Análisis Léxico no Determinista: Etiquetación Eficiente del Lenguaje Natural

Jorge Graña Gil
Miguel A. Alonso Pardo
Alberto Valderruten Vidal.

Octubre 1994

Resumen

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

En este contexto, el trabajo descrito[1] pretende dar un tratamiento práctico al problema de la etiquetación de lenguas naturales, tanto en lo que se refiere a la velocidad de tratamiento como a su dominio de aplicación. Para ello, los autores han partido de la base representada por la experiencia acumulada durante años en el tratamiento de este mismo problema, a nivel de los lenguajes formales de programación. En concreto, la técnica propuesta representa un sustancial distanciamiento de las aplicadas tradicionalmente en el etiquetamiento de textos, fundamentadas en el uso de bases de datos, para orientarse hacia métodos basados en el uso de autómatas de diseño específico que incorporan mecanismos operacionales para el tratamiento del conocimiento lingüistico disponible acerca de la estructura morfológica de las palabras. En este sentido los autores justifican la elección del nuevo formalismo, al tiempo que presentan los primeros resultados prácticos obtenidos.

Palabras Clave: Análisis Léxico, Autómatas Finitos, Etiquetado.

1 Introducción

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

- En un lenguaje de programación, cada palabra se corresponde con un único componente léxico. En el caso de las lenguas naturales, una palabra puede tener múltiples acepciones que se corresponden con diferentes categorías gramaticales.
- Las lenguas naturales poseen un complejo entramado de reglas morfológicas que controlan la construcción de las variantes válidas de cada palabra, aspecto éste que no aparece en los lenguajes de programación.

En consecuencia, un analizador léxico para lenguas naturales debe proveer mecanismos no deterministas que permitan obtener todas las posibles interpretaciones de una palabra, así como mecanismos operacionales que

[1] parcialmente financiado por la Xunta de Galicia, por el proyecto XUGA10501A93.

le permitan rentabilizar las posibilidades que ofrece el análisis morfológico. Más exactamente, el analizador debe proveer todos los posibles reconocimientos para una palabra dada y no debe pasar a reconocer un nuevo componente léxico hasta que no se hayan agotado todas las posibilidades para el precedente. Sólo de esta manera se puede asegurar que el posterior análisis sintáctico y semántico sea capaz de tratar correctamento el texto, seleccionando de entre todas las opciones aquellas que mejor concuerden con la estructura general del mismo.

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

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

- El tamaño desproporcionado que deberá tener la base de datos para ser capaz de obtener buenos resultados con textos grandes. Ello bien obliga a que residan en almacenamiento secundario gran parte de los recursos del sistema, o bien limita notablemente sus prestaciones [8].

- Cada vez que se encuentra una nueva palabra, se deben introducir todas las formas derivadas de su lema, si queremos que la base de datos sea consistente. Por ejemplo, para cada verbo sería necesario introducir todas las formas verbales relativas a su conjugación. Se puede evitar la entrada manual mediante un generador de formas derivadas [7], pero aún así se tiene el problema del crecimiento exponencial del tamaño de la base de datos con respecto al número de lemas, lo que degrada el rendimiento ya que introduce retardos en el tiempo de acceso, siempre considerando que se posee una buena estructura de índices.

Para evitar esos inconvenientes hemos considerado que la mejor solución es utilizar un generador de analizadores léxicos en el que se puedan incorporar las reglas morfológicas y el comportamiento no determinista, adoptando el concepto de autómata como formalismo operacional. El resultado es un programa ejecutable que se encarga de realizar las labores de etiquetación con un buen rendimiento temporal y un consumo de memoria mínimo. Como punto de partida para el diseño de la nueva herramienta se ha optado por utilizar Flex[2] [6], un generador de analizadores léxicos estándar en el sistema operativo Unix.

En la sección 2 se describe el formalismo utilizado para implementar los mecanismos operacionales derivados de las reglas morfológicas y se muestra un pequeño ejemplo. En la sección 3 se describen los mecanismos no deterministas incorporados en el generador de analizaores léxicos propuesto, incluyendo un ejemplo en el que se analizan múltiples interpretaciones de una palabra. Aquellos lectores que no estén interesados en un conocimiento en profundidad de los mecanismos internos del analizador, pueden pasar por alto el código incluido en los ejemplo de las secciones 2 y 3. La integración del analizador léxico con el analizador sintáctico se estudia en la sección 4, mostrando los mecanismos de comunicación entre ambos. En la sección 5 se realiza un estudio acerca de la arquitectura concreta del analizador construido para la lengua española. La sección 6 es una conclusión del trabajo presentado y su desarrollo futuro. En el apéndice A se muestra un abanico de los resultados prŕacticos del prototipo. Finalmente el apéndice B contiene las tablas en las que se indican las categorías y reglas utilizadas para construir el prototipo del analizador en el caso del español.

2 Las reglas léxicas

Las palabras de una lengua como la española se pueden dividir en dos partes: el lexema y los sufijos. El lexema es la parte invariante de la palabra. En ciertos casos, la palabra está formada únicamente por el lexema. Tal

[2] La versión utilizada ha sido la 2.4.6 de Enero de 1994.

es el caso de las preposiciones. Sin embargo, normalmente las palabras se forman mediante la aposición de una serie de sufijos, como ocurre, por ejemplo, en el caso de los sustantivos, los adjetivos y los verbos. Cuando de un lexema se puede obtener un conjunto de palabras que derivan de él, se toma una de esas palabras para representar el conjunto: es lo que se denomina el *lema* de ese conjunto de palabras. En el caso de los sustantivos y los adjetivos el lema lo constituye la forma masculina singular, y en el de los verbos el infinitivo.

En ciertos casos, como en el de los verbos irregulares, las cosas no son tan simples, puesto que no todas las formas de la conjugación tienen la misma raíz. Consideremos por ejemplo el verbo `pensar`. Parte de la conjugación utiliza como raíz `pens`[3], pero otras utilizan `piens` como raíz[4]. Sin embargo, todas las formas de la conjugación tienen un único lema: `pensar`.

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

2.1 Una aproximación intuitiva

Para aclarar el significado de las reglas léxicas utilizaremos como ejemplo los sustantivos. En español hemos detectado hasta 17 formas distintas de formar el género de una palabra, catalogadas como $G1$, $G2$,. . .,$G17$. Hay grupos muy comunes, como por ejemplo el $G1$, que añade una `o` en el caso del masculino y una `a` en el del femenino. Otros son mas inusuales, como el grupo $G7$, que forma el masculino añadiendo la terminación `que` y el femenino mediante la terminación `ca`. Un ejemplo de este grupo lo constituyen las palabras `cacique` y `cacica`.

Para formar el número se han detectado 9 formas, catalogadas como $N1$, $N2$,. . .,$N9$. El grupo más común es el $N1$, que añade una `s` para formar el plural y ningún carácter para el singular. Un grupo bastante inusual es el $N4$, que utiliza la terminación `c` para el singular y la `ques` para el plural. Es el caso de las palabras `frac` y `fraques`.

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

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

Todo este conocimiento debe ser incorporado en la etapa de análisis léxico para determinar los componentes que se han reconocido. Puesto que las reglas léxicas son fijas, para ampliar posteriormente el conjunto de palabras que es capaz de reconocer el analizador tan sólo es necesario introducir el lexema y establecer los grupos con los cuales se forman las palabras válidas que comparten dicho lexema. Estas modificaciones se ven también facilitadas por el hecho de utilizar un generador de analizadores, en lugar de programar directamente en un lenguaje de propósito general.

2.2 Las condiciones de arranque

Para facilitar la implementación de las reglas léxicas, se pueden utilizar las *condiciones de arranque*[5] de Flex, que permiten continuar la búsqueda de expresiones regulares una vez que ha sido detectado el lexema de una palabra.

Las condiciones de arranque proporcionan un mecanismo para establecer la ejecución condicional de las reglas. Cualquier regla cuyo patrón[6] comience por <*sc*>, se activará solamente cuando la condición de arranque *sc* esté activa. Existen dos tipos de condiciones de arranque:

- *Inclusivas*, que se declaran mediante `%s` en la sección de declaraciones del programa Flex. Se caracterizan porque cuando se activan, también están activas todas las reglas que no comienzan por una condición de arranque.

- *Exclusivas*, que se declaran mediante `%x`. Cuando se activa una de estas condiciones de arranque, sólo permanecen activas aquellas reglas que comiencen con la misma condición.

[3]Por ejemplo, las formas **`pensé`**, **`pensaré`**, etc.
[4]**`pienso`**, **`piensas`**, etc.
[5]*Start conditions.*
[6]La parte izquierda de la regla.

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

2.3 Implementación de las reglas léxicas

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

1. Utilizar reglas sin condición de arranque para reconocer los lexemas, los cuales constituirán los patrones de tales reglas.

2. Definir una condición de arranque para cada grupo de reglas léxicas. Utilizar los sufijos como patrones de las reglas incluidas en esa condición.

3. En las acciones que se disparan por el reconocimiento de un lexema, lanzar la condición de arranque que represente al grupo en el cual se crean las palabras derivadas de ese lexema.

4. En las acciones de las reglas con condiciones de arranque se puede tomar una de las siguientes alternativas:

   (a) Lanzar una condición de arranque que indique los sufijos que deben ser reconocidos a continuación. Esto se utiliza, por ejemplo, cuando después de reconocer el género de un sustantivo es preciso reconocer su número.

   (b) Si no es necesario comprobar la existencia de más sufijos, dar por terminado el reconocimiento de la palabra.

Todas las condiciones de arranque deberán ser exclusivas. Con ello se evita el problema de un posible reconocimiento de una palabra dentro de otra.

Es conveniente considerar la existencia de una condición `ERROR`, a la que se envíe el analizador cuando una palabra no pueda ser reconocida[7]. Cuando el analizador entra en este estado de error, debe actuar en *modo pánico*[8] [1], saltando todos los caracteres siguientes hasta que encuentra un carácter de sincronización, que en nuestro caso será un separador.

2.4 Un ejemplo sencillo

Para que se pueda comprender mejor la implementación de las reglas léxicas, vamos a mostrar un pequeño ejemplo. Se trata de reconocer los sustantivos que forman el género mediante $G1$ y el número mediante $N1$.

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

```
/* Error */
%x E

/* gender */
%x G1

/* Number */
%x N1
```

Después definimos los dos patrones siguientes:

[7]Puede ser que no se reconozca el lexema como válido o que se reconozca el lexema pero no se pueda aplicar ninguna de las reglas léxicas para construir la palabra.

[8]*Panic mode* en la literatura anglosajona.

- `sep`, formado por los caracteres separadores. Estos caracteres se utilizan para evitar reconocer una palabra dentro de otra. El conjunto de caracteres separadores está formado por los siguientes elementos:
  - Espacio.
  - Tabulador.
  - Coma.
  - Punto y coma.
  - Dos puntos.
  - Punto.
  - El carácter de nueva línea.
- `nosep`, en el que se engloban todos los caracteres que no son separadores y que por tanto pueden aparecer dentro de una palabra.

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

```
/* Separator characters */
sep   [" "\t,;:\.\n]

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

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

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

Mediante la llamada a la función `yymore` indicamos a Flex que queremos seguir añadiendo en `yytext`[9] los caracteres que sean reconocidos a continuación.

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

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

El prefijo <G1> antes de los patrones `o` y `a` indica que ambas reglas sólo se activarán cuando la condición de arranque `G1` haya sido activada mediante `BEGIN G1`. Como `G1` ha sido definida en modo exclusivo, después de `BEGIN G1` únicamente estas dos reglas estarán activas. Como todas las palabras que forman el género en $G1$ forman el número en $N1$, ambas reglas utilizan `BEGIN N1` para activar la condición de arranque `N1`.

Las reglas para el reconocimiento del número son la siguientes:

[9] En la variable `yytext` se almacena el texto reconocido en cada regla.

```
/* Terminations for number: Case 1 */
<N1>s/{sep}    {/* Plural */
printf(", plural --> %s",yytext);
BEGIN(INITIAL)
}
<N1>""/{sep}   {/* Singular */
printf(", singular --> %s", yytext);
BEGIN(INITIAL);
}
<N1>.          {/* Bad word. Error. */
yymore();
BEGIN E;
}
```

La terminación /{sep} constituye un *contexto de cola*[10]. Se utiliza para indicar que el patrón de esa regla deberá estar seguido por un separador, pero el carácter separador no es incluido en **yytext** y será reconocido en patrones de reglas posteriores, pero no en la actual. Se utiliza en estos patrones para evitar reconocer una palabra incompleta.

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

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

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

La acción BEGIN(INITIAL) hace que el reconocedor vuelva a las reglas que no tienen condición de arranque[11].

Por último las siguientes reglas permiten tratar los separadores.

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

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

En este ejemplo sólo se muestra por pantalla el género y el número. En el analizador real, se almacenan los datos en una estructura que se pasa al analizador sintáctico. El modo en que se enlazan los analizadores léxico y sintáctico se muestra en la seccion 4

[10] *Trailing context.*

[11] Realmente es como si existiese una condición de arranque implícita llamada **INITIAL** en todas aquellas reglas que caracen de condición de arranque.

Un aspecto importante que hay que tener en cuenta es el de la definición de `yytext`. La variable `yytext` está definida por defecto en el generador como un puntero a `char`. La ventaja de usar esta defininción es que no se producirán *overflows* en el buffer cuando se reconozcan componentes léxicos muy largos[12], ya que la memoria se asigna a `yytext` dinámicamente[13]. La desventaja es que no se puede modificar el contenido de `yytext`, ya que los resultados que se obtendrían serían impredecibles. Esto último implica que no se pueden utilizar funciones como `input` y `unput`.

Se puede definir la variables `yytext` como un array mediante la utilización de `%array` en la sección de declaraciones del generador. El tamaño viene determinado por el valor de `YYLMAX`, originalmente 8 Kilobytes. Utilizando esta definicón se puede modificar el valor de `yytext` sin problemas. El único inconveniente que podría surgir sería el derivado del límite de tamaño impuesto por el array. En nuestro caso no es ningún problema, ya que no existen palabras de miles de caracteres[14].

3 El comportamiento no determinista

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

- Es preciso disponer de información estadística actualizada acerca de las probabilidades de aparición conjunta de un conjunto de categorías de palabras[15]. Es necesario disponer de un texto amplio ya etiquetado para poder entrenar al modelo. De la bondad del conjunto de entrenamiento dependerá en buena medida la corrección de los resultados obtenidos.

- Probablemente se está tratando de incorporar a nivel léxico información correspondiente a nivel sintáctico e incluso semántico. Esto provoca una repetición inadecuada de las tareas realizadas en cada uno de dichos niveles y un desaprovechamiento manifiesto de la información disponible. Además, debido a la imposibilidad de aplicar modelos matemáticos complejos a conjuntos grandes de palabras, se están realizando suposiciones sobre ventanas muy pequeñas que no abarcan, en la mayoría de los casos, estructuras sintácticas completas, por lo que la posibilidad de error es grande.

- La carga computacional asociada es muy elevada, lo que disminuye el rendimiento. Para mantenerlo entre límites aceptables, se precisaría ejecutar el programa en máquinas con una disponibilidad generosa de recursos.

- Muchas veces estos modelos trabajan en conjunción con bases de datos, lo que degrada todavía más el rendimiento, ignorando además el conocimiento disponible acerca de la formación de las palabras a nivel léxico.

Ello justifica que el enfoque que hemos adoptado haya sido el de ampliar las capacidades de los reconocedores clásicos en terоría de los lenguajes formales, incorporándoles un comportamiento no determinista.

3.1 El comportamiento determinista de los reconocedores clásicos

Los reconocedores generados para los lenguajes de programación son deterministas. Esto quiere decir que una palabra es analizada una sóla vez y las acciones de las reglas disparadas se llevan a cabo también una sóla vez. Sin embargo, en nuestro caso, mediante la utilización de la acción `REJECT`[16] se puede indicar al reconocedor que rechace la entrada reconocida en los patrones de las reglas de la condición de arranque actual y que seleccione la segunda mejor regla cuyo patrón coincida con la entrada[17]. Aunque el propósito para el que fue creada esta acción se refiere fundamentalmente a la recuperación de errores, su utilización permite conseguir

[12] Por ejemplo, bloques de comentarios de miles de caracteres.

[13] Cuando `yytext` cambia de tamaño es necesario volver a reconocer todos los caracteres del componente léxico, lo cual puede provocar una degradación importante del rendimiento para componentes léxicos muy grandes.

[14] Si se desea, se puede cambiar el tamaño por defecto de `yytext` utilizando un `#define` para establecer `YYLMAX` al valor deseado.

[15] Por ejemplo, la probabilidad de que un sutantivo esté seguido de un verbo y un adverbio, etc.

[16] Esta acción fue incorporada en la versión 2.1 de Flex.

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

un comportamiento no determinista local en la condición de arranque en la que se ejecutó[18]. Sin embargo, mediante la utilización de `REJECT` no es posible ir atrás en el análisis más que hasta el comienzo de la condición de arranque actual. Si el reconocedor pasa por varias de estas condiciones de arranque hasta llegar al final de la palabra, como es nuestro caso, no es posible ir retrocediendo paso a paso hasta la condición inicial. El origen se encuentra en la falta absoluta de dependencia contextual a nivel léxico de los lenguajes de programación, aspecto este ineludible en los lenguajes de comunicación humana.

3.2 La incorporación del no determinismo a los reconocedores clásicos

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

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

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

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

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

El reconocedor intentará entrar otra vez por el mismo patrón de antes, pero esta vez `semaphore` tendrá un valor distinto de 0[19], lo que indica que no debemos escoger ese patrón, sino otro que es la *siguiente mejor elección*. Se decrementa su valor en 1 y se hace un `REJECT`, con lo cual se busca precisamente la siguiente mejor elección. El siguiente patrón reconocido será el correcto puesto que `semaphore` tendrá valor 0. Se incrementa el valor de `match_no` y se prosigue con el proceso. Además, se debe de contemplar la posibilidad de una salida para el caso de que ya no haya más patrones correctos para la palabra examinada. Esto se consigue mediante una llamada a `yymore()` y a `BEGIN S` desde una regla con el punto como patrón. La condición de arranque `S` contiene la regla siguiente:

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

donde la acción `BEGIN 0` es equivalente a la activación de la condición de arranque inicial. Esta regla es la única en la que se realiza un `return`, por lo que una vez que se llama a la función `yylex` desde el analizador sintáctico,

[18] Un aspecto importante a tener en cuenta es que la acción `REJECT` actúa como una ramificación, de modo que el código situado después de ella en las acciones de las reglas no es ejecutado.

[19] Concretamente tendrá valor 1.

no se devuelve el control hasta que se llega a la condición `S`, hecho que se da cuando ya se han examinado todos los posibles componentes léxicos válidos para una palabra.

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

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

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

3.3 Un pequeño ejemplo

Para facilitar la compresión de la incorporación del comportamiento no determinista, vamos a utilizar un pequeño ejemplo. Como en español hay un gran número de reglas léxicas, lo que nos obligaría a definir numerosas condiciones de arranque para crear un ejemplo mínimamente real, hemos considerado más conveniente utilizar como ejemplo un pequeño analizador léxico en el que se muestra cómo reconocer todas las variantes de la palabra inglesa `bellow`. Este ha sido el ejemplo sobre el que hemos trabajado con Vern Paxon, creador de Flex. La palabra `bellow` tiene cuatro entradas posibles en un diccionario[20]:

bellow^1 *noun* bramido, rugido.

bellow^2 *intransitive verb* gritar, vociferar.

bellow^3 *transitive verb* bramar, rugir.

bellows^4 *noun plural* fuelle.

El problema surge al tener que reconocer una palabra como `bellows`[21], que puede ser:

- El singular del sustantivo `bellows`.
- El plural del sustantivo `bellow`.
- La tercera persona del presente del verbo `bellow`.

Para ser capaces de tratar estas definiciones, debemos crear dos condiciones de arranque, que son las siguientes:

- `N` para los sustantivos.
- `V` para los verbos.

Ello es posible incluyendo la siguiente línea de código, en la que también consideramos la declaración de la condición `E` de error:

```
/* Start conditions */
%x E N V
```

A continuación se definen los patrones de los separadores y de los no separadores:

```
/* Separators */
sep     [" ",;:\.\t\n]

/* Non separators */
nosep   [^" ",;:\.\t\n]
```

[20] Estas acepciones han sida sacadas del *Diccionario Collins English-Spanish* editado por Grijalbo.

[21] El honor de encontrar este ejemplo ha sido del Prof. Jorge Graña. Su habilidad en el manejo de diccionarios es incuestionable.

Algo muy sencillo, pero no por ello menos importante es el declarar las variables enteras `match_no` y `semaphore`:

```
%{
unsigned int match_no, semaphore;
%}
```

A continuación ya podemos escribir las reglas léxicas con el no determinismo incorporado, no sin antes haber escrito un pequeño trozo de código en el cual se inicializa a cero el valor de las dos variables anteriores. Esta porción de código situado después de `%%` y encerrada entre `%{` y `%}` se ejecutará cada vez que se llame a la función `yylex`.

```
 %%

 %{
 /* Code executed each time yylex is called */
 match_no = 0;
 semaphore = 0;
 %}

 /* Error condition: Panic mode */
 <E>{nosep}*/{sep}    { printf("%s -> error \n\n", yytext);
BEGIN(INITIAL);RETURN(1);
      }

 /* Verbs */
 <V>s/{sep}           printf("%s -> verb, 3rd        \n",yytext);REJECT;
 <V>ed/{sep}          printf("%s -> verb, past       \n",yytext);REJECT;
 <V>ing/{sep}         printf("%s -> verb, gerund     \n",yytext);REJECT;
 <V>""/{sep}          printf("%s -> verb, infinitive \n",yytext);REJECT;
 <V>""/.              yyless(0); BEGIN(INITIAL);

 /* Nouns */
 <N>s/{sep}           printf("%s -> noun, plural   \n",yytext);REJECT;
 <N>""/{sep}          printf("%s -> noun, singular \n",yytext);REJECT;
 <N>""/.              yyless(0);BEGIN(INITIAL);

 /* WORDS */
 bellow               { if ( semaphore!=0) { semaphore--; REJECT; };
match_no++; semaphore=match_no;

yymore(); BEGIN(V);
      }

 bellow               { if ( semaphore!=0) { semaphore--; REJECT; };
match_no++; semaphore=match_no;

yymore(); BEGIN(N);
      }

 bellows              { if ( semaphore!=0) { semaphore--; REJECT; };
match_no++; semaphore=match_no;

yymore(); BEGIN(N);
      }

 /* Separators */
 " "                  |
```

```
\t                    |
\n                    |
","                   |
","                   |
";"                   |
":"                   |
"."                   printf("%s -> separator \n\n'', yytext);return(1);

.                     yymore(); BEGIN (E);

<<EOF>> yyterminate();

%%
```

Mediante la utilización de las dos líneas iniciales de código de todas las reglas con comportamiento no determinista

```
if ( semaphore!=0) { semaphore--; REJECT; }
match_no++;semaphore=match_no;
```

se comprueba si el valor del semáforo es correcto. Si no lo es, se realiza un `REJECT`, con lo cual el resto de las acciones de esa regla no se ejecutan. En caso contrario, se actualizan las variables convenientemente.

La última regla permite evitar que el reconocedor genere errores cuando se intenta llamar a `yylex` y ya se ha alcanzado el fin de fichero. Puede ser eliminada y el programa seguirá funcionando, pero se recomienda que sea incluida.

En este ejemplo, por sencillez, no se ha utilizado la condición de arranque `S`, sino que `yylex` devuelve el control cuando encuentra un separador. En un programa real los espacios, tabuladores y retornos de carro se ignorarían y las marcas de puntuación se reconocerían como componentes léxicos.

Aunque el ejemplo se ha puesto en inglés por cuestiones de legibilidad, en español se dan muchos más casos de ambigüedades. Un ejemplo clásico es el de la palabra `para` puede ser reconocida como:

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

Ello hace del no determinismo una capacidad fundamental y absolutamente necesaria en el ytratamiento de los lenguajes naturales en general y del español en particular.

4 Integración con el analizador sintáctico

Los analizadores léxicos generados se integran perfectamente con los analizadores sintácticos generados mediante Yacc [5], Bison [3] o ICE [10]. Esta integración se basa fundamentalmente en:

- El valor devuelto por las llamadas a la función `yylex`, que es un entero representando el componente léxico reconocido[22], excepto el valor 0, que representa la condición de fin de fichero.
- El valor semántico de cada componente léxico, almacenado en la variable `yylval`.

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

[22] Este entero se identifica con los componentes léxicos mediante los valores indicados en los `#define` incluidos en el fichero `.tab.h`. Tales valores se generan a partir de las definiciones `%token` incluidas en el programa fuente, que debe tener una extensión `.y`.

4.1 La variable token

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

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

```
struct ice_lex_object
{
char         *word;
INT_LIST     category,
     gender,
     number,
     person,
     verbal_tense;
STRING_LIST  lemma;
}
```

donde el siginificado de los distintos campos es el siguiente:

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

Estos dos campos cosnstituyen la información básica que se debe almacenar en `token` para permitir el enlace entre los anlizadores léxico y sintáctico. Por tanto, debe ser incorporada en cualquier programa que pretenda utilizar el reconocimiento no determinista de los componentes léxicos. El resto de la información almacenada en la variable `token` es específica de la aplicación que se trata en este trabajo. Diferentes aplicaciones pueden redefinir la estructura `ice_lex_object` para adaptar el contenido de los campos a sus necesidades específicas. Otra posible solución hubiese sido situar esta información dependiente de la aplicación en la variable `yylval`. Para ello habría que definir dicha variable como un puntero a una lista de estructuras en las que se almacenaría tal información. El problema surge al tratar de mantener la consistencia entre la lista de componentes léxicos[23] y la lista de `yylval`'s. Para facilitar el mantenimiento de esta consistencia se ha considerado más adecuado almacenar toda la información que tiene relación con las capacidades no deterministas en una única estructura. Con ello se consigue adicionalmente un alto grado de aislamiento de estas nuevas características con respecto a las capacidades estándar de los reconocedores generados por Flex en el caso de los lenguajes de programación. Los campos con información adicional dependiente de la aplicación que se han utilizado son:

- `gender`. Es una lista de enteros que almacena el código que identifica el género de aquellas palabras en las que es aplicable esta característica. Para aquellos componentes léxicos en los que no es aplicable, recibe el entero que se corresponde con el valor `No-Aplicable`. Esta lista está coordinada con la lista `category`, lo que quiere decir que el primer elemento de la lista de género indica el género del primer token en la lista de categorías, el segundo elemento en la lista de géneros el valor del género para el segundo componente léxico, etc.
- `number`. Este campo es una lista de enteros que indica el valor del número para aquellas palabras en las cuales es aplicable. Por tanto, es equivalente a la lista `gender`, salvo que los valores conciernen al número en vez de al género.
- `person`. Al igual que los campos anteriores, es una lista de enteros coordinada con la lista de categorías, pero que en este caso indica la persona[24].

[23] Representada como una lista de `ice_lex_object`'s.
[24] Aplicable en el caso de verbos y pronombres.

- verbal_tense. Lista de enteros que indica el tiempo verbal de una palabra, si es aplicable.
- lemma. Este campo es una lista de cadenas de carácteres que indica el lema de cada palabra según cada tipo de componente léxico que ha sido reconocido para dicha palabra. Por ejemplo, tomemos la palabra para, que es reconocida como:
  - Una preposición, en cuyo caso el lema es para.
  - Dos formas verbales del verbo parar[25], en cuyo caso el lema es parar para ambas. Cada forma verbal se corresponde con un componente léxico distinto y por lo tanto con dos elementos en cada una de las lista presentes en la estructura token.
  - Dos formas verbales del verbo parir[26], en cuyo caso el lema es parir para ambas.

4.2 Estructuras de datos auxiliares

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

4.2.1 La lista de enteros

El tipo INT_LIST implementa las listas de enteros. La declaración de este tipo se realiza como se muestra en las siguientes líneas de código:

```
typedef struct ITEM_INT_LIST *INT_LIST

struct ITEM_INT_LIST
{
   int         data_item;
   INT_LIST    next_item;
};
```

Por tanto, cuando se declara una variable de tipo INT_LIST, se está declarando realmente un puntero a un objeto en memoria de tipo ITEM_INT_LIST, el cual es una estructura que almacena un valor entero, identificado como el campo data_item y un puntero al siguiente elemento de la lista, identificado mediante el campo next_item. Para manejar este tipo de listas, se han definido las siguientes funciones:

- RESET_INT_LIST. Esta función toma como argumento un puntero a un valor de tipo INT_LIST[27] y se encarga de eliminar una lista de enteros, liberando todas las posiciones de memoria ocupadas por todos sus elementos y estableciendo el puntero de inicio de la lista al valor NULL.
- IS_EMPTY_INT_LIST. Esta función comprueba si el valor que se le pasa como argumento es el inicio de una lista de enteros o no. El valor devuelto es un entero con significado booleano.
- CONS_INT_LIST. Esta función, al igual que la función cons del Lisp[4], añade un elemento en la cabeza de una lista de enteros. Para ello toma como argumento un puntero a un elemento INT_LIST y un valor entero, que será asignado al campo data_item de la estructura ITEM_INT_LIST que representa al nuevo elemento de la lista.
- NEXTL_INT_LIST. Esta función realiza un trabajo similar al de su homónima nextl en Lisp: elimina el primer elemento de una lista de enteros y devuelve el resto de la lista. Al igual que la función Lisp, esta función modifica físicamente sus argumentos, ya que no crea una lista nueva, sino que desengancha realmente el primer elemento de la lista argumento[28].

[25] La tercera persona del presente de indicativo y la segunda del imperativo.
[26] La primera y la tercera persona del singular del presente de subjuntivo.
[27] Puesto que los valores de tipo INT_LIST son punteros a valores de tipo struct ITEM_INT_LIST, al valor recibido como argumento por ésta y otras funciones de manejo de listas es realmente un puntero a un puntero a una estructura ITEM_INT_LIST. El doble nivel de indirección es requerido puesto que se va a modificar el valor del puntero incicial de la lista.
[28] El argumento que se pasa a esta función es un puntero a una variable de tipo INT_LIST.

- CAR_INT_LIST. Esta función obtiene, utilizando terminología Lisp, el car de una lista de enteros. Al igual que la función Lisp, sólo realiza operaciones de lectura, por lo que no modifica la lista argumento. En este caso el argumento es realmente un valor de tipo INT_LIST.

Con estas funciones se dispone de un conjunto lo suficientemente amplio de operadores para tratar las listas de enteros sin necesidad de recurrir a manipulaciones manuales de sus elementos.

4.2.2 La lista de cadenas de carácteres

Las listas de cadenas de carácteres, que por ahora sólo se utilizan para tratar el campo lemma de la variable componente léxico, se implementan mediante la utilización del tipo STRING_LIST, el cual se declara tal y como se muestra en las siguientes líneas de código:

```
typedef struct ITEM_STRING_LIST *SRING_LIST;

struct ITEM_STRING_LIST
{
   char *data_item;
   STRING_LIST next_item;
}
```

Al igual que ocurría con las lista de enteros, una variable de tipo STRING_LIST es un puntero a una estructura ITEM_STRING_LIST que representa el primer elemento de la lista de cadenas de carácteres. Estas estructuras tienen dos campos:

- data_item, que almacena un puntero a una cadena de caracteres.
- next_item, que almacena un puntero al siguiente elemento de la lista.

El conjunto de funciones que se han definido para tratar con este tipo de listas es equivalente al que se definió para trabajar con las listas de enteros. Así, se dispone de las siguientes funciones:

- RESET_STRING_LIST, que elimina la lista de cadenas de carácteres, teniendo cuidado de liberar toda la memoria ocupada, no sólo por las estructuras que componen cada elemnto, sino por las propias cadenas de caracteres a las que apunta cada una de dichas estructuras.
- IS_EMPTY_STRING_LIST comprueba si una lista de cadenas de carácteres esta vacía.
- CONS_STRING_LIST. Añade un elemento al principio de lista. Toma como parámetros un puntero a INT_LIST y un puntero a char. Esta función crea una nueva estructura ITEM_STRING_LIST y la enlaza al principio de la lista. Para rellenar el campo data_item se crea una nueva cadena de caracteres en la que se copia el contenido de la que se pasa como argumento, por lo que dicho argumento permanece sin cambios.
- NEXTL_STRING_LIST, elimina el primer elemento de una lista de cadenas de carácteres.
- CAR_STRING_LIST, devuelve la cadena de caracteres almacenada en el primer elemento de la lista de cadenas de carácteres.

4.3 Funciones de actualización del componente léxico

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

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

A continuación se muestra una lista de las funciones que se han considerado. Todas ellas reciben al menos un argumento: un puntero a una estructura ice_lex_object. Si hay argumentos adicionales, se indican en la descripción que se hace de cada una de ellas.

En primer lugar se van a mostrar las funciones que realizan el proceso de inicialición de la variable token y del conjunto de las listas de enteros y cadenas de carácteres:

- `reset_token`. Esta función vacía el contenido de una variable de tipo `ice_lex_token`, lo cual significa eliminar cada una de las listas que forman los distintos campos, para lo cual se utilizan las funciones `RESET_INT_LIST` y `RESET_STRING_LIST`.

- `new_column_token`. Función que crea un nuevo elemento al comienzo de cada una de las listas almacenadas en una variable de tipo `ice_lex_token`, estableciendo su valor a `No-Aplicable`. Esto se corresponde con la inicialización de un nuevo tipo de componente léxico para una palabra[29]. Nótese la diferencia con la función precedente, que realizaba la incialización al comienzo del reconocimiento de una palabra.

- `copy_column_token`. Esta función actúa como la precedente, pero en vez de rellenar el valor de los nuevos elemntos de las listas a `No-Aplicable`, copia en ellos el contenido del elemento siguiente en la lista[30].

- `rm_column_token`. Elimina el primer elemento de todas las listas. Esto equivale a eliminar el último componente léxico detectado en el reconocimiento de una palabra determinada.

A continuación se muestran las funciones encargadas de la actualización de los valores almacenados en las listas de los diferentes campos incluidos en la variable `token`. Cada función recibe al menos como argumento un puntero a una estructura `ice_lex_token`.

- `set_wrd_token`. Recibe como argumento adicional un puntero a `char`, es decir, una cadena de caracteres. Copia dicha cadena en el campo `word` de la estructura `ice_lex_token`. Esto significa que la cadena pasada como argumento no sufre ningún tipo de modificación.

- `set_cat_token`. Recibe como argumento adicional un valor entero que indica la categoría la que pertenece la palabra. Esta función establece el primer elemento de la lista del campo `category` a dicho valor. Para ello utiliza las funciones `NEXTL_INT_LIST` y `CONS_INT_LIST`, con lo cual se machaca el valor `No-Aplicable` introducido en la inicialización del componente léxico llevada a cabo por la función `new_column_token`.

- `set_gen_token`. Recibe como argumento un valor entero que indica el género de la palabra cuando se reconoce como la categoría actual. El proceso es idéntico al que se realiza en la función anterior, salvo que la información que se actualiza es la de la lista almacenada en el campo `gender`.

- `set_num_token`. Esta función es como la precedente, excepto que el argumento que recibe es un entero que indica el número, por lo cual la información que se actualiza es la correspondiente a la de la lista almacenada en el campo `number`.

- `set_per_token`. Igual que la anterior, pero referida a la persona verbal.

- `set_vtn_token`. Actualiza el valor del tiempo verbal, modificando el primer elemento de la lista del campo `verbal_tense`.

Una función adicional que ha sido preciso definir es `concat`. Su misión es la de realizar la concatenación de dos cadenas de caracteres que se pasan como argumentos. A diferencia de la función estándar `strcat`, la cual concatena físicamente la segunda cadena al final de la primera, esta función crea una nueva cadena de caracteres, reservando el espacio de memoria necesario, en el que copia secuencialmente los caracteres de ambas cadenas. El valor retornado es un puntero al primer carácter de esta nueva cadena de caracteres.

4.4 Proceso de actualización de la información del componente léxico

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

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

- Que el valor devuelto sea 0, en cuyo caso indica que se ha alcanzado el final del fichero sobre el cual se está realizando el reconocimiento de los componentes léxicos.

---

[29] Al tratarse de analizadores léxicos no deterministas, una misma palabra puede corresponderse con varios tipos de componente léxico distintos.

[30] Es decir, el que ocupaba anteriormente la cabeza de la lista.

- Cualquier otro valor indica el reconocimiento de alguna palabra. Por defecto el valor devuelto se establece a 1, aunque su valor concreto es irrelevante, basta con que sea distinto de 0. El reconocimiento de una palabra puede tener tres orígenes diferentes:
  - La detección de una correspondencia de la palabra con alguno de los tipos de componentes léxicos declarados en el analizador sintáctico.
  - El reconocimiento como una palabra errónea, esto es, una palabra que está mal construida según las reglas léxicas[31].
  - Una palabra desconocida.

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

4.5 Actualización mediante reglas léxicas

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

4.5.1 El reconocimiento de los lexemas

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

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

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

4.5.2 El reconocimiento de los sufijos

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

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

4.6 Actualización no determinista

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

[31] Esto último se debe generalmente a errores tipográficos o del reconocedor óptico de caracteres en caso de que haya sido escaneada.

4.6.1 La condición de salida

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

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

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

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

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

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

4.6.2 La condición de error

Esta condición se alcanza cuando una palabra ha comenzado a ser reconocida aplicando algunas reglas léxicas, pero se ha detectado que no verifica ninguna de las alternativas para la continuación del proceso de reconocimiento.. En tal caso, en las acciones correspondientes a la regla incluida en esta condición de arranque se llama a la función `rm_column_token` para eliminar la información del análisis erróneo de la variable `token`.

En caso de que se desee realizar una depuración del analizador léxico, podría ser interesante mantener tal información, pero sumando al campo categoría un valor `ERR`[32] que indique que dicha información es errónea.

5 Arquitectura del etiquetador

A continuación mostramos la estrutura formal de un etiquetador de palabras en español construido según las técnicas y métodos mostrados en los capítulos precedentes. La primera tarea a realizar consiste en la determinación de todas las posibles categorías en que se pueden etiquetar las palabras reconocidas. En la tabla 4 se muestra una lista que contiene aquellas que han sido utilizadas en el sistema aquí presentado.

El siguiente paso consiste en la definición de los miniautómatas que realizarán el reconocimiento de los morfemas. Mediante está técnica se incorpora el conocimiento lingüistico relativo a la construcción del léxico español.

5.1 Determinación de género y número

Dentro de la categoría gramatical de género se distinguen las siguientes denominaciones clasificadoras de sustantivos, determinantes, pronombres y adjetivos:

M	=	masculine
F	=	feminine
B	=	both
N	=	neutral

[32]Este valor `ERR` deberá ser igual a 1 más el mayor valor numérico utilizado para designar una categoría. Con ello, el analizador sintáctico, o el programa especial de depuración del analizador léxico, puede determinar que el campo `category` no contiene un valor válido. Restando `ERR` de su valor se obtendrá la categoría fallida que se intentó reconocer.

En la categoría gramatical de número se distinguen las siguientes denominaciones clasificadoras de sustantivos, determinantes, pronombres, adjetivos y verbos:

S = singular
P = plural
A = all

En las tablas 3 y 5 del apéndice B se muestran las agrupaciones más adecuadas de las clases de palabras afectadas por las categorías de género y número, respectivamente, así como las reglas que controlan el comportamiento formal de las mismas.

Los únicos movimientos vivos de formación del femenino en el español actual son los correspondientes a G1, G2 y G3, cuya rentabilidad está plenamente justificada. Los demás emplean sufijos que los convierten en palabras distintas[33]. El reconocimiento de una raíz común a ambas formas del masculino y del femenino es, desde el punto de vista lingüístico cuando menos discutible, y por otra parte, su rentabilidad es limitada[34]. De todos modos, tratando de factorizar al máximo, el mantenimineto de tales grupos es recomendable. En todo caso, si el examen de los textos demostrara que su rentabilidad, por la escasez de palabras que entran dentro de ellos y la baja frecuencia de ocurrencia de éstas, es despreciable, pueden ser eliminados a posteriori y sus componentes remitidos a raices distintas.

En lo que respecta al número, los grupos productivos son el N1 y el N2. Los tres siguientes son variantes ortográficas del segundo y los cuatro últimos responden al intento de captar la forma más habitualmente reconocida para expresar el plural de palabras cultas heredadas del griego o del latín y de palabras extranjeras incorporadas al idioma.

Anteriormente se había considerado un grupo que daba cabida a las formas *un/(uno)/una/unos/unas* ya que, presentando la misma nación de género que G3, exigía clasificación aparte porque en masculino no presentaba después la misma moción de número, en *-es*, sino que formaba el plural añadiendo *-os*, lo que implicaba la existencia de un grupo también exclusivo en la categoría de número. Considerando que, además, cada uno de los elementos de la serie exige ser clasificado categóricamente de modo diferente[35], no es rentable su factorización formal, de la que, de todos modos, la forma *uno* quedaba fuera.

Las palabras que presentan variación acentual en la raíz serán tratadas mediante raíces distintas a las que se asignará ya en este nivel el género o número según corresponda.

5.2 Análisis de formas verbales

En el apéndice B se encuentran las tablas con las reglas de formación verbal. Las claves que se van a utilizar para identificar las diferentes categorías verbales se muestran en la tabla 1.

En la tabla 6 se muestran las reglas utilizadas para reconocer los verbos de la primera conjugación. Las tablas 7 y 8 se refieren a la segunda y tercera conjugación, respectivamente. La aparición de *sep* se refiere a un separador delimitador de palabras. Las tablas restantes se refieren a los diferentes grupos de verbos irregulares, excepto la 43 y la 44, que contienen las reglas de las desinencias verbales.

6 Conclusiones

Es posible incorporar nuevas capacidades a los generadores de reconocedores léxicos clásicos con el fin de adaptarlos al tratamiento de formalismos más complejos que el representado por los lenguajes algebraicos. El enfoque utilizado es conceptualmente simple, pero demuestra una gran potencia en las realizaciones prácticas.

Concretamente, hemos desarrollado un prototipo de generador de etiquetadores para lenguas naturales, que ha probado su eficiencia tanto a nivel del útil de generación como del analizador generado por el mismo. Con el objeto de acelerar la obtención de los primeros resultados prácticos, hemos decidido desarrollar nuestro prototipo inicial a partir de un útil de generación de reconocedores de expresiones regulares ya existente y de probada eficacia, aunque del todo inadaptado al problema genérico de la etiquetación. En este sentido, se ha procedido a la inclusión de reglas morfológicas específicas, así como de facilidades lingüisticas tales como la lematización, que no han mostrado ningún tipo de problemática en cuanto a su implementación. Finalmente, la herramienta presentada incluye de forma natural el no determinismo en el proceso de análisis, característica inusual en los lenguajes de programación y que en el caso de los naturales ha sido abordada de forma tradicional

[33] En realidad heterónimos no del todo distinguibles del procedimiento de formación del femenino en, por ejmplo, el par *toro/vaca*.
[34] En algunos casos dudamos de que vaya más allá de los ejemplo aquí recogidos.
[35] *un* como DAI, *uno* como RNI y RII, *una* como DAI, DNI, RNI y RII, *unos* como DAI o DII y RII, al igual que *unas*.

Tabla 1: Claves utilizadas en las categorías verbales.

	Clave	Descripción
Persona	1	primera persona
	2	segunda persona
	3	tercera persona
Número	S	singular
	P	plural
Tiempo	Prs	presente
	Imp	imperfecto
	Prf	pret. perfecto simple
	Fut	futuro
	Cnd	condicional
	Inf	infinitivo
	Ger	gerundio
	Ppio	participio
Modo	Ind	indicativo
	Sbj	subjuntivo
	Imp	imperativo

mediante la utilización de bases de datos. En el presente trabajo, la novedad estriba en que el tratamiento de esta funcionalidad se reserva a un nucleo operativo estructurado entorno a un autómata.

El util definitivo se ha integrado en el generador de analizadores sintácticos ICE [10, 11, 12, 13]. Finalmente, el desarrollo del módulo de interfaz [2] con el usuario ha merecido especial atención, cuidando la transparencia tanto del proceso de generación del etiquetador, como de su utilización y mantenimiento.

Tabla 2: Estadísticas cualitativas de los ejemplos tratados

Categoría	**Nombres**	**Adjetivos**	**Artículos**	**Pronombres**
Porcentajes	20.80%	14.67%	12.27%	2.67%

Categoría	**Verbos 1^o conj.**	**Verbos 2^o conj.**	**Verbos 3^o conj.**
Porcentajes	6.93%	2.93%	0.53%

Categoría	**Advervios**	**Preposiciones**	**Conjunciones**	**Signos de puntuación**
Porcentajes	3.73%	15.20%	7.73%	12.27%

n

Referencias

[1] **Aho, A. V.; Sethi, R. y Ullman, J. D.** *Compiladores. Principios, Técnicas y Herramientas.* 1990. Addison-Wesley Iberoamericana S.A., Wilmington, Delaware, U.S.A.

[2] **Alonso Pardo, M. A.** *Edición interactiva en entornos incrementales.* 1994. Proyecto fin de licenciatura. Facultad de Informática. Universidad de La Coruña, La Coruña, España.

[3] **Donnelly, C. y Stallman, R.** *BISON. The YACC-compatible Parser Generator.* 1988. Free Software Foundation, Cambridge, Massachussets, U.S.A.

[4] *LE-LISP Version 15.25 Reference Manual.* 1991. INRIA Rocquencourt, Le Chesnay, Francia.

[5] **Johnson, S.C.** *YACC: Yet Another Compiler Compiler.* 1975. Computer Sciences Technical Report n. 32, AT&T Bell Laboratories, Murray Hill, New Jersey, U.S.A.

[6] **Paxon, V.** *Flex 2.4.6 (January 1994). Reference Documentation, Full User Documentation & Changes between Releases.* 1994. Lawrence Berkeley Laboratory, University of California. Berkeley, California, U.S.A.

[7] **Pérez, R, Trotzig, D y Lloré, X.** *MORFEO: Analizador Morfológico y Tagger delEspañol.* 1994. Actas del X Congreso de la SEPLN, Córdoba, España.

[8] **Rodrigo Mateos, J. L.** *Recherche et Extraction Automatique de Lemmes.* 1994. Boletín de la SEPLN, n^o14 (pp. 149-164).

[9] **Grishman, R.** *Computational linguistics.* 1986. Cambridge University Press, Cambridge, Reino Unido.

[10] **Vilares Ferro, M.** *Efficient Incremental Parsing for Context-Free Languages.* 1992. Tesis Doctoral, Université de Nice, Niza, Francia.

[11] **Vilares Ferro, M.** *An Efficient Context Free Backbone for Natural Languages Analyzers.* 1993. Proc. of SEPLN'93, Santiago de Compostela, La Coruña, España.

[12] **Vilares Ferro, M. y Dion, B.** *Efficient Incremental Parsing for Context-Free Languages.* 1994. Proc. of the Fifhty IEEE International Conference on Computer Languages, Toulouse, Francia.

[13] **Vilares Ferro, M. y Graña Gil, J.** *Parsing as Resolution.* 1993. Proc. of the First Compulog Network Meeting of Parallelism and Implementation Technologies, Madrid, España.

A Resultados prácticos

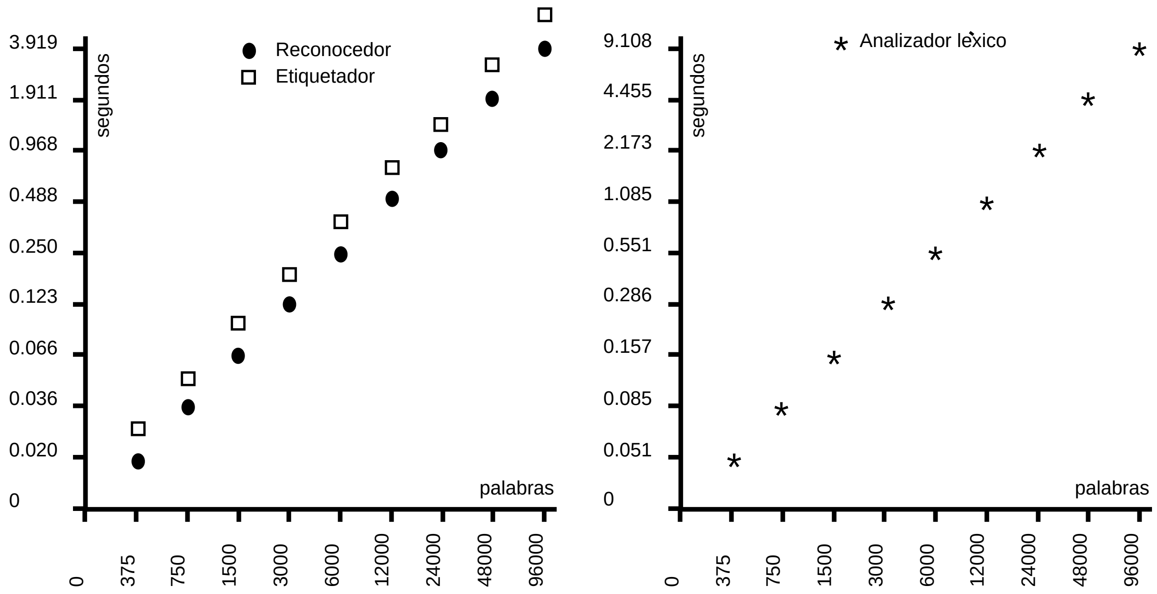
Para mostrar el comportamiento de nuestro prototipo hemos diseñado un analizador léxico para un subconjunto significativo del español. En él incluímos etiquetas para el conjunto de las categorías léxicas además de esquemas de reconocimiento para la totalidad de los tiempos simples de los verbos regulares de las tres conjugaciones y para la mayoría de las reglas de formación del género y del número en dichas categorías. En relación a la naturaleza de los tests aplicados, el lector debe tener en cuenta que:

1. Se ha considerado un amplio espectro de criterios. La figura 1 muestra los tiempos medios de análisis del reconocedor y del analizador completo[36] sobre un conjunto de textos de entrada de longitud creciente, y cuyos porcentajes de distribución por categorías léxicas se muestran en la tabla 2.

2. Con el objeto de hacer una estimación apropiada del comportamiento temporal de nuestro prototipo, los análisis mostrados en la figura 1 se han realizado sobre textos en los que los porcentajes de las categorías léxicas son idénticos, como quedaba de manifiesto en la tabla 2. Para ello, se ha partido de un texto base de 375 palabras que se ha concatenado sucesivamente hasta obtener, en los tests mostrados, un texto de un total de 96000 palabras.

[36] esto es, el reconocedor más el etiquetador.

Figure 1: Gráfica *tiempo-espacio* en los ejemplos considerados



3. Todos los tests han sido realizados separando el tiempo de reconocimiento de los patrones léxicos, del invertido en las acciones de etiquetación.

4. El tiempo invertido en la escritura de la información resultante del etiquetado en soporte físico[37], no ha sido tenido en cuenta en los tests.

5. Con el objeto de hacer una estimación apropiada del comportamiento temporal de nuestro prototipo, los análisis mostrados en la figura 1 se han realizado sobre textos en los que los porcentajes de las categorías léxicas son idénticos, como quedaba de manifiesto en la tabla 2. Para ello, se ha partido de un texto base de 375 palabras que se ha concatenado sucesivamente hasta obtener, en los tests mostrados, un texto de un total de 96000 palabras.

6. Todos los tests han sido realizados separando el tiempo de reconocimiento de los patrones léxicos, del invertido en las acciones de etiquetación.

7. El tiempo invertido en la escritura de la información resultante del etiquetado en soporte físico[38], no ha sido tenido en cuenta en los tests.

8. Con el objeto de evitar errores en el cronómetro UNIX, los tiempos finales considerados son el resultado de establecer la media de diez ejecuciones sobre cada ejemplo.

9. Todas las estimaciones temporales se han realizado sobre una plataforma *Sun SPARCstation 2*, con carga débil de trabajo.

En relación a la eficiencia en la gestión de la memoria del algoritmo operacional considerado, baste resaltar que el número de estados del prototipo es de 1037 con 5471 transiciones, lo que permite el tratamiento de 2066 patrones diferentes. Sin embargo, esta cifra no da sino una idea remota de las posibilidades del útil. En efecto, en este punto es interesante mostrar el comportamiento de nuestro analizador frente al problema de la introducción de nuevos patrones. Para ilustrar dicha cuestión indicaremos el comportamiento observado frente a la incorporación de un nuevo verbo, incluyendo la conjugación de todos sus tiempos simples. Así, el número total de estados y de transiciones se incrementa en tan solo dos unidades.

[37] esto es, sobre un fichero.
[38] esto es, sobre un fichero.

B Tablas de categorías y reglas morfológicas

Este apéndice muestra de forma detallada, a la vez que exhaustiva, el conjunto completo de categorias en las que los autores del presente trabajo han clasificado a la práctica totalidad de los vocablos de la lengua española. Puesto que se trata de un material que sirve de base al diseño del nucleo operativo del sistema presentado, hemos decidido incluirlo como apéndice. De esta forma pretendemos centrar la orientación práctica de nuestro trabajo hacia el diseño computacional de la herramienta, más que hacia el tratamiento de aspectos fundamentalmente lingüisticos, como es el caso actual.

Tabla 3: Grupos y reglas gramaticales de formación del género

Grupo	Marca	Acción	Ejemplo
G1	o	masculino y saltar1 a N1	am-o
	a	femenino y saltar a N1	am-a
G11	o	neutro y saltar a N1	est-o, es-o, aquell-o, l-o
	a	femenino y saltar a N1	est-a, es-a, aquell-a, l-a
G2	e	masculino y saltar a N1	infant-e, sastr-e
	a	femenino y saltar a N1	infant-a, sastr-a
G3	ε	masculino y saltar a N2	escritor, autor
	a	femenino y saltar a N1	escritor-a, autor-a
G4	o	masculino y saltar a N1	diabl-o, vampir-o
	esa	femenino y saltar a N1	diabl-esa, vampir-esa
G5	o	masculino y saltar a N1	gall-o
	ina	femenino y saltar a N1	gall-ina
G6	a	masculino y saltar a N1	guard-a
	esa	femenino y saltar a N1	guard-esa
G7	a	masculino y saltar a N1	pap-a, poet-a, profet-a
	isa	femenino y saltar a N1	pap-isa, poet-isa, prefet-isa
G8	que	masculino y saltar a N1	caci-que
	ca	femenino y saltar a N1	caci-ca
G9	e	masculino y saltar a N1	duqu-e
	esa	femenino y saltar a N1	duq-esa
G10	e	masculino y saltar a N1	sacerdot-e
	isa	femenino y saltar a N1	sacerdot-isa
G11	ε	masculino y saltar a N2	doncel
	la	femenino y saltar a N1	doncel-la
G12		masculino y saltar a N2	abad, juglar
	esa	femenino y saltar a N1	anad-esa, juglar-esa
G13	ε	masculino y saltar a N2	zar
	ina	femenino y saltar a N1	zar-ina
G14	y	masculino y saltar a N2	re-y
	ina	femenino y saltar a N1	re-ina
G15	or	masculino y saltar a N2	act-or
	ri	feemnino y saltar a N3	act-ri(z)
G16	dor	masculino y saltar N2	empera-dor
	tri	femenino y saltar a N3	emperat-ri(z)
G17	f	masculino y saltar a N2	jabalí
	ina	femenino y saltar a N1	jabal-ina

Tabla 4: Etiquetas y denominaciones de las categorías consideradas.

Etiqueta	Denominación	Etiqueta	Denominación
S	sustantive	DPD	definite possessive determiner
A	adjective	DDD	definite demonstrative determiner
P	preposition	DCI	indefinite quantifier determiner
C	conjunction	DLI	indefinite other determiner
V	verb	DXI	indefinite reflexive determiner
RPT	tonic personal pronoum	DOI	indefinite order determiner
RPA	Atom personal pronoum	DNI	indefinite cardinar determinar
RZD	definite totalizer pronoum	DFI	indefinite partitive determiner
RDD	definite demonstrative pronoum	DII	indefinite indefinite determiner
RCI	indefinite quantifier pronoum	DTI	indefinite distributive determiner
RLI	indefinite other pronoum	DMI	indefinite comparative determiner
RNI	indefinite cardinal pronoum	WN	nueclear adverb
RFI	indefinite partitive pronoum	WM	modifier adverb
RII	indefinite indefinite pronoum	WB	both function adverb
RMI	indefinite comparative pronoum	PM	punctuation mark
DZD	definitive totalizer determiner	UKN	unknown word
DAD	definite article determiner	NA	no applicable
DAI	indefinite article determiner		

Tabla 5: Grupos y reglas gramaticales de formación del número

Grupo	term	Descripción	Ejemplo
N1	s ε	plural singular y salto a AV1	perro-s, ama-s, casa-s perro, ama,casa
N2	es ε	plural singular y salto a AV1	mujer-es, árboles mujer, árbol
N3	ces z	plural singular y salto a AV1 si es adjetivo feminine o both	co-ces, actri-ces co-z,actri-z
AV1	mente	cambio a categoría WN. Cambio de número, género y lema	
N4	ques c	plural singular	frac-ques fra-c
N5	ues ε	plural singular	zigzag-ues zigzag
N6	s n	plural singular	hipérbato-s hipérbato-n
N7	os um	plural singular	currículos currícul-um
N8	es ε	plural singular	lor-es, zin-es lor-s, zin-c
N9	is y	plural singular	royalt-is royalt-y

Tabla 6: Grupos y reglas para la primera conjugación verbal.

Grupo	Marca	Acción
V1	o	1 S Prs Ind
V1	a	saltar a V11
V1	é	saltar a V12
V1	ó	3 S Prf Ind
V1	e	Prs Sbj y saltar a VD6
V1	á	Saltar a V13
V11	r	Inf
V11	ba	Imp Ind y saltar a VD1
V11	ría	Cnd Ind y saltar a VD2
V11	ré	Fut Ind y saltar a VD3
V11	rá	Fut Ind y saltar a VD4
V11	re	saltar a V111
V11	ra	Imp Sbj y saltar a VD1
V11	se	Imp Sbj y saltar a VD1
V11	ndo	Ger
V11	d	saltar a V112
V11	ε	saltar a VD5

Grupo	Marca	Acción
V12	is	2 P Prs Sbj
V12	ε	1 S Prf Ind
V13	ba	Imp Ind y saltar a VD8
V13	ra	Imp Sbj y saltar a VD8
V13	se	Imp Sbj y saltar a VD8
V13	re	Fut Sbj y saltar a VD8
V13	is	2 P Prs Ind
V111	mos	1 P Fut Ind
V111	ε	Fut Sbj y saltar a VD1
V112	*sep*	2 p Imp
V112	ε	Ppio y saltar a G1

Tabla 7: Grupos y reglas para la segunda conjugación verbal.

Grupo	Marca	Acción
V2	o	1 S Prs Ind
V2	é	Prs Ind y saltar a VD7
V2	e	saltar a V21
V2	ía	Imp Ind y saltar a VD2
V2	í	1 S Prf Ind
V2	ie	saltar a V23
V2	i	saltar a V22
V2	iéra	Imp Sbj y saltar a VD8
V2	iése	Imp Sbj y saltar a VD8
V2	a	Prs Sbj y saltar a VD6
V2	á	Prs Sbj y saltar a VD7
V2	iére	Fut Sbj y saltar a VD8

Grupo	Marca	Acción
V21	r	Inf
V21	ría	Cnd Ind y saltar a VD2
V21	ré	Fut Ind y saltar VD3
V21	rá	Fut Ind y saltar a VD4
V21	re	Fut Ind y saltar a VD8
V21	d	2 P Imp
V21	ε	Prs Ind y saltar a VD10
V22	d	Ppio y saltar a G1
V22	ε	Prf Ind y saltar a VD11
V23	ra	Imp Sbj y saltar a VD1
V23	se	Imp Sbj y saltar a VD1
V23	ndo	Ger
V23	re	fut Sbj y saltar a VD1
V23	ε	Prf Ind y saltar a VD12

Tabla 8: Grupos y reglas para la tercera conjugación verbal.

Grupo	Marca	Acción
V3	o	1 S Prs Ind
V3	e	Prs Ind y saltar a VD13
V3	ía	Imp Ind y saltar a VD2
V3	í	Saltar a V32
V3	ie	saltar a V23
V3	i	saltar a V31
V3	a	Prs Sbj y saltar a VD6
V3	iéra	Imp Sbj y saltar a VD6
V3	iése	Imp Sbj y saltar a VD7
V3	á	Prs Sbj y saltar a VD7
V3	iére	Fut Sbj y saltar a VD8

Grupo	Marca	Acción
V31	r	Inf
V31	ría	Cnd Inf y saltar a VD2
V31	ré	Fut Ind y saltar a VD3
V31	rá	Fut Ind y saltar a VD4
V31	re	Fut INd y saltar a VD8
V31	mos	1 P Prs Prf Ind
V31	d	saltar a V112
V31	ε	Prf Ind y saltar a VD9
V32	S	2 P Prs Ind
V32	ε	1 S Prf Ind

Tabla 9: Grupos y reglas para el grupo 4 de verbos irregulares.

Grupo	Marca	Acción
Vi4A	o	1 S Prs Ind
Vi4A	a	Prs Ind y saltar a VD13
Vi4A	e	Prs Sbj y saltar a VD14
Vi4B	a	saltar a Vi4B1
Vi4B	é	saltar a V12
Vi4B	ó	3 S Prf Ind
Vi4B	e	Prs Sbj y saltar a VD8
Vi4B	a	saltar a V13

Grupo	Marca	Acción
Vi4B1	r	Inf
Vi4Bl	ba	Imp Ind y saltar a VD1
Vi4B1	ría	Cnd Ind y saltar a VD2
Vi4B1	ré	Fut Ind y saltar a VD3
Vi4B1	rá	Fut Ind y saltar a VD4
Vi4B1	re	saltar a V111
Vi4B1	ra	Imp Sbj y saltar a VD1
Vi4B1	se	Imp Sbj y saltar a VD1
Vi4B1	ndo	Ger
Vi4B1	d	saltar a V112
Vi4B1	ε	Prf Ind y saltar a VD15

Tabla 10: Grupos y reglas para el grupo 5 de verbos irregulares.

Grupo	Marca	Acción
Vi5A	o	1 S Prs Ind
Vi5A	a	Prs Ind y saltar a VD13
Vi5B	a	saltar a Vi4B1
Vi5B	ó	S Prf Ind
Vi5B	á	saltar a V13

Grupo	Marca	Acción
Vi5C	e	Prs Sbj y saltar a VD8
Vi5C	é	saltar a V12
Vi5D	e	Prs Sbj y saltar a VD14

Tabla 11: Grupos y reglas para el grupo 6 de verbos irregulares.

Grupo	Marca	Acción
Vi6A	o	1 S Prs Ind
Vi6A	a	saltar a V11
Vi6A	ó	3 S Prf Ind
Vi6A	á	saltar a V13

Grupo	Marca	Acción
Vi6B	é	saltar a V12
Vi6B	e	Prs Sbj y saltar a VD6

Tabla 12: Grupos y reglas para el grupo 7 de verbos irregulares.

Grupo	Marca	Acción
Vi7A	o	1 S Prs Ind
Vi7A	a	saltar a Vi7A1
Vi7A	e	Prs Sbj y saltar a VD6
Vi7A	é	Prs Sbj y saltar a VD7
Vi7A	á	saltar a Vi7A2
Vi7A1	r	Inf
Vi7A1	ba	Imp Ind y saltar a VD1
Vi7A1	ría	Cnd Ind y saltar a VD2
Vi7A1	ré	Fut Ind y saltar a VD8
Vi7A1	rá	Fut Ind y saltar a VD4
Vi7A1	re	Put Ind y saltar a VD8
Vi7A1	ndo	Ger
Vi7A1	d	saltar a V112
Vi7A1	ε	saltar a VD10

Grupo	Marca	Acción
Vi7A2	ba	Imp Ind y saltar a VD8
Vi7A2	is	2 P Prs Ind
Vi7B	ie	saltar a Vi7B1
Vi7B	iéra	Imp Sbj y saltar a VD8
Vi7B	iése	Imp Sbj y saltar a VD8
Vi7B	iere	Fut Sbj y saltar a VD8
Vi7B	ε	Prf Ind y saltar
Vi7B1	ra	Imp Sbj y saltar a VD1
Vi7B1	se	Imp Sbj y saltar a VD1
Vi7B1	re	Put Sbj y saltar a VD1
Vi7B1	ε	Prf Ind y saltar a VD12

Tabla 13: Grupos y reglas para el grupo 8 de verbos irregulares.

Grupo	Marca	Acción
Vi8A	o	1 S Prs Ind
Vi8A	e	Prs Ind y saltar a VD13
Vi8A	a	Prs Sbj y sa*tar a VD14
Vi8B	é	Prs Ind y saltar a VD7
Vi8B	e	saltar a Vi8B1
Vi8B	ía	Imp Ind y saltar a VD2
Vi8B	í	1 S Prf Ind
Vi8B	ie	saltar a V23
Vi8B	i	saltar a V22
Vi8B	iéra	Imp Sbj y saltar a VD8
Vi8B	iése	Imp Sbj y saltar a VD8
Vi8B	iére	Fut Sbj y saltar a VD8
Vi8B	á	Prs Sbj y saltar a VD7
Vi8B	a	Prs Sbj y saltar a VD8

Grupo	Marca	Acción
Vi8B1	r	Inf
Vi8B1	ria	Cnd Ind y saltar a VD2
Vi8B1	ré	Fut Ind y saltar a VD3
Vi8B1	rá	Fut Ind y saltar a VD4
Vi8B1	re	Fut Ind y saltar a VD8
Vi8B1	d	2 P Imp
Vi8B1	ε	Prs Ind y saltar a VD8

Tabla 14: Grupos y reglas para el grupo 9 de verbos irregulares.

Grupo	Marca	Acción
Vi9B	é	Prs Ind y saltar a VD7
Vi9B	e	saltar a Vi8B1
Vi9B	ía	Imp Ind y saltar a VD2
Vi9B	i	1 S Perf Ind
Vi9B	ie	saltar a V23
Vi9B	i	Prf Ind y saltar a VD11
Vi9B	iéra	Imp Sbj y saltar a VD8
Vi9B	iése	Imp Sbj y saltar a VD8
Vi9B	iére	Fut Sbj y saltar a VD8
Vi9B	á	Prs Sbj y saltar a VD7
Vi9B	a	Prs Sbj y saltar a VD8

Tabla 15: Grupos y reglas para el grupo 10 de verbos irregulares.

Grupo	Marca	Acción
Vi10A	o	1 S Prs Ind
Vi10A	a	Prs Sbj y saltar a VD6
Vi10A	á	Prs Sbj y saltar a VD7
Vi10B	é	Prs Ind y saltar a VD7
Vi10B	e	saltar a V21
Vi10B	ía	Imp Ind y saltar a VD2
Vi10B	í	1 S Prf Ind
Vi10B	ie	saltar a V23
Vi10B	i	saltar a V22
Vi10B	iéra	Imp Sbj y saltar a VD8
Vi10B	iése	Imp Sbj y saltar a VD8
Vi10B	iére	Fut Sbj y saltar a VD8

Tabla 16: Grupos y reglas para el grupo 11 de verbos irregulares

Grupo	Marca	Acción
Vi11A	e	saltar a Vi11A1
Vi11A	é	Prs Ind y saltar a VD7
Vi11A	ía	Imp Ind y saltar a VD2
Vi11A	ie	saltar a VD17
Vi11A	id	Ppio y saltar a G1
Vi11A	ε	saltar a Vi11A2

Grupo	Marca	Acción
Vi11A1	d	2 P Imp
Vi11A1	r	Inf
Vi11A1	ε	Prs Ind saltar a VD10
Vi11A2	ría	Cnd Ind y saltar a VD2
Vi11A2	ré	Fut Ind y saltar a VD3
Vi11A2	ré	Fut Ind y saltar a VD4
Vi11A2	re	Fut Ind y saltar a VD8

Tabla 17: Grupos y reglas para el grupo 12 de verbos irregulares

Grupo	Marca	Acción
Vi12B	é	Prs Ind y saltar a VD7
Vi12B	e	saltar a V21
Vi12B	ía	Imp Ind y saltar a VD2
Vi12B	í	saltar a VD18

Grupo	Marca	Acción
Vi12C	ó	3 S Prf Ind
Vi12C	e	saltar a V23
Vi12C	éra	Imp Sbj y saltar a VD8
Vi12C	ése	Imp Sbj y saltar a VD8
Vi12C	ére	Fut Sbj y saltar a VD8

Tabla 18: Grupos y reglas para el grupo 13 de verbos irregulares

Grupo	Marca	Acción
Vi13B	6	Prs Ind y saltar a VD7
Vi13B	e	sa1tar a Vi13B1
Vil3B	ia	Imp Ind y saltar a VD2
Vi13B	a	Prs Sbj y saltar a VD8
Vi13B	á	Prs Sbj y saltar a VD7
Vi13B	id	Ppio y saltar a G1
Vi13B	ε	saltar a Vi11A2

Tabla 19: Grupos y reglas para el grupo 14 de verbos irregulares

Grupo	Marca	Acción
Vi14A	o	1 S Prs Ind
Vi14A	é	Prs Ind y saltar a VD7
Vi14A	e	saltar a V21
Vi14A	ía	Imp Ind y saltar a VD2
Vi14A	í	saltar a VD18
Vi14A	a	Prs Ind y saltar a VD6
Vi14A	á	Prs Sbj y saltar a VD7

Tabla 20: Grupos y reglas para el grupo 15 de verbos irregulares

Grupo	Marca	Acción
Vi15C	é	Prs Ind y saltar a VD7
Vi15C	e	saltar a Vi8B1
Vi15C	ía	Imp Ind y saltar a VD2
Vi15C	í	1 S Prf Ind
Vi15C	ie	saltar a V23
Vi15C	i	saltar a V22
Vi15C	iéra	Imp Sbj y saltar a VD8
Vi15C	iése	Imp Sbj y saltar a VD8
Vi15C	iére	Fut Sbj y saltar a VD8
Vi15D	a	Prs Sbj y saltar a VD8
Vi15D	á	Prs Sbj y saltar a VD7

Tabla 21: Grupos y reglas para el grupo 16 de verbos irregulares

Grupo	Marca	Acción
Vi16B	é	Prs Ind y saltar a VD7
Vi16B	e	Prs Ind y saltar a VD19
Vi16B	ía	Imp Ind y saltar a VD2
Vi16B	ie	saltar a VD17.
Vi16B	ε	2 S Imp

Tabla 22: Grupos y reglas para el grupo 17 de verbos irregulares

Grupo	Marca	Acción
Vi17B	e	Prs Ind y saltar a VD4
Vi17C	e	saltar a Vi13B1
Vi17C	é	Prs Ind y saltar a VD7
Vi17C	éa	Imp Ind y saltar a VD2
Vi17C	ie	saltar a VD17
Vi17C	id	Ppio y saltar a G1
Vi17C	ε	2 S Imp

Tabla 23: Grupos y reglas para el grupo 18 de verbos irregulares

Grupo	Marca	Acción
Vi18B	e	saltar a VD19
Vi18B	é	Prs Ind y saltar a VD7
Vi18B	ía	Imp Ind y saltar a VD2
Vi18B	ie	saltar a VD17

Grupo	Marca	Acción
Vi18C	e	1 S Prf Ind
Vi18C	ie	saltar a Vi7B1
Vi18C	i	Prf y saltar a VD20
Vi18C	iéra	Imp Sbj y saltar a VD8
Vi18C	iése	Imp Sbj y saltar a VD8
Vi18C	iére	Fut Sbj y saltar a VD8

Tabla 24: Grupos y reglas para el grupo 19 de verbos irregulares

Grupo	Marca	Acción
Vi19B	e	saltar a Vi11A1
Vi19B	é	Prs Ind y saltar a VD7
Vi19B	ía	Imp Ind y saltar a VD2
Vi19B	í	1 S Prf Ind
Vi19B	i	saltar a V22
Vi19B	ie	saltar a V23
Vi19B	iéra	Imp Sbj y saltar a VD8
Vi19B	iése	Imp Sbj y saltar a VD8
Vi19B	iére	Fut Sbj y saltar a VD8
Vi19B	ε	2 S Imp

Tabla 25: Grupos y reglas para el grupo 20 de verbos irregulares

Grupo	Marca	Acción
Vi20A	o	1 S Prs Ind
Vi20A	ía	Imp Ind y saltar a VD2
Vi20A	a	Prs Sbj y saltar a VD6
Vi20A	é	Prs Sbj y saltar a VD7
Vi20A	ε	saltar a Vi11A2

Grupo	Marca	Acción
Vi20B	e	saltar a Vi11A1
Vi20B	é	Prs Ind y saltar a VD7
Vi20B	i	Prf Ind y saltar a VD2I
Vi20B	ie	saltar a V23
Vi20B	iéra	Imp Sbj y saltar a VD8
Vi20B	iése	Imp Sbj y saltar a VDS
Vi20B	iére	Fut Sbj y saltar a VD8

Tabla 26: Grupos y reglas para el grupo 21 de verbos irregulares

Grupo	Marca	Acción
Vi21B	e	saltar a V21
Vi21B	é	Prs Ind y saltar a VD7
Vi21B	ía	Imp Ind y saltar a VD2
Vi21B	id	Ppio y saltar a G1

Grupo	Marca	Acción
Vi21C	era	Imp Sbj y saltar a VD1
Vi21C	ese	Imp Sbj y saltar a VD1
Vi21C	ere	Fut Sbj y saltar a VD1
Vi21C	éra	Imp Sbj y saltar a VD8
Vi21C	ése	Imp Sbj y saltar a VDS
Vi21C	ére	Fut Sbj y saltar a VD8
Vi21C	eron	3 P Prf Ind.

Tabla 27: Grupos y reglas para el grupo 22 de verbos irregulares

Grupo	Marca	Acción
Vi22	o	1 S Prs Ind
Vi22	é	Prs Ind y saltar a VD7
Vi22	ía	Imp Ind y saltar a VD2
Vi22	e	saltar a Vi221
Vi22	a	Prs Sbj y sa*tar a VD6
Vi22	á	Prs Sbj y saltar a VD7
Vi22	id	Ppio y saltar a G1
Vi22	éra	Imp Sbj y saltar a VD8
Vi22	ése	Imp Sbj y saltar a VD8
Vi22	ére	Fut Sbj y saltar a VD8
Vi22	ε	Prf Ind y saltar a VD22

Grupo	Marca	Acción
Vi221	ría	Cnd Ind y saltar a VD2
Vi221	re	Fut Ind y saltar a VD3
Vi221	ré	Fut Ind y saltar a VD4
Vi221	re	saltar a V111
Vi221	ra	Imp Sbj y saltar a VD1
Vi221	se	Imp Sbj y saltar a VD1
Vi221	ndo	Ger
Vi221	ron	3 P Prf Ind
Vi221	ε	saltar a Vi11A1

Tabla 28: Grupos y reglas para el grupo 23 de verbos irregulares

Grupo	Marca	Acción
Vi23A	o	1 S Prs Ind
Vi23A	e	Prs Ind y saltar a VD13
Vi23A	a	Prs Sbj y saltar a VD6
Vi23A	é	Prs Sbj y saltar a VD7
Vi23A	i	saltar a Vi23A1
Vi23A1	é	3 S Prf Ind
Vi23A1	e	saltar a V23
Vi23A1	éra	Imp Sbj y saltar a VD8
Vi23A1	ese	Imp Sbj y saltar a VD8
Vi23A1	ére	Fut Sbj y saltar a VD8

Grupo	Marca	Acción
Vi23B	ía	Imp Ind y saltar a VD2
Vi23B	i	saltar a Vi23B1
Vi23B	í	saltar a V32
Vi23B1	r	Inf
Vi23B1	ría	Cnd Ind y saltar a VD2
Vi23B1	ré	Put Ind y saltar a VD3
Vi23B1	rá	Fut Ind y saltar a VD4
Vi23B1	re	Fut Ind y saltar a VD8
Vi23B1	d	saltar a V112
Vi23B1	ε	Prf Ind y saltar

Tabla 29: Grupos y reglas para el grupo 24 de verbos irregulares

Grupo	Marca	Acción
Vi24B	e	Prs y saltar a VD13
Vi24B	i	saltar a Vi23A1

Tabla 30: Grupos y reglas para el grupo 25 de verbos irregulares

Grupo	Marca	Acción
Vi25A	o	1 S Prs Ind
Vi25A	e	saltar a Vi25A1
Vi25A	a	Prs Sbj y saltar a VD6
Vi25A	á	Prs Sbj y saltar a VD7
Vi25A	ó	3 S Prf Ind
Vi25A	éra	Imp Sbj y saltar a VD8
Vi25A	ése	Imp Sbj y saltar a VD8
Vi25A	ére	Fut Sbj y saltar a VD8

Grupo	Marca	Acción
Vi25A1	ra	Imp Sbj y saltar a VD1
Vi25A1	se	Imp Sbj y saltar a VD1
Vi25A1	re	Fut Sbj y saltar a VD1
Vi25A1	ron	3 S Prf Ind
Vi25A1	ndo	Ger
Vi25A1	ε	Prs Ind y saltar a VD13

Tabla 31: Grupos y reglas para el grupo 26 de verbos irregulares

Grupo	Marca	Acción
Vi26B	ía	Imp Ind y saltar a VD2
Vi26B	ie	saltar a V23
Vi26B	ió	3 S Prf Ind
Vi26B	i	saltar a Vi23B1
Vi26B	í	saltar a V32
Vi26B	iéra	Imp Sbj y saltar a VD8
Vi26B	iése	Imp Sbj y saltar a VD8
Vi26B	iére	Fut Sbj y saltar a VD8
Vi26B	a	Prs Sbj y saltar a VD8
Vi26B	á	Prs Sbj y saltar a VD7

Tabla 32: Grupos y reglas para el grupo 27 de verbos irregulares

Grupo	Marca	Acción
Vi27C	i	saltar a Vi23A1
Vi27C	a	Prs Sbj y saltar a VD8
Vi27C	á	Prs Sbj y saltar a VD7

Tabla 33: Grupos y reglas para el grupo 28 de verbos irregulares

Grupo	Marca	Acción
Vi28B	ia	Imp Ind y saltar a VD2
Vi28B	i	saltar a Vi25B1
Vi28B	í	saltar a V32

Grupo	Marca	Acción
Vi25B1	ría	Cnd Ind y saltar a VD2
Vi28B1	ré	Fut Ind y saltar a VD3
Vi25B1	rá	Fut Ind y saltar a VD4
Vi28B1	re	Fut Ind y saltar a VD4
Vi28B1	d	2 P Imp
Vi28B1	ε	Prf Ind y saltar a VD23

Tabla 34: Grupos y reglas para el grupo 29 de verbos irregulares

Grupo	Marca	Acción
Vi29B	e	Prs Ind y saltar a VD13
Vi29B	ía	Imp Ind y saltar a VD2
Vi29B	i	saltar a Vi23B1
Vi29B	ió	3 S Prf Ind
Vi29B	í	saltar a v32
Vi29B	ie	saltar a v23
Vi29B	iéra	Imp Sbj y saltar a VD8
Vi29B	iése	Imp Sbj y saltar a VD8
Vi29B	iére	Fut Sbj y saltar a VD8

Tabla 35: Grupos y reglas para el grupo 30 de verbos irregulares

Grupo	Marca	Acción
Vi30B	e	Prs Ind y saltar a VD13
Vi30B	ía	Imp Ind y saltar a VD2
Vi30B	i	saltar a Vi30B1
Vi30B	ie	saltar a VD17
Vi30B	í	Prs Ind y saltar a VD24

Grupo	Marca	Acción
Vi30B1	r	Inf
Vi30B1	ría	Cnd Ind y saltar a VD2
Vi30B1	ré	Fut Ind y saltar a VD3
Vi30B1	rá	Fut Ind y saltar a VD4
Vi30B1	re	Fut Ind y saltar a VD8
Vi30B1	d	saltar a V112
Vi30B1	ε	Prs Ind y saltar a VD8

Tabla 36: Grupos y reglas para el grupo 31 de verbos irregulares

Grupo	Marca	Acción
Vi31B	e	Prs Ind y saltar a VD4
Vi31B	ie	saltar a VD7
Vi31C	ía	Imp Ind y saltar a VD2
Vi31C	i	saltar a Vi13B
Vi31C	í	Prs Ind y saltar a VD24

Grupo	Marca	Acción
Vi31E	*sep*	2 S Imp
Vi31E	ε	saltar a Vi11A2

Tabla 37: Grupos y reglas para el grupo 32 de verbos irregulares

Grupo	Marca	Acción
Vi32B	e	Prs Ind y saltar a VD13
Vi32B	ie	saltar a VD17

Grupo	Marca	Acción
Vi32C	ía	Imp Ind y saltar a VD2
Vi32C	i	saltar a Vi8B1
Vi32C	í	Prs Ind y saltar a VD24

Tabla 38: Grupos y reglas para el grupo 33 de verbos irregulares

Grupo	Marca	Acción
Vi33B	e	Prs Ind y saltar a VD4
Vi33B	ía	Imp Ind y saltar a VD2
Vi33B	ie	saltar a V23
Vi33B	iéra	Imp Sbj y sa*tar a VD8
Vi33B	iése	Imp Sbj y saltar a VD8
Vi33B	iére	Fut Sbj y saltar a VD8
Vi33B	í	saltar a V32
Vi33B	i	saltar a Vi33B1
Vi33B	ε	2 S Imp

Grupo	Marca	Acción
Vi33B1	r	Inf
Vi33B1	d	saltar a V112
Vi33B1	ε	prf Ind y saltar

Tabla 39: Grupos y reglas para el grupo 34 de verbos irregulares

Grupo	Marca	Acción
Vi34C	ía	Imp Ind y saltar a VD2
Vi34C	i	saltar a Vi34C1
Vi34C	í	Prs Ind y saltar a VD24
Vi34C	ε	2 S Imp
Vi34C1	r	Inf
Vi34C1	d	saltar a V112
Vi34C1	mos	1 P Prs Ind

Grupo	Marca	Acción
Vi34D	ie	saltar a V23
Vi34D	iéra	Imp Sbj y saltar a VD8
Vi34D	iése	Imp Sbj y saltar a VD8
Vi34D	iére	Fut Sbj y saltar a VD8
Vi34D	ε	Prf Ind y saltar a VD16

Tabla 40: Grupos y reglas para el grupo 35 de verbos irregulares

Grupo	Marca	Acción
Vi35B	e	saltar a Vi25A1
Vi35B	éra	Imp Sbj y saltar a VD8
Vi35B	ése	Imp Sbj y saltar a VD8
Vi35B	ére	Fut Sbj y saltar a VD8
Vi35B	ó	3 S Prf Ind

Grupo	Marca	Acción
Vi35C	ía	Imp Ind y saltar a VD2
Vi35C	í	saltar a Vi35C1
Vi35C	i	saltar a Vi11A2
Vi35C1	d	saltar a V112
Vi35C1	r	Inf
Vi35C1	ε	saltar a VD26

Tabla 41: Grupos y reglas para el grupo 36 de verbos irregulares

Grupo	Marca	Acción
Vi36	o	1 S Prs Ind
Vi36	e	saltar a Vi25A1
Vi36	éra	Imp Sbj y saltar a VD8
Vi36	ése	Imp Sbj y saltar a VD8
Vi36	ére	Fut Sbj y saltar a VD8
Vi36	ía	Imp Ind y saltar a VD2
Vi36	i	saltar a Vi23B1
Vi36	í	saltar a V32
Vi36	a	Prs Sbj y saltar a VD6
Vi36	á	Prs Sbj y saltar a VD7
Vi36	ó	3 S Prf Ind

Tabla 42: Grupos y reglas para el grupo 37 de verbos irregulares

Grupo	Marca	Acción
Vi37B	í	saltar a Vi35C1
Vi37B	ía	Imp Ind y saltar a VD2
Vi37B	i	saltar a Vi11A2

Grupo	Marca	Acción
Vi37C	a	Prs Sbj y saltar a VD8
Vi37C	á	Prs Sbj y saltar a VD7
Vi37C	ε	saltar a Vi23A1

Tabla 43: Desinencias personales.

Grupo	Marca	Acción
VD1	s	2 S
VD1	is	2 P
VD1	n	3 P
VD1	ε	1 3 S
vD2	s	2 S
vD2	mo	1 P
VD2	is	2 P
VD2	n	3 P
VD2	ε	1 3 S
VD3	i	2 P
VD3	ε	1 S
VD4	s	2 S
VD4	n	3 P
VD4	ε	3 S
VD5	S	2 S Prs Ind
VD5	mos	1 P Prs Prf Ind
VD5	n	3 P Prs IndK
VD5	ste	2 S Prf Ind
VD5	steis	2 P Prf Ind
VD5	ron	3 P Prf Ind
VD5	ε	2 S Imp 3 S Prs Ind

Grupo	Marca	Acción
VD6	s	2 S
VD6	mos	1 P
VD6	n	3 P
VD6	ε	1 3 P
VD7	is	2 P
VD8	mos	1 P
VD9	ste	2 S
VD9	ó	3 S
VD9	steis	2 P
VD10	s	2 S
VD10	mos	1 P
VD10	n	3 P
VD10	ε	3 S, 2 S Imp
VD11	ste	2 S
VD11	ó	3 S
VD11	mos	1 P
VD11	steis	2 P
VD12	ron	3 P

Tabla 44: Desinencias personales (cont.).

Grupo	Marca	Acción
VD13	s	2 S
VD13	n	3 P
VD13	ε	3 S, 2 S Imp
VD14	s	2 S
VD14	n	3 P
VD14	ε	1 3 S
VD15	mos	1 P Prs
VD15	ste	2 S
VD15	steis	2 P
VD15	ron	3 P
VD16	e	1 S
VD16	iste	2 S
VD16	o	3 S
VD16	imos	1 P
VD16	isteis	2 P
VD17	ndo	Ger
VD18	ste	2 S Prf Ind
VD18	mos	1 P Prf Ind
VD18	steis	2 P Prf Ind
VD18	d	Ppio y saltar a G1
VD18	ε	1 S Prf Ind
VD19	s	2 S Prs Ind
VD19	mos	1 P Prs Ind
VD19	n	3 P Prs Ind
VD19	d	2 P Imp
VD19	r	Inf
VD19	ε	S Prs Ind

Grupo	Marca	Acción
VD20	ste	2 S
VD20	mos	1 P
VD20	steis	2 P
VD21	ste	2 S
VD21	o	3 S
VD21	mos	1 P
VD21	steis	2 P
VD21	ε	1 S
VD22	í	1 S
VD22	iste	2 S
VD22	ó	3 S
VD22	imos	1 P
VD22	isteis	2 P
VD23	mos	1 P Prs
VD23	ste	2 S
VD23	steis	2 P
VD24	s	2 P
VD25	ste	2 S
VD25	ó	3 S
VD25	mos	1 P Prs
VD25	steis	2 P
VD26	mos	1 P Prs Prf Ind
VD26	steis	2 P Prf Ind
VD26	ste	2 S Prf Ind
VD26	s	2 P Prs Ind
VD26	ε	1 S Prf Ind