variantes específicas. En el predicado **concatena** se distinguen nítidamente las dos partes del mecanismo de concatenación. Ya hemos visto cómo funciona la comprobación de condiciones de concatenación (gracias al predicado `member`). Analicemos ahora el predicado `concat`. Dicho predicado toma el primer argumento (*CadRaiz*) y el segundo (*CadMorf*) y los une para formar el tercer argumento (*FormaFlex*) en el mismo orden en el que los encuentra y sin espacios en blanco entre ellos. El resultado de **concat** es la *forma flexionada.*

Se podría pensar que con esta operación simple se podría dar cuenta sin más de la concatenación de morfemas. Hay que tener en cuenta, sin embargo, que los elementos terminales de las reglas son las entradas de diccionario. Es decir, las reglas prueban *todas* las entradas de diccionario que tienen disponibles en ese momento en memoria. Se puede dar el caso, perfectamente posible y muy corriente, de que existan dos o más cadenas idénticas pero pertenecientes a entradas de diccionario diferentes. Pensemos simplemente en la cadena *a*, que puede ser el morfema de femenino, tercera del singular del presente de indicativo, primera y tercera del singular del presente de subjuntivo, etc. Si estamos intentando comprobar, por ejemplo, si la cadena *niña* está compuesta de la subcadena *niñ* y la subcadena *a* todas las entradas para la cadena *a* que haya en el diccionario vigente en ese momento darán como respuesta que la concatenación es correcta. Sabemos, sin embargo, que sólo la combinación entre raíz nominal y el morfema femenino es la correcta. Para discriminar las otras opciones es por lo que usamos la comprobación de rasgos, y únicamente cuando se han pasado todas las comprobaciones pertinentes la concatenación se hace efectiva. Una vez cumplida con éxito la primera parte de la regla, se pasa a la unificación de la información contenida en las estructuras de rasgos.

Por último, hay que señalar que el mecanismo de concatenación cuenta con otras extensiones para comprobar las restricciones del contexto, además de la disyunción de valores. Debido a que estas extensiones son particulares de reglas y casos concretos, las explicaremos en su momento al presentar la gramática.

7.1.4 El mecanismo de unificación de rasgos gramaticales

La idea fundamental de la unificación (como se ha comentado ya en alguna ocasión) es que dos estructuras de rasgos diferentes pero compatibles pueden combinar su información y obtener una estructura de rasgos que incluya la información de ambas. Por el contrario, dos estructuras de rasgos con información contradictoria, es decir, que tengan valores diferentes para el mismo rasgo, no pueden ser unificadas. Este proceso al nivel de la *palabra* es verdaderamente sencillo, pues no implica *rasgos complejos*, muy habituales cuando se pasa al nivel sintagmático y oracional. La forma de transmisión de la información entre los nudos hijos y el nudo madre (o viceversa cuando se trata de generar una palabra) es mediante *variables*. Tomemos de nuevo la regla modelo y veamos cómo se realiza la operación (en esta ocasión omitiremos los rasgos que no llevan información gramatical, es decir, los de concatenación y las cadenas):

```
palabra(nivel = nivel_0, lex = UL, cat = C, xxx = X, yyy = Y, ...) -->
        raíz(lex = UL, cat = C, xxx = X, ...),
        morfema(..., yyy = Y, ...).


palabra(nivel_0,UL,Cat,X,Y, ... ):-
      raiz(... ,UL,Cat,X,Y, ...),
      morfema(... ,Y, ...),
      concatena(...).
```

La regla nos indica que el valor de un rasgo en el nodo madre es el mismo que el del nodo hijo que contiene la variable idéntica, y viceversa. Es decir, que los valores de **lex** (unidad léxica), **cat**(categoría) y **xxx** (cualquier rasgo que pueda contener una raíz) se transmiten desde la *raíz* a la *palabra*, y el rasgo **yyy** (cualquier rasgo con información flexiva) se transmite desde el *morfema flexivo* a la *palabra*. Es importante tener en cuenta la reversibilidad de esta asignación de información, pues en la generación de palabras lo que se proporciona es la información general de la forma flexionada y la regla copia "hacia abajo," hacia los nudos hijos, seleccionando de esa forma los morfemas adecuados.

Por otra parte, también es posible que el nudo madre posea información adicional que no poseen sus nudos hijos. En nuestro ejemplo, la información de **nivel** en la *palabra* es independiente de los valores que tenga en los nudos inferiores. En niveles superiores a la palabra es muy común que la combinación de varios rasgos en los nodos hijos "induce" a asignar un rasgo que no existe en ellos (por ejemplo, la asignación de propiedades funcionales como *sujeto, objeto*, etc. se deduce de determinadas propiedades sintácticas en los nodos inferiores). Esto no se puede considerar propiamente *unificación* en el sentido algebraico de que la estructura de rasgos R es la unión de las estructuras de rasgos R' y R'' ($R = R' \cup R''$). En nuestro formalismo, por tanto, las reglas pueden añadir información adicional a los nodos resultantes. Esto se conoce por el nombre de *adquisición de un rasgo*.

Por último señalar que la unificación es muy fácil de codificar en Prolog: de hecho, es tan simple como demuestra la regla modelo. El mecanismo de *matching* de Prolog se encarga de instanciar los valores de las variables en todos los elementos de la regla. Esta conversión directa entre un formalismo de unificación basado en rasgos y Prolog ha hecho que gran número de proyectos en procesamiento del lenguage natural escojan este lenguaje de programación para su codificación en programas (y éste es nuestro caso).

7.2 La gramática

Nuestra *Gramática de la Palabra* está codificada directamente en Prolog por varias razones. La primera es la eficiencia computacional. Para utilizar un formalismo de rasgos como el descrito en la regla modelo hubiéramos necesitado un compilador de ese formalismo a Prolog, lo que nos hubiera supuesto la inversión de tiempo en la construcción de tal compilador para obtener resultado similares a los actuales (es decir el programa en versión Prolog). Se podría haber implementado, por ejemplo, un parser que aceptara estructuras de rasgos del tipo de los DAGs (*Directed Acyclic Graphs*). El uso de formalismos especiales para escribir gramáticas (los comúnmente denominados *lenguajes de usuario*) son muy útiles para los "escritores de gramáticas" ya que les permiten convenciones notacionales y sobre todo son más fáciles de leer y documentar que las gramáticas directamente codificadas en Prolog. Pero en nuestro caso el formalismo es muy sencillo y la gramática se compone sólo de siete reglas, por lo que las ventajas de los lenguajes de usuario no compensan la construcción de un compilador[107] .

En segundo lugar, tampoco hemos utilizado una *gramática de cláusula definida* (DCG) porque no se dispone hasta el momento (que el autor tenga conocimiento) de una versión en MS/DOS de una DCG compilable, y las versiones interpretadas no proporcionan ni la rapidez ni las herramientas de "visualización" (ventanas, menús, etc.) que da Turbo-Prolog. A la hora de escoger la versión del lenguaje hemos tenido en cuenta primordialmente que el programa de demostración que se incluye como prueba documental de la tesis debía de ser fácil y agradable para el usuario y que no tuviera tampoco requisitos especiales de memoria. En suma, accesible a cualquier usuario y ordenador personal. Téngase en cuenta que esto influye en el tiempo de respuesta: si se hubiera programado para ser usado en ordenadores más potentes (*workstations* o *mainframes*) el tiempo de procesamiento se vería reducido muy apreciablemente.

Por último, hay una motivación teórica para codificar directamente en Prolog. Queremos demostrar que nuestra gramática es una formalización declarativa: si la implementación lo es, necesariamente la formalización tiene que serlo.

El hecho de que la gramática esté escrita en Prolog implica que se efectúa una variedad de unificación, la *unificación de estructuras de términos* (frente a la *unificación de estructuras de rasgos* que se da en otros formalismos). Un *término* es el equivalente a un *rasgo*. Hay términos *básicos*, que corresponden a los rasgos atómicos; y términos *complejos*, que equivalen a los rasgos complejos. Un nodo o constructo de la gramática es un término complejo, compuesto de términos básicos en su mayoría (los que llevan la información gramatical) pero también de términos complejos (los que proporcionan la información contextual). La notación (como hemos adelantado en el modelo de regla) es muy parecida a la usada en lógica de predicados: el símbolo o nombre del predicado (en nuestro caso, el nombre del constructo) es seguido por los términos o argumentos encerrados entre paréntesis. Cada posición de argumento equivale a un rasgo y la constante o la variable que aparece en dicha posición es el equivalente al valor del rasgo. Esto supone dos diferencias notables con respecto a las estructuras de rasgos:

---

[107] No queremos decir con ello que no sea posible, ni deseable; todo lo contrario si es que se pretende ampliar a la sintaxis o incorporar a una gramática sintáctica existente.

1. los valores se identifican por el **orden** en que aparecen en la estructura de términos, en lugar de identificarse por su asociación con un atributo,
2. la **aridad** o **valencia** del constructo es relevante, es decir, el número de elementos de la estructura de términos es significativo para la unificación.

Esto significa en la práctica que todo constructo tiene definido un orden y número fijo de argumentos que debe ser mantenido si queremos que los nodos se unifiquen. Es decir, el cambio de orden o la omisión de un argumento implica directamente que va a fallar la regla. A continuación damos la **definición de los constructos o términos** de la gramática, especificando el nombre de cada argumento y su posición:

```
/* Morfología verbal */

verbo(Nivel,UnidLex,PersNum,TiempMod,FormaV,FormaFlex).

verbo_lex(FormaFlex,Nivel,UnidLex,PersNum,TiempMod,FormaV).

raiz_verb(Cadena,Nivel,UnidLex,Conjugacion,Tipo_raiz,Tipo_des).

desinencia(Cadena,PersNum,TiempMod,FormaV,Conjugacion,Tipo_raiz,Tipo_des).


/* Morfología nominal */

nominal(Nivel,Cat,UnidLex,Gen,Num,FormaFlex).

nom(Cadena,Nivel,Cat,UnidLéx,Gen,Num,Tipo_gen,Tipo_plu).

morf_gen(Cadena,Gen,Tipo_gen).

morf_plu(Cadena,Num,Tipo_num).
```

Se han señalado varias desventajas de la unificación de términos sobre la de rasgos. Es evidente que la codificación es más engorrosa y exige al gramático más atención. El no tener el nombre expreso del rasgo para asignar los valores dificulta la lectura y escritura de reglas. Además, la aridad exige, si se quiere especificar el valor de un argumento, que todos los argumentos del constructo se mencionen, por lo menos con variables. En una estructura de rasgos, los rasgos que no son pertinentes no hay que especificarlos necesariamente. A pesar de estas incomodidades, el tamaño reducido de la Gramática de la Palabra permite minimizarlas considerablemente, ganando en eficiencia computacional.

7.2.1 Reglas de la Morfología Verbal

Nuestra gramática tiene tres reglas para dar cuenta de todas las formas verbales del español: una para todas las formas que constan de raíz y desinencia, otra para las formas lexicalizadas, y la última para la formación de tiempos compuestos.

La regla básica de la morfología verbal incluye todos los tiempos y personas, tanto de verbos regulares como irregulares, así como las formas inexistentes de los verbos defectivos. La noción básica es que por esta regla se construyen todas las formas que se compongan de una **raiz_verb** y una **desinencia**. Ya hemos hablado extensamente de las características y rasgos que llevan asignados, veamos ahora cómo se integran en la regla:

```
verbo(nivel_0,UL,PersNum,TiempMod,FormaV,FormaFlex):-

    raiz_verb(CadenaR,nivel_1,UL,Conjugacion1,Tipo_raiz1,Tipo_des1),

    desinencia(CadenaD,PersNum,TiempMod,FormaV,Conjugacion2,Tipo_raiz2,Tipo_des2),

    concatena_v(Conjugacion1,Conjugacion2,
            Tipo_raiz1,Tipo_raiz2,
            Tipo_des1,Tipo_des2,
            CadenaR,CadenaD,FormaFlex).
```

La regla se interpreta de la siguiente manera: una **raiz_verb** y una **desinencia** constituyen un **verbo** (o viceversa, un verbo está compuesto por una raíz verbal y una desinencia) si la información de ambas es compatible y si se cumplen los requisitos de **concatena_v**[108] . Este último predicado tiene instanciados los valores respectivos de los nodos hijos para los argumentos **conjugacion, tipo_raiz, tipo_des** y sus *cadenas*. La clave de todo el proceso está en el argumento **FormaFlex**, es decir, la unión de las dos subcadenas. Es el único argumento del predicado **concatena_v** que también está presente en el predicado **verbo** y gracias a ello la regla es independiente del orden (ascendente o descendente). Es decir, en el análisis **FormaFlex** será el primer argumento instanciado de la regla (corresponde a la palabra que el usuario escribe cuando quiere conocer su estructura interna descompuesta en rasgos); por el contrario, **FormaFlex** será el último argumento instanciado en el proceso de generación (corresponde a la forma que devuelve el programa cuando el usuario le proporciona unos datos sobre su estructura morfológica). La concatenación es tan importante como la unificación: si bien es cierto que lo que buscamos es el procesamiento de la información, todo ese proceso no terminará con éxito sin la concatenación adecuada. En términos del programa, todo procesamiento pasa por el predicado **concat** de **concatena**. Es una pieza fundamental de las reglas tanto durante el proceso de análisis como en el de generación. En el análisis comprobará que la **FormaFlex** se descompone en las dos subcadenas que se están probando. En la síntesis, una vez que se ha comprobado que la información asociada a una raíz y una desinencia es compatible, **concat** unirá las dos cadenas para formar **FormaFlex**.

---

[108] Se hace evidente una vez más que el predicado **concatena** es una extensión formal para dar cuenta del contexto, pero dentro de una gramática independiente del contexto.

Una característica sobresaliente de los formalismos de unificación es que "no hay nada intrínsecamente direccional" (Shieber 1986;1989:25), es decir, que se pueden usar igualmente para el reconocimiento que para la síntesis. Esto es lo que hace precisamente nuestra gramática: con la misma regla analiza y genera una palabra dada. En el primer caso se va desde la cadena a la información, en el segundo desde la información a la cadena.

Volviendo a la regla para formas verbales, veremos en primer lugar la parte de la unificación, para pasar seguidamente a las condiciones de concatenación. Si excluimos los argumentos del mecanismo de concatenación, los elementos informativos de la regla son:

```
verbo(nivel_0,UL,PersNum,TiempMod,FormaV):-

      raiz_verb(nivel_1,UL),

      desinencia(PersNum,TiempMod,FormaV).
```

La información gramatical del constructo **verbo** es:

- **nivel 0**,
- la *unidad léxica* es igual a la *unidad léxica* del constructo **raiz_verb**,
- el rasgo de *persona y número* es igual al valor del mismo rasgo en **desinencia**,
- el valor de *tiempo y modo* es igual al de **desinencia**, y
- el valor de la *forma verbal* es el mismo que el de **desinencia**

Todos estos valores se transmiten indistintamente desde el nodo madre o desde los nodos hijos mediante el ligado de variables, en otras palabras, mediante la unificación de términos.

Hay tres cláusulas distintas para el predicado **concatena_v**, que se utilizan respectivamente para los verbos regulares, irregulares y defectivos. La primera de ellas es:

```
/*--- Verbos regulares  ---*/

concatena_v(conjugacion(C2),conjugacion([C1|C3]),tipo_raiz([100]),tipo_raiz(_),
        tipo_des([D1|D3]),tipo_des(D2),CadenaR,CadenaD,FormaFlex):-
          concat(CadenaR,CadenaD,FormaFlex),
          member(C2,[C1|C3]),
          member(D2,[D1|D3]).
```

Por esta cláusula sólo pueden pasar los constructos **raiz_verb** que tengan **tipo_raiz([100])**. En la parte descriptiva de la codificación de la entradas verbales explicamos que escogimos la convención de asignar a todas las raíces de los verbos regulares el valor **100**. Con esto nos evitamos escribir todos los códigos numéricos de la conjugación. Es, por tanto, una convención notacional lo que explica la existencia de

esta cláusula. Por lo demás, la comprobación de las condiciones contextuales es como se ha explicado en la sección 7.1.3, "El mecanismo de concatenación morfológica" en la página 203. En concreto:

- la cadena de la raíz y la de la desinencia unidas forman la cadena de la forma verbal,
- la conjugación de la raíz coincide con alguno de los valores en disyunción de la conjugación de la desinencia, y
- el valor de **tipo_des** de la desinencia está incluido en la lista de valores para **tipo_des** de la raíz.

```
/*--- Verbos irregulares --*/

concatena_v(conjugacion(C2),conjugacion([C1|C3]),tipo_raiz([R1|R3]),tipo_raiz(R2),
        tipo_des([D1|D3]),tipo_des(D2),CadenaR,CadenaD,FormaFlex):-
          concat(CadenaR,CadenaD,FormaFlex),
          member(C2,[C1|C3]),
          member(R2,[R1|R3]),
          member(D2,[D1|D3]).
```

La cláusula **concatena_v** para los verbos irregulares se diferencia de la anterior en que comprueba que el valor de **tipo_raiz** de la desinencia (representado por la variable **R2**) pertenece a la lista de valores de **tipo_raiz** de la raíz verbal. El resto de las comprobaciones son igual.

```
/*--- Verbos defectivos ---*/

concatena_v(conjugacion(C2),conjugacion([C1|C3]),tipo_raiz([R1|R3]),tipo_raiz(R2),
        tipo_des([D1|D3]),tipo_des(_),CadenaR,CadenaD,"(defectivo)"):-
          member("defect",[D1|D3]),
          not(member(R2,[R1|R3])).
```

La cláusula para verbos defectivos es una extensión notacional, al igual que la de los verbos regulares. Se utiliza para "generar" las formas inexistentes de los verbos irregulares. En realidad, es simplemente un recurso para indicar al usuario que dicha forma no existe: cuando el usuario pide el paradigma verbal de un verbo defectivo o una forma inexistente el programa le devuelve (defectivo). En la primera versión no se incluía este mensaje, sino que aparecía en blanco. Esto inducía a pensar que la gramática no era capaz de tratar estos casos y por eso hemos preferido mostrarlos explícitamente.

En la descripción de la codificación de estos casos se dijo que el rasgo **tipo_des** de la raíz llevaba asignado, además de cualquier otro, el valor **defect**. La idea es que sólo se especifiquen en la entrada de diccionario los códigos de las formas que existen, entendiéndose que el resto son formas defectivas. El funcionamiento del mecanismo de concatenación comprueba en primer lugar si el valor **defect** está incluido entre los valores en disyunción del rasgo **tipo_des** de la raíz. Una vez descubierto que se trata de un verbo defectivo, se procede a comprobar si esa forma existe o no, mediante una

operación que se basa en la *no pertenencia* en lugar de la *pertenencia* a la lista. Esto se efectúa por medio del predicado **not(member(R2,[R1|R3]))**, que comprueba que un código numérico determinado no pertenece a la lista de códigos de **tipo_raiz**. O lo que es lo mismo, comprueba que no es una de las formas existentes para ese verbo.

La regla para formas lexicalizadas es realmente sencilla y es la única que no necesita pasar por el mecanismo de concatenación (no se une a ninguna cadena, ni tiene condiciones especiales de aparición como en el caso de los nominales de género inherente):

```
verbo(nivel_0,UL,PersNum,TiempMod,FormaV,FormaFlex):-

    verbo_lex(FormaFlex,nivel_0,UL,PersNum,TiempMod,FormaV).
```

La información que pasa de un constructo al otro es la misma. La única diferencia, aparte del nombre del constructo en sí, es el cambio de orden con respecto al argumento **FormaFlex**. Siguiendo una convención, las cadenas son el primer argumento en las entradas de diccionario, y el último argumento en los constructos resultantes (las palabras). Distinguimos así los elementos terminales (entradas de diccionario) de los elementos no terminales de la gramática. Se podría pensar que las formas lexicalizadas pueden tratarse directamente como **verbo**, y evitarnos así la regla. Hay dos motivos para desestimar esta posibilidad. En primer lugar, uno práctico: por las restricciones de la versión de Prolog que utilizamos, no se pueden separar cláusulas idénticas. En el caso supuesto de igualar las formas lexicalizadas al constructo **verbo** tendríamos que poner dentro de la gramática todas las entradas lexicalizadas, rompiendo así la modularidad entre gramática y diccionario. La razón teórica está relacionada con la anterior: las teorías lingüísticas suelen separar la gramática del lexicón, entendiendo que las piezas léxicas se "insertan" en las reglas gramaticales. En algunos formalismos de sistemas de procesamiento de lenguas naturales se llama a esta operación "la consolidación de estructura": para que cualquier elemento léxico terminal pueda transmitir su información a elementos de nivel superior es necesario que su estructura informativa sea "licenciada" o "consolidada" por una regla de la gramática. De esta manera, la regla convierte una entrada lexicalizada del diccionario como *he* (de *haber*) en un elemento no terminal (**verbo**), de forma que pueda pueda pasar por la regla de tiempos compuestos:

```
/*  Regla para la formación de tiempos compuestos                */

verbo(nivel_0,UL,PersNum,TiempMod,nofin,FormaCompuesta):-

    verbo(nivel_0,haber,PersNum,TiempMod_Aux,fin,Auxiliar),

    verbo(nivel_0,UL,no,part,nofin,Principal),

    concatena_vcomp(TiempMod_Aux,TiempMod,
                    Auxiliar,Principal,FormaCompuesta).
```

Esta regla es la única en la gramática cuyos nodos hijos son elementos no terminales.[109] Es decir, dos **verbos** (con determinadas características) forman otro **verbo**. En algunos procesadores morfológicos (Rodríguez Magro *et al.* 1990) se considera que los tiempos compuestos pertenecen a la sintaxis y, por tanto, no deben tratarse con reglas morfológicas. Por nuestra parte, no distinguimos en la gramática entre reglas morfológicas y sintácticas, ya que consideramos la flexión como un fenómeno gramatical como cualquier otro. Nuestra intención al incluir las formas compuestas es mostrar cómo se integra perfectamente la flexión dentro de la sintaxis sin necesidad de ampliar el formalismo gramatical. De manera análoga, se pueden escribir reglas para fenómenos sintácticos muy cercanos a la morfología, como las perífrasis verbales o los clíticos incorporados al verbo en la misma cadena gráfica. Además, sin cambiar de tipo de gramática se puede hacer la transición desde el morfema hasta la oración, lo que no ocurre en los sistemas que utilizan el *Modelo de dos niveles* para la morfología (es decir, una gramática de estados finitos) y luego una gramática independiente del contexto para la sintaxis[110] . Nuestra aproximación a la flexión está en la línea de las propuestas de integrarla dentro de la Sintaxis y, por tanto, no tener un nivel autónomo para la Morfología, con principios y reglas particulares.

La regla presenta una diferencia con respecto al resto: no hay que especificar el contexto morfológico sino el sintáctico. Esto quiere decir que el mecanismo de concatenación es distinto al de las otras reglas. Como mencionamos al hablar en abstracto del mecanismo de concatenación, el contexto sintáctico se especifica utilizando rasgos con información gramatical que luego pueden transmitirse o no al nodo superior. En nuestro caso, las condiciones para la formación de formas verbales compuestas son:

- en el primer nodo hijo:
  - que la *unidad léxica* sea **haber**,
  - que el valor para *forma verbal* sea **fin** (forma finita o personal)

---

[109] Por definición, en cualquier regla de estructura sintagmática independiente del contexto, el elemento de la derecha es siempre un elemento no terminal, aunque los de la izquierda pueden ser terminales o no terminales.

[110] El hecho de que no se incluyan reglas para las perífrasis verbales o los clíticos, por ejemplo, se debe a condicionamientos de nuestro programa de demostración: el programa no sería eficiente si se añaden nuevas reglas. Por supuesto, en entornos mucho más potentes que el de los ordenadores personales se podrían escribir estas reglas y otras para construir sintagmas y oraciones, sin bloquear el programa. Por otra parte, construir una gramática sintáctica del español, aunque sea muy pequeña, se escapa a los objetivos de esta tesis. Hemos escogido las formas verbales compuestas como botón de muestra de lo que se puede desarrollar y también porque como dice la *Academia*:

> Si nos atenemos a los principios lingüísticos más rigurosos, estas formas llamadas compuestas no constituyen tema propio de la Morfología, sino de la Sintaxis, ni más ni menos que otras perífrasis verbales. Para la inclusión de las formas compuestas en el cuadro de la flexión [...] hemos de tener en cuenta, primero: que el participio aparece siempre en ellas en la forma invariable *-do*, privado así de las variaciones de género y número con que funciona en otras perífrasis [...]. En segundo lugar, ... las formas de *haber* que acompañan al participio... han perdido su contenido semántico originario para convertirse en "mero signo formal".. Diremos por último que el relativo margen de equivalencia con que funcionan a veces algunos pares de formas simples y compuestas: *amé* = *he amado*, *amara* = *había amado*, nos autoriza, con algún fundamento, a no separar las segundas del cuadro morfológico de la flexión. (R.A.E. 1973;1989:252-253).

- en el segundo nodo hijo:
  - que el valor de *tiempo y modo* sea **part**.

El resto de comprobaciones, como el **nivel_0, formav = nofin** y **pers_num = no**, son redundantes. Se podría perfectamente sustituir por variables, pero es más eficiente para Prolog comprobar constantes que variables.

La información gramatical que se pasa al nuevo constructo es:

- la *unidad léxica* es transmitida desde el segundo nodo hijo (el verbo principal)
- la *persona y número* procede del primer nodo hijo (el verbo auxiliar).

El mecanismo de concatenación se utiliza, como es habitual, para concatenar las dos cadenas (verbo auxiliar y verbo principal) que proceden de la concatenación anterior de los elementos terminales. Pero además sirve como extensión notacional para transmitir el rasgo **tiemp_mod** con su valor pertinente, deducido del valor del mismo rasgo en el constructo del participio:

```
/*--- Formas compuestas ---*/

concatena_vcomp(TiempMod_Aux,TiempMod,Auxiliar,Principal,FormaCompuesta):-
          concat(Auxiliar," ",Auxiliar_Bl),
          concat(Auxiliar_Bl,Principal,FormaCompuesta),

          TiempMod_Aux = pres_ind,  TiempMod = pret_perf, '.

concatena_vcomp(TiempMod_Aux,TiempMod,Auxiliar,Principal,FormaCompuesta):-
          concat(Auxiliar," ",Auxiliar_Bl),
          concat(Auxiliar_Bl,Principal,FormaCompuesta),

          TiempMod_Aux = impf_ind,  TiempMod = pret_plus, '.

concatena_vcomp(TiempMod_Aux,TiempMod,Auxiliar,Principal,FormaCompuesta):-
          concat(Auxiliar," ",Auxiliar_Bl),
          concat(Auxiliar_Bl,Principal,FormaCompuesta),

          TiempMod_Aux = pret_ind,  TiempMod = pret_ante, '.
```

```
concatena_vcomp(TiempMod_Aux,TiempMod,Auxiliar,Principal,FormaCompuesta):-
          concat(Auxiliar," ",Auxiliar_Bl),
          concat(Auxiliar_Bl,Principal,FormaCompuesta),

          TiempMod_Aux = pres_subj, TiempMod = pret_perf_subj, '.

concatena_vcomp(TiempMod_Aux,TiempMod,Auxiliar,Principal,FormaCompuesta):-
          concat(Auxiliar," ",Auxiliar_Bl),
          concat(Auxiliar_Bl,Principal,FormaCompuesta),

          TiempMod_Aux = imp_subj,  TiempMod = pret_plus_subj, '.

concatena_vcomp(TiempMod_Aux,TiempMod,Auxiliar,Principal,FormaCompuesta):-
          concat(Auxiliar," ",Auxiliar_Bl),
          concat(Auxiliar_Bl,Principal,FormaCompuesta),

          TiempMod_Aux = futuro,    TiempMod = futur_perf, '.

concatena_vcomp(TiempMod_Aux,TiempMod,Auxiliar,Principal,FormaCompuesta):-
          concat(Auxiliar," ",Auxiliar_Bl),
          concat(Auxiliar_Bl,Principal,FormaCompuesta),

          TiempMod_Aux = cond,      TiempMod = cond_perf, '.
```

La primera cláusula **concatena_vcomp** establece que si el valor de *tiempo y modo* del auxiliar es igual a **pres_ind** entonces el valor de *tiempo y modo* de la forma compuesta es **pret_perf** (Pretérito perfecto de indicativo); y así sucesivamente se asignan los valores de tiempo y modo del nuevo constructo. Otra forma de conseguir el mismo resultado es escribir una regla para cada combinación de auxiliar y verbo principal y asignar el valor como resultado de la aplicación de la regla. Esto supondría escribir siete reglas casi idénticas (con el único cambio de los valores para **tiempmod**) en lugar de la única que tenemos. Por motivos de eficiencia, como hemos comentado anteriormente, es preferible tener la menor cantidad posible de reglas a costa de tener extensiones notacionales incorporadas en el mecanismo de concatenación.

7.2.2 Reglas de la Morfología Nominal

Hay cuatro reglas para dar cuenta de la flexión nominal, tanto para nombres y adjetivos como para el resto de categorías que tengan información de género y número. Debido a esta circunstancia el constructo resultante, (forma) **nominal**, lleva el rasgo **cat** para indicar la categoría del elemento nominal creado. Para las formas verbales este rasgo es redundante porque ya se indica con el nombre mismo del constructo. Las reglas de la flexión nominal son las siguientes:

1. Palabras sin morfema de género y de número,
2. Palabras sin morfema de género pero con morfema de número,
3. Palabras con morfema de género, y
4. Palabras con morfema de género y número

Las reglas (1) y (2) tratan las palabras de género inherente en singular y plural, pero también se utilizan por otros nominales como los *pluralia y singularia tantum* o aquellos que no llevan morfema de género masculino aunque sí de femenino (nombres y adjetivos acabados en *-or, -ol*). Las reglas (3) y (4) se encargan de la flexión de las formas nominales que constan de *raíz* y *morfema/s flexivo/s*.El número de reglas se reduce, por tanto, a la combinación de posibilidades entre tener o no tener morfemas de género y número. En comparación con el verbo, se da un fuerte contraste entre el número de formas flexionadas y el número de reglas: el número máximo de formas de un nominal es de cuatro (nombres y adjetivos con género gramatical) o de cinco (demostrativos, indefinidos, algunos ordinales y adjetivos con forma duplicada para el masculino singular, *primer, primero, primera, primeros, primeras; buen, bueno, buena, buenos, buenas*); en cambio, el número de formas flexionadas del verbo supera las ochenta. Por el contrario, si descontamos la regla para formas lexicalizadas (por tratarse de casos especiales, insignificantes cuantitativamente) y la regla de las formas verbales compuestas (por tratarse de una regla que opera con elementos no terminales), nos encontramos con una regla verbal por cuatro nominales. Una característica relacionada con este hecho es que mientras todas las entradas verbales del diccionario pasan por una única regla, una misma entrada nominal de diccionario puede pasar por dos o más reglas. De hecho, utilizaremos como modelo la entrada de *investigador* para mostrar cómo en una misma codificación se puede incluir todas y sólo las formas flexionadas gramaticales.

La primera observación que debemos mencionar es que para conseguir reducir el número de entradas y de reglas hemos tenido que "relajar" la condición de que las *palabras* (es decir, **nivel_0**) eran los únicos elementos léxicos que podían aparecer en la superficie sin necesidad de unirse a elementos flexivos. Permitimos, entonces, que determinados elementos de **nivel -1** pasen por las reglas propias de las *palabras* con género inherente. Las entradas léxicas de **nivel_-1** con licencia son aquellas a las que atribuimos características de híbridos entre *palabras* y *raíces*, como por ejemplo *investigador, español.* Las presentamos ahora tal y como aparecen en la base de datos-diccionario que utiliza la gramática. Siguen el formato exigido por Turbo-Prolog: cadenas y símbolos entre dobles comillas, y no están permitidas las variables anónimas. Algunos caracteres no están permitidos dentro de los símbolos, como por ejemplo los signos matemáticos + - = > <: ésa es la razón de que escribamos **nivel_1** en lugar de **nivel -1**.

```
nom("investigador","nivel_1","nombre","investigador","mas","sg",
          tipo_gen(["inh","fem"]),tipo_plu(["plu2"]))

nom("español","nivel_1","nombre","español","mas","sg",
          tipo_gen(["inh","fem"]),tipo_plu(["plu2"]))
```

Analizando estas entradas podemos encontrar rasgos propios de las *palabras* como **mas, sg, tipo_gen = inh, tipo_plu = plu2** con otros rasgos inequívocamente de *raíces*: **nivel_-1** y **tipo_gen = fem**. Con todos ellos se pueden analizar y generar sólo las cuatro formas *investigador, investigadora, investigadores, investigadoras.* Cada forma pasa por una regla distinta, empecemos por el masculino singular:

```
/* Palabras sin marca formal de género y número  */

nominal(nivel_0,Cat,UL,Gen,Num,FormaFlex):-

        nom(FormaFlex,Nivel,Cat,UL,Gen,Num,Tipo_gen,Tipo_plu),
        concatena_n1(Tipo_gen,tipo_gen(inh),Num).
```

Lo primero que extraña en esta regla es la presencia del predicado **concatena_n1**. ¿Cuál es su utilidad si no se tiene que concatenar ninguna cadena con otra? Recordemos que para las formas lexicalizadas del verbo no se utilizaba el mecanismo de concatenación. Sin embargo, debemos recordar también que la misión de dicho mecanismo es doble: concatenar cadenas, por un lado, y aplicar extensiones que permitan comprobar las condiciones contextuales, por otro. Bien podríamos concatenar un **nom** con una cadena vacía, pero nos parece una tarea innecesaria computacionalmente, e incorrecta desde el punto de vista de un formalismo basado en la superficie (los morfemas $\emptyset$ no están permitidos). En cambio, **concatena_n1** nos sirve para comprobar dos condiciones:

1. que la entrada **nom** tiene el valor **inh**, y
2. que la entrada tiene algún valor para **num** distinto de **no**

```
concatena_n1(tipo_gen([G1|G3]),tipo_gen(G2),Num):-
          not(Num = "no"),
          member(G2,[G1|G3]).
```

La comprobación de **tipo_gen** es obvia: la función de esta regla es que pasen las palabras de género inherente o no marcado morfológicamente. Dado que el rasgo **tipo_gen** puede tener varios valores en disyunción (por ejemplo, **inh** y **fem**) es necesario efectuar la comprobación. La otra condición está estrechamente relacionada con la anterior: las entradas con rasgo inherente que pasan por esta regla transmiten el valor de **num** al constructo *nominal*, como no puede haber formas flexionadas con el rasgo **num** = **no**, el predicado impide que pasen las entradas con ese valor (a través de la comprobación `not(Num = "no")`. ¿Cuáles son las entradas "prohibidas"? Son las variantes gráficas de plural de las palabras con género inherente, como *camion* o *pec*. Estas entradas llevan precisamente el rasgo **num** = **no** para indicar que no pueden llevar esa información, ya que la toman del morfema de plural, como veremos en la siguiente regla.

La operación de unificación en esta regla se limita a transmitir los valores del nodo hijo, y asigna el **nivel_0** porque ya se ha consolidado la entrada léxica como elemento no terminal. Ya mencionamos al principio de la explicación que por esta regla pasan todas las formas que no necesitan un morfema flexivo para completar su información sintáctica de género y número. Por tanto, pasan por aquí:

1. las palabras de género inherente, en singular (*sol, grande*)
2. palabras invariables (*crisis, isósceles*)
3. *singularia tantum* (*estrés, informática*)
4. *pluralia tantum* (*matemáticas, bienes*)

5. palabras con género gramatical que no tiene morfema de masculino (*investigador, español*)

```
/*--- Palabras de género inherente en plural ---*/

nominal(nivel_0,Cat,UL,Gen,Num,FormaFlex):-

        nom(CadenaR,Nivel,Cat,UL,Gen,_,Tipo_gen,Tipo_plu1),
        morf_plu(CadenaP,Num,Tipo_plu2),

        concatena_n2(Tipo_gen,tipo_gen(inh),
                Tipo_plu1,Tipo_plu2,
                CadenaR,CadenaP,FormaFlex).
```

La siguiente regla sirve únicamente para los grupos (1) y (5) mencionados en la primera regla pues, evidentemente, las palabras invariables, los *singularia* y *pluralia tantum* sólo tienen una forma. Es decir, por aquí pasan *soles, peces, camiones, artistas, investigadores.* La trasmisión de rasgos al nodo madre se realiza como de costumbre:

- la *unidad léxica*, la *categoría* y el *género* se instancian desde el nodo **nom**,
- la información de *número* proviene del **morf_plu**

Las condiciones de concatenación son:

1. que el nodo **nom** tenga el valor **inh** para **tipo_gen**,
2. que el nodo **nom** tenga entre sus valores en disyunción para **tipo_plu** el mismo valor que el del nodo **morf_plu**.

```
concatena_n2(tipo_gen([G1|G3]),tipo_gen(G2),tipo_plu([P1|P3]),tipo_plu(P2),
            CadenaR,CadenaP,FormaFlex):-
            concat(CadenaR,CadenaP,FormaFlex),
            member(G2,[G1|G3]),
            member(P2,[P1|P3]).
```

Por las dos últimas reglas se construyen las formas flexionadas como *gato, gata, gatos, gatas, este, esto, esta, estos, estas...*, que se componen de una *raíz* (**nivel_-1**) y un morfema de género y (a veces) de un morfema de número:

```
/*--- Palabras de género gramatical en singular  ---*/

nominal(nivel_0,Cat,UL,Gen,Num,FormaFlex):-

      nom(CadenaR,nivel_1,Cat,UL,_,Num,Tipo_gen1,_),
      morf_gen(CadenaG,Gen,Tipo_gen2),

      concatena_n3(Tipo_gen1,Tipo_gen2,
              CadenaR,CadenaG,FormaFlex).


/*--- Palabras de género gramatical en plural ---*/

nominal(nivel_0,Cat,UL,Gen,Num,FormaFlex):-

      nom(CadenaR,nivel_1,Cat,UL,_,_,Tipo_gen1,_),
      morf_gen(CadenaG,Gen,Tipo_gen2),
      morf_plu(CadenaP,Num,Tipo_plu),

      concatena_n4(Tipo_gen1,Tipo_gen2,Tipo_plu,
              CadenaR,CadenaG,CadenaP,FormaFlex).
```

La asignación de valores en estas reglas es:

- la *categoría* y la *unidad léxica* de la *raíz nominal*,
- el *género* del *morfema de género*,
- el *número*
  1. o bien del *morfema de plural* (si aparece)
  2. o bien de la *raíz* (**sg**, herencia por defecto)

Nótese que en estas reglas aparecen algunas *variables anónimas*. Esto se debe a que sus valores no son pertinentes y por tanto se ignoran.

Con respecto a las condiciones de concatenación, la comprobación fundamental en ambas reglas es la de **tipo_gen**, que se encarga de asegurar que unan los alomorfos apropiados:

```
concatena_n3(tipo_gen([G1|G3]),tipo_gen(G2),CadenaR,CadenaG,FormaFlex):-
            concat(CadenaR,CadenaG,FormaFlex),
            member(G2,[G1|G3]).


concatena_n4(tipo_gen([G1|G3]),tipo_gen(G2),Tipo_plu,CadenaR,CadenaG,CadenaP,FormaFlex):-
          concat(CadenaR,CadenaG,FormaFlexGen),
          concat(FormaFlexGen,CadenaP,FormaFlex),
          member(G2,[G1|G3])
          Tipo_plu = tipo_plu("plu1").
```

En el predicado **concatena_n4** se comprueba además que el valor de **tipo_plu** del morfema de plural sea **plu1**. Es decir, el único alomorfo de plural que se permite es *-s*, ya que no se puede dar *-es* cuando aparece un morfema de género (**gataes*). Una peculiaridad del mecanismo de concatenación en la regla de formas de plural es que necesita aplicar dos veces el predicado **concat**. Esto se debe a que dicho predicado sólo admite unir dos cadenas al mismo tiempo. Por ello, hemos tenido que hacer la concatenación secuencialmente:

1. se unen la raíz (*CadenaR*) y el morfema de género (*CadenaG*) para formar la *Forma_Flexionada_con_Género*
2. la *FormaFlexGen* se concatena con el morfema de plural (*CadenaP*) y da como resultado *FormaFlex*.

7.2.3 El tratamiento de la derivación y la composición

Nuestra tesis pretende demostrar que con una gramática independiente del contexto, ampliada con un mecanismo de concatenación para dar cuenta de las restricciones contextuales, es posible tratar la Morfología flexiva dentro de la Sintaxis.

Por otra parte, creemos que dicho mecanismo se puede adaptar (o mejor dicho utilizar) a los alomorfos derivativos. En esencia el procedimiento sería el mismo que el utilizado para la morfología flexiva:

1. establecer el inventario de morfemas derivativos y sus alomorfos,
2. establecer las condiciones contextuales de aparición de cada alomorfo,
3. crear rasgos especiales de concatenación para cada morfema, siguiendo el modelo de **tipo_....**, y definir los valores posibles que identifiquen los distintos alomorfos,
4. crear las entradas de diccionario para los alomorfos derivativos, con su información gramatical y contextual, y añadir la información contextual a las entradas léxicas (**raiz_verb, nom**) que lo necesiten,
5. por último, crear reglas para la derivación teniendo en cuenta que habrá que concatenar varias cadenas, con sus respectivas condiciones de concatenación. El sufijo derivativo transmitirá su información específica a la palabra.

Para escoger un ejemplo sencillo, veamos cómo podría implementarse esto para los derivados deverbales en *-miento, -mento*, explicados en la sección 4.4.1. Los primeros tres puntos de la lista se cumplen de manera directa:

```
morfema derivativo:  lex = mento

alomorfos          :  cadena = amento     cadena = imiento   ...

rasgo especial     :  tipo_mento = amento imiento ...
```

Asignar el rasgo **tipo_mento** tampoco supone mayor complicación:

```
cadena = adit      .... tipo_mento = amento

cadena = reconoc   .... tipo_mento = imiento
```

Por último, las reglas necesarias (una para singular y otra para plural) podrían utilizarse también para otros sufijos con las mismas características pero para ello habría que unificar el nombre del rasgo, por ejemplo, **tipo_sufijo**, y asignarle valores específicos a cada cadena. (Todo esto es bastante ambiguo, pero evidentemente no contamos con el inventario de morfemas y alomorfos, que nos permita elaborar un sistema de codificación de la misma forma que hicimos con las variantes de las desinencias y los morfemas flexivos nominales). Un esbozo de reglas para los derivados deverbales sería:

```
nominal(nivel_0,Cat,UL,Gen,Num,XXXX,FormaFlex):-

      raiz_verb(CadenaR,nivel_1,_,UL,_,_,_,_,Tipo_sufijo1),
      sufijo(CadenaS,Gen,Num,XXXX,Tipo_sufijo2),

      concatena_n8(Tipo_sufijo1,Tipo_sufijo2,
                   CadenaR,CadenaS,FormaFlex).



nominal(nivel_0,Cat,UL,Gen,Num,XXXX,FormaFlex):-

      raiz_verb(CadenaR,nivel_1,_,UL,_,_,_,_,Tipo_sufijo1),
      sufijo(CadenaS,Gen,_,XXXX,Tipo_sufijo2),
      morf_plu(CadenaP,Num,Tipo_plu),

      concatena_n9(Tipo_sufijo1,Tipo_sufijo2,Tipo_plu,
                   CadenaR,CadenaS,CadenaP,FormaFlex).
```

Como es habitual, la información no pertinente se ignora (usando la variable anónima). El sufijo aporta la información nominal de género y número (este último sólo cuando no aparece el morfema de plural, es decir, es singular por defecto). Además sube la información específica, que hemos representado con la variable XXXX. La raíz verbal proporciona la **unidad léxica**. Por otra parte, el mecanismo de concatenación se encarga de hacer las comprobaciones de la manera acostumbrada: comprueba la coincidencia de los valores de **tipo_sufijo** y concatena las cadenas. Para los compuestos simples (es decir, los de un palabra como *abrebotellas, rojiblanco*) se podría utilizar un procedimiento similar, y los compuestos complejos se tratarían con reglas de tipo sintagmático. En este sentido, es importante señalar que desde la perspectiva de la morfología generativa actual se considera que "tanto la flexión como la derivación se realizan, en una lengua como la española, por medio de procedimientod de afijación idénticos" (Varela 1990:162).

Por tanto, aparte de un cambio importante en el diccionario (el aumento considerable de nuevas entradas y el cambio de la aridad de las entradas existentes para las raíces nominales y verbales al añadir los nuevos rasgos contextuales), el tratamiento de los sufijos derivativos nos supone la creación de reglas especiales. Pero todo ello, en principio, es manejable utilizando un ordenador potente. Por supuesto, la complejidad del diccionario obligaría a buscar alguna forma de compactación de la información, como existe en todos los sistemas con una base léxica de gran tamaño. Pero, en cualquier caso, creemos que es posible utilizar nuestro formalismo para dar cuenta de la derivación.

Sin embargo, no estamos convencidos de que con el grado de desarrollo de los estudios teóricos sobre la derivación se pueda conseguir un tratamiento verdaderamente útil de la información contenida en los sufijos derivativos. Es decir, el problema fundamental no es la concatenación de las cadenas (esto es simplemente una cuestión de eficiencia computacional, que unos sistemas resuelven mejor que otros), sino el tipo de información que debe llevar un sufijo determinado y cómo se puede utilizar esa información en los distintos niveles de la gramática (esto es una cuestión teórica sin resolver).

Sobre el primer punto, pensemos que es muy difícil determinar el significado exacto de muchos sufijos y prefijos (como por ejemplo, *-mento*). Nos podríamos preguntar por qué necesitamos tratar cadenas que no aportan ninguna información concreta. En otros casos, una misma cadena tiene diferentes funciones/significados. Esto implica una complejidad añadida, pues hay que distinguir esa variación de información. En cuanto al tema del uso de la información de los sufijos derivados en otros niveles gramaticales supone, entre otras cosas, el paso de la información pertinente a través de los distintos constructos gramaticales (sintagmas, oraciones) hasta llegar a los niveles interpretativos. No vamos a descubrir ahora la dificultad tanto computacional como teórica que esto entraña.

El estudio teórico de la derivación ha apuntado soluciones parciales pero creemos que no hay todavía una base sólida para extender un tratamiento determinado no ya a todos los sufijos derivativos sino simplemente a los más productivos. En la actualidad es un tema de investigación prioritario en algunas teorías lingüísticas y acapara el esfuerzo de reconocidos especialistas. No entraba, por tanto, entre las pretensiones teóricas de esta tesis, orientada particularmente a la aplicación computacional de conceptos teóricos contrastados.

Por el contrario, la flexión tiene un información explícita e inequívoca que ha permitido concentrarnos en la tarea de diseñar un modelo que dé cuenta de cómo se

transmite esa información y de cómo se concatenan los elementos superficiales morfológicos a los que está asociada dicha información.

7.2.4 Algunas consideraciones sobre la gramática

En este apartado analizaremos dos cuestiones: la primera tiene que ver con las *posibilidades computacionales* del modelo presentado; la segunda versará sobre la *elegancia teórica* de la gramática.

Hemos visto que nuestra gramática está directamente codificada en Prolog, debido a que su simplicidad y declarativiad lo permite. Pero también se ha comentado en varias ocasiones que para conseguir un resultado satisfactorio del programa de aplicación es necesario que se incluyan módulos "procedurales" que mejoren la velocidad de procesamiento. Como señala Grishman(1986) la ventaja de Prolog sobre los otros lenguajes de programación es que el lingüista se puede olvidar de escribir un algoritmo de *parsing*, pues el motor de inferencia de Prolog incluye un *parser* descendente de izquierda a derecha (*top-down, left-to-right parser*). Esto es lo que hemos hecho nosotros y el proceso de análisis es extremadamente lento. Pensemos que Prolog prueba cada una de las reglas con cada una de las entradas de diccionario para encontrar todas las soluciones posibles a la consulta.[111]

Debido a esto, nos hemos visto obligados a introducir en nuestro programa de demostración GRAMPAL un *segmentador*, muy sencillo pero que mejora la eficiencia considerablemente. Este segmentador es simplemente una rutina iterativa que compara la cadena de entrada (en la fase de análisis, claro está) con las cadenas de las entradas del diccionario, de tal forma que si encuentra alguna subparte de la cadena de entrada en la cadena del diccionario toma dicha entrada y la coloca en un fichero. De esta manera se obtienen entre 10 y 20 segmentaciones por palabra, sin discriminar si las secuencias segmentadas son permitidas o no. Con este fichero de muy pocas entradas de diccionario es con el que trabaja la gramática, consiguiendo respuestas en tiempos inapreciables. Por supuesto, el segmentador se puede mejorar de múltiples maneras (con una búsqueda determinística, con un algoritmo de búsqueda de estados finitos como el que mostramos en la página 61, etc.) pero consideramos que esta es una labor para un especialista de la computación. Igualmente, se puede desarrollar un *parser* de reglas gramaticales con vistas a la ampliación de la gramática para tratar fenómenos sintácticos.

Hay que tener en cuenta, además, que trasladar la codificación de la gramática a otra versión de Prolog es muy sencilla y directa: basta con adaptarla a las especificaciones sintácticas de la versión en cuestión. Evidentemente, si utilizamos ordenadores más potentes (y en ocasiones mucho más potentes) que los personales, el tiempo de procesamiento disminuye en la misma proporción. De hecho, todas las aplicaciones "serias" que se han desarrollados están implementadas para ordenadores del tipo *workstation* o superior.

---

[111] En cambio, para la generación esto no afecta apenas pues las variables instanciadas son mucho más fáciles de tratar. En el reconocimiento de palabras, sin embargo, no tenemos nada más que la cadena y hay que probar todas las combinaciones posibles con los datos que se tienen.

Otra cuestión que está relacionada con las posibilidades computacionales de nuestro modelo es la complejidad (o facilidad) de uso y codificación del diccionario. Queremos insistir en el hecho de que esta tarea es muy engorrosa y compleja si se utiliza un modelo de estados finitos:

> One of the major problems we have found in our work is that although formalisms appear simple when described and initially implemented, actual use often shows them to be complex and difficult to use [...] these morphographemic formalisms require a form of debugger to allow the writer to test the rule set quickly and find its short-comings. Hence we have implemented a debugger for the Koskenniemi Formalism. (Black *et al.* 1987:17)

Como estos mismos autores señalan, la dificultad de tales modelos reside, una vez traducidas las reglas del formalismo a tablas de autómatas finitos, en depurar los errores que se han cometido en la codificación[112] : "debugging of our spelling rules was very difficult and time consuming."

Por el contrario, codificar entradas de diccionario con nuestro sistema es muy sencillo. En GRAMPAL hay una opción para "Actualizar el diccionario" con la que el usuario puede fácilmente añadir nuevas entradas, sin tener que escribir la entrada en su formato Prolog real. El sistema le pide que introduzca los valores correspondientes a cada rasgo y se encarga de crear automáticamente la entrada y añadirla al diccionario. Esta opción evita además que el usuario cometa "errores de forma," aunque por supuesto no reconoce si los datos proporcionados son correctos. Pero gracias a que las entradas léxicas están compuestas de valores (fáciles de distinguir) para los distintos rasgos, esto facilita la localización y corrección de los errores. No hace falta ninguna "herramienta" especial para detectarlos, ya que la codificación es explicativa por sí misma. Hay que tener en cuenta, además, que todos los casos significativos y los más complejos han sido ya codificados, por lo que la persona que se encargue de ampliar el diccionario sólo tiene que seguir (en la mayoría de los casos) los modelos propuestos. Se puede concluir, por tanto, que nuestro método es fácil de usar porque:

- existe una descripción pormenorizada de la distribución de los alomorfos y de los rasgos que llevan asociados,
- la "legibilidad" de la entrada de diccionario permite localizar y corregir los errores sin mucho esfuerzo.

Con respecto a la *elegancia teórica* de la gramática, nos parece muy significativo citar este comentario de Gazdar y Mellish(1989) sobre los requisitos generales que deben cumplir las formalizaciones teóricas:

> In addition to these empirical criteria [whether a grammar assigns the correct structures], standard scientific considerations such as simplicity and generality apply to grammars in much the same way as they do to any other theories about natural phenomena. Other things being equal, a grammar with seven rules is to be preferred to one with 93 rules. And, other things being equal, a grammar for a fragment of English that can be converted into a grammar for the corresponding fragment of French with a few minor changes is to be preferred to a grammar for the English fragment that bears no relation whatsoever to its French

---

[112] Esto se debe fundamentalmente a que estas codificaciones están escritas en un formato poco "legible." Compárese por ejemplo una tabla de autómata con una entrada de nuestro diccionario.

> counterpart. Likewise, a grammar for Swedish relative clauses that can be extended to handle a class of Swedish questions with the addition of a couple of rules and a feature is to be preferred to one that requires a complete new set of parallel rules and features to encompass the questions. (Gazdar y Mellish 1989:109-110).

En cuanto a la preferencia de un número limitado de reglas para dar cuenta de los mismos fenómenos, tenemos que señalar que en nuestra gramática lo hemos conseguido con un número verdaderamente reducido (siete) frente a la prolijidad de otros modelos.

La cuestión de la *universalidad* frente a la *especificidad* de un modelo o teoría tiene importancia singular. Creemos que nuestra propuesta se puede adaptar sin muchos cambios a las lenguas romances y, en general, a las flexivas. Pensamos que es un modelo muy adecuado para tratar problemas como la modificación interna. Por el contrario, el modelo de estados finitos parece más apropiado para tratar los fenómenos morfológicos de las lenguas, donde las variantes alomórficas se producen en contextos fonológicos y gráficos bastante regulares. Quizás, la conclusión de todo ello sería que no debemos hablar de un modelo universal para el tratamiento computacional de la morfología, sino de modelos que se adaptan mejor a un tipo o a otro de lenguas.

Con respecto a la cuestión de si nuestra gramática se puede extender sin grandes modificaciones a otros fragmentos del español, en la sección anterior mostramos cómo podrían tratarse la derivación y la composición, siguiendo un método similar al utilizado para la flexión.

Por último, insistir en el hecho de que el programa se asemeja lo más posible a la especificación formal. Se demuestra así su carácter declarativo.

8.0 Conclusiones

Creemos que nuestra propuesta reúne los requisitos de los formalismos gramaticales enunciados por Shieber (1986):

1. **adecuación lingüística**: nuestra descripción de los fenómenos flexivos del español se basa en la representación gráfica de cada forma flexionada. Siguiendo este criterio formal riguroso proporcionamos una clasificación de morfemas y alomorfos gráficos que puede ser útil para cualquier aplicación informática, independientemente del modelo elegido.

2. **expresividad**: el formalismo es bastante expresivo en el sentido de que es capaz de describir todas las formas posibles y correctas del español. En ese sentido, sigue la tendencia de los formalismos orientados al procesamiento del lenguaje natural de tener un fuerte poder expresivo para conseguir mayor eficiencia. Por otra parte, motivados por consideraciones teóricas hemos restringido la expresividad imponiendo condiciones sobre el uso y aparición de valores en disyunción. De esta manera, se consigue que la gramática analice y genere sólo las formas gramaticales, y rechace las agramaticales.

3. **eficiencia computacional**: la gramática se ha implementado directamente en Prolog, sin necesidad de utilizar un compilador del formalismo, gracias a la sencillez y abstracción de las reglas gramaticales. Proporciona resultados satisfactorios con un diccionario reducido pero que contiene todos los casos significativos de la morfología del español. La eficiencia del sistema se puede mejorar considerablemente utilizando ordenadores más potentes que los personales e incorporando técnicas procedurales (mejora del segmentador y utilización de un parser, por ejemplo). La portabilidad a otro sistema en Prolog es directa.

El mecanismo de concatenación morfológica es la característica más innovadora de nuestro formalismo. Se trata de una extensión notacional que permite dar cuenta del contexto, dentro de una gramática sintagmática independiente del contexto. Se basa en el uso de la operación de concatenación y en la comprobación de las condiciones de combinación de dos cadenas. Para esto último utiliza rasgos que pueden llevar valores en disyunción. Dicho mecanismo puede ser compatible con otros formalismos de unificación, que lo pueden incorporar como un elemento específico para tratar la morfología. La comprobación de esta posibilidad sería una prueba de su *portabilidad* a otros sistemas y lenguas. Las pretensiones de la tesis no incluyen el estudio de la universalidad del modelo, pero se encuentra entre los objetivos prioritarios del desarrollo posterior.

La sencillez y generalidad de las reglas morfológicas se basa en la concepción lexicalista de la gramática, que concentra la especificación de la información no en las

reglas gramaticales, sino en la trasmisión de rasgos mediante la operación de unificación. De hecho, las reglas son simplemente especificaciones de estructuras de rasgos llenas de variables que se instancian con la información contenida en las entradas de diccionario. Esta es la característica más sobresaliente de los enfoques basados en gramáticas morfosintácticas con rasgos, frente a los modelos de estados finitos o los paradigmaticos cuya estrategia reside en el uso exhaustivo de reglas para aportar la información. La experiencia de los últimos años en lingüística computacional ha demostrado que las gramática basadas en rasgos son especialmente eficientes en el tratamiento de la información sintáctica, de lo cual se deduce que extender este tipo de reglas a la morfología permite tratar todo el proceso desde los morfemas hasta la oración con la misma gramática.

La segunda novedad que presenta nuestra gramática es su *declarativídad* o su *bidireccionalidad.* Es decir, expresar la gramaticalidad de las palabras independientemente de si se encuentran en una fase de análisis o de generación es un requisito de elegancia teórica que hasta el momento no se daba en las gramáticas morfosintácticas convencionales. Esta crítica es uno de los argumentos fundamentales de los defensores del modelo de dos niveles: habitualmente se utilizan gramáticas diferentes para el análisis y generación de palabras. Nosotros hemos conseguido la declarativídad al hacer una descripción detallada de todas las asociaciones permitidas entre distintos alomorfos, mediante los rasgos especiales de concatenación. En realidad, la declarativídad es una consecuencia, más que una novedad, de nuestra formalización y del uso del mecanismo de concatenación morfológica.

Por otra parte, la tesis defendida tiene implicaciones teóricas, aunque sólo sean por referencia indirecta. Es una prueba adicional de que gramáticas basadas en rasgos puede tratar fenómenos morfológicos y sintácticos al mismo tiempo y un argumento práctico para defender que no es pertinente la existencia de un componente morfológico autónomo, con reglas específicas distintas de las sintácticas. Nuestro enfoque teórico supone además una reelaboración del modelo *Elementos y Colocación* sobre la base de los formalismos de unificación: los elementos morfológicos superficiales se concatenan, no existen elementos superficiales nulos y las unidades discretas (morfemas) se establecen en función de la información inequívoca que llevan asociada las cadenas superficiales. Por último, desde el punto de vista de la adecuación formal de los tipos de gramáticas, nuestra tesis es un ejemplo de cómo es posible dar cuenta del contexto morfológico mediante una gramática independiente del contexto ampliada.

Apéndice: programa de demostración GRAMPAL

GRAMPAL es un programa escrito para servir de entorno de demostración de nuestra gramática de generación y análisis. Por lo tanto, el programa está supeditado a la gramática y concebido como un mero instrumento de acceso. Para codificarlo se ha escogido Turbo-Prolog, una versión de Prolog que presenta, junto con todas las ventajas del Prolog interpretado, la de ser compilable y capaz de funcionar dentro de DOS con unos requisitos de memoria muy pequeños. Podríamos haber utilizado otros tipos de Prolog concebidos para máquinas más potentes (por ejemplo, para máquinas UNIX o *mainframes*) con lo que hubiéramos ganado en velocidad y capacidad de almacenamiento, pero por razones prácticas nos pareció aconsejable utilizar el tipo de ordenadores más asequibles. Gracias a la posibilidad de compilar el programa la velocidad aumenta considerablemente. Algunas de las características de GRAMPAL obedecen a particularidades de Turbo-Prolog. En este capítulo haremos una descripción del programa y su funcionamiento paso a paso.

El programa comienza con una declaración de los tipos a los que corresponde cada variable (sección DOMAINS) y de los predicados que se van a utilizar tanto en la base de datos (sección DATABASE) como en el programa en sí (sección PREDICATES). El número de argumentos que toma cada predicado, así como el tipo al que estos argumentos pueden pertenecer está fijado en la declaración y debe permanecer invariable durante todo el programa. Esta declaración es un requisito de Turbo-Prolog que la utiliza para la compilación y que además sirve para comprobar que durante la ejecución las variables no tomen valores de tipo diferente del que se pretende así como para asegurarse de que los predicados están correctamente escritos. Por ejemplo, la declaración

```
formaflex, cadenar, cadenad, cadenag, cadenap  = string
```

implica que las variables pertenecientes a los tipos `formaflex`, `cadenar`, `cadenad`, `cadenag` y `cadenap` pertenecen al tipo que se conoce en Prolog como **string**, es decir, son cadenas de caracteres. Mientras que `tipo_raiz2 = tipo_raiz(integer)` implica que el argumento que toma `tipo_raiz` es un número entero. De forma análoga, la declaración

```
verbo_lex(formaflex,nivel,ul,persnum,tiempmod,formav).
```

quiere decir que vamos a usar un predicado al que llamaremos `verbo_lex` que toma siete argumentos. El primero de ellos pertenece al dominio de `formaflex` (= string), el segundo al dominio de `nivel` (= symbol), el tercero al de `ul` (= string) y así sucesivamente. De esta forma se consigue que si en alguna parte del programa usamos

este predicado de una forma no coherente con la declaración -como por ejemplo, pasándole un número de argumentos equivocado-, Turbo-Prolog sea capaz de reconocer el error y avisarnos. Otro de los requisitos de Turbo-Prolog que ha condicionado la disposición del programa es que el compilador exige que todas las cláusulas que se refieren a un mismo predicado estén agrupadas una a continuación de la otra.

La función de cada uno de estos predicados se irá mostrando en detalle a lo largo de la explicación. En el programa aparecen también algunos predicados auxiliares que están declarados en otros ficheros y que no vamos a examinar aquí. Estos ficheros se incorporan a GRAMPAL mediante los predicados de inclusión (`include`) que se encuentran en la cabecera del programa.

A continuación sigue la lista de estas declaraciones que no se comenta detalladamente porque no tiene gran interés para el desarrollo del programa.

```
/*************************************************************************/
/*                    GRAMPAL : Gramática de la palabra                  */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
code = 4000
nowarnings


include "tdoms.pro"
include "tpreds.pro"
include "menu2.pro"
include "carnum.pro"


DOMAINS

nivel, ul, persnum, tiempmod, formav, cat, gen, num = symbol
formaflex, cadenar, cadenad, cadenag, cadenap  = string
conjugacion1 = conjugacion(symbol)
conjugacion2 = conjugacion(listasym)
tipo_raiz1 = tipo_raiz(lista)
tipo_des1 = tipo_des(listasym)
tipo_raiz2 = tipo_raiz(integer)
tipo_des2 = tipo_des(symbol)
tipo_gen1 = tipo_gen(listasym)
tipo_gen2 = tipo_gen(symbol)
tipo_plu1 = tipo_plu(listasym)
tipo_plu2 = tipo_plu(symbol)


file = grampal; todasformas

DATABASE

verbo_lex(formaflex,nivel,ul,persnum,tiempmod,formav).
raiz_verb(cadenar,nivel,ul,conjugacion1,tipo_raiz1,tipo_des1)
desinencia(cadenad,persnum,tiempmod,formav,conjugacion2,tipo_raiz2,tipo_des2)

blanco(string).

nom(cadenar,nivel,cat,ul,gen,num,tipo_gen1,tipo_plu1).
morf_gen(cadenag,gen,tipo_gen2).
morf_plu(cadenap,num,tipo_plu2).

fin_de_diccionario.

PREDICATES

/* predicados principales */
grampal.
```

```
analiza.
analiza1.
analiza2(STRING).
continuar_analizando.
error_analizando.
genera.
menu_genera.
genera1(INTEGER).
genera_verbo.
genera_nombre.
genera1bis_nom(STRING).
genera1bis_verb(STRING).
genera_form_comp(STRING).
genera_form_compbis(STRING).
error_generando_verbo.
error_generando_nom.
continuar_generando_paradigm_nom.
error_generando_paradigm_nom.
continuar_generando_paradigm_verbo.
error_generando_paradigm_verbo.
process(INTEGER).


/* PREDICADOS DE AYUDA */
ayuda_nivel.
ayuda_ul.
ayuda_cjg.
ayuda_tipo_raiz.
ayuda_tipo_des.
ayuda_pers_num.
ayuda_tiemp_mod.
ayuda_formav.
ayuda_cadena.
ayuda_cat.
ayuda_gen.
ayuda_num.
ayuda_tipo_gen.
ayuda_tipo_num.



/* PREDICADOS DE LA GRAMATICA */

/* morfología verbal */
verbo(nivel,ul,persnum,tiempmod,formav,formaflex)

/* morfología nominal */
nominal(nivel,cat,ul,gen,num,string)


/* PREDICADOS DE CONCATENACION */
```

```
member(integer,lista)
member(symbol,listasym)
concatena_v(conjugacion1,conjugacion2,tipo_raiz1,tipo_raiz2,tipo_des1,
            tipo_des2,cadenar,cadenad,formaflex)
concatena_vcomp(tiempmod,tiempmod,formaflex,formaflex,formaflex)
concatena_n1(tipo_gen1,tipo_gen2,string)
concatena_n2(tipo_gen1,tipo_gen2,string,string,string)
concatena_n3(tipo_gen1,tipo_gen2,string,string,string,string)


/* predicados de actualización del diccionario */
updatdba.
updatdba1(integer).
comprobacion(integer).
salida(STRING,INTEGER).
```

La sección GOAL contiene los predicados que comienzan el programa. Un programa típico en Prolog interpretado suele estar formado por una serie de predicados que constituyen las cláusulas o reglas. Para que estas reglas se pongan en marcha el usuario tiene que formular interactivamente una pregunta al programa. Pero en el caso de GRAMPAL, se trata de un programa ejecutable que tiene que arrancar sin la intervención del usuario. La sección GOAL tiene la función de invocar un predicado o una serie de predicados unidos entre sí por la conectiva ',' (que equivale a la conectiva lógica 'y') que pondrán en marcha el resto del programa. Concretamente los predicados `makewindow` son predicados de los que dispone Turbo-Prolog para construir ventanas y a los que hay que proporcionar una serie de parámetros de tamaño y color y, en su caso, el encabezamiento. Mediante estos predicados construimos una ventana de tamaño equivalente a la pantalla con otra ventana más pequeña en su interior. A continuación, usando los predicados `cursor` y `write` nos vamos situando en distintos puntos de la pantalla para escribir el mensaje de presentación. El predicado `readchar` capta un carácter desde el teclado, es decir, el predicado tiene éxito si detecta que se ha pulsado una tecla cualquiera. Gracias a él conseguimos que las ventanas que hemos construido permanezcan en pantalla hasta que el usuario decida pulsar una tecla. En ese momento, el programa intentará satisfacer el siguiente predicado. Hasta ahora, todos los predicados que se han usado en esta sección GOAL corresponden al tipo de los predicados comúnmente llamados `built-in`, es decir, predicados que forman parte del propio lenguaje y que Prolog puede interpretar sin necesidad de que se los definamos. Sin embargo, lo característico de un programa Prolog es que el programador cree un sistema propio de predicados y de reglas. El último predicado de la sección GOAL corresponde a este tipo. `Grampal` es un predicado que no toma ningún argumento y para satisfacer esta meta Turbo-Prolog tendrá que buscar una cláusula cuya cabeza sea el predicado `grampal` e intentar satisfacer todos los predicados que contenga. Por el momento, todos los predicados de la sección GOAL han sido satisfechos, ahora Turbo-Prolog recorrerá el fichero hasta dar con la cláusula o cláusulas correspondientes a `grampal`.

```
GOAL
     makewindow(1,112,3," GRAMPAL ",0,0,25,80),
     makewindow(10,0,0,"",9,21,9,40),
     makewindow(11,27,0,"",8,20,9,40),
     shiftwindow(11),
     cursor(1,13), write("G R A M P A L "),
     cursor(3,8), write("Gramática de la Palabra"),
     cursor(5,8), write("Antonio Moreno Sandoval"),
     cursor(7,13),write("Tesis Doctoral"),
     readchar(_),
     removewindow(11,1), removewindow(10,1),
     grampal.
```

Turbo-Prolog no tendrá que buscar demasiado, puesto que grampal es el primer predicado que se encuentra dentro de la sección de reglas o cláusulas. La función de este predicado es mostrar un menú principal con varias opciones. Para construir este menú se han usado predicados built-in y otros predicados auxiliares. Sobre el menú aparecerá resaltada la opción vigente y el usuario puede moverse con las teclas del cursor hasta situarse sobre la opción que desee. Si el usuario pulsa la tecla de entrada (RETURN, INTRO, ENTER) el número de la opción en la que se encuentre se tomará como argumento del siguiente predicado (process) que pasa a ser la meta que hay que satisfacer.

```
CLAUSES


/*-----------------------------------------------------------------
                    M E N U    P R I N C I P A L
-----------------------------------------------------------------*/

grampal :-
           repeat,
           shiftwindow(1),
           clearwindow,
         menu(5,27,103,7,
         [ "",
           "   Analizar una palabra",
           "   Generar una palabra",
           "",
           "Ver/actualizar diccionario",
           "   Editar el diccionario",
           "     Editar un fichero",
           "",
           "  Introducción a GRAMPAL",
           "   Sistema Operativo DOS",
           "     Salir de GRAMPAL",
           ""]," MENU PRINCIPAL ",9,
   CHOICE) ,
   process(CHOICE),
   CHOICE=0,'.
```

Si el usuario pulsa sobre una línea en blanco, el argumento que toma process es 0, por lo cual intentará satisfacer la siguiente cláusula:

```
process(0):-
    write("\n\n\n\n  ¿ Está seguro de querer salir de GRAMPAL ? (s/n): "),
    readchar(T),
    T='s',
    removewindow(1,1),
    exit.
```

Como se puede deducir, esta cláusula pregunta al usuario si desea abandonar la ejecución del programa. Si la respuesta es 's', la ventana desaparece y el programa termina. Si el usuario da cualquier otra respuesta process(0) falla y Prolog intenta satisfacer otra vez grampal, por lo que retorna al menú.

Dedicaremos a continuación un apartado a cada una de las opciones que aparecen en este menú principal.

```
/*-----------------------------------------------------------
          A N A L I Z A R   U N A   P A L A B R A
-----------------------------------------------------------*/
```

Si el usuario eligió la opción **Analizar una palabra** el argumento que se pasa a process es 2 (la posición 1 del menú está en blanco). El programa intenta entonces satisfacer el predicado

```
process(2) :- analiza.
```

El cuerpo de esta cláusula consiste en una llamada al predicado analiza que no toma ningún argumento. El programa va a buscar la regla correspondiente y entra en la parte de análisis. analiza es un predicado cuya misión es vaciar la memoria, construir las ventanas del entorno de análisis y llamar al predicado de análisis propiamente dicho, que es analiza1.

```
analiza :-
    retractall(_),
    removewindow(4,1), removewindow(5,1),
    clearwindow,
    makewindow(2,30,11," Analizar ",0,0,25,80),
    makewindow(3,48,11," Mensajes ",19,20,4,52),

    analiza1.
```

La cláusula analiza1 cumple varias funciones importantes. La primera es pedir al usuario que introduzca la palabra que desea analizar. El predicado readln captura la entrada del teclado. Se diferencia de readchar en que puede tomar un número indeterminado de caracteres. Es importante que el usuario no introduzca ningún carácter que no pertenezca a la palabra, como por ejemplo, espacios en blanco detrás o delante, signos de puntuación, etc, porque se tomarán como parte de la cadena que se quiere analizar y harán fallar el análisis. Por supuesto se deben escribir blancos en las formas compuestas de los verbos. A continuación se comprueba que la cadena de caracteres no esté vacía (o formada por blancos). Esta comprobación sirve para evitar que se inicie el proceso de análisis si el usuario pulsó ENTER por error antes de haber

introducido la palabra. Luego se convierte la cadena a minúsculas, que es la forma en la que aparecen en el diccionario, mediante el predicado upper_lower. Esta conversión evita el fallo en el caso de que se usen mayúsculas en la cadena de entrada. Los siguientes predicados se encargan de realizar una llamada al programa de segmentación SEGMENTA.C. Este programa tiene su documentación aparte y aquí sólo diremos que el resultado de la segmentación es un diccionario de tamaño reducido que contiene sólo las entradas léxicas que pueden ser necesarias para el análisis (las posibles segmentaciones). En esa documentación se explica más detalladamente la función de los predicados str_char y concat. El diccionario de segmentación se carga en la memoria del programa mediante el predicado consult y es el único que se usará durante el análisis. El segmentador se encarga de borrar el fichero antiguo y crear uno nuevo para cada análisis. Por último, se produce la llamada al predicado analiza2 que toma como único argumento la cadena que queremos analizar.

```
analiza1 :-
    shiftwindow(3),
    cursor(1,2), write("  Escriba una palabra.   Pulse ESC para salir "),
    gotowindow(2),cursor(1,11),
    readln(Cadena1), Cadena1 >< " " ,
    upper_lower(Cadena1,Cadena),
    str_char(Comilla,'"'),
    concat("segmenta ",Comilla,Llamada1),
    concat(Llamada1,Cadena,Llamada2),
    concat(Llamada2,Comilla,Llamada),
    system(Llamada,0,_),
    consult("segmenta.dic"),
    analiza2(Cadena).
```

Una vez que la variable Cadena está ya instanciada con la palabra que se va a analizar, por ejemplo, analiza2("he cantado"), GRAMPAL va a intentar encontrar en primer lugar un análisis como verbo. Para ello se produce la llamada al predicado verbo que, como se ha visto anteriormente, es uno de los predicados de la gramática. En este momento la única variable instanciada es la correspondiente al argumento Cadena. No nos extenderemos ahora en el funcionamiento de este predicado, puesto que ya se ha visto en la sección dedicada a la gramática. Baste decir que si la palabra tiene algún análisis como verbo, las variables Nivel, UL, Pers_num, Tiemp_Modo y FormaV retornarán ligadas a sus valores correspondientes. En el caso de "he cantado", el predicado retornará lo siguiente: verbo(nivel_0,"cantar",sg1,pret_perf,fin,"he cantado") Los predicados siguientes se encargan de mostrar los resultados en pantalla, escribiendo el nombre del rasgo junto al valor que ha resultado del análisis. Podría darse el caso de que una misma palabra tenga dos o más análisis diferentes: por ejemplo, formas que sirven para la primera y la tercera del singular o palabras que pertenecen a dos unidades léxicas diferentes. Esto no es problema ya que el funcionamiento de Prolog hace que una vez que ha satisfecho las condiciones de la cláusula con un conjunto de valores para las variables, intente encontrar otros valores de las variables que satisfagan también las condiciones de la regla. En otras palabras, el programa busca todos los análisis posibles. Por lo tanto, se le pide al usuario que pulse una tecla y el predicado vuelve a intentar otro análisis y a mostrarlo en la pantalla a continuación del anterior.

```
analiza2(Cadena):-

      verbo(Nivel,UL,Pers_Num,Tiemp_Modo,FormaV,Cadena),  <-- Llamada a gramática
      write("\n  Nivel          = ", Nivel), nl,
      write("\n  Unidad Léxica  = ", UL), nl,
      write("\n  Persona_Numero = ", Pers_Num), nl,
      write("\n  Tiempo_Modo    = ", Tiemp_Modo), nl,
      write("\n  Forma Verbal   = ", FormaV, "\n "),
      write("\n  OBJETO Prolog:   verbo(",Nivel,",", UL ,  ",", Pers_Num,
                  ",", Tiemp_Modo, ",", FormaV, ",", Cadena, ").\n"),

      gotowindow(3),
      resizewindow(1,34,4,45), clearwindow,
      cursor(1,2), write("    Pulse cualquier tecla para ver
                              otro posible análisis"),
      gotowindow(2),
      readchar(_).
```

Cuando ya no hay otro conjunto de variables que satisfaga la cláusula, el predicado falla y Prolog sigue buscando otra regla cuya cabeza unifique con el predicado `analiza2(Cadena)`.

Prolog efectivamente encuentra otra cláusula que unifica con la meta `analiza2(Cadena)`. En primer lugar intentará satisfacer `nominal(Nivel,Cat,UL,Gen,Num,Cadena)` que es el predicado encargado del análisis de las formas nominales en la gramática. Lo mismo que ocurría con el verbo, si existe algún análisis las variables retornarán ligadas a sus valores. Los restantes predicados se encargan de la presentación en pantalla de los resultados y de pedir al usuario que pulse una tecla para continuar. Este predicado también se satisfará tantas veces como análisis haya.

```
analiza2(Cadena):-
      nominal(Nivel,Cat,UL,Gen,Num,Cadena),  <-- Llamada a la gramática
      write("\n    Nivel         = ", Nivel), nl,
      write("\n    Categoría     = ", Cat), nl,
      write("\n    Unidad Léxica = ", UL), nl,
      write("\n    Género        = ", Gen), nl,
      write("\n    Número        = ", Num, "\n"),
      write("\n  OBJETO Prolog:   nominal(",Nivel,",", Cat , ",", UL,
                              ",", Gen , ",", Num , ",", Cadena, ").\n"),
      gotowindow(3),
      resizewindow(1,34,4,45), clearwindow,
      cursor(1,2), write("     Pulse cualquier tecla para ver
                              otro posible análisis"),

      gotowindow(2),
      readchar(_) .
```

Una vez que el programa ha agotado todos los análisis como verbo o como nominal, probará también la tercera cláusula para `analiza2(Cadena)`. El programa llega a este tercer predicado `analiza2(Cadena)` después de que los dos anteriores han fallado. Esto puede ocurrir cuando no se encuentra ningún análisis para la palabra o

bien cuando ya se han agotado todos los posibles. Por ello, este predicado emite un mensaje que informa al usuario de que el proceso de análisis ha concluido. También se ocupa de vaciar la memoria de la base de datos de segmentación y llamar al predicado continuar_analizando para volver a iniciar el proceso.

```
analiza2(Cadena):-
      gotowindow(3), clearwindow, cursor(1,1),
      write(" No hay ningún análisis para esa palabra o bien \n
             no se encontraron más análisis       "),
      readchar(_),
      retractall(_),
      continuar_analizando.
```

El predicado continuar_analizando es el encargado de limpiar la pantalla y llamar de nuevo a analizal de forma que pueda comenzarse otra vez todo el proceso con otra palabra. Además lleva a cabo algunas tareas de manejo de ventanas que no explicaremos.

```
continuar_analizando:-
      gotowindow(3),
      resizewindow(19,22,4,52), clearwindow,
      gotowindow(2), clearwindow,
      analizal.


/*-----------------------------------------------------------------
                    G E N E R A C I Ó N
-----------------------------------------------------------------*/
```

Si la opción elegida por el usuario fue generar una palabra, el predicado process se encargará de llamar al predicado genera.

```
process(3) :- shiftwindow(1), genera.
```

El predicado genera tiene varias funciones; primero vacía la memoria (retractall), a continuación carga el diccionario (en este caso el diccionario completo) en memoria y por último llama al menú de generación. Mientras se está cargando el diccionario, aparece un mensaje en la pantalla que advierte al usuario de la operación que se está realizando.

```
genera :-
    shiftwindow(1),
    removewindow(2,1), removewindow(3,1),
    clearwindow, cursor(15,12),
    retractall(_),
    write(" >>  cargando el diccionario, espere unos segundos"),
    consult("grampal.dic"),
    clearwindow,
    menu_genera.

/*-----------------------------------------------------------------
            M E N Ú   D E   G E N E R A C I O N
-----------------------------------------------------------------*/
```

A diferencia del análisis (donde se le ofrecía al usuario una sola posibilidad), la generación dispone de un menú con varias opciones. El usuario puede elegir entre Generar un verbo, Generar un nombre o adjetivo, Generar todas las formas disponibles, Generar un paradigma nominal completo o Generar un paradigma verbal. La razón para hacerlo así en lugar de ofrecer una sola opción es que los datos que se le van a pedir al usuario varían de una categoría a otra. Por lo tanto, resultaba menos confuso presentarlas separadamente.

```
menu_genera:-
    repeat,
    shiftwindow(1), clearwindow,
    menu(10,25,7,7,
     ["Generar un verbo",
      "Generar un nombre o adjetivo",
      "Generar todas las formas",
      " ",
      "Generar un paradigma nominal",
      "Generar un paradigma verbal" ],"Menú de generación",
      1,CHOICE),
    general(CHOICE),
    CHOICE=0, ' .

/*-----------------------------------------------------------------
            G E N E R A R   U N A   P A L A B R A
-----------------------------------------------------------------*/
```

Si el usuario escoge la primera opción, Generar un verbo, se producirá la llamada al predicado general(1) que es el encargado de la generación verbal. La misión de este predicado es crear una serie de ventanas y, lo más importante, llamar al predicado genera_verbo, que es el que efectuará el proceso de síntesis. Si la palabra que hemos solicitado no está en el diccionario o si se produce cualquier otra circunstancia por la que es imposible generar esa forma, el programa nos avisará mediante un mensaje de error producido por el predicado error_generando, que se activa al producirse el fallo en la parte precedente de la cláusula (concretamente en genera_verbo), gracias a la conectiva ';' ('o').

```
general(1) :-

    makewindow(41,48,15," Generación verbal ",0,0,25,80),
    makewindow(51,31,15," Mensajes ",15,12,8,54),
    cursor(2,1), write("     Escriba los datos necesarios "),
    genera_verbo ; error_generando_verbo.
```

El predicado genera_verbo solicita al usuario que introduzca la información que es necesaria para generar una forma concreta. En el caso del verbo estos datos son:

- **Forma léxica** (el infinitivo del verbo que deseamos generar)
- **Persona y número**: *sg_1, sg_2, sg_3, pl_1, pl_2, pl_3.*
- **Tiempo y modo**: *pres_ind, impf_ind, pret_ind, futuro, pres_subj, imp_subj, cond, imper, infin, ger, part, pret_perf, pret_plus, pret_ante, futur_perf, pret_perf_subj, cond_perf.*

a medida que el usuario se va moviendo de un campo a otro para rellenar los distintos datos, el programa se encarga de ir mostrando una ayuda en la ventana de mensajes. Estos predicados de ayuda se verán más adelante. Pero la generación no se produce hasta que no se llama al predicado `verbo(nivel_0,UL,Persona,Tiempo,FormaV,Cadena)` que es el que corresponde a la gramática. A diferencia del análisis, todas las variables están instanciadas excepto `Cadena`. Por ejemplo, si deseamos generar la tercera persona del singular del pretérito perfecto de indicativo del verbo 'escribir', tenemos que suministrar a GRAMPAL los siguientes datos:

- **UL**: ***escribir***
- **Persona y Número**: ***sg_3***
- **Tiempo y Modo**: ***pret_perf***

con lo cual se llamará al predicado verbo de la siguiente manera: `verbo(nivel_0,"escribir",sg_3,pret_perf,fin,Cadena)` y la gramática nos devolverá la forma generada: `verbo(nivel_0,"escribir",sg_3,pret_perf,fin,"ha escrito")` que será mostrada en la pantalla mediante el predicado `write(Cadena)`

```
genera_verbo:-

    gotowindow(41),
    field_str(3,18,18,"Unidad léxica    ="),
    field_str(5,18,18,"Persona y Número ="),
    field_str(7,18,18,"Tiempo y Modo    ="),
    cursor(11,18),
    write("FORMA GENERADA ="),
    makewindow(88,54,0,"",11,38,2,25),
    gotowindow(41),

    cursor(3,40), ayuda_ul, gotowindow(41),       readln(UL),
    cursor(5,40), ayuda_pers_num, gotowindow(41), readln(Persona),
    cursor(7,40), ayuda_tiemp_mod, gotowindow(41), readln(Tiempo), ',
    verbo(nivel_0,UL,Persona,Tiempo,FormaV,Cadena),  <-- llamada a la gramática
    gotowindow(88), cursor(1,1),
    write(Cadena),

    gotowindow(51), clearwindow, cursor(1,1),
    write("     Pulse cualquier tecla para ver    \n
               otra forma posible    \n
               generada con estos datos.       "),
    readchar(_),
    gotowindow(88), clearwindow.
```

Después de mostrar la cadena generada, el usuario tiene que pulsar una tecla para que el programa continúe e intente satisfacer de nuevo este predicado. Si es posible generar otra forma distinta con estos mismos datos, la cláusula tendrá éxito por segunda vez y esta segunda cadena aparecerá también en la pantalla.

Si la opción elegida fue la segunda (`Generar un nombre o adjetivo`) entraremos en el panel de Generación nominal. Lo mismo que para la generación de verbos, se crean las ventanas necesarias y se produce la llamada al predicado de generación nominal `genera_nombre`. Como veíamos anteriormente, el operador ';' (equivalente a una disyunción) hace que el programa pase a `error_generando` si se ha producido un fallo en la generación.

```
general(2) :-
    makewindow(42,48,15," Generación nominal ",0,0,25,80),
    makewindow(51,31,15," Mensajes ",15,12,8,54),
    cursor(2,1), write("     Escriba los datos necesarios "),
    genera_nombre ; error_generando_nom.
```

En `genera_nombre` el programa solicita al usuario que introduzca los siguientes datos y le va presentando unos predicados de ayuda para cada rasgo:

- **Unidad léxica**: la forma no marcada de la palabra. En general, el masculino singular en las palabras con cuatro formas y el singular en las palabras de género inherente.
- **Género**: *mas*, *fem* o *mas/fem*

- **Número**: *sg* o *pl*

Estos datos se ligan a las variables correspondientes y se produce la llamada al predicado de la gramática con las variables ya instanciadas. Por ejemplo, para generar el masculino plural de la palabra "investigador":

```
nominal(nivel_0,Cat,"investigador",mas,pl,FormaFlex)
```

La gramática retorna el valor de la variable FormaFlex ligado a la forma correspondiente ("investigadores" en nuestro caso). Esta forma se imprime en la pantalla dentro de una ventana apropiada. Prolog intentará satisfacer la cláusula tantas veces como sea posible. Por este motivo, se le pide al usuario que pulse una tecla para ver otra formas que se hayan podido generar con esos mismos datos.

```
genera_nombre:-

    gotowindow(42),
    field_str(3,18,16,"Unidad léxica  ="),
    field_str(5,18,16,"Género         ="),
    field_str(7,18,16,"Número         ="),
    cursor(11,18),
    write("FORMA GENERADA ="),
    makewindow(89,54,0,"",11,38,2,25),
    gotowindow(42),

    cursor(3,38), ayuda_ul, gotowindow(42), readln(UL),
    cursor(5,38), ayuda_gen, gotowindow(42), readln(Gen),
    cursor(7,38), ayuda_num, gotowindow(42), readln(Num), ',
    nominal(nivel_0,Cat,UL,Gen,Num,FormaFlex),
    gotowindow(89), cursor(1,1),
    write(FormaFlex),

    gotowindow(51), clearwindow, cursor(1,1),
    write("     Pulse cualquier tecla para ver    \n
               otra forma posible    \n
               generada con estos datos.       "),
    readchar(_),
    gotowindow(89), clearwindow.



/*-----------------------------------------------------------------
      G E N E R A R   T O D A S   L A S   F O R M A S
-----------------------------------------------------------------*/
```

La opción tercera Generar todas las formas posibles está concebida esencialmente como un sistema de comprobación. Permite generar de forma automática todas las formas posibles con las entradas de diccionario disponibles y dada la gramática que hemos visto. Como la operación consume unos segundos (tantos más cuanto mayor sea la longitud del diccionario), se imprime en la pantalla un mensaje para el usuario. Se abre un fichero al que se llama "formas.dic" y se desvía hacia él la corriente de escritura (que por defecto se dirige a la memoria RAM del ordenador). A

continuación se llama al predicado verbo con todas sus variables sin ligar, lo cual da lugar a que el predicado pueda ser satisfecho con todas las combinaciones posibles. Cada una de las palabras resultantes (las sucesivas instanciaciones de la variable Cadena) se escribe en una línea del fichero.

```
general(3) :-
    makewindow(43,48,15," Generación ",0,0,25,80),
    cursor(5,10), write("Se están generando todas las formas posibles"),
    cursor(6,10), write("con los datos disponibles en el diccionario."),
    cursor(8, 16), write("Espere unos segundos, por favor."),
    openwrite(todasformas,"formas.dic"),
    writedevice(todasformas),
    verbo(_,_,_,_,_,Cadena),
    not(Cadena = "(defectivo)"),
    write(Cadena,' '),nl,
    fail.
```

A continuación se efectúa la misma operación, pero llamando al predicado que construye formas nominales como nombres, adjetivos o pronombres. Los resultados se colocan también en el fichero "formas.dic"

```
general(3) :-
    nominal(nivel_0,_,_,_,_,FormaFlex),
    write(FormaFlex,' '),nl,
    fail.
```

La última llamada al predicado general(3) se ocupa de cerrar el fichero y mostrarlo en pantalla. Al mismo tiempo, se dan instrucciones al sistema operativo para que guarde el fichero de modo que además de poder consultarlo en pantalla, se guarde una copia en el disco. El fichero se almacenará en el mismo disco y directorio desde el que se esté ejecutando el programa.

```
general(3):-
   closefile(todasformas),
   system("formas.dic",0,_),
   file_str("formas.dic",FORMAS),
   display(FORMAS).


/*-------------------------------------------------------------------
    G E N E R A R   U N   P A R A D I G M A   C O M P L E T O
-------------------------------------------------------------------*/
```

La quinta opción da la oportunidad de generar el paradigma completo para una forma nominal. La única información que se le solicita al usuario es la unidad léxica. Se toma la información del teclado y se transmite la variable al predicado generalbis_nom que será el encargado de generar el paradigma y presentar los resultados.

```
general(5):-
  makewindow(45,48,15," Generar un paradigma nominal ",0,0,25,80),
  makewindow(55,31,15," Mensajes ",18,16,6,44),
  cursor(0,1), write("   Escriba la forma base:\n\n    ej.  'niño',
                  'camión', 'casa'   \n        'bonito', 'grande'"),
  gotowindow(45),
  cursor(2,25),
  write(" Unidad Léxica = "),readln(UL), nl,nl,
  generalbis_nom(UL).
```

Existen once cláusulas diferentes para el predicado generalbis_nom que consisten en llamadas a los predicados de la gramática con distintas combinaciones de variables. Por ejemplo, la cláusula que sigue es la que genera paradigmas con cuatro formas. Por ejemplo, si la unidad léxica elegida fuera **niño**, esta cláusula realizaría el siguiente ligado de variables

```
nominal(nivel_0,Cat,"niño",mas,sg,MasSg)
nominal(nivel_0,Cat,"niño",fem,sg,FemSg)
nominal(nivel_0,Cat,"niño",mas,pl,MasPl)
nominal(nivel_0,Cat,"niño",fem,pl,FemPl)
```

En este caso, la gramática sería capaz de generar una forma para cada uno de estos predicados. Las variables MasSg, FemSg, MasPl y FemPl retornarían ligadas a sus valores correspondientes: "niño," "niña," "niños" y "niñas." El predicado continuaría entonces con la parte que se encarga de mostrar los resultados en la pantalla de forma que además de ver las formas se proporciona información sobre el género y el número de cada una de ellas. Por último, el predicado continuar_generando_paradigm_nom vuelve a iniciar el proceso.

```
generalbis_nom(UL):-

  nominal(nivel_0,Cat,UL,mas,sg,MasSg),  <-- llamada a la gramática
  nominal(nivel_0,Cat,UL,fem,sg,FemSg),',
  nominal(nivel_0,Cat,UL,mas,pl,MasPl),',
  nominal(nivel_0,Cat,UL,fem,pl,FemPl),',
  cursor(5,15),

  write("              MASCULINO            FEMENINO "),nl,
  cursor(7,15),
  writef(" SINGULAR:    %-20 %-22",MasSg,FemSg),
  cursor(9,15),
  writef("   PLURAL:    %-20 %-22",MasPl,FemPl),

  continuar_generando_paradigm_nom.
```

Pero por supuesto, no todas las palabras pertenecen a un paradigma con cuatro formas por lo que es bastante posible que el predicado que acabamos de ver falle. En ese caso, se intentaría satisfacer la siguiente cláusula de generalbis_nom que genera un paradigma con dos formas, las del masculino singular y plural. Este sería el paradigma para: **Palabras de género inherente en masculino.** En realidad es posible generar también palabras femeninas si ponemos una variable en el lugar de la constante **mas.** Se ha

escogido hacerlo de esta forma por razones de presentación. Las restantes cláusulas generan distintos tipos de paradigmas por el mismo procedimiento.

```
generalbis_nom(UL):-

   nominal(nivel_0,Cat,UL,mas,sg,MasSg),',
   nominal(nivel_0,Cat,UL,mas,pl,MasPl),',
   cursor(5,15),

   write("              MASCULINO    "),nl,
   cursor(7,15),
   writef(" SINGULAR:    %-25 ",MasSg),
   cursor(9,15),
   writef("   PLURAL:    %-27",MasPl),

   continuar_generando_paradigm_nom.
```

Palabra de género inherente en femenino

```
generalbis_nom(UL):-

   nominal(nivel_0,Cat,UL,fem,sg,FemSg),',
   nominal(nivel_0,Cat,UL,fem,pl,FemPl),',
   cursor(5,15),

   write("              FEMENINO    "),nl,
   cursor(7,15),
   writef(" SINGULAR:    %-25 ",FemSg),
   cursor(9,15),
   writef("   PLURAL:    %-27 ",FemPl),

   continuar_generando_paradigm_nom.
```

Palabras con género indeterminado

```
generalbis_nom(UL):-

   nominal(nivel_0,Cat,UL,"mas/fem",sg,InhSg),',
   nominal(nivel_0,Cat,UL,"mas/fem",pl,InhPl),',
   cursor(7,15),
   writef(" SINGULAR:    %-25 ",InhSg),
   cursor(9,15),
   writef("   PLURAL:    %-27 ",InhPl),

   continuar_generando_paradigm_nom.
```

Palabras masculinas invariables en cuanto al número

```
generalbis_nom(UL):-

   nominal(nivel_0,Cat,UL,mas,"sg/pl",MasNoplu),',
   cursor(5,15),
   write("              MASCULINO    "),nl,
   cursor(7,15),
   writef(" SINGULAR:    %-25 ",MasNoplu),
   cursor(9,15),
   writef("   PLURAL:    %-27 ",MasNoplu),

   continuar_generando_paradigm_nom.
```

Palabras femeninas invariables en cuanto al número

```
generalbis_nom(UL):-

   nominal(nivel_0,Cat,UL,fem,"sg/pl",FemNoplu),',
   cursor(5,15),
   write("              FEMENINO    "),nl,
   cursor(7,15),
   writef(" SINGULAR:    %-25 ",FemNoplu),
   cursor(9,15),
   writef("   PLURAL:    %-27 ",FemNoplu),

   continuar_generando_paradigm_nom.
```

Singularia tantum en masculino

```
generalbis_nom(UL):-

   nominal(nivel_0,Cat,UL,mas,sg,Singularia),',
   cursor(6,15),
   write("              MASCULINO    "),nl,
   cursor(8,15),
   writef("   SINGULAR:    %-27 ",Singularia),

   continuar_generando_paradigm_nom.
```

Singularia tantum en femenino

```
generalbis_nom(UL):-

   nominal(nivel_0,Cat,UL,fem,sg,Singularia),',
   cursor(6,15),
   write("              FEMENINO    "),nl,
   cursor(8,15),
   writef("   SINGULARIA:    %-27 ",Singularia),

   continuar_generando_paradigm_nom.
```

Pluralia tantum en masculino

```
generalbis_nom(UL):-

   nominal(nivel_0,Cat,UL,mas,pl,MasNoSing),',
   cursor(6,15),
   write("              MASCULINO    "),nl,
   cursor(8,15),
   writef("   PLURAL:    %-27 ",MasNoSing),

   continuar_generando_paradigm_nom.
```

Pluralia tantum en femenino

```
generalbis_nom(UL):-

   nominal(nivel_0,Cat,UL,fem,pl,FemNoSing),',
   cursor(6,15),
   write("              FEMENINO    "),nl,
   cursor(8,15),
   writef("   PLURAL:    %-27 ",FemNoSing),

   continuar_generando_paradigm_nom.
```

Palabras indeterminadas en cuanto a género y número

```
generalbis_nom(UL):-

   nominal(nivel_0,Cat,UL,"mas/fem","sg/pl",Especial),',
   cursor(5,15),

   write("              MASCULINO          FEMENINO "),nl,
   cursor(7,15),
   writef(" SINGULAR:    %-20 %-22",Especial,Especial),
   cursor(9,15),
   writef("   PLURAL:    %-20 %-22",Especial,Especial),

   continuar_generando_paradigm_nom ; error_generando_paradigm_nom.
```

La opción 6 permite generar la conjugación completa de un verbo, tanto las formas simples como las compuestas. El sistema que se sigue es similar al que hemos visto para las formas nominales y, aunque las cláusulas son de una longitud considerable, es en realidad más sencillo porque para todos los verbos existen el mismo número de formas y, por tanto, sólo tenemos que componer un paradigma. Lo mismo que para los nombres, la única información que se le solicita al usuario es que introduzca la unidad léxica, en este caso el infinitivo. Por lo tanto, se trata de pedir a la gramática que encuentre la forma correspondiente a cada una de las personas de cada tiempo. Por ejemplo,

```
verbo(nivel_0,"llegar",sg_1,pres_ind,FormaV,P11)
```

nos retorna la primera persona del singular del presente de indicativo del verbo "llegar":

```
verbo(nivel_0,"llegar",sg_1,pres_ind,FormaV,"llego")
```

De esta forma se generan todas las personas de cada tiempo y después se imprimen en la pantalla de forma que queden agrupadas por tiempos y la denominación del tiempo - por ejemplo, "presente de indicativo" - quede situada sobre ellos. Por razones puramente técnicas, ya que Turbo Prolog tiene un número máximo de predicados por cláusula, ha sido necesario desdoblar esta cláusula en dos, de forma que `general(6)` termina con una llamada a `generalbis_verb` que sirve para continuar el proceso hasta que se han generado todas las formas simples. Como es habitual, el predicado que contiene los mensajes de error se encuentra al final de la cláusula precedido por una disyunción que permite que Prolog lo llame si ha fallado en los predicados anteriores.

```
general(6):-
   makewindow(46,48,15," Generar un paradigma verbal ",0,0,25,80),
   makewindow(56,31,15," Mensajes ",20,14,4,50),
   gotowindow(56),
   cursor(1,1), write("    Escriba el infinitivo "),
   gotowindow(46),
   cursor(1,25), write("  Unidad Léxica = "),readln(UL),nl,

   verbo(nivel_0,UL,sg_1,pres_ind,FormaV,P11),',
   verbo(nivel_0,UL,sg_2,pres_ind,FormaV,P12),',
   verbo(nivel_0,UL,sg_3,pres_ind,FormaV,P13),',
   verbo(nivel_0,UL,pl_1,pres_ind,FormaV,P14),',
   verbo(nivel_0,UL,pl_2,pres_ind,FormaV,P15),',
   verbo(nivel_0,UL,pl_3,pres_ind,FormaV,P16),',

   verbo(nivel_0,UL,sg_1,impf_ind,FormaV,P21),',
   verbo(nivel_0,UL,sg_2,impf_ind,FormaV,P22),',
   verbo(nivel_0,UL,sg_3,impf_ind,FormaV,P23),',
   verbo(nivel_0,UL,pl_1,impf_ind,FormaV,P24),',
   verbo(nivel_0,UL,pl_2,impf_ind,FormaV,P25),',
   verbo(nivel_0,UL,pl_3,impf_ind,FormaV,P26),',

   verbo(nivel_0,UL,sg_1,pret_ind,FormaV,P31),',
   verbo(nivel_0,UL,sg_2,pret_ind,FormaV,P32),',
   verbo(nivel_0,UL,sg_3,pret_ind,FormaV,P33),',
   verbo(nivel_0,UL,pl_1,pret_ind,FormaV,P34),',
   verbo(nivel_0,UL,pl_2,pret_ind,FormaV,P35),',
   verbo(nivel_0,UL,pl_3,pret_ind,FormaV,P36),',

   verbo(nivel_0,UL,sg_1,futuro,FormaV,P41),',
   verbo(nivel_0,UL,sg_2,futuro,FormaV,P42),',
   verbo(nivel_0,UL,sg_3,futuro,FormaV,P43),',
   verbo(nivel_0,UL,pl_1,futuro,FormaV,P44),',
   verbo(nivel_0,UL,pl_2,futuro,FormaV,P45),',
   verbo(nivel_0,UL,pl_3,futuro,FormaV,P46),',
   cursor(3,2),
   write("PRESENTE IND    IMPERFECTO IND    PRET. IND.      FUTURO"),nl,
   writef("  %-15 %-17 %-15 %-15",P11,P21,P31,P41),nl,
   writef("  %-15 %-17 %-15 %-15",P12,P22,P32,P42),nl,
   writef("  %-15 %-17 %-15 %-15",P13,P23,P33,P43),nl,
   writef("  %-15 %-17 %-15 %-15",P14,P24,P34,P44),nl,
   writef("  %-15 %-17 %-15 %-15",P15,P25,P35,P45),nl,
   writef("  %-15 %-17 %-15 %-15",P16,P26,P36,P46),nl,nl,
   general_bis_verb(UL) ;  error_generando_paradigm_verbo.
```

Como hemos visto general_bis_verb(UL) es esencialmente idéntico a general(6) y se encarga de generar las formas de subjuntivo que se le presentarán al usuario en la mitad inferior de la pantalla. También se encarga de llamar al predicado que genera las formas compuestas genera_form_comp(UL).

```
generalbis_verb(UL):-

    verbo(nivel_0,UL,sg_1,pres_subj,FormaV,P51),',
    verbo(nivel_0,UL,sg_2,pres_subj,FormaV,P52),',
    verbo(nivel_0,UL,sg_3,pres_subj,FormaV,P53),',
    verbo(nivel_0,UL,pl_1,pres_subj,FormaV,P54),',
    verbo(nivel_0,UL,pl_2,pres_subj,FormaV,P55),',
    verbo(nivel_0,UL,pl_3,pres_subj,FormaV,P56),',

    verbo(nivel_0,UL,sg_1,imp_subj,FormaV,P61),',
    verbo(nivel_0,UL,sg_2,imp_subj,FormaV,P62),',
    verbo(nivel_0,UL,sg_3,imp_subj,FormaV,P63),',
    verbo(nivel_0,UL,pl_1,imp_subj,FormaV,P64),',
    verbo(nivel_0,UL,pl_2,imp_subj,FormaV,P65),',
    verbo(nivel_0,UL,pl_3,imp_subj,FormaV,P66),',

    verbo(nivel_0,UL,sg_1,cond,FormaV,P71),',
    verbo(nivel_0,UL,sg_2,cond,FormaV,P72),',
    verbo(nivel_0,UL,sg_3,cond,FormaV,P73),',
    verbo(nivel_0,UL,pl_1,cond,FormaV,P74),',
    verbo(nivel_0,UL,pl_2,cond,FormaV,P75),',
    verbo(nivel_0,UL,pl_3,cond,FormaV,P76),',

    verbo(nivel_0,UL,sg_2,imper,FormaV,P82),',
    verbo(nivel_0,UL,pl_2,imper,FormaV,P85),',

    verbo(nivel_0,UL,no,infin,nofin,P00),',
    verbo(nivel_0,UL,no,ger,nofin,P90),',
    verbo(nivel_0,UL,no,part,nofin,P91),',

    write(" PRES. SUBJ.  IMPERF. SUBJ.    CONDICIONAL     IMPERATIVO
           IMPERSONALES"),nl,
    writef(" %-14 %-15 %-15 %-13 %-15",P51,P61,P71,P82,P00),nl,
    writef(" %-14 %-15 %-15 %-13 %-15",P52,P62,P72,P85,P90),nl,
    writef(" %-14 %-15 %-15              %-15",P53,P63,P73,P91),nl,
    writef(" %-14 %-15 %-15",P54,P64,P74),nl,
    writef(" %-14 %-15 %-15",P55,P65,P75),nl,
    writef(" %-14 %-15 %-15",P56,P66,P76),nl,nl,

    genera_form_comp(UL) ;  error_generando_paradigm_verbo.
```

Esta cláusula cumple la misma función que las dos anteriores con la única diferencia de que permite al usuario elegir entre ver la siguiente pantalla donde se le muestran las formas compuestas o retornar al panel de generación pulsando ESCape. Lo mismo que ocurría con general(6) ha sido necesario desdoblarlo en dos cláusulas distintas.

```
genera_form_comp(UL):-

   gotowindow(56), clearwindow, cursor(1,1),
   write("   Pulse cualquier tecla para ver \n   las formas compuestas.
                        ESC  para  salir"),

   readkey(K),
   K = esc, removewindow(46,1), removewindow(56,1),
   menu_genera, ' ;

   gotowindow(46), clearwindow,

   verbo(nivel_0,UL,sg_1,pret_perf,FormaV,P11),',
   verbo(nivel_0,UL,sg_2,pret_perf,FormaV,P12),',
   verbo(nivel_0,UL,sg_3,pret_perf,FormaV,P13),',
   verbo(nivel_0,UL,pl_1,pret_perf,FormaV,P14),',
   verbo(nivel_0,UL,pl_2,pret_perf,FormaV,P15),',
   verbo(nivel_0,UL,pl_3,pret_perf,FormaV,P16),',

   verbo(nivel_0,UL,sg_1,pret_plus,FormaV,P21),',
   verbo(nivel_0,UL,sg_2,pret_plus,FormaV,P22),',
   verbo(nivel_0,UL,sg_3,pret_plus,FormaV,P23),',
   verbo(nivel_0,UL,pl_1,pret_plus,FormaV,P24),',
   verbo(nivel_0,UL,pl_2,pret_plus,FormaV,P25),',
   verbo(nivel_0,UL,pl_3,pret_plus,FormaV,P26),',

   verbo(nivel_0,UL,sg_1,pret_ante,FormaV,P31),',
   verbo(nivel_0,UL,sg_2,pret_ante,FormaV,P32),',
   verbo(nivel_0,UL,sg_3,pret_ante,FormaV,P33),',
   verbo(nivel_0,UL,pl_1,pret_ante,FormaV,P34),',
   verbo(nivel_0,UL,pl_2,pret_ante,FormaV,P35),',
   verbo(nivel_0,UL,pl_3,pret_ante,FormaV,P36),',

   verbo(nivel_0,UL,sg_1,futur_perf,FormaV,P41),',
   verbo(nivel_0,UL,sg_2,futur_perf,FormaV,P42),',
   verbo(nivel_0,UL,sg_3,futur_perf,FormaV,P43),',
   verbo(nivel_0,UL,pl_1,futur_perf,FormaV,P44),',
   verbo(nivel_0,UL,pl_2,futur_perf,FormaV,P45),',
   verbo(nivel_0,UL,pl_3,futur_perf,FormaV,P46),',
   cursor(3,2),
   write("PRET. PERFECTO   PRET. PLUSCUAM.    PRET. ANTERIOR
                                              FUTURO PERF."),nl,
   writef("  %-16 %-18 %-20 %-18",P11,P21,P31,P41),nl,
   writef("  %-16 %-18 %-20 %-18",P12,P22,P32,P42),nl,
   writef("  %-16 %-18 %-20 %-18",P13,P23,P33,P43),nl,
   writef("  %-16 %-18 %-20 %-18",P14,P24,P34,P44),nl,
   writef("  %-16 %-18 %-20 %-18",P15,P25,P35,P45),nl,
   writef("  %-16 %-18 %-20 %-18",P16,P26,P36,P46),nl,nl,
   genera_form_compbis(UL) ;  error_generando_paradigm_verbo.
```

La siguiente cláusula, la última de las encargadas de la generación de paradigmas verbales se distingue de las demás porque incorpora el predicado que vuelve a iniciar el proceso, dando al usuario la oportunidad de probar un nuevo paradigma. Si se produjo el fallo en la generación de alguna de las formas tras haber probado todas las cláusulas, el camino alternativo permite que aparezca el mensaje de error para el usuario.

```
genera_form_compbis(UL):-

   verbo(nivel_0,UL,sg_1,pret_perf_subj,FormaV,P51),',
   verbo(nivel_0,UL,sg_2,pret_perf_subj,FormaV,P52),',
   verbo(nivel_0,UL,sg_3,pret_perf_subj,FormaV,P53),',
   verbo(nivel_0,UL,pl_1,pret_perf_subj,FormaV,P54),',
   verbo(nivel_0,UL,pl_2,pret_perf_subj,FormaV,P55),',
   verbo(nivel_0,UL,pl_3,pret_perf_subj,FormaV,P56),',

   verbo(nivel_0,UL,sg_1,pret_plus_subj,FormaV,P61),',
   verbo(nivel_0,UL,sg_2,pret_plus_subj,FormaV,P62),',
   verbo(nivel_0,UL,sg_3,pret_plus_subj,FormaV,P63),',
   verbo(nivel_0,UL,pl_1,pret_plus_subj,FormaV,P64),',
   verbo(nivel_0,UL,pl_2,pret_plus_subj,FormaV,P65),',
   verbo(nivel_0,UL,pl_3,pret_plus_subj,FormaV,P66),',

   verbo(nivel_0,UL,sg_1,cond_perf,FormaV,P71),',
   verbo(nivel_0,UL,sg_2,cond_perf,FormaV,P72),',
   verbo(nivel_0,UL,sg_3,cond_perf,FormaV,P73),',
   verbo(nivel_0,UL,pl_1,cond_perf,FormaV,P74),',
   verbo(nivel_0,UL,pl_2,cond_perf,FormaV,P75),',
   verbo(nivel_0,UL,pl_3,cond_perf,FormaV,P76),',


   write("  PRET. PERF. SUBJ.     PRET. PLUSC. SUBJ.    CONDICIONAL PERF.   "),nl,
   writef("  %-22  %-25 %-25",P51,P61,P71),nl,
   writef("  %-22  %-25 %-25",P52,P62,P72),nl,
   writef("  %-22  %-25 %-25",P53,P63,P73),nl,
   writef("  %-22  %-25 %-25",P54,P64,P74),nl,
   writef("  %-22  %-25 %-25",P55,P65,P75),nl,
   writef("  %-22  %-25 %-25",P56,P66,P76),nl,nl,
   continuar_generando_paradigm_verbo ; error_generando_paradigm_verbo.



/*-----------------------------------------------------------------
            M E N U   D E  A C T U A L I Z A C I O N
-----------------------------------------------------------------*/
```

La quinta opción es la opción de actualización que está constituida por una llamada al predicado updatdba. Esta opción está concebida para dar al usuario la posibilidad de añadir nuevas entradas el diccionario de una forma más cómoda que escribiéndolas directamente en el fichero de base de datos.

```
process(5):-  shiftwindow(1), updatdba.
```

El predicado updatdba carga el diccionario en memoria y construye el menú de actualización. En primer lugar, existe un grupo de opciones que dan la posibilidad de ver en la pantalla listas de las palabras que contiene el diccionario (verbos, formas lexicalizadas, nombres o adjetivos). Hay un segundo grupo de opciones que permite introducir nuevas entradas. Existe una opción distinta para cada categoría por la misma razón que ya vimos en el caso de la generación, esto es, porque los datos que el usuario tiene que suministrar son distintos para cada una de ellas.

```
updatdba:-
    makewindow(7,112,3," Actualizar el diccionario ",0,0,25,80),
    retractall(_),
    cursor(15,15),
    write(" >>  cargando el diccionario, espere unos segundos"),
    consult( "grampal.dic"),
    clearwindow,

    repeat,
    shiftwindow(7), clearwindow,
    menu(10,20,7,7,
     ["Mostrar verbos",
      "Mostrar formas verbales lexicalizadas",
      "Mostrar nombres",
      "Mostrar adjetivos",
      "",

      "Añadir una raíz verbal",
      "Añadir una forma verbal flexionada",
      "Añadir un nombre o adjetivo"],"Actualizar el diccionario",
      1,CHOICE),
    updatdba1(CHOICE),
    CHOICE=0,'.


/*-----------------------------------------------------------------
          V E R    E L    D I C C I O N A R I O
-----------------------------------------------------------------*/
```

Las opciones para ver el diccionario son muy sencillas. El procedimiento general es el mismo para todas las categorías. Se trata de hacer una llamada al predicado correspondiente del diccionario con todas las variables libres. Esto implica que la meta unificará con cualquier entrada del diccionario. De todos los valores que nos retorna el predicado, se toman los de :Cadena y UL y se imprimen en pantalla separados por ':'. Por ejemplo, el predicado para ver los verbos simplemente hace una llamada al predicado raiz_verb(CadenaR,_,UL,_,_,_) y las variables del predicado quedarán instanciadas con los valores que tenga la primera entrada de diccionario. Los valores de las variables UL y Cadena se imprimen en pantalla de forma que resulten parejas formadas por el infinitivo del verbo y cada uno de los alomorfos que existen para la raíz del verbo. Por ejemplo:

```
llegar : llegu
llegar : lleg
```

Prolog repetirá este predicado hasta que no queden más entradas de raiz_verb en el diccionario y, por lo tanto, falle al intentar instanciar las variables. Estas parejas se mantendrán en pantalla hasta que el usuario pulse una tecla. Entonces se borra la pantalla y se vuelve al menú de actualización.

```
updatdba1(1):-
     clearwindow,
     raiz_verb(CadenaR,_,UL,_,_,_),
     write(UL,':',CadenaR,"    ").

updatdba1(1):- nl,readchar(_),clearwindow.
```

De la misma forma se pueden visualizar otros tipos de entradas. Únicamente cambia el predicado del diccionario al que se llama. Por ejemplo, para ver las formas verbales lexicalizadas:

```
updatdba1(2):-
     clearwindow,
     verbo_lex(FormaFlex,_,UL,_,_,_),
     write(UL,':',FormaFlex,"    ").

updatdba1(2):- nl,readchar(_),clearwindow.
```

El predicado updatdba1(3) muestra en la pantalla los nombres con sus correspondientes alomorfos. Por ejemplo:

```
león : león
león : leon
```

```
updatdba1(3):-
     clearwindow,
     nom(FormaFlex,_,nombre,UL,_,_,_,_),
     write(UL,':',FormaFlex,"    ").

updatdba1(3):- nl,readchar(_),clearwindow.
```

Updatdba1(4) hace lo mismo que el anterior, pero con los adjetivos.

```
updatdba1(4):-
     clearwindow,
     nom(FormaFlex,_,adj,UL,_,_,_,_),
     write(UL,':',FormaFlex,"    ").

updatdba1(4):- nl,readchar(_),clearwindow.
```

```
/*-------------------------------------------------------------
      A C T U A L I Z A R     E L     D I C C I O N A R I O
-------------------------------------------------------------*/
```

El grupo de predicados de actualización contiene tres opciones: la primera de ellas permite introducir una raíz verbal (corresponde a la opción sexta del menú), la segunda es para formas verbales ya flexionadas (opción 7 del menú) y la tercera permite introducir nuevas raíces para nombres o adjetivos.

Si se elige la opción `Añadir una raíz verbal` se le pedirán al usuario los datos pertinentes que constituirán el valor de cada una de las variables que tenemos en la entrada

```
raiz_verb(Cadena,nivel_1,UL,conjugacion(Conjugacion1),
        tipo_raiz(Tipo_raiz1),tipo_des(Tipo_des1))
```

Los datos necesarios para completar la entrada de diccionario son:

- **Unidad léxica** : el infinitivo del verbo.
- **Conjugación** : ***cjg_1***, ***cjg_2*** o ***cjg_3***.
- **Tipo de raíz**: El usuario debe introducir, separados por blancos, los códigos numéricos de las personas a las que corresponde el alomorfo que quiere codificar (para averiguarlo lo mejor es usar una de las plantillas que se han descrito en la sección de morfología verbal). Por ejemplo, si se va a introducir una entrada para la raíz `pong` del verbo `poner` hay que introducir los datos:

  `tipo de raíz` = 11 51 52 53 54 55 56

- **tipo_des**: el rasgo que indica el tipo de desinencia que toma la raíz: desinencia regular o desinencia irregular de distintos tipos. Los valores que se pueden dar son: ***reg, pres, pret1, pret2, fut_cond, imp_subj, imper, infin, ger, part1, part2*** y ***defect.***
- **Cadena o alomorfo**: el alomorfo de la raíz al que corresponde la entrada. Por ejemplo, `pong`.

Los predicados `readln` son los encargados de captar esta información y ligarla a las variables correspondientes. Para los rasgos del tipo lista como **tipo_raiz** y **tipo_des** existe un pequeño mecanismo que los adapta a la forma definitiva que adoptan en el diccionario (introduce las comas entre los códigos, cambia el tipo de las variables de `string` a `integer` y convierte el resultado en una lista). A continuación, y antes de hacer la aserción del nuevo predicado en el diccionario, se le da al usuario la oportunidad de comprobar los datos y decidir si quiere que la entrada se añada como está o si, por el contrario, desea volver a escribirla. Esta es la misión del predicado `comprobacion(6)`. Si el usuario ha confirmado la entrada, el predicado `assertz` que permite añadir hechos a la base de datos durante la ejecución del programa, procederá a añadir la entrada en la base de datos, colocándola al final del grupo de predicados con el mismo nombre.

```
assertz(raiz_verb(Cadena,nivel_1,UL,conjugacion(Conjugacion1),
        tipo_raiz(Tipo_raiz1),tipo_des(Tipo_des1))),
```

Para que la aserción sea admitida, TODAS las variables tienen que estar ligadas (es decir, haber adoptado un valor), ya que Turbo Prolog no admite ninguna variable libre dentro de la base de datos. Dado que `assertz` se limita a introducir el nuevo hecho en memoria, es necesario salvar el fichero en el disco para conservar la nueva entrada, pues de otra forma se perdería en el momento en que vaciáramos la memoria. Durante el proceso de introducción de los datos se dispone de una ayuda que irá apareciendo en

el momento oportuno en la ventana de **Mensajes**. A estas ayudas corresponden las llamadas a los predicados ayuda_ul, ayuda_cjg que indican al usuario el tipo de datos que debe introducir y de los que hablaremos más adelante.

```
updatdbal(6):-
    clearwindow,
    makewindow(51,31,15," Mensajes ",15,12,8,54),
    cursor(2,1), write("     Escriba los datos necesarios "),
    gotowindow(7),
    field_str(3,18,20,"Unidad léxica      ="),
    field_str(5,18,20,"Conjugación        ="),
    field_str(7,18,20,"Tipo de raíz       ="),
    field_str(9,18,20,"Tipo de desinencia ="),
    field_str(11,18,20,"Cadena o alomorfo  ="),

    cursor(3,40), ayuda_ul, gotowindow(7),      readln(UL),
    cursor(5,40), ayuda_cjg, gotowindow(7),    readln(Conjugacion1),
    cursor(7,40), ayuda_tipo_raiz, gotowindow(7), readln(Cadenanum),
    cursor(9,40), ayuda_tipo_des, gotowindow(7),  readln(Cadenasimb),
    cursor(11,40), ayuda_cadena, gotowindow(7),   readln(Cadena),

    comprobacion(6),
    convierte(Cadenanum,Tipo_raiz1),
    convierte2(Cadenasimb,Tipo_des1),
    assertz(raiz_verb(Cadena,nivel_1,UL,conjugacion(Conjugacion1),
            tipo_raiz(Tipo_raiz1),tipo_des(Tipo_des1))),

    save("grampal.dic"),
    gotowindow(7), clearwindow,
    updatdbal(6).
```

La opción para introducir una forma lexicalizada sigue básicamente el mismo procedimiento. El predicado que vamos a introducir en la base de datos es diferente del de las raíces, porque la información que necesitamos es también distinta.

```
assertz(verbo_lex(Cadena,nivel_0,UL,Persona,Tiempo,FormaV))
```

En este caso se puede prescindir de toda la información relativa a la concatenación de la raíz y de la desinencia, es decir, no aparecen los rasgos **tipo_raiz, tipo_des** y **conjugacion** (por tanto, no es necesario hacer uso de los predicados de conversión). Dado que las formas lexicalizadas contienen ya toda la información flexiva, hay que introducir los datos referentes a **persona y número, tiempo y modo** y **forma verbal** (formas finitas o personales y formas no finitas o impersonales). El ligado de las variables y la introducción del predicado en la base de datos se hacen de la misma forma que ya vimos para las raíces verbales.

```
updatdbal(7):-
    clearwindow,
    makewindow(51,31,15," Mensajes ",15,12,8,54),
    cursor(2,1), write("     Escriba los datos necesarios "),
    gotowindow(7),
    field_str(3,18,20,"Unidad léxica      ="),
    field_str(5,18,20,"Persona y número   ="),
    field_str(7,18,20,"Tiempo y modo      ="),
    field_str(9,18,20,"Forma verbal       ="),
    field_str(11,18,20,"Cadena o alomorfo  ="),

    cursor(3,40), ayuda_ul, gotowindow(7),    readln(UL),
    cursor(5,40), ayuda_pers_num, gotowindow(7),   readln(Persona),
    cursor(7,40), ayuda_tiemp_mod, gotowindow(7), readln(Tiempo),
    cursor(9,40), ayuda_formav, gotowindow(7),  readln(FormaV),
    cursor(11,40), ayuda_cadena, gotowindow(7),  readln(Cadena),

    comprobacion(7),
    assertz(verbo_lex(Cadena,nivel_0,UL,Persona,Tiempo,FormaV)),
    save("grampal.dic"),
    gotowindow(7), clearwindow,
    updatdbal(7).
```

Para introducir nombres y adjetivos se usa el mismo predicado ya que las entradas que se van a añadir al diccionario son iguales con la excepción del valor del rasgo categoría y desde el punto de vista de la gramática las dos categorías se tratan de la misma forma:

```
nom(CadenaR,Nivel,Cat,UL,Gen,Num,tipo_gen(Tipo_gen),Tipo_plu))
```

El procedimiento en sí es el mismo que para las raíces verbales, es decir, se imprimen en la pantalla los nombres de los atributos que se le piden al usuario, se recogen los datos que el usuario introduce por el teclado, se hacen las conversiones de cadenas a listas y finalmente se hace la aserción del nuevo hecho en la base de datos. Por supuesto, los rasgos son bastante distintos a los del verbo, ya que hay que introducir la especificación de la **categoría** (***nombre*** o ***adj***), el **género** y el **número**.

```
updatdba1(8):-
    clearwindow,
    makewindow(51,31,15," Mensajes ",15,12,8,54),
    cursor(2,1), write("     Escriba los datos necesarios "),
    gotowindow(7),
    field_str(3,5,16,"Categoría      ="),
    field_str(5,5,16,"Nivel          ="),
    field_str(7,5,16,"Unidad léxica  ="),
    field_str(9,5,16,"Género         ="),

    cursor(3,23), ayuda_cat, gotowindow(7), readln(Cat),
    cursor(5,23), ayuda_nivel, gotowindow(7), readln(Nivel),
    cursor(7,23), ayuda_ul, gotowindow(7), readln(UL),
    cursor(9,23), ayuda_gen, gotowindow(7), readln(Gen),

    field_str(3,45,16,"Número         ="),
    field_str(5,45,16,"Tipo de género ="),
    field_str(7,45,16,"Tipo de plural ="),
    field_str(9,45,16,"Cadena         ="),
    cursor(3,63), ayuda_num, gotowindow(7), readln(Num),
    cursor(5,63), ayuda_tipo_gen, gotowindow(7), readln(Cadenasimb1),
    cursor(7,63), ayuda_tipo_num, gotowindow(7), readln(Cadenasimb2),
    cursor(9,63), ayuda_cadena, gotowindow(7), readln(CadenaR),

    comprobacion(8),
    convierte2(Cadenasimb1,Tipo_gen),
    convierte2(Cadenasimb2,Tipo_plu),
    assertz(nom(CadenaR,Nivel,Cat,UL,Gen,Num,tipo_gen(Tipo_gen),Tipo_plu)),
    save("grampal.dic"),
    gotowindow(7), clearwindow,
    updatdba1(8).


/*-----------------------------------------------------------------
        E D I T A R     E L     D I C C I O N A R I O
-----------------------------------------------------------------*/
```

La sexta opción del menú principal está pensada para que sea posible trabajar directamente sobre el fichero de base de datos que contiene el diccionario. Es muy útil en ocasiones en las que después de haber introducido una entrada averiguamos que no es correcta. Entonces podemos ver el diccionario y hacer sobre él las pequeñas modificaciones que sean necesarias. En cambio, para introducir entradas completamente nuevas es más conveniente usar la opción de **Actualizar**. El cuerpo de la cláusula process(6) está enteramente formado por predicados *built-in* que se encargan de convertir el fichero en una cadena de caracteres, editar esa cadena en la pantalla, comprobar si se ha producido alguna modificación en ella mediante la comparación de la cadena antigua y la nueva. En caso de que ambas cadena difieran, se le pregunta al usuario si desea guardar las modificaciones y se recoge su respuesta. Si la respuesta es afirmativa, se guarda el diccionario con las modificaciones, en caso contrario el predicado falla en ese punto, por lo que el nuevo fichero no se guarda y permanece el anterior. El entorno de edición es el que proporciona Turbo-Prolog.

```
process(6):-
    makewindow(9,112,3," Editar el diccionario ",0,0,25,80),
    file_str("grampal.dic",DB),
    edit(DB,DBNEW),
    clearwindow,
    not(DB = DBNEW),
    write("\n\n\n\n ¿ Desea guardar los cambios ? (s/n) "),
    readchar(Ans), str_char(Ans1,Ans),
    upper_lower(Ans1,Ans2), clearwindow ,
    Ans2 = "s",
    file_str("grampal.dic",DBNEW),
    retractall(_),
    removewindow(9,1) ; shiftwindow(1).


/*-----------------------------------------------------------------
            E D I T A R   U N   F I C H E R O
-----------------------------------------------------------------*/
```

La opción séptima es una opción que permite editar y ver un fichero cualquiera, pero no permite hacer modificaciones en él. El usuario deberá escribir el nombre del fichero, que puede contener una ruta de acceso. Como en el caso de editar el diccionario, el fichero se convierte en una cadena de caracteres, pero en lugar de llamar al entorno de edición se usa el predicado display(Fichero) que se limita a mostrarlo en la pantalla. Si por alguna razón no se encuentra el fichero solicitado, la primera cláusula falla y pasa a la segunda que sirve para producir el mensaje de error.

```
process(7):-
    shiftwindow(1),
    makewindow(20,32,3,"Editar el fichero: ",8,25,3,35),
    readln(Fichero),
    file_str(Fichero,FICH),
    removewindow(20,1),
    makewindow(11,112,3," Editar un fichero ",0,0,25,80),
    display(FICH),  removewindow(11,1), '.

process(7):- gotowindow(1),
             cursor(5,8), write(">> No se encontró ese fichero "),
             readchar(_),clearwindow.


/*-----------------------------------------------------------------
        I N T R O D U C C I Ó N   A   G R A M P A L
-----------------------------------------------------------------*/
```

La siguiente opción muestra al usuario un fichero de introducción al sistema que contiene información general acerca de GRAMPAL. El funcionamiento de la cláusula es esencialmente el mismo que para la opción anterior con la diferencia de que el nombre del fichero ("intro.hlp") está ya fijado.

```
process(9):-
    makewindow(99,112,3," Introducción a GRAMPAL ",0,0,25,80),
    file_str("intro.hlp",TXT),
    display(TXT),
    removewindow(99,1), shiftwindow(1), '.

process(9):- gotowindow(1),
             cursor(5,8), write(">> No se pudo leer intro.hlp\n"),
             readchar(_),clearwindow.


/*-------------------------------------------------------------
      S A L I D A   A L  S I S T E M A   O P E R A T I V O
-------------------------------------------------------------*/
```

La décima opción permite efectuar una salida momentánea al sistema operativo y retornar a GRAMPAL escribiendo simplemente EXIT. La cláusula, extremadamente sencilla, consta de una llamada al predicado system sin ningún argumento. Si GRAMPAL no es capaz de acceder al sistema operativo, avisa al usuario con un mensaje.

```
process(10):-
    system(""),'.

process(10):-
    write(">> command.com no está accesible"),
    readchar(_),
    removewindow.



/*-------------------------------------------------------------
             S A L I R   D E L   P R O G R A M A
-------------------------------------------------------------*/
```

La última opción detiene la ejecución, borrando previamente la ventana exterior (y todas las que haya dentro de ella) y nos devuelve al sistema operativo.

```
process(11) :- removewindow(1,1), exit.



/*-------------------------------------------------------------
             P R E D I C A D O S   A U X I L I A R E S
-------------------------------------------------------------*/
```

Dentro del programa hay otros predicados que podríamos llamar *auxiliares*, que cumplen funciones de menor importancia.

Uno de estos predicados es comprobacion que toma como único argumento un número entero que coincide con la opción del menú desde la que se efectúa la llamada

a este predicado. Es decir, si nos encontramos en la opción 6 *Editar el diccionario*, se llama al predicado como comprobacion(6) La finalidad de pasar este argumento es que si el usuario desea corregir, el programa pueda saber a cuál de los distintos paneles tiene que retornar. El predicado comprobacion pide al usuario que revise los datos y que pulse la letra 'g' si quiere guardar la entrada o los cambios que haya hecho, la letra 'm' si la quiere modificar, y la letra 's' si quiere salir sin guardar la entrada. La respuesta del usuario, junto con el número de la opción, se pasan como argumentos al predicado salida(Ans2,N).

```
comprobacion(N):-
    nl, gotowindow(51), clearwindow, cursor(1,1),
    write("     Compruebe los datos:\n\n
                para guardar esta entrada     'g' \n
                para modificar                'm' \n
                para salir de ACTUALIZAR      's' o ESC "),
    readchar(Ans), str_char(Ans1,Ans),
    upper_lower(Ans1,Ans2), clearwindow,
    salida(Ans2,N).
```

La primera cláusula para el predicado salida(Ans2,N), comprueba si la respuesta del usuario fue 'g', en cuyo caso el predicado simplemente tiene éxito, de forma que permite que se siga intentando satisfacer la cláusula desde la cual se le ha llamado. Si la cláusula era de actualización, continuará con la aserción del nuevo hecho en la base de datos; si era de edición, salvará en el disco los cambios que se hayan hecho.

```
salida(Ans2,N):-

    Ans2 = "g", '.
```

En la segunda cláusula se comprueba que la elección del usuario fue la letra 'm' (modificar). En ese caso, se llama al mismo predicado desde el que se partía para que el usuario tenga la oportunidad de corregir.

```
salida(Ans2,N):-

    Ans2 = "m", ',
    updatdbal(N).
```

La última cláusula borra las ventanas y vuelve al menú general, ya que la opción deseada era salir sin guardar la entrada.

```
salida(Ans2,N):-

    Ans2 = "s", ',
    removewindow(7,1), removewindow(51,1),
    grampal.
```

Si se pulsa una letra distinta de 'g', 'm' o 's', el predicado falla, lo que implica que no se guardarán los cambios y que volverá a intentarse la misma cláusula desde la que se le llamó.

```
salida(Ans2,N):- fail.
```

Los predicados que siguen son los que se encargan de avisar al usuario cuando se ha producido el fallo de una cláusula en algunos casos durante la generación en otros durante el análisis.

El primero de ellos error_generando_verbo se llama desde la cláusula general(1) cuando se produce el fallo en alguno de los predicados que la forman, gracias al operador de alternancia. El fallo de la cláusula puede producirse en el primer intento, lo que indicaría que el sistema no ha sido capaz de generar la forma requerida por el usuario, bien porque no exista esa unidad léxica en el diccionario bien porque alguno de los datos sea erróneo. También puede producirse el fallo después de uno o más éxitos lo que significa que se han agotado todas las posibilidades de generar formas con esos datos. El mensaje da cuenta de estas dos circunstancias distintas. Si el usuario pulsa la tecla ESC, vuelve al menú de generación. Si pulsa cualquier otra tecla puede volver a intentarlo en el panel de generar verbos, puesto que gracias al operador de alternancia, se llama al predicado general(1) cuando se produce el fallo en la parte anterior de la cláusula (ya que la tecla que se pulsó no era ESCape).

```
error_generando_verbo:-
      gotowindow(51), clearwindow, cursor(1,1),
      write(" No se pudo generar esa forma verbal por:\n
           * un error en los datos  y/o  el diccionario \n
           * o porque no hay más formas alternantes \n\n
           ESC  para salir  ;  cualquier tecla para continuar" ),
      readkey(K),
      K = esc, removewindow(41,1), removewindow(51,1),
      menu_genera, ' ;  general(1).
```

El predicado error_generando_nom es similar al de los verbos, pero con ligeras variaciones en el mensaje y, por supuesto, llama a general(2) en lugar de general(1).

```
error_generando_nom:-
      gotowindow(51), clearwindow, cursor(1,1),
      write(" No se pudo generar esa forma nominal por:\n
            * un error en los datos  y/o  el diccionario \n
            * o porque no hay más formas alternantes \n\n
              ESC  para salir  ;  cualquier tecla para continuar" ),

      readkey(K),
      K = esc, removewindow(42,1), removewindow(51,1),
      menu_genera, ' ;  general(2).
```

Los restantes predicados de error siguen el mismo esquema: primero situarse en la ventana de mensajes, después imprimir en la ventana el mensaje de error apropiado, comprobar si la tecla pulsada por el usuario ha sido ESCape (en cuyo caso irá a menu_genera) y en caso contrario volver al panel de la opción en la que se encontraba anteriormente. Los dos predicados que vemos a continuación son los que aparecen cuando hay uun fallo en la generación de paradigmas nominales o verbales.

```
error_generando_paradigm_nom:-
      gotowindow(55), clearwindow, cursor(1,1),
      write(" No se pudo generar ese paradigma nominal \n
             Pulse cualquier tecla para continuar o ESC para salir" ),
      readkey(K),
      K = esc, removewindow(45,1), removewindow(55,1),
      menu_genera, ' ;  general(5).

error_generando_paradigm_verbo:-
      gotowindow(56), clearwindow, cursor(1,1),
      write(" ERROR generando:  Pulse cualquier tecla para
             continuar  o ESC   para salir" ),
      readkey(K),
      K = esc, removewindow(46,1), removewindow(56,1),
      menu_genera, ' ;  general(6).
```

También existe unos predicados de continuidad cuyo único fin es dar la opción de volver a repetir ciertos procesos o abandonarlos para volver al menú de generación. Estos predicados se llaman desde generalbis_nom o desde generalbis_verb.

```
continuar_generando_paradigm_nom:-
     gotowindow(55), clearwindow, cursor(1,1),
     write("     Para probar otro paradigma nominal, \n
                 pulse cualquier tecla. \n
                 ESC   para salir"),
     readkey(K),
     K = esc, removewindow(45,1), removewindow(55,1),
     menu_genera, ' ;  general(5).

continuar_generando_paradigm_verbo:-
     gotowindow(56), clearwindow, cursor(1,1),
     write("     Para probar otro paradigma verbal, \n
                 pulse cualquier tecla.   ESC   para salir"),
     readkey(K),
     K = esc, removewindow(46,1), removewindow(56,1),
     menu_genera, ' ;  general(6).


/*-----------------------------------------------------------------
      A Y U D A   ( G E N E R A R   Y   A C T U A L I Z A R )
-----------------------------------------------------------------*/
```

En cualquier momento durante la ejecución, el usuario puede pulsar la tecla de función número 1 (PF 1) y aparecerá un fichero de ayuda que contiene información general sobre el programa.

```
help:-
    makewindow(99,112,3," Introducción a GRAMPAL ",0,0,25,80),
    file_str("intro.hlp",TXT),
    display(TXT),
    removewindow(99,1), '.
```

Hay otros predicados de ayuda específicos que van a apareciendo en la ventana de mensajes a medida que son necesarios, de acuerdo con las distintas opciones de generación o actualización que se hayan escogido y con los rasgos que sean pertinentes para cada caso. Los predicados son muy sencillos: simplemente se colocan en la ventana apropiada, la limpian y escriben en ella un mensaje. Estos predicados siempre tienen éxito.

Así el primero de estos predicados proporciona ayuda sobre el rasgo **nivel** informando al usuario de los distintos valores que puede adoptar y de cuándo se debe usar uno u otro. Los restantes predicados cumplen una función similar con cada uno de los rasgos. En primer lugar se encuentran los rasgos para las formas nominales y a continuación las ayudas para los rasgos de las entradas verbales.

```
ayuda_nivel:-
     gotowindow(51), clearwindow,
     cursor(1,1),
     write("   'nivel_0'  para palabras independientes \n
               'nivel_1'  para raíces  ").
```

Predicado de ayuda para el rasgo **unidad léxica**:

```
ayuda_ul:-
     gotowindow(51), clearwindow,
     cursor(0,1),
     write("      Escriba: \n\n
                  * el infinitivo para los verbos, \n
                  * la forma singular  ej. 'camión', 'casa',  \n
                  * la forma masculino singular ej. 'niño', 'bonito'  ").
```

Predicado de ayuda para el rasgo **unidad léxica**:

```
ayuda_cjg:-
     gotowindow(51), clearwindow,
     cursor(1,1),
     write("    'cjg1' --> verbos en -AR \n
                'cjg2' --> verbos en -ER \n
                'cjg3' --> verbos en -IR").
```

Predicado de ayuda para el rasgo **tipo_raíz**:

```
ayuda_tipo_raiz:-
    gotowindow(51), clearwindow,
    cursor(0,1),
    write("Escriba los códigos separados por un blanco: \n\n  100
         (verbos regulares), 11 (sg_1, pres_ind) \n  00 (infin) 90 (ger) 91
         (part) 82 (sg_2 imper) \n  2X (impf_ind) 3X (pret_ind) 4X
         (futuro) \n   5X (pres_subj)  6X (imp_subj) 7X (cond) ").
```

Predicado de ayuda para el rasgo **tipo_des**:

```
ayuda_tipo_des:-
    gotowindow(51), clearwindow,
    cursor(1,1),
    write("pret1 (pus-e,pus-o) ; pret2 (ca-í,cay-ó,cay-eron)\n  pres (ca-ímos);
          fut_cond (hab-ré, hab-ría) \n    imp_subj (dij-era)  ;
          imper (re-íd) \n       infin (re-ír)  ;
          ger (ri-endo)   \n  part1(hech-o) ;
          part2(ca-ído) ;  defect (soler)").
```

Predicado de ayuda para el rasgo **categoría**:

```
ayuda_cat:-
    gotowindow(51), clearwindow,
    cursor(2,1),
    write("          'nombre'   'adj'     ").
```

Predicado de ayuda para el rasgo **género**:

```
ayuda_gen:-
    gotowindow(51), clearwindow,
    cursor(1,1),
    write("    MASCULINO:  mas
               FEMENINO: fem   \n \n
               para palabras con AMBOS GENEROS:  \n
                     (p.ej. artista)        mas/fem ").
```

Predicado de ayuda para el rasgo **número**:

```
ayuda_num:-
    gotowindow(51), clearwindow,
    cursor(2,1),
    write("        SINGULAR: sg
                   PLURAL: pl  \n\n
                   palabras con la misma forma para sing. y plur.: \n
                         (p.ej. 'virus', 'crisis')    sg/pl ").
```

Predicado de ayuda para el rasgo **tipo_gen**:

```
ayuda_tipo_gen:-
    gotowindow(51), clearwindow,
    cursor(1,1),
    write("    masculino en 'o'  --> mas1 \n
               masculino en 'e'  --> mas2\n
               femenino en 'a'   --> fem  \n
               sin marca         --> inh   ").
```

Predicado de ayuda para el rasgo **tipo_plu**:

```
ayuda_tipo_num:-
    gotowindow(51), clearwindow,
    cursor(1,1),
    write("      plural en 's'  -->  plu1\n
                 plural en 'es' -->  plu2\n
                 sin marca      -->  noplu  ").
```

Predicado de ayuda para el rasgo **persona y número**:

```
ayuda_pers_num:-
    gotowindow(51), clearwindow,
    cursor(1,1),
    write("        sg_1   sg_2   sg_3  \n \n      pl_1   pl_2   pl_3  ").
```

Predicado de ayuda para el rasgo **tiempo y modo**:

```
ayuda_tiemp_mod:-
    gotowindow(51), clearwindow,
    cursor(1,1),
    write(" pres_ind  impf_ind  pret_ind  futuro \n  pres_subj
            imp_subj  cond  imper \n       infin   ger   part    ").
```

Predicado de ayuda para el rasgo **forma verbal**:

```
ayuda_formav:-
    gotowindow(51), clearwindow,
    cursor(1,1),
    write("  fin (FORMA FINITA) \n   nofin  (FORMA NO PERSONAL)  ").
```

Predicado de ayuda para la **cadena**:

```
ayuda_cadena:-
    gotowindow(51), clearwindow,
    cursor(1,1),
    write("  Escriba la variante alomórfica correspondiente:\n \n
             'dec', 'dij', 'dic'...  \n \n      'lápiz', 'lápic'   ").
```

9.0 El programa de segmentación

SEGMENTA.C

El segmentador de GRAMPAL (SEGMENTA.C) es un programa sencillo escrito en lenguaje C que compara la palabra que se quiere analizar con el diccionario y cuya finalidad es seleccionar sólo aquellas entradas de diccionario que sean pertinentes para el análisis de esa forma. Gracias a este segmentador se consigue que la gramática pueda trabajar con un diccionario muy pequeño, reduciendo muchísimo el tiempo de proceso.

Para la implementación de este segmentador se ha escogido C en lugar de PROLOG porque es un lenguaje más rápido y flexible en el manejo de cadenas y que, como se verá en detalle más adelante, permite realizar todo el proceso mediante una rutina *iterativa*, donde PROLOG hubiera tenido que utilizar un *procedimiento recursivo* (con el consiguiente consumo de tiempo y memoria).

SEGMENTA.C comienza a funcionar a partir de una llamada desde GRAMPAL. El proceso es el siguiente:

1. Una vez que el usuario ha escogido la opción *Analizar una palabra* en el menú principal, entra en el panel de análisis y el sistema le pide que escriba la palabra que desee analizar. GRAMPAL toma esta palabra y la envía al segmentador, y aquí es donde GRAMPAL transfiere el control a SEGMENTA.C. La llamada al programa, que se realiza desde el sistema operativo, tiene esta forma:

```
str_char(Comilla,'"'),
concat("segmenta ",Comilla,Llamada1),
concat(Llamada1,Cadena,Llamada2),
concat(Llamada2,Comilla,Llamada),
system(Llamada,0,_),
```

   La instrucción `str_char(Comilla,'"')` nos permite convertir un carácter en una cadena para así poder concatenar las comillas con la palabra de entrada, de forma que la llamada a SEGMENTA.C tenga su argumento entrecomillado. Con esto se consigue poder analizar formas en las que haya espacios en blanco (por ejemplo, las formas compuestas de lo verbos). SEGMENTA.C admite también una entrada sin entrecomillar, pero en ese caso toma únicamente la cadena hasta el primer blanco. Por lo tanto, suponiendo que la palabra que quisiéramos analizar fuera **ha cantado**, las tres instrucciones concat producirían el siguiente resultado en sustitución de la variable *Llamada*:

```
system(segmenta "ha cantado",0,_)
```

lo que produce la salida del programa a DOS y la llamada a SEGMENTA con *ha cantado* como argumento.

2. La definición de la función SEGMENTA toma dos argumentos. El primero de ellos es un número entero que sirve para controlar que en la llamada a la función se suministre el número correcto de argumentos. El valor del primer argumento es 2 (lo que implica que la función toma dos argumentos) y el segundo argumento es una cadena de caracteres:

```
main(int argc, char *argv[])
```

   El grupo de instrucciones

```
if(argc'=2)
  {
    exit(0);
  }
```

   tiene como misión comprobar que la cadena de caracteres esté presente en la entrada de la función. Si por alguna razón el número de argumentos no es igual a 2, la ejecución se detiene y se produce la salida al sistema. No es probable que esto ocurra porque GRAMPAL se encarga de verificar que efectivamente hayamos introducido una cadena de caracteres. Si el usuario, por error, pulsa el INTRO antes de haber escrito nada, el programa no continúa sino que vuelve a hacer la petición para que se escriba una palabra.

3. El siguiente paso consiste en copiar el segundo argumento (la cadena de caracteres de entrada) en una variable a la que llamaremos CAD (que es un *array* de 50 caracteres) y que será la variable que manipula el programa:

```
strcpy(cad,argv[1]);
```

4. A continuación el programa abre dos ficheros: el fichero GRAMPAL.DIC que contiene el diccionario general y un fichero al que llamamos SEGMENTA.DIC y que va a almacenar el resultado de la segmentación. GRAMPAL.DIC se abre en modo de lectura, pues lo que pretendemos es leer las entradas de diccionario contenidas en él, sin modificarlas. Por el contrario, SEGMENTA.DIC se abre en modo de escritura porque lo que queremos es copiar en él las entradas pertinentes resultado de la segmentación. La apertura del fichero en modo de escritura borra el contenido que tuviera el fichero anteriormente, lo que asegura que sólo se almacene el resultado de la segmentación que se va a hacer. En ambos casos, se realiza una comprobación para que si se produce algún error que impide la apertura de alguno de los ficheros el programa pare y retorne al sistema. A partir de este momento, la función dispone de dos corrientes, una de lectura representada por la variable *fp* y otra de escritura (variable *fp2*). Las variables son del tipo puntero de fichero.

```
if ((fp = (FILE *) fopen("grampal.dic","r")) == NULL)
        { puts("No se pudo abrir el fichero\n");
          exit(0);
         }
    if ((fp2 = (FILE *) fopen("segmenta.dic","w")) == NULL)
        { puts("No se pudo crear el archivo\n");
          exit(0);
         }
```

5. El siguiente grupo de instrucciones es un bucle, encabezado por una instrucción `while`, que se repite hasta que se termina de leer el diccionario. Estas instrucciones son las que verdaderamente realizan la segmentación.

```
while('feof(fp))

    {
      fgets(entr,280,fp);
      strcpy(cad2,entr);
      cad1 = (char *) strtok(cad2,"\"");
      cad1 = (char *) strtok(NULL,"\"");
      str  = (char *) strstr(cad,cad1);
      if (str '= NULL) fprintf(fp2,entr);
    }
```

`fgets(entr,280,fp)` comienza a leer el diccionario hasta que encuentra un carácter de fin de línea (o hasta que ha leído 280 caracteres, que es la longitud máxima de la línea en un fichero de base de datos en PROLOG). La variable ENTR (un *array* de 280 caracteres) recoge el valor de esta lectura de manera que en esta variable vamos almacenando una entrada de diccionario distinta en cada iteración. Por motivos que veremos más adelante, `strcpy(cad2,entr)` copia el contenido de ENTR en una nueva variable del mismo tipo que se llama CAD2. Suponiendo que la raíz del verbo **cantar** fuera la primera entrada que tenemos en el diccionario, el valor de la variable ENTR en la primera ejecución del bucle sería:

```
raiz_verb("cant","nivel_1","cantar",conjugacion("cjg2"),tipo_raiz([100]),tipo_des(["reg"]))
```

La siguiente cuestión es localizar dentro de esta entrada de diccionario la cadena que nos interesa para hacer la comparación. Para ello hemos dispuesto las instrucciones `cad1 = (char *) strtok(cad2,"\"");` y `cad1 = (char *) strtok(NULL,"\"");` que dividen la cadena de entrada en dos partes separadas por el carácter que se le da como segundo argumento. Con la primera instrucción, por tanto, queda almacenado en CAD1 lo siguiente:

```
CAD1 = raiz_verb(
```

mientras que la segunda instrucción separa la cadena que estamos buscando y guarda su valor en la variable CAD1 (no hay inconveniente en que sea la misma variable, ya que el valor que tuviera anteriormente no nos interesa):

```
CAD1 = "cant"
```

Llegado este punto tenemos almacenadas en sendas variables CAD y CAD1 las dos cadenas que queremos comparar. En nuestro ejemplo, el valor de CAD es "he cantado" y el de CAD1 es "cant". La siguiente instrucción:

```
str  = (char *) strstr(cad,cad1);
```

hace uso de la utilísima función de C `strstr`. Esta función toma como argumentos dos cadenas, las compara y comprueba si la segunda cadena es igual o está incluida en la primera. La ventaja de esta función es que no requiere que la coincidencia empiece al principio o al final de la cadena, sino que la reconoce en cualquier posición. Esto es especialmente útil para reconocer la existencia de morfemas en el interior de una palabra, por ejemplo, el morfema de masculino singular en la palabra *extranjer -o- s.* El valor de retorno de `strstr` es un puntero que será *nulo* si ambas cadenas no tienen nada en común y *no nulo* si la primera cadena contiene a la segunda (concretamente retorna un puntero a la posición donde ambas cadena terminan de coincidir). En el ejemplo que estábamos siguiendo, el valor contenido en el puntero STR es "t", el carácter donde la función ha terminado la comparación. Por lo tanto, un valor no nulo de STR nos indica que la cadena a la que se refiere la entrada de diccionario que estamos comparando es una posible segmentación de nuestra cadena de entrada, mientras que un valor nulo nos indica claramente que la entrada de diccionario en cuestión no tiene ninguna relación con el aducto y no va a contribuir a su análisis. Ahora sólo queda ir almacenando en el fichero SEGMENTA.DIC las entradas pertinentes, cosa que hacemos mediante la instrucción:

```
if (str != NULL) fprintf(fp2,entr);
```

que imprime el valor de la variable ENTR (en la que habíamos copiado la entrada de diccionario completa para tenerla disponible en este momento con la segunda instrucción del bucle) en la corriente FP2 que corresponde al fichero SEGMENTA.DIC si y sólo si el valor del puntero STR no es nulo. Si el puntero es nulo quiere decir que esa entrada de diccionario no nos interesa como segmentación posible y, por tanto, la instrucción `fprintf` no se ejecuta. El programa vuelve en este punto a lo alto del bucle y toma la siguiente entrada de diccionario, repite con ella todas las operaciones y se detiene sólo cuando encuentra el carácter que indica EOF (fin de fichero).

6. Una vez terminada la segmentación y concluido el bucle principal, solamente resta cerrar las dos corrientes de escritura y lectura que habíamos abierto en el transcurso del programa. Esto es un requisito de C, que no permite terminar la ejecución mientras queden ficheros abiertos. Las dos últimas instrucciones

```
fclose(fp);
fclose(fp2);
```

cumplen con este requisito y el programa llega a su fin. GRAMPAL recobra entonces el control.

7. Ya de vuelta en GRAMPAL, el predicado

```
consult("segmenta.dic")
```

carga el contenido del fichero SEGMENTA.DIC en la memoria RAM de forma que la gramática va a trabajar con un diccionario muy reducido, con un número de entradas que suele oscilar entre cinco y veinte.

Este es el listado completo del programa:

```
#include "stdio.h"
#include "string.h"
#include "conio.h"


main(int argc, char *argv[])
{
 FILE *fp;
 FILE *fp2;
 char cad[50];
 char entr[280];
 char cad2[280];
 char *cad1;
 char *str;

 if(argc'=2)
   {
    exit(0);
   }
 strcpy(cad,argv[1]);

 if ((fp = (FILE *) fopen("grampal.dic","r")) == NULL)
           { puts("No se pudo abrir el fichero\n");
             exit(0);
            }
      if ((fp2 = (FILE *) fopen("segmenta.dic","w")) == NULL)
           { puts("No se pudo crear el archivo\n");
             exit(0);
            }

   while('feof(fp))

        {
          fgets(entr,280,fp);
          strcpy(cad2,entr);
          cad1 = (char *) strtok(cad2,"\"");
          cad1 = (char *) strtok(NULL,"\"");
          str  = (char *) strstr(cad,cad1);
          if (str '= NULL) fprintf(fp2,entr);
        }


   fclose(fp);
   fclose(fp2);

  }
```

Bibliografía

Aguirre, E., Alegria, I., Arregui, X., Artola, X., Díaz, A. y Sarasola, K. (1989): "Aplicación de la morfología de dos niveles al euskera," en *Procesamiento del Lenguaje Natural*, 7, 87-103.

Aho, A.V. y Ullman, J.D. (1972): *The Theory of Parsing, Translation, and Compiling: Volume I, Parsing*, Englewood Cliffs, Prentice-Hall.

Aho, A.V., Sethi, R. y Ullman, J.D. (1990): *Compiladores: principios, técnicas y herramientas*, Wilmington, Addison-Wesley Iberoamericana.

Alcina, A. y Blecua, J.M. (1975;1980): *Gramática Española*, Barcelona, Ariel.

Allen, J. (1987): *Natural Language Understanding*, Menlo Park, Benjamin / Cummings.

Anderson, S.R. (1982): "Where's Morphology?," *Linguistic Inquiry*, 13, 571-612.

Aronoff, M. (1976): *Word Formation in Generative Grammar*, Cambridge, M.I.T. Press.

Bach, E. (1974;1976): *Syntactic Theory*, New York, Holt, Rinehart & Winston. Trad. española *Teoría sintáctica*, Barcelona, Anagrama, 1976.

Báez San José, V. (1988): *Fundamentos Críticos de la Gramática de Dependencias*, Madrid, Síntesis.

Barton, G., Berwick, R. y Ristad, E. (1987): *Computational complexity and natural language*, Cambridge, MIT. (Capítulo 5: "The complexity of the two-level morphology").

Bátori, I., Lenders W. y Putschke W. (eds.) (1989): *Computational Linguistics / Computerlinguistik*, Berlin, Walter de Gruyter.

Bear, J. (1986): "A Morphological Recogniser with Syntactic and Phonological Rules," en *COLING-86, 11th International Conference on Computational Linguistics*, Bonn, 272-276.

Bello, A. (1984): *Gramática de la Lengua Castellana*, Madrid, EDAF.

Benson, D. (1970): "Syntax and Semantics: A Categorial View," *Information and Control*, 17, 145-160.

Benson, D. (1979): "Formal languages vis-à-vis 'natural' languages," en Sedelow y Sedelow (1979), 97-164.

Berwick, R.C. (1984): "Strong Generative Capacity, Weak Generative Capacity and Modern Linguistic Theories," *Computational Linguistics*, 10, 3-4, ACL, 189-202.

Blåberg, O. (1985): "A Two-level Description of Swedish," en Karlsson (ed.) (1985), 43-62.

Black, A., Ritchie, G., Pulman, S. y Russell, G. (1987): "Formalisms for morphographemic description," en *Proceedings of the Third Conference of the European Chapter of the ACL*, Copenhagen, 11-18.

Boguraev, B. (1988): "A Natural Language Toolkit: Reconciling Theory with Practice," en Reyle y Rohrer(eds.) (1988), Dordrecht, D. Reidel, 95-130.

Bosque, I. y Pérez Fernández, M. (1987): *Diccionario Inverso de la lengua española*, Madrid, Gredos.

Calder, J. (1989): "Paradigmatic morphology," en *Proceedings of 4th Conf. of European Chapter of ACL*, Manchester, 58-65.

Casajuana, R. y Rodríguez Magro, C. (1986): "Clasificación de los verbos castellanos para un diccionario en ordenador," en *Actas del I Congreso de lenguajes naturales y lenguajes formales*, Barcelona, Universitat de Barcelona, 221-244.

Chomsky, N. (1957): *Syntactic Structures*, Le Hague, Mouton. Trad. esp. *Estructuras Sintácticas*, Madrid, Siglo XXI, 1974.

Chomsky, N. (1965): *Aspects of the Theory of Syntax*, Cambridge, M.I.T. Press. Trad. esp. *Aspectos de la Teoría de la Sintaxis*, Madrid, Aguilar, 1970.

Chomsky, N. (1970): "Remarks on Nominalization," en Jacobs y Rosembaum (eds.): *Readings in English Transformational Grammar*, Waltham, Ginn and Co., 184-221. Trad. esp. "Observaciones sobre la Nominalización," en Sánchez Zavala (comp.): *Semántica y Sintaxis en la Lingüística Transformatoria, I: Comienzos y Centro de la Polémica*, Madrid, Alianza, 1974, cap. 3.

Chomsky, N. (1981;1984): *Lectures on Government and Binding*, Dordrecht (Holland), Foris.

Chomsky, N. (1985): "Cambios de perspectiva sobre el conocimiento y uso del lenguaje," en *Teorema*, XV/1-2, 11-71.

Chomsky, N. (1986a): *Knowledge of Language. Its Nature, Origin and Use*, New York, Praeger Publishers.

Chomsky, N. (1986b): *Barriers*, Cambridge, MIT Press.

Chomsky, N. (1988): *Language and Problems of Knowledge. The Managua Lectures*, Cambridge, MIT.

Chomsky, N. y Halle, M. (1968): *The Sound Pattern of English*, New York, Harper and Row. Trad. esp. *Principios de Fonología Generativa*, Madrid, Fundamentos, 1979.

Church, K. (1982): *On memory limitations in natural language processing*, Bloomington, Indiana University Linguistics Club.

Church, K. y Kaplan, R. (1981): "Removing recursion from natural language processors based on phrase structure grammars," comunicación presentada en la conferencia sobre *Modeling Human Parsing Strategies*, 12 de Mayo, Austin, Texas (fotocopia).

Clocksin, W.F. y Mellish, C.S. (1981): *Programming in Prolog*, Berlin, Springer-Verlag.

Corbin, D. (1976): "Peut-on faire l'hypothèse d'une derivation en morphologie?," en Chevalier, J.Cl.: *Grammaire transformationelle. Syntaxe et lexique*, Villeneuve d'Ascq, Publications de l'Université de Lille, 49-91.

Crain, S. y Fodor, J.D. (1985): "How can grammars help parsers?," en Dowty et al. (1985), *Estudies in NLP*, Cambridge, Cambridge University, 94-128.

De Roeck, A. (1983): "An Underview of Parsing," en King (ed.) (1983), 3-18.

Delogu, C. (1989): "The Morphological Lexicon of a Speech Recognition System for Italian," *Rivista di Linguistica*, 1, 95-114,.

Domenig, M. (1990): "Lexeme-based Morphology: A Computationally Expensive Approach Intended for a Server-Architecture," en *COLING-90, Papers presented to the 13th International Conference on Computational Linguistics*, Helsinki, vol. 2, 77-82.

Dowty, D., Karttunen, L. y Zwicky, A. (1985): "Natural Language Parsing: Psychological, computational, and theoretical perspectives," en *Estudies in NLP*, Cambridge, Cambridge University.

Ejerhed, E. (1986): "A finite state parser for Swedish with morphological analyzer and semantics," en *Proceedings of SAIS-86*, University of Linköping.

Ejerhed, E. (1988): "Finding clauses in unrestricted text by finitary and stochastic methods," en *Proceedings of the Second Conference on Applied Natural Language Processing*, Austin (Texas), ACL.

Ejerhed, E. y Church, K. (1983): "A finite state parsing," en Karlsson, F (ed.), *Papers from the Seventh Scandinavian Conference of Linguistics*, Publication No. 9, Dept. of General Linguistics, University of Helsinki.

Emmorey, K. (1987): *Morphogical Structure and Parsing in the Lexicon. Ph.D, UCLA*, Los Angeles, Univ. Microfilms International.

Evans, R. (1985): "ProGram -a development tool for CPSG grammars," en Gazdar, G. (ed.), (1985): *Linguistics*, 23-2, Berlin, Mouton, 213-243.

Evans, R. y Gazdar, G. (1989): "Inference in DATR," *Proceedings of 4th Conf. of European Chapter of ACL*, Manchester, 66-71.

Everaert, M., Evers, A., Huybregts, R. y Trommelen, M. (eds.) (1985): *Morphology and Modularity* , Dordrecht, Foris.

Fodor, D. J. y Crain, S. (1985): "How can grammars help parsers?," en Dowty et al. (1985): *Estudies in NLP*, Cambridge, Cambridge University, 94-128.

Gazdar, G. (1982): "Phrase Structure Grammar," en Jacobson y Pullum (eds.) (1982), 131-186.

Gazdar, G. (1985): "Review article: finite state morphology," en *Linguistics*, 23, 597-607..

Gazdar, G. (ed.) (1985): "Computational Tools for Doing Linguistics," *Linguistics*, 23-2, Berlin, Mouton.

Gazdar, G. y Evans, R. (1989): "Inference in DATR," *Proceedings of 4th Conference of European Chapter of ACL*, Manchester, 66-71.

Gazdar, G. y Mellish, C. (1989): *Natural Language Processing in Prolog: An introduction to Computational Linguistics*, Wokingham (England), Addison-Wesley.

Gazdar, G. y Mellish, C. (1989): *Natural Language Processing in LISP: An introduction to Computational Linguistics*, Wokingham (England), Addison-Wesley.

Golding, A.R. y Thompson, H.S. (1985): "A morphology component for language programs," en Gazdar, G. (1985) (ed.): *Linguistics*, 23-2, Berlin, Mouton, 263-283.

Grishman, R. (1986): *Computational Linguistics*, Cambridge, Cambridge University Press.

Gross, M. (1972): *Mathematical models in linguistics*, Englewood Cliffs (USA), Prentice Hall. Trad. española *Modelos matemáticos en lingüística*, Madrid, Gredos, 1976.

Gross, M. y Lentin, A. (1970): *Introduction to formal grammars*, Berlin, Springer.

Halle, M. (1973): "Prolegomena to a Theory of Word Formation," *Linguistic Inquiry*, 4, 3-16.

Hallebeck, J. (1989): "Una gramática de afijo extendida para el español," en *Procesamiento del Lenguaje Natural*, 7, 147-157.

Hays, D.G. (1966): "Parsing," en Hays (ed.) (1966): *Readings in Automatic Language Proccesing*, New York, American Elsevier, 73-82.

Hockett, C.F. (1958): *A Course in Modern Linguistics*, Toronto, Macmillan. Trad. esp. *Curso de Lingüística Moderna*, Buenos Aires, Eudeba, 1971.

Hopcroft, J.E. y Ullman, J.D. (1979): *Introduction to Automata Theory, Languages and Computation*, Reading, Addison-Wesley.

Jackendoff, R. (1975): "Morphological and Semantic Regularities in the Lexicon," en *Language*, 51, 639-671. Versión esp. : "Regularidades morfológicas y Semánticas en el Lexicón" en Chomsky et al. (1979): *La Teoría Estándar Extendida*, Madrid, Cátedra, 73-116.

Jacobson, P. y Pullum, G. (eds.) (1982): *The Nature of Syntactic Representation*, Dordrecht, Reidel.

Jansana, R. I. y Quesada, J.D. (1985): "Nuevas tendencias en la teoría de modelos de lenguajes naturales," *Teorema*, XV/1-2, 185-211.

Jäppinen, H. y Ylilammi, M. (1986): "Associative Model of Morphological Analysis: An Empirical Inquiry. Linguistic Theories," *Computational Linguistics*, 12 4, 257-272.

Johnson, M. (1985): "Computer aids for comparative dictionaries," en Gazdar, G. (ed.) (1985): *Linguistics*, 23-2, Berlin, Mouton, 285-301.

Joshi, A. (1985): "Tree adjoining grammars: How much context-sensitivity is

required to provide reasonable structural description," Dowty et al. (1985), 206-250.

Kaminger, F.P. (1970): "The Noncomputability of the Channel Capacity of Context-Sensitive Languages," *Information and Control*, 17, 175-182.

Kaplan, R. y Kay, M. (1981): "Phonological rules and finite-state transducers," *Annual Meeting of the Linguistic Society of America in New York City*.

Karlsson, F. (ed.) (1985a): *Computational Morphosyntax*, Helsinki, Dept. of General Linguistics, Univ. of Helsinki.

Karlsson, F. (1985b): "Parsing Finnish in terms of Process Grammar," en Karlsson (ed.) (1985), 137-176.

Karlsson, F. (1989): "Computational Testing of Linguistic Models in Morphology," Bátori et al.: *Computational Linguistics*, Berlin, Walter de Gruyter, 245-251.

Kartunnen, L. y Kay, M. (1985): "Parsing in a free word order language," en Dowty et al.: *Natural Language Parsing*, Cambridge, Cambridge University, 279-306.

Kartunnen, L., Dowty, D. y Zwicky, A. (1985): *Natural Language Parsing: Psychological, computational, and theoretical perspectives*, Cambridge, Cambridge University.

Kay, M. (1985): "Parsing in functional unification grammar," en Dowty et al.: *Natural Language Parsing*, Cambridge, Cambridge University, 251-278.

Kay, M. (1987): "Nonconcatenative Finite-State Morphology," en *Proceedings of the Third Conference of the European Chapter of the ACL*, Copenhagen, 2-10.

Kay, M. y Karttunen, L. (1985): "Parsing in a free word order language," en Dowty et al.: *Natural Language Parsing*, Cambridge, Cambridge University, 279-306.

King, M. (ed.) (1983): *Parsing Natural Language*, London, Academic Press.

Klerer, M. (1987): *User-Oriented Computer Languages*, New York, Macmillan Publishing.

Koskenniemi, K. (1983): *Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production*, Helsinki, University of Helsinki.

Koskenniemi, K. (1983): "Two-level model for morphological analysis," en *Proceedings of IJCAI-83*, Karlsruhe.

Koskenniemi, K. (1985a): "A general two-level computationalmodel for word-form recognition and production," en Karlsson (1985), 1-18.

Koskenniemi, K. (1985b): "An application of the two-level model to Finnish," en Karlsson (1985), 19-37.

Lau, P. y Perschke, S. (1987): "Morphology in the Eurotra Base Level Concept," en *Proceedings of the Third Conference of the European Chapter of the ACL*, Copenhagen, 19-25.

Lees, R.B. (1960): *The Grammar of English Nominalizations*, Le Hague, Mouton. Cambridge, M.I.T..

Lenders, W., Putschke W. y Bátori, I. (eds.) (1989): *Computational Linguistics / Computerlinguistik*, Berlin, Walter de Gruyter.

Lieber, R. (1980): *On the Organization of the lexicon*, tesis doctoral inédita, Cambridge, M.I.T..

Lieber, R. (1988): "Configurational and Nonconfigurational Morphology," en Everaert et al (eds.) (1988).

Luscher y Schäpers (1981): *Gramática del alemán contemporáneo*, München, Max Hueber.

Lyons, J. (1979): *Introducción a la lingüística teórica*, Barcelona, Teide.

Manaster-Ramer,A. (1988): reseña de Savitch et al. (1987) en *Computational Linguistics*, 14, 4, 98-103.

Mañas, J.A. (1986): "Tratamiento previo de textos redactados en castellano," *Procesamiento del lenguaje natural*, 4, 1-21.

Marcos Marín, F. (1980;1985): *Curso de gramática española*, Madrid, Cincel.

Marcos Marín, F., Moreno, A. y Sánchez F. (1989): "El proyecto EUROTRA en el marco de la investigación sobre traducción por ordenador," *TELOS*, 16, 80-107, y *LEA*, XI, 165-178.

Marcos Marín, F. (1990): *Introducción a la Lingüística: Historia y modelos*, Madrid, Síntesis.

Martí, M.A. (1986a): "Un sistema d'anàlisi morfològica per ordinador," *Actas del I Congreso de lenguajes naturales y lenguajes formales*, Barcelona, Universitat de Barcelona, 351-365.

Martí, M.A. (1986b): "Un sistema de análisis morfológico por ordenador," *Procesamiento del lenguaje natural*, 4, 104-110.

Matthews, P.H. (1970): "Recent Developments in Morphology," en Lyons (ed.) (1970), 97-114. Trad. esp. "Evolución de la morfología en los últimos años" en Lyons (ed.) (1975): *Nuevos Horizontes de la Lingüística*, Madrid, Alianza, 1975, 99-117.

Matthews, P.H. (1974): *Morphology: an Introduction to the Theory of Word-Structure*, Cambridge, Cambridge University. Trad. esp. *Morfología: Introducción a la Teoría de la Estructura de la Palabra*, Madrid, Paraninfo, 1980.

Meya, M. (1986): "Análisis morfológico como ayuda a la recuperación de información," *Procesamiento del lenguaje natural*, 4, 91-103.

Miller, G. y Chomsky, N. (1963): "Finitary model of language users," en Luce, R., Bush, R. y Galanter, E. (eds.), *Handbook of mathematical psychology*, New York, John Wiley & Sons.

Moortgat, M. (1988): "Lambek Categorial Grammar and The Autonomy Thesis," en Everaert et al (eds.) (1988).

Moreno Cabrera, J.C. (1987): *Fundamentos de Sintaxis General*, Madrid, Síntesis.

Moreno Sandoval, A. (1989):"Flexión nominal: un problema de contexto " en *Procesamiento del lenguaje natural*, 7, 91-99.

Navarro Tomás, T. (1980): *Manual de pronunciación española*, Madrid, CSIC, 20ª ed.

Pereira, F. (1985): "A new characterization of attachment preferences," en Dowty et al. (1985), 307-319.

Pereira, F. y Shieber, S.M. (1987): *Prolog and Natural-Language Analysis*, Stanford, CSLI.

Pérez Fernández, M. y Bosque, I. (1987): *Diccionario Inverso de la Lengua Española*, Madrid, Gredos.

Perrault, C.R. (1984): "On the Mathematical Properties of Linguistic Theories," *Computational Linguistics*, 10, 3-4, 165-176.

Pesetsky, D. (1985): "Morphology and Logical Form," en *Linguistic Inquiry*, 16, 193-246.

Peters, P.S. y Ritchie, R.W. (1973): "Context-sensitive immediate constituent analysis: context-free languages revisited," en *Mathematical Systems Theory*, 6, 324-333.

Phillips, J.D. y Thompson, H.S. (1985): "GPSGP - a parser for generalized phrase structure grammars," en Gazdar, Gerald (ed.): *Linguistics*, 23-2, Berlin, Mouton, 245-261.

Putnam, H. (1985): "¿Es posible la semántica?," *Teorema*, XV/1-2, 131-145.

QUINCE mil verbos (1980): *Los quince mil verbos españoles: su gramática, clasificación y conjugación*, Barcelona, Ramón Sopena.

Ratti, D., Saba, A., Catarsi, M.N. y Cappelli, G. (198?) : *Analizador morfosintáctico de textos en lengua española* , Pisa, Giardini.

Real Academia Española (1973;1989): *Esbozo de una nueva gramática de la lengua española*, Madrid, Espasa Calpe.

Reyle, U. y C. Rohrer (eds.) (1988): *Natural Language Parsing and Linguistic Theories*, Dordrecht, D. Reidel.

Ritchie, G., Pulman, S., Black, A. y Russell, G. (1987): "A Computational Framework for Lexical Description," en *Computational Linguistics*, 13, 3-4, 290-307.

Ritchie, G. (1989): "On the generative power of two-level morphological rules," en *Proceedings of 4th Conference of European Chapter of ACL*, Manchester, 51-57.

Rodríguez Magro, C., Sopeña, L., Valladares, C, y Villar, C. (1990): "Clasificación morfológica del léxico castellano para un analizador en ordenador" en *Actas del VII Congreso Nacional de Lingüística Aplicada*, Sevilla, Universidad de Sevilla, 491-503.

Salomaa, A. (1969): *Theory of automata*, Oxford, Pergamon. (Capítulo IV: "Formal languages and generalized automata").

Sampson, G. (1983a): "Deterministic Parsing," en King (ed.) (1983), 91-116.

Sampson, G. (1983b): "Context-free Parsing and the Adequacy of Context-free Grammars," en King (ed) (1983).

Saussure, F. (1980): *Curso de Lingüística General*, Madrid, Akal, (1ª ed. 1922).

Savitch, W. et al. (eds.) (1987): *The Formal Complexity of Natural Language*, Dordrecht, D. Reidel.

Scalise, S. (1984): *Generative Morphology*, Dordrecht, Foris. Trad. esp. *Morfología Generativa*, Madrid, Alianza, 1987.

Schane, S. (1979): *Introducción a la Fonología Generativa*, Barcelona, Labor.

Schraeder, B. y Willée, G. (1989): "Computergestützte Verfahren morphologischer Beschreibung," en Bátori et al. (1989), 188-203.

Seco, M. (1986): *Diccionario de dudas y dificultades de la lengua española*, Madrid, Espasa-Calpe.

Sedelow, W. y Sedelow, S. (eds.) (1979): *Computers in Language Research: Formal Methods*, Le Hague, Mouton.

Selkirk, L. (1982): *The Syntax of Words*, Cambridge, M.I.T. Press.

Sells, P. (1987): *Lectures on Contemporary Syntactic Theories*, Stanford, S.C.L.I.

Siegel, D. (1982): *Topics in English Morphology*, tesis doctoral inédita, Cambridge, MIT.

Shieber, S. M. (1985): "Criteria for designing computer facilities for linguistic analysis," en Gazdar, G. (ed.): *Linguistics (Computational Tools for Doing Linguistics)*, 23-2, Berlin, Mouton, 198-211.

Shieber, S. M. (1986): *An introduction to unification- based approaches to grammar*, Stanford, CSLI. Traducción española: *Introducción a los formalismos gramaticales de unificación*, Barcelona, Teide, 1989.

Shieber, S. M. (1988): "Separating Linguistic Analyses From Linguistic Theories," en Reyle y Rohrer(eds)., 33-68. Traducido al español en *Introducción a los formalismos gramaticales de unificación.*

Shieber, S. M. y Pereira, F. (1987): *Prolog and Natural-Language Analysis*, Stanford, CSLI.

Schultink, H. (1988): "Some Remarks on the Relations between Morphology and Syntax in Twentieth-Century Linguistics," en Everaert et al (eds) (1988), 1-9.

Stockwell, R.P., Bowen, J.D. y Martin, J.W. (1965): *The grammatical structures of English and Spanish*, Chicago, The University Press.

Sopeña, L. (1982): *Grammar of Spanish for User Specialty Languages*, Heidelberg, IBM Wissentschaftliches Zentrum.

Sopeña, L. (1986): "Diccionarios del castellano en ordenador para la composición y verificación de textos," en *Procesamiento del lenguaje natural*, 4, Barcelona, SEPLN, 26-33.

Sproat, R. (1985): *On deriving the lexicon*, Ph.D. Diss., Cambridge, MIT.

Sproat, R. (1988): "Bracketing Paradoxes, Clitization and Other Topics: The Mapping between Syntactic and Phonological Structure," en Everaert et al (eds.) (1988).

Tennant, H. (1981): *Natural Language Processing*, Petrocelli, New York.

Thompson, H.S. y Golding, A.R. (1985): "A morphology component for language programs," en *Linguistics*, 23-2, 263-283.

Thompson, H.S. y Phillips, J.D. (1985): "GPSGP - a parser for generalized phrase structure grammars," en *Linguistics*, 23-2, 213-241.

Trost, H. (1990): "The application of two-level morphology to non-concatenative German morphology," en *COLING-90, Papers presented to the 13th International Conference on Computational Linguistics*, Helsinki, vol. 2, 371-376.

Turabian, K.L. (1987): *A Manual for Writers of Term Papers, Theses, and Dissertations*, Chicago, University of Chicago Press.

Tzoukermann, E. y Liberman, M. (1990): "A finite-state morphological procesor for Spanish," en *COLING-90, Papers presented to the 13th International Conference on Computational Linguistics*, Helsinki, vol. 3, 277-282.

Ullman, J.D. y Hopcroft, J.E. (1979): *Introduction to Automata Theory, Languages, and Computation*, Reading, Addison-Wesley.

Varela Ortega, S. (1990): *Fundamentos de Morfología*, Madrid, Síntesis.

Wasow, T. (1987): "Postscript," en Sells (1987), 193-205.

Wehrli, E. (1988): "Parsing with a GB-Grammar," en Reyle y Rohrer(eds) (1988), 177-201.

Whitelock, P. (1988): "A feature-based categorial morpho-syntax for japanese," en Reyle y Rohrer (eds) (1988).

Willem, G y Schraeder, B. (1989): "Computergestützte Verfahren morphologischer Beschreibung," en Bátori et al. (1989), 188-203.

Williams, E. (1981): "Argument Structure and Morphology," *The Linguistic Review*, 1, 81-114.

Winograd, T. (1983) : *Language as a Cognitive Process: Syntax*, Reading, Addison-Wesley.

Ylilamni, M. y Jappinen, H. (1986): "Associative Model of Morphological Analysis: An Empirical Inquiry," *Computational Linguistics*, 12, 4, 257-272.

Zoeppritz, M. (1984): *Syntax for German in the User Specialty Languages System*, Tübingen, Max Niemeyer.

Zwicky, A. y Karttunen, L. (1985): "Introduction," en Dowty et al. (1985), 1-25.

Zwicky, A., Dowty, D. y Karttunen, L. (eds.) (1985): *Natural Language Parsing: Psychological, computational, and theoretical perspectives*, Cambridge, Cambridge University.

REUNIDO, EN EL D'A DE LA FECHA, EL TRIBUNAL QUE SUSCRIBE, ACORDO CONCEDER
A LA PRESENTE TESIS DOCTORAL LA CALIFICACION DE Apto "cum laude"
MADRID, 13-Diciembre de 1991

EL PRESIDENTE,

FDO.: JOSE HIERRO

EL SECRETARIO,

FDO.: JUAN CARLOS MORENO

PRIMER VOCAL,

FDO.: PILAR RODRIGUEZ

SEGUNDO VOCAL,

FDO.: LUIS ROPERTA

TERCER VOCAL,

FDO.: JOAQUIN GARRIDO

FFC-TC-110

Signatura ....

UAM
UNIVERSIDAD AUTONOMA
DE MADRID

BIBLIOTECA

AUTOR MORENO SANDOVAL, A.

Título Un modelo computacional basado en la unificación para el análisis y generación de la morfología

Pie editorial ....

núm. de vols. 1

Prestado el día 22-4-93 *Devuelto el día* 28-5-93

A D. Univ. Carlos III de Madrid. Biblioteca MARIA MOLINER. Servicio de Acceso al Documento.

núm. de carnet ....

Dirección y Teléfono C/ Madrid, 126. 28903 GETAFE (MADRID) Telef. 624 97 23

Prestado el día 20-5-94 *Devuelto el día* 4 julio 94

A D. UNIVERSIDAD DEL PAIS VASCO. FACULTAD DE DERECHO SERVICIO DE DOCUMENTACION

núm. de carnet ....

Dirección y Teléfono Apartado 1223; 20080 SAN SEBASTIAN

Prestado el día .... *Devuelto el día* ....

A D. ....

núm. de carnet ....

Dirección y Teléfono ....

Prestado el día .... *Devuelto el día* ....

A D. ....

núm. de carnet ....

Dirección y Teléfono ....

Prestado el día .... *Devuelto el día* ....

A D. ....

núm. de carnet ....

Dirección y Teléfono ....

Prestado el día .... *Devuelto el día* ....
A D. ....
....
núm. de carnet ....
Dirección y Teléfono ....
....

Prestado el día .... *Devuelto el día* ....
A D. ....
....
núm. de carnet ....
Dirección y Teléfono ....
....

Prestado el día .... *Devuelto el día* ....
A D. ....
....
núm. de carnet ....
Dirección y Teléfono ....
....

Prestado el día .... *Devuelto el día* ....
A D. ....
....
núm. de carnet ....
Dirección y Teléfono ....
....

Prestado el día .... *Devuelto el día* ....
A D. ....
....
núm. de carnet ....
Dirección y Teléfono ....
....

Prestado el día .... *Devuelto el día* ....
A D. ....
....
núm. de carnet ....
Dirección y Teléfono ....
....

Prestado el día .... *Devuelto el día* ....
A D. ....
....
núm. de carnet ....
Dirección y Teléfono ....
....

UAM
UNIVERSIDAD AUTONOMA
DE MADRID

Servicio de Acceso al Documento
Edificio Rectorado, entreplanta 3.ª
CANTOBLANCO
28049 MADRID

Consulta obligatoria en Sala de Lectura de la Biblioteca solicitante

Código de solicitud: 943075
Código de suministro: 5572

Se autoriza su reproducción:

- ☐ Total
- ☐ Parcial
- ~~☒~~ Ninguna

Fecha de devolución: 20-6-94

GRAMPAL
Programa de demostración

Antonio Moreno Sandoval

Guía del Usuario

Contenido

1.0 Introducción

GRAMPAL es un analizador y generador morfológico del español basado en la unificación de rasgos. Se compone fundamentalmente de tres partes:

- **ANALIZADOR** de palabras,
- **GENERADOR** de palabras,
- herramientas para editar y actualizar el DICCIONARIO.

Además, desde el MENU PRINCIPAL se puede hacer uso de otras utilidades secundarias, como salir momentáneamente al sistema operativo o consultar una breve INTRODUCCION al sistema.

Dispone también de ayudas *on-line* dentro de las opciones principales, que guían al usuario en todo momento. Pulsando la tecla de escape (ESC) se vuelve al MENÚ PRINCIPAL.

2.0 Cómo empezar

2.1 Requisitos de la configuración

GRAMPAL está pensado para ordenadores personales del tipo AT (procesador 286) o superiores. Por supuesto, requiere el sistema operativo MS-DOS y es aconsejable que la memoria RAM disponible sea de al menos 512 K (aunque puede correr desde 370 K). Puede funcionar desde disqueteras de 5 1/4 y 3 1/2, pero es aconsejable instalarlo en un disco duro. En su versión actual ocupa menos de 300 K, pero si se quiere ampliar el diccionario hay que pensar en dejar espacio en el disco.

La versión que presentamos sirve tanto para monitores monocromo como para monitores de color. Los mejores resultados se obtienen utilizando una tarjeta gráfica VGA.

El programa reconoce y permite los caracteres españoles especiales como los acentos y la *ñ*, pero no es sensible a las diferencias entre mayúsculas y minúsculas. Es decir, cualquier cadena se tratará como si fuera una *constante* de Prolog (en minúsculas, por tanto). GRAMPAL no hace uso de otros dispositivos periféricos como la impresora o el ratón.

2.2 Cómo instalar GRAMPAL

Los ficheros del programa se presentan en disquetes de 5 1/4 y 3 1/2, de doble densidad. Aunque el programa se puede correr directamente desde estos disquetes (tecleando simplemente `grampal` una vez situado en la unidad de disco donde se halle GRAMPAL.EXE) recomendamos que se utilice una unidad de disco duro. La razón principal es que el programa consulta y crea habitualmente ficheros durante su ejecución. Evidentemente, este proceso se realiza más eficientemente desde un disco duro.

Para facilitarle la instalación del programa en el disco duro, incluimos un fichero .BAT que copia todos los ficheros necesarios en un directorio GRAMPAL que crea en el disco C. Téngase en cuenta que si utiliza este programa de instalación, el disco original tiene que estar en la unidad A y el programa será instalado en la unidad C, en un directorio que "cuelga" directamente del directorio raíz de la unidad C.

Para instalar GRAMPAL, sitúese en la unidad A y, con el disquete de GRAMPAL en dicha unidad, teclee

```
instala
```

Aparecerán en pantalla los ficheros que se están copiando en el disco duro. Terminada la operación, puede teclear `grampal` y seguir las instrucciones del próximo capítulo.

3.0 Menú principal: cómo navegar por GRAMPAL

Una vez pasada la pantalla de presentación (para ello, pulsar cualquier tecla), aparece el MENÚ PRINCIPAL. Desde aquí se puede ir a todas las utilidades del programa y es requisito imprescindible volver a él para pasar de una opción a otra.

Como norma general, para escoger una opción de cualquier menú de GRAMPAL hay que colocarse en la opción elegida y pulsar ENTER (o INTRO). Para desplazarse, se utilizan las teclas del cursor hacia arriba (↑) o hacia abajo (↓). Todos los menús tienen una opción por defecto (que en los monitores en color aparece resaltada). En el MENÚ PRINCIPAL es "Introducción a GRAMPAL." Pulse ENTER para ver cómo funciona (cuando quiera salir, pulse ESC).

Desde todos los menús, Ud. puede acceder a una pequeña ayuda, pulsando cualquiera de estas teclas de función:

- F1: "Ayuda general"
- F2: "Ayuda del proceso de Análisis"
- F3: "Ayuda del proceso de Generación"
- F4: "Ayuda del Diccionario"

Compruebe su efecto (para salir de la ayuda, pulse ESC).

Las tres opciones principales de GRAMPAL ("Analizar," "Generar" y "el Diccionario") se explicarán en capítulos especiales. Describiremos a continuación las otras opciones:

- **Salir a DOS**: si en cualquier momento de la ejecución Ud. desea salir al sistema operativo, escoja esta opción. Podrá trabajar de la forma acostumbrada (copiar, renombrar, borrar ficheros...) y cuando quiera volver a GRAMPAL teclee

```
exit
```

- **Editar un fichero**: esta opción está pensada sobre todo para ver el fichero SEGMENTA.DIC. Este fichero se crea en la opción de "Análisis" por el programa segmentador SEGMENTA.EXE. Es interesante editarlo para comprobar las segmentaciones que hace de las palabras. Este es el fichero con el que en realidad trabaja la Gramática cada vez que se le pide un análisis. Por otra parte, hay que hacer notar que esta opción sólo permite "visualizar" un fichero, pero no modificarlo (en realidad, no es una opción de edición propiamente dicha). Para volver al MENÚ PRINCIPAL, pulsar ESC.
- **Salir de** GRAMPAL: esta opción termina la ejecución del programa y vuelve al sistema operativo. Se puede conseguir el mismo efecto pulsando ESC desde el menú principal. En este caso el programa pedirá confirmación al usuario.

3.1 Analizar una palabra

Esta opción le permite preguntar al sistema sobre una palabra y éste devolverá todos los análisis posibles que existan de dicha palabra, teniendo en cuenta los datos disponibles en el diccionario. Si el sistema no es capaz de proporcionar ninguno (bien sea porque la palabra ha sido escrita incorrectamente, bien porque no existe en el diccionario, o porque su entrada estuviera mal codificada) le devolverá un mensaje dándole cuenta del error.

Tenga en cuenta que sólo puede escribir una palabra cada vez (a no ser que sea un forma compuesta de un verbo) y no debe dejar blancos ni delante ni detrás de la palabra. Las características del programa, por el momento, no permiten errores de formato en los datos.

La información morfosintáctica de la palabra consultada se mostrará en forma de rasgos. Igualmente, se incluye el *objeto Prolog* generado por la Gramática, es decir, la estructura de rasgos resultado de la aplicación de las reglas de la Gramática.

El proceso de análisis consiste en los siguientes pasos:

1. el usuario escribe la forma flexionada que quiere que sea analizada. Pulsa ENTER.
2. el sistema devuelve una respuesta, según los datos disponibles en las entradas de diccionario y la gramática. Puede haber dos respuestas:

- hay al menos un análisis de la estructura interna de la palabra. En este caso el sistema, después de proporcionarle la información, le pide que pulse cualquier tecla para mostrar otro/s análisis posibles. Al final, le comunica que "no se encontraron más análisis" para esa palabra. Para continuar, el usuario tiene que presionar cualquier tecla y aparece de nuevo el mensaje solicitando una palabra para analizar. El proceso continúa de la misma manera hasta que el usuario decide salir pulsando ESC.
- no se encontraron análisis para esa palabra. Compruebe que la palabra estaba bien escrita y que no dejó blancos (excepto en las formas de los verbos). Si no es un error gráfico, entonces se debe a algún problema con el diccionario: compruebe que existe la entrada apropiada para dicha palabra, y si existe, busque el error de codificación (por ejemplo, un valor cambiado u omitido). Después de hacer la corrección pertinente, vuelva al panel de "Análisis" y compruebe su arreglo. (Para realizar todas las comprobaciones mencionadas, Ud. ha tenido que salir previamente del panel de "Análisis" y escoger la opción "Editar el diccionario"; guardar los cambios realizados y entrar de nuevo en el panel de "Análisis").

Le sugerimos, para que vaya familiarizándose con el programa, que pruebe las siguientes palabras:

- *niños* (un único análisis),
- *asdfasdf* (ningún análisis),
- *hubiera* (dos análisis)
- *hubiera amado* (dos análisis)
- *programas* (dos análisis, uno como verbo y otro como nombre)
- *maravedíes, maravedís, maravedises* (un análisis cada uno)

3.2 Generar una palabra o un paradigma

Al elegir la opción de "Generación," el sistema carga el diccionario en la memoria RAM y nos presenta un "Menú de Generación." Como se explicó en 3.0, Menú principal: cómo navegar por GRAMPAL, el usuario debe escoger la opción deseada con las teclas del cursor y pulsar ENTER. También está disponible una pequeña ayuda *on-line* pulsando la tecla de función F1 (la ayuda sólo se puede solicitar cuando aparece en la pantalla el "Menú de Generación." Una vez dentro de de las opciones de "Generación," las instrucciones y ayudas aparecen en la ventana de "Mensajes").

El menú ofrece 5 opciones, aunque en conjunto se resumen en tres posibilidades:

1. **Generar una palabra**: ya sea una forma verbal (opción 1), o bien una forma nominal (opción 2).
2. **Generar un paradigma o conjugación completa**: pueden ser igualmente todas las formas flexionadas o bien de un verbo (opción 5) o de un nominal (opción 4).
3. **Generar todas las formas posibles**, con los datos del diccionario (opción 3).

3.2.1 Generar un verbo o un nominal

Elija la opción 1 ó 2 del menú y rellene los datos que se le van pidiendo desde las ventana de "Mensajes." En todo momento se muestran los valores posibles que pueden asignarse al rasgo pertinente. Recuerde que no debe dejar blancos delante o detrás de los valores que escriba ni los encierre entre comillas u otros signos. Tampoco puede escribir varios valores para un mismo rasgo. Cuando escriba el valor del rasgo, pulse ENTER para pasar al siguiente rasgo. Si después de pulsar ENTER se da cuenta de que ha escrito algo incorrecto, no podrá volver atrás usando el cursor. Para corregir, continúe pulsando ENTER para los demás datos, lo que generará un mensaje de error y le permitirá introducir los datos de nuevo.

Al igual que en el panel de "Análisis," si el sistema puede generar una forma con los datos que Ud. le ha dado se la mostrará (resaltada en rojo en los monitores de color), al tiempo que le pedirá

que pulse cualquier tecla para ver otras formas posibles. Cuando acabe de generar todas las formas alternantes (no más de tres) pulse de nuevo cualquier tecla para limpiar la pantalla e introducir nuevos datos.

Si la gramática no ha podido generar ninguna forma se lo indicará en la ventana de "Mensajes." Las causas pueden ser o bien un error en los datos suministrados (no existe esa forma, se escribió mal alguno de los valores...) o bien un error en el diccionario (la entrada no existe o tiene mal asignados los rasgos). Como en el proceso de "Análisis," compruebe primero los datos y si no encuentra fallo en ellos, edite el diccionario.

Para conocer la expresividad del sistema, le recomendamos que pruebe formas que presentan alternancias, como los plurales de *jabalí* o *maravedí*, o cualquier persona del imperfecto del subjuntivo. Comprobará así que la gramática puede generar todas las formas posibles.

3.2.2 Generar un paradigma nominal o verbal

Esta opción es muy sencilla de utilizar. Simplemente hay que escribir la **unidad léxica** y el programa mostrará en pantalla todo el paradigma completo. La conjugación completa de todas las formas verbales se ha divido en dos partes por motivos de claridad: en la primera se muestran las formas simples y, pulsando cualquier tecla, aparecen las formas compuestas. Téngase en cuenta que la generación de estas últimas conlleva, por lo general, el doble de tiempo de procesamiento que la generación de las formas simples (hay que concatenar y unificar 4 subcadenas, en lugar de 2 en los tiempos simples).

Como característica destacada de estas opciones señalaremos que sólo se muestra una forma en aquellos casos donde pueden aparecer dos variantes en alternancia (por ejemplo, en el imperfecto de subjuntivo). Esto se debe no a una imposibilidad de la gramática sino a una complicación en la presentación de los resultados. Para comprobar cada una de las variantes existe la opción individual expuesta en la sección anterior.

Con respecto a los errores en la generación del paradigma, compruebe primero que existe/n entrada/s para la **unidad léxica** propuesta. Si no es ésa la causa, hay dos indicios para encontrar el problema en la codificación del diccionario:

1. si el sistema nos "saca" al menú de Generación es que se ha omitido alguno de los códigos numéricos de los verbos (es decir, no se puede generar alguna de las formas de la conjugación y por eso falla toda la generación).
2. si el sistema nos devuelve el mensaje de que "no se pudo generar el paradigma," seguramente se debe a algún error en los datos de las entradas del diccionario.

Posiblemente esta opción sea la más representativa de las características generales del sistema pues se puede apreciar la versatilidad y expresividad que se puede conseguir con tan reducido número de reglas. Para los paradigmas verbales recomendamos que se empiece por los más regulares (*amar*) y se contraste el tiempo de respuesta con los más irregulares (*ir*). En la opción de los paradigmas nominales es interesante sobre todo preguntar por unidades léxicas que presentan formas flexionadas dispares (por ejemplo, *investigador, actor, rey*). Si pregunta por las formas del lexema *toro* comprobará las posibilidades que permite la gramática.

3.2.3 Generar todas las formas

Esta opción genera todas las formas posibles con los datos existentes en el diccionario. Hay que esperar unos segundos (la espera aumenta con el tamaño del diccionario). Se crea un fichero llamado FORMAS.DIC, que se puede editar saliendo al sistema operativo. Hay que advertir que la presentación de las formas es completamente desordenada y tiene que ver con el orden en que aparecen las entradas del diccionario. Esta opción es útil, por ejemplo, para comprobar el número de formas que se generan y, por extensión, que se pueden analizar (ya que la gramática analiza y genera exactamente las mismas formas).

3.3 Actualizar y editar el diccionario

En esta sección se describen las distintas maneras de trabajar con el diccionario. En principio, la única opción necesaria es la de "Editar el diccionario," pero hemos desarrollado la opción "Ver / Actualizar el diccionario" como herramienta para facilitar la codificación de nuevas entradas.

3.3.1 Ver y Actualizar el diccionario

Si se escoge alguna de las opciones de "Ver," en la pantalla aparecerán pares compuestos por *unidades léxicas* y *cadenas*, separados por dos puntos. La utilidad es comprobar rápidamente si existe una entrada concreta y, en función de la existencia o no, crear una entrada nueva (con la opción "Actualizar") o corregir los datos equivocados (con la opción "Editar")

Para "Actualizar el diccionario," una vez escogida la modalidad pertinente (raíz verbal, forma verbal lexicalizada, nominal), basta con seguir las instrucciones que aparecen en la pantalla de "Mensajes." Ténganse en cuenta los comentarios anteriores sobre los blancos, comillas, etc., pues el programa no puede detectar esos errores y los incluye en las entradas que produce, con el consiguiente fallo en análisis y generación.

Nota: Hay una excepción a la norma de no dejar espacios en blanco. En aquellos casos donde se requiera escribir varios valores para un mismo rasgo **es obligatorio separar cada valor con un y sólo un espacio en blanco.** Esto se aplica por tanto a todos los rasgos con disyunción de valores:

- **tipo_raiz**
- **tipo_des**
- **tipo_plu**
- **tipo_gen**

Recomendamos que antes de crear nuevas entradas, se hayan identificado los valores que llevarán los rasgos (bien consultándolos en los modelos que se proporcionan, bien elaborándose tablas particulares). De esa manera, se evitarán errores que, aunque fáciles de corregir, son fruto de la precipitación.

3.3.2 Editar el diccionario

Con esta opción podemos trabajar con el diccionario (o base de datos léxica). Es decir, podemos corregir valores equivocados, copiar y borrar entradas, etc.. Para ello tenemos que seguir las convenciones de Turbo-Prolog para escribir bases de datos internas:

- no pueden aparecer caracteres en mayúsculas (es decir, no puede haber variables, sólo constantes)
- no puede haber espacios, excepto dentro de cadenas entrecomilladas,
- no se pueden escribir comentarios
- no pueden aparecer líneas vacías
- todos los símbolos deben aparecer entre comillas. Esto significa que todos los valores de los rasgos deben ir entrecomillados

El editor de GRAMPAL es el mismo que el de Turbo-Prolog. Dentro de esta opción hay una ayuda especial para el editor proporcionada por la compilación. Se obtiene apretando la tecla de función F1.

Para salir del editor del diccionario, pulse ESC. Si ha hecho algún cambio, el programa le preguntará si quiere guardarlos y saldrá al Menú Principal. Si no tiene experiencia en Turbo-Prolog, le recomendamos que utilice preferentemente la opción "Actualizar" para crear nuevas entradas. Sin embargo, editar el diccionario es interesante para ver cómo está codificado. Por supuesto, es imprescindible utilizarlo para corregir errores.

Apéndice A. Mensajes de error

El programa de demostración no pretende ser una aplicación contrastada y consistente a los problemas y errores. Por ello, es posible que durante la ejecución el programa se "quede colgado" y aparezca un mensaje de error numérico. Lo más habitual es que se trate de un problema de memoria: esto sucede cuando se consulta el diccionario (es decir, se carga en memoria) repetidamente.

Otro error común es que se hayan editado más de 36 ventanas durante la ejecución. Esta es una limitación de Turbo-Prolog, que no permite utilizar muchas ventanas. Como GRAMPAL tiene bastantes opciones con diferentes ventanas, si cambiamos varias veces de menú llega un momento en el que el programa se interrumpe.

En los dos casos, la solución es volver a ejecutar GRAMPAL.

Si le aparece el mensaje de error número 2001 es que el programa ha intentado escribir en el disco vigente y no hay espacio suficiente en él. Esto puede ocurrir cuando se le pide al sistema que "Genere todas las formas posibles" (ver la sección 3.2.3). Se crea un fichero llamado FORMAS.DIC, cuyo tamaño dependerá del número de entradas de diccionario. Es aconsejable que disponga de al menos 100 K libres en el disco.

Apéndice B. Lista de Unidades Léxicas del Diccionario

El diccionario que se incluye con el programa de demostración contiene todos los casos significativos de la Flexión del español, tal y como se exponen en la tesis. Es decir, aparecen codificadas todas las palabras que se dan como ejemplos de un determinado tipo de flexión. El diccionario se puede aumentar fácilmente siguiendo los modelos pertinentes para cada caso.

B.1 Nombres

actor
afueras
agua
alicates
bienes
camión
casa
crisis
estrés
estudiante
examen
hipérbaton
ilusión
informática
investigador
jabalí
jersey
león
lord
luna
maniquí
maravedí
matemáticas
niño
pez
poeta
presidente
príncipe
programa
régimen
rey
sol
tijeras
toro
virus

B.2 Adjetivos

bonito
bueno
grande
grandote
isósceles
rubiales

B.3 Verbos

abolir
acertar
acontecer
amar
andar
asir
aullar
averiguar
caer
caminar
cantar
cazar
ceñir
cocer
comer
concernir
conducir
conocer
contar

dar
decir
empezar
errar
estar
fabricar
haber
hacer
huir
informar
ir
jugar
leer
mullir
oír
oler
pagar
partir
placer
poder
poner
programar
querer
reír
saber
seguir
sentir
ser
tener
traer
valer
venir

B.4 Pronombres

este
ese
aquel
quién
qué
cuánto
cuál
quien
que
cuyo

B.5 Artículos

el
un

B.6 Ordinales

primero
segundo
tercero

B.7 Estadística

nombres: 34
adjetivos: 6
verbos: 50
pronombres: 10
artículos: 2
ordinales: 3

Total unidades léxicas: 105

Total entradas de diccionario: 490 (incluyendo desinencias y morfemas flexivos)

Total formas generadas 5431