Universidad de Santiago de Compostela

Una contribución al procesamiento automático de la sinonimia utilizando Prolog

Santiago Fernández Lanza

Tesis de Doctorado

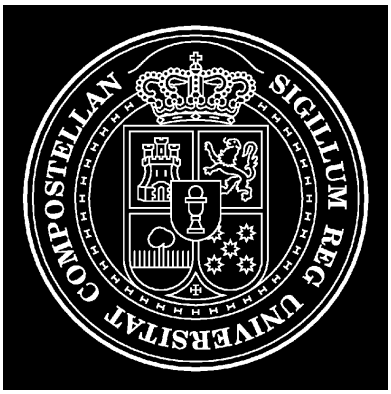
Facultad: Filosofía

Director: Enrique Trillas Ruiz

2001

UNIVERSIDAD DE SANTIAGO DE COMPOSTELA

Departamento de Lógica y Filosofía Moral



Tesis doctoral

Una contribución al procesamiento automático de la sinonimia utilizando Prolog

Autor: Santiago Fernández Lanza

A mis padres Amalia y Claudio;

dignos merecedores del fruto de mi esfuerzo.

Agradecimientos

Aunque la elaboración de una tesis requiere un esfuerzo personal que debe ser realizado de forma solitaria, ninguna de ellas puede ser llevada a cabo sin el apoyo científico, moral y afectivo de las personas que rodean a su autor. El agradecimiento de este apoyo es el motivo de estas primeras líneas.

Es justo comenzar por la persona que ha seguido más de cerca el transcurso de este trabajo, su director, Alejandro Sobrino. A él debo agradecer su constancia, eficacia y rapidez de respuesta para la toma de decisiones respecto a la elaboración de este análisis sobre la sinonimia.

El marco del proyecto de investigación "Interrogación de bases de datos textuales estructuradas" ha servido de apoyo económico y científico desde el año 1999 hasta hoy y me ha permitido entrar, de forma directa o indirecta, en contacto interdisciplinar con personas del campo de la computación, la lingüística o las matemáticas. En las reuniones de trabajo del citado proyecto

hemos discutido numerosos problemas que se han visto plasmados en los resultados del trabajo. En concreto, debo agradecer a Francisco Fernández sus comentarios sobre las cuestiones relativas a las medidas de similaridad que figuran en el capítulo tercero.

El Área de Lógica y Filosofía de la Ciencia de la Universidad de Santiago de Compostela es el marco institucional que ha acogido la elaboración de esta tesis. Con su personal he convivido durante casi cuatro años en los que pude comprobar su calidad humana, lo que hace que sea tan obligado como satisfactorio mencionarlos uno por uno. Uxía Rivas contribuyó en gran medida a la clarificación de la mayor parte de las nociones relativas a la filosofía analítica expuestas en el capítulo segundo de este trabajo. Con Luis Villegas he tenido conversaciones que han resultado de lo más fructífero para muchas de las ideas de este trabajo, en ocasiones realizadas en los largos paseos de camino a casa después de un día de trabajo, con lo que se podría afirmar que gozan del más puro espíritu peripatético. Confieso haber "saqueado" las bibliotecas de José Luis Falguera y Juan Vázquez bajo consentimiento de sus dueños. De Conchín Martínez he recibido ánimos a cambio de haber llorado mis penas, ojalá todas las transacciones fuesen así de fructíferas. Sin duda, guardo el mejor de los recuerdos para mis "compañeros de batalla" José Miguel Sagüillo y Antonio Blanco, con ellos compartí momentos de diversión y de trabajo duro. Finalmente, quedan por mencionar mis becarias favoritas (no por ser las únicas becarias del área) Celeste Cancela y María Caamaño a las que pido disculpas por haberles

hecho perder el tiempo en incontables ocasiones cuando sentía la necesidad de comunicar alguna idea nueva que me surgía de forma intempestiva.

A mi familia debo el cariño y los ánimos necesarios para que esta tesis llegase a ser finalizada, especialmente a mis padres a quienes está merecidamente dedicada. Es preciso mencionar aquí a mi hermana Almudena, puesto que jugó un papel fundamental en la elaboración y corrección de la base de datos del diccionario electrónico que será presentado en el cuarto capítulo. No contenta con ello, se tomó la molestia de leer el documento completo de la tesis una vez finalizada. Sin sus conocimientos sobre lingüística, el período de elaboración de este trabajo hubiese sido mucho mayor.

A todos los mencionados, a los que quedan en el tintero y a los que prefieren permanecer en el anonimato (como el "duendecillo" que colaboró en la transcripción de la bibliografía), muchas gracias por el apoyo prestado, el cariño recibido o los ánimos proporcionados.

ÍNDICE

INTRODUCCIÓN

Este trabajo proporciona una serie de propuestas referentes al procesamiento automático de algunos aspectos de la sinonimia. Para ello se utilizará el Prolog, un lenguaje de programación basado en la lógica que resulta especialmente adecuado para el procesamiento del lenguaje natural. La actitud metodológica del presente análisis de la sinonimia es descriptiva más que explicativa. No se pretenderá, por tanto, proporcionar una explicación o definición de esta relación semántica sino más bien describir su comportamiento tal y como se manifiesta en la práctica. Para que esto último tenga lugar, en buena parte de este trabajo se utilizará un diccionario de sinónimos que será considerado como una muestra razonablemente fiel del comportamiento de la sinonimia en el uso de una lengua. Esto tiene como consecuencia la elaboración de un diccionario electrónico de sinónimos que pondrá de manifiesto el carácter

aplicado de este trabajo y la relevancia del estudio de este aspecto del lenguaje en algunos ámbitos como el procesamiento del lenguaje natural, la lexicografía computacional o la recuperación de información, entre otros.

El interés por la sinonimia nace en la Grecia antigua a manos del sofista Pródico de Ceos. Los primeros trabajos consistían en el análisis de parejas concretas de sinónimos con el fin de matizar el significado de cada uno de los términos, consiguiendo así una mejora en el uso de las palabras. Más tarde, con el nacimiento de la retórica, la sinonimia fue considerada un recurso fundamental que contribuía a evitar la repetición de palabras en un discurso. Con el tiempo, la sinonimia comenzó a estudiarse desde un punto de vista más teórico, hasta que en el siglo XVIII tiene lugar con G. Girard una revolución que marca el origen moderno de los estudios sobre este aspecto del lenguaje. Lo que interesa destacar aquí es que el camino recorrido parece ir de la descripción de la sinonimia en el lenguaje, mediante el análisis de parejas concretas de sinónimos, al establecimiento de definiciones y explicaciones sobre ella.

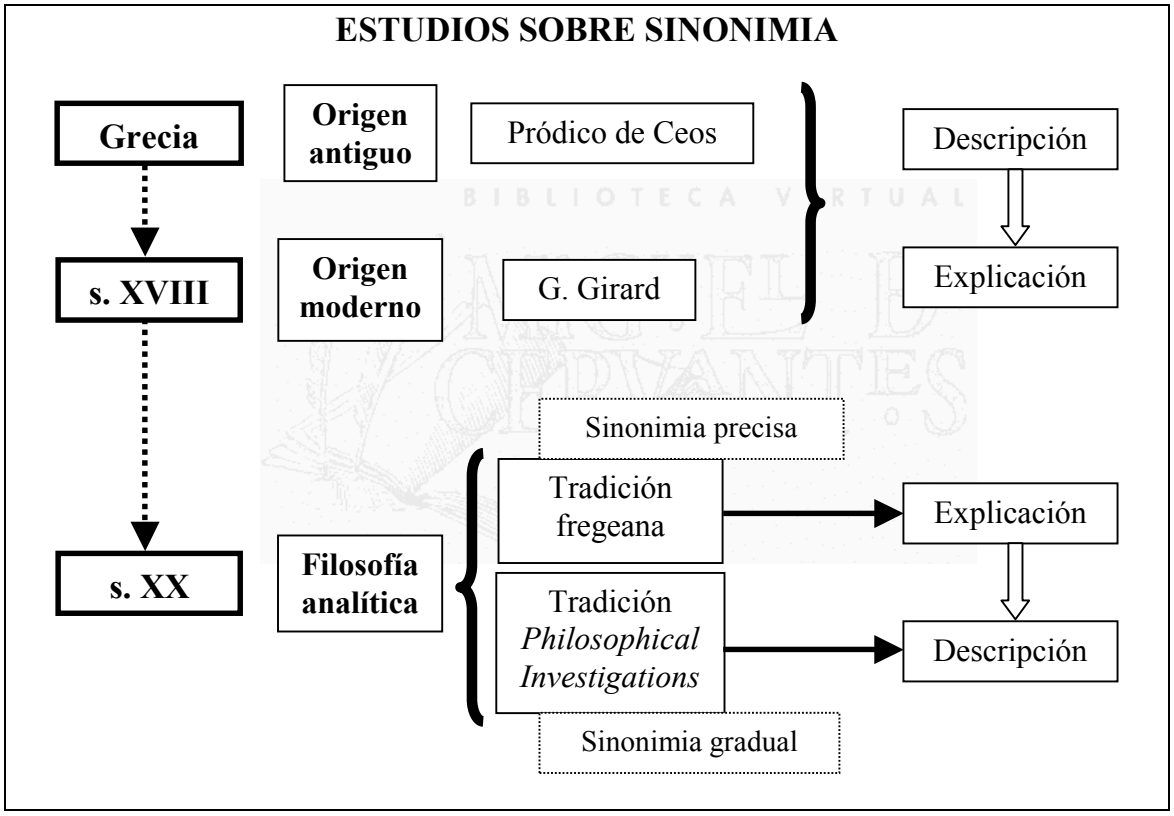
A finales del siglo XIX, Frege publica *Begriffsschrift* convirtiéndose en el padre de la lógica moderna e influyendo de modo decisivo en el surgimiento, a principios del siglo XX, de un cambio en la metodología del filosofar que afectó también a la forma de concebir la sinonimia. Los problemas filosóficos se entendieron fundamentalmente como problemas expresados en un lenguaje, lo que llevó a pensar que un análisis del lenguaje podría arrojar luz sobre aquellos. Este cambio de metodología fue conocido como giro lingüístico y una de sus

consecuencias fue que dos disciplinas como la filosofía y la lingüística se aproximaron. La tendencia más común en la primera mitad de siglo es fiel a la tradición fregeana y trata de llevar a cabo el estudio de cuestiones de los lenguajes naturales con la metodología de análisis de los lenguajes formales, como la lógica de primer orden recién creada por Frege. En este marco, la sinonimia se concibe como una cuestión precisa de todo o nada, de tal forma que dos expresiones serán sinónimas o no lo serán sin que quepan interpretaciones intermedias. Así entendida, la relación de sinonimia debería comportarse en el lenguaje natural como una relación de equivalencia; sin embargo, esto no sucede así en la práctica. Las teorías derivadas de la aplicación de los métodos formales no respondían a las expectativas a la hora de ser contrastadas con el uso del lenguaje natural. Esto provocó el surgimiento, a mediados del siglo XX, de una tendencia alternativa dentro de esta corriente filosófica que tiene su origen en las *Philosophical Investigations* de Wittgenstein y en la que Quine juega un papel importante como se verá en el segundo capítulo de este trabajo. La idea fundamental de esta nueva tradición es que la labor filosófica no debe ser la de explicar o definir los fenómenos, sino la de describirlos. Surge, consiguientemente, una forma de estudiar la sinonimia más centrada en la práctica lingüística, que la concibe de modo gradual más que preciso. Ya no es una cuestión de todo o nada sino que cabe la posibilidad de que dos expresiones sean sinónimas en mayor o menor medida dando lugar a interpretaciones intermedias.

Estos trabajos más pegados al uso lingüístico indican también el carácter contextual de esta relación, es decir, que el hecho de considerar dos expresiones como sinónimas depende del contexto en el que éstas sean utilizadas. De esta forma, dos palabras podrían ser consideradas sinónimas en un contexto y no serlo en otros. Como se verá más adelante, esto puede deberse a dos motivos: por una parte, que el umbral exigido para que dos expresiones sean consideradas sinónimas varíe de unos contextos a otros; por otra parte, que cuando las expresiones son polisémicas, contextos diferentes pueden fijar significados distintos de éstas. Gradualidad y contextualidad son dos de las características de la sinonimia que serán tenidas en cuenta en este trabajo.

De la misma forma que los estudios sobre sinonimia, más propios del ámbito de la lingüística, que tuvieron lugar desde los orígenes griegos hasta finales del siglo XIX, siguen una trayectoria que va de la descripción a la explicación, en los estudios de la filosofía analítica el recorrido parece ser el inverso. Se comienza con la búsqueda de una definición de sinonimia; sin embargo, se observa que surgen problemas al contrastar las propuestas con el uso del lenguaje natural, lo que trae como consecuencia la apuesta por el estudio descriptivo de la sinonimia en el lenguaje natural. A modo de síntesis se podría afirmar que los estudios encaminados a la definición y los orientados a la descripción se complementan de alguna forma, ya que de los primeros surgen problemas que deben ser considerados en los segundos, los cuales suponen un marco relativamente fiable para contrastar los intentos de definición. El análisis

de la sinonimia proporcionado por este trabajo es, más bien, de corte descriptivo. Por su orientación computacional aspira a ser no sólo un marco para contrastar teorías sobre la sinonimia sino también una herramienta para cualquier usuario de la lengua.



La distinción entre lenguaje natural y lenguaje formal permitirá contextualizar el modo en cómo la sinonimia será tratada en este trabajo. Podría considerarse a un lenguaje formal como un lenguaje creado de forma artificial para ser utilizado con un propósito determinado. Las reglas que rigen un lenguaje formal son precisas, están bien definidas y la modificación o cambio de las mismas es un hecho consciente realizado por los propios humanos. Los lenguajes formales son conjuntos de cadenas de símbolos concatenados, obtenidas a partir de un alfabeto o vocabulario finito mediante una serie de reglas de formación.

Para que el lenguaje sea formal, tanto el vocabulario como las reglas de formación de expresiones han de poder ser definidas sin hacer referencia a cuestiones de índole semántica. Un ejemplo de lenguaje formal es el lenguaje de cualquier sistema de lógica simbólica. Podría considerarse a un lenguaje natural como cualquier lenguaje utilizado para la comunicación dentro de una comunidad. Normalmente los lenguajes naturales tienen dos formas, una hablada y otra escrita. Son producto de un proceso lento e impredecible que nunca cesa de cambiar, al que se podría llamar evolución histórica de la comunidad en cuestión. La adquisición del mismo en el individuo forma parte de una de las etapas de su evolución psicológica. La complejidad y el carácter cambiante de los lenguajes naturales son características que hacen de ellos un objeto de estudio muy problemático. Es muy difícil formular reglas precisas sobre ellos, y cuando se hace no es fácil aplicarlas a la totalidad del lenguaje. Tratar de manejar computacionalmente todo un lenguaje natural como el español es una tarea imposible salvo que se limite su aplicación a contextos pequeños y concretos del mismo.

Aunque los lenguajes formales se pueden definir sin hacer referencia a cuestiones semánticas, para que resulten interesantes es necesario asociarles una semántica que asigne un valor a cada una de las expresiones mediante reglas que, al igual que las reglas sintácticas, deben estar bien definidas. Si tiene sentido hablar de sinonimia en el contexto de los lenguajes formales, dos expresiones serían sinónimas si, mediante el uso de las reglas semánticas, se les asocia el

mismo valor a ambas. Dada la precisión de dichas reglas, dos expresiones serán o no serán sinónimas de modo exclusivo sin que haya lugar para matices intermedios. En este sentido, la sinonimia parece coincidir con la equivalencia. Sin embargo, en los lenguajes naturales las cosas no funcionan del mismo modo. Con todo, hubo quienes trataron la sinonimia de los lenguajes naturales en estos términos precisos como por ejemplo los filósofos analíticos herederos de la tradición fregeana. En este trabajo, se denominará indistintamente sinonimia total, absoluta o precisa a este modo de concebir la sinonimia.

La semántica de los lenguajes naturales no posee reglas precisas que asignen significados a cada una de sus expresiones. El concepto de significado continúa siendo un enigma y esto no ayuda en nada a la caracterización de la sinonimia. Dos expresiones pueden considerarse sinónimas pero aún así cualquier hablante medianamente competente puede detectar matices entre una y otra. Por eso la sinonimia no se debería entender como una cuestión de todo o nada sino como una cuestión de grado, de tal forma que se pueda hablar de expresiones más o menos sinónimas. Este mayor o menor grado de sinonimia vendrá dado por el uso lingüístico, es decir, dos expresiones serán sinónimas si son utilizadas como sinónimas. En este sentido, la similaridad proporcionará un modelo más adecuado que la equivalencia.

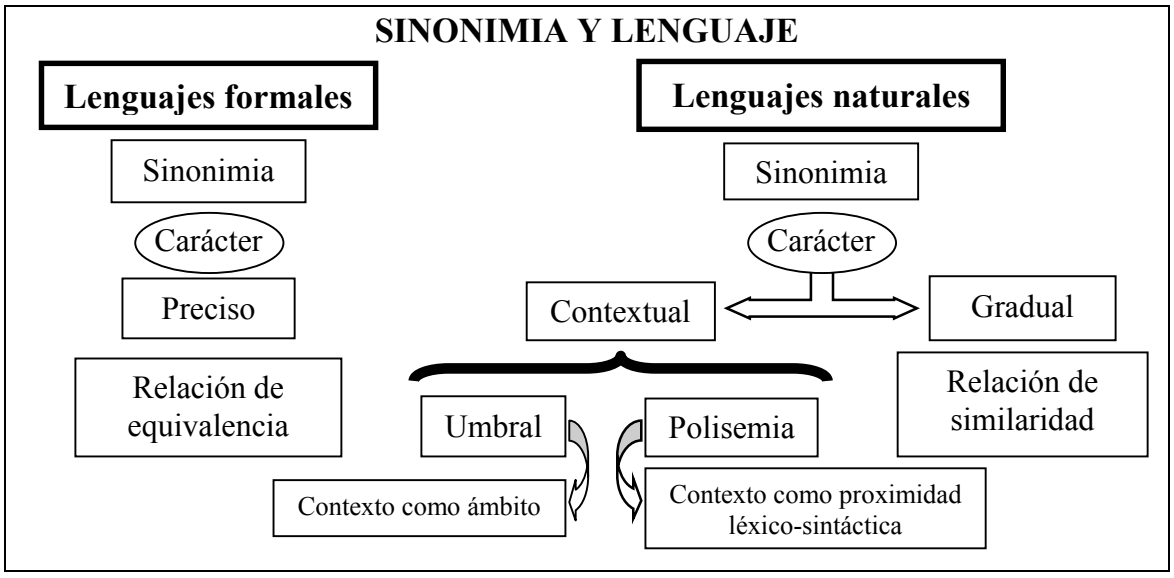
En el presente trabajo se abordará la sinonimia desde este punto de vista. No se tratará de proporcionar una definición de este fenómeno sino una descripción de su comportamiento dentro de un diccionario de sinónimos, que

podría considerarse como la herramienta empírica con la que se pretende reflejar el uso de la sinonimia en la práctica. Un diccionario de sinónimos proporciona para cada una de sus entradas el conjunto de sus sinónimos. El hecho de tener conjuntos hace posible la utilización de medidas de similaridad con las que se puede establecer, de forma numérica, el grado de sinonimia de dos expresiones.

Aparte del carácter gradual de la sinonimia al que se viene aludiendo hasta el momento, un estudio descriptivo como éste también reflejará el carácter contextual de la misma. Este rasgo se manifiesta de dos formas:

1. Dado que dos expresiones pueden ser más o menos sinónimas atendiendo a su carácter gradual, el contexto puede fijar un umbral por debajo del cual dos expresiones no sean consideradas sinónimas. De este modo, dos expresiones podrían ser sinónimas en un contexto donde el umbral exigido sea bajo y no serlo en otro en el que el umbral requerido sea mayor. En este caso, el contexto se concibe como el ámbito o entorno en el que las expresiones son utilizadas.
2. Una expresión puede ser polisémica, es decir, puede tener varios significados o acepciones. Sin embargo, en la práctica, cuando se realiza un uso concreto de ella, se suele emplear solamente uno de estos significados. Eso hace que dos expresiones consideradas sinónimas puedan no serlo por el hecho de estar empleando acepciones de ellas que no las hacen sinónimas. El contexto juega un papel fundamental a la hora de seleccionar el significado adecuado. En este

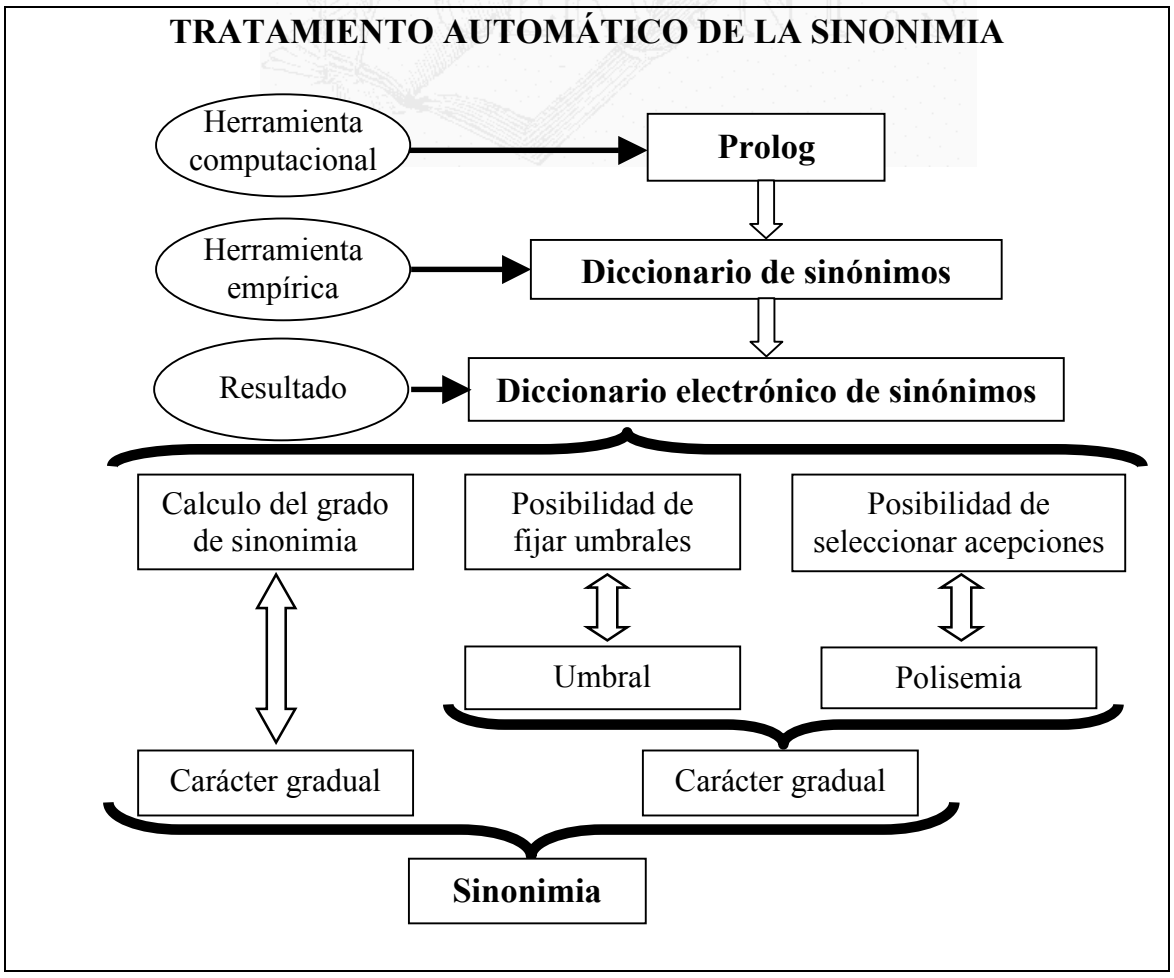
caso, el contexto se concebirá desde un punto de vista más sintáctico como el grupo de expresiones que acompañan a una palabra dentro de una oración. Esta caracterización del contexto posibilita el tratamiento computacional de la desambiguación de palabras, que consiste en la detección automática del significado que se está empleando en un uso concreto de ellas dentro de una oración. Esta es una labor que un humano realiza generalmente (aunque no en todas las ocasiones) sin demasiada dificultad; sin embargo, su tratamiento automático es realmente complicado como se podrá comprobar en el capítulo tercero cuando se dé cuenta de la sinonimia entre oraciones. En el diccionario electrónico de sinónimos resultante de este trabajo, es el usuario quien debe seleccionar el significado de la palabra que proporciona como entrada. Una vez hecho esto, las que el sistema proporciona como respuesta serán desambiguadas de modo automático.



El punto de vista computacional de este trabajo se caracteriza por el uso del lenguaje de programación Prolog para la automatización de la sinonimia teniendo en cuenta los aspectos que se acaban de mencionar: gradualidad, contextualidad etc. El resultado será la elaboración de un diccionario electrónico de sinónimos que calcula grados de sinonimia mediante el uso de medidas de similaridad en el que podrán fijar umbrales de sinonimia y seleccionar determinadas acepciones de las palabras dependiendo del contexto.

De todas formas, un diccionario de sinónimos proporciona una visión parcial de lo que es la sinonimia en el sentido de que dos palabras pueden ser sinónimas y no figurar como tales en él. Además, es frecuente encontrar en ellos pares de expresiones cuyo vínculo quedaría mejor caracterizado mediante otras relaciones semánticas como la hiponimia, la hiperonimia, etc. La primera de estas dos objeciones es parcialmente resoluble completando el diccionario mediante la incorporación de casos no contemplados. La segunda no tiene una solución tan sencilla dado que no se puede establecer una línea divisoria clara entre un diccionario de sinónimos y un diccionario conceptual en el que sí tendrían cabida otras relaciones semánticas como las citadas. Este hecho quizás se deba a que tampoco se pueda trazar una línea divisoria nítida entre la sinonimia y algunas relaciones semánticas como la hiponimia, la hiperonimia, etc. Comúnmente se acepta que la relación que existe entre las palabras de un diccionario de sinónimos es una relación de semejanza de significado, es decir, lo que Lyons denominó cuasisinonimia. De esta relación así entendida pretende dar

cuenta este trabajo. Es cierto que hay otras estrategias empíricas que proporcionan descripciones del comportamiento de la sinonimia en el uso, como por ejemplo la realización de test sobre hablantes de determinada lengua o el análisis de textos, pero el carácter aplicado de esta propuesta hace que el uso de un diccionario de sinónimos sea más atractivo, porque dado el desarrollo actual de los medios de computación, su tratamiento automático es realizable sin un coste excesivo. En resumen, puede considerarse a un diccionario de sinónimos como una muestra razonablemente fiable del comportamiento de la sinonimia en la práctica lingüística cuyo tratamiento computacional es cabalmente posible.



El trabajo consta de cuatro capítulos: En el primero se realiza una descripción del lenguaje de programación Prolog que será utilizado para el tratamiento automático de la sinonimia. Se pondrá de manifiesto la vinculación del Prolog con la parte de la lógica de primer orden que subyace al mismo, así como se establecerán las estrategias de programación en este lenguaje que serán utilizadas en el análisis posterior.

En el segundo capítulo se hará un breve recorrido histórico por algunos intentos de estudio de la sinonimia. Este recorrido involucra disciplinas como la lingüística o la filosofía. Se podrá ver que hay dos formas de estudiar la sinonimia, la que persigue una definición de la misma y la que busca describir su comportamiento en el uso de una lengua. Además se indicará que la polémica existente entre la concepción de la sinonimia como relación precisa que la define como identidad de significado y la concepción de la sinonimia como relación aproximada que la define como parecido o similaridad de significado ya ha existido desde los orígenes de su estudio y continúa existiendo a lo largo de su historia. El último apartado del capítulo describe el camino a seguir en el presente trabajo respecto al análisis de la sinonimia.

Utilizando el Prolog, en el tercer capítulo se dará cuenta de distintos aspectos de la sinonimia en el lenguaje natural siguiendo un modo de exposición común en el marco de la programación lógica y el procesamiento del lenguaje natural. Concibiendo la sinonimia como una cuestión interlingüística se tratará brevemente el tema de la traducción y se verá que, a pesar de la complejidad de

su tratamiento respecto a todo el leguaje natural, su aplicación a ejemplos concretos puede ser viable. Posteriormente, se afrontará la cuestión de la sinonimia entre palabras que servirá de base para la elaboración del diccionario electrónico de sinónimos descrito en el último capítulo. En el tratamiento de la sinonimia entre oraciones, por una parte, se pondrá de manifiesto la complejidad del problema de la desambiguación automática de palabras y por otra se establecerá el análisis de los casos de la pasiva y los términos inversos donde la sinonimia oracional se muestra como una cuestión más bien precisa.

Finalmente, basándose en el análisis sobre sinonimia entre palabras al que se acaba de hacer referencia, el capítulo cuarto describe un diccionario electrónico de sinónimos realizado con Visual Prolog que es capaz, entre otras cosas, de calcular el grado de sinonimia de dos expresiones. Este cálculo del grado de sinonimia permite al diccionario:

1. La detección automática de las palabras que proporciona como respuesta. Es decir, si el usuario introduce una palabra como “fallo” y selecciona una acepción de ésta (por ejemplo, la primera), el diccionario electrónico será capaz de detectar cuál de las acepciones de las expresiones que da como respuesta (“sentencia”, “resolución”, “decisión”, “laudo”) es la sinónima de la entrada. De esta forma, indicará que es la segunda acepción de “sentencia”, la segunda acepción de “resolución”, la segunda acepción de “decisión” y la primera acepción de “laudo” las que son sinónimas de la primera acepción de “fallo”.

2. El establecimiento de un umbral de sinonimia que permite la reducción de las respuestas del diccionario. En el ejemplo anterior, "sentencia", "resolución", "decisión", "laudo" son los sinónimos de la primera acepción de "fallo" cuando 0 es el umbral de sinonimia. Con un umbral más alto (0,7 por ejemplo), el diccionario electrónico no proporcionará como sinónimas aquellas expresiones cuyo grado de sinonimia sea menor que 0,7, con lo que el listado estará constituido ahora por las palabras "sentencia" y "laudo" exclusivamente.
3. El ordenamiento de las respuestas según su mayor proximidad de grado de sinonimia respecto a la expresión de entrada. En el ejemplo, el orden de los sinónimos ("sentencia", "resolución", "decisión", "laudo") es el que proporciona el diccionario impreso. A petición del usuario, el diccionario electrónico los puede ordenar de mayor a menor grado de sinonimia con respecto a "fallo" de la siguiente manera ("laudo", "sentencia", "resolución", "decisión").
4. El aumento del número de respuestas con aquellas expresiones que, aún no siendo consideradas sinónimas en el diccionario impreso, podrían serlo por el hecho de tener un grado de sinonimia distinto de 0 con respecto a la entrada. En el ejemplo, el diccionario electrónico, después de hacer un recorrido por toda la base de datos, es capaz de detectar que las expresiones "acuerdo", "aforismo", "albedrío", "apotegma", "arbitraje", etc. (el listado es relativamente amplio) poseen cierto grado de sinonimia

(bastante bajo en la mayoría de los casos) respecto a la primera acepción de “fallo”.

Tras haber sido realizada la descripción del diccionario electrónico se compararán sus características con las de WordNet y se analizará brevemente su utilidad en ámbitos como el procesamiento del leguaje natural, la lexicografía o la recuperación de información.

CAPÍTULO I

Procesamiento del lenguaje natural

Es común presentar el Prolog como un lenguaje de programación basado en la lógica que está muy próximo a la forma en cómo los humanos nos expresamos en lenguaje natural. El nombre de Prolog es una abreviatura de la expresión francesa "*Programmation avec Logique*". Las ideas que están detrás de la programación lógica en general son el uso de la lógica de predicados de primer orden como medio de expresión y manejo del conocimiento, y el uso de las estrategias propuestas en el ámbito de la demostración automática de teoremas con el fin de capacitar a una máquina para extraer conclusiones a partir de un conjunto finito de fórmulas llamado programa lógico.

Este capítulo posee dos partes: en la primera se tratará de dar cuenta de la lógica que subyace al Prolog, para lo cual se presentará la sintaxis de una lógica de primer orden, su semántica correspondiente, basada en la noción de modelo

Herbrand mínimo y el mecanismo de inferencia, que consiste en la *regla de resolución-SLD*, a la que posteriormente se le añadirá la meta-regla de la *negación como fallo finito* que posibilita el tratamiento de la negación. En la segunda parte, se expondrán las herramientas más comunes de la programación lógica que serán empleadas en el resto del trabajo. Partiendo de la sintaxis del Prolog y con ayuda de ejemplos, se mostrará cómo se construye una base de hechos y una base de reglas, para continuar con el tratamiento de listas y finalmente con el de gramáticas, aspecto éste, que hace del Prolog un lenguaje de programación especialmente adecuado para el procesamiento del lenguaje natural.

1.- La lógica del Prolog

1.1.- Sintaxis

Como se ha indicado, una de las ideas que están detrás del Prolog es el uso de la lógica de predicados de primer orden para representar conocimiento. Un lenguaje formal para un sistema de lógica de primer orden suele venir dado por:

- *Constantes*: Cualquier sucesión de letras minúsculas, que generalmente forman expresiones significativas con el fin de identificar el significado de la constante.
- *Variables*: Cualquier sucesión de letras, la primera de las cuales debe ser mayúscula.

- *Functores*: Cualquier sucesión de letras minúsculas, que generalmente forman expresiones significativas con el fin de identificar la función que realiza el functor.
- *Términos*:
  - Una constante es un término.
  - Una variable es un término.
  - Un functor seguido de una serie de términos entre paréntesis y separados por comas es también un término:

    $$functor(término_1, ..., término_n)$$

    El número de términos que figuran dentro del paréntesis indica la aridad o número de argumentos del functor.
- *Predicados*: Cualquier sucesión de letras en minúsculas, que generalmente forman expresiones significativas con el fin de identificar la propiedad o relación que describe el predicado.
- *Conectivas*: Generalmente se utilizan la *negación* "¬", *conjunción* "∧", *disyunción* "∨", *condicional* "→" y *bicondicional* "↔".
- *Cuantificadores*: El *generalizador* o *cuantificador universal* "∀" y el *particularizador* o *cuantificador existencial* "∃".
- *Fórmulas bien formadas* (*fbf*):
  - *Formulas atómicas*: Un predicado seguido de una serie de términos entre paréntesis y separados por comas es una *fbf* llamada fórmula atómica:

predicado(término$_1$, ..., término$_n$)

El número de términos que figuran dentro del paréntesis indica la aridad o número de argumentos del predicado.

- Si A y B son *fbfs*, $\neg A$, $A \wedge B$, $A \vee B$, $A \rightarrow B$, $A \leftrightarrow B$ también lo son.
- Si A es una *fbf* y X una variable, $\forall X\ A$ y $\exists X\ A$ también lo son. El alcance del cuantificador afecta a toda la fórmula A. Una variable ocurre ligada en una fórmula si figura bajo el alcance de un cuantificador, en otro caso ocurre libre.

- *Literales*: Un literal es una fórmula atómica (*literal positivo*) o la negación de una fórmula atómica (*literal negativo*).

Una manera usual de presentar las fórmulas de la lógica de primer orden para su tratamiento computacional es la *forma clausal*. Los algoritmos de conversión de fórmulas de primer orden a forma clausal suelen seguir generalmente los siguientes pasos:

1. Convertir la fórmula a *forma normal prenexa*. El resultado será una fórmula cuyos cuantificadores figuran todos al principio de la fórmula. Todas las fórmulas de primer orden pueden ser convertidas a forma normal prenexa.
2. Pasar de la forma normal prenexa a *forma de Skolem*. El resultado de este paso es una fórmula que sólo posee cuantificadores universales, los cuales, debido a esa uniformidad, no se suelen indicar. Es importante señalar que en este paso, se pierde la equivalencia de ambas

fórmulas pero se mantiene la propiedad de que la fórmula es satisfacible si y sólo si su forma de Skolem también lo es.

3. Tomar la forma de Skolem sin cuantificadores y pasarla a *forma normal conjuntiva*, es decir, a conjunción de disyunciones de literales. Cada uno de estos grupos de disyunciones son denominados cláusulas.

- *Cláusulas*: Una cláusula es una *fbf* de la forma:

$$\forall X_1 \ldots \forall X_s(A_1 \vee \ldots \vee A_k \vee \neg B_1 \vee \ldots \vee \neg B_n).$$

donde $A_1, \ldots, A_k, B_1, \ldots, B_n$ son fórmulas atómicas y $X_1, \ldots, X_s$ son todas las variables que ocurren en ellas.

El hecho de que cualquier fórmula de la lógica de primer orden sea satisfacible si y sólo si su forma clausal lo es, hace que representar las fórmulas en forma clausal no constituya una restricción para la programación lógica respecto a la capacidad expresiva de la lógica de primer orden. Lo que sí constituye una restricción es que sólo se considerará un subconjunto de las cláusulas: las *cláusulas de Horn*.

- *Cláusulas de Horn*: Son cláusulas que poseen como máximo un literal positivo. Se pueden distinguir dos tipos de cláusulas de Horn:

1. Las que tienen un literal positivo:

$$\forall X_1 \ldots \forall X_s(A \vee \neg B_1 \vee \ldots \vee \neg B_n)$$

2. Las que no tienen ningún literal positivo:

$$\forall X_1 \ldots \forall X_s(\neg B_1 \vee \ldots \vee \neg B_n)$$

No todas las fórmulas de la lógica de primer orden son expresables en cláusulas de Horn, aunque sí un conjunto interesante de ellas. Prueba de ello es que la mayoría de las teorías matemáticas parecen ser axiomatizables en términos de cláusulas de Horn (Shöning, U. [1989] véase apartado 3.2). Por contrapartida, las ventajas de restringir el sistema a este tipo de cláusulas son grandes. Una de ellas es la obtención de un procedimiento eficiente, cuya complejidad no crece de manera exponencial, para dar cuenta de la satisfacibilidad de las fórmulas. Algo que no acontece en la lógica de primer orden, donde los procedimientos que realizan esa labor, como el método de las tablas de verdad, son de carácter exponencial. Además, como consecuencia de la restricción, se puede contar con un mecanismo de inferencia que es completo para cláusulas de Horn y, al mismo tiempo, tratable computacionalmente de una forma secuencial. Este mecanismo de inferencia es conocido como *resolución-SLD*. Al final de este capítulo se tratará la extensión que se realiza en programación lógica para dar cuenta de cláusulas en general y no sólo de cláusulas de Horn.

Las cláusulas de Horn del tipo 1 señaladas anteriormente se pueden expresar en forma de *cláusulas de programa* o *cláusulas definidas* y las del tipo 2 en forma de *objetivos definidos* que serán presentados en el apartado dedicado a la resolución.

- *Cláusulas definidas*: Sean $A, B_1, \ldots, B_n$ fórmulas atómicas. Una cláusula definida es una expresión de la forma:

$$A \text{ :- } B_1, \ldots, B_n$$

donde A es la cabeza, $B_1, ..., B_n$ el cuerpo y ":-" es un condicional que se lee de forma inversa, de tal modo que la *fbf* de la lógica de primer orden correspondiente tendría la forma:

$$\forall X_1 \ldots \forall X_s(B_1 \wedge \ldots \wedge B_n \rightarrow A)$$

que es equivalente a una cláusula de Horn de la forma:

$$\forall X_1 \ldots \forall X_s(A \vee \neg B_1 \vee \ldots \vee \neg B_n)$$

Hay dos tipos de cláusulas definidas:

- *Hechos*: Son cláusulas definidas que sólo tienen cabeza. Por ejemplo, *niño(luis)* que corresponde a la oración declarativa "Luis es un niño" o *derecha(luis, pablo)* que corresponde a "Luis está a la derecha de Pablo".
- *Reglas*: Son cláusulas definidas con cabeza y cuerpo. Por ejemplo, *izquierda(Y, X)* :- *derecha(X,Y)* que corresponde a la regla "si una persona está a la derecha de otra, entonces ésta está a la izquierda de la primera". Parafraseándolo en un lenguaje más próximo a la lógica: "para todo *X*, para todo *Y*, si *X* está a la derecha de *Y* entonces *Y* está a la izquierda de *X*".

- *Programas Lógicos*: Un programa lógico es un conjunto finito de cláusulas definidas (hechos y reglas). Por ejemplo, las siguientes cláusulas constituirían el programa lógico *P1*:

derecha(luis, pablo)
derecha(lola, luis)
derecha(pepe, lola)
derecha(ana, pepe)

izquierda(Y, X) :- *derecha(X, Y)*

Generalmente un programa Prolog está constituido por dos partes: la base de hechos y la base de reglas. En la base de hechos o base de datos, figura la información sobre las propiedades y relaciones que satisfacen determinados objetos considerados por el programa. La base de reglas expresa conocimiento sobre aspectos que tienen que ver con la base de hechos ampliando así la información sobre dichos objetos. Los hechos y las reglas de los programas lógicos sólo expresan conocimiento positivo, es decir, indican qué elementos de determinada estructura están relacionados, pero no dicen cuándo la relación no tiene lugar.

1.2.- Semántica

Dicho de un modo intuitivo, la semántica del sistema presentado establece correspondencias entre los componentes sintácticos (constantes, functores y predicados) y los elementos, propiedades y relaciones de un mundo. Estas correspondencias permiten interpretar un programa lógico de tal forma que a cada fórmula se le pueda asignar un valor de verdad (verdadero o falso). Si la interpretación que se da a un programa lógico hace verdaderas a todas las fórmulas del programa, se dice que dicha interpretación es un modelo del programa. Cuando el programa lógico tiene al menos un modelo, el conjunto de fórmulas que lo constituyen es satisfacible. Debido a que el tipo de expresiones consideradas se restringe a cláusulas definidas y dado que éstas no contienen

negaciones, resulta imposible que un programa lógico esté compuesto de un conjunto insatisfacible de fórmulas, ya que, de un programa no se podrá deducir una fórmula y su negación, es decir, no se podrán deducir contradicciones. En consecuencia, siempre existirá un modelo que hará verdadero al programa. Además, siempre existirá un modelo mínimo que reflejará la información expresada única y exclusivamente por el programa. Este modelo mínimo se construye recurriendo al concepto de *modelo Herbrand*, que se describirá a través de las siguientes definiciones:

- *Universo Herbrand*: Sea A un alfabeto que contiene al menos una constante, el universo Herbrand U_A es el conjunto de todos los términos construidos a partir de functores y constantes de A, quedando excluidas las variables y todos los términos formados con ellas. En caso de que A no tenga constantes, se elige una arbitrariamente.
- *Base Herbrand*: Sea A un alfabeto que contiene al menos una constante, la base Herbrand B_A es el conjunto de todas las fórmulas atómicas construidas a partir de A.

Estos dos conceptos son relativos al programa sobre el que se aplican. Se presupone que el alfabeto del programa consiste exclusivamente en aquellos símbolos del programa y que éste o bien tiene al menos una constante o bien ésta es elegida de modo arbitrario. El universo Herbrand del programa *P1*, propuesto anteriormente, está formado solamente por constantes, dado que no posee functores:

U_{P1} = *{pablo, luis, lola, pepe, ana}*.

La base Herbrand será el conjunto de todas las fórmulas atómicas formadas con sus dos predicados *derecha* e *izquierda* y sus constantes:

B_{P1} = *{derecha(pablo, pablo), derecha(pablo, luis), derecha(pablo, lola), derecha(pablo, pepe), derecha(pablo, ana), derecha(luis, pablo), derecha(luis, luis), derecha(luis, lola), derecha(luis, pepe), derecha(luis, ana), derecha(lola, pablo), derecha(lola, luis), derecha(lola, lola), derecha(lola, pepe), derecha(lola, ana), derecha(pepe, pablo), derecha(pepe, luis), derecha(pepe, lola), derecha(pepe, pepe), derecha(pepe, ana), derecha(ana, pablo), derecha(ana, luis), derecha(ana, lola), derecha(ana, pepe), derecha(ana, ana), izquierda(pablo, pablo), izquierda(pablo, luis), izquierda(pablo, lola), izquierda(pablo, pepe), izquierda(pablo, ana), izquierda(luis, pablo), izquierda(luis, luis), izquierda(luis, lola), izquierda(luis, pepe), izquierda(luis, ana), izquierda(lola, pablo), izquierda(lola, luis), izquierda(lola, lola), izquierda(lola, pepe), izquierda(lola, ana), izquierda(pepe, pablo), izquierda(pepe, luis), izquierda(pepe, lola), izquierda(pepe, pepe), izquierda(pepe, ana), izquierda(ana, pablo), izquierda(ana, luis), izquierda(ana, lola), izquierda(ana, pepe), izquierda(ana, ana)}*

Sea ahora *P2* el siguiente programa que posee el functor *sucesor*:

```
impar(sucesor(0))
impar(sucesor(sucesor(X))) :- impar(X)
```

El universo Herbrand correspondiente es un conjunto infinito:

U_{P2} = *{0, sucesor(0), sucesor(sucesor(0)), sucesor(sucesor(sucesor(0))), ...}*

y, por lo tanto, la base Herbrand también:

B_{P2} = *{impar(0), impar(sucesor(0)), impar(sucesor(sucesor(0))), ...}*

- *Interpretación Herbrand*: Una interpretación Herbrand *I* de un programa *P*, está formada por el dominio Herbrand U_P y por una función que asigna:
  - A cada constante *c*, esa misma constante.
  - A cada functor n-ario *f*, el functor denotado por f_I y definido como

$$f_I(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$$

- A cada símbolo predicativo n-ario p, la relación p_I que es un subconjunto del conjunto U_p^n de todas las n-tuplas formadas con los objetos del dominio.

Una interpretación Herbrand proporciona un mecanismo para asignar valores de verdad a las fórmulas de un lenguaje lógico formal mediante la asignación de valores semánticos a cada uno de los componentes de dichas fórmulas. Los valores semánticos correspondientes a constantes y functores están predefinidos en la interpretación Herbrand, por eso, para especificarla sólo es necesario dar cuenta de la relación asociada con cada predicado. Por ejemplo, para el programa *P1*, se podrían dar las siguientes interpretaciones entre otras:

$I_1 = \varnothing$

$I_2 =$ *{derecha(lola, luis), izquierda(luis, lola)}*

$I_3 =$ *{derecha(luis, pablo), derecha(lola, luis), derecha(pepe, lola), derecha(ana, pepe)}*

$I_4 =$ *{derecha(luis, pablo), derecha(lola, luis), derecha(pepe, lola), derecha(ana, pepe), izquierda(pablo, luis), izquierda(luis, lola), izquierda(lola, pepe), izquierda(pepe, ana)}*

$I_5 = B_{P1}$

La interpretación I_1, por ser vacía, hace falsos a todos los hechos (las cuatro primeras fórmulas) del programa *P1*, mientras que hace vacuamente verdadera a la regla (última fórmula) de *P1*. I_2 hace verdadera a la segunda fórmula de *P1* dado que en la interpretación figuran relacionados los individuos Lola y Luis mediante la relación "… está a la derecha de …" y esto es lo que

expresa la segunda fórmula. I_2 también hace verdadera a la regla de *P1*, ya que para todo par de elementos *X* e *Y* de la interpretación, si *X* esta relacionado con *Y* mediante la relación "… está a la derecha de …", entonces *Y* está relacionado con *X* mediante la relación "… está a la izquierda de …" que es precisamente lo que establece la regla. La interpretación I_3 hace verdaderos a todos los hechos, ya que relaciona mediante "… está a la derecha de …", a Luis y Pablo, a Lola y Luis, a Pepe y Lola, y a Ana y Pepe, que es lo que se establece en los hechos. Sin embargo, I_3 hace falsa a la regla ya que se pueden encontrar contraejemplos para ella, es decir, existe un *X* y existe un *Y* tal que *X* e *Y* están en la relación "… está a la derecha de …" pero *Y* y *X* no están en la relación "… está a la izquierda de …". En la interpretación I_3 Pepe y Lola están en la relación "… está a la derecha de …" pero Lola y Pepe no están en la relación "… está a la izquierda de …". Esto no sucede en I_4 que hace verdaderas a todas las fórmulas de *P1*. I_5 también hace verdaderas a todas las fórmulas de *P1* por el hecho de incluir todas las combinaciones que se pueden realizar con los predicados y las constantes.

Para el programa *P2*, se podrían dar las siguientes interpretaciones:

$I_1 = \varnothing$

$I_2 = \{impar(sucesor(0))\}$

$I_3 = \{impar(sucesor(0)), impar(sucesor(sucesor(0)))\}$

$I_4 = \{impar(sucesor^n(0)) \mid n \in \{1, 3, 5, 7, ...\}\} =$
$\{impar(sucesor(0)), impar(sucesor(sucesor(sucesor(0)))), ...\}$

$I_5 = B_{P2}$

La interpretación I_1 hace falsa a la primera fórmula de *P2* y hace vacuamente verdadera a la segunda. I_2 hace verdadera a la primera fórmula pero falsa a la segunda, ya que si *sucesor(0)* pertenece al conjunto de los impares, entonces, mediante la regla del programa, *sucesor(sucesor(sucesor(0)))* también debería pertenecer, pero en la interpretación no se indica esta pertenencia. I_3 también hace verdadera a la primera fórmula y falsa a la segunda por el mismo motivo que I_2. I_4 hace verdaderas a ambas fórmulas; de la misma forma que I_5, debido, este último, a que incluye todas las fórmulas atómicas formadas con el predicado impar y los términos de U_{P2}.

- *Modelo Herbrand*: Un modelo Herbrand de un programa *P* es una interpretación Herbrand que hace verdaderas a todas las fórmulas de *P*.

De esta forma las interpretaciones I_1, I_2 e I_3, presentadas respectivamente para *P1* y *P2*, no son modelos Herbrand de dichos programas, mientras que I_4 e I_5, sí que lo son al hacer verdaderas respectivamente a todas las fórmulas de ambos.

Como se ha podido comprobar en las interpretaciones I_5 para *P1* y *P2*, la base Herbrand de un programa siempre es un modelo Herbrand de dicho programa ya que para toda cláusula definida A_0 :- $A_1, ..., A_n$ todas las instancias de $A_0, A_1, ..., A_n$ estarán en la base Herbrand. Sin embargo, lo interesante es encontrar el modelo que incluya exactamente aquellos elementos de la base Herbrand y ningún otro que se sigan del programa. Se llamará *modelo Herbrand mínimo* a este modelo. Para cualquier programa *P*, existe un único modelo

Herbrand mínimo de P que es un subconjunto de la base Herbrand de P y que contiene toda la información positiva del programa. Con el fin de mostrar la existencia de modelos Herbrand mínimos, es suficiente mostrar que la intersección de todos los modelos Herbrand es también un modelo Herbrand. Este modelo resultante es el modelo Herbrand mínimo que corresponde al mundo descrito por el programa. La única información que se puede obtener a partir de éste es la de las consecuencias lógicas derivadas de los hechos y las reglas, que no es otra que la que figura en el modelo Herbrand mínimo del programa.

Intuitivamente la construcción del modelo Herbrand mínimo se basa en el concepto de punto fijo. Un punto fijo de una función $f: A \rightarrow A$ es un elemento $x \in A$ tal que $f(x) = x$. El primer paso para la construcción del modelo Herbrand mínimo de un programa P es crear una interpretación I_1 que incluya todos los hechos del programa. Si en una interpretación falta alguno de los hechos del programa, ese hecho será falso para la interpretación y ésta no podrá ser nunca un modelo del programa. Si en el programa figura una regla A_0 :- $A_1, ..., A_m$ ($m > 0$), y hay instancias de $A_1, ..., A_m$ en la interpretación I_1, es decir, I_1 hace verdaderas a esas instancias, entonces las instancias respectivas de A_0 también tienen que estar en I_1, o lo que es lo mismo, I_1 tiene que hacerlas verdaderas también a ellas. Utilizando todas las instancias de $A_1, ..., A_m$ de todas las reglas del programa se construirá una interpretación I_2, que incluya a I_1 e incorpore todas las instancias de A_0 resultantes. Este proceso se continuará siempre que se sigan generando nuevos elementos en la interpretación. Estos nuevos elementos son aquellos que

se siguen inmediatamente de la interpretación anterior, es decir, son las consecuencias inmediatas de la anterior. Si una interpretación I_{i+1} no incluye elementos nuevos con respecto a I_i, ésta será el modelo Herbrand mínimo del programa. Llamando *operador de consecuencia inmediata*, T_P, a una función que toma como argumento una interpretación I de un programa P y da como valor la interpretación que contiene a I y a las consecuencias inmediatas de I, se puede afirmar que la aplicación iterativa de la función T_P termina en un punto fijo tal que $T_P(I) = I$, de tal forma que I será, en este caso, el modelo Herbrand mínimo de P. Los elementos de la secuencia generada por la aplicación iterativa de la función T_P se suelen denotar como $T_P\uparrow 0$, $T_P\uparrow 1$, $T_P\uparrow 2$, … Si la secuencia de iteraciones continúa hacia el infinito y no se ha llegado a un punto fijo, el modelo Herbrand mínimo es el límite de dicha secuencia y se denota como $T_P\uparrow\omega$. La construcción del modelo Herbrand mínimo del programa *P1*, se realiza con la siguiente secuencia finita de iteraciones de T_P:

$T_{P1}\uparrow 0 = \varnothing$

$T_{P1}\uparrow 1 = T_{P1}(T_{P1}\uparrow 0) =$ *{derecha(luis, pablo), derecha(lola, luis), derecha(pepe, lola), derecha(ana, pepe)}*

$T_{P1}\uparrow 2 = T_{P1}(T_{P1}\uparrow 1) =$ *{derecha(luis, pablo), derecha(lola, luis), derecha(pepe, lola), derecha(ana, pepe), izquierda(pablo, luis), izquierda(luis, lola), izquierda(lola, pepe), izquierda(pepe, ana)}*

$T_{P1}\uparrow 3 = \boxed{T_{P1}(T_{P1}\uparrow 2) = T_{P1}\uparrow 2}$ (punto fijo)

Sin embargo, para *P2* la secuencia es infinita:

$T_{P2}\uparrow 0 = \varnothing$

$T_{P2}\uparrow 1 = T_{P2}(T_{P2}\uparrow 0) =$ *{impar(sucesor(0))}*

$T_{P2}\uparrow 2 = T_{P2}(T_{P2}\uparrow 1) = \{impar(sucesor(0)),$
$impar(sucesor(sucesor(sucesor(0))))\}$

. . .

$T_{P2}\uparrow \omega = \{impar(sucesor^n(0)) \mid n \in \{1, 3, 5, 7, ...\}\}$

1.3.- El mecanismo de inferencia

Hasta el momento, se ha presentado buena parte de la sintaxis y la semántica que subyace a los programas Prolog, sin embargo, falta mencionar una parte fundamental de todo sistema lógico: el mecanismo de inferencia. El mecanismo de inferencia utilizado en programación lógica es la *resolución-SLD* (resolución Lineal para cláusulas Definidas con función de Selección). Éste será el que permita obtener consecuencias lógicas a partir de un programa. Como se ha visto en el caso de *P2*, las consecuencias lógicas de un programa pueden ser infinitas. Por esta razón, la ejecución de un programa lógico no consiste en el establecimiento de todas las consecuencias del mismo, sino que se espera que el usuario introduzca una *pregunta* o *consulta* con la que éste selecciona un conjunto, que puede ser a su vez finito o infinito, de consecuencias lógicas del programa. Esta pregunta o consulta se expresa mediante un *objetivo definido*:

- *Objetivos definidos*: Sean $B_1, ..., B_n$ fórmulas atómicas. Un objetivo definido es una expresión de la forma:

  $$:- B_1, ..., B_n$$

  que corresponde a una cláusula de Horn de la forma:

  $$\forall X_1 \ldots \forall X_s(\neg B_1 \vee \ldots \vee \neg B_n).$$

Por ejemplo, si para el programa *P1* se establece el objetivo:

:- *izquierda(X, lola)*

se está indicando que para todo *X*, *X* no está a la izquierda de Lola, o lo que es lo mismo, no existe un *X* que esté a la izquierda de Lola. A pesar de esta interpretación lógica del objetivo, podría darse una interpretación más lingüística diciendo que corresponde a la pregunta "¿qué individuos están a la izquierda de Lola?". La respuesta consistirá en instanciar la variable *X* con aquellos individuos que están a la izquierda de Lola. Tal y como está definido el programa, la variable *X* se instanciaría con Luis. Si la consulta no tiene variables, la respuesta será de tipo sí/no. Por ejemplo, si sobre *P1* se realiza la consulta:

:- *izquierda(pepe, ana)*

la respuesta será afirmativa. Mientras que la respuesta a:

:- *derecha(pepe, ana)*

será negativa.

- *Objetivo vacío*: Es un objetivo que no contiene fórmula atómica alguna y se representa mediante el símbolo "□". En el proceso de resolución, se alcanza el objetivo vacío cuando se llega a una contradicción.

La diferenciación entre cláusulas y objetivos definidos permite una distinción, que es a veces explícita en algunos compiladores de Prolog, entre lo que se suele llamar *módulo de usuario* y *módulo de consulta*. En el módulo de usuario es donde se insertan las cláusulas del programa y en el de consulta se

insertan los objetivos. Generalmente se utiliza un símbolo como "`?-`" (puede variar según el compilador) en lugar del ":-" para indicar que la cláusula que figurará a continuación del mismo no pertenece al código del programa, sino que es una pregunta o consulta que se propone al programa, tras haber sido compilado, para que proporcione una respuesta. En algunas ocasiones, cuando la consulta tiene variables y no interesa que éstas sean instanciadas en la respuesta, se utilizan las llamadas variables anónimas, representadas con un carácter de subrayado "_". De esta manera, el sistema considerará la expresión como variable, pero no la instanciará en la ejecución.

La resolución-SLD se define para *cláusulas de Horn*. Dicho de una forma intuitiva, consiste en deducir $C_1 \vee C_2$ a partir de dos cláusulas de Horn de la forma $A \vee C_1$ y $\neg A \vee C_2$, donde A es una fórmula atómica. El problema es que A está formada por un predicado y una serie de términos, los cuales no siempre coinciden respectivamente en A y en $\neg A$. Por ejemplo, puede deducirse $C_1 \vee C_2$ a partir de $predicado(a, b) \vee C_1$ y $\neg predicado(a, b) \vee C_2$, pero también pueden darse casos como $predicado(X, b) \vee C_1$ y $\neg predicado(a, Y) \vee C_2$, donde lo que figura a la izquierda de ambas disyunciones no son la misma fórmula atómica. Sin embargo, si se sustituye la constante a por la variable X y la constante b por la variable Y en ambas cláusulas, se consigue llegar a la misma fórmula atómica, y por lo tanto, deducir la expresión $C_1 \vee C_2$ tras haber realizado la sustitución en ella. Esta sustitución es admisible dado que todas las variables de las cláusulas de Horn están cuantificadas universalmente. Es decir, si es cierto que para todo X se

da que *predicado(X, b)* $\vee C_1$, entonces también se dará el caso particular *predicado(a, b)* $\vee C_1$. A este proceso se le conoce con el nombre de unificación y como se ha podido observar con el ejemplo, resulta necesario para que la resolución se lleve a cabo. A continuación se establecerán una serie de definiciones para dar cuenta del procedimiento de unificación.

- *Sustitución*: Una sustitución θ es un conjunto finito de la forma $\{t_1/v_1, \ldots, t_n/v_n\}$, donde cada v_i es una variable y cada t_i es un término distinto de v_i, las variables $v_1, \ldots, v_n$ son todas distintas y el símbolo "/" puede leerse como "sustitución de la variable v_i por el término t_i". De esta forma, si N es un conjunto de términos, $N\theta$ es el resultado de sustituir en N todas las ocurrencias de las variables v_i por los respectivos términos t_i ($i = 1, \ldots, n$). Si $\theta = \varnothing$, entonces la sustitución se denomina sustitución idéntica y se denota por ε, de tal forma que para todo conjunto de términos N, $N\varepsilon = N$.
- *Composición de sustituciones*: Sean dos sustituciones $\theta = \{s_1/u_1, \ldots, s_n/u_n\}$ y $\sigma = \{t_1/v_1, \ldots, t_n/v_n\}$, donde cada s_i y t_i son términos y cada u_i y v_i son variables. La composición de sustituciones denotada por $\theta \circ \sigma$ es la sustitución resultante de borrar en el conjunto $\{s_1\sigma/u_1, \ldots, s_m\sigma/u_m, t_1/v_1, \ldots, t_m/v_m\}$ todas las $s_i\sigma/u_i$ tales que $u_i = s_i\sigma$, y todas las t_j/v_j tales que $v_j \in \{u_1, \ldots, u_m\}$.
- *Unificador*: Dado un conjunto de términos N, se dice que la sustitución θ es un unificador de N si y sólo si $N\theta$ es un conjunto con un único

elemento. Es decir, un unificador de un conjunto de términos es una sustitución que hace idénticos a todos los términos de dicho conjunto.

- *Unificador más general (UMG)*: Dado un conjunto finito de términos *N*, se dice que el unificador θ es el unificador más general para *N* si para todo unificador γ de *N* existe una sustitución λ tal que $\theta \circ \lambda = \gamma$.

El procedimiento de resolución-SLD consiste en averiguar si un objetivo es derivable de un programa. Para ello se parte del objetivo y se trata de llegar al objetivo vacío (contradicción). A esta manera de proceder se le conoce con el nombre de *refutación* o *reducción al absurdo*. Si el objetivo inicial tenía variables, el hecho de haber llegado al objetivo vacío supone haber sustituido dichas variables por términos. Esta sustitución constituye la respuesta a la consulta. Si el objetivo no tenía variables, la respuesta a la consulta será afirmativa si a partir del objetivo se llega al objetivo vacío o negativa en caso contrario.

Para llegar al objetivo vacío a partir de un objetivo definido se aplica, de forma reiterada, la regla de resolución-SLD, la cual permitirá el paso de un objetivo a otro hasta llegar, en caso de que la consulta tenga éxito, al objetivo vacío. Añadiendo ahora el concepto de unificación a la explicación intuitiva de la regla de resolución-SLD proporcionada anteriormente, se podría indicar que a partir de dos cláusulas de Horn $p(t_1, \ldots, t_n) \vee C_1$ y $\neg p(t'_1, \ldots, t'_n) \vee C_2$, se puede concluir $C_1\theta \vee C_2\theta$, donde $t_1, \ldots, t_n$ y $t'_1, \ldots, t'_n$ son términos tales que cada t_i y t'_i son unificables mediante el *UMG* θ y p es un predicado. A continuación se

presentará la regla de resolución-SLD utilizando la sintaxis para objetivos y cláusulas definidas considerada en este trabajo.

- *Regla de resolución-SLD*: Sean $A_1, \ldots, A_{i-1}, A_i, A_{i+1}, \ldots, A_m, B_0, B_1, \ldots, B_n$ fórmulas atómicas y θ una sustitución, donde A_i es de la forma $p(t_1, \ldots, t_s)$, B_0 es de la forma $p(t'_1, \ldots, t'_s)$, p es un predicado y θ el *UMG* de cada t_i, t'_i. La regla de resolución-SLD dice que a partir de un objetivo definido de la forma:

  $$:- A_1, \ldots, A_{i-1}, A_i, A_{i+1}, \ldots, A_m$$

  y una cláusula definida de la forma:

  $$B_0 :- B_1, \ldots, B_n$$

  se puede obtener un nuevo objetivo:

  $$:- (A_1, \ldots, A_{i-1}, B_1, \ldots, B_n, A_{i+1}, \ldots, A_m)\theta.$$

Otra de las estrategias que forman parte del proceso de resolución es la denominada *regla de computación* o *función de selección*, la cual selecciona, a partir de un objetivo, el subobjetivo sobre el que se aplicará la unificación. Visto sobre la regla de resolución-SLD que se acaba de presentar, lo que hace la función de selección es elegir el A_i del primer objetivo sobre el que se va a aplicar la regla.

Sea el siguiente programa *P3*:

```
derecha(luis, pablo)
derecha(lola, luis)
derecha(pepe, lola)
derecha(ana, pepe)
izquierda(Y, X) :- derecha(X, Y)
entre(X, Y, Z) :- derecha(X, Y), izquierda(X, Z)
```

Supóngase que un usuario del programa quiere averiguar ¿quién está entre Pablo y Lola?. Una pregunta como esta se formalizaría con el objetivo O_0:

:- *entre(Q, pablo, lola)*

A partir de este objetivo y aplicando el principio de resolución-SLD se tratará de llegar a una contradicción (objetivo vacío). Si se observa el programa, la última regla indica cuáles son las condiciones para que alguien esté entre otros dos:

entre(X, Y, Z) :- *derecha(X, Y), izquierda(X, Z)*

Aplicando el principio de resolución-SLD, se obtendrá un nuevo objetivo sustituyendo el anterior por la expresión que figura a la derecha de la regla después de haber unificado la fórmula mediante el *UMG {Q/X, pablo/Y, lola/Z}*, que, como se puede observar, unifica respectivamente los conjuntos de términos *{Q, X}*, *{pablo, Y}* y *{lola, Z}*. El nuevo objetivo es O_1:

:- *derecha(Q, pablo), izquierda(Q, lola)*

Este objetivo tiene dos subobjetivos, con lo que la función de selección es necesaria aquí para elegir uno de ellos. Seleccionando, por ejemplo, el que está más a la izquierda, podría aplicarse la regla de resolución-SLD con la primera cláusula del programa que dice que "Luis está a la derecha de Pablo":

derecha(luis, pablo)

Para obtener un nuevo objetivo O_2 se requiere el unificador *{luis/Q}*:

:- *izquierda(luis, lola)*

La penúltima cláusula del programa es una regla que indica las condiciones bajo las cuales un individuo está a la izquierda de otro:

izquierda(Y, X) :- *derecha(X, Y)*

Aplicando una vez más el principio de resolución-SLD con el objetivo O_2 y esta regla, se obtiene un nuevo objetivo O_3 tras haber utilizado el unificador *{luis/Y, lola/X}*:

:- *derecha(lola, luis)*

La segunda cláusula del programa establece:

derecha(lola, luis)

O_3 dice que "*lola* no está a la derecha de *luis*" y esta cláusula del programa dice que "*lola* está a la derecha de *luis*", con lo que se llega a una contradicción y por tanto al objetivo vacío O_4:

□

Por lo tanto, se puede negar el objetivo O_0, diciendo que efectivamente existe un individuo que está entre Pablo y Lola, ese individuo es Luis que fue por quien se ha sustituido la variable *Q* en la construcción del objetivo O_2.

Todo este proceso que se acaba de describir se denomina *derivación-SLD* del objetivo O_0 y puede esquematizarse con la siguiente figura:

O_0 :- *entre(Q, pablo, lola)*
entre(X, Y, Z) :- *derecha(X, Y), izquierda(X, Z)* *{Q/X, pablo/Y, lola/Z}*

O_1 :- *derecha(Q, pablo), izquierda(Q, lola)*
derecha(luis, pablo) *{luis/Q}*

O_2 :- *izquierda(luis, lola)*
izquierda(Y, X) :- *derecha(X, Y)* *{luis/Y, lola/X}*

O_3 :- *derecha(lola, luis)*
derecha(lola, luis) ∅

O_4 □

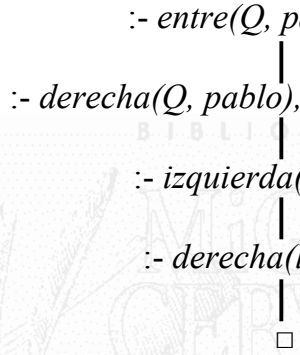
Una derivación-SLD puede ser finita o infinita. Dentro de las finitas se pueden distinguir las que terminan con el objetivo vacío, llamadas *refutaciones-SLD* y las que no terminan con el objetivo vacío, llamadas *derivaciones-SLD fallidas* o *finitamente fallidas*.

Vinculado al concepto de resolución-SLD está el concepto de *árbol-SLD*, que permite esquematizar cómo Prolog busca las respuestas correctas ante un objetivo:

- *Árbol-SLD*: Sea P un programa, G_0 un objetivo y R una función de selección, el árbol-SLD de G_0 es un árbol cuyos nodos están etiquetados con objetivos de tal forma que:
  - El objetivo que corresponde a la raíz del árbol es G_0.
  - Si el árbol contiene un nodo etiquetado con el objetivo G_i y hay alguna cláusula de P tal que aplicando R se obtiene otro objetivo G_{i+1} mediante la regla de resolución, entonces del nodo etiquetado con G_i saldrá una rama hacia el nodo etiquetado con G_{i+1}.

- *Árbol-SLD finitamente fallido*: A un árbol-SLD finito en el que ninguna de sus ramas termina con el objetivo vacío, se le denomina árbol-SLD finitamente fallido.

El árbol-SLD que corresponde al objetivo O_0 del ejemplo anterior es el siguiente:



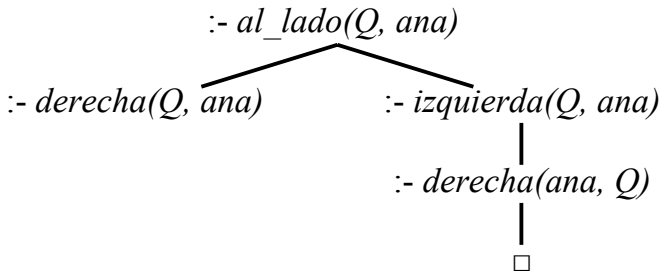
Cuando en el programa aparecen varias reglas para un mismo predicado, el árbol que haga referencia a tal predicado tendrá bifurcaciones. Por ejemplo, sea *P4* el siguiente programa:

```
derecha(luis, pablo)
derecha(lola, luis)
derecha(pepe, lola)
derecha(ana, pepe)
izquierda(Y, X) :- derecha(X, Y)
al_lado(X, Y) :- derecha(X, Y)
al_lado(X, Y) :- izquierda(X, Y)
```

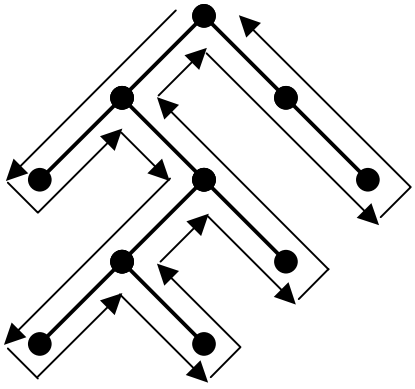
A partir del objetivo O_0:

```
:- al_lado(Q, ana)
```

se puede obtener el árbol-SLD:

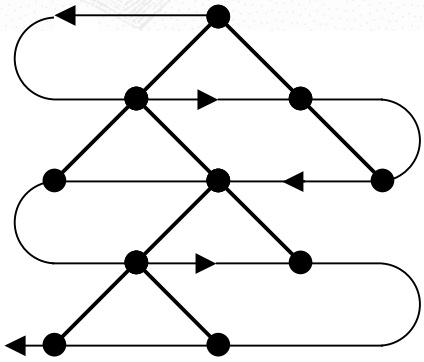


El problema de cómo Prolog encuentra una respuesta a un objetivo, se reduce a una búsqueda sistemática en el árbol-SLD. Normalmente los sistemas Prolog aprovechan el orden de las cláusulas que figuran en el código del programa para ordenar las ramas que descienden del árbol-SLD. En un primer momento, la búsqueda es *prioritaria en profundidad*, que consiste en recorrer los nodos del árbol de arriba hacia abajo hasta encontrar un nodo final. Cuando dicho nodo es encontrado, se vuelve hacia atrás hasta el nodo precedente que tenga ramas sin explorar. Este procedimiento de vuelta atrás se conoce con el nombre de *reevaluación* (*backtracking*). La estrategia general queda esquematizada con la siguiente figura:



En caso de que el nodo final encontrado sea el objetivo vacío, antes de realizarse la reevaluación, se proporciona una respuesta con la sustitución de las

variables del objetivo inicial por aquellas constantes que llevaron al objetivo vacío. Normalmente, en este punto el sistema queda a la espera de que el usuario le indique si desea más respuestas. El tipo de indicación estándar es el punto y coma. Al ser introducido, el sistema debe proporcionar una nueva respuesta a la consulta en caso de que exista, o responder "no" en caso de que no haya más respuestas. Si el árbol es finito, con esta estrategia de búsqueda siempre se obtiene una respuesta. Sin embargo, la profundidad de las ramas de un árbol puede ser infinita. Cuando esto sucede, la estrategia descrita no terminará nunca, con lo que muchas de las respuestas podrían no haber sido proporcionadas. La solución a este problema consiste en emplear la estrategia de búsqueda *prioritaria en anchura*, que se puede esquematizar de la siguiente forma:



La implementación de esta estrategia requiere un complicado manejo de la memoria, por eso la mayoría de los sistemas Prolog utilizan la búsqueda prioritaria en profundidad.

Un recurso que optimiza la búsqueda en un árbol-SLD es el *corte*. Sintácticamente, se podría considerar al corte como un predicado con cero argumentos denotado con el símbolo "!". Este predicado se satisface

inmediatamente y no puede ser satisfecho de nuevo, de tal forma que, cuando aparece en un árbol-SLD, se está indicando que la búsqueda no continúe en el subárbol que figura debajo de él. Esto puede hacerse por varias razones:

1. Para evitar la búsqueda por ramas que no conducen a éxito por ser infinitas o porque no llevan al objetivo vacío. Éste suele denominarse *corte verde* (*green cut*) y es considerado inofensivo puesto que no cambia el significado declarativo del programa.
2. Para evitar la búsqueda por ramas que conducen a respuestas que no son interesantes para los propósitos del programa. Éste suele denominarse *corte rojo* (*red cut*) y su uso suele considerarse peligroso puesto que cambia el significado declarativo del programa debido a que no serán proporcionadas aquellas respuestas derivadas de las ramas afectadas por el corte.

Este recurso hace que el programa funcione más deprisa y consuma menos memoria al no malgastar el tiempo intentando recorrer ramas de forma innecesaria. La importancia del corte llega a tal extremo que, en ocasiones, su carencia puede hacer que el programa no funcione o lo haga de modo incorrecto. En el apartado 2.6 de este capítulo se ha incorporado, a modo de ejemplo, un árbol-SLD donde se ilustra el funcionamiento de este recurso.

1.4.- La negación

Como ya se ha indicado, la resolución-SLD tal y como ha sido presentada aquí, se restringe a cláusulas y objetivos definidos, o lo que es lo mismo, a cláusulas de Horn. Ello significa que ni los objetivos ni las cláusulas del programa pueden contener negadores. Sin embargo, puede ampliarse el alcance de la lógica del Prolog a cláusulas en general. Como se ha visto, cualquier cláusula (general) posee la forma:

$$\forall X_1 \ldots \forall X_s(A_1 \vee \ldots \vee A_k \vee \neg B_1 \vee \ldots \vee \neg B_n)$$

que es equivalente a la *fbf*:

$$\forall X_1 \ldots \forall X_s(\neg A_2 \wedge \ldots \wedge \neg A_k \wedge B_1 \wedge \ldots \wedge B_n \rightarrow A_1)$$

y puede expresarse como un nuevo tipo de cláusula de programa en cuyo cuerpo figuran negadores. Su forma es la siguiente:

$$A_1 \text{ :- } \neg A_2 \wedge \ldots \wedge \neg A_k \wedge B_1 \wedge \ldots \wedge B_n$$

donde $A_1, \ldots, A_k, B_1, \ldots, B_n$ son fórmulas atómicas.

La solución al problema de cómo dar cuenta de la negación en el proceso de resolución se basa en la llamada *hipótesis del mundo cerrado*, que consiste en afirmar que si una fórmula atómica A no es derivable de un programa P entonces es derivable su negación $\neg A$. Esto constituye una meta-regla que podría ser formulada de la siguiente forma en un metalenguaje donde la expresión "de … se deriva …" es expresada mediante el símbolo "ĭ":

$$\frac{P \quad A}{P \text{ ĭ } \neg A}$$

Lo que se afirma en la regla anterior no se cumple en lógica de primer orden. Que una fórmula no se derive de un conjunto de fórmulas no implica que se derive su negación. Puede darse que ni la fórmula ni su negación lo hagan. Por ejemplo, del conjunto de fórmulas de la lógica de primer orden:

$$padre(juan, antonio)$$
$$padre(alberto, pablo)$$
$$\forall X(padre(X, Y) \rightarrow hijo(Y, X)$$

no se deduce *hijo(alberto, juan)* pero tampoco se deduce $\neg$*hijo(alberto, juan)*. Con la hipótesis del mundo cerrado se está indicando que todo lo que el programador asevera en el programa es verdadero mientras que aquello que no está contemplado en el mismo es falso. Realmente esta es una hipótesis muy fuerte pero cotidianamente la carencia de determinada información *A* suele interpretarse como la negación de *A*. Si en el menú de un restaurante no se indica explícitamente que la langosta es uno de los platos, los comensales se inclinarán a pensar que la langosta no es un plato del restaurante aunque esto último no sea cierto.

Un problema grave es que el sistema se vuelve inconsistente al permitir derivar negaciones de un programa. Esto se debe a que, como la base Herbrand es un modelo del mismo y en ésta figuran todas las fórmulas atómicas que se pueden construir con los predicados, functores y constantes del programa, la negación de algunas de estas fórmulas también podrá ser derivada. Las propuestas para evitar este problema se basan generalmente en no considerar aquellos modelos del programa que resultan problemáticos y no interesantes.

A pesar de estas dificultades, la gran ventaja de considerar la hipótesis del mundo cerrado es que se consigue equiparar la capacidad expresiva de la lógica de primer orden con la de la lógica que subyace al Prolog.

Desde el punto de vista de la resolución-SLD, que un objetivo *A* no sea derivable de un programa *P* significa que, o bien la derivación-SLD de *A* es finitamente fallida o bien es infinita. Este último caso es problemático porque no se puede saber si el objetivo ha sido finalmente derivado o no, cosa que no sucede con las derivaciones finitamente fallidas, en las que se tiene la certeza de que el objetivo no puede ser derivado. Por este motivo, para dar cuenta de la negación en Prolog se utiliza una versión más débil de la hipótesis del mundo cerrado denominada *regla de negación como fallo (finito)*. Dicha meta-regla dice que ¬*A* es derivable de un programa *P* si el objetivo :- *A* tiene una derivación-SLD finitamente fallida:

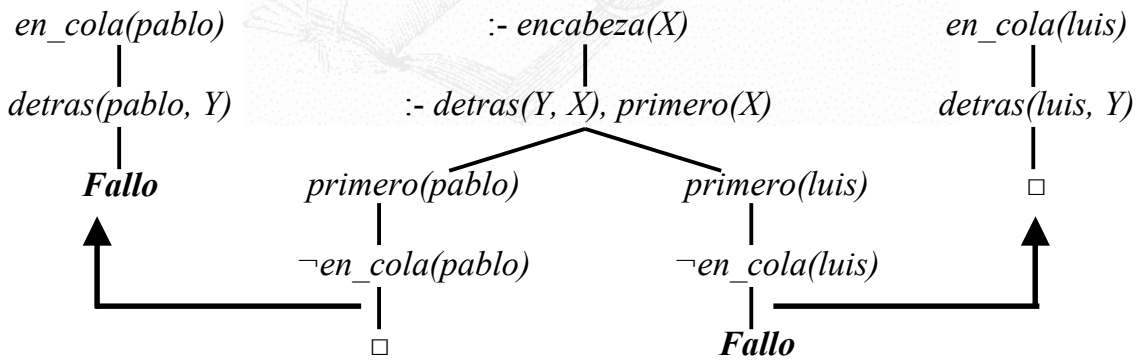
$$\frac{\text{:- } A \text{ tiene una derivación-SLD finitamente fallida a partir de } P}{P \vdash \neg A}$$

Incorporando esta regla al proceso de resolución-SLD se obtiene la llamada resolución-SLDNF (resolución-SLD con regla de negación como fallo finito) que es un procedimiento más complejo que el de resolución-SLD, ya que para probar ¬*A* debe existir una resolución-SLD finitamente fallida de :- *A*, o lo que es lo mismo, un árbol-SLD finitamente fallido de :- *A*. Dicho árbol puede contener, a su vez, objetivos formados por literales negativos, para los cuales habrá que construir otros árboles que pueden ser exitosos o fallidos. El hecho de que existan varios árboles en el proceso de resolución de un objetivo hace que en la

resolución-SLDNF se introduzca el concepto de *bosque-SLDNF*. Una *derivación-SLDNF* es una secuencia de objetivos que pueden incluir negaciones y un *árbol-SLDNF* es la combinación de todas las derivaciones-SLDNF posibles. Un bosque-SLDNF consiste en un conjunto de árboles cuyos nodos son etiquetados por objetivos que pueden incluir negaciones. Sea *P5* el siguiente programa:

detras(luis, pablo)
detras(lola, luis)
en_cola(X) :- *detras(X, Y)*
primero(X) :- ¬*en_cola(X)*
encabeza(X) :- *detras(Y, X), primero(X)*

El bosque-SLDNF correspondiente al objetivo :- *encabeza(X)* sobre *P5* sería el siguiente:



2.- La programación en Prolog

2.1.- Sintaxis

El lenguaje en el que se suelen escribir los programas en Prolog está formado por tres tipos de objetos:

- *Constantes*: Hay dos tipos de constantes:

- Átomos: Pueden ser construidos de tres formas:
  - Como cadenas de letras, cifras y el carácter de subrayado "_", comenzando con una letra minúscula.
  - Como cadenas de caracteres del estilo de "+", "-", "*", "/", "<", ">", "=", ":", ".", "&", "_", "~", etc. Los símbolos "?-", ":-" utilizados respectivamente para los objetivos y las reglas son casos especiales de este tipo de átomos, así como los correspondientes a la conjunción "," y la disyunción ";".
  - Como cadenas de caracteres escritos entre comillas simples. Son útiles cuando se quiere que un átomo comience por mayúscula, que contenga la letra "ñ", o que incluya caracteres con tilde o con diéresis.
- *Números*: Incluye los enteros y los reales dentro de los límites considerados por cada compilador.

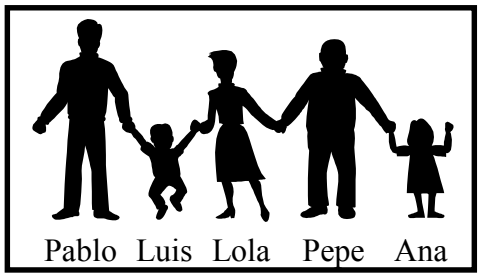
- *Variables*: Se construyen como cadenas de letras, cifras y el carácter de subrayado "_", comenzando con una letra mayúscula o un carácter de subrayado. Un tipo especial de variable son las denominadas:
  - *Variables anónimas*: Se indican con un carácter de subrayado solamente y se utilizan:
    - Cuando una variable aparece una sola vez en una cláusula de programa.

  - Cuando no interesa que determinada variable sea instanciada en la respuesta a un objetivo.

- *Estructuras*: Son objetos compuestos que vienen especificados por un funtor y varios componentes, que pueden a su vez, ser estructuras. Este tipo de objetos ayudan a organizar la información al permitir tratar como un único objeto una serie de datos relacionados. Por ejemplo, `derecha(ana, pepe)`, donde el funtor es la expresión "`derecha`" y los componentes son las constantes "`ana`" y "`pepe`".

2.2.- Hechos

Como se ha dicho, el Prolog es un lenguaje de programación basado en la lógica de primer orden. También se ha indicado que el modelo Herbrand mínimo de un programa realiza una descripción del mundo al que éste hace referencia. Por tanto, podría considerarse un programa lógico como una descripción formal de un mundo. Si éste describe el mundo que su creador pretende describir, entonces es correcto, en otro caso, se requiere modificación. La idea del programa lógico como descripción formal hace que la programación en Prolog sea una actividad similar a la de formalizar en lógica. El primer paso será conocer el mundo que se pretende formalizar. Dicho mundo estará constituido por un conjunto de elementos sobre los que se pueden predicar determinadas propiedades y relaciones. Sea la siguiente figura una representación de un mundo:



Sobre este mundo de cinco elementos se pueden predicar cosas como "Luis es un niño", "Lola es una mujer", "Ana es una niña", "Luis está a la derecha de Pablo", "Pepe está entre Lola y Ana", etc. Como se puede observar, se utilizan predicados para expresar la información referente al mundo. Según su aridad o número de argumentos, los predicados pueden ser monádicos, diádicos, triádicos, etc. Los predicados monádicos poseen un único argumento y suelen manifestar propiedades de los objetos que figuran como argumento. Algunas veces es conveniente expresar previamente en lenguaje natural lo que se quiere implementar en Prolog, de esta forma el código será el resultado de formalizar lo escrito en lenguaje natural utilizando el lenguaje de programación. Algunos ejemplos de predicados monádicos son "... es un hombre", "... es una mujer", etc. Con respecto al mundo del ejemplo, y teniendo en cuenta que en Prolog, el final de las cláusulas y de los objetivos se indica con un punto, las cláusulas:

```
hombre(pablo).
mujer(lola).
```

corresponderían respectivamente a la formalización de "Pablo es un hombre" y "Lola es una mujer".

Los predicados diádicos poseen dos argumentos y suelen representar relaciones entre los objetos que figuran como argumentos. Son ejemplos de

predicados diádicos "… está inmediatamente a la derecha de …", "… está inmediatamente a la izquierda de …", etc. Respecto al mundo propuesto se podría utilizar el siguiente código:

```
inm_derecha(luis,pablo).
inm_derecha(lola,luis).
inm_derecha(pepe,lola).
inm_derecha(ana,pepe).

inm_izquierda(pablo,luis).
inm_izquierda(luis,lola).
inm_izquierda(lola,pepe).
inm_izquierda(pepe,ana).
```

Los predicados triádicos poseen tres argumentos y, al igual que los diádicos, suelen manifestar relaciones entre los objetos que sirven como argumento. Un ejemplo de predicado triádico es "… está entre … y …". Continuando con la descripción del mundo propuesto, el siguiente código da cuenta de la relación "está entre" en dicho mundo:

```
entre(luis,pablo,lola).
entre(lola,luis,pepe).
entre(pepe,lola,ana).
```

Una vez que el programa que contiene los predicados `inm_derecha`, `inm_izquierda`, y `entre` ha sido compilado, se podrían realizar algunas consultas como por ejemplo "¿está Ana inmediatamente a la derecha de Pepe?":

```
?- inm_derecha(ana,pepe).
```

a lo que el compilador responderá:

```
yes
```

Una consulta que daría respuesta negativa debido a que alguno de los objetos que instancian el argumento no posee la propiedad que representa el predicado sería "¿está Pepe inmediatamente a la derecha de Luis?":

```
?- inm_derecha(pepe,luis).
no
```

La respuesta a una consulta también puede ser negativa cuando el predicado no está definido en el programa o el objeto que instancia algún argumento no existe, es decir, no pertenece al universo Herbrand del programa. Algunos ejemplos de esto son los siguientes:

```
?- padre(pablo,luis).
```
(El predicado `padre` no está definido)

```
no
?- entre(rosa,lola,ana).
```
(`rosa` no es un elemento del mundo)

```
no
```

Si se utilizan variables se podrán realizar consultas como "¿quién está inmediatamente a la izquierda de Luis?":

```
?- inm_izquierda(X,luis).
X = pablo
```

2.3.- Reglas

El mundo que se está empleando como ejemplo no es muy grande, ello nos permite definir predicados para cada uno de los elementos del mismo sin demasiada dificultad. Con un mayor número de elementos es necesario el uso de reglas para reducir el coste que supone definir predicados para cada elemento. Utilizando reglas, podemos definir unos predicados en función de otros. En el ejemplo presentado, el predicado "… está entre … y …" se puede definir en función de los predicados "… está inmediatamente a la derecha de …" y "… está

inmediatamente a la izquierda de …", o incluso en función de uno de ellos solamente. La regla resultante expresada en lenguaje natural diría:

Un objeto *X* está entre otros dos *Y* y *Z* si:

- *X* está inmediatamente a la derecha de *Y* y
- *X* está inmediatamente a la izquierda de *Z*.

Una vez establecida la versión en lenguaje natural de la regla, sabiendo que el condicional se simboliza mediante "`:-`" y la conjunción ("y") mediante la coma, la versión en Prolog es una mera formalización de lo expresado en la descripción en lenguaje natural:

```
entre(X,Y,Z):-
      inm_derecha(X,Y),
      inm_izquierda(X,Z).
```

Si sustituimos el predicado `entre` de la base de datos por la regla anterior, el resultado de cualquier consulta será el mismo que el obtenido en las consultas realizadas más arriba. Además, si la base de datos se modifica añadiendo o quitando elementos, no es necesario cambiar la regla, cosa que no sucedía cuando el predicado `entre` formaba parte de la base de datos.

De una forma más precisa y coherente con una concepción intuitiva de la relación "… está entre … y …", se podría definir un nuevo predicado, en función del predicado `entre`, de tal forma que si un elemento *X* está entre otros dos *Y* y *Z* entonces *X* también está entre *Z* e *Y*. Para definir dicho predicado se utilizará la disyunción ("o") que se denota con el símbolo ";". Si se llama `sim_entre` al predicado así concebido la versión en Prolog del mismo sería la siguiente:

```
sim_entre(X,Y,Z):-
```

```
entre(X,Y,Z);
entre(X,Z,Y).
```

Mediante el uso de reglas, también se puede definir el predicado `inm_izquierda` en función del predicado `inm_derecha` y viceversa, con lo que el programa se abrevia. Por ejemplo, se puede sustituir el predicado `inm_izquierda` por la siguiente regla de tal forma que sólo haga falta tener el predicado `inm_derecha` en la base de datos:

```
inm_izquierda(X,Y):-
      inm_derecha(Y,X).
```

2.4.- Reglas recursivas

Si se realiza una descripción más exigente del mundo propuesto como ejemplo, se pueden definir las relaciones "… está a la derecha de …" y "… está a la izquierda de …" en lugar de "… está inmediatamente a la derecha de …" y "… está inmediatamente a la izquierda de …". Hay dos formas de hacer esto, una de ellas es aumentar la base de datos añadiendo las nuevas cláusulas, es decir:

```
derecha(luis,pablo).        izquierda(pablo,luis).
derecha(lola,pablo).        izquierda(pablo,lola).
derecha(pepe,pablo).        izquierda(luis,lola).
derecha(ana,pablo).         izquierda(pablo,pepe).
derecha(lola,luis).         izquierda(luis,pepe).
derecha(pepe,luis).         izquierda(lola,pepe).
derecha(ana,luis).          izquierda(pablo,ana).
derecha(pepe,lola).         izquierda(luis,ana).
derecha(ana,lola).          izquierda(lola,ana).
derecha(ana,pepe).          izquierda(pepe,ana).
```

Como se puede observar, la lista de combinaciones es bastante larga teniendo en cuenta que el mundo considerado solamente tiene cinco elementos. Con más elementos el listado sería realmente costoso de implementar. Para evitar

este coste se utilizan las reglas recursivas. Estas reglas definen un predicado haciendo referencia a ese mismo predicado. La idea es que el compilador busque la solución a determinada pregunta utilizando recursivamente el predicado hasta que llegue a una condición límite. Por esta razón, lo usual es que las reglas recursivas estén formadas por dos cláusulas como mínimo, de tal forma que una de ellas indique la condición límite y las demás sean las cláusulas propiamente recursivas que definen el predicado haciendo referencia a él mismo. Por ejemplo, manteniendo en la base de datos el predicado original `inm_derecha`, se podría definir el predicado `derecha` de la siguiente forma, siendo la primera cláusula la condición límite:

```
derecha(X,Z):-
      inm_derecha(X,Z).
derecha(X,Z):-
      inm_derecha(X,Y),
      derecha(Y,Z).
```

Si se realiza la consulta: ¿qué objetos están a la derecha de otros?, la respuesta serán las diez combinaciones de elementos que satisfacen esa relación:

```
?- derecha(X,Y).

X = luis        X = ana         X = pepe        X = ana
Y = pablo       Y = pepe        Y = pablo       Y = pablo

X = lola        X = lola        X = ana
Y = luis        Y = pablo       Y = lola

X = pepe        X = pepe        X = ana
Y = lola        Y = luis        Y = luis
```

2.5.- Listas

Supongamos que queremos que el número de elementos del mundo considerado varíe según la pregunta realizada. Esto significa dejar en manos del usuario del programa el número de elementos de dicho mundo. Una forma inmediata de hacer esto sería añadiendo o eliminando elementos en la base de datos. Si se contempla la posibilidad de variar el número de elementos en cada consulta, ello implica diseñar un programa nuevo cada vez. Éste es un trabajo excesivamente tedioso que puede ser realizado de una forma más cómoda con el uso de listas. Una lista no es más que una secuencia de ítems denotada mediante corchetes ("`[`", "`]`"). Una lista puede ser vacía (se denota como "`[]`") o puede tener elementos que irán separados por comas. Los siguientes serían ejemplos de listas:

```
[a,b,c,d,e]
['my','love','she´s','like','some','raven']
[mi,amor,es,como,un,cuervo]
[1,3,2.25,430,0.1]
[tren]
```

Una lista no vacía se divide en cabeza y cola. La cabeza es el primer elemento de la lista, la cola es el resto de la lista. Para denotar la separación entre cabeza y cola se utiliza la barra vertical "`|`" de tal forma que la cabeza aparece a la izquierda de la barra y la cola a la derecha. Debe quedar claro que mientras que las cabezas son elementos, las colas siempre son listas, las cuales, a su vez, también se componen de cabeza y cola. Asimismo, se pueden indicar varios

elementos del principio de la lista, como por ejemplo `[a,b|[c,d,e]]`, y se contempla la posibilidad de que una lista pueda ser un elemento de otra lista, como en el ejemplo `[['my','love'],['she´s','like','some','raven']]`. Al trabajar con listas se podrán utilizar variables, tanto para los elementos como para la cabeza o la cola; estas variables serán necesarias a la hora de diseñar reglas con listas.

Uno de los predicados que usualmente se definen utilizando listas es el predicado `miembro`, que indica si un elemento pertenece o no a una lista. Posee dos argumentos, un elemento y una lista. El predicado se define de la siguiente manera:

1. Un elemento es miembro de una lista si es su cabeza.
2. Un elemento es miembro de una lista si es miembro de su cola.

En Prolog:

```
miembro(A,[A|_]).
miembro(A,[_|C]):-
      miembro(A,C).
```

A partir de este predicado se podría definir el predicado `derecha` para dar cuenta de una relación análoga a la que aparecía en apartados anteriores, con la diferencia que en este caso el número de elementos sobre los que definimos la relación puede variar en cada consulta. El predicado se define de la siguiente manera:

1. *A* está a la derecha de *B* en una lista cuya cabeza es *B* si:
   - *A* es miembro de la cola de la lista.
2. *A* está a la derecha de *B* en una lista si:

- *A* está a la derecha de *B* en la cola de esa lista.

En Prolog:

```
derecha(A,B,[B|D]):-
      miembro(A,D).
derecha(A,B,[_|D]):-
      derecha(A,B,D).
```

El predicado `izquierda` se puede definir en función del predicado `derecha` con una regla que indique que *A* está a la izquierda de *B* en una lista si *B* está a la derecha de *A* en esa lista:

```
izquierda(A,B,C):-
      derecha(B,A,C).
```

Si se utilizase el predicado `derecha` para definir el predicado `entre`, la regla tendría la siguiente estructura:

A está entre *B* y *C* en una lista si:

- *A* está a la derecha de *B* en dicha lista y
- *C* está a la derecha de *A* en dicha lista.

```
entre(A,B,C,D):-
      derecha(A,B,D),
      derecha(C,A,D).
```

Si preguntamos qué objetos están entre otros dos en la lista propuesta, tenemos:

```
      ?- entre(X, Y, Z, [pablo,luis,lola,pepe,ana,rosa]).

X = luis        X = lola        X = lola        X = ana
Y = pablo       Y = pablo       Y = luis        Y = luis
Z = lola        Z = ana         Z = pepe        Z = rosa

X = luis        X = lola        X = lola        X = pepe
Y = pablo       Y = pablo       Y = luis        Y = lola
Z = pepe        Z = rosa        Z = ana         Z = ana

...             ...             ...             ...
```

Haciendo hincapié en la idea de que no existe una única forma de diseñar un programa, se puede tomar otra alternativa para definir en Prolog los predicados `derecha`, `izquierda` y `entre` utilizando un predicado que proporcione el número de orden que determinado elemento tiene en la lista. Dados un elemento y una lista en la que figura dicho elemento, el predicado `numorden` indica la posición que posee el elemento en la lista. Además, dada una lista y un número *N*, proporciona a su vez el elemento que está en la posición *N*. El predicado consta de tres argumentos y se define de la siguiente manera:

1. Si el elemento es la cabeza de la lista, su posición es la 1.
2. Un elemento está en la posición *Num* de una lista si:
   - Ese elemento está en la posición *N* de la cola de la lista y
   - *Num* es igual a *N*+1. (Más adelante, al hablar de los predicados predefinidos, se indicará cómo se utilizan las operaciones aritméticas en Prolog).

El código Prolog correspondiente es:

```
numorden(A,[A|C],1).
numorden(A,[B|C],Num):-
      numorden(A,C,N),
      Num is N+1.
```

Al preguntar en qué posición está el elemento "g" de la lista "[a, b, c, d, e, f, g, h]" el compilador responde de la siguiente manera:

```
?- numorden(g,[a,b,c,d,e,f,g,h],X).

X = 7
```

Al poder obtener el número de orden que un elemento tiene en una lista, solamente hay que percatarse de que si el número de orden de un elemento *a* es

mayor que el de otro *b* entonces *a* está a la derecha de *b*. Luego el predicado `derecha` puede definirse de la siguiente forma:

A está a la derecha de *B* en una lista si:

- El número de orden de *A* en la lista es *X* y
- El número de orden de *B* en la lista es *Y* y
- *X* es mayor que *Y*. (Las comparaciones aritméticas en Prolog serán tratadas más adelante al hablar de los predicados predefinidos).

En Prolog:

```
derecha(A,B,C):-
      numorden(A,C,X),
      numorden(B,C,Y),
      X > Y.
```

Análogamente, podrían definirse el predicado `izquierda` indicando que si el número de orden de un elemento *a* es menor que el de otro *b* entonces *a* está a la izquierda de *b* y el predicado `entre` indicando que si el número de orden de un elemento *a* es mayor que el de otro *b* y el de otro elemento *c* es mayor que el de *a*, entonces *a* está entre *b* y *c*. Cualquier consulta realizada con los predicados `derecha`, `izquierda` y `entre` así definidos, coincide con las realizadas para los mismos predicados cuando fueron definidos haciendo uso del predicado `miembro`.

2.6.- Algunos predicados Prolog

A continuación se presentarán una serie de predicados Prolog, algunos de los cuales serán utilizados repetidas veces en capítulos posteriores. Uno de ellos

es, el predicado `min` que calcula el mínimo de dos números. Sus dos primeros argumentos son números, el tercer argumento es el mínimo entre ellos:

```
min(Numero1, Numero2, Minimo)
```

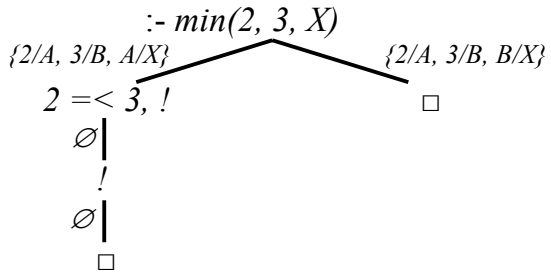
El predicado se define de la manera habitual en Prolog, y en él se emplea el recurso del corte al que ya se ha hecho referencia en este capítulo:

1. El mínimo de dos números es el primero de ellos si éste es menor o igual al segundo.
2. En otro caso, el mínimo de dos números es el segundo de ellos.

El código es el siguiente:

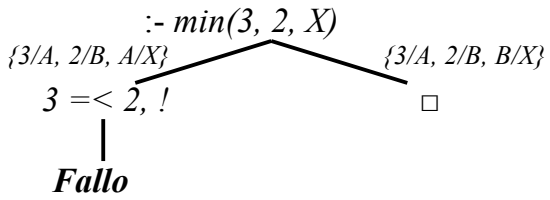
```
min(A,B,A):-
      A=<B,
      !.
min(A,B,B).
```

Para ver el funcionamiento del corte, se puede emplear, a modo de ejemplo, el árbol-SLD del objetivo :- *min(2, 3, X)* con el código correspondiente al predicado `min`:



Con una búsqueda prioritaria en profundidad la rama de la izquierda conduciría a éxito y la respuesta del compilador sería *X = 2*. Si no se hubiese puesto el corte en esa rama, el compilador seguiría buscando soluciones en la rama de la derecha y proporcionaría como respuesta también *X = 3*, pero se sabe de antemano que si

encuentra solución en una rama no la debe encontrar en la otra. Si el objetivo fuese :- *min(3, 2, X)*, el árbol sería el siguiente:



La rama de la izquierda conduciría a fallo y la de la derecha sería la única que conduciría a éxito con la respuesta *X = 2*.

El predicado min podría haberse definido sin utilizar el corte haciendo explícitas las condiciones que, de modo exclusivo, deben cumplirse en cada rama del árbol:

```
min(A,B,A):-
      A=<B.
min(A,B,B):-
      B<A.
```

El predicado `maxim` calcula el máximo de dos números y se define de la siguiente forma:

1. El máximo de dos números es el primero de ellos si éste es mayor o igual al segundo.
2. En otro caso, el máximo de dos números es el segundo de ellos.

En Prolog:

```
maxim(A,B,A):-
      A>=B,
      !.
maxim(A,B,B).
```

Un predicado muy común en el manejo de listas es `num`, que indica el número de elementos de una lista. Posee dos argumentos, el primero es una lista y el segundo el número de elementos de dicha lista:

`num(Lista,Numero_de_Elementos)`

El predicado se define de la siguiente forma:

1. El número de elementos de la lista vacía es 0.
2. *Num* es el número de elementos de una lista si:
   - *N* es el número de elementos de su cola y
   - *Num* es igual a *N*+1.

En Prolog:

```
num([],0).
num([A|B],Num):-
      num(B,N),
      Num is N+1.
```

El predicado `concatenar` proporciona la concatenación de dos listas y posee tres argumentos, los dos primeros son las listas a concatenar y el tercero el resultado de la concatenación:

`concatenar(Lista1,Lista2,Resultado)`

El predicado se define de la siguiente manera:

1. La concatenación de la lista vacía con cualquier otra lista es esa otra lista.
2. La concatenación de una lista cuya cabeza es *A* y otra lista *C* es [*A*|*D*] si:
   - La concatenación de la cola de la primera lista y *C* es *D*.

```
concatenar([],X,X).
```

```
concatenar([A|B],C,[A|D]):-
      concatenar(B,C,D).
```

El predicado `union` establece el resultado de la unión de dos listas y posee tres listas como argumentos:

`union(Lista1, Lista2, ListaResultado)`

El predicado se define de la siguiente manera:

1. La unión de la lista vacía con cualquier lista, resulta ser la misma lista.
2. Si la primera lista no es vacía, compruébese que la cabeza pertenece a la segunda, en caso de ser así, deténgase la búsqueda con este elemento y pásese al siguiente elemento de la primera lista.
3. Si la cabeza de la primera lista no pertenece a la segunda lista añádase dicho elemento a la lista del resultado.
4. Repítase el proceso con los demás elementos hasta agotar la primera lista. La unión será el resultado de añadir a la segunda lista aquellos elementos de la primera que no estaban en la segunda.

El código es el siguiente:

```
union([],X,X).
union([A|B],Y,Z):-
      miembro(A,Y),
      !,
      union(B,Y,Z).
union([A|B],Y,[A|Z]):-
      union(B,Y,Z).
```

El predicado `inter` establece el resultado de la intersección de dos listas y posee igualmente tres listas como argumentos:

`inter(Lista1, Lista2, ListaResultado)`

El predicado se define de la siguiente manera:

1. La intersección de la lista vacía con cualquier lista, resulta ser la lista vacía.
2. Si la primera lista no es vacía, compruébese que la cabeza pertenece a la segunda, en caso de ser así, añádase la cabeza a la lista del resultado.
3. Si la cabeza de la primera lista no pertenece a la segunda lista no se añada dicho elemento.
4. Repítase el proceso con los demás elementos hasta agotar la primera lista. La intersección será el resultado de ir añadiendo en una lista vacía aquellos elementos de la primera lista que también están en la segunda.

El código es el siguiente:

```
inter([],_,[]).
inter([A|B],Y,[A|Z]):-
      miembro(A,Y),
      !,
      inter(B,Y,Z).
inter([_|B],Y,Z):-
      inter(B,Y,Z).
```

2.7.- Predicados predefinidos

Normalmente en los compiladores de Prolog vienen predefinidos algunos predicados. En este apartado se dará cuenta de aquellos que serán utilizados posteriormente, teniendo en cuenta que muchos de ellos pueden no figurar como tales en determinadas versiones de algunos compiladores.

Uno de estos predicados es `findall`, que recoge en una lista todas las instancias de una variable en un objetivo. Posee tres argumentos:

`findall(Variable, Objetivo, Lista)`

El primer argumento es una variable, el segundo es un objetivo que contiene dicha variable y el tercero, la lista de todas las instancias que posee la variable en el objetivo. Por ejemplo, habiendo definido el predicado `padre` de la siguiente manera:

```
padre(pepe,lola).
padre(pepe,ana).
padre(pepe,rosa).
```

La siguiente consulta con el predicado `findall` proporcionará la lista de todas las instancias de la variable *X* en el objetivo `padre(pepe,X)`:

```
?- findall(X, padre(pepe,X), Lista).

Lista = [lola,ana,rosa]
```

Otro predicado predefinido usual es `name`, que proporciona la lista de códigos ASCII de una expresión cualquiera, o viceversa, la expresión correspondiente a una lista de códigos ASCII. Este predicado posee dos argumentos, el primero de ellos es una expresión cualquiera y el segundo es la lista de códigos ASCII relativos a cada uno de los símbolos de la expresión.

`name(Expresion, Lista_de_codigos_ASCII)`

Es necesario instanciar uno de los argumentos de este predicado para obtener respuesta. Por ejemplo:

```
?- name('cigüeña', ASCII).

ASCII = [99,105,103,-4,101,-15,97]

?- name(Expresion, [99,105,103,-4,101,-15,97]).

Expresion = cigüeña
```

La expresión “cigüeña” debe aparecer entre comillas simples debido a que contiene caracteres como “ü” y “ñ”. En caso contrario el compilador daría un error al compilar el programa.

El predicado `write` escribe cualquier cadena de caracteres en la pantalla y posee como único argumento dicha cadena de caracteres:

```
write(Cadena de caracteres)
```

Si este predicado figura en una regla, cuando ésta sea ejecutada, la expresión que el predicado `write` contiene como argumento será escrita en la pantalla.

Ligado al predicado `write` está el predicado `nl`, que no hace más que escribir una nueva línea en la pantalla. Es útil cuando se quiere que la expresión que es el argumento de un `write` comience en la línea siguiente.

Cuando el programa requiere que el usuario le proporcione determinada información, el programador puede utilizar el predicado `read`. Este predicado posee como único argumento una variable:

```
read(Variable)
```

Si su argumento no es una variable el compilador indicará un error. Cuando este predicado figura en una regla y ésta es ejecutada, el programa quedará a la espera de que el usuario introduzca la información con la que se instanciará la variable que posee como argumento. Esto es indicado por el sistema con la expresión “|:”. A continuación el usuario deberá escribir la información seguida de un punto. El siguiente programa ejemplifica estos tres predicados predefinidos:

```
procesa:-
      nl,write('¿Quién escribió los siguientes versos?'),nl,nl,
      write('My love she´s like some raven'),nl,
      write('at my window with a broken wing'),nl,nl,
      read(X),
      (X = 'Bob Dylan';X = 'Robert Zimmerman').
```

Si se realiza la consulta:

```
procesa.
```

el sistema escribirá en la pantalla:

```
¿Quién escribió los siguientes versos?

My love she´s like some raven
at my window with a broken wing

|:
```

Si el usuario introduce ahora:

```
'Bob Dylan'.
```

o

```
'Robert Zimmerman'.
```

el sistema responderá afirmativamente. En caso contrario lo hará negativamente.

Por último, conviene dar cuenta del tratamiento de operaciones aritméticas con Prolog. Para realizar dichas operaciones se emplea la expresión "`is`", que hace las veces del símbolo "=". Las expresiones correspondientes a cada operación aritmética básica son "`+`" para la suma, "`-`" para la resta, "`*`" para el producto, "`/`" para la división real y "`div`" para la división entera (a veces, también se utiliza "//"). Para la raíz cuadrada se utiliza el predicado `sqrt`, que posee como único argumento un número:

```
sqrt(Numero)
```

Por ejemplo, en los objetivos:

```
X is sqrt(4).
X is 2+3.
X is 2-3.
X is 2*3.
X is 2 div 3.
X is 2/3.
```

la variable *X* se instanciará respectivamente con los números 2, 5, -1, 6, 0, 0.666667.

Las comparaciones aritméticas como el mayor o igual (≥), mayor estricto (>), menor o igual (≤) y menor estricto (<), se expresan en Prolog respectivamente como "`>=`", "`>`", "`=<`" y "`<`". De esta forma, el compilador responderá "`yes`" con las siguientes consultas:

```
7 >= 7.
7 >= 5.
7 > 5.
7 =< 7.
7 =< 8.
7 < 8.
```


2.8.- Gramáticas

Una gramática es una descripción de la estructura que subyace a una oración de un lenguaje. Si se implementa una gramática en Prolog, al realizar una consulta, el compilador es capaz de reconocer qué oraciones son gramaticales y qué oraciones no lo son. Utilizando variables en una consulta el compilador pasa de reconocer a generar aquellas oraciones que son gramaticales. De esta forma el compilador puede funcionar como un reconocedor gramatical o como un generador de oraciones de un lenguaje dado.

Una gramática consta de un vocabulario terminal y un vocabulario no terminal, y viene definida por una serie de reglas de producción. Un ejemplo de gramática podría ser el siguiente:

$S \rightarrow ab$
$S \rightarrow aSb$

donde *S* es el vocabulario no terminal y *a*, *b* el vocabulario terminal. La primera regla indica que la secuencia *ab* es una oración del lenguaje. La segunda regla dice que se puede formar una oración del lenguaje a partir de otra oración añadiéndole una *a* a la izquierda y una *b* a la derecha. Como puede observarse, esta segunda regla es recursiva. Para que esta gramática genere oraciones se parte del símbolo no terminal *S* empleando las reglas hasta que solamente queden símbolos terminales. Si aplicamos la primera regla a partir de *S* obtenemos *ab*. Como esta expresión sólo consta de símbolos no terminales ya no se pueden aplicar más reglas sobre ella y la secuencia *ab* es una oración del lenguaje. Si aplicamos la segunda regla a partir de *S* obtenemos *aSb*, expresión que posee un símbolo no terminal. Sobre dicho símbolo no terminal se puede aplicar la primera regla, obteniendo la oración *aabb* o se puede aplicar de nuevo la segunda, obteniendo *aaSbb*, sobre la que una vez más, se puede aplicar cualquiera de las dos reglas. Por lo tanto, esta gramática genera oraciones como *ab*, *aabb*, *aaabbb*; en general cadenas del tipo $a^n b^n$, para $n \geq 1$.

Para implementar una gramática en Prolog es necesario tener en cuenta que las oraciones se representan normalmente como listas. Los símbolos no terminales figuran como predicados y los terminales como constantes. Las reglas

gramaticales se implementan, como se verá más adelante, en términos de diferencia de listas, de tal forma que toda regla gramatical deberá tener dos argumentos como mínimo, uno para cada lista con la que se hace la diferencia. Para implementar la regla no recursiva de la gramática anterior se utiliza una cláusula Prolog donde el no terminal figura como predicado y los terminales aparecen en la cabeza de la lista que figura en el primer argumento:

```
s([a,b|Resto],Resto).
```

Para implementar la regla recursiva, el nombre del predicado continúa siendo el símbolo no terminal. Todo lo que la gramática debe generar a la izquierda del terminal (*a* en este caso), debe colocarse en la cabeza de la lista del primer argumento del predicado que aparece antes del símbolo ":-". Todo lo que la gramática debe generar a la derecha del terminal (*b* en este caso), debe colocarse en la cabeza de la lista del segundo argumento del predicado que aparece después del símbolo ":-":

```
s([a|Lista],Resto):-
      s(Lista,[b|Resto]).
```

Si se hace la siguiente consulta ¿es *aabb* una oración correcta?, que se suele expresar con:

```
?- s([a,a,b,b],[]).
```

aunque también sería válida si a la derecha de la primera lista y en la segunda se le añade la misma serie de elementos (por ejemplo, `s([a,a,b,b,r,s,t],[r,s,t])` sería la misma consulta), el compilador responderá:

```
yes
```

Si se pregunta ahora ¿es *aab* una oración correcta?:

```
?- s([a,a,b],[]).
```

El compilador responde:

```
no
```

Una vez más, si se realiza la siguiente consulta; ¿cuáles son las oraciones que genera la gramática?:

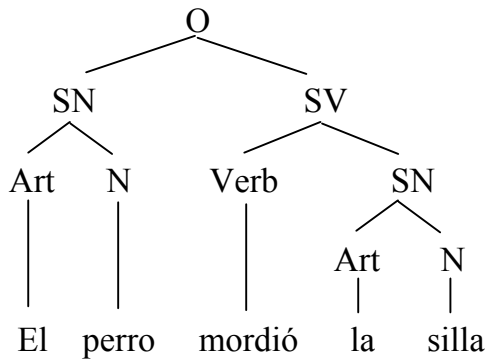
```
?- s(X,[]).
```

El compilador responde:

```
X = [a,b]
X = [a,a,b,b]
X = [a,a,a,b,b,b]
...
```

La gramática anterior genera oraciones para un lenguaje artificialmente creado. Tratar de generar oraciones de un lenguaje natural es algo más complejo dado que, en ocasiones, el lenguaje natural no se rige por reglas estrictas, sino que está plagado de excepciones. Por eso, cuando se habla de gramáticas para el lenguaje natural se debe tener la cautela de indicar que no se pretende llegar a generar todas las oraciones correctas en un lenguaje natural, sino un conjunto de ellas útil para determinado propósito.

Supongamos la siguiente oración del español: “El perro mordió la silla”. Un primer paso para implementar en Prolog una gramática que genere oraciones de este tipo es hacer su análisis sintáctico, que vendría dado por el siguiente árbol de estructura de frase:



Este análisis sintáctico permite obtener las reglas de la gramática, que en formato de reglas de estructura de frase serían las siguientes:

O → *SN SV*
SN → *Art N*
SV → *Verb SN*
Art → *el*
Art → *la*
N → *perro*
N → *silla*
Verb → *mordió*

En la última fila del árbol figura el vocabulario terminal; el resto de los nodos están ocupados por el vocabulario no terminal. Si analizamos el árbol de arriba a abajo, la primera regla se leería diciendo que una oración (O) está formada por un sintagma nominal (SN) y un sintagma verbal (SV). Esto se expresa en Prolog de la siguiente forma:

```
o(Lista1,Resto):-
      sn(Lista1,Lista2),
      sv(Lista2,Resto).
```

Insistiendo en la idea de que las gramáticas se implementan mediante diferencia de listas, esta regla podría interpretarse diciendo que una oración es el resultado de eliminar en `Lista1` la lista `Resto` si un sintagma nominal es el resultado de eliminar en `Lista1` la lista `Lista2` y un sintagma verbal es el

resultado de eliminar en `Lista2` la lista `Resto`. Dicho de otra forma, se trata de indicar que lo que está al principio de una lista (`Lista1`) es una oración seguido de otra lista `Resto`, si lo que está al principio de `Lista1` es un sintagma nominal seguido de `Lista2` y lo que está al principio de `Lista2` es un sintagma verbal seguido de `Resto`. Por ejemplo, la diferencia entre la expresión [el, perro, mordió, la, silla] y [] será una oración si existe otra expresión `Lista2` de tal forma que la diferencia entre [el, perro, mordió, la, silla] y `Lista2` es un sintagma nominal y la diferencia entre `Lista2` y [] es un sintagma verbal. Como se podrá comprobar cuando sea presentada la gramática completa, la expresión `Lista2` corresponderá a la lista [mordió, la, silla]. De esta forma se verificará que:

```
o([el,perro,mordió,la,silla],[]):- %([el,perro,mordió,la,silla] es una oración)
      sn([el,perro,mordió,la,silla],[mordió,la,silla]),
                                          %([el,perro] es un sintagma nominal y)
      sv([mordió,la,silla],[]).%([mordió,la,silla] es un sintagma verbal)
```

Un sintagma nominal (SN) está formado por un artículo (Art) y un nombre (N):

```
sn(Lista1,Resto):-
      art(Lista1,Lista2),
      n(Lista2,Resto).
```

Un sintagma verbal (SV) está formado por un verbo (Verb) y un sintagma nominal (SN):

```
sv(Lista1,Resto):-
      verb(Lista1,Lista2),
      sn(Lista2,Resto).
```

Respecto al vocabulario terminal se precisa una cláusula para cada palabra. Por ejemplo, para señalar que "el" y "la" son artículos se utilizan las siguientes cláusulas:

```
art([el|Resto],Resto).
art([la|Resto],Resto).
```

Para señalar que "perro" y "silla" son nombres:

```
n([perro|Resto],Resto).
n([silla|Resto],Resto).
```

Para señalar que "mordió" es un verbo:

```
verb(['mordió'|Resto],Resto).
```

Si se hace la siguiente pregunta; ¿es una oración "el perro mordió la silla"?, expresada en Prolog de la siguiente forma:

```
?- o([el,perro,mordió,la,silla],[]).
```

El compilador responde:

```
yes
```

Si se consulta cuáles son las oraciones que genera la gramática:

```
?- o(X,[]).
```

```
X = [el,perro,mordió,el,perro]   ...
X = [el,perro,mordió,el,silla]   X = [la,silla,mordió,el,silla]
X = [el,perro,mordió,la,perro]   X = [la,silla,mordió,la,perro]
...                              X = [la,silla,mordió,la,silla]
```

Como se puede comprobar, algunas de las oraciones generadas tienen problemas de concordancia entre el género del artículo y el del nombre. Así, aparecen expresiones como "la perro" o "el silla". Los problemas de concordancia se pueden evitar modificando la gramática expresada en Prolog. Lo usual en estos casos es incluir el género, el número, el tiempo, la persona, etc. como argumentos en los predicados que sea preciso. De esta forma, se puede

exigir en las reglas que determinados componentes de una oración concuerden en género o número; esto es algo de lo que no dan cuenta las reglas de estructura de frase. Por ejemplo, para que una oración sea correcta, el sintagma nominal y el sintagma verbal deben concordar en número. Esto se expresa en Prolog con el argumento adicional `Numero`:

```
o(Lista1,Resto):-
      sn(_,Numero,Lista1,Lista2),
      sv(_,_,Numero,_,Lista2,Resto).
```

Los componentes de un sintagma nominal, artículo y nombre, deben concordar en género y número:

```
sn(Genero,Numero,Lista1,Resto):-
      art(Genero,Numero,Lista1,Lista2),
      n(Genero,Numero,Lista2,Resto).
```

El tiempo, modo, número y persona del sintagma verbal coincide con el tiempo, modo, número y persona del verbo:

```
sv(Tiempo,Modo,Numero,Persona,Lista1,Resto):-
      verb(Tiempo,Modo,Numero,Persona,Lista1,Lista2),
      sn(_,_,Lista2,Resto).
```

El artículo “el” es de género masculino y de número singular:

```
art(masculino,singular,[el|Resto],Resto).
```

El artículo “la” es de género femenino y de número singular:

```
art(femenino,singular,[la|Resto],Resto).
```

El nombre “perro” es de género masculino y de número singular:

```
n(masculino,singular,[perro|Resto],Resto).
```

El nombre “silla” es de género femenino y de número singular:

```
n(femenino,singular,[silla|Resto],Resto).
```

El verbo "mordió" está en el tiempo pretérito indefinido del modo indicativo y es la tercera persona del singular:

```
verb(preterito_perfecto,indicativo,singular,tercera,['mordió'|Re
sto],Resto).
```

Aunque para el ejemplo no es relevante la información sobre el tiempo, el modo o la persona a la hora de establecer concordancia entre los componentes de la oración, se puede observar que los artículos y nombres de las oraciones que ahora genera la gramática concuerdan en género y número:

```
?- o(X,[]).

X = [el,perro,mordió,el,perro]   X = [la,silla,mordió,el,perro]
X = [el,perro,mordió,la,silla]   X = [la,silla,mordió,la,silla]
```

Las oraciones generadas son gramaticalmente correctas aunque algunas de ellas no sean semánticamente correctas. Es evidente que un objeto inanimado como una silla no puede realizar la acción de morder; sin embargo las dos últimas oraciones predican tal acción. Se puede establecer una restricción semántica, para una situación de uso convencional del lenguaje, que indique que sólo los objetos animados pueden realizar la acción de morder; sin embargo, no tendría mucho sentido establecer dicha restricción si se considera, por ejemplo, un uso poético del lenguaje. Un programa Prolog que dé cuenta de esta restricción semántica no es difícil de escribir, pero se necesita previamente que la gramática sea capaz de asociar un significado, valor semántico o estructura semántica (se utilizarán las tres expresiones como equivalentes cuando se hable de este procedimiento dentro del tratamiento de gramáticas en Prolog) a cada oración que genera. El valor semántico se obtiene en función de los valores

semánticos de los componentes de la oración. De esta forma, y continuando con el ejemplo anterior, el valor semántico o significado asociado a un nombre es el propio nombre como aparece expresado en el primer argumento de las siguientes cláusulas:

```
n(perro,masculino,singular,[perro|Resto],Resto).
n(silla,femenino,singular,[silla|Resto],Resto).
```

El valor semántico de un sintagma nominal será el mismo que el valor semántico del nombre que es su componente:

```
sn(Sgdo,Genero,Numero,Lista1,Resto):-
      art(Genero,Numero,Lista1,Lista2),
      n(Sgdo,Genero,Numero,Lista2,Resto).
```

El valor semántico para el verbo "morder" es un predicado cuyos argumentos serán instanciados con los valores semánticos de otras partes de la oración (a este proceso se le suele denominar herencia). Para indicar esto en la cláusula se incluyen tres argumentos: el primero es el predicado "morder(Mordedor,Mordido)", cuyos argumentos figuran como variables; el segundo argumento es "Mordedor" que es el primer argumento del predicado "morder", y el tercer argumento es "Mordido" que es el segundo argumento del predicado "morder". El código es el siguiente:

```
verb(morder(Mordedor,Mordido),Mordedor,Mordido,preterito_perfect
o,indicativo,singular,tercera,['mordió'|Resto],Resto).
```

Las variables "Mordedor" y "Mordido" serán instanciadas con los valores semánticos de los nombres en el resto del análisis gramatical.

El valor semántico de un sintagma verbal es el mismo que el del verbo (en el ejemplo, "morder(Mordedor,Mordido)") pero con el segundo argumento

instanciado por el valor semántico del sintagma nominal que es componente del sintagma verbal:

```
sv(Sgdo,Mordedor,Tiempo,Modo,Numero,Persona,Lista1,Resto):-
      verb(Sgdo,Mordedor,Mordido,Tiempo,Modo,Numero,Persona,
            Lista1,Lista2),
      sn(Mordido,_,_,Lista2,Resto).
```

Como segundo argumento del sintagma verbal se pone el segundo argumento del verbo ("Mordedor") que será instanciado con el valor semántico del sintagma nominal componente de la oración, la cual tendrá como valor semántico el mismo que el del verbo pero con las variables "Mordedor" y "Mordido" ya instanciadas:

```
o(Sgdo,Lista1,Resto):-
      sn(Mordedor,_,Numero,Lista1,Lista2),
      sv(Sgdo,Mordedor,_,_,Numero,_,Lista2,Resto).
```

Ahora el programa tiene capacidad para asociar un valor semántico o significado a cada una de las oraciones que genera. Con la siguiente consulta se obtienen todas las oraciones con sus correspondientes valores semánticos:

```
?- o(S,X,[]).

S = morder(perro,perro)          S = morder(silla,perro)
X = [el,perro,mordió,el,perro]   X = [la,silla,mordió,el,perro]

S = morder(perro,silla)          S = morder(silla,silla)
X = [el,perro,mordió,la,silla]   X = [la,silla,mordió,la,silla]
```

Una vez hecho esto, sólo hace falta establecer la restricción semántica que indica que un objeto inanimado no puede realizar la acción de morder. La regla sería "Una entidad muerde a otra si la primera es animada":

```
restriccion(morder(Mordedor,Mordido)):-
      animado(Mordedor).
```

Ahora es necesario indicar que "perro" es un objeto animado:

```
animado(perro).
```

Con el predicado `oracion` se incorpora esta restricción de la siguiente forma "*X* es una oración si es generada por la gramática, su significado es *S* y *S* cumple la restricción":

```
oracion(X):-
      o(S,X,[]),
      restriccion(S).
```

Con la siguiente consulta obtenemos todas las oraciones que es capaz de generar la gramática restringida semánticamente:

```
?- oracion(X).

X = [el,perro,mordió,el,perro]
X = [el,perro,mordió,la,silla]
```

Hay una forma de operar con gramáticas cuya consulta resulta mucho más natural para un usuario. Consiste en que el programa proporcione oraciones correctas del español y no listas de palabras. Para ello es necesario que la gramática genere los códigos ASCII correspondientes a cada palabra. En el siguiente cuadro figuran los códigos ASCII de las palabras de la gramática propuesta como ejemplo:

Palabra	**Código ASCII**
el	[101,108]
perro	[112,101,114,114,111]
mordió	[109,111,114,100,105,-13]
la	[108,97]
silla	[115,105,108,108,97]
Espacio en blanco	[32]

Para que el programa funcione de esta forma, en el código hay que sustituir las palabras por sus correspondientes códigos ASCII. Las siguientes serían las cláusulas que corresponden a los terminales del ejemplo anterior:

```
art(masculino,singular,[101,108|Resto],Resto).
art(femenino,singular,[108,97|Resto],Resto).
n(perro,masculino,singular,[112,101,114,114,111|Resto],Resto).
n(silla,femenino,singular,[115,105,108,108,97|Resto],Resto).
verb(morder(Mordedor,Mordido),Mordedor,Mordido,preterito_perfect
o,indicativo,singular,tercera,[109,111,114,100,105,-13|Resto],
Resto).
```

El espacio en blanco debe ir entre el artículo y el nombre del sintagma nominal y antes y después del verbo. Con lo que las reglas correspondientes quedarían de la siguiente forma:

```
sn(Sgdo,Genero,Numero,Lista1,Resto):-
      art(Genero,Numero,Lista1,[32|Lista2]),
      n(Sgdo,Genero,Numero,Lista2,Resto).
sv(Sgdo,Mordedor,Tiempo,Modo,Numero,Persona,[32|Lista1],Resto):-
      verb(Sgdo,Mordedor,Mordido,Tiempo,Modo,Numero,Persona,
            Lista1,[32|Lista2]),
      sn(Mordido,_,_,Lista2,Resto).
```

Utilizando el predicado `name` aludido anteriormente, se puede modificar el predicado `oracion` del ejemplo anterior de la siguiente forma:

```
oracion(X):-
      o(S,Y,[]),
      restriccion(S),
      name(X,Y).
```

Ante la siguiente consulta:

```
?- oracion(X).
```

El compilador responde:

```
X = el perro mordió el perro
X = el perro mordió la silla
```

2.8.1.- Un ejemplo para un fragmento breve del español

Poniendo en práctica todo lo visto en este apartado, se propondrá una gramática en la que se conjuga un verbo, se halla el plural de los nombres

comunes y se les hace concordar con los artículos determinados o indeterminados de un sintagma nominal. Todo ello atendiendo a cuestiones ortográficas, de tal forma que las oraciones generadas son completamente gramaticales en español.

Una oración del español cuyo verbo esté en indicativo se compone de un sujeto y un predicado que deben concordar en persona y número o bien se compone de un predicado solamente. Si el verbo está en subjuntivo se compone de una interjección, un sujeto y un predicado, debiendo concordar estos dos últimos en persona y número, o bien se compone de una interjección y un predicado. Un sujeto se compone de un pronombre. Un predicado de un verbo y un complemento directo o de un verbo solamente. La palabra "ojalá" (cuya lista de códigos ASCII es [111,106,97,108,-31]) es una interjección. Un complemento directo está compuesto por un artículo y un nombre común que concuerdan en género y número, o bien por un nombre común en plural. El siguiente cuadro indica las características y los códigos ASCII de los pronombres personales:

NÚMERO	PERSONA	PRONOMBRE	CÓDIGO ASCII
Singular	Primera	yo	[121,111]
	Segunda	tú	[116,117]
	Tercera	él	[-23,108]
Plural	Primera	nosotros	[110,111,115,111,116,114,111,115]
	Segunda	vosotros	[118,111,115,111,116,114,111,115]
	Tercera	ellos	[101,108,108,111,115]

Un verbo está formado por una raíz y una desinencia si está conjugado en un tiempo simple y por un verbo auxiliar y un participio si está conjugado en un tiempo compuesto. La sucesión de letras "cant" (cuya lista de códigos ASCII correspondiente es [99,97,110,116]) es una raíz (la raíz del verbo cantar). Las

desinencias indican la persona, el número, el tiempo y el modo del verbo. En la siguiente tabla figuran todas las desinencias del verbo cantar:

MODO	TIEMPO	DESINENCIAS
Indicativo	Presente	"o", "as", "a", "amos", "ais", "an"
	P. Imperfecto	"aba", "abas", "aba", "ábamos", "abais", "aban"
	P. Indefinido	"é", "aste", "ó", "amos", "asteis", "aron"
	F. Imperfecto	"aré", "arás", "ará", "aremos", "aréis", "arán"
	Cond. Simple	"aría", "arías", "aría", "aríamos", "aríais", "arían"
Subjuntivo	Presente	"e", "es", "e", "emos", "éis", "en"
	P. Imperfecto	"ara", "aras", "ara", "áramos", "arais", "aran" "ase", "ases", "ase", "ásemos", "aseis", "asen"
	F. Imperfecto	"are", "ares", "are", "áremos", "areis", "aren"

Los verbos auxiliares del resto de los tiempos verbales figuran en la siguiente tabla:

MODO	TIEMPO	VERBOS AUXILIARES
Indicativo	P. Perfecto	"he", "has", "ha", "hemos", "habéis", "han"
	P. Pluscuamp.	"había", "habías", "había", "habíamos", "habíais", "habían"
	P. Anterior	"hube", "hubiste", "hubo", "hubimos", "hubisteis", "hubieron"
	F. Perfecto	"habré", "habrás", "habrá", "habremos", "habréis", "habrán"
	Cond. Comp.	"habría", "habrías", "habría", "habríamos", "habríais", "habrían"
Subjuntivo	P. Perfecto	"haya", "hayas", "haya", "hayamos", "hayáis", "hayan"
	P. Pluscuamp.	"hubiera", "hubieras", "hubiera", "hubiéramos", "hubierais", "hubieran" "hubiese", "hubieses", "hubiese", "hubiésemos", "hubieseis", "hubiesen"
	F. Perfecto	"hubiere", "hubieres", "hubiere", "hubiéremos", "hubiereis", "hubieren"

La palabra "cantado" es el participio del verbo cantar. Los artículos determinados son "el" (masculino singular), "la" (femenino singular), "los" (masculino plural) y "las" (femenino plural); y los indeterminados "un"

(masculino singular), "una" (femenino singular), "unos" (masculino plural) y "unas" (femenino plural). La palabra "pasodoble" ([112,97,115,111,100,111,98, 108,101]) es un nombre común masculino singular y la palabra "canción" ([99,97,110,99,105,-13,110]) es un nombre común femenino singular. Además, un nombre común en plural se construye añadiendo un sufijo de número a un nombre en singular. Dicho sufijo de número varía según el nombre en singular termine en vocal o consonante. Si el nombre termina en vocal como "pasodoble", el sufijo es "s" ([115]). Si el nombre termina en consonante como "canción", el sufijo es "es". Finalmente, el predicado `oracion` permite pasar de las listas de códigos ASCII de las oraciones generadas por la gramática a las oraciones correctas en español. El programa completo es el siguiente:

```
%GRAMÁTICA PARA UN FRAGMENTO DEL ESPAÑOL

oracion(X):-
      o(Y,[]),
      name(X,Y).

o(Lista1,Resto):-
      sujeto(Persona,Numero,Lista1,[32|Lista2]),
      predicado(Persona,Numero,_,indicativo,Lista2,Resto).
o(Lista1,Resto):-
      predicado(_,_,_,indicativo,Lista1,Resto).
o(Lista1,Resto):-
      interjeccion(Lista1,[32|Lista2]),
      sujeto(Persona,Numero,Lista2,[32|Lista3]),
      predicado(Persona,Numero,_,subjuntivo,Lista3,Resto).
o(Lista1,Resto):-
      interjeccion(Lista1,[32|Lista2]),
      predicado(_,_,_,subjuntivo,Lista2,Resto).

sujeto(Persona,Numero,Lista1,Resto):-
      pronombre(_,Persona,Numero,Lista1,Resto).

predicado(Persona,Numero,Tiempo,Modo,Lista1,Resto):-
      verbo(Persona,Numero,Tiempo,Modo,Lista1,[32|Lista2]),
      complemento_directo(Lista2,Resto).
predicado(Persona,Numero,Tiempo,Modo,Lista1,Resto):-
      verbo(Persona,Numero,Tiempo,Modo,Lista1,Resto).
```

```
interjeccion([111,106,97,108,-31|Resto],Resto).

complemento_directo(Lista1,Resto):-
      articulo(_,Genero,Numero,Lista1,[32|Lista2]),
      nombre(_,Genero,Numero,Lista2,Resto).
complemento_directo(Lista1,Resto):-
      nombre(_,Genero,plural,Lista1,Resto).

pronombre(personal,primera,singular,[121,111|Resto],Resto).
pronombre(personal,segunda,singular,[116,117|Resto],Resto).
pronombre(personal,tercera,singular,[-23,108|Resto],Resto).
pronombre(personal,primera,plural,[110,111,115,111,116,114,111,
      115|Resto],Resto).
pronombre(personal,segunda,plural,[118,111,115,111,116,114,111,
      115|Resto],Resto).
pronombre(personal,tercera,plural,[101,108,108,111,115|Resto],
      Resto).

verbo(Persona,Numero,Tiempo,Modo,Lista1,Resto):-
      raiz(Lista1,Lista2),
      desinencia(Persona,Numero,Tiempo,Modo,Lista2,Resto).
verbo(Persona,Numero,Tiempo,Modo,Lista1,Resto):-
      verbo_auxiliar(Persona,Numero,Tiempo,Modo,Lista1,
            [32|Lista2]),
      participio(Lista2,Resto).

raiz([99,97,110,116|Resto],Resto).

desinencia(primera,singular,presente,indicativo,[111|Resto],
      Resto).
desinencia(segunda,singular,presente,indicativo,[97,115|Resto],
      Resto).
desinencia(tercera,singular,presente,indicativo,[97|Resto],
      Resto).
desinencia(primera,plural,presente,indicativo,[97,109,111,115|
      Resto],Resto).
desinencia(segunda,plural,presente,indicativo,[97,105,115|
      Resto],Resto).
desinencia(tercera,plural,presente,indicativo,[97,110|Resto],
      Resto).

desinencia(primera,singular,'pretérito imperfecto',indicativo,
      [97,98,97|Resto],Resto).
desinencia(segunda,singular,'pretérito imperfecto',indicativo,
      [97,98,97,115|Resto],Resto).
desinencia(tercera,singular,'pretérito imperfecto',indicativo,
      [97,98,97|Resto],Resto).
desinencia(primera,plural,'pretérito imperfecto',indicativo,
      [-31,98,97,109,111,115|Resto],Resto).
desinencia(segunda,plural,'pretérito imperfecto',indicativo,
      [97,98,97,105,115|Resto],Resto).
desinencia(tercera,plural,'pretérito imperfecto',indicativo,
      [97,98,97,110|Resto],Resto).
```

```
desinencia(primera,singular,'pretérito indefinido',indicativo,
      [-23|Resto],Resto).
desinencia(segunda,singular,'pretérito indefinido',indicativo,
      [97,115, 116,101|Resto],Resto).
desinencia(tercera,singular,'pretérito indefinido',indicativo,
      [-13|Resto],Resto).
desinencia(primera,plural,'pretérito indefinido',indicativo,
      [97,109,111,115|Resto],Resto).
desinencia(segunda,plural,'pretérito indefinido',indicativo,
      [97,115,116,101,105,115|Resto],Resto).
desinencia(tercera,plural,'pretérito indefinido',indicativo,
      [97,114,111,110|Resto],Resto).

desinencia(primera,singular,'futuro imperfecto',indicativo,
      [97,114,-23|Resto],Resto).
desinencia(segunda,singular,'futuro imperfecto',indicativo,
      [97,114,-31,115|Resto],Resto).
desinencia(tercera,singular,'futuro imperfecto',indicativo,
      [97,114,-31|Resto],Resto).
desinencia(primera,plural,'futuro imperfecto',indicativo,
      [97,114,101,109,111,115|Resto],Resto).
desinencia(segunda,plural,'futuro imperfecto',indicativo,
      [97,114,-23,105,115|Resto],Resto).
desinencia(tercera,plural,'futuro imperfecto',indicativo,
      [97,114,-31,110|Resto],Resto).

desinencia(primera,singular,'condicional simple',indicativo,
      [97,114,-19,97|Resto],Resto).
desinencia(segunda,singular,'condicional simple',indicativo,
      [97,114,-19,97,115|Resto],Resto).
desinencia(tercera,singular,'condicional simple',indicativo,
      [97,114,-19,97|Resto],Resto).
desinencia(primera,plural,'condicional simple',indicativo,
      [97,114,-19,97,109,111,115|Resto],Resto).
desinencia(segunda,plural,'condicional simple',indicativo,
      [97,114,-19,97,105,115|Resto],Resto).
desinencia(tercera,plural,'condicional simple',indicativo,
      [97,114,-19,97,110|Resto],Resto).

desinencia(primera,singular,presente,subjuntivo,[101|Resto],
      Resto).
desinencia(segunda,singular,presente,subjuntivo,[101,115|Resto],
      Resto).
desinencia(tercera,singular,presente,subjuntivo,[101|Resto],
      Resto).
desinencia(primera,plural,presente,subjuntivo,[101,109,111,115|
      Resto],Resto).
desinencia(segunda,plural,presente,subjuntivo,[-23,105,115|
      Resto],Resto).
desinencia(tercera,plural,presente,subjuntivo,[101,110|Resto],
      Resto).

desinencia(primera,singular,'pretérito imperfecto',subjuntivo,
```

```
      [97,114, 97|Resto],Resto).
desinencia(segunda,singular,'pretérito imperfecto',subjuntivo,
      [97,114, 97,115|Resto],Resto).
desinencia(tercera,singular,'pretérito imperfecto',subjuntivo,
      [97,114, 97|Resto],Resto).
desinencia(primera,plural,'pretérito imperfecto',subjuntivo,
      [-31,114, 97,109,111,115|Resto],Resto).
desinencia(segunda,plural,'pretérito imperfecto',subjuntivo,
      [97,114, 97,105,115|Resto],Resto).
desinencia(tercera,plural,'pretérito imperfecto',subjuntivo,
      [97,114, 97,110|Resto],Resto).
desinencia(primera,singular,'pretérito imperfecto',subjuntivo,
      [97,115, 101|Resto],Resto).
desinencia(segunda,singular,'pretérito imperfecto',subjuntivo,
      [97,115, 101,115|Resto],Resto).
desinencia(tercera,singular,'pretérito imperfecto',subjuntivo,
      [97,115, 101|Resto],Resto).
desinencia(primera,plural,'pretérito imperfecto',subjuntivo,
      [-31,115, 101,109,111,115|Resto],Resto).
desinencia(segunda,plural,'pretérito imperfecto',subjuntivo,
      [97,115, 101,105,115|Resto],Resto).
desinencia(tercera,plural,'pretérito imperfecto',subjuntivo,
      [97,115, 101,110|Resto],Resto).

desinencia(primera,singular,'futuro imperfecto',subjuntivo,
      [97,114,101|Resto],Resto).
desinencia(segunda,singular,'futuro imperfecto',subjuntivo,
      [97,114, 101,115|Resto],Resto).
desinencia(tercera,singular,'futuro imperfecto',subjuntivo,
      [97,114,101 |Resto],Resto).
desinencia(primera,plural,'futuro imperfecto',subjuntivo,
      [-31,114,101, 109,111,115|Resto],Resto).
desinencia(segunda,plural,'futuro imperfecto',subjuntivo,
      [97,114,101, 105,115|Resto],Resto).
desinencia(tercera,plural,'futuro imperfecto',subjuntivo,
      [97,114,101, 110|Resto],Resto).

verbo_auxiliar(primera,singular,'pretérito perfecto',indicativo,
      [104,101|Resto],Resto).
verbo_auxiliar(segunda,singular,'pretérito perfecto',indicativo,
      [104,97,115|Resto],Resto).
verbo_auxiliar(tercera,singular,'pretérito perfecto',indicativo,
      [104,97|Resto],Resto).
verbo_auxiliar(primera,plural,'pretérito perfecto',indicativo,
      [104,101,109,111,115|Resto],Resto).
verbo_auxiliar(segunda,plural,'pretérito perfecto',indicativo,
      [104,97,98,-23,105,115|Resto],Resto).
verbo_auxiliar(tercera,plural,'pretérito perfecto',indicativo,
      [104,97,110|Resto],Resto).

verbo_auxiliar(primera,singular,'pretérito pluscuamperfecto',
      indicativo,[104,97,98,-19,97|Resto],Resto).
verbo_auxiliar(segunda,singular,'pretérito pluscuamperfecto',
```

```
     indicativo,[104,97,98,-19,97,115|Resto],Resto).
verbo_auxiliar(tercera,singular,'pretérito pluscuamperfecto',
     indicativo,[104,97,98,-19,97|Resto],Resto).
verbo_auxiliar(primera,plural,'pretérito pluscuamperfecto',
     indicativo,[104,97,98,-19,97,109,111,115|Resto],Resto).
verbo_auxiliar(segunda,plural,'pretérito pluscuamperfecto',
     indicativo,[104,97,98,-19,97,105,115|Resto],Resto).
verbo_auxiliar(tercera,plural,'pretérito pluscuamperfecto',
     indicativo, [104,97,98,-19,97,110|Resto],Resto).

verbo_auxiliar(primera,singular,'pretérito anterior',indicativo,
     [104,117,98,101|Resto],Resto).
verbo_auxiliar(segunda,singular,'pretérito anterior',indicativo,
     [104,117,98,105,115,116,101|Resto],Resto).
verbo_auxiliar(tercera,singular,'pretérito anterior',indicativo,
     [104,117,98,111|Resto],Resto).
verbo_auxiliar(primera,plural,'pretérito anterior',indicativo,
     [104,117,98,105,109,111,115|Resto],Resto).
verbo_auxiliar(segunda,plural,'pretérito anterior',indicativo,
     [104,117,98,105,115,116,101,105,115|Resto],Resto).
verbo_auxiliar(tercera,plural,'pretérito anterior',indicativo,
     [104,117,98,105,101,114,111,110|Resto],Resto).

verbo_auxiliar(primera,singular,'futuro perfecto',indicativo,
     [104,97,98,114,-23|Resto],Resto).
verbo_auxiliar(segunda,singular,'futuro perfecto',indicativo,
     [104,97,98,114,-31,115|Resto],Resto).
verbo_auxiliar(tercera,singular,'futuro perfecto',indicativo,
     [104,97,98,114,-31|Resto],Resto).
verbo_auxiliar(primera,plural,'futuro perfecto',indicativo,
     [104,97,98,114,101,109,111,115|Resto],Resto).
verbo_auxiliar(segunda,plural,'futuro perfecto',indicativo,
     [104,97,98,114,-23,105,115|Resto],Resto).
verbo_auxiliar(tercera,plural,'futuro perfecto',indicativo,
     [104,97,98,114,-31,110|Resto],Resto).

verbo_auxiliar(primera,singular,'condicional compuesto',
     indicativo,[104,97,98,114,-19,97|Resto],Resto).
verbo_auxiliar(segunda,singular,'condicional compuesto',
     indicativo,[104,97,98,114,-19,97,115|Resto],Resto).
verbo_auxiliar(tercera,singular,'condicional compuesto',
     indicativo,[104,97,98,114,-19,97|Resto],Resto).
verbo_auxiliar(primera,plural,'condicional compuesto',
     indicativo,[104,97,98,114,-19,97,109,111,115|Resto],Resto).
verbo_auxiliar(segunda,plural,'condicional compuesto',
     indicativo,[104,97,98,114,-19,97,105,115|Resto],Resto).
verbo_auxiliar(tercera,plural,'condicional compuesto',
     indicativo,[104,97,98,114,-19,97,110|Resto],Resto).

verbo_auxiliar(primera,singular,'pretérito perfecto',subjuntivo,
     [104, 97,121,97|Resto],Resto).
verbo_auxiliar(segunda,singular,'pretérito perfecto',subjuntivo,
     [104, 97,121,97,115|Resto],Resto).
```

```
verbo_auxiliar(tercera,singular,'pretérito perfecto',subjuntivo,
     [104, 97,121,97|Resto],Resto).
verbo_auxiliar(primera,plural,'pretérito perfecto',subjuntivo,
     [104,97, 121,97,109,111,115|Resto],Resto).
verbo_auxiliar(segunda,plural,'pretérito perfecto',subjuntivo,
     [104,97, 121,-31,105,115|Resto],Resto).
verbo_auxiliar(tercera,plural,'pretérito perfecto',subjuntivo,
     [104,97, 121,97,110|Resto],Resto).

verbo_auxiliar(primera,singular,'pretérito pluscuamperfecto',
     subjuntivo,[104,117,98,105,101,114,97|Resto],Resto).
verbo_auxiliar(segunda,singular,'pretérito pluscuamperfecto',
     subjuntivo,[104,117,98,105,101,114,97,115|Resto],Resto).
verbo_auxiliar(tercera,singular,'pretérito pluscuamperfecto',
     subjuntivo,[104,117,98,105,101,114,97|Resto],Resto).
verbo_auxiliar(primera,plural,'pretérito pluscuamperfecto',
     subjuntivo,[104,117,98,105,-23,114,97,109,111,115|Resto],
     Resto).
verbo_auxiliar(segunda,plural,'pretérito pluscuamperfecto',
     subjuntivo,[104,117,98,105,101,114,97,105,115|Resto],
     Resto).
verbo_auxiliar(tercera,plural,'pretérito pluscuamperfecto',
     subjuntivo,[104,117,98,105,101,114,97,110|Resto],Resto).
verbo_auxiliar(primera,singular,'pretérito pluscuamperfecto',
     subjuntivo,[104,117,98,105,101,115,101|Resto],Resto).
verbo_auxiliar(segunda,singular,'pretérito pluscuamperfecto',
     subjuntivo,[104,117,98,105,101,115,101,115|Resto],Resto).
verbo_auxiliar(tercera,singular,'pretérito pluscuamperfecto',
     subjuntivo,[104,117,98,105,101,115,101|Resto],Resto).
verbo_auxiliar(primera,plural,'pretérito pluscuamperfecto',
     subjuntivo,[104,117,98,105,-23,115,101,109,111,115|Resto],
     Resto).
verbo_auxiliar(segunda,plural,'pretérito pluscuamperfecto',
     subjuntivo,[104,117,98,105,101,115,101,105,115|Resto],
     Resto).
verbo_auxiliar(tercera,plural,'pretérito pluscuamperfecto',
     subjuntivo,[104,117,98,105,101,115,101,110|Resto],Resto).

verbo_auxiliar(primera,singular,'futuro perfecto',subjuntivo,
     [104,117, 98,105,101,114,101|Resto],Resto).
verbo_auxiliar(segunda,singular,'futuro perfecto',subjuntivo,
     [104,117, 98,105,101,114,101,115|Resto],Resto).
verbo_auxiliar(tercera,singular,'futuro perfecto',subjuntivo,
     [104,117, 98,105,101,114,101|Resto],Resto).
verbo_auxiliar(primera,plural,'futuro perfecto',subjuntivo,
     [104,117,98,105,-23,114,101,109,111,115|Resto],Resto).
verbo_auxiliar(segunda,plural,'futuro perfecto',subjuntivo,
     [104,117, 98,105,101,114,101,105,115|Resto],Resto).
verbo_auxiliar(tercera,plural,'futuro perfecto',subjuntivo,
     [104,117, 98,105,101,114,101,110|Resto],Resto).

participio([99,97,110,116,97,100,111|Resto],Resto).

articulo(determinado,masculino,singular,[101,108|Resto],Resto).
```

```
articulo(determinado,femenino,singular,[108,97|Resto],Resto).
articulo(determinado,masculino,plural,[108,111,115|Resto],
     Resto).
articulo(determinado,femenino,plural,[108,97,115|Resto],Resto).
articulo(indeterminado,masculino,singular,[117,110|Resto],
     Resto).
articulo(indeterminado,femenino,singular,[117,110,97|Resto],
     Resto).
articulo(indeterminado,masculino,plural,[117,110,111,115|Resto],
     Resto).
articulo(indeterminado,femenino,plural,[117,110,97,115|Resto],
     Resto).

nombre(comun,masculino,singular,[112,97,115,111,100,111,98,108,
     101|Resto],Resto).
nombre(comun,femenino,singular,[99,97,110,99,105,-13,110|Resto],
     Resto).
nombre(Tipo,Genero,plural,Lista1,Resto):-
     nom(Tipo,Genero,Terminacion,Lista1,Lista2),
     sufijo_de_numero(Terminacion,Lista2,Resto).

nom(comun,masculino,vocal,[112,97,115,111,100,111,98,108,
     101|Resto],Resto).
nom(comun,femenino,consonante,[99,97,110,99,105,111,110|Resto],
     Resto).

sufijo_de_numero(vocal,[115|Resto],Resto).
sufijo_de_numero(consonante,[101,115|Resto],Resto).
```

Realizando la siguiente consulta se obtienen todas las oraciones generadas por la gramática:

```
?- oracion(O).

O = yo canto el pasodoble
...
O = yo cantaba el pasodoble
...
O = yo canté el pasodoble
...
O = yo cantaré el pasodoble
...
O = yo cantaría el pasodoble
...
O = yo he cantado el pasodoble
...
O = yo había cantado el pasodoble
...
O = yo hube cantado el pasodoble
...
O = yo habré cantado el pasodoble
```

```
...
O = yo habría cantado el pasodoble
...
O = yo canto
...
O = canto el pasodoble
...
```

El procedimiento se repite con todas las personas del singular y del plural, para todos los tiempos del modo indicativo y del subjuntivo.

Resumiendo todo lo visto en este capítulo, se comenzó con la presentación de la lógica que subyace al Prolog. El lenguaje de esta lógica se restringe a cláusulas de Horn, lo que supone cierta pérdida de capacidad expresiva con respecto a la lógica de primer orden pero, a cambio, se obtiene un procedimiento eficiente y no exponencial para la satisfacibilidad de las fórmulas. Después del lenguaje, se presentó la semántica, basada en el concepto de modelo Herbrand mínimo y el mecanismo de inferencia, fundamentado en la regla de resolución-SLD. También se dio cuenta del problema de la negación en programación lógica, para cuya incorporación se utiliza la meta-regla de la negación como fallo y da lugar al procedimiento de resolución-SLDNF.

Con respecto al Prolog, lenguaje de programación que será empleado para el tratamiento de la sinonimia en este trabajo, se aludió brevemente a su sintaxis y a cómo se construye una base de hechos y una base de reglas. El tratamiento de listas será utilizado reiteradas veces y el de gramáticas constituirá un recurso fundamental cuando en el capítulo tercero se analicen los problemas de la traducción y ciertos aspectos de la sinonimia entre oraciones. Se han

proporcionado las definiciones de una serie de predicados que serán empleados en esta propuesta, y también se ha descrito cómo se usan algunos predicados predefinidos a los que se hará referencia en los capítulos tercero y cuarto.

Hasta el momento, queda presentada la herramienta con la que se pretende realizar el tratamiento computacional de la sinonimia. Esta presentación se ha realizado de un modo breve y haciendo hincapié sólo en aquellas estrategias que resultarán interesantes para el análisis posterior. Pero antes de comenzar con el procesamiento automático de la sinonimia conviene hacer un recorrido por distintas concepciones de la misma que proporcionarán cierta caracterización del objeto de estudio del trabajo. El próximo capítulo está dedicado a ello.

CAPÍTULO II

Concepciones de la sinonimia

Con el fin de caracterizar el objeto de estudio de este trabajo, en el presente capítulo se realizará una breve exposición de distintas concepciones de la sinonimia que se han dado a lo largo de la historia. Dicha exposición sólo pretende poner de manifiesto:

1. Que principalmente hay dos formas de estudiar la sinonimia: una de ellas es la que trata de proporcionar una definición de la misma; la otra es la que intenta describir su comportamiento en el uso lingüístico.
2. Que la polémica entre considerar la sinonimia como identidad o como parecido de significado, tiene lugar ya desde los orígenes del análisis sobre el tema y perdura durante todo su desarrollo.

Hoy en día se acepta normalmente que la sinonimia concebida como relación precisa, esto es, como relación de equivalencia entre dos expresiones, o

como identidad de significado, no existe en la práctica. Podría parecer que esto es fruto de la evolución y de la consiguiente maduración de las ideas en torno a esta característica del lenguaje; sin embargo, un breve examen histórico induce a pensar que esto no es así.

Este capítulo consta de tres partes. En la primera de ellas se realiza una exposición sobre los orígenes del estudio de la sinonimia, localizados en la Grecia antigua. Desde la antigüedad y durante toda su evolución hasta la etapa moderna ha habido quienes la consideraban como identidad estricta de significado y quienes afirmaban que puede haber diferencias de significado entre dos términos sinónimos sin que signifique la negación de la existencia de sinonimia entre ambos. En la época de la Ilustración, los estudios de sinonimia comienzan una etapa de esplendor que marcó el origen moderno de los mismos. A partir de aquel momento, parece que la balanza se inclina hacia el parecido y la diferencia de significado en la concepción de la sinonimia, sin embargo esta polémica nunca ha desaparecido del todo.

La segunda parte del capítulo da cuenta de los análisis sobre sinonimia en el marco de la lógica y la filosofía del siglo XX. Al principio, estos análisis se caracterizaron, en general, por el examen de los lenguajes naturales con los métodos y estrategias de carácter preciso propios de lenguajes formales como la lógica de primer orden. Esto trae como consecuencia cierta tendencia a considerar la sinonimia como una cuestión más bien precisa. Sin embargo, a partir de mediados de siglo, ante las dificultades y problemas que acarrea el

análisis del lenguaje natural con métodos formales, algunos filósofos del lenguaje se inclinan hacia una concepción gradual de la sinonimia así como también abogan por la necesidad de un estudio empírico de la misma.

En la tercera parte del capítulo se marcarán las pautas para el estudio de la sinonimia que se va a proponer en este trabajo. El enfoque de esta propuesta será de corte descriptivo y computacional. Se tratará de describir el comportamiento de la sinonimia en la práctica lingüística, la cual revelará su carácter gradual y contextual. La gradualidad se tratará en términos de similaridad de conjuntos de sinónimos y el resultado será un diccionario electrónico de sinónimos implementado en Prolog, que calculará el grado de sinonimia de las expresiones que contiene, teniendo en cuenta el carácter contextual de la misma.

1.- Origen de los estudios sobre sinonimia

Históricamente, los estudios de sinonimia se remontan a la época de los sofistas en la Grecia antigua. El primero en preocuparse por este tema fue el ateniense Pródico de Ceos en el siglo V a. c., del que se tiene noticia por haber sido mencionado en el *Protágoras*, *Eutidemo* y *Laques* de Platón. En dichas fuentes se destaca su empeño por la precisión verbal llevada a cabo mediante el método conocido como *corrección o exactitud de los nombres*, cuya invención se le atribuye. Esa tendencia a la corrección le lleva a sostener la existencia de diferencias de significado en los sinónimos a pesar de su aparente semejanza. Su aportación sirvió de estímulo a otros sofistas como Protágoras o Gorgias en la

práctica de distinguir el significado de las palabras. Tanto Tucídides como el resto de los discípulos de Pródico, heredaron su empeño por el uso exacto de las palabras y se dedicaron a realizar distinciones minuciosas entre conceptos.

Con todo, parece haber sido Aristóteles el primero en emplear el término *συνωνυμία* y quien sentó los pilares de una teoría de la sinonimia. Sin embargo, éste la concebía como una propiedad de las cosas y no de los nombres. En las *Categorías* explica el concepto de sinonimia junto al de homonimia diciendo:

> «Se llaman homónimas las cosas que sólo tienen el nombre en común, pero cuya definición de acuerdo con el nombre es diferente; así, por ejemplo, tanto un hombre como su representación pictórica son animales. Ciertamente, esas cosas sólo tienen el nombre en común y la definición de acuerdo con el nombre es diferente; puesto que si alguien tuviese que dar cuenta de en qué consiste para cada una de ellas el ser un animal, daría una definición propia para cada una.
>
> Se llaman sinónimas las cosas que tienen el nombre en común, pero cuya definición de acuerdo con el nombre es la misma; así, por ejemplo, tanto un hombre como un buey son animales. Ciertamente, a cada uno de ellos se lo denomina mediante un nombre común, 'animal', y la definición es también la misma; pues si alguien tuviera que proporcionar la definición de cada uno de ellos, a saber: en qué consiste para cada uno de ellos el ser un animal, daría la misma definición.» (*Categorías* 1a 1, traducción de Valdés Villanueva, L. [1983]).

Al contrario de Aristóteles, Espeusipo, sucesor de Platón al frente de la Academia, consideró la sinonimia como una propiedad de las expresiones y no de las cosas, dando lugar a una versión que concuerda más con la concepción actual de sinonimia. Esta definición aristotélica de sinonimia no coincide con el

uso que Aristóteles hace del término en otros contextos. En el siguiente texto de la *Retórica* parece concebirla como una relación entre palabras:

> «Por lo demás, de entre los nombres, los homónimos son útiles para el sofista (pues en ellos basa sus fraudes) y los sinónimos, para el poeta. Por mi parte, llamo específicos y sinónimos a, por ejemplo, 'caminar' y 'andar', porque ambos son específicos y sinónimos entre sí.» (*Retórica* 1404b 39 y ss., traducción de Racionero, Q. [1990])

A pesar de que la totalidad de los escritos estoicos han desaparecido, todo parece apuntar a que estos también trataron el tema de la sinonimia. Esta sospecha se basa en la reconstrucción de fragmentos breves de sus obras y comentarios de autores posteriores a ellos como Sexto Empírico. Afirmaban la existencia de irregularidades en el lenguaje, por eso no es de extrañar que se fijasen sobre todo en las diferencias entre sinónimos más que en la identidad. A comienzos de la Edad Media, los estoicos eran conocidos por la realización de distinciones muy sutiles entre palabras. El estoicismo se estableció de forma rápida en Alejandría y sus ideas influyeron sobre los gramáticos alejandrinos. Algunos de estos, como Aristófanes de Bizancio, Aristarco o Dídimo hacían hincapié principalmente en la diferencia de los sinónimos. Ambas escuelas sirvieron de modelo para los gramáticos romanos.

En los inicios del Imperio Romano el estudio de la diferencia entre sinónimos comienza una etapa de esplendor. Aunque Aristóteles ya había destacado el interés que la sinonimia tiene dentro de la retórica, es en la época romana cuando esta importancia llega a su apogeo. La sinonimia pasó a ser considerada como una figura retórica de primera magnitud utilizada para evitar la

repetición de palabras en un discurso. El estudio de la diferencia de significado en los sinónimos era muy útil para la codificación del lenguaje jurídico. Uno de los rétores que trataron el tema de la sinonimia fue Áquila Romano al que se le atribuye la traducción del término aristotélico *συνωνυμία* por *communio nominis* (*comunidad de nombre*) en su *Tratado de las figuras*, del que se tiene noticia al haber sido citado por Marciano Capella en el libro V dedicado a la retórica de su obra *De nuptiis Philologiae et Mercuri* (*Las bodas de Mercurio y de Filología*). Sin embargo, Áquila interpretó la definición de Aristóteles de forma diferente concibiendo la sinonimia como aquella figura retórica que se da cuando diversas palabras expresan la misma idea. Cicerón, uno de los más grandes oradores romanos, utilizó los sinónimos de manera tan magistral en sus escritos, que la mayoría de los estudiosos que le precedieron ilustraban sus reflexiones sobre sinonimia con ejemplos extraídos de sus obras. Tanto es así que se llegó a construir un glosario que lleva su nombre (*Synonyma Ciceronis*), cuyo uso se extiende hasta la época medieval.

El estudio de la sinonimia, que se había extendido en Roma vinculado a la retórica, se hace independiente de ésta a partir del siglo II, lo cual ha sido importante para el nacimiento y desarrollo de la lexicología y de la gramática. Para ello, los gramáticos romanos, como Seleuco de Alejandría, que escribió el tratado *Sobre la diferencia en los sinónimos*, se habían inspirado en los griegos, sobre todo en los trabajos de los estoicos y también de los gramáticos alejandrinos. El apogeo de este estudio continuará hasta el final de la

Antigüedad, en la que Isidoro de Sevilla realiza una obra de compilación que lleva por título *Differentiae*.

En la época romana también había circulado el léxico *Sobre las expresiones semejantes y diferentes* creado por Amonio. Consistía en un diccionario alfabético con 525 entradas en las que no sólo se trata la sinonimia, sino también la homonimia y la polisemia. Igualmente, en Bizancio se realizaron glosas de sinónimos, como el diccionario de verbos sinónimos confeccionado por Constantino Harmenópulo, un jurista de Tesalónica del siglo XIV. El estudio de la sinonimia continuará en la Edad Media, donde proliferan las colecciones de sinónimos, algunas de ellas con nombres de autores clásicos como el ya citado glosario *Synonyma Ciceronis*, que al parecer, tenía un gran éxito en las escuelas medievales. Entre estas colecciones cabe señalar el *Glosario Patavino* anónimo del siglo XIII. En el Renacimiento hay que destacar las *Elegantiae* de Lorenzo Valla, que constituyen un sutil análisis de la lengua latina, así como el *Universal vocabulario en latín y en romance* de Alfonso de Palencia, contemporáneo del *Diccionario latino-español* de Antonio de Nebrija.

Un nombre clave en la historia de los estudios sobre sinonimia es el gramático francés de principios del siglo XVIII, Gabriel Girard, que con su obra *La justesse de la langue françoise, ou les différentes significations des mots qui passent pour synonymes*, aparecida en 1718, puede considerarse el padre de los estudios modernos de sinonimia. El autor sostiene que no debemos esperar una semejanza exacta en toda la significación de dos expresiones sinónimas. Admite

que lo que hace sinónimas a dos palabras es la semejanza en una idea principal o general que ambas enuncian, pero que cada una de ellas matiza de forma particular, estableciendo así diferencias que hacen que la sinonimia no sea del todo perfecta. Con Girard tiene lugar un renacimiento y un giro en los estudios de sinonimia, cuyo influjo llega hasta nuestros días. Una de las principales consecuencias de este influjo ha sido la confección de diccionarios de sinónimos, en un primer momento en lengua francesa y más tarde en otras lenguas como el español.

El primer estudio sobre sinonimia en español, escrito por Manuel Dendo y Ávila en 1757, se titula *Ensayo de los synonimos*. Consiste en un conjunto de apuntes donde se analizan trece parejas de sinónimos, según él, algunos más perfectos que otros. Años más tarde, en 1789, aparece el *Exámen de la posibilidad de fixar la significación de los sinónimos de la lengua castellana* de José López de la Huerta, que es un diccionario de sinónimos en español, en el que se puede percibir el influjo de los sinonimistas franceses.

La trayectoria de investigación en sinonimia fundada por Girard todavía sigue en vigor con el nacimiento de la semántica en el marco de la lingüística, a manos de Michel Bréal, a finales del siglo XIX. Una de las ideas de éste es conocida con el nombre de *ley de repartición* (véase Bréal, M. [1924] capítulo II) que consiste en que a palabras distintas le corresponden cosas o conceptos diferentes, lo que le lleva a manifestar la inexistencia de sinonimia perfecta.

Como conclusión, se puede observar que la trayectoria general de los estudios sobre sinonimia desde la Antigüedad hasta comienzos del siglo XX, es la que, *grosso modo*, parte del análisis en la práctica de fenómenos lingüísticos y llega al establecimiento de propuestas teóricas. Los primeros análisis consistían, generalmente, en la disertación sobre parejas concretas de sinónimos, lo cual tuvo como consecuencia la generación de léxicos y glosas de términos. Estas obras mostraron ser de gran utilidad para los rétores, juristas, oradores o poetas de la antigüedad, que tenían en ellas una magnífica herramienta de ayuda. Por otro lado, la ulterior discusión sobre cuestiones de índole más teórica respecto a la sinonimia, resultó ser útil para el nacimiento de disciplinas como la lexicología o la gramática. Parece desprenderse de su evolución histórica, que las propuestas teóricas surgen, en cierto modo, de los análisis de la sinonimia en la práctica. Las concepciones de la sinonimia que serán presentadas en la próxima sección parten, en general, de la búsqueda de definiciones de sinonimia, que al ser contrastadas con el comportamiento de ésta en la práctica, resultan problemáticas y se abandonan generalmente en favor de los estudios más pegados al análisis del uso del lenguaje natural. Sin embargo, los estudios encaminados a la definición y los centrados en el uso se complementan, ya que los segundos proporcionan un marco que hace posible que los primeros puedan ser, de alguna forma, contrastados, y los primeros, por su parte, plantean problemas que deben ser tenidos en cuenta en los segundos. En este sentido, a pesar de que la orientación de este trabajo es más bien práctica, lo que tiene como resultado la elaboración

de un diccionario electrónico de sinónimos, pretende beneficiar tanto a los que necesitan acudir al recurso de la sinonimia para mejorar su uso del lenguaje, como a los que persiguen su definición.

2.- La sinonimia en la filosofía del lenguaje del siglo XX

En la primera mitad del siglo XX, se produjo en filosofía un cambio de orientación que supuso un enfoque novedoso en el estudio de algunos problemas filosóficos. Este cambio en la metodología del filosofar recibe el nombre de *giro lingüístico*. Dicha denominación fue popularizada en los años sesenta por R. Rorty (véase Rorty, R. [1967]), aunque éste la toma de G. Bergman, que la caracteriza de la siguiente forma:

> «Todos los filósofos lingüísticos hablan acerca del mundo por medio de un hablar sobre un lenguaje apropiado. Este es el giro lingüístico, la táctica fundamental a manera de método, sobre el que están de acuerdo los filósofos del lenguaje ordinario e ideal.» (véase Bergman, G. [1964])

Para muchos filósofos, el estudio de los problemas filosóficos requería el análisis del lenguaje con el fin de asegurar un uso correcto del mismo que no fuese problemático por conducir a falacias, contradicciones, razonamientos circulares, paradojas, etc. Una parte de los estudios filosóficos se centró en el lenguaje aproximándose, de este modo, a la lingüística y algunas figuras relevantes de la filosofía de este siglo, como Frege, Russell, Wittgenstein o Carnap se agrupan en lo que se conoce como *Filosofía Analítica* (también

denominada, en ocasiones, *Filosofía Lingüística*). J. Peregrin describe esta situación diciendo que los filósofos se convierten en lingüistas y los lingüistas en filósofos, de tal forma que en la relación entre ambas disciplinas se da un círculo vicioso: los lingüistas, para explicar algunos fenómenos, utilizan conceptos filosóficos que son caracterizados por los filósofos en términos de los fenómenos que los lingüistas tratan de explicar (véase Peregrin, J. [1998]).

En el marco de la lógica y la filosofía del siglo XX pueden distinguirse dos tendencias:

1. *Propuestas basadas en los lenguajes formales*: Estas propuestas son deudoras de una tradición que se origina en G. Frege, quien, con su obra *Conceptografía*, se convierte en el padre de la lógica moderna, proporcionando con ello una herramienta de análisis del lenguaje: la lógica formal. La tendencia usual en este contexto es la de tratar aspectos del lenguaje natural con la metodología de análisis de los lenguajes formales como la lógica de primer orden. L. Wittgenstein siguió esta tradición en el *Tractatus Logico-Philosophicus*, pero se percató de que esta tendencia encuentra dificultades a la hora de ser aplicada en la práctica lingüística.
2. *Propuestas basadas en el uso del lenguaje natural*: Estas propuestas son deudoras de la tradición que se origina en las *Investigaciones filosóficas* de Wittgenstein, el cual, consciente de las dificultades surgidas al utilizar los métodos formales para el análisis del lenguaje

> natural, se inclina por un estudio más próximo al uso lingüístico. Para Wittgenstein los problemas filosóficos se resuelven sondeando el funcionamiento de nuestro lenguaje y la tarea de la filosofía debe ser la descripción de los fenómenos y no su explicación o definición. Además, según él, la filosofía no puede interferir con el uso del lenguaje, solamente puede describirlo.

Esta distinción también afecta a la manera de abordar el problema de la sinonimia. Con respecto a la primera tendencia, el resultado de emplear la metodología de los lenguajes formales para el análisis de la sinonimia lleva a concebirla como una relación precisa de todo o nada, es decir, como una relación de equivalencia, al estilo de cómo se concebiría en un lenguaje formal. A pesar de que estas propuestas se caracterizan por realizar intentos explícitos de análisis y definición de la sinonimia, ésta no suele ser un tema fundamental sino que su definición es un medio para clarificar otros conceptos como el significado, la analiticidad, la traducción, etc., los cuales, están vinculados de tal forma que no hay manera de saber cuál es más primitivo que los demás.

Por otra parte están los estudios comprometidos con el uso de la sinonimia en el lenguaje natural, encaminados, no a su análisis y definición explícitas, sino a la descripción de su comportamiento en la práctica lingüística. En este marco, los estudios de corte empírico son fundamentales, ya que reflejan el uso de una lengua y revelan características importantes de la sinonimia, como su carácter gradual y contextual. El presente trabajo está más próximo a esta tendencia de

talante más descriptivo que explicativo. Al estilo de los textos de Austin, heredero de la tradición wittgensteiniana, y Ullmann, perteneciente al campo de la lingüística (ambos citados en el apartado 2.2), se presupone cierta comprensión de en qué consiste la sinonimia. No se perseguirá una definición de la misma, ni siquiera su descripción completa, sino reflejar algún aspecto de su comportamiento en el marco restringido de un diccionario de sinónimos. A pesar de esta proximidad con la segunda tendencia, en este capítulo se examinarán tanto las basadas en los lenguajes formales como las basadas en el uso, ya que unas y otras representan intentos de análisis de la sinonimia que exponen problemas a tener en cuenta en un estudio aplicado como éste.

Lo destacable de la distinción es que proporciona dos caracterizaciones bien diferentes de la sinonimia. La primera la concibe como una relación precisa entre expresiones que se podría caracterizar como identidad o equivalencia de significado, tal y como esta relación se comportaría en un lenguaje formal. La segunda concibe la sinonimia como una relación aproximada o gradual que se podría caracterizar como parecido o similaridad de significado, tal y como parece deducirse del comportamiento de ésta en el lenguaje natural. Mediante la utilización de un diccionario de sinónimos como muestra razonablemente fiable de la práctica lingüística, en este trabajo, se podrá ver cómo dicha concepción gradual da cuenta del comportamiento de la sinonimia en el uso de una lengua.

2.1.- Propuestas basadas en los lenguajes formales

Históricamente, uno de los primeros intentos explícitos de definición de la sinonimia en el marco de la lógica y la filosofía del siglo XX es el proporcionado por K. Ajdukiewicz en la década de los treinta, utilizado como paso intermedio para llegar a una definición de significado (véase Ajdukiewicz, K. [1931] y Ajdukiewicz, K. [1934]). Mediante el uso de conceptos cuya definición no entra dentro del alcance de este trabajo, establece una compleja definición de sinonimia y concibe el significado de una expresión como la propiedad común a todos sus sinónimos. La propuesta es llamativa debido a que las caracterizaciones más populares definen la sinonimia en términos de significado y no el significado en términos de sinonimia. La definición de Ajdukiewicz tiene cierto parecido con la propuesta de este trabajo, puesto que recurre al conjunto de todos los sinónimos de una expresión para definir su significado. Ciertamente, no es sencillo tener un listado de todos los sinónimos de una expresión, sin embargo, un diccionario de sinónimos proporciona un considerable número de ellos. Quizás la definición de Ajdukiewicz no sea demasiado clarificadora, puesto que quedaría el problema de cómo detectar cuál es esa propiedad común a todos los sinónimos de una expresión, sin embargo, aunque el conjunto de sinónimos no proporcione una explicación clarificadora del significado sí puede posibilitar una forma de representarlo cuyo tratamiento computacional resulta relativamente cómodo y factible como se verá más adelante.

Carnap es uno de los más claros ejemplos del tratamiento de la sinonimia con el uso de estrategias propias de los lenguajes formales (véase Carnap, R. [1947-56] y Carnap, R. [1955]). Éste trata de elaborar un modelo para analizar el significado de las expresiones del lenguaje. Con este fin, construye un lenguaje formal donde tanto la extensión como la intensión son interpretadas en términos de conceptos fácticos y lógicos. La extensión de un predicado es la clase de objetos a los que éste se refiere y la intensión es la propiedad que predica. Dos predicados son *equivalentes* si tienen la misma extensión y *lógicamente equivalentes* (*L-equivalentes*) si tienen la misma intensión. Podría entenderse que dos expresiones simples son sinónimas si tienen la misma intensión, con lo cual, para este tipo de expresiones, la sinonimia coincide con la L-equivalencia. Sin embargo, lo que interesa a Carnap es la sinonimia entre expresiones complejas, ya que el hecho de que éstas sean L-equivalentes no significa que sean sinónimas. Si lo fuesen habría que considerar sinónimas a todas las verdades lógicas; para evitar esto, concibe la sinonimia como *isomorfismo intensional*. Dos expresiones son intensionalmente isomórficas si son L-equivalentes tomadas como un todo y también lo son respectivamente cada uno de sus componentes simples, para lo cual, la estructura de ambas expresiones debe ser la misma. Por ejemplo, sean las expresiones "2 + 5" y "II sum V" pertenecientes a un lenguaje *S*. Suponiendo que en las reglas semánticas de *S* se indique que "+" y "sum" son functores para la misma función aritmética, con lo que ambos serían L-equivalentes, y que los signos correspondientes a expresiones numéricas tienen el

significado que usualmente se les da, con lo que "2" y "II" serían L-equivalentes, así como también "5" y "V". Entonces se podría decir que "2 + 5" y "II sum V" son intensionalmente isomórficas o que tienen la misma estructura intensional, algo que no se podría decir de las expresiones "2 + 5" y "7", que a pesar de ser L-equivalentes, no tienen la misma estructura.

El isomorfismo intensional proporciona, según Carnap, una explicación de la sinonimia tal y como es comúnmente entendida. El proceso de construcción de su definición comienza por el establecimiento de una serie de *reglas de designación* para expresiones primitivas; por ejemplo, *H* representa a "humano" y *m* a "Miguel de Cervantes". Mediante reglas de formación se pueden generar oraciones simples como *Hm*, que es verdadera si Miguel de Cervantes es humano, y complejas como $Hm \vee \sim Hm$ cuyo valor de verdad depende de sus componentes. Utilizando la noción de *descripción de estado*, introduce una serie de conceptos lógicos como el de equivalencia lógica o el de verdad lógica. Una descripción de estado contiene cada oración atómica del lenguaje o su negación, de tal forma que una oración puede darse o no darse en la descripción de estado. Una oración se dice que es *L-verdadera* si se da en toda descripción de estado y dos oraciones son L-equivalentes si su equivalencia se da en toda descripción de estado. A partir de aquí Carnap distingue entre extensión e intensión haciendo corresponder la equivalencia a la igualdad de extensión y la L-equivalencia a la igualdad de intensión. Posteriormente define el isomorfismo intensional en

términos de L-equivalencia que, como se ha indicado, coinciden para el caso de las expresiones simples.

El siguiente ejemplo ilustra la propuesta de Carnap: suponiendo que es un hecho empírico que todos los seres humanos son bípedos implumes y viceversa, algo que se podría expresar formalmente con *(x) Hx ≡ BIx*, donde *H* representa a "humano" y *BI* a "bípedo implume"; esta oración es verdadera ya que se puede sustituir la *x* por cualquier constante de individuo *a*, *b*, … de tal forma que *Ha ≡ BIa*, *Hb ≡ BIb*, … sean todas verdaderas, lo que hace que *H* y *BI* sean equivalentes. Por otra parte, Carnap sostiene que la verdad de la oración *(x) Hx ≡ ARx*, donde *H* representa a "humano" y *AR* a "animal racional", se puede establecer sin hacer referencia a los hechos utilizando las reglas semánticas del sistema formal diseñado por él, esto significa que dicha oración es L-verdadera y por consiguiente *H* y *AR* L-equivalentes, es decir, que las expresiones "humano" y "animal racional" significan lo mismo. Carnap proporciona reglas, procedimientos y criterios para reducir expresiones complejas a sus componentes con el fin de evaluarlas y compararlas, haciendo descansar la noción de sinonimia en el concepto de isomorfismo intensional, el cual depende, a su vez del de L-equivalencia y del hecho de que los términos primitivos que contienen dichas expresiones complejas tengan el mismo significado, pero no clarifica cuándo sucede esto último. Carnap proporciona un importante análisis formal de la sinonimia, sin embargo, parece centrar su atención en las expresiones de un sistema formal o en ciertas partes del lenguaje natural que involucran conceptos

precisos como los de la matemática, la lógica etc. y no en el lenguaje natural como un todo. Esto le lleva a ser estricto con el hecho de que dos expresiones sinónimas deben tener la misma estructura para que se pueda comprobar si sus componentes son L-equivalentes respectivamente.

C. I. Lewis también explica la sinonimia en términos de intensión (véase Lewis, C. I. [1943]). Parte de cierto análisis del significado en el que se distinguen cuatro modos de significar: la *denotación* o extensión, la *connotación* o intensión, la *comprensión* y la *significación*. Considera la sinonimia como igualdad de connotación o intensión. La intensión de un término está delimitada, según él, por la definición correcta de dicho término. La caracterización que proporciona de la intensión depende del concepto "… ser nombrable correctamente", del cual, Lewis no porporciona una definición. Esta caracterización se puede expresar del siguiente modo: Si cualquier cosa que es nombrable correctamente por *T*, es también nombrable correctamente por A_1, A_2, …, A_n y viceversa, entonces este término compuesto o cualquiera de sus sinónimos, especifica la connotación de *T*.

Un término tiene intensión cero si se puede aplicar a cualquier cosa, e intensión universal si su aplicación entraña la aplicación de cualquier otro término; por ejemplo, "esto es un cuadrado redondo" entraña cualquier otra proposición. La definición de sinonimia como igualdad de intensión es problemática cuando los términos tienen intensión cero o universal, por eso Lewis utiliza una noción especial para este tipo de casos, a la que llama

equivalencia de significado analítico y que define diciendo que dos expresiones son equivalentes en su significado analítico si:

1. Al menos una de ellas es elemental y tienen la misma intensión.
2. Siendo ambas complejas pueden ser analizadas según sus constituyentes de tal forma que:
    - para todo constituyente de una, existe un constituyente correspondiente en la otra que tiene la misma intensión;
    - ningún constituyente de ninguna de ambas expresiones tiene extensión cero o universal;
    - el orden de los constituyentes correspondientes es el mismo en ambas expresiones, o puede ser el mismo cambiando los constituyentes sin que cambie la intensión de la expresión completa.

Dos expresiones serán sinónimas para Lewis si tienen la misma intensión y ésta no es cero o universal, o bien, siendo su intensión cero o universal, son equivalentes en su significado analítico. Aunque las nociones de equivalencia de significado analítico e isomorfismo intensional se parecen en buena medida, una de las diferencias es que mientras que la primera sólo se aplica a expresiones con intensión cero o universal, la segunda se aplica a cualquier expresión. Además, para Carnap, el hecho de que dos expresiones fuesen intensionalmente isomórficas, y por lo tanto sinónimas, exigía que ambas tuviesen la misma estructura, condición que para Lewis sólo debe cumplirse en caso de que la

intensión de ambas expresiones sea cero o universal. De este modo, Lewis aceptaría como sinónimas las expresiones "cuadrado" y "rectángulo con todos los lados iguales", mientras que Carnap no. A pesar de todo, Lewis incurre en el mismo error que Carnap al no indicar nada sobre cómo saber cuándo dos términos tienen la misma intensión; su caracterización de la intensión de un término sólo indica que viene definida mediante otros términos. Aunque Lewis parece definir mejor que Carnap la sinonimia respecto al lenguaje natural por el hecho de dar cuenta de ella para expresiones con distinta estructura, tanto uno como otro adoptan una estrategia formal de análisis y la tratan como sinonimia total.

Dadas las dificultades que acarrea la definición de la sinonimia en términos de intensión, N. Goodman trata de estudiar si la identidad extensional puede proporcionar una definición adecuada de la misma (véase Goodman, N. [1949]). Dicho brevemente, dos expresiones serían sinónimas para Goodman si se refieren al mismo conjunto de objetos. Sin embargo, esta definición no parece adecuada, ya que hace sinónimos a todos los predicados vacuos, es decir, a todos los predicados que no tienen referencia, como por ejemplo "unicornio" y "centauro". Por este motivo, distingue entre extensión primaria y secundaria, es decir, entre la extensión de la palabra "unicornio" y la extensión de cualquier descripción de un unicornio. De esta forma "unicornio" y "centauro" no serán sinónimos ya que sus extensiones secundarias son distintas. Sólo las palabras que verdaderamente sean sinónimas tendrán extensiones secundarias equivalentes. El

problema es que para cualquier par de objetos objetos *A* y *B* diferentes, siempre se puede construir una descripción correspondiente a uno de ellos, por ejemplo *A*, de la forma "un *A* que no es un *B*", con lo que parece que ningún par de términos podrán ser sinónimos. Esto lleva a Goodman a considerar que es mejor hablar de parecido de significado y no de sinonimia, algo que se basa en el error de considerarla como sinonimia total. Sin embargo, esta conclusión es más coherente con el punto de vista del lenguaje natural, ya que en la práctica, cuando se dice que dos términos tienen el mismo significado, normalmente se está indicando, según Goodman, que el tipo y el grado de parecido de significado son suficientes para los propósitos del discurso inmediato.

B. Mates recurre a la sinonimia para definir conceptos como la traducción o la interpretación. Concibe la traducción como la sustitución de una expresión por su sinónima, y la interpretación, como la sustitución de una expresión por un sinónimo más clarificador (véase Mates, B. [1950]). Una manera de tratar el problema de definir la sinonimia es proporcionar previamente un criterio de adecuación que toda definición debe satisfacer. La preservación del valor de verdad bajo sustitución parece ser condición necesaria para la sinonimia. Aunque esto no proporciona una definición de sinonimia, puede utilizarse para comprobar la adecuación de cualquier definición de la misma. Si se restringe el análisis a lenguajes extensionales, este criterio no proporciona sinonimia sino sólo equivalencia. A pesar de todo, Mates define la sinonimia en un lenguaje con operadores modales debido a que sólo los sinónimos genuinos podrán ser

sustituidos en cualquier oración de dicho lenguaje sin que cambie el valor de verdad de ésta. El problema es saber si hay expresiones que puedan ser sinónimas de acuerdo con este criterio, ya que resulta dudoso, por ejemplo, que alguien que crea que Luis es almeriense, crea también que Luis es urcitano, ya que no tiene por qué saber que las expresiones "almeriense" y "urcitano" son sinónimas. Sin embargo, lo son, aunque no se pueda sustituir una por otra en la oración "creo que Luis es almeriense".

W. O. Quine podría considerarse como uno de los críticos de las concepciones que analizan los problemas del lenguaje natural con métodos de los lenguajes formales. Sus ideas están a medio camino entre éstas y las que se centran en el uso para estudiar cuestiones del lenguaje natural. En esta breve exposición, Quine servirá de puente entre unas y otras. Consciente de los problemas que conlleva el análisis del significado, indica que podría ser preferible prescindir de esta noción ya que es de escasa utilidad si se concibe, en su sentido habitual o clásico, como una entidad abstracta. A pesar de esto, no pasará por alto los estudios propios de lo que él llama "teoría de la significación" cuyo objeto es la sinonimia de las expresiones lingüísticas y la analiticidad de los enunciados. En este sentido Quine pretende mostrar que hay una dependencia mutua entre la sinonimia y la analiticidad a la hora de definir esta última (véase Quine, W. O. [1951]). Un enunciado es analítico si y sólo si, o bien es una verdad lógica, o bien es transformable en una verdad lógica sustituyendo algunas expresiones por sus sinónimas. Quine denomina *sinonimia cognitiva* a este tipo

de sinonimia que permite convertir un enunciado analítico en una verdad lógica, y lo ejemplifica con el enunciado "ningún soltero está casado", que es analítico al poder convertirse en una verdad lógica sustituyendo "soltero" por su sinónimo (cognitivo) "hombre no casado". Esta definición de analiticidad no está exenta de problemas, algunos de ellos, surgidos de los intentos de definición de la sinonimia que el propio Quine analiza. Comienza considerando que dos expresiones son sinónimas por definición, pero esto no es satisfactorio debido a que, o bien está sometido a la arbitrariedad del que realizó la definición, o bien depende del uso lingüístico reflejado, por ejemplo, en un diccionario; lo que equivaldría a considerar que dos expresiones son sinónimas si son utilizadas como sinónimas, algo que, para Quine, no debería ser admisible como definición de sinonimia.

A continuación analiza los problemas de considerar la sinonimia como *intercambiabilidad salva veritate* en todos los contextos. Sin embargo, según él esta definición tampoco es adecuada por varios motivos:

1. La sustitución de la palabra "soltero" por "hombre no casado" en la oración "'soltero' tiene menos de diez letras" no conserva el valor de verdad de la misma, por tanto, no es cierto que los sinónimos sean intercambiables en todos los contextos *salva veritate*. Esta objeción se desvanece si la intercambiabilidad no se aplica a partes de palabras y se considera 'soltero' como una palabra entera incluidas las comillas.

2. La intercambiabilidad no proporciona sinonimia cognitiva, incluso aunque se restrinja el análisis a lenguajes extensionales, es decir, a aquellos en los que se cumple que si dos predicados coinciden extensionalmente, son intercambiables *salva veritate*. El hecho de que "soltero" y "hombre no casado" sean intercambiables *salva veritate* en un lenguaje extensional sólo nos garantiza la verdad de la oración "todos y sólo los solteros son hombres no casados", pero no su analiticidad. No tenemos la seguridad de que la coincidencia extensional de "soltero" y "hombre no casado" se deba a su significado y no a circunstancias fácticas, como acontece con "criatura con corazón" y "criatura con riñones".

Debido a las dificultades de definir la analiticidad en términos de sinonimia, Quine intenta definir la sinonimia en términos de analiticidad diciendo que dos términos son cognitivamente sinónimos si son intercambiables *salva analiticitate* y no solamente *salva veritate*. Pero esto no parece resolver el problema, ya que conduce a pensar que la analiticidad depende de las reglas semánticas de un lenguaje y esta dependencia no lleva a una buena explicación de la analiticidad.

Otro de sus intentos es definir la sinonimia en términos de verificación. Dos enunciados son sinónimos si y sólo si sus métodos de confirmación o invalidación empírica coinciden. Pero, una vez más, esta definición tampoco parece válida, ya que no está claro en qué consisten tales métodos.

Quine trae a colación un tema cuyo tratamiento computacional será abordado en el próximo capítulo: el caso de la sustituibilidad. El hecho de poder sustituir en una oración una expresión por su sinónima marca una de las utilidades prácticas de la sinonimia de la que ya se habían percatado los rétores de la Grecia antigua. La utilidad de un diccionario de sinónimos radica en el hecho de poder buscar sustitutos de palabras dentro de una oración, con el fin de evitar su repetición, aumentar la precisión o reforzar la expresión de un concepto. No se planteará allí el tema de la analiticidad pero sí el de la ambigüedad o polisemia de las palabras. La sustitución de una palabra *A* por otra *B* dentro de una oración puede no ser adecuada en algunas ocasiones cuando *A* posee varios significados y *B*, a pesar de ser sinónima de *A*, lo es para un significado de ésta distinto del empleado en la oración. Generalmente, los humanos logramos desambiguar las palabras empleadas con relativo éxito, sin embargo, que una máquina sea capaz de realizar esta labor es una tarea realmente compleja debido a la falta de precisión de las reglas por las que se rige el lenguaje natural. A pesar de esto, en el citado capítulo se esbozarán algunos intentos de tratamiento del problema.

2.2.- Propuestas basadas en el uso del lenguaje natural

El pensamiento de Quine se caracteriza por la crítica de las nociones de la semántica tradiccional como el significado, la sinonimia, la analiticidad, la extensión etc. que han sido manejadas en las propuestas del apartado anterior.

Para llevar a cabo esta crítica establece su *tesis de la indeterminación de la traducción* (véase Quine, W. O. [1960]), que será presentada a continuación y que pone de manifiesto la proximidad entre el tema de la sinonimia y el de la traducción.

Para Quine la noción de significado debe basarse en la conducta observable de los hablantes. Su propuesta se apoya en conceptos procedentes de la psicología conductista. En este sentido, define el significado como el conjunto de estimulaciones asociado a una oración que provoca una disposición a asentir o disentir dicha oración. Esta noción es conocida con el nombre de *significado estimulativo* y debe comprenderse en el marco de la *traducción radical*. Para explicar esta noción, Quine recurre al siguiente ejemplo imaginario: una situación de traducción radical sería aquella en la que se encuentra un lingüista que pretende traducir un lenguaje nativo completamente desconocido para él. En esta situación, el lingüista sólo puede observar la conducta del hablante ante determinadas estimulaciones verbales. Supóngase que, dado un determinado estímulo, el nativo utiliza una expresión que el lingüista desconoce por completo, por ejemplo, dice "gavagai" en el momento en que aparece en escena un conejo. El significado estimulativo de la expresión "gavagai" es el conjunto de estimulaciones asociado a dicha expresión que provoca la disposición del nativo a asentir o disentir cuando es pronunciada por el lingüista en una situación análoga a la descrita con el fin de averiguar si está acertado al sospechar que la expresión nativa "gavagai" podría traducirse por "conejo".

A partir de este experimento, que será tanto más fiable cuantas más veces se repita, el lingüista podrá, poco a poco y siempre con ciertas reservas, ir concluyendo que la expresión "gavagai" se puede traducir por la expresión "conejo". En este sentido ambas expresiones podrán considerarse sinónimas. Para Quine dos expresiones son *estimulativamente sinónimas* si se da una situación de igualdad de sus correspondientes significados estimulativos. El problema es que esto es una afirmación demasiado fuerte, ya que nunca podemos estar completamente seguros de que "gavagai" y "conejo" estén siendo utilizados de la misma forma para la misma cosa. Aunque se hagan concesiones, la definición de sinonimia estimulativa nunca podrá ser irrefutable, porque siempre habrá cierta incertidumbre en el mecanismo de referencia, que conduce a lo que Quine llama tesis de la indeterminación de la traducción, que se podría ejemplificar cuando varios lingüistas tratan de realizar distintas traducciones en idiomas diferentes. El resultado sería una serie de diccionarios de traducción compatibles cada uno de ellos con el comportamiento verbal del nativo pero incompatibles entre sí. Esta incompatibilidad se basa en lo que Quine llama *tesis de la indeterminación de la referencia*, que da cuenta del error en el que cae el lingüista al presuponer que la ontología a la que se refieren las expresiones del nativo es semejante a la que él posee. Al traducir "gavagai" por "conejo" se está presuponiendo que la manera de concebir el mundo que tiene el nativo es semejante a la del lingüista y esto no tiene por qué ser así. Cuando el nativo dice "gavagai" puede estar refiriéndose a una pata del conejo, la cabeza del conejo, un

ojo del conejo, o incluso al concepto de conejo. Si los significados fuesen entidades abstractas e independientes de cualquier lenguaje tal y como han sido concebidos tradicionalmente, esta indeterminación no podría darse porque todos los términos de cualquier idioma estarían asociados a sus respectivos significados de tal forma que estos proporcionarían el vínculo necesario para su traducción en cualquier otro idioma. Por un lado estarían los significados y por el otro las expresiones de cada idioma, a cada una de éstas le correspondería un significado de tal forma que una expresión *A* sería la traducción de otra *B* si a ambas les corresponde el mismo significado. Si las cosas fuesen de esta manera, la sinonimia funcionaría en la práctica como una relación de equivalencia, pero, según Quine, esto no sucede así.

La conclusión de Quine resulta interesante desde un punto de vista filosófico, pero en la práctica es posible establecer, de una forma razonable, cuándo dos expresiones están siendo utilizadas aproximadamente de la misma forma, y el problema de la traducción, a pesar de ser indeterminado, no lo es hasta el punto de que la traducción resulte imposible. Una forma de abordar este problema es la restricción a contextos concretos donde el conjunto de expresiones a traducir sea limitado pero resulte interesante desde el punto de vista de la comunicación humana. En la primera parte del próximo capítulo, se describirá una situación en la que un pequeño interfaz automático de traducción puede resultar útil para la realización de consultas sobre un horario de trenes. En este caso, el diálogo de pregunta y respuesta podrá ser realizado en castellano,

gallego o inglés y lo que vincula a cada una de las preguntas y respuestas de cada idioma es el objetivo en Prolog que les corresponde.

M. G. White también insiste en la necesidad de un estudio empírico del uso lingüístico pero lo hace de un modo relativamente distinto a como lo hacía Quine (véase White, M. G. [1950]). Parte de la búsqueda de un sinónimo de la palabra "sinónimo", para lo cual, dice que lo necesario en primer lugar es entender la palabra "sinónimo". Sin embargo, nadie ha propuesto un equivalente extensional que sea más claro que la propia palabra, y no será satisfactorio un criterio de sinonimia mientras no la explique con mayor claridad que el propio término. De todas formas, lo usual ha sido utilizarla para clarificar conceptos como por ejemplo el de "analiticidad". Según White, tal y como está planteado el problema, el enunciado "'todos los hombres son animales racionales' es analítico" es un enunciado empírico ya que para decidir si 'todos los hombres son animales racionales' es analítico tendremos que averiguar si "hombre" es sinónimo de "animal racional" y esto requiere un examen empírico del uso lingüístico. En este caso, parece no consistir en un examen de las estimulaciones asociadas a una oración, sino en un examen sobre si los hablantes de determinada lengua usan ambas expresiones como sinónimas o no. No estamos, por tanto, ante una situación de traducción radical como la descrita por Quine sino ante una situación de sinonimia intralingüística en la que el lingüista y el nativo se pueden entender.

White también trata el tema de la ambigüedad de las palabras del lenguaje natural y el lenguaje científico. Dice que una palabra puede tener muchos significados dependiendo del contexto y para analizar el problema recurre a John Stuart Mill, quien admite que un biólogo pueda considerar como sinónimo de "hombre" la expresión "animal mamífiero que tiene dos manos", pero también sostiene que, en el uso común, el sinónimo de "hombre" es "animal racional". Para Mill, el significado de una palabra depende de la situación en la que es utilizada. Según él, éste es un modo de pensar superior al de quien cree que existe un único significado verdadero. Por tanto, una palabra *X* es sinónima de otra palabra *Y* en una situación *S*. White critica a Mill diciendo que relativizar el problema a una situación *S* no es suficiente; todavía se necesita clarificar mejor cómo establecer la sinonimia en dicha situación.

Uno de los intentos de tratar la sinonimia de forma empírica dentro del lenguaje natural, no desde el marco de la traducción radical quineana, sino desde el punto de vista de cómo usan la sinonimia los hablantes de determinada lengua, se debe a A. Naess, quien, mediante conceptos definidos formalmente, considera la sinonimia como una relación entre los usos de una palabra en un contexto (véase Naess, A. [1949]). Utilizando una interpretación intuitiva, su principal objetivo es examinar los distintos factores que intervienen en el análisis y la comparación de expresiones respecto a su sinonimia. Partiendo del análisis de la forma que pueden tomar en el lenguaje natural los enunciados sobre sinonimia y de los factores que influyen en ésta, concluye proponiendo la utilización de

procedimientos empíricos basados en el establecimiento de cuestionarios como medios de interpretación de la misma. Estos procedimientos indican que dos expresiones tienen el mismo significado en determinados contextos y para determinados usuarios del lenguaje. Una vez analizada la forma de oraciones del lenguaje natural que expresan sinonimia como "… tiene el mismo significado que …", "… y … son sinónimas", etc., Naess considera las características del contexto al que pueden referir dichas oraciones. Esta caracterización le permite establecer la estructura del esquema conceptual de la sinonimia diciendo que la oración (o designación) *a* es, para la persona p_i, en el tipo de situación s_j, sinónima de la oración (o designación) *b*, para la persona p_m, en el tipo de situación s_n.

Naess considera otros conceptos relacionados con la sinonimia. La *interpretación* es uno de ellos, que caracteriza diciendo que si hay una oración interpretativa como "'*a*' significa *b*" entonces hay una oración de la forma "'*a*' significa lo mismo que '*b*'". Dado un conjunto de interpretaciones de una expresión, se llamarán *alternativas sinonímicas* a las expresiones que representan esas interpretaciones. Una expresión *A* es una *precisión* de otra expresión *B* cuando el conjunto de alternativas sinonímicas de *A* está incluido en el de *B*. Otros conceptos relacionados con la sinonimia son los dos tipos de *definición* que Naess concibe: la normativa, considerada como una declaración de sinonimia, y la descriptiva, considerada como una hipótesis de sinonimia.

En su análisis sobre la sinonimia, distingue varias versiones de ésta:

1. La sinonimia establecida o inferida a partir de una regla.
2. La sinonimia establecida mediante el uso hablado.
3. La sinonimia determinada por la identidad de las condiciones de verdad, la identidad de certidumbre, la identidad de probabilidad, etc.
4. La sinonimia representada por el parecido del estado argumental, lo que significa que dos expresiones son sinónimas si los argumentos que funcionan para una, funcionan también para la otra.

Naess está interesado sobre todo en los tres últimos, los cuales interpreta en términos de cuestionarios de tipo empírico, concluyendo que la sinonimia se comporta como una relación reflexiva, simétrica y transitiva en estos casos. Lo importante de su análisis es el examen meticuloso que proporciona sobre la interpretación y comparación del uso de expresiones en contextos del lenguaje natural, así como sus propuestas detalladas de estudio empírico para la comprobación de distintos tipos de sinonimia. Él mismo, consciente de la complejidad de este procedimiento empírico, sostiene que ha de ser utilizado como último recurso para poder dar una definición de lo que es la sinonimia.

Aunque se parece mucho a los métodos llevados a cabo para la elaboración de diccionarios, el método de Naess, al no estar pensado con fines prácticos, es más riguroso y exhaustivo que estos. Además, no afirma que cada tipo de sinonimia considerado sea una interpretación de la sinonimia en general, sino que son interpretaciones establecidas de forma cautelar para arrojar luz sobre ésta. La propuesta de Naess establece un vínculo entre los análisis lógico-

filosóficos y los lingüísticos; el problema de los primeros es que analizan la sinonimia en términos de sinonimia total, de tal forma que no representan realmente lo que es la sinonimia en los lenguajes naturales; los segundos tratan la sinonimia dentro del lenguaje natural pero no se interesan tanto por proporcionar una definición de la misma.

Una muestra de esta actitud, caracterizada por la falta de persecución de definiciones, puede verse en J. L. Austin, quien examina palabras describiendo e ilustrando sus usos en varios contextos, comparándolas y contrastándolas sin establecer ningún tipo de discusión explícita sobre los conceptos de similaridad o parecido entre palabras que está utilizando (véase Austin, J. L. [1961], capítulo 8). Cuando se examinan y comparan las circunstancias bajo las cuales se utilizan las palabras, se está, de modo indirecto, dando cuenta de cuestiones como la similaridad o sinonimia entre ellas. Austin parece presuponer que cuando tal cosa sucede, se sabe lo que es que dos expresiones sean utilizadas de una misma forma o parecida, sin que sea necesario un análisis previo de la noción de parecido que ello involucra; basta con examinar la relación mediante ejemplos concretos.

Del mismo modo que Austin, en el ámbito de la lingüística, S. Ullmann analiza la sinonimia presuponiendo que se sabe en qué consiste tal cosa (véase Ullmann, S. [1962]). La concibe en primer lugar como sinonimia total y después argumenta que usualmente las palabras del lenguaje natural no son sinónimas en este sentido. Se interesa por los usos de los sinónimos y los concibe como

expresiones que generalmente tienen el mismo significado pero que, en determinadas circunstancias, podrían no ser consideradas sinónimas. Afirma que el mejor método para la detección de sinonimia es el test de la sustitución, el cual, indica si los sinónimos son intercambiables; sin embargo, no entra en detalles sobre el uso de la sustitución. Además, afirma que se puede distinguir entre sinónimos buscando los antónimos de las palabras, pero tampoco da cuenta, explícitamente, de en qué consiste la antonimia.

3.- Hacia un estudio empírico de la sinonimia

Después de lo visto, parece haber dos opciones en el análisis de la sinonimia: una que se podría denominar teórica, que pretende proporcionar una explicación de este fenómeno mediante el establecimiento de una definición del mismo; y otra que se podría calificar de empírica, que pretende describirlo tal como se presenta en el uso lingüístico mediante estrategias como la introducción de cuestionarios sobre sinonimia, el análisis de textos o la observación de regularidades en el comportamiento lingüístico. Ambas tendencias parecen operar de forma recíproca. De los intentos explicativos de los trabajos teóricos surgen problemas que deberían ser tenidos en cuenta en los trabajos empíricos, los cuales, proporcionan descripciones de un fenómeno que podrían ser de ayuda para su definición. El presente trabajo es fundamentalmente una apuesta empírica que tratará de facilitar una descripción de cómo se comporta la sinonimia en un diccionario de sinónimos. Esto proporciona una visión parcial del problema, por

eso resulta de vital importancia seleccionar el diccionario de sinónimos más adecuado, que será aquel cuya calidad permita un análisis de la sinonimia que, aunque no sea completo, resulte suficientemente interesante y responda de modo correcto a las intuiciones de los hablantes sobre el comportamiento de esta relación en la práctica lingüística.

Antes de un estudio empírico es conveniente establecer un criterio previo para la sinonimia consistente en una caracterización intuitiva que indique el objeto de dicho estudio, de tal forma que un usuario medianamente competente de una lengua, sepa lo que se está queriendo decir cuando se utiliza el término "sinonimia". Podrían proporcionarse varias caracterizaciones de este tipo. Una de ellas correspondería a la definición de un diccionario para la entrada "sinónimo". Por ejemplo, en el de la *Real Academia Española* se indica de la siguiente forma: "dícese de los vocablos y expresiones que tienen una misma o muy parecida significación". Esta definición podría carecer de poder explicativo desde un punto de vista teórico, ya que hace descansar la sinonimia en un concepto poco claro como es la significación. Sin embargo, en la práctica, basta con una caracterización de este estilo para que un usuario competente de la lengua española sepa en qué consiste la sinonimia.

La intercambiabilidad podría ser considerada como criterio previo además de mostrar una de las utilidades de la sinonimia en la práctica. Usualmente se sustituyen unas expresiones sinónimas por otras dentro de una oración para evitar repeticiones en un discurso, para dar con el término que mejor expresa lo que

queremos decir o para reforzar y amplificar la expresión de un concepto. Independientemente de si en la sustitución se requiere la invariabilidad del valor de verdad, del significado o de la analiticidad de la oración, la intercambiabilidad no parece ser tanto una definición de la sinonimia como un indicio de la existencia de ésta. Indicio que no es fiable en todos los casos dado que no siempre es condición suficiente y no siempre es condición necesaria para la sinonimia. Que no es condición suficiente lo indica el hecho de que dos expresiones que no son sinónimas se puedan intercambiar en una oración sin que cambie el significado general de la misma. Por ejemplo, para jugar al ajedrez no importa la estatura que uno tenga, con lo que en la oración "no importa lo alto que seas para jugar al ajedrez", "alto" es intercambiable por "bajo" a pesar de que ambas expresiones no son sinónimas, con lo que el significado general de la oración, que podría ser expresado como "no importa la estatura para jugar al ajedrez", no se ve alterado por la sustitución, o al menos no lo hace en la misma medida que en las oraciones "Hermes es alto" y "Hermes es bajo". El hecho de que la sustituibilidad no sea condición necesaria para la sinonimia, tiene que ver con dos aspectos:

1. *El carácter gradual de la sinonimia*. Dos expresiones pueden ser sinónimas en cierto grado pero dicho grado no ser lo suficientemente alto como para que sean intercambiables en una oración dentro de un contexto concreto. Por ejemplo, coloquialmente, las palabras "fuerza" y "energía" pueden ser consideradas sinónimas en cierto grado, como

así lo establecen algunos diccionarios de sinónimos. Sin embargo, este grado no es tan alto como para que puedan ser sustituidas en una oración del contexto de la física donde el matiz que diferencia el significado de cada una de las palabras hace que el umbral exigido sea lo suficientemente alto como para que la sustitución no sea posible.

2. *La polisemia de las palabras*. En algunas ocasiones, dos expresiones sinónimas no pueden ser intersustituidas debido a que alguna de ellas es polisémica y en la oración no se emplea el significado que la hace sinónima de la otra. Por ejemplo, la palabra polisémica "banco" es sinónima en una de sus acepciones, de la palabra "cardumen"; sin embargo, aquélla no podrá ser sustituida por ésta en la oración "debo ir al banco para pedir un crédito hipotecario".

Una tercera caracterización de la sinonimia más cercana a la propuesta que se establecerá en este trabajo, podría ser la que considera sinónimas a dos palabras si figuran como tales en un diccionario de sinónimos. Este criterio proporciona una visión del objeto de estudio sesgada por el uso de un diccionario de sinónimos, el cual, podría ser considerado como un reflejo razonablemente fiel del uso de la sinonimia. Tras afirmar esto, debe señalarse lo siguiente:

1. En primer lugar, en un diccionario de sinónimos pueden no figurar pares de palabras que realmente sean sinónimas. Una posible solución a este problema es la de incorporar en el diccionario aquellos casos que no son contemplados. La cuestión es si resulta posible hacerlo con

todos los pares de palabras que son sinónimas de hecho. Da la impresión de que algunas características de los lenguajes naturales, como su carácter evolutivo y cambiante, y otras de la sinonimia, como su carácter gradual y dependiente del contexto, dificultan en buena medida esta labor que podría denominarse de *completud del diccionario*. Cuantos más casos de sinonimia se incorporen al diccionario, más completo será éste y mejor reflejará el uso de dicha relación en la práctica. Con respecto al resultado de este trabajo, su tratamiento computacional será lo suficientemente versátil como para que puedan ser incorporados nuevos pares de sinónimos o incluso pueda ser utilizado más de un diccionario de sinónimos. De todas formas, sólo se experimentará con un único diccionario y no se realizarán más modificaciones en éste que las que se deriven de errores tipográficos.

2. En muchas ocasiones, un diccionario no sólo da cuenta de la relación de sinonimia sino también de otras relaciones como la hiponimia, la hiperonimia, la meronimia, etc. Este problema puede ser debido al carácter vago y aproximado de la sinonimia. Las dificultades para su definición, a las que se hizo referencia en este capítulo, son una muestra de este carácter que provoca la falta de claridad a la hora de explicar en qué consiste realmente la sinonimia, lo que lleva, en numerosas ocasiones, a que sea confundida con otras relaciones

semánticas. Esta dificultad parece haber sido percibida también por J. Lyons cuando afirmaba que la relación que se da entre las palabras de un diccionario de sinónimos podría calificarse de *cuasisinonimia* más que de sinonimia propiamente dicha (véase Lyons, J. [1995]). Por este motivo, no es de extrañar que en el diccionario, como se verá en el capítulo cuarto de este trabajo, la sinonimia no satisfaga propiedades que, en principio, y de forma intuitiva, se podría afirmar que debería satisfacer, como por ejemplo la propiedad simétrica. Parece razonable pensar que si una palabra *A* es sinónima de otra *B*, *B* también lo será de *A*. Pues bien, en un diccionario de sinónimos esto no siempre sucede. Ello puede estar motivado en algunos casos por el olvido del lexicógrafo que elaboró el diccionario, pero en otros, la causa es que la relación que vincula a *A* y a *B* es una relación distinta de la sinonimia, como las citadas hiponimia, hiperonimia, etc., de las que intuitivamente se podría decir que no son simétricas.

Pero el uso de un diccionario de sinónimos tiene un atractivo especial que no tienen otros métodos empíricos como el análisis de textos, la introducción de cuestionarios, etc. Se trata de su adecuación al tratamiento computacional. Éste es uno de los motivos que más ha pesado para elegir este tipo de metodología y no otra. El hecho de que los diccionarios de sinónimos consistan en una colección de palabras, los hace incluso más idóneos para su automatización que los diccionarios convencionales, en los que a cada entrada le corresponde una

definición y no un conjunto de palabras. Dado el avance actual de los medios de computación, la implementación de un diccionario de sinónimos puede realizarse sin un coste excesivo, como se podrá comprobar en el capítulo cuarto de este trabajo.

El hecho de fundamentar el estudio empírico en el uso de un diccionario de sinónimos permite tener un conjunto relativamente grande de pares de palabras consideradas sinónimas. Una primera aproximación descriptiva indicará que hay cierto porcentaje de palabras que no satisfacen la propiedad simétrica. Acaban de realizarse algunas indicaciones sobre las posibles causas de ello. Por otra parte, hay un alto porcentaje de pares que no satisfacen la propiedad transitiva. Esto significa que una expresión *A* puede figurar como sinónima de otra *B* y ésta como sinónima de otra *C* y *A* no figurar como sinónima de *C*. Estos datos revelan que la sinonimia no debe ser entendida en la práctica como sinonimia total, es decir, como una relación de equivalencia. Concebirla como una relación gradual parece ser más coherente con los datos. Un modelo a utilizar podría ser el de las medidas de similaridad, ya que éstas no cumplen, en general, la propiedad transitiva, algo que parece acorde con el comportamiento gradual de la sinonimia en la práctica. De todas formas, las medidas de similaridad satisfacen, salvo algunos casos especiales (Véase Tversky, A. [1977]), la propiedad simétrica, lo que parece suponer cierto desajuste respecto a los datos extraídos del análisis empírico. La solución a este problema podría llevarse a cabo por dos vías: bien utilizando medidas de similaridad no simétricas como la

de Tversky, o bien tratando de averiguar por qué esta propiedad no se da. Ciertamente, una de las razones es que muchas de las palabras que figuran como sinónimas en el diccionario no son entradas del mismo; sin embargo también se dan muchos casos en los que una entrada figura como sinónima de otra sin que se dé la conversa. Este problema podría evitarse mediante la corrección del diccionario, una labor que no será llevada a cabo en la presente propuesta dado que esta tarea lexicográfica no entra dentro del alcance de este trabajo.

Además de revelar el carácter gradual de la sinonimia, el estudio empírico muestra también su carácter contextual. El hecho de poder hablar de la existencia de un grado de sinonimia entre expresiones hace pensar en la posibilidad de establecer umbrales de sinonimia más abajo de los cuales una palabra podría no ser considerada como sinónima en determinado contexto. De esta forma, habrá contextos donde se concede mucha importancia al más mínimo matiz diferenciador del significado de determinadas expresiones, lo que tiene como consecuencia que la precisión exigida a la hora de considerarlas sinónimas sea muy alta y consiguientemente también el umbral de sinonimia fijado. En este sentido no será de extrañar que dos expresiones sean consideradas sinónimas en un contexto y no lo sean en otro, algo que parece haber tenido presente Goodman cuando afirmaba que si decimos que dos términos son sinónimos, estamos indicando que su grado de sinonimia es suficiente para los propósitos del discurso inmediato (véase Goodman, N. [1949]). El citado ejemplo de las palabras “fuerza” y “energía” vuelve a ser aquí ilustrativo. Si el propósito del

discurso inmediato es poner en conocimiento de los interlocutores un estado de ánimo matutino, uno puede expresarlo de igual modo diciendo "esta mañana me levanté con fuerza" o "esta mañana me levanté con energía". Pero si el propósito del discurso inmediato es proporcionar una disertación sobre física teórica, no se podrá decir de igual modo "fuerza es igual a masa por aceleración" y "energía es igual a masa por aceleración". Esto significa que en el contexto de la física, el umbral exigido respecto a la sinonimia de ambas expresiones es lo suficientemente alto como para que "fuerza" y "energía" no sean consideradas sinónimas, mientras que en un contexto más coloquial, el umbral es menos restrictivo y consiguientemente ambas palabras podrán ser utilizadas de modo indistinto.

Además, tener en cuenta el contexto, puede contribuir a evitar un problema que surge parejo al análisis de la sinonimia: la polisemia. El hecho de que una expresión pueda tener varios significados o acepciones supone que antes de hablar de sus correspondientes sinónimos tenga que ser detectado el significado que realmente se está utilizando. Para ello, el contexto resulta de vital importancia, pues éste es suficiente, en la mayoría de las ocasiones, para indicar cuál de los significados de la expresión en cuestión se está utilizando. Una concepción de la sinonimia que no dé cuenta de este carácter contextual no está siendo fiel al comportamiento de ésta en la práctica.

La polisemia o ambigüedad de las palabras resulta problemática cuando éstas son utilizadas dentro de una oración, ya que, para que ésta pueda ser

entendida correctamente, es necesario detectar cuál es el significado adecuado de las palabras polisémicas que contiene. En algunas ocasiones, la propia oración proporciona pistas que contribuyen, de una forma relativamente fiable, a esta detección. Por ejemplo, en la oración "acabo de pedir un crédito hipotecario en el banco", se puede interpretar que la palabra polisémica "banco" se está empleando en su acepción de entidad financiera. Otras veces, es necesario recurrir a información extralingüística como en el caso de "nos vemos delante del banco que hay en mi calle", aquí los interlocutores deben conocer la citada calle para saber si la palabra "banco" se está empleando en su acepción de entidad financiera o en su acepción de asiento para varias personas. Aun así, en otras ocasiones, ni las oraciones, ni la información extralingüística son de ayuda para detectar las acepciones adecuadas de las palabras. La oración anterior sería un ejemplo de ello si en dicha calle hubiese un banco para sentarse y una sucursal de una entidad bancaria. Esto obligaría a los interlocutores a establecer un diálogo en el que tratarían de identificar las acepciones correctas de la palabra "banco", en otro caso, estarían abocados a no entenderse. Con todo, los humanos realizan esta detección de una forma relativamente cómoda y, usualmente, eficiente. Sin embargo, el tratamiento computacional de este problema es una labor altamente compleja si se persigue cierto grado de eficacia en la detección de las acepciones correctas. Este problema de la desambiguación automática de palabras será abordado en el próximo capítulo cuando se trate la sustituibilidad en el marco de la sinonimia oracional, donde se pondrá de manifiesto que parece no haber una

forma completamente eficiente de tratarlo, y que sólo se pueden aventurar algunos criterios poco nítidos para dar cuenta de él. En el caso del diccionario electrónico de sinónimos resultado de este trabajo, es el usuario quien debe seleccionar las acepciones de las entradas que son objeto de su búsqueda. A partir de esto, el procedimiento que realiza el cálculo del grado de sinonimia presentado en el próximo capítulo permitirá la detección automática de las acepciones de las palabras que proporciona como respuesta.

Otro de los objetivos de este trabajo es explorar si las características reveladas por el estudio empírico de la sinonimia, como el carácter gradual o el contextual pueden ser, de alguna forma, mecanizables. Históricamente los análisis de la sinonimia en el campo de la computación comienzan a partir de mediados del siglo XX, debido a que pueden ser de gran ayuda en cualquier entorno que conlleve cierto diálogo entre un humano y una máquina. Los trabajos pioneros eran más bien teóricos; los primeros intentos de aplicación de las ideas teóricas eran muy simples y poco voluminosos en cuanto a número de expresiones consideradas, debido al escaso desarrollo de los medios de computación; sin embargo, esta situación cambió al cabo de los años. Entre los trabajos pioneros se pueden destacar el de M. Masterman en la década de los cincuenta (véase Masterman, M. [1957]), así como el de K. Sparck Jones en los sesenta reeditado y modificado en Sparck Jones, K. [1986]. A finales de la década de los ochenta, E. Foxley y G. Gwei se proponen la generación automática de sinónimos basándose en el diccionario conceptual del inglés

Roget's Thesaurus (véase Foxley, E., Gwei, G. M. [1989]); y en la década de los noventa, B. Cases trata de formalizar un diccionario de sinónimos como un autómata que se automodifica con el fin de establecer clases de equivalencia de palabras sinónimas (véase Cases, B. [1996]). Más allá de estas propuestas, llevadas a cabo individualmente o por grupos de investigación relativamente pequeños, sin duda, la principal aportación a este campo proviene del extenso equipo que realizó el llamado proyecto WordNet (véase Fellbaum, C. [1998]). Este proyecto se originó en la Universidad de Princeton en 1985 tras unos 20 años de trabajos previos y su resultado fue la construcción de una gran base de datos electrónica de palabras en lengua inglesa. Posteriormente, entre 1996 y 1999, se ha desarrollado para distintas lenguas europeas con el nombre de EuroWordNet (véase Vossen, P. [1998]).

El objetivo de este trabajo será el cálculo automático del grado de sinonimia teniendo en cuenta los dos aspectos señalados sobre su carácter contextual: el umbral de sinonimia y la polisemia. Un diccionario de sinónimos asocia, a cada entrada (generalmente una palabra) una serie de acepciones que corresponden a cada uno de los significados de la misma, y a cada acepción un conjunto de sinónimos. El hecho de contar con conjuntos permite la utilización de las medidas de similaridad con las que se calcularán los grados, que serán números reales en el intervalo [0, 1], de tal forma que dos expresiones con grado 0 no serán sinónimas, mientras que dos expresiones con grado 1 serán totalmente sinónimas, cabiendo la posibilidad teórica de la existencia de infinitos grados

intermedios. Visto de esta forma, es posible construir un programa que calcule el grado de sinonimia de dos expresiones teniendo en cuenta el carácter polisémico de las entradas y contemplando la posibilidad de establecer umbrales de sinonimia. Para ello será necesario introducir la información que contiene el diccionario de sinónimos en una base de datos electrónica y diseñar un algoritmo que calcule el grado de sinonimia.

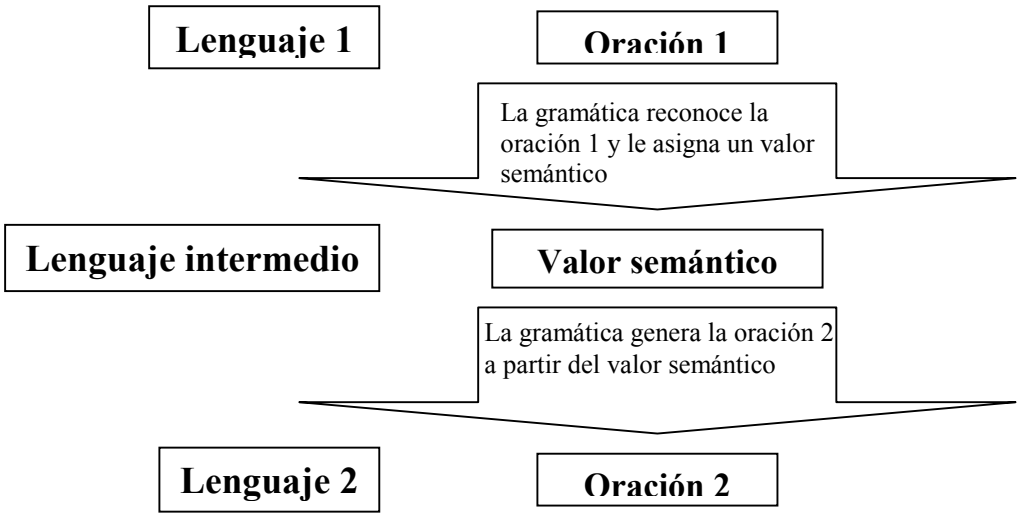
Se utilizará el lenguaje de programación Prolog para llevar a cabo esta propuesta, debido a las ventajas que este lenguaje tiene con respecto a otros para afrontar cuestiones relativas al procesamiento del lenguaje natural, algo que se ha mencionado ya en el capítulo anterior. El resultado será la creación de un diccionario electrónico de sinónimos que ofrece una descripción del comportamiento de la sinonimia en un diccionario de sinónimos publicado y que puede ser útil en general para los usuarios de la lengua, para los que traten de comprobar hipótesis teóricas sobre sinonimia o para los que estudian la sinonimia desde un punto de vista aplicado, como los lexicógrafos, lingüistas computacionales, etc.

CAPÍTULO III

Procesamiento automático de la sinonimia en los lenguajes naturales

En este capítulo se establecerán una serie de propuestas de automatización de la sinonimia en el lenguaje natural que responderán a los enfoques interlingüístico e intralingüístico, abordados respectivamente en las secciones primera y segunda. Tratar la sinonimia desde un punto de vista interlingüístico supone considerarla en términos de traducción. En cierto modo, es posible afirmar que dos expresiones de distintos idiomas son sinónimas si significan más o menos lo mismo. De esta forma, se admitirá que una de ellas es la traducción de la otra. Este vínculo entre ambos conceptos está presente ya en los trabajos pioneros sobre sinonimia del campo de la computación. Por ejemplo, Masterman utilizaba un tesauro para desambiguar palabras sobre un esquema de traducción de un texto de Virgilio (véase Masterman, M. [1957]).

En el ámbito de la traducción automática existen, dicho a grandes rasgos, dos enfoques: uno de ellos consiste en relacionar, de modo directo, las expresiones de un lenguaje con las de otro. El segundo consiste en traducir una expresión del primer lenguaje en un lenguaje intermedio, que permitirá la traducción al segundo de ellos. El análisis del significado en Prolog puede ser de gran ayuda para esta segunda forma de operar, ya que proporciona ese lenguaje intermedio necesario para que el proceso se lleve a cabo. La traducción de una expresión a dicho lenguaje intermedio correspondería al valor semántico asignado a la oración tras haber sido reconocida por una gramática. Debido a que el procedimiento de análisis del significado con Prolog es reversible, es decir, se puede pasar tanto de la oración al significado como del significado a la oración, una vez obtenido el valor semántico de la primera oración, se puede obtener su traducción en el segundo lenguaje generando la oración correspondiente a partir de dicho valor semántico. La siguiente figura expresa de una forma gráfica el procedimiento:



Sin embargo, debido a la complejidad del lenguaje natural, sólo se han obtenido resultados parciales. En el ámbito de la traducción automática surgen problemas como el que se da cuando las palabras no tienen una traducción exacta en otros idiomas o cuando una palabra tiene distintos significados según el contexto en el que es empleada. A pesar de todo, los sistemas de traducción automática son eficaces cuando se restringe su aplicación a contextos concretos de comunicación persona-máquina donde el conjunto de expresiones manejadas es pequeño y está bien definido. Para ilustrar esto, en el apartado primero se describirá un ejemplo en el que un humano podrá realizar preguntas en lenguaje natural a una máquina con respecto a un horario de trenes utilizando indistintamente el castellano, el gallego o el inglés. Para ello se recurrirá al procesamiento de gramáticas y a la gestión del significado en Prolog. El procedimiento a seguir será ilustrado previamente con un ejemplo sencillo.

La caracterización de la sinonimia desde el punto de vista intralingüístico ha sido abordada en el capítulo anterior. Como se ha indicado allí, el enfoque presente en este trabajo es más bien empírico. Eso conduce a fijar la atención en el uso lingüístico de la sinonimia más que en la búsqueda de su definición. También se ha dicho que un diccionario de sinónimos proporciona una herramienta empírica y computable muy útil para comprobar el comportamiento en la práctica de este aspecto del lenguaje. Mediante el uso de medidas de similaridad y con las estrategias de la programación lógica, en el segundo apartado se proporcionará un programa que calcula el grado de sinonimia entre

dos entradas de un diccionario de sinónimos. Basándose en el hecho de que uno de los objetivos del manejo de éste es la búsqueda de los sinónimos de una palabra para sustituirla por otra en una oración, se considerarán sinónimas dos oraciones cuando la segunda de ellas sea el resultado de la sustitución de algunas de las palabras de la primera por sus sinónimas. Se proporcionará un nuevo programa para tratar este tipo de sinonimia oracional cuya implementación ilustrará problemas como el de la polisemia y la desambiguación de palabras.

Finalmente, se analizarán otros dos casos de sinonimia entre oraciones. Uno de ellos es el de la pasiva: se dice que dos oraciones son sinónimas si una es la versión en pasiva de la otra. Un ejemplo de esto son las oraciones "Salieri odia a Mozart" y "Mozart es odiado por Salieri". El otro caso es el de las oraciones que contienen términos inversos. Ejemplifican términos inversos los pares "padre-hijo", "abuelo-nieto", "tío-sobrino", los cuales, producen sinonimia entre dos oraciones cambiando simplemente el orden de los argumentos como en "Luis es padre de Alberto" y "Alberto es hijo de Luis". La peculiaridad de estos dos últimos casos es que la sinonimia no resulta ser en ellos una cuestión de grado sino precisa.

1.- Sinonimia interlingüística en los lenguajes naturales: Traducción

Se comenzará describiendo un programa que ilustra el tratamiento computacional de la traducción de cuatro oraciones generadas por dos gramáticas correspondientes al español y al inglés. Con ello se pretende ejemplificar, de un

modo sencillo, cómo se puede tratar el tema de la traducción automática con Prolog. En el apartado 1.2 se aplicará este tratamiento al contexto específico de un horario de trenes.

1.1.- Un ejemplo sencillo

1.1.1.- Las gramáticas

Supongamos que tenemos dos gramáticas que generan respectivamente las oraciones del español "yo soy listo", "yo soy muy listo", "yo no soy listo", "yo no soy muy listo" y las oraciones del inglés "I am smart", "I am very smart", "I am not smart", "I am not very smart". La primera de ellas se podría especificar diciendo que una oración consta de un sujeto y un predicado. Un sujeto es un pronombre. Un predicado está compuesto por una cópula y un atributo o por una negación, una cópula y un atributo. Un atributo es un adjetivo solamente o un modificador y un adjetivo. La expresión "yo" es un pronombre, "no" es una negación, "soy" es una cópula, "muy" es un modificador y "listo" es un adjetivo. En el siguiente cuadro figura el código correspondiente:

```
%GRAMÁTICA PARA CUATRO ORACIONES DEL ESPAÑOL

oracion(Lista1,Resto):-
     sujeto(Lista1,Lista2),
     predicado(Lista2,Resto).

sujeto(Lista1,Resto):-
     pronombre(Lista1,Resto).

predicado(Lista1,Resto):-
     copula1(Lista1,Lista2),
     atributo(Lista2,Resto).
predicado(Lista1,Resto):-
     negacion(Lista1,Lista2),
```

```
        copula1(Lista2,Lista3),
        atributo(Lista3,Resto).

atributo(Lista1,Resto):-
        adjetivo(Lista1,Resto).
atributo(Lista1,Resto):-
        modificador(Lista1,Lista2),
        adjetivo(Lista2,Resto).

pronombre([yo|Resto],Resto).

negacion([no|Resto],Resto).

copula1([soy|Resto],Resto).

modificador([muy|Resto],Resto).

adjetivo([listo|Resto],Resto).
```

Las reglas de la segunda gramática indicarían que una oración (sentence) está compuesta por un sujeto (subject) y un predicado (predicate). Un sujeto (subject) es un pronombre (pronoun). Un predicado (predicate) está compuesto por una cópula (en el código se empleara el nombre de predicado "copula2" para evitar ser confundida con la cópula de la gramática del español) y un atributo (attribute) o por una cópula (copula2), una negación (negation) y un atributo (attribute). Un atributo (attribute) está compuesto por un adjetivo (adjective) solamente o por un modificador (modifier) y un adjetivo. La expresión "I" es un pronombre (pronoun), "am" es una cópula, "not" es una negación (negation), "very" es un modificador (modifier) y "smart" es un adjetivo (adjective):

```
%GRAMÁTICA PARA CUATRO ORACIONES DEL ESPAÑOL

sentence(List1,Rest):-
        subject(List1,List2),
        predicate(List2,Rest).

subject(List1,Rest):-
        pronoun(List1,Rest).

predicate(List1,Rest):-
```

```
        copula2(List1,List2),
        attribute(List2,Rest).
predicate(List1,Rest):-
        copula2(List1,List2),
        negation(List2,List3),
        attribute(List3,Rest).

attribute(List1,Rest):-
        adjective(List1,Rest).
attribute(List1,Rest):-
        modifier(List1,List2),
        adjective(List2,Rest).

pronoun(['I'|Rest],Rest).

copula2([am|Rest],Rest).

negation(['not'|Rest],Rest).

modifier([very|Rest],Rest).

adjective([smart|Rest],Rest).
```

1.1.2.- La gestión del significado

Supongamos que determinada oración de una de las lenguas es la traducción de otra en la otra lengua si su estructura semántica es la misma. Para determinar de forma automática si una de las oraciones es la traducción de la otra, es necesario que el sistema sea capaz de obtener dicha estructura semántica. En el primer capítulo de este trabajo se ha estudiado como hacerlo con Prolog. Allí se decía que la estructura semántica de una oración completa viene dada por el valor semántico asociado a cada uno de los componentes de la misma. De esta forma, al pronombre "yo" se le hará corresponder la expresión "yo" como valor semántico y al adjetivo "listo" la expresión "listo". A la negación se le asocia el predicado monádico "*no(X)*", donde la variable *X* será instanciada con el valor semántico de la cópula. A la cópula le corresponderá el predicado diádico

"*ser(X,Y)*", donde la variable *X* será instanciada con el valor semántico del sujeto de la oración (que coincide con el del pronombre) y la variable *Y* con el del atributo. Al modificador "muy" se le asocia el predicado monádico "*muy(X)*", donde la variable *X* será instanciada con el valor semántico del adjetivo. A un atributo le corresponde el valor semántico de un adjetivo cuando éste es su único componente; o depende del valor semántico de un modificador y un adjetivo si está formado por ambos. El sujeto tiene el mismo valor semántico que el pronombre que lo constituye. El del predicado dependerá del de la cópula y el atributo cuando estos sean sus dos componentes; o dependerá del de la negación, la cópula y el atributo cuando esté formado por estos tres componentes. Finalmente, la estructura semántica de la oración dependerá del valor semántico asignado al sujeto y al predicado.

Como se puede comprobar en el código que figura más abajo, a cada componente de la gramática que genera las oraciones en inglés se le asocia el mismo valor semántico que el de su análogo en la que se acaba de escribir para el español, de tal forma que una oración generada por la primera gramática puede tener la misma estructura semántica que otra generada por la segunda y viceversa. Cuando tal cosa sucede, cualquiera de estas oraciones es una traducción de la otra. El predicado `traduccion` da cuenta de este hecho. Posee dos argumentos, el primero es una oración generada por la gramática correspondiente al español y el segundo es una oración generada por la gramática correspondiente al inglés:

```
traduccion(Oracion,Sentence)
```

El predicado se define de la siguiente forma:

1. Una oración en español (*Orac*) es una traducción de otra oración en inglés (*Sent*) si:
   - *S* es la estructura semántica de *Orac*.
   - *S* también es la estructura semántica de *Sent*.

El código Prolog de todo el programa es el siguiente:

```
%GESTIÓN DEL SIGNIFICADO DE LA GRAMÁTICA ESPAÑOLA

oracion(S,Lista1,Resto):-
      sujeto(X,Lista1,Lista2),
      predicado(S,X,Lista2,Resto).

sujeto(S,Lista1,Resto):-
      pronombre(S,Lista1,Resto).

predicado(S,Y,Lista1,Resto):-
      copula1(S,Y,Z,Lista1,Lista2),
      atributo(Z,Lista2,Resto).
predicado(S,Y,Lista1,Resto):-
      negacion(S,X,Lista1,Lista2),
      copula1(X,Y,Z,Lista2,Lista3),
      atributo(Z,Lista3,Resto).

atributo(S,Lista1,Resto):-
      adjetivo(S,Lista1,Resto).
atributo(S,Lista1,Resto):-
      modificador(S,X,Lista1,Lista2),
      adjetivo(X,Lista2,Resto).

pronombre(yo,[yo|Resto],Resto).

negacion(no(X),X,[no|Resto],Resto).

copula1(ser(X,Y),X,Y,[soy|Resto],Resto).

modificador(muy(X),X,[muy|Resto],Resto).

adjetivo(listo,[listo|Resto],Resto).

%GESTIÓN DEL SIGNIFICADO DE LA GRAMÁTICA INGLESA

sentence(S,List1,Rest):-
      subject(X,List1,List2),
      predicate(S,X,List2,Rest).
```

```
subject(S,List1,Rest):-
      pronoun(S,List1,Rest).

predicate(S,Y,List1,Rest):-
      copula2(S,Y,Z,List1,List2),
      attribute(Z,List2,Rest).
predicate(S,Y,List1,Rest):-
      copula2(X,Y,Z,List1,List2),
      negation(S,X,List2,List3),
      attribute(Z,List3,Rest).

attribute(S,List1,Rest):-
      adjective(S,List1,Rest).
attribute(S,List1,Rest):-
      modifier(S,X,List1,List2),
      adjective(X,List2,Rest).

pronoun(yo,['I'|Rest],Rest).

copula2(ser(X,Y),X,Y,[am|Rest],Rest).

negation(no(X),X,['not'|Rest],Rest).

modifier(muy(X),X,[very|Rest],Rest).

adjective(listo,[smart|Rest],Rest).

%PREDICADO QUE PERMITE LA TRADUCCIÓN

traduccion(Orac,Sent):-
      oracion(S,Orac,[]),
      sentence(S,Sent,[]).
```

En cualquier consulta, instanciando con una oración correcta uno de los argumentos del predicado `traduccion` obtendremos su traducción en el otro idioma. Si se realiza la consulta con variables en ambos argumentos se obtienen todas las oraciones generadas y sus correspondientes traducciones:

```
?- traduccion(Oracion,Sentence).

Oracion = [yo,soy,listo]          Oracion = [yo,no,soy,listo]
Sentence = [I,am,smart]           Sentence = [I,am,not,smart]

Oracion = [yo,soy,muy,listo]      Oracion = [yo,no,soy,muy,listo]
Sentence = [I,am,very,smart]      Sentence = [I,am,not,very,smart]
```

1.2.- Aplicación en un contexto concreto

A continuación se presentará un pequeño contexto de diálogo persona-máquina en lenguaje natural donde un usuario podrá utilizar tres idiomas diferentes: español, gallego e inglés. El contexto es un horario de trenes sobre el que un usuario podrá hacer determinadas preguntas. El sistema es capaz de responder en el mismo idioma en el que le fue preguntado. Para ello, la estructura semántica de una pregunta será un objetivo que se lanza contra el programa Prolog que implementa el horario. Las variables a instanciar son aquello que el usuario desea saber y sus instancias son la respuesta. Ante la consulta, el sistema responde con una oración en lenguaje natural a la que se le ha asociado la misma estructura semántica que a la pregunta, pero con las variables de la respuesta instanciadas. El programa consta de dos partes: la primera es el código que implementa el horario de trenes; la segunda está constituida por tres gramáticas, una para cada idioma considerado, que actúan a modo de interfaz entre el usuario y el programa de la primera parte.

1.2.1.- Implementación del horario de trenes

El predicado `viaje` posee como argumentos los siguientes datos de un viaje en tren: el tipo de tren, el lugar de salida, la hora de salida, el lugar de llegada, la hora de llegada y el tiempo empleado en realizar el viaje:

```
viaje(Tipo,Lugar_sal,Hora_sal,Lugar_lleg,Hora_lleg,Tiempo)
```

Como el horario de trenes posee dos direcciones (A Coruña-Vigo y Vigo-A Coruña) se ha construido una cláusula para cada dirección. Para la dirección A Coruña-Vigo:

1. Se puede realizar un viaje cuyos datos son *Tipo*, *Lugar_salida*, *Hora_salida*, *Lugar_llegada*, *Hora_llegada*, *Tiempo* si:
   - *Horario* es la lista con el horario del tren del tipo *Tipo*.
   - *Num1* es el número de la estación correspondiente al lugar de salida.
   - *Hora_salida* es el elemento de la lista *Horario* cuyo número de orden es *Num1*.
   - *Hora_salida* es distinto de "no".
   - *Num2* es el número de la estación correspondiente al lugar de llegada.
   - *Hora_llegada* es el elemento de la lista *Horario* cuyo número de orden es *Num2*.
   - *Hora_llegada* es distinto de "no".
   - *Num1* es menor que *Num2*.
   - *Tiempo* es lo que tarda el tren entre la hora de salida y la hora de llegada.

Para la dirección Vigo-A Coruña:

2. Se puede realizar un viaje cuyos datos son *Tipo*, *Lugar_salida*, *Hora_salida*, *Lugar_llegada*, *Hora_llegada*, *Tiempo* si:

- Las condiciones son las mismas que para la dirección A Coruña-Vigo con la diferencia que en este caso hay que invertir el número de orden de las estaciones. (La estación 31 pasa a ser ahora la primera, la 30 pasa a ser la segunda y así sucesivamente).

Con los predicados `tren1` y `tren2` se constituyen dos bases de datos en las que se indican el tipo de tren y el horario que realiza éste para la dirección A Coruña-Vigo (`tren1`) y Vigo-A Coruña (`tren2`). Poseen, por tanto, dos argumentos cada uno. El primero es el nombre del tren, que se instanciará según tres posibilidades: "Regional", "Costa de Galicia" y "TRD". El segundo es una lista en la que aparecen las horas de parada del tren en cada una de las 31 estaciones. Por ejemplo, si el tercer elemento de esta lista es 6.45, significa que ésa es la hora de llegada-salida de la tercera estación en la que dicho tren para. Si el tren no para en determinada estación, en lugar de una hora, aparece la expresión "no":

```
tren1(Tipo_de_tren,Lista_horario)
tren2(Tipo_de_tren,Lista_horario)
```

Con el predicado `estacion` se crea otra base de datos con los nombres de las estaciones y su correspondiente número de orden en la dirección A Coruña-Vigo. Cuando se considera la direción Vigo-A Coruña es necesario invertir los números de orden de las estaciones como se puede observar en la tercera y séptima condiciones de la segunda cláusula del predicado `viaje`. Los argumentos del predicado `estacion` son dos, el nombre de la estación y el número de orden:

```
estacion(Nombre,Numero_de_orden)
```

El predicado `tarda` realiza la diferencia de tiempo entre dos horas del día. Consta de tres argumentos, los dos primeros son dos horas y el tercero la diferencia entre ellas:

```
tarda(Hora1,Hora2,Diferencia)
```

El procedimiento consiste en hallar la diferencia entre la parte correspondiente a las horas de *Hora1* y *Hora2* y la parte correspondiente a los minutos. Si esta última diferencia es positiva, es decir, mayor o igual a 0, la diferencia de tiempo es la concatenación de la diferencia de las horas con la diferencia de los minutos (añadiendo un punto en el medio). Si es negativa hay que sumarle a 60 ese número negativo y restar 1 en la diferencia de la parte de las horas antes de concatenar las diferencias de ambas partes. Se definirá el predicado para estos dos casos de la siguiente forma:

1. *C* es la diferencia de tiempo entre *A* y *B* si:
    - *D* es la división entera de *A* entre 1.
    - *E* es la división entera de *B* entre 1.
    - *F* es la diferencia entre *A* y *D*.
    - *G* es la diferencia entre *B* y *E*.
    - *H* es la diferencia entre *E* y *D*. (Ésta es la diferencia de la parte de las horas).
    - *I* es la diferencia entre *G* y *F*. (Ésta es la diferencia de la parte de los minutos).
    - *I* es mayor o igual a 0.

- *C* es la concatenación de la diferencia de la parte de las horas *H* y la diferencia de la parte de los minutos *I*.

2. *C* es la diferencia de tiempo entre *A* y *B* si:
   - Las seis primeras condiciones de la cáusula 1 se cumplen.
   - *I* es menor que 0. (En este caso la diferencia de la parte de los minutos es un número negativo).
   - *J* es igual a sumarle a 0.60 el número negativo *I*.
   - *K* es el resultado de restarle 1 a la diferencia de la parte de las horas *H*.
   - *C* es la concatenación de la diferencia modificada de la parte de las horas *K* y la diferencia modificada de la parte de los minutos *J*.

El predicado `h_m` concatena las diferencias de las partes de las horas y partes de los minutos poniendo un punto en medio de ellas. Posee tres argumentos, una parte de las horas, una parte de los minutos y el resultado de la concatenación:

```
h_m(Diferencia_horas,Diferencia_minutos,Concatenacion)
```

Si la parte de los minutos es 0, la expresión que hay que concatenarle a la parte de las horas es ".00". Si la parte de los minutos es un número entre 0 y 1 con una sola cifra decimal *A*, la expresión que hay que concatenarle a la parte de las horas es ".*A*0". Si la parte de los minutos es un número entre 0 y 1 con dos cifras decimales *A* y *B*, la expresión que hay que concatenarle a la parte de las

horas es "*.AB*". La siguiente definición del predicado, da cuenta de estos tres casos:

1. *C* es la concatenación de la parte de las horas *A* y la parte de los minutos *B* si:
   - *D* es la lista de caracteres ASCII de *A*.
   - La lista de caracteres ASCII de la parte de los minutos *B* es [48] (es decir, la parte de los minutos es 0).
   - *F* es el resultado de concatenar la lista *D* con la lista [46,48,48] (es decir, ".00").
   - *C* es la expresión correspondiente a la lista *F* de caracteres ASCII.
2. *C* es la concatenación de la parte de las horas *A* y la parte de los minutos *B* si:
   - Las condiciones son análogas a las de 1 salvo en que la lista de caracteres ASCII de la parte de los minutos *B* es [48,46,*E*] (es decir, la parte de los minutos es un número entre 0 y 1 con una sola cifra decimal) y *F* es el resultado de concatenar la lista *D* con la lista [46,*E*,48] (es decir, "*.E*0").
3. *C* es la concatenación de la diferencia de la parte de las horas *A* y la parte de los minutos *B* si:
   - Las condiciones son análogas a las de 1 salvo en que la lista de caracteres ASCII de la diferencia de la parte de los minutos *B* comienza por 48 ("0") (es decir, la diferencia de los minutos es un

número entre 0 y 1 con más de una cifra decimal) y *F* es el resultado de concatenar la lista *D* con la cola de la lista de caracteres ASCII de la diferencia de los minutos.

El código Prolog correspondiente al programa figura en el siguiente cuadro:

```
%PROGRAMA PARA EL HORARIO DE TRENES

viaje(Tipo,Lugar_salida,Hora_salida,Lugar_llegada,Hora_llegada,
      Tiempo):-
            tren1(Tipo,Horario),
            estacion(Lugar_salida,Num1),
            numorden(Hora_salida,Horario,Num1),
            Hora_salida \== 'no',
            estacion(Lugar_llegada,Num2),
            numorden(Hora_llegada,Horario,Num2),
            Hora_llegada \== 'no',
            Num1 < Num2,
            tarda(Hora_salida,Hora_llegada,Tiempo).
viaje(Tipo,Lugar_salida,Hora_salida,Lugar_llegada,Hora_llegada,
      Tiempo):-
            tren2(Tipo,Horario),
            estacion(Lugar_salida,Num1),
            X is (31-Num1)+1,
            numorden(Hora_salida,Horario,X),
            Hora_salida \== 'no',
            estacion(Lugar_llegada,Num2),
            Y is (31-Num2)+1,
            numorden(Hora_llegada,Horario,Y),
            Hora_llegada \== 'no',
            Num1 > Num2,
            tarda(Hora_salida,Hora_llegada,Tiempo).

tren1('Regional',[no,no,no,no,no,no,no,no,no,no,no,no,6.20,no,
      no,no,6.38,6.41,6.49,6.58,no,7.13,no,7.38,no,7.48,8.04,
      8.08,8.10,8.15,8.21]).
tren1('TRD',[6.20,no,no,no,no,no,no,no,no,no,no,no,no,7.12,no,no,
      no,no,no,no,7.42,no,no,no,8.07,no,no,no,no,no,no,8.38]).
tren1('Regional',[6.30,6.39,6.45,6.53,6.58,7.01,7.06,7.11,7.16,
      7.20,7.24,7.30,7.44,7.49,7.54,8.00,8.08,8.12,8.20,8.31,
      8.45,8.51,8.59,9.10,9.23,9.28,9.32,9.35,9.38,9.44,9.52]).
tren1('Costa de Galicia',[7.30,no,no,7.49,7.53,7.57,no,no,no,no,
      no,no,8.38,no,no,no,8.56,9.00,9.09,9.18,no,no,no,9.51,no,
      no,no, 10.08,10.11,no,10.23]).
tren1('TRD',[8.20,no,no,no,no,no,no,no,no,no,no,no,no,9.15,no,no,
      no,no,no,no,9.45,no,no,no,10.10,no,no,no,no,no,no,10.38]).
tren1('Costa de Galicia',[9.30,no,no,no,no,no,no,no,no,no,no,no,
```

```
      10.22,10.39,no,no,no,10.56,11.00,11.08,11.17,no,11.29,no,
      11.50,no,11.59,12.04,no,12.10,12.15,12.20]).
tren1('TRD',[10.20,no,no,no,no,no,no,no,no,no,no,no,11.14,no,no,
      no,no,no,no,11.44,no,no,no,12.10,no,no,no,no,no,no,12.36]).
tren1('Costa de Galicia',[11.30,no,no,no,no,no,no,no,no,no,no,
      12.21,12.39,no,no,no,12.57,13.02,no,13.17,no,13.29,no,
      13.51,no,13.59,14.03,no,14.09,14.14,14.21]).
tren1('TRD',[12.20,no,no,no,no,no,no,no,no,no,no,no,13.14,no,no,
      no,no,no,no,13.44,no,no,no,14.10,no,no,no,no,no,no,14.36]).
tren1('Regional',[no,no,no,no,no,no,no,no,no,no,no,no,13.35,no,
      no,no,13.53,13.56,14.05,14.14,no,14.24,no,14.45,no,14.53,
      14.56, 15.00,15.03,15.10,15.16]).
tren1('Costa de Galicia',[13.30,13.38,no,13.49,13.54,13.58,no,
      no,no,no,no,14.23,14.39,no,no,no,14.55,15.00,15.08,15.17,
      no,no, no,15.51,no,15.58,no,no,16.07,no,16.17]).
tren1('TRD',[14.20,no,no,no,no,no,no,no,no,no,no,no,15.14,no,no,
      no,no,no,no,15.44,no,no,no,16.10,no,no,no,no,no,no,16.36]).
tren1('Regional',[no,no,no,no,no,no,no,no,no,no,no,no,15.35,no,
      no,no,15.53,15.56,16.05,16.14,no,16.24,no,16.45,no,16.53,
      16.56,16.59,17.01,17.09,17.15]).
tren1('Costa de Galicia',[15.30,no,no,15.49,15.54,15.58,no,no,
      no,no,no,no,16.39,no,no,no,16.57,17.01,no,17.16,no,17.28,
      no,17.51,no,no,18.04,no,18.09,18.15,18.26]).
tren1('TRD',[16.20,no,no,no,no,no,no,no,no,no,no,no,17.14,no,no,
      no,no,no,no,17.44,no,no,no,18.10,no,no,no,no,no,no,18.36]).
tren1('Regional',[no,no,no,no,no,no,no,no,no,no,no,no,17.35,no,
      no,no,17.51,17.55,18.05,18.13,no,18.25,no,18.46,no,18.54,
      18.57, 19.00,19.03,19.10,19.17]).
tren1('Costa de Galicia',[17.30,17.39,no,no,no,no,no,no,no,no,
      no,no,18.39,no,no,no,18.56,19.00,no,19.16,no,19.28,no,
      19.51,no, 19.59,20.04,20.07,20.09,20.17,20.25]).
tren1('TRD',[18.20,no,no,no,no,no,no,no,no,no,no,no,19.14,no,no,
      no,no,no,no,19.44,no,no,no,20.11,no,no,no,no,no,no,20.35]).
tren1('Costa de Galicia',[18.36,no,no,18.55,19.02,19.05,no,no,
      no,no,no,19.29,19.43,no,no,no,20.05,20.08,20.16,20.25,no,
      no,no, 20.53,no,21.01,no,21.07,21.09,21.14,21.23]).
tren1('TRD',[19.45,no,no,no,no,no,no,no,no,no,no,no,20.39,no,no,
      no,no,no,no,21.09,no,no,no,21.35,no,no,no,no,no,no,22.01]).
tren1('Regional',[19.50,no,no,no,no,no,no,no,no,no,no,no,21.06,
      no,no,no,21.38,21.41,no,21.57,no,no,no,22.35,no,no,no,no,
      22.57, no,23.10]).
tren1('Regional',[20.10,20.18,20.24,20.31,20.40,20.43,20.47,
      20.56,21.01,21.05,21.09,21.15,21.49,21.54,21.59,22.05,
      22.09,22.12,22.20,22.29,22.34,22.40,22.48,22.58,23.03,
      23.08,23.12, 23.15,23.17,23.22,23.28]).
tren1('Regional',[22.25,22.34,no,22.45,22.54,22.57,no,no,23.13,
      no,no,23.25,23.38,no,no,no,no,no,no,no,no,no,no,no,no,no,
      no,no, no,no,no]).

tren2('Regional',[no,no,no,no,no,no,no,no,no,no,no,no,no,no,no,
      no,no,no,6.25,6.37,no,no,6.49,no,no,7.10,7.14,7.19,no,7.30,
      7.50]).
tren2('Regional',[6.00,6.05,6.10,6.12,6.15,6.19,6.24,6.29,6.39,
```

```
      6.47,6.54,7.00,7.08,7.15,7.17,7.25,7.32,7.37,7.45,7.57,
      8.03, 8.07,8.13,8.18,8.26,8.31,8.34,8.38,8.46,8.51,9.00]).
tren2('TRD',[6.55,no,no,no,no,no,no,7.18,no,no,no,7.43,no,no,no,
      no,no,no,8.15,no,no,no,no,no,no,no,no,no,no,no,9.16]).
tren2('Costa de Galicia',[7.40,7.45,7.51,7.53,no,7.59,no,8.09,
      no,8.27,no,8.39,8.47,8.54,8.57,no,no,no,9.16,no,no,no,no,
      no,no, 10.07,10.10, 10.14,no,no,10.35]).
tren2('Regional',[8.08,8.14,8.23,8.29,no,8.35,no,8.45,no,no,no,
      9.19,no,9.46,9.49,no,no,no,10.07,no,no,no,no,no,no,no,no,
      no,no, no,no]).
tren2('TRD',[8.55,no,no,no,no,no,no,9.17,no,no,no,9.43,no,no,no,
      no,no,no,10.16,no,no,no,no,no,no,no,no,no,no,no,11.13]).
tren2('Costa de Galicia',[9.40,no,9.50,9.52,9.55,no,no,10.10,no,
      no,no,10.39,no,10.54,10.57,no,no,no,11.16,11.29,no,no,no,
      no,no, 12.07,12.10,12.14,no,no,12.35]).
tren2('TRD',[10.55,no,no,no,no,no,no,11.17,no,no,no,11.43,no,no,
      no,no,no,no,12.15,no,no,no,no,no,no,no,no,no,no,no,13.15]).
tren2('Costa de Galicia',[11.40,11.45,11.50,no,no,no,no,12.09,
      no,no,no,12.39,no,no,12.57,no,no,no,13.16,13.29,no,no,
      13.50,no, no,no,no,no,no,14.24,14.35]).
tren2('TRD',[12.55,no,no,no,no,no,no,13.17,no,no,no,13.43,no,no,
      no,no,no,no,14.15,no,no,no,no,no,no,no,no,no,no,no,15.13]).
tren2('Regional',[no,no,no,no,no,no,no,no,no,no,no,no,no,no,no,no,
      no,no,no,14.46,14.58,no,no,15.17,no,15.26,15.31,15.35,
      15.41,no, 16.00,16.10]).
tren2('Costa de Galicia',[13.40,13.45,13.50,13.53,13.56,14.00,
      no,14.09,no,14.26,no,14.38,14.46,no,14.57,no,no,no,15.16,
      no,no, no,no,no,no,16.07,16.11,16.15,no,no,16.35]).
tren2('TRD',[14.55,no,no,no,no,no,no,15.17,no,no,no,15.43,no,no,
      no,no,no,no,16.15,no,no,no,no,no,no,no,no,no,no,no,17.13]).
tren2('Costa de Galicia',[15.40,no,15.50,no,15.56,16.00,no,
      16.09,no,16.26,no,16.38,16.46,16.53,16.57,no,no,no,17.16,
      17.29, no,no,no,no,no,no,no,no,no,no,18.35]).
tren2('TRD',[16.55,no,no,no,no,no,no,17.17,no,no,no,17.43,no,no,
      no,no,no,no,18.15,no,no,no,no,no,no,no,no,no,no,no,19.14]).
tren2('Costa de Galicia',[17.40,17.45,17.50,17.53,17.56,18.00,
      no,18.09,no,no,no,18.37,18.45,18.53,18.57,no,no,no,19.16,
      19.27, no,no,no,no,no,19.53,19.56,20.00,no,no,20.30]).
tren2('TRD',[18.55,no,no,no,no,no,no,19.17,no,no,no,19.43,no,no,
      no,no,no,no,20.15,no,no,no,no,no,no,no,no,no,no,no,21.17]).
tren2('Costa de Galicia',[19.20,no,19.30,no,no,no,no,19.49,no,
      20.10,no,20.23,no,20.41,20.44,no,no,no,21.09,21.21,no,no,
      no,no, no,21.45,21.49,21.53,no,22.03,22.13]).
tren2('Regional',[20.00,20.05,20.11,20.13,20.16,20.20,20.27,
      20.32,20.42,20.52,20.59,21.05,21.17,21.24,21.27,21.31,
      21.38,21.43,21.51,22.05,22.18,22.21,22.25,22.30,22.35,
      22.41,22.45, 22.49,22.55,23.01,23.12]).
tren2('TRD',[21.10,no,no,no,no,no,no,21.33,no,no,no,22.00,no,no,
      no,no,no,no,22.31,no,no,no,no,no,no,no,no,no,no,no,23.27]).
tren2('Regional',[21.32,21.37,21.50,21.53,21.56,22.00,no,22.10,
      no,22.29,no,22.52,23.00,23.07,23.11,no,no,no,23.29,no,no,
      no,no, no,no,no,no,no,no,no,24.40]).
```

```
estacion('A Coruña',1).
estacion('Uxes',2).
estacion('Bregua',3).
estacion('Meirama',4).
estacion('Vila da igrexa',5).
estacion('Cerdeda',6).
estacion('Queixas-Londoño',7).
estacion('Gorgullos-T',8).
estacion('Ordes-Pontraga',9).
estacion('Garga-Trasmonte',10).
estacion('Oroso-Vilacide',11).
estacion('Berdía',12).
estacion('Santiago',13).
estacion('Casal',14).
estacion('Osebe',15).
estacion('A Escravitude',16).
estacion('Padrón',17).
estacion('Pontecesures',18).
estacion('Catoira',19).
estacion('Vilagarcía de Arousa',20).
estacion('Rubiáns',21).
estacion('Portas',22).
estacion('Portela',23).
estacion('Pontevedra',24).
estacion('Figueirido',25).
estacion('Arcade',26).
estacion('Cesantes',27).
estacion('Redondela-Picota',28).
estacion('Redondela de G.',29).
estacion('Chapela',30).
estacion('Vigo',31).

tarda(A,B,C):-
      D is A//1,
      E is B//1,
      F is A-D,
      G is B-E,
      H is E-D,
      I is G-F,
      I >= 0,
      h_m(H,I,C),
      !.
tarda(A,B,C):-
      D is A//1,
      E is B//1,
      F is A-D,
      G is B-E,
      H is E-D,
      I is G-F,
      I < 0,
      J is 0.60+I,
      K is H-1,
      h_m(K,J,C),
```

```
      !.

h_m(A,B,C):-
      name(A,D),
      name(B,[48]),
      concatenar(D,[46,48,48],F),
      name(C,F),
      !.
h_m(A,B,C):-
      name(A,D),
      name(B,[48,46,E]),
      concatenar(D,[46,E,48],F),
      name(C,F),
      !.
h_m(A,B,C):-
      name(A,D),
      name(B,[48|E]),
      concatenar(D,E,F),
      name(C,F),
      !.

numorden(A,[A|C],1).
numorden(A,[B|C],Num):-
      numorden(A,C,N),
      Num is N+1.

concatenar([],X,X).
concatenar([A|B],C,[A|D]):-
      concatenar(B,C,D).
```

(Los predicados `numorden` y `concatenar` empleados en este programa, han sido definidos en el primer capítulo de este trabajo).

Se pueden hacer distintas consultas con el predicado `viaje`. Por ejemplo, "¿A qué hora llega a Santiago (o cualquier otro lugar de llegada) un tren que sale de Vigo (o cualquier otro lugar de salida) a las 11.40 (o cualquier otra hora de salida)?":

```
?- viaje(_,'Vigo',11.40,'Santiago',X,_).

X = 13.16
```

"¿A qué hora sale de Santiago (o cualquier otro lugar de salida) un tren que llega a Vigo (o cualquier otro lugar de llegada) a las 17.15 (o cualquier otra hora de llegada)?":

```
?- viaje(_,'Santiago',X,'Vigo',17.15,_).

X = 15.35
```

"¿Cuánto tarda en llegar a Santiago (o cualquier otro lugar de llegada) un tren que sale de A Coruña (o cualquier otro lugar de salida) a las 7.30 (o cualquier otra hora de salida)?":

```
?- viaje(_,'A Coruña',7.30,'Santiago',_,X).

X = 1.08
```

"¿Qué trenes van de A Coruña (o cualquier otro lugar de salida) a Santiago (o cualquier otro lugar de llegada)?"

```
?- viaje(X,'A Coruña',Y,'Santiago',Z,W).

X = TRD                 X = Costa de Galicia  X = Costa de Galicia
Y = 6.20                Y = 7.30              Y = 9.30
Z = 7.12                Z = 8.38              Z = 10.39
W = 0.52                W = 1.08              W = 1.09

X = Regional            X = TRD               X = TRD
Y = 6.30                Y = 8.20              Y = 10.20
Z = 7.44                Z = 9.15              Z = 11.14
W = 1.14                W = 0.55              W = 0.54
                                              ...
```

1.2.2.- Las gramáticas para el horario

Las gramáticas que se presentarán a continuación funcionan a modo de interfaz entre un usuario y el programa que se acaba de describir posibilitando un diálogo pregunta-respuesta en español, gallego o inglés. Dicho diálogo es relativamente rígido, pero se puede flexibilizar incorporando más opciones de

pregunta y respuesta. La cuestión es que para hacer una consulta al programa que implementa el horario de trenes, tenemos que utilizar expresiones del estilo de "`viaje(_,'Santiago',X,'Vigo',17.15,_).`", siendo necesario que el usuario conozca al menos la sintaxis de las consultas en Prolog para poder utilizar el programa. Con el fin de evitar esta necesidad, se hará corresponder como valor semántico de determinadas preguntas en lenguaje natural, las cláusulas Prolog de los objetivos que verbalizan dichas preguntas.

Por otra parte, la respuesta del compilador a la consulta indicada arriba es del estilo "`X = 15.35`", que se podría verbalizar como "sale a las 15.35" o, de una forma más completa, y repitiendo la información proporcionada en la pregunta "Un tren sale de Santiago a las 15.35 y llega a Vigo a las 17.15". Incluso se podría añadir información que no figura en la pregunta pero que el programa puede incluir en la respuesta, como el tipo de tren o lo que tarda éste en realizar el trayecto. Se proporcionará la verbalización de la respuesta a la consulta asociando, una vez más, el predicado `viaje` como valor semántico de cada respuesta.

El predicado `horario` escribe un listado de posibles preguntas en español, gallego e inglés antes de preguntarle al usuario cuál es la pregunta que desea realizar. Una vez que el usuario introduce la pregunta, el programa obtiene todas sus respuestas y las escribe en la pantalla en el mismo idioma de aquélla. Después de escribirlo, consulta al usuario si desea realizar más preguntas; en caso de respuesta afirmativa el proceso se repetirá.

El predicado `escribir` escribe en la pantalla cada uno de los elementos de una lista. Su único argumento es dicha lista:

```
escribir(Lista)
```

El predicado se define de la siguiente forma:

1. Si la lista es vacía, no se escribe nada en la pantalla.
2. Si la lista no es vacía:
   - Escribe la cabeza de la lista.
   - Escribe dos líneas en blanco.
   - Repítase el mismo procedimiento con la cola de la lista.

El predicado `mas_preg` pregunta al usuario si desea hacer más consultas al programa. Tras escribir dos líneas en blanco, el compilador escribe en pantalla "¿Más preguntas? / ¿Máis preguntas? / More questions?" y queda a la espera de la respuesta del usuario. Si la respuesta es afirmativa, el código correspondiente al predicado `horario` volverá a ejecutarse. Para responder afirmativamente a esta pregunta basta con teclear "s" o "y" tanto en mayúsculas como en minúsculas así como "si", "Si", "yes" o "Yes".

El predicado `pregunta_respuesta` empleado como argumento del `findall` que aparece en el predicado `horario` (véase el código más adelante), proporciona la respuesta a una pregunta sobre el horario de trenes en el mismo idioma en que fue realizada la pregunta. Este predicado posee dos argumentos, una pregunta en lenguaje natural y su correspondiente respuesta:

```
pregunta_respuesta(Pregunta,Respuesta)
```

Este predicado se define de la siguiente forma:

1. *D* es la respuesta a la pregunta *A* si:
    - *C* es la lista de códigos ASCII de la pregunta *A*.
    - Se da una de estas tres opciones:
        - *Sgdo* es el valor semántico asociado a la lista de códigos ASCII *C* de la pregunta en español y *Sgdo* es el valor semántico asociado a la lista de códigos ASCII *B* de la respuesta o respuestas en español de la pregunta; o bien,
        - *Sgdo* es el valor semántico asociado a la lista de códigos ASCII *C* de la pregunta en gallego y *Sgdo* es el valor semántico asociado a la lista de códigos ASCII *B* de la respuesta o respuestas en gallego de la pregunta; o bien,
        - *Sgdo* es el valor semántico asociado a la lista de códigos ASCII *C* de la pregunta en inglés y *Sgdo* es el valor semántico asociado a la lista de códigos ASCII *B* de la respuesta o respuestas en inglés de la pregunta.
    - *D* es la expresión que corresponde a la lista de códigos ASCII *B* de la respuesta.

Lo que indican las tres opciones del predicado anterior es que el valor semántico de una pregunta en español es el mismo que el de esa pregunta en gallego y en inglés, con lo cual, las tres preguntas son sinónimas interlingüísticamente. Con las respuestas sucede lo mismo, su valor semántico

coincide, ya estén expresadas en español, gallego o inglés, lo que las hace sinónimas en este contexto.

Gramática para el español

El predicado `pregunta` es el que genera y reconoce las listas de códigos ASCII que corresponden a las preguntas en español, asociándole a cada una de éstas un valor semántico. Está formado por tres argumentos: el primero es el valor semántico de la expresión generada y los dos restantes son las listas necesarias para la generación de los códigos ASCII de las preguntas:

```
pregunta(Valor_semantico,Lista,Resto)
```

El primer tipo de pregunta es del estilo de "¿A qué hora llega a ___ un tren que sale de ___ a las ___?", donde los tres espacios libres instancian respectivamente el lugar de llegada, el lugar de salida y la hora de salida del predicado `viaje`. Esta pregunta comenzará por la lista de códigos ASCII de la expresión "¿A qué hora llega a " (-65 para "¿", 65 para "A", 32 para el espacio en blanco, y así sucesivamente. Véase el programa más adelante), le seguirá el nombre de un lugar y la lista de códigos ASCII de la expresión " un tren que sale de ", a continuación figurará otro nombre de lugar, la lista correspondiente a la expresión " a las ", una hora del día y la lista [63], que es el ASCII del signo "?". El valor semántico de esta pregunta es el predicado `viaje` donde el lugar de llegada ha sido instanciado con el valor semántico del primero de los nombres de lugar de la pregunta, el lugar de salida con el del segundo de los nombres de lugar y la hora de salida con el de la hora que figura en la pregunta. De modo

análogo se construirán las reglas para el resto de las preguntas: "¿A qué hora sale de ___ un tren que llega a ___ a las ___?", "¿Cuánto tarda en llegar a ___ un tren que sale de ___ a las ___?" y "¿Qué trenes van de ___ a ___?".

El predicado `lugar` genera las listas de códigos ASCII de los nombres de lugares donde el tren para y proporciona el valor semántico de cada uno de ellos. Consta de tres argumentos, el primero es el valor semántico y el segundo las dos listas que generan los códigos ASCII de los nombres de lugares:

```
lugar(Valor_semantico,Lista,Resto)
```

El predicado se define de la siguiente manera:

1. *A* es el valor semántico de una lista de códigos ASCII correspondiente a un lugar; lista que es el resultado de la diferencia entre una lista *C* y otra lista *Resto* si:
   - *A* es el nombre de una estación donde el tren para.
   - *B* es la lista de códigos ASCII que corresponde a *A*.
   - *C* es el resultado de concatenar la lista de códigos *B* con la lista *Resto*.

El predicado `hora` genera las listas de códigos ASCII de las horas del día y proporciona el valor semántico para cada una de ellas. Está formado por tres argumentos, el primero es el valor semántico y los dos siguientes las listas que generan los códigos ASCII de las horas:

```
hora(Valor_semantico,Lista,Resto)
```

Las horas del día serán dos números separados por un punto. El primer número puede tener una o dos cifras y el segundo siempre tiene dos cifras. Se

definirá el predicado para el caso en que el primer número tenga una cifra y para el caso en que el primer número tenga dos cifras:

1. *A* es el valor semántico de una lista de códigos ASCII correspondiente a una hora del día; lista que posee cuatro elementos [*B*, *C*, *D*, *E*] si:
   - [*B*, *C*, *D*, *E*] es la lista de códigos ASCII de *A*.
   - *B* es un número entre 48 y 57 (son los códigos ASCII de las cifras de 0 a 9).
   - *C* es 46 (es el código ASCII del punto).
   - *D* es un número entre 48 y 57.
   - *E* es un número entre 48 y 57.
2. El segundo caso es análogo al primero, la diferencia es que la lista es ahora [*B*, *C*, *D*, *E*, *F*], correspondiendo *D* al ASCII del punto.

Una vez más, se podrían establecer distintos tipos de respuestas según la información que se pretenda proporcionar. Como el objetivo es mostrar la comunicación en lenguaje natural entre una persona y una máquina, sólo se tendrá en cuenta un tipo de respuesta que sirva para cualquiera de las preguntas. Este tipo de respuesta contendrá toda la información que proporciona el predicado `viaje` y tiene la siguiente forma: "Un ___ sale de ___ a las ___ y llega a ___ a las ___ empleando ___ horas", donde los espacios libres se instanciarán respectivamente con los argumentos *tipo de tren*, *lugar de salida*, *hora de salida*, *lugar de llegada*, *hora de llegada* y *tiempo empleado* del predicado `viaje`.

El predicado `respuesta` es el que genera gramaticalmente las listas de códigos ASCII que corresponden a las respuestas en español, asociándole a cada una de éstas un valor semántico. Está formado por tres argumentos: el primero es el valor semántico de la expresión generada y los dos restantes son las listas necesarias para la generación de los códigos ASCII de las respuestas:

```
respuesta(Valor_semantico,Lista,Resto)
```

Una respuesta estará formada por la lista de códigos ASCII de "Un ", el nombre de un tren, la lista correspondiente a " sale de ", un nombre de lugar, la lista de " a las ", una hora del día, la lista de " y llega a ", otro nombre de lugar, la lista de " a las ", una hora del día, la lista de " empleando ", otra hora y la lista " horas". El valor semántico de una respuesta es el predicado `viaje` donde el tipo de tren ha sido instanciado con el valor semántico del nombre de tren de la respuesta, el lugar de salida con el del primero de los nombres de lugar de la respuesta, la hora de salida con el de la primera de las horas, el lugar de llegada con el del segundo de los nombres de lugar, la hora de llegada con el de la segunda de las horas y el tiempo que tarda el tren en realizar el trayecto con la tercera de las horas de la respuesta.

El predicado `nombretren` genera los códigos ASCII de los nombres de los tipos de tren para el español, así como asocia un valor semántico para cada uno de ellos. Consta de tres argumentos, el primero es el valor semántico y los dos siguientes las listas que generan los códigos ASCII de los nombres de tipos de tren:

```
nombretren(Valor_semantico,Lista,Resto)
```

Como hay tres tipos de tren, el predicado se define de la siguiente manera:

1. "Regional" es el valor semántico de la lista de códigos ASCII [82, 101, 103, 105, 111, 110, 97, 108] que corresponde a la expresión "Regional".
2. "TRD" es el valor semántico de la lista de códigos ASCII [84, 82, 68] que corresponde a la expresión "TRD".
3. "Costa de Galicia" es el valor semántico de la lista de códigos ASCII [67, 111, 115, 116, 97, 32, 100, 101, 32, 71, 97, 108, 105, 99, 105, 97] que corresponde a la expresión "Costa de Galicia".

Gramática para el gallego

El predicado `gpregunta` es el que genera y reconoce las listas de códigos ASCII que corresponden a las cuatro preguntas en gallego, asociándole a cada una de éstas el mismo valor semántico que en español. Está formado por tres argumentos: el primero es el valor semántico de la expresión generada y los dos restantes son las listas necesarias para la generación de los códigos ASCII de las preguntas.

```
gpregunta(Valor_semantico,Lista,Resto)
```

El procedimiento es análogo al presentado para el predicado `pregunta` de la gramática para el español, con la diferencia que ahora las preguntas generadas son las traducciones correspondientes a aquellas: "¿A que hora chega a ___ un tren que sae de ___ ás ___?", "¿A que hora sae de ___ un tren que chega a ___ ás

___?", "¿Canto tarda en chegar a ___ un tren que sae de ___ ás ___?" y "¿Que trens van de ___ a ___?".

El tipo de respuestas para el gallego, generadas con el predicado `grespuesta`, también corresponde a la traducción del tipo de respuesta de la gramática para el español: "Un ___ sae de ___ ás ___ e chega a ___ ás ___ empregando ___ horas", y su valor semántico es el mismo: El predicado se define de modo análogo al predicado `respuesta` de la gramática para el español:

```
grespuesta(Valor_semantico,Lista,Resto)
```

Gramática para el inglés

El predicado `ipregunta` es el que genera y reconoce las listas de códigos ASCII que corresponden a las cuatro preguntas en inglés, asociándole a cada una de éstas el mismo valor semántico que en español y en gallego. Está formado por tres argumentos: el primero es el valor semántico de la expresión generada y los dos restantes son las listas necesarias para la generación de los códigos ASCII de las preguntas.

```
ipregunta(Valor_semantico,Lista,Resto)
```

El procedimiento también es análogo al presentado en las gramáticas para los otros dos idiomas. Los cuatro tipos de pregunta en inglés son: "What is the arrival time in ___ of the train leaving from ___ at ___?", "What is the leaving time from ___ of the train arriving in ___ at ___?", "How long does the train leaving from ___ at ___ take to arrive in ___?", "What are the trains going from ___ to ___?".

El tipo de respuestas para el inglés son generadas por el predicado `irespuesta`, tienen la forma "A ___ leaves from ___ at ___ and arrives to ___ at ___ taking ___ hours" y su definición es análoga a la de los otros dos idiomas:

`irespuesta(Valor_semantico,Lista,Resto)`

El código Prolog de todo el interfaz es el siguiente:

```
%INTERFAZ PARA EL PROGRAMA DEL HORARIO DE TRENES

horario:-
      write('Pregunta'),nl,
      write('¿A qué hora llega a ___ un tren que sale de ___ a
            las ___?'),nl,
      write('¿A qué hora sale de ___ un tren que llega a ___ a
            las ___?'),nl,
      write('¿Cuánto tarda en llegar a ___ un tren que sale de
            ___ a las ___?'),nl,
      write('¿Qué trenes van de ___ a ___?'),nl,nl,
      write('Pregunta'),nl,
      write('¿A que hora chega a ___ un tren que sae de ___ ás
            ___?'),nl,
      write('¿A que hora sae de ___ un tren que chega a ___ ás
            ___?'),nl,
      write('¿Canto tarda en chegar a ___ un tren que sae de ___
            ás ___?'),nl,
      write('¿Que trens van de ___ a ___?'),nl,nl,
      write('Question'),nl,
      write('What is the arrival time in ___ of the train leaving
            from ___ at ___?'),nl,
      write('What is the leaving time from ___ of the train
            arriving in ___ at ___?'),nl,
      write('How long does the train leaving from ___ at ___ take
            to arrive in ___?'),nl,
      write('What are the trains going from ___ to ___?'),nl,nl,
      read(A),nl,
      findall(B,pregunta_respuesta(A,B),C),
      escribir(C),
      mas_preg.

escribir([]).
escribir([A|B]):-
      write(A),
      nl,
      nl,
      escribir(B).

mas_preg:-
      nl,nl,
      write('¿Más preguntas? / ¿Máis preguntas? / More
```

```
            questions?'),
      read(Q),
      (Q='s';Q='S';Q='si';Q='Si';Q='y';Q='Y';Q='yes';Q='Yes'),
      horario.

pregunta_respuesta(A,D):-
      name(A,C),
      ((pregunta(Sgdo,C,[]),respuesta(Sgdo,B,[]));
      (gpregunta(Sgdo,C,[]),grespuesta(Sgdo,B,[]));
      (ipregunta(Sgdo,C,[]),irespuesta(Sgdo,B,[]))),
      name(D,B).

pregunta(viaje(_,B,C,D,_,_),[-65,65,32,113,117,-23,32,104,111,
114,97,32,108,108,101,103,97,32,97,32|Lista1],Resto):-
      lugar(D,Lista1,[32,117,110,32,116,114,101,110,32,113,117,
            101,32,115,97,108,101,32,100,101,32|Lista2]),
      lugar(B,Lista2,[32,97,32,108,97,115,32|Lista3]),
      hora(C,Lista3,[63|Resto]).
pregunta(viaje(_,B,_,D,E,_),[-65,65,32,113,117,-23,32,104,111,
114,97,32,115,97,108,101,32,100,101,32|Lista1],Resto):-
      lugar(B,Lista1,[32,117,110,32,116,114,101,110,32,113,117,
            101,32,108,108,101,103,97,32,97,32|Lista2]),
      lugar(D,Lista2,[32,97,32,108,97,115,32|Lista3]),
      hora(E,Lista3,[63|Resto]).
pregunta(viaje(_,B,C,D,_,_),[-65,67,117,-31,110,116,111,32,116,
97,114,100,97,32,101,110,32,108,108,101,103,97,114,32,97,32|
Lista1],Resto):-
      lugar(D,Lista1,[32,117,110,32,116,114,101,110,32,113,117,
            101,32,115,97,108,101,32,100,101,32|Lista2]),
      lugar(B,Lista2,[32,97,32,108,97,115,32|Lista3]),
      hora(C,Lista3,[63|Resto]).
pregunta(viaje(_,B,_,D,_,_),[-65,81,117,-23,32,116,114,101,110,
101,115,32,118,97,110,32,100,101,32|Lista1],Resto):-
      lugar(B,Lista1,[32,97,32|Lista2]),
      lugar(D,Lista2,[63|Resto]).

lugar(A,C,Resto):-
      estacion(A,_),
      name(A,B),
      concatenar(B,Resto,C).

hora(A,[B,C,D,E|Resto],Resto):-
      name(A,[B,C,D,E]),
      (B=48;B=49;B=50;B=51;B=52;B=53;B=54;B=55;B=56;B=57),
      C=46,
      (D=48;D=49;D=50;D=51;D=52;D=53;D=54;D=55;D=56;D=57),
      (E=48;E=49;E=50;E=51;E=52;E=53;E=54;E=55;E=56;E=57).
hora(A,[B,C,D,E,F|Resto],Resto):-
      name(A,[B,C,D,E,F]),
      (B=48;B=49;B=50;B=51;B=52;B=53;B=54;B=55;B=56;B=57),
      (C=48;C=49;C=50;C=51;C=52;C=53;C=54;C=55;C=56;C=57),
      D=46,
      (E=48;E=49;E=50;E=51;E=52;E=53;E=54;E=55;E=56;E=57),
```

```
      (F=48;F=49;F=50;F=51;F=52;F=53;F=54;F=55;F=56;F=57).

respuesta(viaje(A,B,C,D,E,F),[85,110,32|Lista1],Resto):-
      viaje(A,B,C,D,E,F),
      nombretren(A,Lista1,[32,115,97,108,101,32,100,101,32|
            Lista2]),
      lugar(B,Lista2,[32,97,32,108,97,115,32|Lista3]),
      hora(C,Lista3,[32,121,32,108,108,101,103,97,32,97,32|
            Lista4]),
      lugar(D,Lista4,[32,97,32,108,97,115,32|Lista5]),
      hora(E,Lista5,[32,101,109,112,108,101,97,110,100,111,32|
            Lista6]),
      hora(F,Lista6,[32,104,111,114,97,115|Resto]).

nombretren('Regional',[82,101,103,105,111,110,97,108|Resto],
      Resto).
nombretren('TRD',[84,82,68|Resto],Resto).
nombretren('Costa de Galicia',[67,111,115,116,97,32,100,101,32,
      71,97,108,105,99,105,97|Resto],Resto).

gpregunta(viaje(_,B,C,D,_,_),[-65,65,32,113,117,101,32,104,111,
114,97,32,99,104,101,103,97,32,97,32|Lista1],Resto):-
      lugar(D,Lista1,[32,117,110,32,116,114,101,110,32,113,117,
            101,32,115,97,101,32,100,101,32|Lista2]),
      lugar(B,Lista2,[32,-31,115,32|Lista3]),
      hora(C,Lista3,[63|Resto]).
gpregunta(viaje(_,B,_,D,E,_),[-65,65,32,113,117,101,32,104,111,
114,97,32,115,97,101,32,100,101,32|Lista1],Resto):-
      lugar(B,Lista1,[32,117,110,32,116,114,101,110,32,113,117,
            101,32,99,104,101,103,97,32,97,32|Lista2]),
      lugar(D,Lista2,[32,-31,115,32|Lista3]),
      hora(E,Lista3,[63|Resto]).
gpregunta(viaje(_,B,C,D,_,_),[-65,67,97,110,116,111,32,116,97,
114,100,97,32,101,110,32,99,104,101,103,97,114,32,97,32|Lista1],
Resto):-
      lugar(D,Lista1,[32,117,110,32,116,114,101,110,32,113,117,
            101,32,115,97,101,32,100,101,32|Lista2]),
      lugar(B,Lista2,[32,-31,115,32|Lista3]),
      hora(C,Lista3,[63|Resto]).
gpregunta(viaje(_,B,_,D,_,_),[-65,81,117,101,32,116,114,101,110,
115,32,118,97,110,32,100,101,32|Lista1],Resto):-
      lugar(B,Lista1,[32,97,32|Lista2]),
      lugar(D,Lista2,[63|Resto]).

grespuesta(viaje(A,B,C,D,E,F),[85,110,32|Lista1],Resto):-
      viaje(A,B,C,D,E,F),
      nombretren(A,Lista1,[32,115,97,101,32,100,101,32|Lista2]),
      lugar(B,Lista2,[32,-31,115,32|Lista3]),
      hora(C,Lista3,[32,101,32,99,104,101,103,97,32,97,32|
            Lista4]),
      lugar(D,Lista4,[32,-31,115,32|Lista5]),
      hora(E,Lista5,[32,101,109,112,114,101,103,97,110,100,111,
            32|Lista6]),
```

```
      hora(F,Lista6,[32,104,111,114,97,115|Resto]).

ipregunta(viaje(_,B,C,D,_,_),[87,104,97,116,32,105,115,32,116,
104,101,32,97,114,114,105,118,97,108,32,116,105,109,101,32,105,
110,32|Lista1],Resto):-
      lugar(D,Lista1,[32,111,102,32,116,104,101,32,116,114,97,
            105,110,32,108,101,97,118,105,110,103,32,102,114,111,
            109,32|Lista2]),
      lugar(B,Lista2,[32,97,116,32|Lista3]),
      hora(C,Lista3,[63|Resto]).
ipregunta(viaje(_,B,_,D,E,_),[87,104,97,116,32,105,115,32,116,
104,101,32,108,101,97,118,105,110,103,32,116,105,109,101,32,102,
114,111,109,32|Lista1],Resto):-
      lugar(B,Lista1,[32,111,102,32,116,104,101,32,116,114,97,
            105,110,32,97,114,114,105,118,105,110,103,32,105,110,
            32|Lista2]),
      lugar(D,Lista2,[32,97,116,32|Lista3]),
      hora(E,Lista3,[63|Resto]).
ipregunta(viaje(_,B,C,D,_,_),[72,111,119,32,108,111,110,103,32,
100,111,101,115,32,116,104,101,32,116,114,97,105,110,32,108,101,
97,118,105,110,103,32,102,114,111,109,32|Lista1],Resto):-
      lugar(B,Lista1,[32,97,116,32|Lista2]),
      hora(C,Lista2,[32,116,97,107,101,32,116,111,32,97,114,114,
            105,118,101,32,105,110,32|Lista3]),
      lugar(D,Lista3,[63|Resto]).
ipregunta(viaje(_,B,_,D,_,_),[87,104,97,116,32,97,114,101,32,
116,104,101,32,116,114,97,105,110,115,32,103,111,105,110,103,32,
102,114,111,109,32|Lista1],Resto):-
      lugar(B,Lista1,[32,116,111,32|Lista2]),
      lugar(D,Lista2,[63|Resto]).

irespuesta(viaje(A,B,C,D,E,F),[65,32|Lista1],Resto):-
      viaje(A,B,C,D,E,F),
      nombretren(A,Lista1,[32,108,101,97,118,101,115,32,102,114,
            111,109,32|Lista2]),
      lugar(B,Lista2,[32,97,116,32|Lista3]),
      hora(C,Lista3,[32,97,110,100,32,97,114,114,105,118,101,115,
            32,116,111,32|Lista4]),
      lugar(D,Lista4,[32,97,116,32|Lista5]),
      hora(E,Lista5,[32,116,97,107,105,110,103,32|Lista6]),
      hora(F,Lista6,[32,104,111,117,114,115|Resto]).
```

(Los predicados `write`, `nl`, `read`, `findall`, `name` y `concatenar` utilizados en este programa, han sido presentados en el primer capítulo de este trabajo).

Añadiendo este código al presentado en el apartado 1.2.1, el diálogo comenzará al teclear:

```
horario.
```

Al pulsar la tecla *Intro* aparecerá la siguiente lista de tipos de preguntas posibles que se pueden realizar:

```
Pregunta
¿A qué hora llega a ___ un tren que sale de ___ a las ___?
¿A qué hora sale de ___ un tren que llega a ___ a las ___?
¿Cuánto tarda en llegar a ___ un tren que sale de ___ a las ___?
¿Qué trenes van de ___ a ___?

Pregunta
¿A que hora chega a ___ un tren que sae de ___ ás ___?
¿A que hora sae de ___ un tren que chega a ___ ás ___?
¿Canto tarda en chegar a ___ un tren que sae de ___ ás ___?
¿Que trens van de ___ a ___?

Question
What is the arrival time in ___ of the train leaving from ___ at
___?
What is the leaving time from ___ of the train arriving in ___
at ___?
How long does the train leaving from ___ at ___ take to arrive
in ___?
What are the trains going from ___ to ___?

|:
```

Si ahora se escribe:

```
'¿A qué hora llega a Santiago un tren que sale de Vigo a las
11.40?'.
```

El programa dara como respuesta:

```
Un Costa de Galicia sale de Vigo a las 11.40 y llega a Santiago
a las 13.16 empleando 1.36 horas

¿Más preguntas? / ¿Máis preguntas? / More questions?|:
```

Respondiendo afirmativamente a esta última cuestión, el sistema desplegará de nuevo el listado de preguntas posibles para que realicemos otra consulta. Si se pregunta en gallego el resultado es el siguiente:

```
'¿A que hora sae de Santiago un tren que chega a Vilagarcía de
Arousa ás 7.42?'.

Un TRD sae de Santiago ás 7.12 e chega a Vilagarcía de Arousa ás
7.42 empregando 0.30 horas
```

Análogamente para el inglés:

```
'What are the trains going from A Coruña to Vigo?'.

A TRD leaves from A Coruña at 6.20 and arrives to Vigo at 8.38
taking 2.18 hours

A Regional leaves from A Coruña at 6.30 and arrives to Vigo at
9.52 taking 3.22 hours

A Costa de Galicia leaves from A Coruña at 7.30 and arrives to
Vigo at 10.23 taking 2.53 hours

...
```

Son diversas las preguntas que se pueden realizar sobre el horario de trenes, incluso puede haber distintas formas de plantear una misma pregunta dentro del mismo idioma. Por ejemplo, en lugar de "¿Qué trenes van de Santiago a Vigo?" podría decirse también "¿Qué trenes salen de Santiago para Vigo?", "¿Qué trenes parten de Santiago hacia Vigo?", "¿Qué trenes viajan desde Santiago hasta Vigo?", etc. A cualquiera de ellas se le puede asociar el valor semántico "`viaje(X,'Santiago',Y,'Vigo',Z,W).`", lo cual indica que podrían considerarse sinónimas. Estas otras opciones de pregunta son resultado de sustituir algunas palabras por otras que son sinónimas de ellas en este contexto, por ejemplo "van", "salen", "parten" y "viajan" o "de" y "desde" etc. Otros casos más complejos son los que se dan cuando la pregunta sinónima tiene una estructura distinta a la pregunta original como por ejemplo "De Santiago a Vigo, ¿qué trenes hay?". Si el programa fuese capaz de tratar ambos casos de sinonimia, el diálogo usuario-máquina sería más flexible y por lo tanto más cómodo. Esto pone de manifiesto la importancia del tratamiento computacional de la sinonimia en el ámbito de la traducción automática.

El problema de la sustitución de unas palabras por otras dentro de una oración será abordado en la sección siguiente, pero para ello es necesario tratar el problema de la sinonimia entre palabras, el cual podrá ser generalizado a un diccionario de sinónimos. Se verá que el tratamiento automático de la sustitución no es una tarea fácilmente generalizable y mucho menos el problema de la sinonimia entre oraciones con distinta estructura como las oraciones en pasiva; sin embargo, en el contexto del horario de trenes podrían obtenerse buenos resultados con una realización no demasiado costosa.

2.- Sinonimia intralingüística en los lenguajes naturales

A continuación se proporcionarán distintos intentos de automatización de la sinonimia con Prolog para la lengua española. El tratamiento se realizará exclusivamente respecto a palabras y oraciones. Para la sinonimia entre palabras, desarrollada en el apartado 2.1, se calculará el grado de sinonimia de las entradas de un diccionario de sinónimos mediante el uso de medidas de similaridad. La importancia de esta propuesta es que, a diferencia de los demás ejemplos de este trabajo, no se restringe a expresiones concretas sino que será generalizada a todo el diccionario. Ello posibilitará la elaboración de un diccionario electrónico de sinónimos que será abordada en el próximo capítulo.

Con todo, y a pesar de las dificultades de su generalización, en el apartado 2.2 se tratarán tres casos de sinonimia oracional. El primero de ellos es el de la *sustituibilidad*, que partiendo de lo visto en el apartado 2.1, calculará el grado de

sinonimia de dos oraciones cuando la segunda se obtiene sustituyendo en la primera algunas palabras por sus sinónimas. La sustitución plantea problemas debido al carácter polisémico de las palabras, ya que al ser utilizadas en una oración, lo usual es que éstas se empleen en una acepción concreta. En el apartado se esbozan algunas propuestas de cómo detectar esta acepción empleada. Los otros dos casos, el de las oraciones en pasiva y el de las que contienen términos inversos, son de naturaleza precisa y pueden ser tratados recurriendo una vez más al procesamiento de gramáticas y a la gestión del significado con Prolog.

2.1.- Sinonimia intralingüística entre palabras

Uno de los problemas de definir la sinonimia en términos de significado es que no se ha proporcionado una caracterización satisfactoria de éste. Aunque no es el propósito de este trabajo ofrecer una definición de sinonimia, sino describir su comportamiento en el marco restringido de un diccionario de sinónimos, se podría indicar que el criterio previo que subyace a la propuesta que se establecerá a continuación es la que dice que dos expresiones son sinónimas si figuran como tales en un diccionario de sinónimos.

Otra manera de ver el problema, que es igualmente válida para los propósitos de este trabajo es la que concibe a la sinonimia como similaridad de significado entre dos expresiones y representa a éste como el conjunto de sinónimos que ambas poseen en un diccionario de sinónimos. Nótese que se

habla del conjunto de sinónimos como una forma de representar el significado y no como una forma de definirlo. Esta representación tiene la ventaja añadida de su adecuación para el tratamiento computacional. Caracterizando la sinonimia de esta manera, se puede medir el grado de similaridad de dos conjuntos de sinónimos correspondientes a dos entradas mediante el uso de una medida de similaridad. Esto proporciona un grado de similaridad entre ambas representaciones de los significados que podría ser considerado como el grado de sinonimia entre las expresiones correspondientes. En el procesamiento automático de información, son habituales las siguientes medidas de similaridad (véase Liao, T. W., Zhang Z. M., Mount, C. R. [1998]), que aquí se han adaptado para el caso de la sinonimia. Sean dos palabras *a* y *b* y sus respectivas listas de sinónimos *A* y *B*:

1. *Coeficiente jaccard* (*jaccard coefficient*): Calcula el coeficiente *C* de dos palabras como:

$$C = \frac{\text{Número de sinónimos que aparecen en } A \text{ y en } B}{\text{Número de sinónimos que aparecen en } A \text{ o en } B}.$$

Dicho de otra forma, esta medida establece el grado de sinonimia dividiendo la cardinalidad de la intersección de ambas listas de sinónimos partido por la cardinalidad de la unión de las mismas. Utilizando la notación usual de la teoría de conjuntos, el grado de sinonimia *GRS(a, b)* es:

$$GRS(a,b) = \frac{|A \cap B|}{|A \cup B|}.$$

2. *Coeficiente del dado* (*dice coefficient*):

$$C = \frac{\text{Número de sinónimos que aparecen en } A \text{ y en } B}{\text{Media aritmética entre el número de elementos de } A \text{ y el de } B}.$$

Es decir:

$$GRS(a,b) = \frac{|A \cap B|}{\frac{|A| + |B|}{2}} = \frac{2 \cdot |A \cap B|}{|A| + |B|}.$$

3. *Coeficiente del coseno* (*cosine coefficient*):

$$C = \frac{\text{Número de sinónimos que aparecen en } A \text{ y en } B}{\text{Media geométrica entre el número de elementos de } A \text{ y el de } B}.$$

Es decir:

$$GRS(a,b) = \frac{|A \cap B|}{\sqrt{|A|} \cdot \sqrt{|B|}}.$$

4. *Coeficiente de similaridad mutua* (*mutual similarity coefficient*):

$$C = \frac{\text{Número de sinónimos que aparecen en } A \text{ y en } B}{\text{Media armónica entre el número de elementos de } A \text{ y el de } B}.$$

Es decir:

$$GRS(a,b) = \frac{|A \cap B|}{\frac{2}{\frac{1}{|A|} + \frac{1}{|B|}}} = \frac{\frac{|A \cap B|}{|A|} + \frac{|A \cap B|}{|B|}}{2}.$$

5. *Coeficiente de solapamiento* (*overlap coefficient*):

$$C = \frac{\text{Número de sinónimos que aparecen en } A \text{ y en } B}{\text{Menor del número de elementos de } A \text{ y de } B}.$$

Es decir:

$$GRS(a,b) = \frac{|A \cap B|}{min(|A|, |B|)}.$$

Cada una de estas medidas de similaridad proporciona un grado de sinonimia más alto o más bajo en el intervalo [0, 1] respecto al mismo par de expresiones. El orden es el siguiente:

Jaccard ≤ *dado* ≤ *coseno* ≤ *similaridad mutua* ≤ *solapamiento*,

es decir:

$$\frac{|A \cap B|}{|A \cup B|} \leq \frac{|A \cap B|}{\frac{|A|+|B|}{2}} \leq \frac{|A \cap B|}{\sqrt{|A|} \cdot \sqrt{|B|}} \leq \frac{|A \cap B|}{\frac{2}{\frac{1}{|A|}+\frac{1}{|B|}}} \leq \frac{|A \cap B|}{min(|A|, |B|)}.$$

Para probar que son ciertas las desigualdades anteriores, como el numerador de los cocientes de cada medida de similaridad es el mismo (|A ∩ B|), basta probar que:

Unión ≥ *media aritmética* ≥ *media geométrica* ≥ *media armónica* ≥ *mínimo*,

es decir

$$|A \cup B| \geq \frac{|A|+|B|}{2} \geq \sqrt{|A|} \cdot \sqrt{|B|} \geq \frac{2}{\frac{1}{|A|}+\frac{1}{|B|}} \geq min(|A|, |B|).$$

Teniendo en cuenta que A y B son conjuntos finitos no vacíos y por consiguiente $|A| \geq 1$ y $|B| \geq 1$, no es dificil mostrar que se cumple que *unión* ≥ *media aritmética,* puesto que la cardinalidad de la unión de dos conjuntos siempre será mayor o igual a la cardinalidad del mayor de ellos y la media aritmética siempre estará entre la cardinalidad del mayor y la del menor. Las desigualdades *media aritmética* ≥ *media geométrica* ≥ *media armónica* son un resultado clásico en estadística. Finalmente, *media armónica* ≥ *mínimo* también

se cumple por el hecho de que la media armónica entre dos cardinalidades será un número entre la menor de ellas y la mayor, que siempre será mayor o igual al mínimo de ambas.

De todas formas, sea cual sea el valor proporcionado por cada una de estas medidas, lo cierto es que todas ellas proporcionan un grado de similaridad de dos conjuntos. Considerando a éstos como los conjuntos que proporciona un diccionario de sinónimos para cada acepción de cada una de las entradas, la tesis de este trabajo es concebir este grado como el grado de sinonimia de dos palabras. El cálculo del mismo puede ser realizado de forma automática por un diccionario electrónico de sinónimos como el que se presentará en el próximo capítulo. El criterio empleado será que dos entradas figurarán como sinónimas en él si su grado de sinonimia es distinto de 0. Tal y como están definidas las medidas de similaridad, esto sucede cuando la intersección de los conjuntos de sinónimos no es vacía ($|A \cap B| \neq \varnothing$), o lo que es lo mismo, cuando poseen al menos un sinónimo en común. De esta forma, la relación de sinonimia en el diccionario electrónico será:

1. Reflexiva: porque el conjunto de sinónimos de una palabra siempre tiene elementos en común (todos) con el conjunto de sinónimos de ella misma.
2. Simétrica: porque si el conjunto de sinónimos de una palabra *A* tiene elementos en común con el de otra *B* entonces el de *B* tiene elementos en común (los mismos) con el de *A*.

3. Pero no es transitiva: porque el conjunto de sinónimos de una palabra *A* puede tener elementos en común con el de otra *B* y el de *B* con el de otra *C* pero el de *A* no tiene porque tener elementos en común con el de *C*. (En el proximo capítulo figuran ejemplos concretos de este hecho respecto al diccionario empleado).

Todo esto lleva a considerar la sinonimia como una relación distinta de la equivalencia, al contrario de cómo se había afirmado históricamente en muchos contextos teóricos. El carácter práctico de este trabajo hace que la similaridad sea un modelo más apropiado para la sinonimia que la equivalencia, ya que, como se verá en el próximo capítulo, la sinonimia, o si se quiere, la relación entre expresiones de la que da cuenta un diccionario de sinónimos (algunos afirman que esta relación no es sinonimia propiamente dicha), es una relación más flexible que la equivalencia. Además, el objetivo de la obtención de este grado no es tanto poner en conocimiento del mismo a un usuario potencial del diccionario electrónico sino aprovechar las ventajas que dicho cálculo proporciona al diccionario electrónico con respecto al diccionario impreso. Estas ventajas son la detección automática de las acepciones de las palabras que proporciona como respuesta, la reducción del número de sinónimos mediante el establecimiento de umbrales de sinonimia, el ordenamiento de éstos según el grado de sinonimia respecto a la entrada y la ampliación de las respuestas con aquellas expresiones que, aunque no figuran explícitamente como sinónimas en el diccionario

impreso, sí podrían serlo por el hecho de no poseer un grado de sinonimia distinto de 0 con respecto a la entrada.

El tratamiento automático que se realizará a continuación es lo suficientemente versátil como para poder incorporar nuevas medidas de similaridad, eliminar algunas de las existentes por no responder de modo correcto al comportamiento de la sinonimia o realizar operaciones con ellas.

Antes de comenzar el análisis, es necesario indicar brevemente algunos aspectos que serán tenidos en cuenta en el resto del apartado. El primero de ellos es el carácter polisémico de las palabras. Debido al hecho de que las entradas pueden tener diversos significados, algunos diccionarios de sinónimos distinguen varias acepciones para cada una de ellas asociando un grupo de sinónimos distinto a cada acepción. De esta forma, se puede comparar cada palabra teniendo en cuenta el conjunto genérico de sus sinónimos o bien solamente el conjunto correspondiente a acepciones concretas, obteniendo así un análisis más contextualizado. Esta segunda propuesta parece más coherente, ya que evitará la interferencia de las otras acepciones en la obtención del grado de sinonimia de dos palabras cuando se utilicen en un contexto concreto.

Otra de las características de los diccionarios de sinónimos es que no incluyen el caso trivial de que, fijada una acepción, una palabra sea sinónima de sí misma; algo que resulta razonable, puesto que el empleo de un diccionario de sinónimos se basa en la búsqueda de palabras distintas a la entrada. Sin embargo, este hecho puede llevarnos a situaciones como ésta: en el diccionario que será

utilizado en este trabajo (Blecua J. M. [1997]), figura solamente la palabra "pernil" como único sinónimo de "pernera" en la única acepción de ésta y la palabra "pernera" como único sinónimo de la segunda acepción de "pernil". Si comparamos ambas palabras según esas acepciones, la intersección de ambos conjuntos de sinónimos es vacía, con lo que el grado de sinonimia medido con cualquiera de los coeficientes anteriores sería 0, es decir, no serían sinónimas, cosa que no resulta coherente. La forma de evitar casos como éste es incluir el caso trivial en el conjunto de los sinónimos de una palabra. En el ejemplo aludido, el conjunto de sinónimos de la única acepción de "pernera" sería {"pernera", "pernil"} y el de la segunda acepción de "pernil" {"pernil", "pernera"}, de esta forma el resultado sería que son sinónimas en grado 1, algo que parece más adecuado.

2.1.1.- La base de datos: El diccionario de sinónimos

Se considerará a un diccionario de sinónimos como una base de datos de palabras (entradas) y listas de palabras (sinónimos) agrupadas por acepciones. Para ello será utilizado el predicado:

```
sin_dic(Palabra, Lista_sinónimos, Acepción)
```

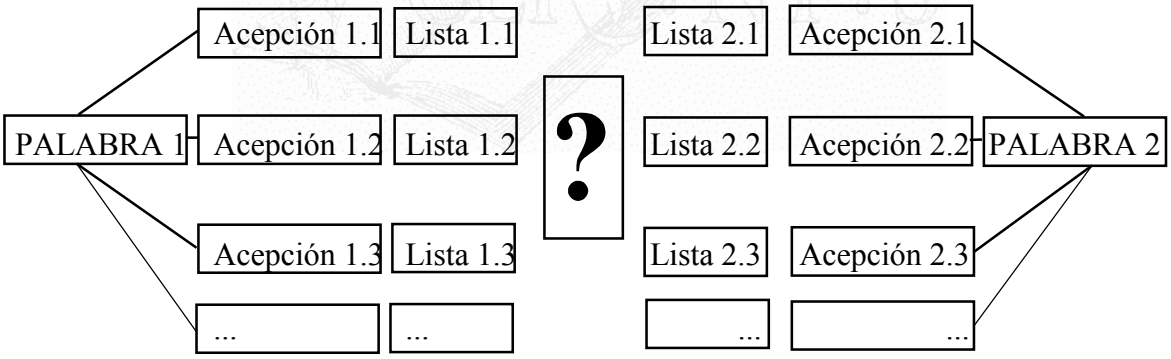
cuyo primer argumento es la entrada del diccionario, el segundo la lista de sus sinónimos y el tercero la acepción correspondiente a esa lista.

La información para elaborar esta base de datos será extraída del *Diccionario avanzado de sinónimos y antónimos de la lengua española* (Blecua

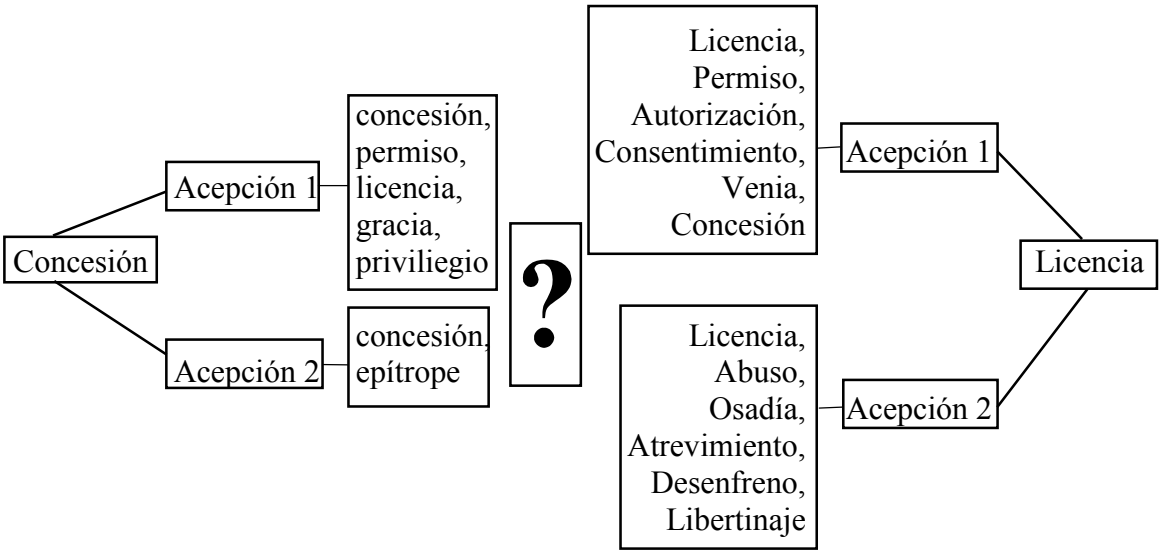
J. M. [1997]). En el primer apartado del próximo capítulo figura una descripción más detallada del comportamiento de la sinonimia en el mismo.

2.1.2.- La base de reglas: El programa

Si nos preguntamos ahora cuál será el grado de sinonimia de dos palabras, nos encontraremos con una situación que podríamos esquematizar de la siguiente forma:

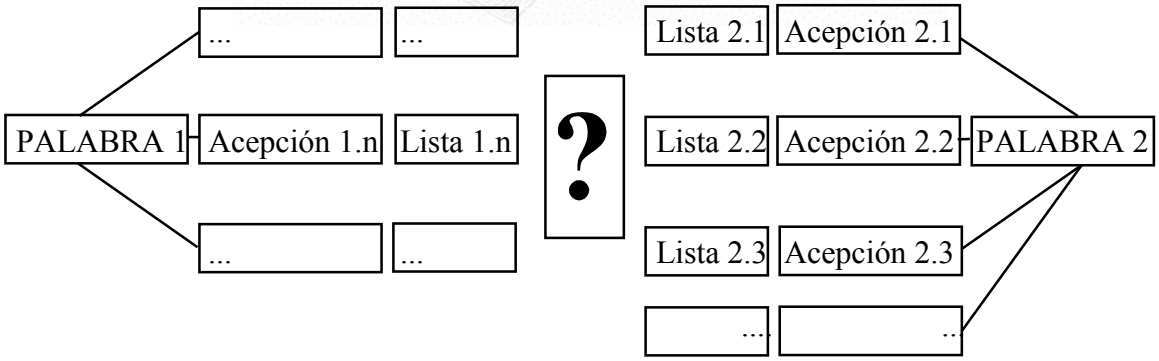


Para un ejemplo concreto:

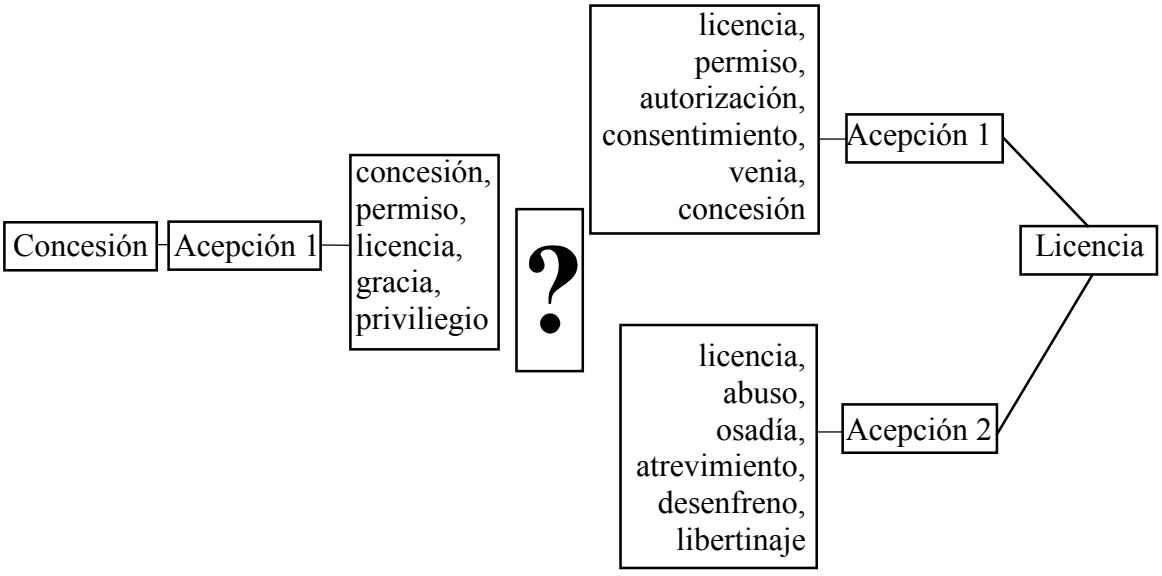


Por tanto, la cuestión es averiguar qué listas correspondientes a la primera y segunda palabras deben utilizarse para calcular el grado. Los pasos a seguir son los siguientes:

Paso 1. *Comprobar que la segunda palabra pertenece a alguna de las listas de la primera*. Según esto, si dos palabras no figuran como sinónimas en el diccionario, el compilador no hará ningún tipo de cálculo y responderá que no son sinónimas. Por tanto, dos palabras serán consideradas sinónimas si figuran como tales en un diccionario de sinónimos, que es algo que da cuenta de la caracterización previa que se ha proporcionado de la sinonimia. Para realizar este paso se definirá más adelante el procedimiento `comparable`. Una vez realizado este paso, queda fijada la acepción correspondiente a la primera palabra:



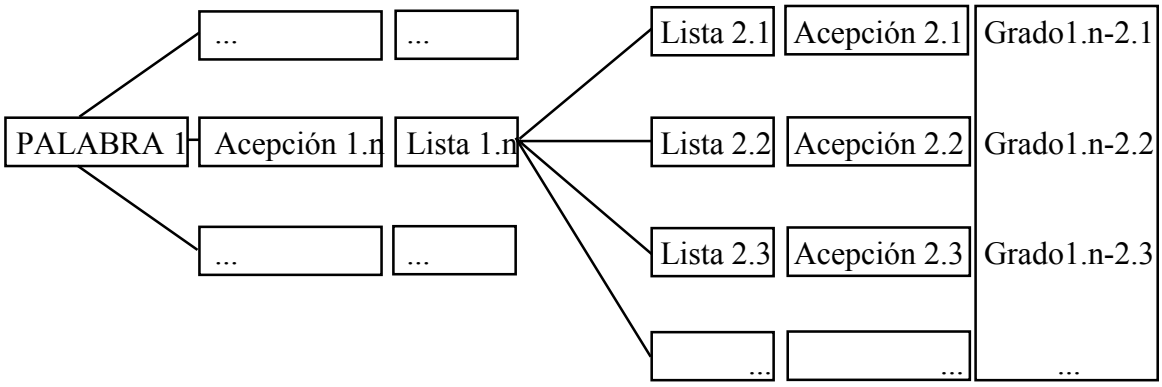
Para el ejemplo:



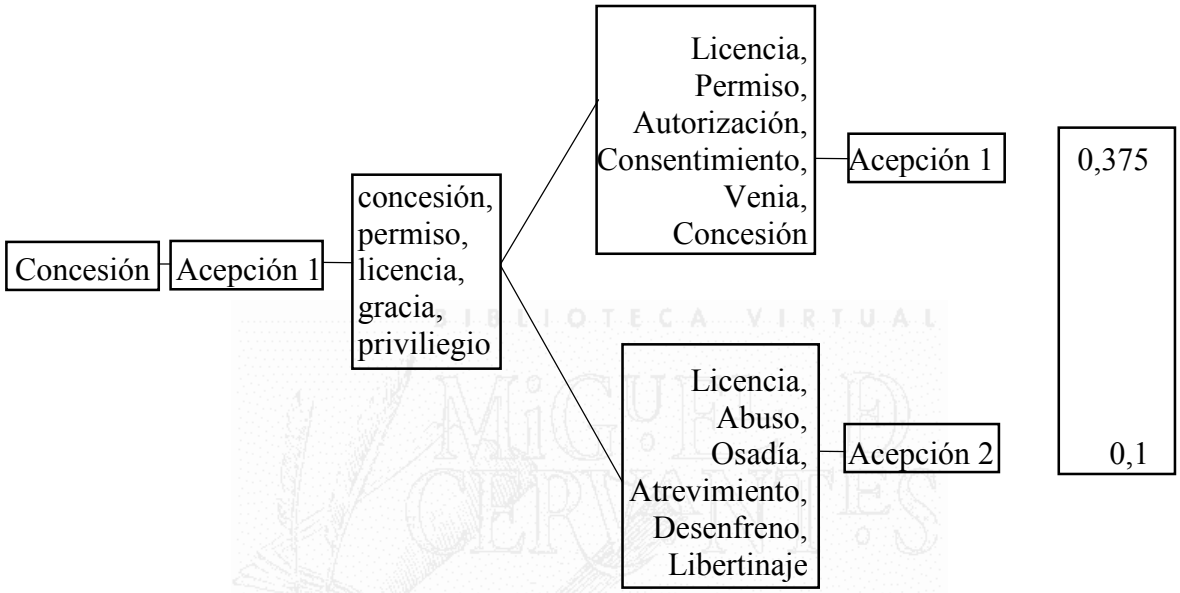
En el ejemplo anterior sólo es fijada una de las acepciones de la primera palabra. Sin embargo, puede pasar que se detecte más de una, en tales casos, el procedimiento que sigue a continuación se repetirá tantas veces como acepciones hayan sido obtenidas.

Paso 2. *Detectar cuál es la acepción de la segunda palabra que debemos utilizar*. Este segundo paso se divide en otros dos:

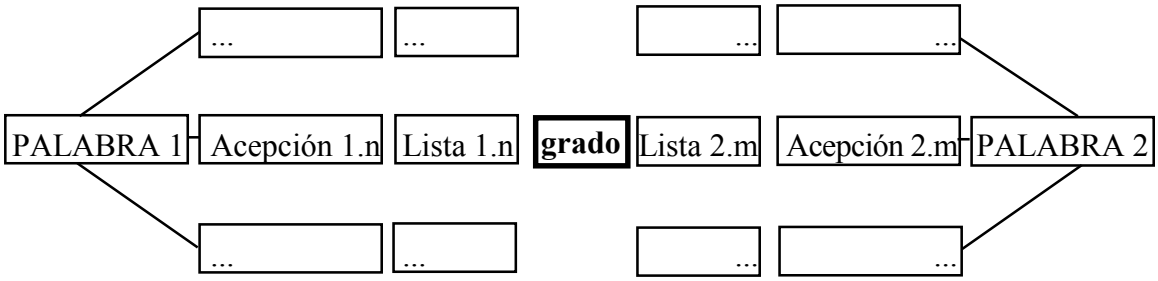
2.1. *Utilizar una medida de similaridad para calcular el grado de sinonimia de la acepción obtenida en el paso 1 con todas las acepciones de la segunda palabra*. Para la realización de este paso se utilizará el predicado `sin_acep` que será definido más adelante. Mediante el predicado `findall` se obtendrá la lista de todos los grados calculados y la lista de las acepciones de la segunda palabra correspondientes a esos grados:



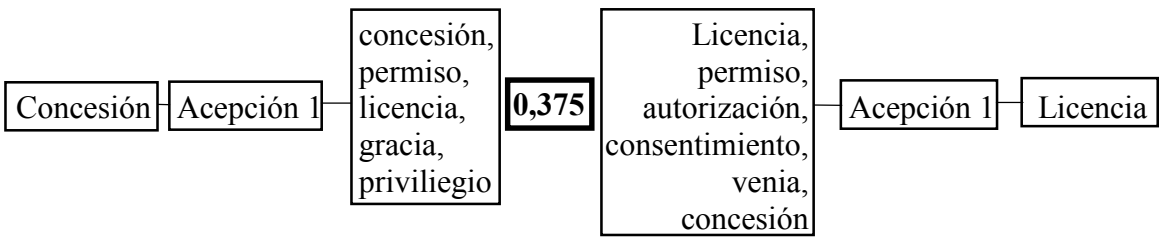
Para el ejemplo:



2.2. *Hallar el máximo de los grados calculados en 2.1*. La acepción correspondiente al máximo es la acepción buscada y el grado máximo es el grado de sinonimia entre las dos palabras. El predicado `max` realizará este cálculo, permitiendo detectar la acepción correspondiente a la segunda palabra y el grado de sinonimia entre ambas palabras:



Para el ejemplo:



El predicado `sinonimo` calcula el grado de sinonimia entre dos palabras. Sus argumentos son una palabra *A*, una acepción *AcepA* de ésta, una palabra *B*, su acepción *AcepB*, el grado de sinonimia *GRS* entre *A* y *B* para las acepciones indicadas, un umbral de sinonimia y una medida de similaridad:

`sinonimo(A, AcepA, B, AcepB, GRS, Umbral, Medida_de_Similaridad)`

El umbral es un número real perteneciente al intervalo [0, 1] que indicará al programa que no se consideren sinónimas las palabras *A* y *B* si su grado de sinonimia es menor que él.

El predicado se define de la siguiente forma para cada una de las medidas de similaridad:

1. Cualquier palabra es sinónima de sí misma en grado 1 si pertenece al diccionario y su acepción no varía; sea cual sea el umbral y la medida de similaridad.
2. Dos palabras distintas son sinónimas en un grado *GRS_MAX*, dados un umbral y una medida de similaridad, si
   - fijada una acepción de la primera, la segunda pertenece al listado de sinónimos de aquélla.
   - *Lista1* es la lista de todos los grados que se pueden obtener variando todas las acepciones de la segunda palabra para ese umbral y esa medidad de similaridad.
   - *Lista2* es la lista de todas las acepciones que corresponden respectivamente a los grados calculados en el punto anterior.

- *GRS_MAX* es el máximo de los grados de *Lista1* y *AcepB* es la acepción correspondiente a ese grado.

El predicado `comparable` permite comprobar que una palabra está en la lista de sinónimos de otra, una vez fijada determinada acepción de esta última. Sus argumentos son dos palabras *A* y *B* y una acepción *AcepA* de la primera:

```
comparable(A, B, AcepA)
```

Dos palabras *A* y *B* son comparables fijada una de las acepciones de la primera, si:

- *X* es la lista de sinónimos de *A*.
- *B* es uno de los elementos de *X*.

El máximo de los grados con la correspondiente acepción se calcula mediante el predicado `max`, cuyo primer argumento es una lista de acepciones, el segundo la acepción correspondiente al mayor grado, el tercero es una lista de grados y el cuarto el mayor de los grados:

```
max(Lista_Acepciones, Acep, Lista_Grados, GRS_MAX)
```

El predicado opera de la siguiente manera:

1. El máximo de una lista con un solo elemento es ese elemento tanto para las acepciones como para los grados.
2. Si las listas tienen más de un elemento compruébese si el primer elemento de la lista de los grados es mayor o igual al segundo. En caso afirmativo, elimínese el segundo de ellos en ambas listas.
3. Si el primero no es mayor o igual al segundo, elimínese el primero.

Este proceso debe repetirse hasta que sólo quede un elemento en ambas listas, llegando así a la condición límite especificada en 1.

El predicado `sin_acep` realiza el cálculo para dos acepciones concretas de dos palabras. Sus argumentos son los mismos que los del predicado `sinonimo`:

```
sin_acep(A, AcepA, B, Acep, GRS, Umbral, Medida_de_Similaridad)
```

Se definirá este predicado para cada una de las medidas de similaridad citadas anteriormente:

1. *Coeficiente jaccard*: La acepción *AcepA* de una palabra *A* es sinónima de la acepción *AcepB* de otra palabra *B* en un grado *GRS* dado un umbral si:
   - no es el caso que *A* coincida con *B* (esto evita que el compilador haga el cálculo del grado de sinonimia de dos palabras iguales para una misma acepción, puesto que el resultado será trivialmente 1).
   - *X* es la lista de los sinónimos de *A* para la acepción *AcepA*.
   - *Y* es la lista de los sinónimos de *B* para la acepción *AcepB*.
   - *U* es la unión de *X* e *Y*.
   - *I* es la intersección de *X* e *Y*.
   - *NU* es la cardinalidad de la unión *U*.
   - *NI* es la cardinalidad de la intersección *I*.
   - el grado de sinonimia *GRS* es *NI/NU*.
   - dicho grado *GRS* es mayor o igual al umbral.

Los procedimientos para la implementación de las demás medidas de similaridad, *coeficiente del dado*, *coeficiente del coseno*, *coeficiente de similaridad mutua* y *coeficiente de solapamiento* son similares al que se acaba de presentar para el *coeficiente jaccard*.

Como se ha visto, el grado de sinonimia se indica mediante un número real entre 0 y 1. Esto puede resultar poco informativo a un usuario convencional. Empleando etiquetas lingüísticas para expresar los grados de sinonimia se facilitará un resultado que quizás sea más comprensivo para el usuario. Una manera inmediata y muy sencilla de hacer esto es utilizando un predicado que realiza la misma labor que `sinonimo`, pero que en lugar de proporcionar un grado de sinonimia ofrece una verbalización de dicho grado. El nombre de este predicado es `sino_verb` y sus argumentos son dos palabras *A* y *B*, sus respectivas acepciones *AcepA* y *AcepB*, la etiqueta lingüística *Verb* que corresponde al grado de sinonimia, el umbral fijado y la medida de similaridad empleada para medir el grado:

```
sino_verb(A, AcepA, B, AcepB, Verb, Umbral, Medida_de_Similaridad)
```

El predicado funciona de la siguiente manera:

1. La verbalización del grado de sinonimia es “palabras idénticas”, sea cual sea el umbral y la medida de similaridad, si tanto las palabras como las acepciones coinciden y pertenecen al diccionario.
2. Dadas dos acepciones concretas de dos palabras, *Verb* es la verbalización de su grado de sinonimia para un umbral y una medida de similaridad si:

- *GRS* es el grado de sinonimia entre ambas palabras para dichas acepciones, umbral y medida de similaridad.
- las palabras no son iguales.
- ese grado *GRS* se verbaliza como *Verb*.

El predicado `verbaliz` asocia una verbalización a determinado grado de sinonimia, su primer argumento es un grado (número real entre 0 y 1) y el segundo la expresión que verbaliza dicho grado:

```
verbaliz(Grado_de_sinonimia, Verbalización)
```

El criterio para la distribución de los intervalos de cada verbalización es arbitrario aunque debe ser coherente desde un punto de vista intuitivo. Lo que se persigue con la propuesta que se hará a continuación es establecer un procedimiento razonable para que la respuesta del compilador sea más atractiva (obviamente son posibles otras propuestas):

1. Todo grado mayor que 0 y menor o igual a 0.25 se verbaliza con la etiqueta “muy poco sinónimas”.
2. mayor que 0.25 y menor o igual a 0.5 con “poco sinónimas”.
3. mayor que 0.5 y menor o igual a 0.75 con “bastante sinónimas”.
4. mayor que 0.75 y menor o igual a 1 con “muy sinónimas”.

En el procedimiento anterior se ha dividido el intervalo [0, 1] en cuatro sub-intervalos de tal forma que a cada parte le corresponda una verbalización. En principio, no parece haber muchos más modificadores que los citados “muy poco”, “poco”, “bastante” y “muy” para el término “sinónimas”, salvo que se construyan de una forma sofisticada como “entre poco y bastante sinónimas”, “o

bastante o muy sinónimas", etc. La distribución que se acaba de presentar no evita situaciones paradójicas como la que se daría al tener que calificar de "poco sinónimas" a dos expresiones con grado 0.5 y de "bastante sinónimas" a dos expresiones con grado 0.5000001 cuando la distinción entre ambos grados es tan pequeña; y sin embargo, dos grados más dispares como 0.51 y 0.75 serán verbalizados ambos del mismo modo. Por este motivo, en el diccionario electrónico de sinónimos que será descrito en el próximo capítulo, se emplearán estrategias propias de la programación visual para evitar este problema. El uso de una barra de desplazamiento para indicar la verbalización del grado de sinonimia proporcionará una forma gráfica de visualizar si dos expresiones son muy poco, poco, bastante o muy sinónimas. Pero por el momento, en este capítulo se empleará la distribución que se acaba de describir.

El código correspondiente al programa completo figura en el siguiente cuadro:

```
%CÁLCULO DEL GRADO DE SINONIMIA ENTRE PALABRAS

sin_dic(concesion,[concesion,permiso,licencia,gracia,
     privilegio],1).
sin_dic(concesion,[concesion,epitrope],2).
sin_dic(permiso,[permiso,autorizacion,consentimiento,licencia,
     venia, beneplacito,aquiescencia],1).
sin_dic(licencia,[licencia,permiso,autorizacion,consentimiento,
     venia,concesion],1).
sin_dic(licencia,[licencia,abuso,osadia,atrevimiento,desenfreno,
     libertinaje],2).
sin_dic(gracia,[gracia,beneficio,favor,merced,don,regalo],1).
sin_dic(gracia,[gracia,perdon,indulto],2).
sin_dic(gracia,[gracia,benevolencia,amistad,afabilidad,agrado],
     3).
sin_dic(gracia,[gracia,garbo,donaire,sal,salero,angel,atractivo,
     encanto],4).
sin_dic(gracia,[gracia,chiste,agudeza,ocurrencia],5).
sin_dic(privilegio,[privilegio,prerrogativa,concesion],1).
sin_dic(epitrope, [epitrope,permision],1).
```

```
sin_dic(epitrope, [epitrope,concesion],2).

sinonimo(A,X,A,Y,1.0,_,coeficiente_jaccard):-
      sin_dic(A,Z,X),
      X=Y.
sinonimo(A,AcepA,B,AcepB,GRS_MAX,Umbral,coeficiente_jaccard):-
      comparable(A,B,AcepA),
      findall(GRS,sin_acep(A,AcepA,B,Acep,GRS,Umbral,
            coeficiente_jaccard),Lista1),
      findall(Acep,sin_acep(A,AcepA,B,Acep,GRS,Umbral,
            coeficiente_jaccard),Lista2),
      max(Lista2,AcepB,Lista1,GRS_MAX).
sinonimo(A,X,A,Y,1.0,_,coeficiente_del_dado):-
      sin_dic(A,Z,X),
      X=Y.
sinonimo(A,AcepA,B,AcepB,GRS_MAX,Umbral,coeficiente_del_dado):-
      comparable(A,B,AcepA),
      findall(GRS,sin_acep(A,AcepA,B,Acep,GRS,Umbral,
            coeficiente_del_dado),Lista1),
      findall(Acep,sin_acep(A,AcepA,B,Acep,GRS,Umbral,
            coeficiente_del_dado),Lista2),
      max(Lista2,AcepB,Lista1,GRS_MAX).
sinonimo(A,X,A,Y,1.0,_,coeficiente_del_coseno):-
      sin_dic(A,Z,X),
      X=Y.
sinonimo(A,AcepA,B,AcepB,GRS_MAX,Umbral,
coeficiente_del_coseno):-
      comparable(A,B,AcepA),
      findall(GRS,sin_acep(A,AcepA,B,Acep,GRS,Umbral,
            coeficiente_del_coseno),Lista1),
      findall(Acep,sin_acep(A,AcepA,B,Acep,GRS,Umbral,
            coeficiente_del_coseno),Lista2),
      max(Lista2,AcepB,Lista1,GRS_MAX).
sinonimo(A,X,A,Y,1.0,_,coeficiente_de_similaridad_mutua):-
      sin_dic(A,Z,X),
      X=Y.
sinonimo(A,AcepA,B,AcepB,GRS_MAX,Umbral,
coeficiente_de_similaridad_mutua):-
      comparable(A,B,AcepA),
      findall(GRS,sin_acep(A,AcepA,B,Acep,GRS,Umbral,
            coeficiente_de_similaridad_mutua),Lista1),
      findall(Acep,sin_acep(A,AcepA,B,Acep,GRS,Umbral,
            coeficiente_de_similaridad_mutua),Lista2),
      max(Lista2,AcepB,Lista1,GRS_MAX).
      sinonimo(A,X,A,Y,1.0,_,coeficiente_de_solapamiento):-
      sin_dic(A,Z,X),
      X=Y.
sinonimo(A,AcepA,B,AcepB,GRS_MAX,Umbral,
coeficiente_de_solapamiento):-
      comparable(A,B,AcepA),
      findall(GRS,sin_acep(A,AcepA,B,Acep,GRS,Umbral,
            coeficiente_de_solapamiento),Lista1),
      findall(Acep,sin_acep(A,AcepA,B,Acep,GRS,Umbral,
```

```
            coeficiente_de_solapamiento),Lista2),
      max(Lista2,AcepB,Lista1,GRS_MAX).

comparable(A,B,AcepA):-
      sin_dic(A,X,AcepA),
      miembro(B,X).

max([B],B,[A],A).
max([D,E|F],Y,[A,B|C],X):-
      A >= B,
      max([D|F],Y,[A|C],X),
      !.
max([D,E|F],Y,[A,B|C],X):-
      max([E|F],Y,[B|C],X).

sin_acep(A,AcepA,B,AcepB,GRS,Umbral,coeficiente_jaccard):-
      not(A=B),
      sin_dic(A,X,AcepA),
      sin_dic(B,Y,AcepB),
      union(X,Y,U),
      inter(X,Y,I),
      num(U,NU),
      num(I,NI),
      GRS is (NI/NU),
      GRS >= Umbral.
sin_acep(A,AcepA,B,AcepB,GRS,Umbral,coeficiente_del_dado):-
      not(A=B),
      sin_dic(A,X,AcepA),
      sin_dic(B,Y,AcepB),
      inter(X,Y,I),
      num(I,NI),
      num(X,NX),
      num(Y,NY),
      GRS is (2.0*NI)/(NX+NY),
      GRS >= Umbral.
sin_acep(A,AcepA,B,AcepB,GRS,Umbral,coeficiente_del_coseno):-
      not(A=B),
      sin_dic(A,X,AcepA),
      sin_dic(B,Y,AcepB),
      inter(X,Y,I),
      num(I,NI),
      num(X,NX),
      num(Y,NY),
      SQRX is sqrt(NX),
      SQRY is sqrt(NY),
      GRS is NI/(SQRX*SQRY),
      GRS >= Umbral.
sin_acep(A,AcepA,B,AcepB,GRS,Umbral,
coeficiente_de_similaridad_mutua):-
      not(A=B),
      sin_dic(A,X,AcepA),
      sin_dic(B,Y,AcepB),
      inter(X,Y,I),
```

```
      num(I,NI),
      num(X,NX),
      num(Y,NY),
      SLR is NI/NX,
      SRL is NI/NY,
      GRS is (SLR+SRL)/2.0,
      GRS >= Umbral.
sin_acep(A,AcepA,B,AcepB,GRS,Umbral,
coeficiente_de_solapamiento):-
      not(A=B),
      sin_dic(A,X,AcepA),
      sin_dic(B,Y,AcepB),
      inter(X,Y,I),
      num(I,NI),
      num(X,NX),
      num(Y,NY),
      min(NX,NY,Min),
      GRS is NI/Min,
      GRS >= Umbral.

sino_verb(A,AcepA,A,AcepA,palabras_identicas,_,_):-
      sin_dic(A,_,AcepA).
sino_verb(A,AcepA,B,AcepB,Verb,Umbral,MedSim):-
      sinonimo(A,AcepA,B,AcepB,GRS,Umbral,MedSim),
      not(A=B),
      verbaliz(GRS,Verb).

verbaliz(Grado,muy_poco_sinonimas):-
      Grado > 0.0,
      Grado =< 0.25.
verbaliz(Grado,poco_sinonimas):-
      Grado > 0.25,
      Grado =< 0.5.
verbaliz(Grado,bastante_sinonimas):-
      Grado > 0.5,
      Grado =< 0.75.
verbaliz(Grado,muy_sinonimas):-
      Grado > 0.75,
      Grado =< 1.0.

miembro(A,[A|_]).
miembro(A,[_|C]):-
      miembro(A,C).

min(A,B,A):-
A=<B,
!.
min(A,B,B).

num([],0.0).
num([A|B],Num):-
      num(B,N),
      Num is N+1.0.
```

```
union([],X,X).
union([A|B],Y,Z):-
      miembro(A,Y),
      !,
      union(B,Y,Z).
union([A|B],Y,[A|Z]):-
      union(B,Y,Z).

inter([],_,[]).
inter([A|B],Y,[A|Z]):-
      miembro(A,Y),
      !,
      inter(B,Y,Z).
inter([_|B],Y,Z):-
      inter(B,Y,Z).
```

(Los predicados `miembro`, `min`, `num`, `union`, `inter`, `sqrt` y `findall` utilizados en este programa, han sido presentados en el primer capítulo de este trabajo).

Podemos ahora preguntar cuál es el grado de sinonimia entre dos palabras pertenecientes al diccionario, como "licencia" y "concesión", dado un umbral de sinonimia (0 en el ejemplo) y una medida de similaridad (*coeficiente jaccard* en el ejemplo). Una consulta de ese tipo se formula de la siguiente manera:

```
?- sinonimo(concesion,Acep_concesion,licencia,Acep_licencia,
Grado,0,coeficiente_jaccard).

Acep_concesion = 1
Acep_licencia = 1
Grado = 0.375
```

Asimismo, podremos preguntar qué palabras son sinónimas de "concesión" dada una acepción de la misma (la primera), un umbral (0) y una medida de similaridad (*coeficiente jaccard*):

```
?- sinonimo(concesion,1,Y,Acep_Y,Grado,0,coeficiente_jaccard).

Y = concesion          Y = licencia          Y = privilegio
Acep_Y = 1             Acep_Y = 1            Acep_Y = 1
Grado = 1.0            Grado = 0.375         Grado = 0.333333

Y = permiso            Y = gracia
Acep_Y = 1             Acep_Y = 2
```

```
Grado = 0.2          Grado = 0.142857
```

Con el predicado `sino_verb` se obtiene la siguiente respuesta para la primera consulta:

```
?- sino_verb(concesion,Acep_concesion,licencia,Acep_licencia,
Verb,0,coeficiente_jaccard).

Acep_concesion = 1
Acep_licencia = 1
Verb = poco_sinonimas
```


2.2.- Sinonimia intralingüística entre oraciones

En este apartado serán tratados en Prolog tres casos de sinonimia entre oraciones. El primero de ellos recibirá el nombre de *caso de la sustituibilidad* y tiene lugar cuando en una oración se intercambian algunas palabras por sus sinónimas obteniéndose una oración sinónima de la primera. Dado que la sinonimia entre las palabras sustituidas se considerará desde un punto de vista gradual como en el apartado anterior, la sinonimia oracional será tratada de la misma forma. Para dar cuenta de ello se presentará un programa que calcula el grado de sinonimia entre dos oraciones.

Los otros dos casos de sinonimia oracional serán considerados desde un punto de vista preciso. Uno de estos es el *caso de la pasiva*, que consiste en afirmar la sinonimia de dos oraciones cuando una de ellas es la transformación en pasiva de la otra. Por último, se analizará el *caso de los inversos*, que consiste en considerar sinónimas a oraciones que contienen términos pertenecientes a alguno de los pares del estilo "padre-hijo", "abuelo-nieto", "tio-sobrino", etc., dispuestos del modo ejemplificado en "Luis es padre de Alberto" y "Alberto es hijo de

Luis". Ambos casos serán tratados haciendo uso del procesamiento de gramáticas y la gestión del significado con Prolog.

2.2.1.- El caso de la sustituibilidad

Se tratará de averiguar cómo afecta al grado de sinonimia de dos oraciones la sustitución de una palabra por su sinónima. En principio, parece razonable pensar que si en una oración se intercambia una palabra por un sinónimo en grado *GRS* de ésta, el grado de sinonimia entre la oración inicial y la que resulta de la sustitución debería ser *GRS*. Sin embargo, se plantea un problema cuando el intercambio se aplica a más de una palabra en la misma oración. Si tal cosa sucede, existirán tantos grados GRS_1, …, GRS_n como sustituciones se hayan realizado, con lo que resulta necesario efectuar alguna operación con ellos que proporcione el grado de sinonimia oracional. Una propuesta intuitivamente aceptable parece el uso de una t-norma para llevar a cabo este cálculo. Las t-normas son utilizadas en lógica borrosa para modelizar la conjunción y en este trabajo se experimentará con tres de las más habituales (véase Trillas, E., Alsina, C., Terricabras, J. M. [1995] capítulo 2).:

1. *El producto*: $GRS = GRS_1 * GRS_2$.
2. *El mínimo*: $GRS = min(GRS_1, GRS_2)$.
3. *La t-norma de £ukasiewicz*: $GRS = max(GRS_1 + GRS_2 - 1, 0)$.

Sinonimia entre oraciones sin fijar las acepciones

El predicado `orac_sin` calcula el grado de sinonimia entre dos oraciones:

```
orac_sin(O1, O2, Gr, Acep1, Acep2, Umb, MS, T_norma)
```

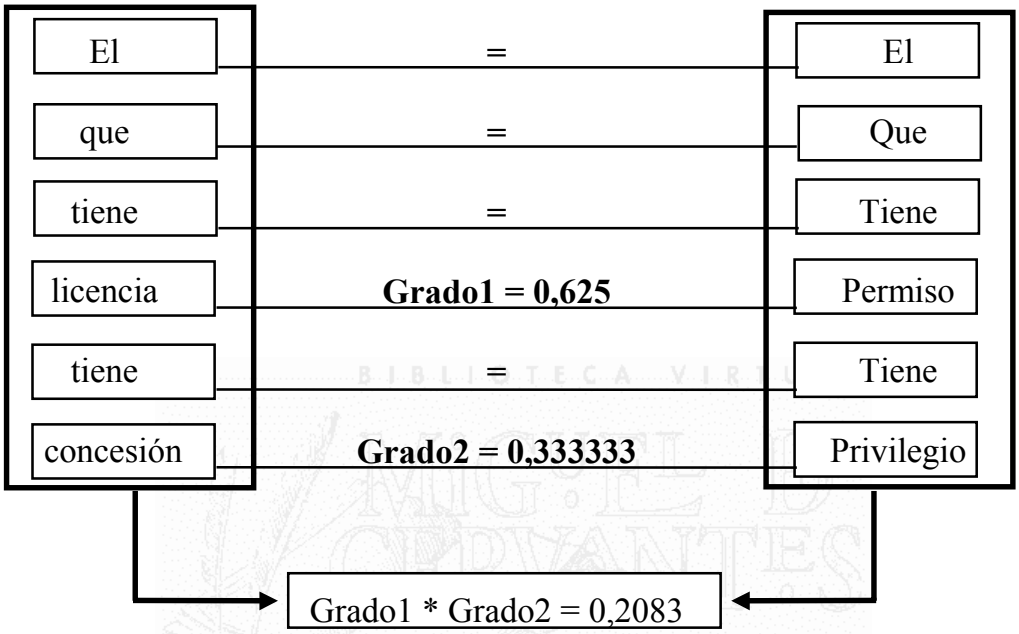
Los dos primeros argumentos son listas donde se incluirán oraciones del lenguaje natural, el tercero es el grado de sinonimia entre ambas oraciones, el cuarto es una lista cuyos elementos serán las palabras sustituidas en la primera oración seguidas de la acepción correspondiente y el quinto lo mismo que el cuarto pero para la segunda oración, el sexto y el séptimo corresponden al umbral y la medida de similaridad que se emplearán para medir el grado de sinonimia entre las palabras y el octavo corresponde a la t-norma empleada para realizar el cálculo de la sinonimia entre ambas oraciones. El predicado se define de la siguiente manera, dados un umbral de sinonimia, una medida de similaridad y una t-norma, siendo necesaria una cláusula para cada una de estas últimas:

1. La lista vacía es sinónima de la lista vacía en grado 1.
2. Si las cabezas son iguales respectivamente en las listas que representan la primera y segunda oración entonces el grado de sinonimia no variará.
3. Si las cabezas no son iguales entonces:
    - Si son sinónimas, en la penúltima lista se incluirá la cabeza de la primera oración seguida de la acepción que la hace sinónima de la cabeza de la segunda; en la última lista se incluirá la cabeza de la segunda oración seguida de la acepción que la hace sinónima de la primera.

- Opérese el grado de sinonimia que la oración tenga en ese momento con el grado de sinonimia de las cabezas mediante la t-norma y guárdese el resultado para la computación de las siguientes palabras.
- Realícese todo esto siempre y cuando el grado de sinonimia de las oraciones sea mayor o igual al grado fijado en el umbral.

4. Repítase el procedimiento anterior con las restantes palabras de la oración.

Lo que hace el compilador es comprobar una a una cada palabra de ambas oraciones respectivamente. Si las palabras coinciden, el grado de sinonimia no variará, si no coinciden comprobará si son sinónimas y, si de hecho lo son, calculará su grado. Finalmente, opera ese resultado con el resultado obtenido en computaciones anteriores mediante la t-norma. Esto queda esquematizado en la siguiente figura para la t-norma producto, utilizando como ejemplo palabras empleadas en el apartado anterior:



El problema ahora es que cuando se le pregunta al compilador por las oraciones sinónimas de una dada, éste realiza todas las combinaciones de palabras en todas sus acepciones. Sin embargo, lo usual es que una oración sirva de contexto suficiente para fijar una de las acepciones de determinada palabra. Es decir, las palabras que figuran en una oración son empleadas usualmente en una acepción concreta (salvo que estemos haciendo un uso del lenguaje distinto del común, como por ejemplo, el literario). La detección automática de las acepciones que, de determinada palabra, se están empleando en una oración es un problema complejo e interesante en el ámbito del procesamiento del lenguaje natural que es conocido con el nombre de *desambiguación de significados* (*sense disambiguation*). Aunque no siempre es el caso, la oración suele darnos pistas sobre cuál de las acepciones se está empleando en ella. Otras veces tendremos que recurrir a otro tipo de información extralingüística para dilucidar este problema.

Sinonimia entre oraciones fijando las acepciones

A continuación se tratará de dar cuenta de cómo una oración puede servir de contexto para detectar las acepciones de las palabras que contiene. Para ello se establecerán los siguientes criterios teniendo en cuenta que, aunque son razonables, no son efectivos en todos los casos sobre los que se podrían aplicar:

1. Si en una misma oración aparecen dos palabras que son sinónimas en determinadas acepciones respectivamente, es muy probable que las acepciones empleadas sean las que las hacen sinónimas.
2. Muchas veces, determinadas palabras, que no tienen por qué ser sinónimas de una dada, son indicio de que dicha palabra se está empleando en determinada acepción. Por ejemplo, la palabra "banco" tiene en castellano varias acepciones como "entidad financiera", "asiento para varias personas" o "grupo extenso de peces"; pero si en determinada oración aparece junto a palabras como "dinero", "cheque", "crédito", "préstamo", etc. éstas serán un buen indicio de que "banco" responde a la acepción de "entidad financiera".

Para estos criterios se utilizará el siguiente predicado, constituido por los mismos argumentos que `orac_sin`:

`orac_sinonima(O1, O2, Gr, Acep1, Acep2, Umb, MS, T_norma)`

Se define de la siguiente manera, dados un umbral de sinonimia, una medida de similaridad y una t-norma:

1. Una oración *A* es sinónima de otra *B* (fijando las acepciones) si:

- *Z* es la lista de acepciones de palabras fijadas en *A*.
- *A* es sinónima de *B*.
- ni *Z* ni la lista de acepciones de *B* son vacías (esto evita que el programa proporcione como respuesta el caso trivial de que dos oraciones idénticas son sinónimas en grado 1 con las listas de acepciones de *A* y *B* vacías).

2. Cuando en la oración no hay ninguna palabra que fije las acepciones de ninguna otra, entonces la lista de acepciones de *A* es vacía; en este caso la respuesta del sistema debe combinar todas las acepciones de las palabras sin fijar ninguna.

El predicado `acep_orac_orac` es el que nos permite comprobar si se pueden asociar todas las palabras de una oración con todas las palabras de otra. Eventualmente, esta segunda oración será la misma que la primera, aunque pudiera no ser así si lo que queremos es fijar las acepciones de las palabras de una oración con respecto a otras oraciones distintas a la dada. El predicado posee tres listas como argumentos:

```
acep_orac_orac(Oración, Oración, Lista_de_acepciones)
```

En las dos primeras listas se incluirán oraciones (eventualmente la misma) y en la tercera, palabras de la oración seguidas de las acepciones fijadas. El predicado se define de la siguiente forma:

1. La lista de acepciones fijadas de la lista vacía respecto a otra lista es la lista vacía.

2. Se añadirá *R* y *S* a la lista de acepciones fijadas de una oración respecto a otra *X* si:
   - *Z* es la lista de acepciones fijadas de la primera palabra de la oración respecto a *X*.
   - *Z* no es la lista vacía (es decir, no es el caso que *X* no fija ninguna acepción de la primera palabra).
   - La lista [*R*,*S*], es el resultado de eliminar en *Z* las acepciones que se repiten (es decir, en *Z* sólo se fija una acepción).
3. Se añadirá una variable a la lista de acepciones fijadas de una oración respecto a otra *X* (con el fin de que el programa proporcione todas las combinaciones de acepciones) si:
   - *Z* es la lista de acepciones fijadas de la primera palabra de la oración respecto a *X*.
   - *Z* no es la lista vacía.
   - La lista *W*, es el resultado de eliminar en *Z* las acepciones que se repiten.
   - En *W* hay más de una acepción fijada.
4. Este proceso debe repetirse con todas las palabras de la primera oración.

El predicado `acep_pal_orac` es el que, dada una palabra y una oración, fija las acepciones de dicha palabra respecto a la oración; para ello, busca qué palabras de ésta se pueden asociar con la dada. El resultado es una lista donde

aparece la palabra inicial seguida de la acepción empleada en la asociación y repetida tantas veces como asociaciones se hayan realizado. Sus argumentos son una palabra, una oración y una lista donde se incluirá la palabra con las acepciones que resultan de las asociaciones realizadas.

acep_pal_orac(Palabra, Oración, Lista_de_acepciones)

El predicado funciona de la siguiente manera:

1. La lista de las acepciones fijadas de cualquier palabra respecto a la lista vacía será la lista vacía.
2. La oración fija una de las acepciones de la palabra y consiguientemente ésta, seguida de la acepción fijada, es incluida en la lista de acepciones, si:
   - la palabra y la cabeza de la lista de la oración son distintas.
   - la palabra y la cabeza son asociables para esa acepción fijada.
3. Este proceso debe repetirse con todas las palabras de la oración.

Cuando en la oración se realizan varias asociaciones de la palabra para la misma acepción, en la última lista aparecerán la palabra seguida de la acepción tantas veces como asociaciones se hayan hecho. Para evitar la repetición de acepciones, en el predicado `acep_orac_orac` se hace referencia al predicado `quitar_rep`, cuyos dos argumentos son listas:

quitar_rep(Lista_con_repetición, Lista_sin_repetición)

El primer argumento es la lista con acepciones repetidas y el segundo la lista sin repeticiones. La estructura es la siguiente:

1. La eliminación de las repeticiones en la lista vacía es la lista vacía.

2. Si la lista no es vacía elimínense los dos primeros elementos si coinciden respectivamente con los dos siguientes.
3. Repítase el proceso anterior comprobando los elementos de la primera lista dos a dos hasta agotarla.

El predicado `asocia` empleado en `acep_pal_orac` nos permite asociar palabras. Este predicado está formado por dos palabras y una acepción fijada correspondiente a la primera palabra:

```
asocia(A, B, Acep_A)
```

Se asociarán dos palabras, y por tanto, se podrá fijar la acepción de la primera, dado un umbral y una medida de similaridad, si:

- La primera en determinada acepción es sinónima de la segunda en cualquier acepción en cierto grado, o bien,
- la segunda en cualquier acepción es sinónima de la primera en determinada acepción en cierto grado.

Para dar cuenta del segundo de los criterios que fijan las acepciones en una oración, se necesita crear otra base de datos que sea capaz de asociar determinadas palabras con otras y fijar las acepciones de estas últimas. No es fácil encontrar palabras que ayuden a identificar las acepciones de palabras como "concesión", con la misma efectividad con la que "dinero", "cheque", etc. lo hacen para "banco" en el ejemplo mencionado más arriba. Una opción podría ser considerar que el uso de verbos como "tener" indican que se está utilizando la primera acepción de "concesión" y verbos como "emplear" indican que se usa la segunda.

Cuantos más criterios se incorporen mejor detectará el programa las acepciones de las palabras utilizadas en cada oración. Solamente hay que tener la cautela de que los criterios sean compatibles y no interfieran los unos con los otros fijando acepciones diferentes para la misma palabra bajo las mismas condiciones.

Tal y como se ha hecho para la sinonimia entre palabras se puede definir un predicado que en lugar de proporcionar el grado numérico de sinonimia, nos dé una verbalización del mismo. Es el siguiente:

orac_sinonima_verb(O1, O2, Verb, Acep1, Acep2, Umb, MS, T_norma)

Este predicado está formado por dos oraciones, la verbalización del grado de sinonimia entre ellas, dos listas que contienen las acepciones fijadas en la primera y segunda oración respectivamente, un umbral de sinonimia, una medida de similaridad y una t-norma. Dados un umbral de sinonimia, una medida de similaridad y una t-norma, *Verb* es la verbalización del grado de sinonimia de dos oraciones si:

- *GRS* es el grado de sinonimia entre ambas.
- *GRS* se verbaliza como *Verb*.

El código correspondiente a lo descrito figura en el siguiente cuadro:

```
%CÁLCULO DEL GRADO DE SINONIMIA ENTRE ORACIONES
(SUSTITUIBILIDAD)

orac_sin([],[],1.0,[],[],_,_,producto).
orac_sin([A|B],[A|D],X,Y,Z,Umbral,MedSim,producto):-
     orac_sin(B,D,X,Y,Z,Umbral,MedSim,producto).
orac_sin([A|B],[C|D],Y,[A,Acep_A|W],[C,Acep_C|Q],Umbral,MedSim,
producto):-
     sinonimo(A,Acep_A,C,Acep_C,Z,Umbral,MedSim),
     orac_sin(B,D,X,W,Q,Umbral,MedSim,producto),
```

```
      Y is X*Z,
      Y >= Umbral.
orac_sin([],[],1.0,[],[],_,_,minimo).
orac_sin([A|B],[A|D],X,Y,Z,Umbral,MedSim,minimo):-
      orac_sin(B,D,X,Y,Z,Umbral,MedSim,minimo).
orac_sin([A|B],[C|D],Y,[A,Acep_A|W],[C,Acep_C|Q],Umbral,MedSim,
minimo):-
      sinonimo(A,Acep_A,C,Acep_C,Z,Umbral,MedSim),
      orac_sin(B,D,X,W,Q,Umbral,MedSim,minimo),
      min(X,Z,Y),
      Y >= Umbral.
orac_sin([],[],1.0,[],[],_,_,lukasiewicz).
orac_sin([A|B],[A|D],X,Y,Z,Umbral,MedSim,lukasiewicz):-
      orac_sin(B,D,X,Y,Z,Umbral,MedSim,lukasiewicz).
orac_sin([A|B],[C|D],Y,[A,Acep_A|W],[C,Acep_C|Q],Umbral,MedSim,
lukasiewicz):-
      sinonimo(A,Acep_A,C,Acep_C,Z,Umbral,MedSim),
      orac_sin(B,D,X,W,Q,Umbral,MedSim,lukasiewicz),
      SUM is X+Z-1,
      maxim(0,SUM,Y),
      Y >= Umbral.

orac_sinonima(A,B,GR,Z,Y,Umbral,MedSim,T_norma):-
      acep_orac_orac(A,A,Z),
      orac_sin(A,B,GR,Z,Y,Umbral,MedSim,T_norma),
      not(Z = []),
      not(Y = []).
orac_sinonima(A,B,GR,W,Y,Umbral,MedSim,T_norma):-
      acep_orac_orac(A,A,Z),
      Z = [],
      orac_sin(A,B,GR,W,Y,Umbral,MedSim,T_norma),
      not(W = []),
      not(Y = []).

acep_orac_orac([],X,[]).
acep_orac_orac([A|B],X,[R,S|Y]):-
      acep_pal_orac(A,X,Z),
      not(Z=[]),
      quitar_rep(Z,[R,S]),
      acep_orac_orac(B,X,Y),
      !.
acep_orac_orac([A|B],X,[A,K|Y]):-
      acep_pal_orac(A,X,Z),
      not(Z=[]),
      quitar_rep(Z,W),
      not(num(W,2)),
      acep_orac_orac(B,X,Y),
      !.
acep_orac_orac([A|B],X,Y):-
      acep_orac_orac(B,X,Y).

acep_pal_orac(A,[],[]).
acep_pal_orac(A,[B|C],[A,AcepA|E]):-
```

```
      not(A=B),
      asocia(A,B,AcepA),
      acep_pal_orac(A,C,E),
      !.
acep_pal_orac(A,[B|C],D):-
      acep_pal_orac(A,C,D).

quitar_rep([],[]).
quitar_rep([A,AcepA,A,AcepA|C],[A,AcepA|D]):-
      quitar_rep([A,AcepA|C],[A,AcepA|D]),
      !.
quitar_rep([A,AcepA|C],[A,AcepA|D]):-
      quitar_rep(C,D).

%PREDICADO PARA EL PRIMER CRITERIO

asocia(A,B,AcepA):-
      sinonimo(A,AcepA,B,AcepB,GR,0,_);
      sinonimo(B,AcepB,A,AcepA,GR,0,_).

%PREDICADO PARA EL SEGUNDO CRITERIO

asocia(concesion,tengo,1).
asocia(concesion,empleo,2).

orac_sinonima_verb(A,B,Verb,W,Y,Umbral,MedSim,T_norma):-
      orac_sinonima(A,B,GRS,W,Y,Umbral,MedSim,T_norma),
      verbaliz(GRS,Verb).

maxim(A,B,A):-
A>=B,
!.
maxim(A,B,B).
```

(El predicado `maxim` utilizado en este programa, ha sido definido en el primer capítulo de este trabajo).

Añadiendo el código anterior al programa que da cuenta de la sinonimia entre palabras del apartado 2.1, se puede preguntar el grado de sinonimia de dos oraciones dadas. (En los ejemplos que siguen, se pedirá que el cálculo de sinonimia se realice exclusivamente con el *coeficiente jaccard* y con la t-norma producto):

```
orac_sinonima([el,chico,tiene,licencia],[el,chico,tiene,
concesion],GR,Y,Z,0,coeficiente_jaccard, producto).
```

```
GR = 0.375
Y = [licencia,1]
Z = [concesion,1]
```

Con la siguiente consulta:

```
orac_sinonima([el,que,tiene,licencia,tiene,concesion],B,Grado,Y,
Z,0,coeficiente_jaccard, producto).
```

Se observará que el sistema fija la acepción 1 de "licencia" y la acepción 1 de "concesión" debido a que ambas palabras aparecen en la oración y son asociables por el hecho de ser sinónimas en esas acepciones:

```
B = [el,que,tiene,licencia,tiene,concesion]
Grado = 1
Y = [licencia,1,concesion,1]
Z = [licencia,1,concesion,1]

B = [el,que,tiene,licencia,tiene,permiso]
Grado = 0.2
Y = [licencia,1,concesion,1]
Z = [licencia,1,permiso,1]

B = [el,que,tiene,licencia,tiene,licencia]
Grado = 0.375
Y = [licencia,1,concesion,1]
Z = [licencia,1,licencia,1]

B = [el,que,tiene,licencia,tiene,gracia]
Grado = 0.142857
Y = [licencia,1,concesion,1]
Z = [licencia,1,gracia,2]

...
```

Respecto al segundo criterio, en la siguiente consulta podemos observar como el sistema fija la acepción 1 de "concesión" por el hecho de utilizar el verbo "tener":

```
orac_sinonima([yo,tengo,concesion],B,Grado,Y,Z,0,
coeficiente_jaccard, producto).

B = [yo,tengo,concesion]          B = [yo,tengo,gracia]
Grado = 1                         Grado = 0.142857
Y = [concesion,1]                 Y = [concesion,1]
Z = [concesion,1]                 Z = [gracia,2]
```

```
B = [yo,tengo,permiso]
Grado = 0.2
Y = [concesion,1]
Z = [permiso,1]

B = [yo,tengo,licencia]
Grado = 0.375
Y = [concesion,1]
Z = [licencia,1]
```

```
B = [yo,tengo,privilegio]
Grado = 0.333333
Y = [concesion,1]
Z = [privilegio,1]
```

Si probamos lo mismo con la oración "empleo una concesión en el texto", el sistema fijará la acepción 2 de "concesión" por el hecho de utilizar el verbo "emplear":

```
orac_sinonima([empleo,una,concesion,en,el,texto],B,Grado,Y,Z,0,
coeficiente_jaccard,producto).

B = [empleo,una,concesion,en,el,texto]
Grado = 1
Y = [concesion,2]
Z = [concesion,2]

B = [empleo,una,epitrope,en,el,texto]
Grado = 1
Y = [concesion,2]
Z = [epitrope,2]
```

Con el predicado `orac_sinonima_verb` se puede hacer la primera consulta, con lo que se obtendrá la verbalización del grado de sinonimia:

```
orac_sinonima_verb([el,chico,tiene,licencia],[el,chico,tiene,
concesion],Verb,Y,Z,0, coeficiente_jaccard, producto).

Verb = poco_sinonimas
Y = [licencia,1]
Z = [concesion,1]
```

El tratamiento automático de cuestiones relativas al lenguaje natural siempre conlleva dudas sobre si es generalizable y aplicable a todos los casos. La propuesta para oraciones descrita anteriormente es una prueba de ello. Como se ha podido ver, los criterios establecidos para fijar acepciones de palabras funcionan sólo en algunas ocasiones. Por ejemplo, "cheque" y "cobrar"

contribuyen a fijar adecuadamente la acepción de "entidad financiera" de la palabra "banco" en la oración "voy al banco a cobrar el cheque", pero no lo hacen correctamente en la oración "me he dejado olvidado encima del banco del parque el cheque que iba a cobrar". En este caso, las palabras "encima" y "parque" parecen tener mayor peso para fijar la acepción correspondiente a "asiento para varias personas". En este sentido podrían darse tres tipos de casos:

1. Aquellos en los que las demás palabras de la oración sirven de contexto suficiente para fijar las acepciones de una de ellas, la cual, puede ser desambiguada de un modo automatizable. Éste es el caso de la oración "el que tiene licencia tiene concesión" empleada en las consultas del programa anterior.
2. Aquellos en los que una máquina no es capaz de desambiguar la palabra pero un humano podría hacerlo por el hecho de tener un mayor dominio de lenguaje natural o por contar con información extralingüística. Éste sería el caso de la oración del ejemplo anterior "me he dejado olvidado encima del banco del parque el cheque que iba a cobrar".
3. Aquellos en los que ni una máquina ni un humano serían capaces de desambiguar la palabra. Un ejemplo de esto es el que se daría si alguien dijese "quedamos delante del banco que hay en mi calle" cuando en dicha calle hay un banco para sentarse y una sucursal de una

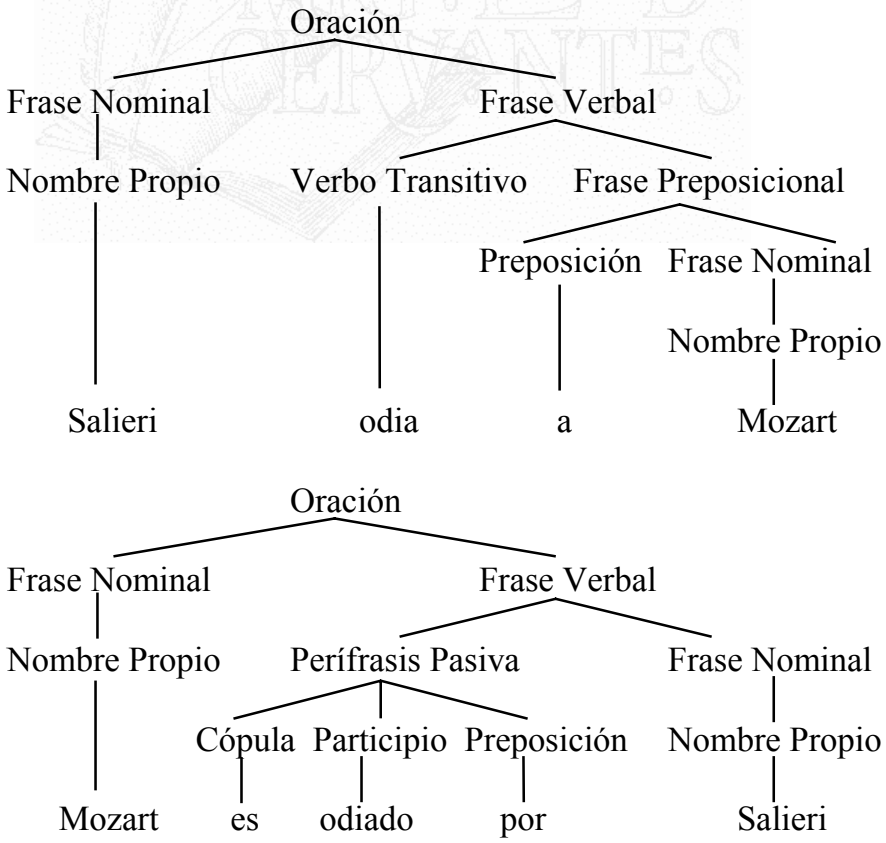
entidad bancaria. Sin más información que ésta, ambos interlocutores estarían abocados a no entenderse.

Dada la potencia expresiva del lenguaje natural, podrían definirse un sinfín de situaciones como las descritas en 2 y 3, que provocan frecuentemente malos entendidos incluso en la comunicación humana, por eso no es de extrañar que en la comunicación persona-máquina se ponga en duda la existencia de solución para este problema, lo que conduce a la tendencia común de evitarlo. Una forma de hacerlo es instando al usuario a fijar las acepciones, algo a lo que también se recurre en las situaciones de comunicación humana mencionadas cuando uno de los interlocutores pregunta "¿a qué te refieres cuando dices …?", "¿qué quieres decir con …?", etc.

La propuesta descrita para palabras adolece de esta dificultad en menor medida, ya que permite extender el análisis al conjunto de expresiones que figuran en un diccionario de sinónimos, el cual puede constituir una muestra del uso de la sinonimia en el lenguaje natural. El capítulo IV de este trabajo tratará de proporcionar un análisis más completo de la sinonimia entre palabras mediante la elaboración de un diccionario electrónico de sinónimos que es capaz de calcular el grado de sinonimia de dos entradas. Aunque el usuario deberá fijar la acepción de la palabra que introduce como entrada, los sinónimos que el sistema proporcionará como respuesta serán desambiguados indicándose la acepción correspondiente.

2.2.2.- El caso de la pasiva

Es común considerar como sinónimas dos oraciones cuando una de ellas es la versión en pasiva de la otra. Podría afirmarse que la oración "Salieri odia a Mozart" tiene el mismo significado que "Mozart es odiado por Salieri" aunque sean dos oraciones distintas con una estructura gramatical diferente, tal y como se manifiesta en sus respectivos árboles gramaticales:



Para detectar de forma automática este tipo de sinonimia se recurrirá de nuevo al análisis semántico de la oración utilizando Prolog. De ello resultará que el significado de "Salieri odia a Mozart" es el mismo que el de "Mozart es odiado por Salieri", es decir, *odia(Salieri, Mozart)*, aunque para obtenerlo se haga uso de distintas reglas.

El programa que reconoce oraciones del tipo de las explicitadas anteriormente es el siguiente:

```
%GRAMÁTICA PARA SINONIMIA ENTRE ORACIONES (PASIVA)

oracion(Lista1, Resto):-
     frase_nominal(Lista1, Lista2),
     frase_verbal(Lista2, Resto).

frase_nominal(Lista1, Resto):-
     nombre_propio(Lista1, Resto).

frase_verbal(Lista1, Resto):-
     verbo_transitivo(Lista1, Lista2),
     frase_preposicional(Lista2, Resto).
frase_verbal(Lista1, Resto):-
     perifrasis_pasiva(Lista1, Lista2),
     frase_nominal(Lista2, Resto).

frase_preposicional(Lista1, Resto):-
     preposicion(Lista1, Lista2),
     frase_nominal(Lista2, Resto).

perifrasis_pasiva(Lista1, Resto):-
     copula(Lista1, Lista2),
     participio(Lista2, [por|Resto]).

nombre_propio(['Salieri'|Resto], Resto).
nombre_propio(['Mozart'|Resto], Resto).

verbo_transitivo([odia|Resto], Resto).

preposicion([a|Resto], Resto).

copula([es|Resto], Resto).

participio([odiado|Resto], Resto).
```

A partir de esta gramática se pueden incorporar en ella los criterios por los que se debe guiar su análisis semántico, los cuales indican que el significado asociado a un nombre propio es ese mismo nombre, el del verbo transitivo y el participio será el predicado diádico *odia(Agente,Paciente)*, el de la perífrasis de pasiva es heredado del participio que es una de sus partes integrantes, el de una frase nominal es el mismo que el del nombre propio que es su único componente,

y el de una frase preposicional es heredado de la frase nominal que es uno de sus integrantes. La mayor dificultad está en la frase verbal. Si está en activa, el significado de la frase nominal que va después del verbo transitivo es el paciente. En ese caso, como es necesaria la información sobre quién es el agente, que se obtendrá de la frase nominal que encabeza la oración, permanecerá variable en el segundo argumento. Si la frase verbal está en pasiva, el significado de la frase nominal que va después del verbo transitivo es el agente. En ese caso, como es necesaria la información sobre quién es el paciente, que se obtendrá del significado de la frase nominal que encabeza la oración, permanecerá, de igual modo, variable en el segundo argumento. Finalmente, el significado de la oración completa es el de la frase verbal compuesto con el de la frase nominal que encabeza la oración, que proporciona la información de quién es el agente en el caso de las oraciones en activa y quién es el paciente en el caso de las oraciones en pasiva.

Los demás componentes de la oración (`preposicion` y `copula`) no influyen en el significado tal y como aquí es concebido para este ejemplo concreto; sin embargo, son necesarios para la generación de las oraciones gramaticales.

El predicado `oracion_restricc` establece la restricción semántica que indica que el agente y el paciente deben ser dos individuos distintos. Con esto se evita la generación de oraciones como "Mozart odia a Mozart", "Mozart es odiado por Mozart", etc. que, aunque son gramaticalmente correctas y

perfectamente comprensibles podrían ser expresadas de una forma más usual como "Mozart se odia a sí mismo". Dicho predicado tendrá como argumentos un significado y su oración correspondiente:

oracion_restricc(Significado,Oración)

Mediante el predicado `sinonima` se establece que dos oraciones son sinónimas si son gramaticales, satisfacen la restricción y su significado es el mismo. Tal predicado tendrá dos oraciones como argumentos:

sinonima(Oración1,Oración2)

El código correspondiente figura a continuación:

```
%SINONIMIA ENTRE ORACIONES (PASIVA)

oracion(Sgdo,Lista1,Resto):-
      frase_nominal(X,Lista1,Lista2),
      frase_verbal(Sgdo,X,Lista2,Resto).

frase_nominal(Sgdo,Lista1,Resto):-
      nombre_propio(Sgdo,Lista1,Resto).

frase_verbal(Sgdo,Agente,Lista1,Resto):-
      verbo_transitivo(Sgdo,Agente,Paciente,Lista1,Lista2),
      frase_preposicional(Paciente,Lista2,Resto).
frase_verbal(Sgdo,Paciente,Lista1,Resto):-
      perifrasis_pasiva(Sgdo,Agente,Paciente,Lista1,Lista2),
      frase_nominal(Agente,Lista2,Resto).

perifrasis_pasiva(Sgdo,Agente,Paciente,Lista1,Resto):-
      copula(Lista1,Lista2),
      participio(Sgdo,Agente,Paciente,Lista2,[por|Resto]).

frase_preposicional(Sgdo,Lista1,Resto):-
      preposicion(Lista1,Lista2),
      frase_nominal(Sgdo,Lista2,Resto).

nombre_propio(salieri,['Salieri'|Resto],Resto).
nombre_propio(mozart,['Mozart'|Resto],Resto).

verbo_transitivo(odia(Agente,Paciente),Agente,Paciente,
      [odia|Resto],Resto).

participio(odia(Agente,Paciente),Agente,Paciente,[odiado|Resto],
      Resto).
```

```
preposicion([a|Resto],Resto).

copula([es|Resto],Resto).

oracion_restricc(odia(Agente,Paciente),A):-
      oracion(odia(Agente,Paciente),A,[]),
      not(Agente = Paciente).

sinonima(A,B):-
      oracion_restricc(Significado,A),
      oracion_restricc(Significado,B).
```

Ahora se puede consultar qué oraciones son sinónimas de una dada:

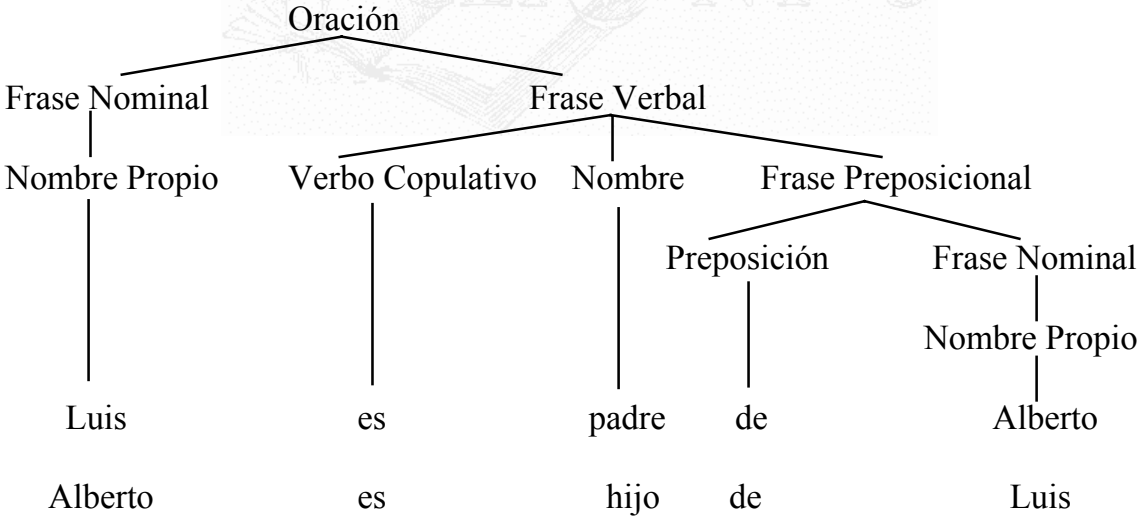
```
?- sinonima(['Salieri',odia,a,'Mozart'],Sinonima).

Sinonima = [Salieri,odia,a,Mozart]

Sinonima = [Mozart,es,odiado,por,Salieri]
```

También se puede preguntar cuáles de las oraciones generadas por la gramática son sinónimas. Como toda oración es sinónima de sí misma, hecho del que da cuenta el programa como se acaba de comprobar en la consulta anterior, se evitarán los casos de sinonimia que incluyan oraciones idénticas formulando la consulta de la siguiente forma:

```
sinonima(Oracion1,Oracion2), Oracion1 \== Oracion2.

Oracion1 = [Salieri,odia,a,Mozart]
Oracion2 = [Mozart,es,odiado,por,Salieri]

Oracion1 = [Salieri,es,odiado,por,Mozart]
Oracion2 = [Mozart,odia,a,Salieri]

Oracion1 = [Mozart,odia,a,Salieri]
Oracion2 = [Salieri,es,odiado,por,Mozart]

Oracion1 = [Mozart,es,odiado,por,Salieri]
Oracion2 = [Salieri,odia,a,Mozart]
```

2.2.3.- El caso de los inversos

Otro ejemplo de sinonimia entre oraciones es aquel en el que figuran términos inversos. Ejemplos de términos inversos son padre-hijo, abuelo-nieto, tio-sobrino, etc. Estos pares, generan sinonimia entre oraciones como "Luis es padre de Alberto" y "Alberto es hijo de Luis". A diferencia del caso de la pasiva, aquí la estructura de ambas oraciones sinónimas coincide; lo único que varían son los terminales:



La gramática que genera oraciones con esta estructura es la siguiente:

```
%GRAMÁTICA PARA SINONIMIA ENTRE ORACIONES (INVERSOS)

oracion(Lista1,Resto):-
      frase_nominal(Lista1,Lista2),
      frase_verbal(Lista2,Resto).

frase_nominal(Lista1,Resto):-
      nombre_propio(Lista1,Resto).

frase_verbal(Lista1,Resto):-
      verbo_copulativo(Lista1,Lista2),
      nombre(Lista2,Lista3),
      frase_preposicional(Lista3,Resto).

frase_preposicional(Lista1,Resto):-
      preposicion(Lista1,Lista2),
      frase_nominal(Lista2,Resto).
```

```
nombre_propio(['Luis'|Resto],Resto).
nombre_propio(['Alberto'|Resto],Resto).

nombre([padre|Resto],Resto).
nombre([hijo|Resto],Resto).
nombre([abuelo|Resto],Resto).
nombre([nieto|Resto],Resto).
nombre([tio|Resto],Resto).
nombre([sobrino|Resto],Resto).

preposicion([de|Resto],Resto).

verbo_copulativo([es|Resto],Resto).
```

Asociando un significado a cada componente de la gramática, se podrá obtener la estructura semántica de cualquier oración generada. El significado de un nombre propio es ese nombre, el de un nombre es un predicado con dos argumentos, los cuales serán instanciados con los valores semánticos de frases nominales, el de una frase preposicional coincide con el de la frase nominal que la compone, el de una frase nominal es el mismo que el del nombre propio que la integra. El significado de la frase verbal es el del nombre que figura como uno de sus componentes; es un predicado con dos argumentos, el segundo de los cuales es instanciado con el significado de la frase preposicional. A la oración completa le corresponde el significado de la frase verbal compuesto con el de la frase nominal que figura al principio de aquélla.

El predicado `oracion_restricc` establece la restricción que evita que se generen oraciones del estilo de "Luis es padre de Luis", "Alberto es padre de Alberto", etc. que o bien, no son semánticamente correctas, ya que un individuo no puede ser padre de sí mismo, o bien son ambiguas debido a que no se puede

distinguir el individuo denotado por cada una de las ocurrencias de los nombres propios.

El predicado `sinonima` indica que dos oraciones son sinónimas si son generadas por la gramática, satisfacen la restricción y sus significados correspondientes son inversos. Por último, el predicado `inverso` indica qué significados corresponden a términos inversos.

En el siguiente cuadro figura el código correspondiente:

```
%SINONIMIA ENTRE ORACIONES (INVERSOS)

oracion(Sgdo,Lista1,Resto):-
      frase_nominal(Sgdo1,Lista1,Lista2),
      frase_verbal(Sgdo,Sgdo1,Lista2,Resto).

frase_nominal(Sgdo,Lista1,Resto):-
      nombre_propio(Sgdo,Lista1,Resto).

frase_verbal(Sgdo,Sgdo1,Lista1,Resto):-
      verbo_copulativo(Lista1,Lista2),
      nombre(Sgdo,Sgdo1,Sgdo2,Lista2,Lista3),
      frase_preposicional(Sgdo2,Lista3,Resto).

frase_preposicional(Sgdo,Lista1,Resto):-
      preposicion(Lista1,Lista2),
      frase_nominal(Sgdo,Lista2,Resto).

nombre_propio(luis,['Luis'|Resto],Resto).
nombre_propio(alberto,['Alberto'|Resto],Resto).

nombre(padre(Padre,Hijo),Padre,Hijo,[padre|Resto],Resto).
nombre(hijo(Hijo,Padre),Hijo,Padre,[hijo|Resto],Resto).
nombre(abuelo(Abuelo,Nieto),Abuelo,Nieto,[abuelo|Resto],Resto).
nombre(nieto(Nieto,Abuelo),Nieto,Abuelo,[nieto|Resto],Resto).
nombre(tio(Tio,Sobrino),Tio,Sobrino,[tio|Resto],Resto).
nombre(sobrino(Sobrino,Tio),Sobrino,Tio,[sobrino|Resto],Resto).

preposicion([de|Resto],Resto).

verbo_copulativo([es|Resto],Resto).

oracion_restricc(padre(Padre,Hijo),A):-
      oracion(padre(Padre,Hijo),A,[]),
      not(Padre = Hijo).
oracion_restricc(hijo(Hijo,Padre),A):-
```

```
      oracion(hijo(Hijo,Padre),A,[]),
      not(Hijo = Padre).
oracion_restricc(abuelo(Abuelo,Nieto),A):-
      oracion(abuelo(Abuelo,Nieto),A,[]),
      not(Abuelo = Nieto).
oracion_restricc(nieto(Nieto,Abuelo),A):-
      oracion(nieto(Nieto,Abuelo),A,[]),
      not(Nieto = Abuelo).
oracion_restricc(tio(Tio,Sobrino),A):-
      oracion(tio(Tio,Sobrino),A,[]),
      not(Tio = Sobrino).
oracion_restricc(sobrino(Sobrino,Tio),A):-
      oracion(sobrino(Sobrino,Tio),A,[]),
      not(Sobrino = Tio).

sinonima(O1,O2):-
      oracion_restricc(Sgdo1,O1),
      oracion_restricc(Sgdo2,O2),
      inverso(Sgdo1,Sgdo2).

inverso(padre(X,Y),hijo(Y,X)).
inverso(abuelo(X,Y),nieto(Y,X)).
inverso(tio(X,Y),sobrino(Y,X)).
```

Con la siguiente consulta se obtienen todas las oraciones sinónimas que genera la gramática:

```
?- sinonima(O1,O2).

O1 = [Luis,es,padre,de,Alberto]
O2 = [Alberto,es,hijo,de,Luis]

O1 = [Alberto,es,padre,de,Luis]
O2 = [Luis,es,hijo,de,Alberto]

O1 = [Luis,es,abuelo,de,Alberto]
O2 = [Alberto,es,nieto,de,Luis]

O1 = [Alberto,es,abuelo,de,Luis]
O2 = [Luis,es,nieto,de,Alberto]

O1 = [Luis,es,tio,de,Alberto]
O2 = [Alberto,es,sobrino,de,Luis]

O1 = [Alberto,es,tio,de,Luis]
O2 = [Luis,es,sobrino,de,Alberto]
```

Recapitulando lo visto en este capítulo, se comenzó por el tratamiento de la traducción (sinonimia interlingüística) con Prolog. Respecto a la sinonimia intralingüística se analizó el caso de la sinonimia entre palabras y entre oraciones, dentro de la cual, se estudió el caso de la sustituibilidad, donde pareció más adecuado concebir la sinonimia desde un punto de vista gradual, y los casos de la pasiva y las oraciones que contienen términos inversos, donde su orientación fue de corte preciso.

De todos los aspectos tratados, se podría indicar que su generalización a todo el lenguaje natural es una tarea irrealizable por su excesiva complejidad. Sin embargo, se ha podido ver que algunos problemas como el de la traducción pueden proporcionar resultados interesantes si se restringen a contextos concretos. A pesar de todo esto, el tratamiento que se ha realizado sobre la sinonimia entre palabras requiere cierta atención, ya que, a pesar de la complejidad de su generalización, su extensión al marco de un diccionario de sinónimos parece lo suficientemente global como para que resulte interesante.

Ya se ha indicado en el capítulo anterior que un diccionario de sinónimos no incluye todos los casos de sinonimia de un lenguaje natural y también se ha señalado allí que, en ocasiones concretas, estos diccionarios dan cuenta de relaciones semánticas distintas de la sinonimia. Sin embargo, también se ha afirmado que constituye una muestra razonablemente fiel del comportamiento de la sinonimia en la práctica y que además posee la ventaja de su adecuación al procesamiento automático.

El próximo capítulo se dirige al diseño de un diccionario electrónico de sinónimos basado en el tratamiento de la sinonimia entre palabras establecida en este capítulo. Ello pondrá de manifiesto el carácter aplicado de la presente propuesta.

CAPÍTULO IV

Un diccionario electrónico de sinónimos

Ya se ha dicho que parece ser algo comúnmente admitido que la sinonimia como relación precisa, sinonimia total o sinonimia absoluta no existe en general, lo que lleva a considerarla como una cuestión más bien aproximada. Este hecho puede dar pie para afirmar que el conjunto de sinónimos de una expresión *a* es un conjunto borroso al estilo de los descritos por L. A. Zadeh (véase Zadeh, L. A. [1965]). Como tal conjunto borroso, cada elemento *b*, pertenecerá a él con un cierto grado: el grado de sinonimia entre *b* y *a*. Si somos capaces de proporcionar criterios que posibiliten el cálculo de este grado, podremos obtener el conjunto borroso de expresiones sinónimas a partir de una dada. Si dichos criterios son automatizables, la obtención de dichos conjuntos borrosos podrá ser también automática. El diccionario electrónico de sinónimos (DES) que se describirá en este capítulo permite obtener dichos conjuntos borrosos de forma automática

para un diccionario de sinónimos en lengua española. El tipo de aproximación al problema consta de dos fases: una primera fase de observación de regularidades en dicho diccionario y una segunda fase de búsqueda de un modelo que dé cuenta de dichas regularidades.

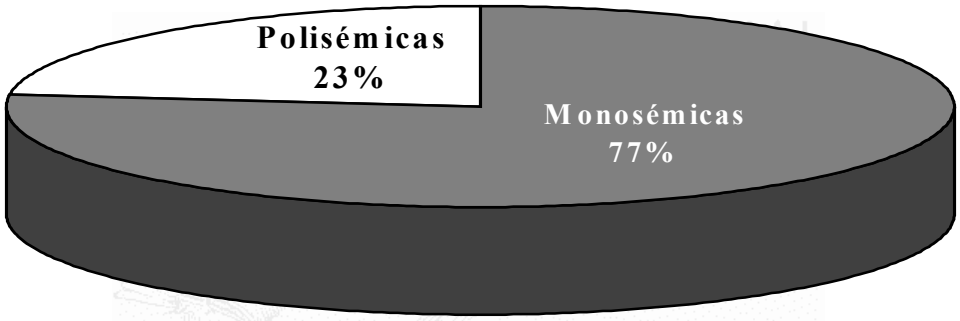
1.- Primera fase: El diccionario de sinónimos

Se utilizará el *Diccionario avanzado de sinónimos y antónimos de la lengua española* (Blecua J. M. [1997]) como herramienta para analizar las características de la relación de sinonimia. Este diccionario está formado por un conjunto de 27.043 expresiones. Para referir a este conjunto se utilizará, de aquí en adelante, el nombre de "DIC". Los elementos de DIC son, por regla general, palabras, aunque también se pueden encontrar locuciones, frases hechas, etc. Se utilizarán indistintamente los términos "palabra" o "expresión" para referir a las entidades pertenecientes a DIC.

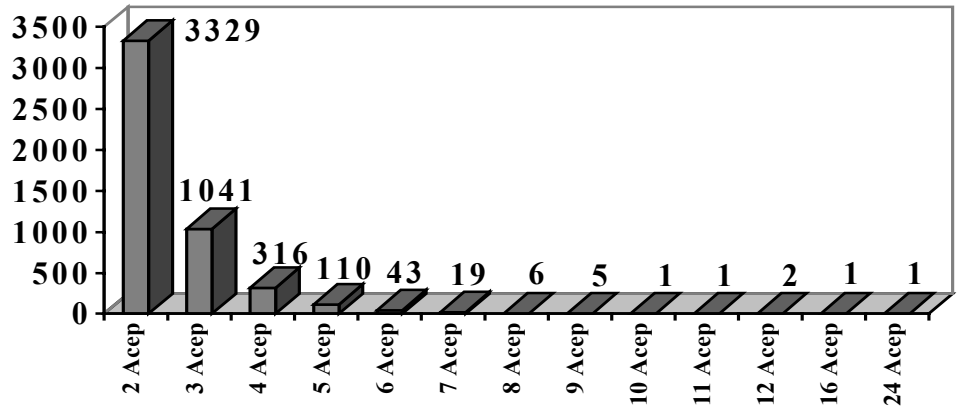
El diccionario proporciona, dada una palabra de DIC, un conjunto de sinónimos para cada una de sus acepciones. Según esto pueden observarse los siguientes casos:

d1. Hay palabras que sólo tienen una acepción, como por ejemplo "abadía". A la única acepción de esta palabra le corresponde el conjunto de sinónimos {"convento", "monasterio", "cartuja", "cenobio", "priorato"}.

d2. Hay palabras que tienen varias acepciones como “abordar”, a cuya primera acepción le corresponde el conjunto de sinónimos {“chocar”, “aportar”, “atracar”} y a la segunda {“emprender”, “plantear”}. Se podría definir el concepto de “ser polisémica dentro del diccionario” como aquella propiedad que poseen las palabras que tienen más de una acepción. El siguiente gráfico muestra los porcentajes de palabras polisémicas y monosémicas que posee el diccionario:

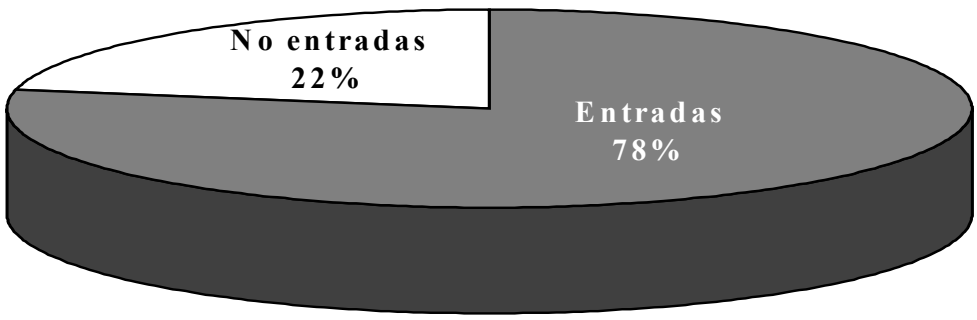


Dentro de las palabras polisémicas, la mayor parte de ellas poseen dos o tres acepciones. El número de palabras con más de cuatro desciende considerablemente, como se puede apreciar en el siguiente gráfico:



Normalmente, los conjuntos de sinónimos correspondientes a cada acepción de la misma palabra no poseen palabras en común, como sucede en el ejemplo proporcionado anteriormente. Sin embargo, ésta no es una regla general que se da en todo el diccionario; prueba de ello son los conjuntos correspondientes a la primera y cuarta acepciones de la palabra "abandonado", {"dejado", "descuidado", "desidioso", "negligente", "desamparado"} y {"desvalido", "desamparado"}, que, como se puede observar, poseen en común la palabra "desamparado".

d3. Es importante señalar que también hay palabras que pertenecen a DIC cuyo conjunto de sinónimos es vacío. Si definimos el concepto "ser entrada del diccionario" como aquella propiedad que cumplen las palabras a las que el diccionario asocia un conjunto no vacío de sinónimos, las palabras cuyo conjunto de sinónimos es vacío, son aquellas que figuran como sinónimas de otras pero no son entradas del diccionario. Un ejemplo de esto es la palabra "cartuja", que no es entrada aunque figura como sinónima de "abadía". El siguiente gráfico indica los porcentajes de palabras que son entradas y que no lo son:



d4. Según lo dicho en el punto anterior, habrá entradas en cuyo conjunto de sinónimos figuren palabras que no son entradas como sucede en el ejemplo de "abadía", pudiendo darse el caso límite de que una entrada tenga como único sinónimo una palabra que no figura como entrada. Éste es el caso de la palabra "abanarse" a la que, para su única acepción, le corresponde el conjunto de sinónimos {"abanicarse"}, cuyo único elemento no figura como entrada.

d5. Por otra parte, dos palabras pueden no figurar como sinónimas en el diccionario pero sus correspondientes conjuntos para determinadas acepciones de las mismas tener expresiones en común. Por ejemplo, la palabra "toldo" tiene como conjunto de sinónimos para su primera acepción {"tendal", "vela", "pabellón"}, y la palabra "dosel" tiene como conjunto de sinónimos para su única acepción {"pabellón"}. Ambos conjuntos de sinónimos poseen en común la palabra "pabellón". Intuitivamente, en estos casos, debería haber cierto grado de relación entre ambas palabras.

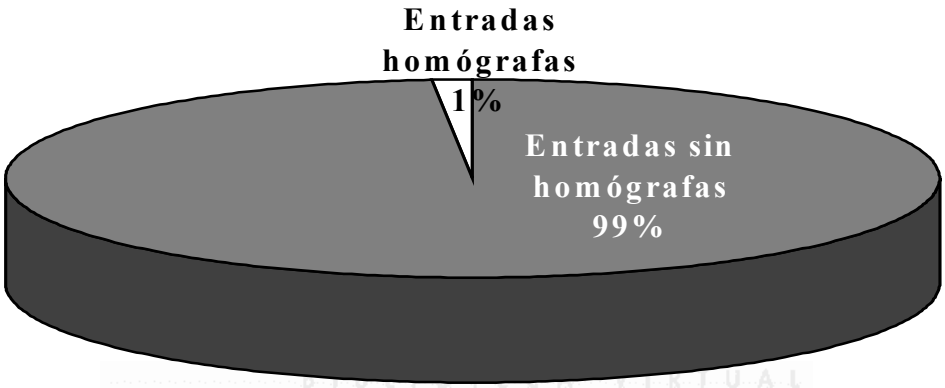
d6. Un caso límite del descrito en el punto anterior es el que se da cuando a dos palabras distintas le corresponde el mismo conjunto de sinónimos. Tal cosa sucede con las palabras "acromatismo", y "acromatopsia", cada una de las cuales sólo posee una acepción, a la que corresponde el conjunto de sinónimos {"daltonismo"}. Intuitivamente, parece que ambas palabras deberían ser sinónimas por

corresponderles el mismo conjunto de sinónimos, sin embargo no figuran como tales en el diccionario.

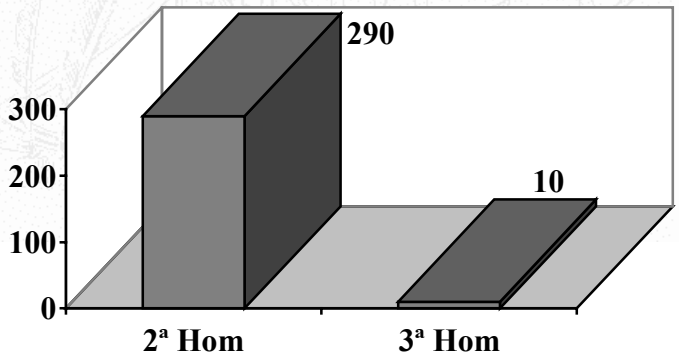
d7. A pesar de todo, lo usual es que a dos palabras distintas le correspondan dos conjuntos distintos de sinónimos. Sin embargo, todavía se da un caso límite cuando una palabra figura como única sinónima de otra para una acepción de ésta, y ésta como única sinónima de la primera para una acepción de aquélla. Por ejemplo, la palabra "pernil" figura como única sinónima de "pernera" para su única acepción y la palabra "pernera" figura como única sinónima de la segunda acepción de "pernil". Intuitivamente, en estos casos, el grado de sinonimia debería ser relativamente alto, ya que la relación entre ambas palabras, o mejor dicho, entre ambas acepciones de las palabras, parece ser bastante estrecha.

d8. Finalmente, existe un problema que afecta a todos los diccionarios cuando tienen que tratar con palabras homógrafas, es decir, palabras de distinta procedencia que evolucionaron hacia una misma forma, lo que tiene como consecuencia la existencia de palabras distintas, y por lo tanto con significado distinto, que se escriben de igual modo. En este caso no se está hablando de dos acepciones diferentes de la misma palabra sino de dos palabras distintas. Generalmente, este problema se soluciona de una forma relativamente eficiente poniendo índices a estas palabras. Éste es el caso de las palabras "I aportar" y "II aportar",

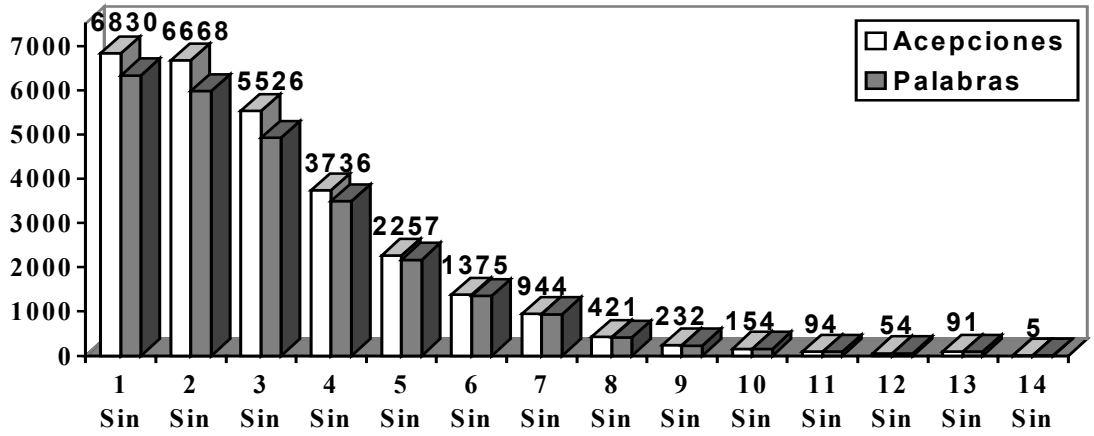
la primera de las cuales posee una acepción y la segunda dos. Con todo, el porcentaje de entradas homógrafas de DIC es mínimo, como se puede comprobar en el siguiente gráfico:



Además sólo figuran entradas con dos o tres homógrafas:



Se puede definir el concepto "figurar como sinónimo en el diccionario" como la relación que se da entre dos expresiones *a* y *b* de DIC cuando *a* pertenece al conjunto de sinónimos correspondiente a al menos una acepción de *b*. En el diccionario empleado, el número de elementos de dicho conjunto no sobrepasa los 14. En el siguiente gráfico se muestra el número de acepciones y de palabras según la cardinalidad del conjunto de sus sinónimos:



Una vez definida esta relación de sinonimia relativa al diccionario que se está manejando, se puede ver cómo se comporta dicha relación en ese contexto. Las observaciones son las siguientes:

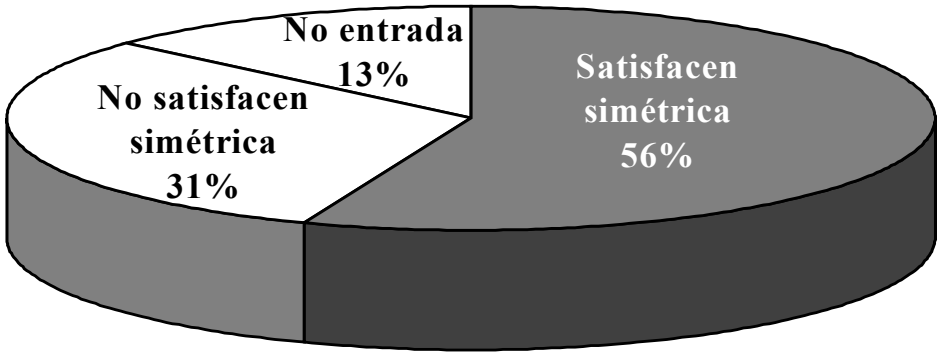
1. Se cumple que para todo elemento *a* de DIC, *a* no figura como sinónimo de sí mismo; bien porque *a* no es entrada o bien porque, siendo *a* entrada del diccionario, no pertenece a ninguno de los conjuntos de sinónimos correspondientes a cada una de sus acepciones. Esto es algo que resulta natural, dado que el objetivo del uso del diccionario es la búsqueda de palabras sinónimas distintas de las que se proporcionan como entrada. Sin embargo, incluir la propia palabra como sinónima de sí misma en todas sus acepciones evita algunos de los problemas indicados anteriormente, como se verá más adelante.
2. La simetría de la relación de sinonimia así definida no se da siempre en el diccionario. Es bastante frecuente que una expresión *a* figure como sinónima de otra *b* y *b* figure como sinónima de *a* como en el caso de las palabras “concesión” y “licencia”, pero puede darse el caso de que

a figure como sinónima de *b* y *b* no figure como sinónima de *a*. Hay dos situaciones en las que esto sucede:

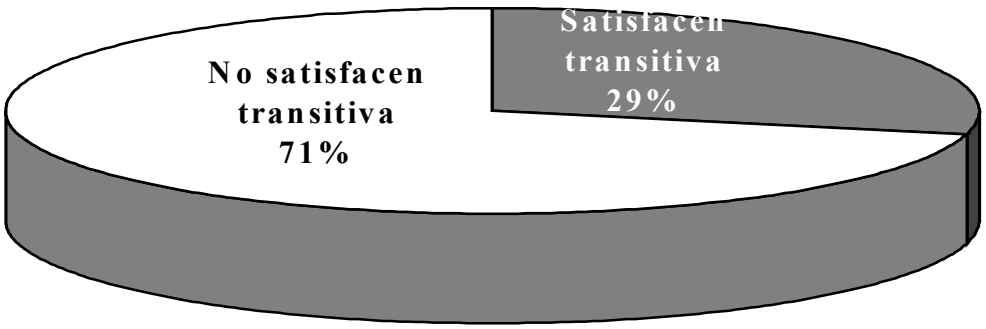
2.1. Cuando *a* pertenece a algún conjunto de sinónimos correspondiente a alguna acepción de *b* y *b* no pertenece a ningún conjunto de sinónimos correspondiente a cualquier acepción de *a*. Por ejemplo, "gracia" figura como sinónima para la primera acepción de "concesión", no figurando esta última como sinónimo para ninguna de las acepciones de la primera.

2.2. Cuando *a* pertenece a un conjunto de sinónimos correspondiente a alguna acepción de *b* y *a* no es entrada del diccionario, como sucede en el caso de "brizo", que figura como sinónimo de la primera acepción de "cuna", pero no es entrada.

El siguiente gráfico indica los porcentajes de pares que satisfacen la propiedad simétrica y los que no la satisfacen, dentro de los cuales, se distingue el porcentaje de aquéllos que no la cumplen por no figurar como entrada uno de los elementos del par:



3. Con la transitividad pasa lo mismo que con la simetría, hay casos en los que se da que una expresión *a* figura como sinónima de otra *b*, esta como sinónima de otra *c* y *a* como sinónima de *c*; como "permiso" que figura como sinónima para la primera acepción de "licencia", ésta como sinónima para la primera acepción de "concesión" y "permiso" como sinónima para la primera acepción de "concesión". Pero hay casos en los que una expresión *a* figura como sinónima de otra *b*, ésta como sinónima de otra *c* y *a* no figura como sinónima de *c*; como "concesión" que figura como sinónima para la primera acepción de "licencia", y ésta como sinónima para la primera acepción de "consentimiento", pero "concesión" no figura como sinónima de "consentimiento". El siguiente gráfico indica los porcentajes de tripletas que satisfacen (no trivialmente) la propiedad transitiva y que no la satisfacen:



Hasta el momento se han descrito algunas características del diccionario de sinónimos que servirá de base de datos para el DES que se presentará en este capítulo. La implementación de esta base de datos ha procurado ser lo más

objetiva posible, en el sentido de que se ha llevado a cabo sin realizar ningún tipo de manipulación en la información contenida en el diccionario. Quizás con cierta manipulación se podrían conseguir mejoras con respecto al cálculo de los grados de sinonimia. Por ejemplo, muchas expresiones poseen como sinónimas expresiones que no figuran como entradas por ser el resultado de incorporar sufijos en otras que sí aparecen como entrada. Éste es el caso de la palabra "mano" que posee como sinónima para su segunda acepción a la palabra "facultades" que no aparece como entrada, pero la palabra "facultad" sí aparece. A veces una variación de género o de número puede llevarnos a una expresión completamente diferente, sin embargo, éste no es el caso del ejemplo, ya que si observamos la segunda acepción de la palabra "facultad", vemos que tiene sinónimos en común con la segunda acepción de la palabra "mano", con lo que resulta plausible que ambas palabras sean sinónimas para las acepciones citadas, a pesar de que no figuran en el diccionario como tales. Lo mismo sucede con las formas pronominales de los verbos. Por ejemplo, "abandonar" posee como sinónima para su tercera acepción, la forma pronominal "separarse"; una vez más, ésta no figura como entrada, pero la palabra "separar" sí, y de nuevo, la cuarta acepción de "separar" tiene sinónimos en común con la tercera acepción de "abandonar" con lo que es razonable pensar que ambas palabras podrían tener cierto grado de sinonimia para esas acepciones. Este problema que tiene lugar con los verbos es especialmente incierto, ya que otras veces los verbos en forma pronominal figuran como entradas. La existencia de casos como estos es uno de

los factores que influye en que la sinonimia no sea simétrica en el diccionario. Otras veces, la carencia de simetría no se debe tanto a la incorporación de sufijos en las palabras como al hecho de que en el diccionario se esté dando cuenta de relaciones distintas a la de sinonimia, como es el caso de "granito", que figura como sinónima de "piedra" pero no viceversa. En este ejemplo, "granito" es un hipónimo de "piedra" mientras que la conversa no se da. En otros casos, como los citados "brizo" y "cuna" o "gracia" y "concesión", parece que la causa se debe al olvido por parte del lexicógrafo que elaboró el diccionario. En principio parece haber dos formas de tratar esta cuestión: bien modificando la base de datos para conseguir la simetría o bien buscando un modelo que tenga en cuenta la carencia de esta propiedad.

Otro problema diferente es el de la palabra "sigilo" que figura como sinónima de la única acepción de la palabra "estampilla". La entrada "sigilo" sólo tiene una acepción, a la que le corresponde el conjunto de sinónimos {"silencio", "secreto"}. Ciertamente, la palabra "sigilo" debería tener otra acepción que correspondería a la que la hace sinónima de "estampilla". Este caso afecta al programa descrito en el capítulo anterior respecto a la sinonimia entre palabras en lo tocante a la detección de las acepciones de las palabras que proporciona como respuesta, ya que resulta imposible detectar la acepción correcta de "sigilo" cuando se obtiene como sinónima de "estampilla" debido a que dicha acepción no existe en el diccionario. Sin duda modificar el diccionario para evitar estas irregularidades supondría una mejora en los resultados del DES. Pero dicha labor

de modificación no entra dentro del alcance de este trabajo, por tanto, se utilizará la base de datos tal y como se nos presenta en su versión impresa.

Todas estas observaciones son relativas al diccionario de sinónimos descrito; con un diccionario distinto podrían variar en buena medida. Ésta es la razón por la que se ha tenido la cautela de definir conceptos como "figurar como sinónimo en el diccionario", "ser polisémica dentro del diccionario", etc. y no "ser sinónima", "ser polisémica", etc. Esto resta validez universal a los conceptos definidos, sin embargo, el interés del DES radica en su utilidad práctica, algo a lo que se hará referencia al final de este capítulo.

El objeto de la descripción que se acaba de realizar no era otra que la búsqueda de regularidades en el diccionario para tratar de encontrar un modelo lo más adecuado posible al comportamiento de la sinonimia con el fin de calcular grados de sinonimia entre palabras dentro del diccionario. Sin embargo, como se ha podido observar, las regularidades son escasas; la única propiedad que cumple la relación "figurar como sinónima en el diccionario" es la irreflexiva, aunque, como se ha apuntado, en general se puede admitir que toda palabra es sinónima de sí misma para cada una de sus acepciones. El hecho de que una expresión no figure como sinónima de sí misma, se debe más bien a cuestiones de economía a la hora de elaborar el diccionario.

2.- Segunda fase: El modelo para calcular grados de sinonimia

Con esta panorámica, no parece fácil encontrar un modelo adecuado para la sinonimia tal y como se manifiesta en el diccionario utilizado. No es la pretensión de este trabajo indagar en los motivos por los que esto es así, pero todo apunta a que Lyons parece estar acertado cuando afirma que la relación que se da entre los términos que figuran en un diccionario de sinónimos es una relación de mayor o menor semejanza de significado, a la que él llamaba cuasisinonimia, y no tanto una relación de sinonimia propiamente dicha (véase Lyons, J. [1995]). Y es que si se observa un diccionario de sinónimos con cierto detenimiento se pueden encontrar pares de expresiones que figuran como sinónimas pero que en realidad son hipónimas, hiperónimas, etc. tal es el caso de "granito" que figura como sinónimo de "piedra", o "menta" que figura como sinónimo de "hierba". Incluso puede darse que la palabra que figura como entrada es una simple palabra destacada dentro de la expresión sinónima que facilita su búsqueda como sucede en el caso de "pan" que tiene como sinónimo para su primera acepción la expresión "comer el pan de los niños".

A pesar de todo esto, en Sparck Jones, K. [1986], se propone un modelo para el cálculo del grado de sinonimia utilizando medidas de similaridad como el *coeficiente jaccard* empleado en el capítulo anterior. Aunque el interés primordial de la autora no era el cálculo del grado de sinonimia entre palabras sino el uso de este grado para la realización de grupos de palabras, esta medida se interpreta como la asociación entre dos listas de sinónimos.

Parece, por tanto, que las medidas de similaridad son un modelo adecuado para el cálculo del grado de sinonimia como así lo testimonia su uso en este contexto. Sin embargo, el *coeficiente jaccard* no es la única que existe. Otras medidas de similaridad, como los ya mencionados *coeficiente del dado*, *coeficiente del coseno*, *coeficiente de similaridad mutua* y *coeficiente de solapamiento*, proporcionan nuevas opciones para el cálculo del grado de sinonimia.

Una posible forma de medir la sinonimia que requiere mención especial es el *ratio model* proporcionado por Tversky, (véase Tversky, A. [1977]). Dadas dos expresiones *a* y *b*, esta medida consiste en el cociente entre la cardinalidad del conjunto de sinónimos comunes a *a* y a *b* partido por la suma de las cardinalidades de tres conjuntos de sinónimos:

a) El conjunto de los sinónimos comunes a *a* y a *b* ($A \cap B$).

b) El conjunto de los sinónimos de *a* que no están en *b* ($A - B$); al que se le puede atribuir un peso $\alpha \geq 0$.

c) El conjunto de los sinónimos de *b* que no están en *a* ($B - A$); al que se le puede atribuir un peso $\beta \geq 0$.

$$GRS(a,b) = \frac{|A \cap B|}{|A \cap B| + \alpha|A - B| + \beta|B - A|}$$

Esta medida generaliza algunas de las medidas citadas anteriormente. Por ejemplo, cuando α y β valen ambos 1, coincide con el *coeficiente jaccard*. Cuando α y β valen ambos ½, coincide con el *coeficiente del dado*.

La peculiaridad que hace interesante esta medida de similaridad en este marco es que solamente es simétrica en algunas ocasiones de la misma forma que sucedía con la relación "figurar como sinónima en el diccionario". El propio Tversky indica que la simetría, *GRS(a,b)=GRS(b,a)*, se cumple cuando el número de elementos de *A* es igual al de *B* (|*A*|=|*B*|) o cuando α y β tienen el mismo valor ($\alpha=\beta$) (véase Tversky, A. [1977], p. 333). Por el hecho de no ser simétrica, parece que esta medida de similaridad da cuenta mejor del comportamiento de la sinonimia en el diccionario empleado. Sin embargo, su utilización entraña cierta dificultad, debido a que, como las cardinalidades de *A* y *B* vienen proporcionadas por el propio diccionario, la carencia de simetría radica en que los pesos α y β sean diferentes. Para el establecimiento de dichos pesos se necesitan criterios que indiquen qué pares de expresiones del diccionario satisfacen la simetría y qué pares no la satisfacen. Además, si el objetivo a perseguir en este trabajo es el tratamiento automático de la sinonimia, dichos criterios deberían ser también automatizables y esto no es algo fácil de conseguir.

Un criterio podría ser el de tratar de proporcionar mayor grado de sinonimia a *GRS(a,b)* que a *GRS(b,a)* cuando *a* figura como sinónimo de *b* en el diccionario y *b* no figura como sinónimo de *a*. Esto se consigue de hecho en el DES que se presentará dando el doble de peso a β que a α. Es decir, si *GRS(a,b)* se lee como el grado de sinonimia no simétrico de un sinónimo *a* con respecto a su entrada correspondiente *b*, se le da el doble de peso a la entrada que al

sinónimo. En este caso, la medida de similaridad vendría dada por la siguiente formula:

$$GRS(a,b) = \frac{|A \cap B|}{|A \cap B| + |A - B| + 2 \cdot |B - A|}$$

Así $GRS(a,b) \geq GRS(b,a)$ cuando *a* figura como sinónimo de *b* y *b* no figura como sinónimo de *a* en el diccionario. En los demás casos la medida se comportaría como el *coeficiente jaccard*, al otorgar el valor 1 a α y a β. Con todo, no hay que descartar la posibilidad indicada anteriormente de modificar la base de datos con el fin de evitar la existencia de casos que no satisfacen la simétrica.

Una vez más se debe insistir que todo lo que aquí se ha dicho sobre la sinonimia depende del diccionario empleado. Se ha indicado que la medida de similaridad propuesta por Tversky podría ser la más adecuada al comportamiento de determinados pares de sinónimos en dicho diccionario, sin embargo, otras medidas podrían ser adecuadas para otros pares, éste es el motivo por el que en el DES se han implementado todas las medidas de similaridad citadas aquí con el fin de que el usuario elija la que cree que proporciona resultados más coherentes o adecuados al uso lingüístico. Con cualquiera de estas medidas, el grado de sinonimia será un número real entre 0 y 1, de tal forma que, dos expresiones con grado 0 no figurarán como sinónimas en el DES y dos expresiones con grado 1 figurarán como sinónimas en el grado máximo.

En el DES, el cálculo del grado de sinonimia será realizado según el procedimiento descrito en el capítulo anterior. La acepción correspondiente a la

palabra de entrada es seleccionada por el usuario de la misma forma que la medida de similaridad y el umbral de sinonimia empleados para el cálculo. La acepción correspondiente a la palabra sinónima es detectada automáticamente al mismo tiempo que el grado de sinonimia. Esto supone una ventaja con respecto al diccionario de sinónimos impreso, en el que no se indican las acepciones de las palabras que figuran como sinónimas de una dada, y si lo queremos averiguar hay que hacerlo manualmente buscando la palabra sinónima. Es conveniente señalar que para que la detección de la acepción de la palabra sinónima sea la correcta, esta acepción debe ser contemplada en la base de datos, ya que pueden darse casos como el descrito anteriormente con las palabras "estampilla" y "sigilo", la segunda de las cuales figura como sinónima de la primera pero al buscarla como entrada no se encuentra una acepción de esta que la haga sinónima de la primera. Ya se ha señalado que la solución a este tipo de problemas pasa por la corrección de la base de datos.

Otro problema es que, como el cálculo del grado se realiza operando con los conjuntos de sinónimos de las expresiones del diccionario, no se puede calcular el grado de sinonimia de cualquier par de palabras de DIC. Para poder hacerlo, ambas palabras deben ser entradas en el diccionario y consiguientemente tener asociado un conjunto de sinónimos para cada una de sus acepciones. Una vez más sería necesaria la manipulación de la base de datos para incorporar como entradas aquellas palabras que no lo son.

Tal y como están definidas las medidas de similaridad utilizadas para medir el grado, los pares de palabras que sean entrada podrán tener un grado de sinonimia distinto de 0 cuando sus conjuntos de sinónimos correspondientes a determinadas acepciones de las mismas tengan elementos en común, con lo que se cumpliría la condición $A \cap B \neq \varnothing$, y por lo tanto el numerador que figura en todas las medidas será mayor que cero ($|A \cap B| > 0$). Cuando no tengan elementos en común el grado será 0, es decir, no serán sinónimas. Con esto, las expresiones que no figuran como sinónimas en el diccionario pero sus conjuntos de sinónimos poseen algunos elementos en común (como en el caso descrito en d5) o todos los elementos en común (como en el caso descrito en d6), tendrán ahora un grado de sinonimia distinto de 0, algo que intuitivamente parece correcto como ya se había anticipado al explicitar d5 y d6. Para el ejemplo propuesto en d5 y con el *coeficiente jaccard* como medida de similaridad, el grado de sinonimia entre las palabras "toldo" en su primera acepción y "dosel" en su única acepción es 0.2, ya que:

$$GRS(\text{“toldo”}, \text{“dosel”}) =$$

$$= \frac{|\{\text{"toldo"}, \text{"tendal"}, \text{"vela"}, \text{"pabellón"}\} \cap \{\text{"dosel"}, \text{"pabellón"}\}|}{|\{\text{"toldo"}, \text{"tendal"}, \text{"vela"}, \text{"pabellón"}\} \cup \{\text{"dosel"}, \text{"pabellón"}\}|} =$$

$$= \frac{|\{\text{"pabellón"}\}|}{|\{\text{"toldo"}, \text{"tendal"}, \text{"vela"}, \text{"pabellón"}, \text{"dosel"}\}|} = \frac{1}{5} = 0.2 .$$

Para el ejemplo propuesto en d6, las palabras "acromatismo" y "acromatopsia", poseen el grado de sinonimia 0.3333333333.

En el ejemplo de “toldo” y “dosel” se puede observar que estas palabras figuran en sus respectivos conjuntos de sinónimos. Para todas las medidas de similaridad presentadas se cumple que $GRS(a,a) = 1$, con lo cual, toda expresión será sinónima de sí misma en el DES. Incorporar el caso reflexivo en la base de datos hace que sea 1 el grado de sinonimia de expresiones cuando una de ellas figura como única sinónima de otra para una acepción de ésta, y ésta como única sinónima de la primera para una acepción de aquélla, algo que resulta intuitivamente correcto como se afirmaba en el caso d7 del apartado anterior.

Las observaciones d1 y d2, realizadas en el apartado anterior para el diccionario son válidas también en el DES, a las cuales se pueden añadir las siguientes comparándolas con las allí proporcionadas:

e1. Con respecto a d3, como se ha indicado, no se puede calcular el grado de sinonimia de los sinónimos que no figuran como entradas en el diccionario, por lo tanto, tampoco pueden ser elementos del conjunto borroso de sinónimos de una palabra; sin embargo, este hecho no significa que estos carezcan de importancia, ya que contribuyen al cálculo del grado de sinonimia de aquellas expresiones en las que aparecen como sinónimas, como se podrá observar en el punto siguiente. Por eso en el DES se ha habilitado una lista en la que figuran todos estos sinónimos que no son entrada.

e2. Cuando sucede lo que se describe en d4, que una expresión *a* que no es entrada figura como única sinónima de otra *b* que sí lo es, no se puede

calcular el grado de sinonimia entre ambas, pero la entrada *b* puede tener otros sinónimos. Esto puede suceder por dos motivos:

e2.1. Porque la expresión *a* figure como sinónima de otras, lo que hace que *b* tenga al menos un sinónimo en común con estas. Este sinónimo sería *a*. Esto puede suceder aunque *a* no sea entrada. Por ejemplo, la palabra "yesar" que no es entrada del diccionario, figura como única sinónima para la única acepción de "aljezar" y además figura como sinónima para la única acepción de "yesal"; por lo tanto, "aljezar" y "yesal" poseen cierto grado de sinonimia puesto que poseen en común al menos la palabra "yesar" aunque ésta no sea entrada.

e2.2. Porque la propia palabra *b* figure como sinónima de otra entrada *c* aunque *c* no figure como sinónima de *b*. Esto sucede debido a que la relación "figurar como sinónima en el diccionario" definida en el apartado anterior no es simétrica. Por el hecho de haber incorporado en la base de datos la propia palabra como sinónima, *b* figurará en el conjunto de sinónimos de *b* y también de *c*, con lo cual, *b* es un elemento común al conjunto de sinónimos de *b* y de *c*; por tanto, el grado de sinonimia entre las dos palabras es distinto de 0, poseyendo de esta forma cierto grado de sinonimia. Casualmente, el ejemplo descrito en a2.1 ilustra también el caso a2.2, ya que la palabra "yesar" que no es

entrada del diccionario, figura como única sinónima para la única acepción de "aljezar" y ésta figura como sinónima para la única acepción de "yesal", por lo tanto, "aljezar" y "yesal" poseen cierto grado de sinonimia puesto que poseen en común la palabra "aljezar" que figura como sinónima de "yesal" aunque "yesal" no figure como sinónima de "aljezar".

e3. Para el caso de las palabras homógrafas, descrito en d8, se emplean índices para distinguirlas, de modo análogo a como se hace en el diccionario original. La ventaja que tiene el DES es que, así como detecta automáticamente la acepción que corresponde a la palabra sinónima también detecta el índice correspondiente en el caso de que la palabra tenga alguna homógrafa. Por ejemplo, "alistar" tiene dos homógrafas indexadas como "1" y "2", cada una de las cuales sólo posee una acepción. A la primera de ellas se le hace corresponder en el diccionario el conjunto de sinónimos {"poner", "sentar en lista", "listar", "inscribir", "afiliar", "matricular", "sentar plaza", "enganchar", "reclutar"}, a la segunda el conjunto {"prevenir", "preparar", "aparejar", "disponer", "aprestar"}. Cada vez que la palabra "alistar" figure como sinónima de otra, el DES detectará automáticamente cuál de las homógrafas es la que se emplea como sinónima de esa otra. Por ejemplo, si buscamos la palabra "afiliar" en su única acepción, se detectará como sinónima la primera homógrafa

de “alistar”, sin embargo, si buscamos la palabra “preparar”, es la segunda homógrafa de “alistar” la detectada. En el diccionario convencional, esta información que aquí se consigue de modo automático sólo se indica en contadas ocasiones.

El DES proporciona para cada palabra de DIC, el conjunto borroso de sus sinónimos agrupados por acepciones. Es borroso porque cada elemento pertenece al conjunto con un grado: el grado de sinonimia con respecto a la entrada. Dicho conjunto será vacío para las expresiones que no son entrada en el diccionario impreso. Dada una entrada, en el conjunto borroso de sus sinónimos para una acepción se pueden distinguir, como de hecho se hará, entre aquellas palabras que ya figuraban en el diccionario convencional como sinónimas y aquellas que aun no figurando poseen cierto grado de sinonimia al tener conjuntos de sinónimos con elementos en común.

Se puede definir el concepto “figurar como sinónima en el DES” como la relación que se da entre dos expresiones *a* y *b* de DIC cuando dada una acepción de *b*, el grado de sinonimia con alguna de las acepciones de *a* es distinto de 0, para lo cual, sus respectivos conjuntos de sinónimos deben tener algún elemento en común. Las regularidades que se pueden observar ahora con respecto a esta relación son las siguientes:

1. Para toda entrada *a* del diccionario, *a* figura como sinónima de *a* en todas sus acepciones. Además el grado de sinonimia de *a* consigo misma es 1. Por tanto, esta relación es reflexiva. Esto se debe a que en

la base de datos se ha incluido el caso trivial de que toda expresión que figura como entrada es sinónima de sí misma (ya se ha indicado la conveniencia de hacer esto).

2. La simétrica también se cumple. Dadas dos entradas *a* y *b* del diccionario, si una acepción de *a* figura como sinónima de otra de *b* en el DES entonces esa acepción de *b* figura como sinónima de la de *a*. Tal y como está definida la relación, si el conjunto de sinónimos de *a* tiene elementos en común con el conjunto de sinónimos de *b* en el diccionario impreso, entonces el conjunto de sinónimos de *b* tendrá esos mismos elementos en común con el conjunto de sinónimos de *a*. Además, para todas las medidas de similaridad descritas, excepto para la de Tversky, el grado de sinonimia entre dos expresiones *a* y *b* es igual al grado de sinonimia entre *b* y *a*.
3. La transitiva no siempre se cumple en el DES. Del hecho de que, en el diccionario impreso, el conjunto de sinónimos de *a* tenga elementos en común con el conjunto de sinónimos de *b* y este con el conjunto de sinónimos de *c*, no se desprende que el conjunto de sinónimos de *a* tenga elementos en común con el conjunto de sinónimos de *c*. En el diccionario impreso, el conjunto de sinónimos de la única acepción de la palabra "ataviado" es, incluyendo el caso reflexivo, {"ataviado", "apañado", "arreglado", "aderezado", "compuesto", "apuesto", "adornado"}, el de la única acepción de la palabra "aderezado" es

{“aderezado”, “apañado”, “arreglado”, “ataviado”, “compuesto”}, y el de la segunda acepción de la palabra “apañado”, {“apañado”, “arreglado”, “ataviado”, “aderezado”, “compuesto”}. Como se puede observar, los dos primeros conjuntos tienen en común las palabras “ataviado”, “apañado”, “arreglado”, “aderezado” y “compuesto”, con lo cual, la única acepción de la palabra “ataviado” figurará como sinónima de la única acepción de la palabra “aderezado” en el DES; los dos últimos conjuntos tienen en común todas las palabras, con lo que la única acepción de la palabra “aderezado” figurará como sinónima de la segunda acepción de la palabra “apañado” en el DES; el primero y el último tienen en común las palabras “ataviado”, “apañado”, “arreglado”, “aderezado” y “compuesto”, con lo que la única acepción de la palabra “ataviado” figurará como sinónima de la segunda acepción de la palabra “apañado” en el DES. En el ejemplo que se acaba de describir, se da la transitividad, sin embargo, esto no siempre sucede así. El conjunto de sinónimos de la única acepción de la palabra “acoplar” es, incluyendo el caso reflexivo, {“acoplar”, “unir”, “ajustar”, “combinar”, “juntar”, “encajar”}, el de la primera acepción de la palabra “asociar” {“asociar”, “juntar”, “reunir”, “agrupar”, “aliar”, “federar”} y el de la única acepción de la palabra “aliar” {“aliar”, “unirse”, “coligarse”, “confederarse”, “ligarse”}. Como se puede observar, los dos primeros conjuntos tienen en común la palabra

> "juntar", con lo cual, la única acepción de la palabra "acoplar" figurará como sinónima en el DES de la primera acepción de la palabra "asociar"; los dos últimos conjuntos tienen en común la palabra "aliar", con lo que la primera acepción de la palabra "asociar" figurará como sinónima en el DES de la única acepción de la palabra "aliar"; pero el primero y el último no tienen palabras en común con lo que las únicas acepciones de las palabras "acoplar" y "aliar" no figurarán como sinónimas.

En el apartado anterior se afirmaba que la relación "figurar como sinónima en el diccionario" era irreflexiva, aunque ello estaba justificado por motivos de economía en la elaboración del diccionario. La simetría y la transitividad sólo se cumplían en algunas ocasiones. En este apartado se ha observado que la relación "figurar como sinónima en el DES" es reflexiva y simétrica mientras que la transitiva sólo se cumple en algunas ocasiones como ocurría en el diccionario impreso.

Ya se ha indicado los beneficios que reporta el hecho de incorporar el caso trivial de que la propia palabra figure como sinónima de sí misma. Que la primera relación sea irreflexiva y la segunda reflexiva no es excesivamente preocupante ya que el hecho de que se dé una propiedad o la otra depende de la decisión de incorporar o no los casos triviales.

Lo que es más preocupante es que la primera relación no sea simétrica mientras que la segunda sí. Una manera de evitar esto es que el DES sólo calcule

los grados de sinonimia de las palabras que figuren como sinónimas en el diccionario impreso. Esta restricción en el cálculo del grado de sinonimia ha sido practicada en el capítulo anterior, sin embargo, esta práctica hace que el DES pierda la capacidad de detectar sinonimia entre palabras que pueden tener cierto grado de relación, a pesar de que no figuren como sinónimas en el diccionario, por el hecho de poseer conjuntos de sinónimos con elementos en común. El dilema es el siguiente: si se restringe el cálculo de grados de sinonimia a las palabras que figuran como sinónimas en el diccionario impreso, la relación no sería simétrica ni en el diccionario ni en el DES, pero éste pierde la capacidad de detectar expresiones relacionadas que no figuran como sinónimas en dicho diccionario. Por otra parte, si se permite el calculo de palabras relacionadas en el DES, se cumplirá una propiedad en éste que no se da en el diccionario impreso. Esta decisión se deja en manos del usuario, al cual se le presentan separados en dos grupos los sinónimos para una entrada y una acepción de la misma: el grupo de los sinónimos que figuran en el diccionario impreso y el grupo de las expresiones relacionadas cuyos correspondientes conjuntos de sinónimos del diccionario tienen elementos en común con el conjunto de sinónimos de la entrada. A estos dos grupos hay que añadir, como ya se ha indicado, el grupo de aquellas palabras que figuran como sinónimas pero no son entradas del diccionario. Estas palabras, al no ser entradas, no tienen asociado un conjunto de sinónimos, lo que imposibilita el cálculo de su grado de sinonimia con respecto a cualquier palabra del diccionario.

Otra forma de evitar que el diccionario impreso y el DES difieran en la propiedad simétrica es el uso de la *medida de Tversky* con el establecimiento de umbrales. Como ya se había afirmado, dadas dos expresiones *a* y *b*, la *medida de Tversky* hace que en algunos casos el grado de sinonimia entre *a* y *b* sea distinto al grado de sinonimia entre *b* y *a*. Si se fija un umbral entre el grado de sinonimia de *a* y *b* y el grado de sinonimia de *b* y *a*, entonces *a* puede figurar como sinónima de *b* en el DES y *b* no figurar como sinónima de *a* o viceversa, con lo que la simetría no se cumpliría. Por ejemplo, utilizando esta medida, la segunda acepción de la palabra "gracia" figura como sinónima de la primera acepción de "concesión" en grado 0.1111111111, mientras que la primera acepción de "concesión" figura como sinónima de la segunda acepción de "gracia" en grado 0.0909090909. En el DES, el usuario dispone de una barra de desplazamiento con la que puede fijar un umbral entre 0 y 1 con el fin de reducir el número de sinónimos que proporciona como respuesta. Si fijamos el umbral en 0.1, tendríamos que la segunda acepción de la palabra "gracia" figuraría como sinónima de la primera acepción de "concesión", mientras que la primera acepción de "concesión" no figuraría como sinónima de la segunda acepción de "gracia".

Finalmente, conviene insistir una vez más en que el problema de la simetría podría evitarse modificando el diccionario de sinónimos empleado, haciendo figurar como entradas aquellas expresiones que no lo son e incorporando los casos que no satisfacen la propiedad simétrica. Sin embargo,

esta tarea requiere una labor lexicográfica cuidadosa que no está dentro del alcance de este trabajo.

3.- Descripción del diccionario electrónico de sinónimos

A continuación se presentará un diccionario de sinónimos electrónico que realiza el cálculo del grado de sinonimia para cualquier par de entradas del *Diccionario avanzado de sinónimos y antónimos de la lengua española* (Blecua J. M. [1997]). Este cálculo permite, entre otras cosas:

1. La detección automática de la acepción correspondiente a los sinónimos a partir de una entrada.
2. La detección automática de la homógrafa del sinónimo, en el caso de que éste tenga varias palabras homógrafas.
3. El ordenamiento de las expresiones sinónimas según el grado de sinonimia con respecto a la entrada.
4. La detección de palabras que aunque no figuran explícitamente como sinónimas en el diccionario de sinónimos impreso, pueden considerarse relacionadas por el hecho de tener un grado de sinonimia distinto de cero.
5. La reducción del número de respuestas mediante el establecimiento de umbrales por parte del usuario.

Se ha dividido este apartado en tres subapartados: en el primero de ellos se describe el proceso de implementación de la base de datos así como los

predicados empleados para ello. Esta descripción va acompañada de ejemplos, sin embargo, no se ha incluido la base de datos completa debido a su excesivo volumen. En el segundo subapartado se incluye una descripción de los predicados empleados para el cálculo del grado de sinonimia. Muchos de estos predicados ya han sido utilizados en el capítulo anterior de este trabajo, otros son variaciones de los que allí se han descrito. El código, en formato de *Visual Prolog 5.0*, correspondiente a todos los predicados a los que se hará referencia se ha incluido en los apartados primero, segundo y tercero del apéndice de este trabajo. En el tercer subapartado, se describe el interfaz utilizado para el diccionario electrónico. El interfaz consiste en un cuadro de diálogo que se despliega según la cantidad de información que el usuario quiera recibir. El código asociado a este interfaz es también un documento de *Visual Prolog 5.0* que figura en el cuarto apartado del apéndice.

3.1.- La base de datos

Uno de los problemas, ajenos a los fines de esta investigación, aunque no por ello menos importante, es el descrito en Wilks, Y. A., Slator, B. M., Guthrie, L. M. [1996] con respecto al tema de los derechos legales. Ellos señalan la frustración de muchos investigadores al encontrarse con la reticencia de los editores de diccionarios en lo tocante a la concesión de permisos con el fin de utilizar los trabajos de estos para la investigación. Aunque el fin de la herramienta presentada aquí es puramente científico y no comercial, ha sido

imposible obtener, por parte de la editorial, una versión electrónica del diccionario impreso que se utilizará como base de datos, por eso ha tenido que realizarse, combinando técnicas manuales y mecánicas, mediante el uso de un escáner y un OCR. Dicho diccionario es el descrito en el apartado primero de este capítulo y su implementación se ha desarrollado en Prolog siguiendo las fases que a continuación se enuncian:

1. Transcripción a soporte informático de la información que contiene el diccionario. Para ello, se ha utilizado un ordenador *Power Macintoch 4400/200*, con un escáner modelo *Color OneScanner 1200/30* también de *Macintosh*. El OCR empleado fue *OnmiPage Pro* versión 6.0. Los textos devueltos por el escáner fueron convertidos a documentos de *Microsoft Word 97*, realizándose el resto de la labor a partir de este momento en un PC con un procesador *Intel Pentium III* a 600 Mhz y 64 Mb de memoria RAM. Dichos documentos contenían numerosos errores y el código no podía añadirse de forma automática.
2. Corrección de errores *grosso modo* sobre los documentos devueltos por el escáner. Después de esta fase, buena parte del código, aunque no todo, podía añadirse de forma automática.
3. Incorporación, automática en algunos casos y manual en otros, del código Prolog, eliminando toda aquella información contenida en el diccionario que resultaba innecesaria para los propósitos del trabajo a realizar.

4. Detección automática de errores en el código generado. Para lo cual se ha utilizado el depurador de código (*debugger*) del Visual Prolog. El resultado de esta fase es una base de datos en Prolog relativamente extensa (unas 673 páginas aproximadamente), que se puede compilar, pero que todavía contiene errores tipográficos en algunas palabras.
5. Corrección pormenorizada y exhaustiva del código para la eliminación de los errores que todavía persisten después de haber realizado las fases anteriores. Esta fase implica la lectura completa del código y su comparación con la información del diccionario de sinónimos utilizado.
6. Creación de una nueva base de datos a partir de la ya generada, que incorpora información adicional que había sido pasada por alto en la primera versión, como los antónimos de cada entrada, la información sobre sufijos flexivos, cuestiones gramaticales, tecnicismos, dialectalismos, préstamos lingüísticos, cuestiones pragmáticas, variantes diacrónicas, etc.

Durante la labor de implementación de la base de datos se han detectado algunas erratas en el diccionario que han sido subsanadas sobre la marcha en los casos en que la corrección no requería del consejo de un lingüista.

Para la base de datos de los sinónimos se ha empleado el predicado `dic` que consta de cuatro argumentos respectivamente:

```
dic(Entrada,Lista_de_sinónimos,Acepción,Homógrafa).
```

La palabra de entrada, la lista de sinónimos, la acepción y el índice de palabra homógrafa (que es 1 por defecto, en caso que la palabra no tenga homógrafas). Veamos unos cuantos ejemplos:

1. Una palabra con una sola acepción:

```
dic(“ababa”, [“amapola”, “ababol”], “1”, “1”).
```

2. Una palabra con varias acepciones:

```
dic(“abadejo”, [“bacalao”], “1”, “1”).
dic(“abadejo”, [“reyezuelo”], “2”, “1”).
dic(“abadejo”, [“cantárida”], “3”, “1”).
```

3. Dos palabras homógrafas, la primera de las cuales tiene dos acepciones:

```
dic(“cala”, [“perforación”, “taladro”], “1”, “1”).
dic(“cala”, [“supositorio”], “2”, “1”).
dic(“cala”, [“ensenada”], “1”, “2”).
```

Para la base de datos de los antónimos y la información adicional se ha utilizado el predicado `ant`, que consta de cinco argumentos:

`ant(Entrada,Acep,Hom,Lista_información,Lista_antónimos).`

El primero es la palabra de entrada, el segundo la acepción, el tercero el índice de palabra homógrafa, el cuarto la lista con la información adicional sobre sufijos flexivos, cuestiones gramaticales, tecnicismos, dialectalismos, préstamos lingüísticos, cuestiones pragmáticas y variantes diatópicas, y el quinto la lista de antónimos. Algunos ejemplos son los siguientes:

```
ant("ababa", "1", "1", ["s", "f", "p us"], []).

ant("abadejo", "1", "1", ["s", "m", "pez"], []).
ant("abadejo", "2", "1", ["s", "m", "ave"], []).
ant("abadejo", "3", "1", ["s", "m", "insecto"], []).

ant("agudeza", "1", "1", ["s", "f"],["simpleza", "ingenuidad"]).
ant("agudeza", "2", "1", ["s", "f"],[]).
```

Como se puede observar, la información adicional figura en una lista con abreviaturas. Posteriormente se utilizarán una serie de predicados que distribuirán esta información según distintas categorías y convertirán la abreviatura que figura en el diccionario en su palabra correspondiente, como se verá en el próximo apartado.

3.2.- El programa que calcula el grado de sinonimia

El código Prolog correspondiente a los predicados que serán descritos a continuación aparece en el primer apartado del apéndice de este trabajo. Uno de los predicados que figuran en este apéndice es el predicado `sin_dic` que incluye, de forma automática, el caso trivial de que una palabra figure como sinónima de sí misma. Este predicado posee los mismos argumentos que el predicado `dic` utilizado para la base de datos:

```
sin_dic(Entrada,Lista_sinónimos,Acepción,Homógrafa).
```

La diferencia es que, en el predicado `sin_dic`, la primera expresión de la lista de sinónimos que figura como segundo argumento coincide con la expresión correspondiente a la entrada que figura en el primer argumento.

El predicado `sinonimo` calcula el grado de sinonimia entre dos palabras. El procedimiento para el cálculo es análogo al descrito en el capítulo anterior para el mismo menester, pero en este caso, además de detectar la acepción de la palabra que proporciona como respuesta, también detecta el índice

correspondiente a la homógrafa de la respuesta supuesto que ésta tuviese varias homógrafas. El predicado `sinonimo` tiene nueve argumentos:

```
sinonimo(A,AcepA,B,AcepB,GRS,Umb,MS,HomA,HomB).
```

Esto es, dos palabras *A* y *B* con sus correspondientes acepciones *AcepA* y *AcepB*, el grado de sinonimia entre ambas palabras para esas acepciones, el umbral de sinonimia y la medida de similaridad fijados por el usuario y las homógrafas de *A* y de *B*.

El predicado `dic2` no aparecía en el capítulo anterior y es el que devuelve como respuesta las palabras que figuran como sinónimas en el diccionario impreso pero no son entradas del mismo. Dado que no figuran como entradas, no se puede calcular el grado de sinonimia con estas palabras; sin embargo, resulta interesante que el usuario pueda visualizarlas. Este predicado posee cuatro argumentos:

```
dic2(Entrada,Acep,Homógrafa,Sinónimo_que_no_es_entrada).
```

Es decir, la entrada, la acepción y homógrafa que el usuario proporciona y el sinónimo que no figura como entrada correspondiente a esa acepción y homógrafa de la entrada.

El predicado `sinonimo` solamente proporciona la lista de aquellos sinónimos que figuran como tales en el diccionario convencional, pero como ya se ha indicado, hay palabras que, aunque no aparezcan como sinónimas de una dada en el diccionario, pueden estar relacionadas con ella por el hecho de tener sinónimos en común. El predicado `sin_acep2` devuelve como respuesta estas palabras una vez proporcionada una entrada, una acepción y, si es el caso, un

índice de palabra homógrafa. Este predicado posee los mismos argumentos que el predicado `sinonimo`:

```
sin_acep2(A,AcepA,B,AcepB,GRS,Umb,MS,HomA,HomB).
```

En el DES se referirá a las palabras que proporciona el predicado `sin_acep2` con el nombre de "palabras relacionadas" y serán desplegadas en una lista distinta de aquella en la que figuran las palabras que proporciona el predicado `sinonimo`.

El predicado `comparable` es análogo al descrito en el capítulo anterior. La diferencia es que ahora posee un argumento más correspondiente al índice de homógrafa de la palabra del primer argumento:

```
comparable(A,B,AcepA,HomA).
```

El predicado `max` también realiza la misma labor que el descrito en el capítulo anterior. Ahora incluye dos argumentos más con el fin de que calcule, además de la acepción correspondiente al grado máximo, su índice de palabra homógrafa.

```
max(LisHom,HomMax,LisAcep,AcepMax,LisGr,GrMax).
```

Con el predicado `sin_acep` se implementan las medidas de similaridad que el usuario puede seleccionar: *coeficiente jaccard*, *coeficiente del dado*, *coeficiente del coseno*, *coeficiente de similaridad mutua*, *coeficiente de solapamiento* y la *medida de Tversky*. Este predicado también posee los mismos argumentos que el predicado `sinonimo`:

```
sin_acep(A,AcepA,B,AcepB,GRS,Umb,MS,HomA,HomB).
```

Con los predicados `union`, `inter` y `dif` se implementan respectivamente la unión, intersección y diferencia conjuntísticas utilizadas en el cálculo por las medidas de similaridad. El predicado `num` calcula el número de elementos de una lista, `min` el mínimo de dos números y `miembro`, como es usual en Prolog, indica qué elementos son miembros de una lista. Todos ellos han sido presentados en el capítulo primero de este trabajo.

En el segundo apartado del apéndice, aparecen una serie de predicados que realizan una labor que no ha sido descrita anteriormente: consiste en ordenar las cuatro listas de los sinónimos y sus correspondientes acepciones, homógrafas e indicadores de sufijo flexivo según su grado de sinonimia. El predicado `ordenar` realiza esta labor. Posee ocho argumentos correspondientes a las listas desordenadas y ordenadas de sinónimos, acepciones, homógrafas y grados:

```
ordenar(SinD,SinO,AcepD,AcepO,HomD,HomO,GraD,GraO).
```

El predicado `maxi` tiene la misma función que el predicado `max` anteriormente descrito, con la diferencia de que ahora tiene dos argumentos más, correspondientes a la lista de sinónimas y el sinónimo que posee el grado máximo:

```
maxi(LSin,SMax,LHom,HMax,LAcep,AMax,LGr,GrMax).
```

El predicado `bor` borra de una lista el elemento que figura en determinada posición. Sus argumentos son tres, la posición del elemento a borrar, la lista donde figura el elemento y la lista con el elemento eliminado:

```
bor(Posición,Lista_con_elemento,Lista_sin_elemento).
```

El predicado `pos` indica la posición que tiene un elemento en una lista. Sus argumentos son tres; el elemento, la lista donde figura el elemento y la posición del mismo:

```
pos(Elemento,Lista_con_elemento,Posición).
```

Queda por describir el código que figura en el tercer apartado del apéndice. Éste realiza la labor de clasificar la información adicional que aparece en la lista del cuarto argumento del predicado `ant`. En primer lugar, están las siete pequeñas bases de datos que corresponden a las distintas categorías en las que se va a clasificar la información. Como la lista del predicado `ant` contiene, en algunos casos, abreviaturas, estos predicados permitirán el paso de la abreviatura a la expresión completa. Para la creación de estas siete bases de datos se utilizan los siguientes predicados:

```
flexion(Flexión).
gramatical(Abreviatura,Expresión_sin_abreviar).
tecnicismo(Abreviatura,Expresión_sin_abreviar).
dialectalismo(Abreviatura,Expresión_sin_abreviar).
prestamo(Abreviatura,Expresión_sin_abreviar).
pragmatica(Abreviatura,Expresión_sin_abreviar).
diacronico(Abreviatura,Expresión_sin_abreviar).
```

Para cada una de estas categorías, se generaron una serie de predicados que proporcionan la información adicional correspondiente dada una entrada, una acepción y una homógrafa:

```
flex(Entr,Acep,Hom,Flex).
gram(Entr,Acep,Hom,Gram).
tecn(Entr,Acep,Hom,Tecn).
dial(Entr,Acep,Hom,Dial).
prest(Entr,Acep,Hom,Prest).
prag(Entr,Acep,Hom,Prag).
diac(Entr,Acep,Hom,Diac).
```

Finalmente, con el predicado `otros_datos` se clasifica toda aquella información que no ha sido clasificada en ninguna de las categorías anteriores dada una entrada, una acepción y una homógrafa:

```
otros_datos(Entr,Acep,Hom,Item).
```

3.3.- El interfaz

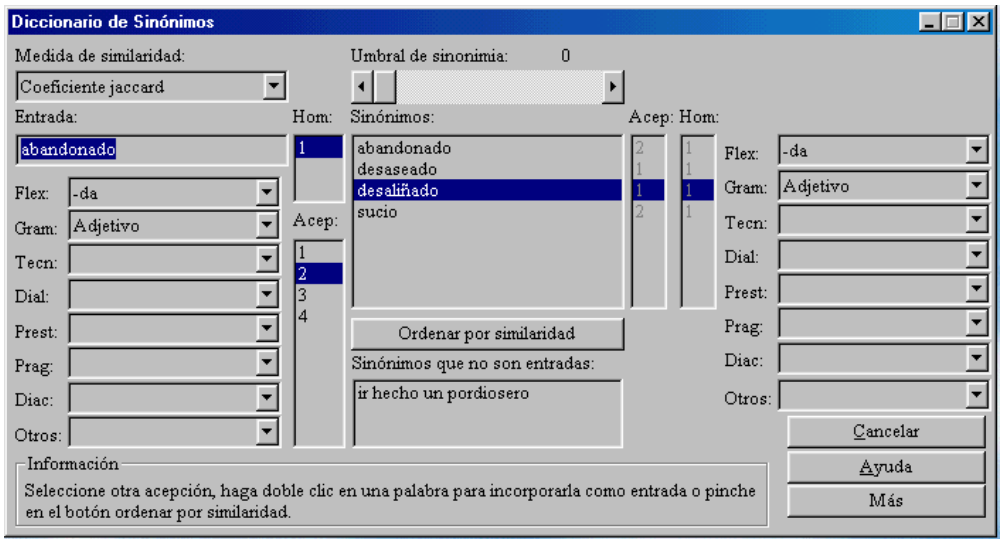
El interfaz visual del DES consiste en un cuadro de diálogo que ofrece más o menos información según los requerimientos del usuario. Habrá dos versiones del cuadro de diálogo:

1. Una versión breve, en la que introduciendo una entrada, seleccionando una homógrafa (si la entrada tiene varias homógrafas) y una acepción, proporciona el conjunto de sinónimos que el diccionario convencional daría, separando los sinónimos que son entradas de los que no lo son. El DES detectará, para cada uno de los sinónimos, excepto para los que no figuran como entrada, la acepción y, si es el caso, la homógrafa correspondientes. Además el usuario puede seleccionar una entre las seis medidas de similaridad definidas, puede fijar un umbral para reducir el número de sinónimos de la respuesta y puede ordenar estos sinónimos de mayor a menor grado de similaridad con respecto a la entrada, obteniendo siempre la información adicional tanto para la entrada introducida como para los sinónimos que figuran como entrada.

2. La versión extendida ofrece las mismas prestaciones que la versión breve proporcionando además una lista de expresiones que no figuran como sinónimas de la entrada en el diccionario convencional pero que están relacionadas con ella por el hecho de que sus respectivos conjuntos de sinónimos tienen elementos en común, detectando, también en este caso, la acepción y, si es el caso, la homógrafa correspondientes a cada una de ellas. Las expresiones relacionadas también se pueden ordenar por similaridad y, seleccionando una expresión cualquiera de las que se proporcionan como respuesta, ya sea sinónima o relacionada, esta versión del cuadro de diálogo proporciona el grado de sinonimia y la verbalización de dicho grado de la expresión seleccionada con respecto a la entrada.

A continuación se describirá de una manera más pormenorizada todos los controles del cuadro de diálogo. El código correspondiente a este cuadro está en formato *Visual Prolog 5.0* y figura en el cuarto apartado del apéndice al final de este trabajo.

La versión breve del cuadro de diálogo figura en pantalla de la siguiente forma:

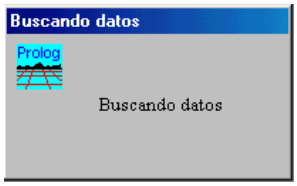


En la parte superior izquierda del cuadro de diálogo aparece una lista desplegable en la que el usuario puede seleccionar una de las medidas de similaridad a las que se ha hecho referencia a lo largo de este trabajo: *coeficiente jaccard*, *coeficiente del dado*, *coeficiente del coseno*, *coeficiente de similaridad mutua*, *coeficiente de solapamiento* y la *medida de Tversky*.

En la parte superior derecha aparece una barra de desplazamiento con la que se puede seleccionar un umbral de sinonimia por debajo del cual no será proporcionada una expresión como sinónima de otra. La utilidad del umbral es la de reducir el número de expresiones que se proporcionan como respuesta, quedándonos con las que poseen un grado de sinonimia mayor respecto a la que figura en la entrada. Un usuario convencional puede no saber que este umbral es un número real entre 0 y 1, incluso puede tener esta información pero no saber como utilizarla. El uso de una barra de desplazamiento hace que el hecho de fijar un umbral se realice de una forma más bien intuitiva, basta con saber que cuanto

más a la izquierda esté situada la barra de desplazamiento, el umbral será más bajo y, por lo tanto, menos restrictivo, con lo que el número de respuestas será mayor. Inversamente, cuanto más a la derecha esté la barra, el umbral será más alto, más restrictivo y, consiguientemente, el número de respuestas será menor. Con todo, en el texto que figura sobre la barra de desplazamiento se indica el valor numérico del umbral fijado. La granularidad de los movimientos de la barra de desplazamiento es de 0.01 en el avance de línea y de 0.1 en el avance de página.

Debajo de la lista de medidas de similaridad figura un control de edición en el que un usuario puede introducir una expresión. En el momento que se introduce la expresión y ésta figura como entrada, el sistema despliega los índices de sus homógrafas en la lista que aparece inmediatamente a su derecha y sus acepciones en la lista que está debajo de ella. En este momento, el usuario debe seleccionar una homógrafa y una acepción. Una vez seleccionada la acepción, aparecerá provisionalmente un pequeño cuadro que indica que el sistema está buscando datos:



Este cuadro desaparece una vez obtenidos los datos. Inmediatamente después, en las tres listas que figuran debajo de la barra de desplazamiento que fija el umbral, se despliegan respectivamente, la lista de las palabras que figuran

en el diccionario como sinónimas de la entrada para la acepción seleccionada, así como las correspondientes acepciones y homógrafas. Si la entrada tiene, para la acepción seleccionada, sinónimos que no figuran como entradas en el diccionario, estos sinónimos se desplegarán en la lista que aparece etiquetada con la expresión "Sinónimos que no son entradas". Además, al seleccionar una de las acepciones, también se proporciona información adicional sobre la palabra de entrada para esa acepción y homógrafa en las listas que figuran debajo del control de edición. Lo mismo sucede en las listas que figuran a la derecha del cuadro de diálogo cuando se selecciona uno de los sinónimos.

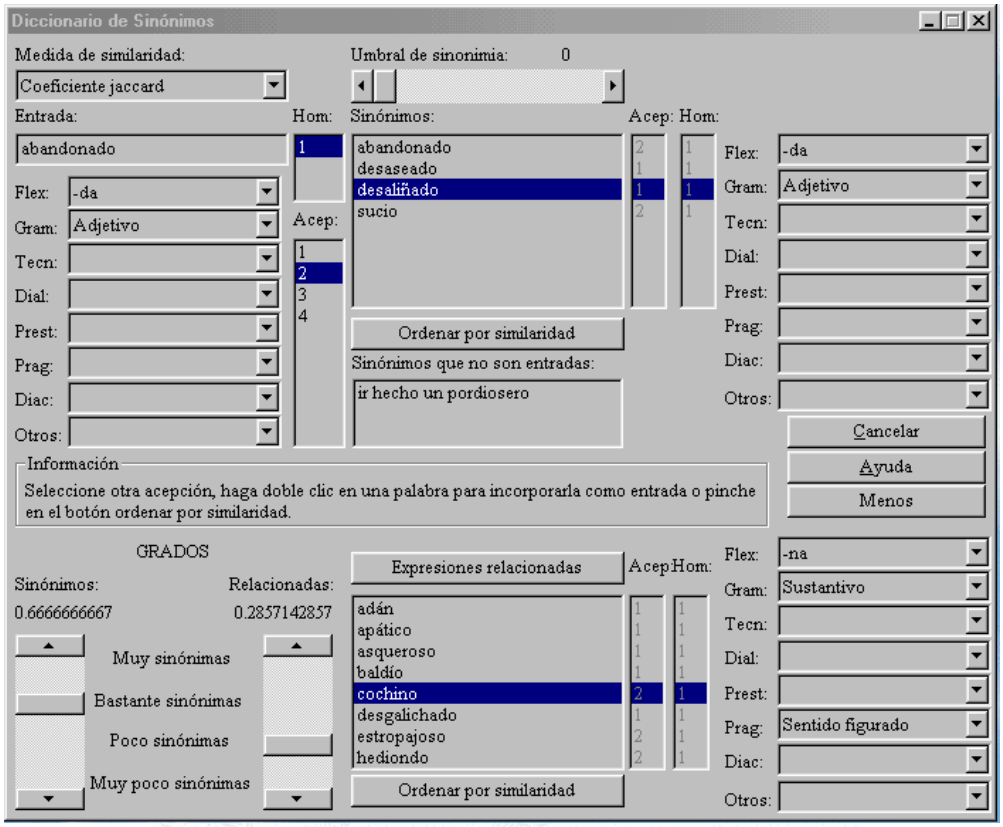
Al hacer doble clic en cualquiera de las expresiones que figuran como sinónimas, ésta pasará a ser entrada, con su índice de homógrafa y la acepción correspondientes. De esta forma el cuadro de diálogo gana en dinamismo.

Debajo de la lista de palabras sinónimas figura un botón etiquetado como "ordenar por similaridad", cuando se presiona este botón, la lista de sinónimos con sus correspondientes acepciones y homógrafas se ordenan de mayor a menor grado de similaridad con respecto a la entrada.

En la esquina inferior izquierda figura un texto que indica al usuario información sobre los pasos a seguir en cada momento.

En la esquina inferior derecha figura un botón con la etiqueta "Más". Haciendo clic en este botón, se pasa de la versión breve a la versión extendida del cuadro de diálogo. En el momento en que se realiza este paso, la etiqueta del botón pasa de "Más" a "Menos". Si ahora se vuelve a pulsar el botón, el sistema

realizará el proceso inverso: se pasará de la versión extendida a la breve y la etiqueta cambiará a “Más”. La versión extendida tiene el siguiente aspecto:



Pulsando el botón etiquetado como “expresiones relacionadas”, aparece de nuevo el cuadro con la información “buscando datos”. Una vez encontrados los datos, el cuadro desaparece y se despliega respectivamente, en las tres listas que figuran debajo del botón, la lista de las expresiones relacionadas, sus acepciones, y homógrafas.

Si se selecciona una de las expresiones relacionadas se obtiene información adicional sobre la misma en las listas que figuran en la esquina inferior derecha del cuadro de diálogo. Seleccionando cualquiera de las expresiones, ya sean sinónimas o relacionadas, aparecerá, en los textos que

figuran en la parte inferior izquierda del cuadro de diálogo, el grado de sinonimia de dicha expresión con respecto a la entrada. La verbalización de ese grado se representará gráficamente con el uso de las barras de desplazamiento que figuran debajo de los respectivos grados. Una vez más, al hacer doble clic en cualquiera de las expresiones relacionadas, ésta pasará a ser entrada, con su índice de homógrafa y la acepción correspondientes.

Debajo de la lista de palabras relacionadas figura un botón etiquetado como "ordenar por similaridad" que opera sobre las listas correspondientes a las palabras relacionadas, de la misma forma que el anteriormente descrito para las sinónimas.

4.- Comparación con WordNet

La herramienta que se acaba de presentar posee ciertas características en común con WordNet: una base de datos léxica que, en sus 15 años de andadura explícita, ha sabido hacerse un lugar de prestigio mostrando su interés en diversos ámbitos. WordNet fue un proyecto originado en la Universidad de Princeton en 1985 tras unos 20 años de trabajos previos cuyo resultado fue la construcción de una gran base de datos electrónica de palabras en lengua inglesa (véase Fellbaum, C. [1998]). Esta base de datos está estructurada en forma de redes semánticas, cada una de las cuales corresponde a la categoría sintáctica de nombre, verbo, adjetivo o adverbio. En el proyecto de Princeton han colaborado numerosas universidades de todo el mundo y un buen número de investigadores

de reconocido prestigio entre los que cabría destacar a G. A. Miller como uno de los principales precursores del proyecto. Además, los resultados han sido aplicados en diversos ámbitos de investigación dentro de la lingüística, el procesamiento del lenguaje natural y la inteligencia artificial.

Una de las primeras coincidencias entre WordNet y la propuesta de este trabajo, que tiene más que ver con las pretensiones de la investigación que con la herramienta propiamente dicha, es algo que queda sintetizado en las palabras de G. A. Miller del prefacio de Fellbaum, C. [1998]: "Siempre hemos considerado WordNet como un experimento, no como un producto". Pero a pesar de tener características en común, los objetivos e intereses de ambas herramientas difieren en buena medida. Mediante la presentación de los parecidos y diferencias con WordNet se establecerán los objetivos, intereses y logros de esta herramienta, así como las ventajas y desventajas que ésta posee frente a aquélla. Sin embargo, no se trata de hacer una comparación, sino un test, ya que WordNet supone un buen punto de referencia para cualquier investigación como la que aquí se propone.

Se dice que hay dos métodos para elaborar un diccionario electrónico: uno mediante la automatización de un diccionario ya editado, tal y como se ha procedido en este trabajo; y otro mediante la elaboración a mano del diccionario, tal y como se ha procedido en la elaboración de WordNet. Ambas tienen sus ventajas y sus inconvenientes. Una ventaja de la primera opción es que es menos laboriosa ya que el procedimiento se realiza de una forma mecánica casi en su totalidad, sobre todo si se cuenta con la versión electrónica del diccionario

impreso. Además, este tipo de procedimientos están avalados por la seriedad y la rigurosidad del trabajo del lexicógrafo que construyó el diccionario editado. El desarrollo actual del escáner y los OCR permite realizar esta labor de una forma relativamente cómoda y sin un coste económico excesivo. Uno de los inconvenientes es que resulta ser un método demasiado dependiente de la versión editada del diccionario, el cual pudo haber sido construido con unos objetivos distintos a los de su versión electrónica. Por su parte, la elaboración manual del diccionario es una tarea excesivamente laboriosa cuya realización sería impensable en el caso de la propuesta presentada aquí, ya que requiere de la intervención de grandes equipos de investigación interdisciplinar (lingüistas, informáticos, etc.). Sin embargo, la ventaja es que la elaboración a mano de las entradas del diccionario puede estar guiada por objetivos próximos a su aplicación y funcionamiento finales, con lo que los resultados de la elaboración pueden ser mejores.

Al igual que WordNet, el DES presentado no divide las palabras en unidades más pequeñas y, a pesar de que en ella figuran palabras compuestas, expresiones complejas, frases hechas, etc., la unidad básica es también la palabra. Sin embargo, no organiza las palabras según distintas relaciones semánticas como hace WordNet, solamente lo hace mediante la relación de sinonimia tal y como figura en el diccionario impreso. Aunque resultaría interesante completar la herramienta propuesta con otras relaciones semánticas, como la antonimia, la hiponimia, la hiperonimia, la meronimia, el entrañamiento, etc., la sinonimia

supone un buen punto de partida ya que goza de una mayor popularidad respecto a las demás relaciones, en el sentido de que cualquier hablante de determinada lengua sabe en qué consiste la sinonimia, algo que es difícil de sostener con las demás relaciones semánticas (con la salvedad, quizás, de la antonimia).

En el DES aparece información relativa a cuestiones gramaticales de las entradas, esta información no figura en WordNet y de esta carencia se han quejado muchos usuarios del mismo, algunos de los cuales han tenido que complementar la información semántica proporcionada por ella con la información de algunas bases de datos sintácticas. A pesar de todo, WordNet consta de cuatro redes semánticas separadas, una para nombres, otra para verbos, otra para adjetivos y otra para adverbios, de tal forma que la información respecto a la categoría sintáctica está, en cierto modo, implícita. Dada esta constitución de WordNet, no se vinculan palabras pertenecientes a distintas categorías sintácticas. Esto sí es posible en el DES que se acaba de presentar, debido a que la base de datos utilizada lo hace en muchas ocasiones. Por ejemplo, se relaciona un nombre como "casa" y un verbo como "alborotar" por el hecho de que éste es sinónimo de la expresión "hacer temblar la casa", que figura asociada al nombre "casa".

En la elaboración del diccionario electrónico presentado aquí, se ha incorporado cierto tipo de conocimiento que figuraba en el diccionario impreso, que no era puramente léxico y al que podría denominarse, en cierto modo, conocimiento enciclopédico. Por ejemplo, para la primera acepción de "abadejo"

se indica que la palabra se refiere al nombre de un pez, para la segunda al de un ave y para la tercera al de un insecto. Bien es verdad que, por ser esta base de datos un diccionario de sinónimos y no una enciclopedia, dicho conocimiento no era demasiado extenso y sólo figuraba en algunas entradas. WordNet no trata explícitamente este tipo de conocimiento, aunque en muchas ocasiones, conocimiento léxico y enciclopédico van mezclados en las definiciones de algunas entradas.

Con respecto a la organización de la información, WordNet es un diccionario semántico que fue diseñado como una red, combinando características de los tesauros y de los diccionarios tradicionales (no diccionarios de sinónimos, sino aquellos que proporcionan definiciones de las entradas). La organización de los conceptos no se realiza por orden alfabético sino por sus propiedades semánticas. En el DES propuesto, la información de la base de datos figura tal y como aparece en el diccionario de sinónimos empleado, es decir, por orden alfabético de las entradas; conservándose también el orden en que figuran las acepciones y las palabras sinónimas, el cual casi nunca es orden alfabético.

Tanto en WordNet como en el DES descrito se emplean conjuntos de sinónimos (llamados *synsets*) para representar el concepto que es expresado por una palabra, de tal forma que, un usuario puede introducir una palabra y obtener un conjunto de sinónimos de la misma. Sin embargo, el interés de WordNet se centra en que la sinonimia no es la única relación a la que hace referencia. Son tratadas de forma explícita otras relaciones como la hiponimia, meronimia,

entrañamiento, antonimia, etc., de tal manera pueden ser seleccionadas por el usuario para orientar su búsqueda. El interés de la herramienta aquí presentada no es el de dar cuenta de otras relaciones, sino el cálculo del grado de sinonimia de dos palabras dentro de un diccionario de sinónimos, información que WordNet no proporciona, como afirman algunos usuarios entre los que se encuentran G. Hirst y D. St-Onge (véase Fellbaum, C. [1998], capítulo 13). Por eso, en cierto modo, esta investigación completa a la llevada a cabo por WordNet. Quizás sería interesante aplicar, para las demás relaciones, un estudio análogo al realizado aquí. Ello conllevaría la búsqueda de un modelo para cada una de ellas con el que poder calcular grados de hiponimia, meronimia, entrañamiento, antonimia, etc. si es que se puede hablar de la gradualidad de tales relaciones.

Otra diferencia con WordNet es que éste contiene definiciones de las palabras como las que aparecen en un diccionario tradicional (no de sinónimos). Además se incluye información sobre palabras que están morfológicamente relacionadas como nombres y adjetivos con la misma raíz, etc.

WordNet hace la distinción entre las relaciones semánticas conceptuales que vinculan conceptos y las relaciones léxicas que vinculan palabras. En el DES la única relación que se trata es la de sinonimia y se hace más bien desde un punto de vista léxico, debido a que se ha tomado como base de datos un diccionario de sinónimos y no un diccionario de conceptos.

En WordNet no se centra la atención en la semántica al nivel de texto o discurso, lo que hace que muchas palabras no se relacionen aun cuando pudieran

estarlo por pertenecer al mismo tema dentro de un discurso. Éste es el caso de las palabras "raqueta", "pelota" y "red" que podrían estar relacionadas por el hecho de referirse a objetos utilizados en determinados juegos deportivos. Tampoco contiene información sobre el contexto en el que aparecen las palabras y las acepciones. El DES contiene este tipo de información siempre que figure en la versión impresa del diccionario, además, aunque el usuario debe desambiguar la palabra que introduce como entrada seleccionando una de sus acepciones, las de las expresiones que se proporcionan como respuesta son detectadas automáticamente; labor que no realiza WordNet. El problema surge cuando se necesita desambiguar automáticamente las palabras (incluidas las entradas). Este tema ha sido tratado en el capítulo tercero cuando se establecían criterios para detectar las acepciones de las palabras que figuran en una oración. Allí se han proporcionado ejemplos que muestran las dificultades que conlleva la aplicación de estos criterios. En el laboratorio de ciencias cognitivas de Princeton se ha afrontado este problema mediante el uso de concordancias semánticas (*semantic concordances*). Una concordancia semántica consiste en la combinación de un lexicón y un corpus textual de tal forma que las palabras de éste se vinculan con las acepciones apropiadas de aquél. Como lexicón utilizaron WordNet y como corpus textual un corpus estándar del inglés americano y una novela. El resultado fue una herramienta que ha resultado útil en desambiguación de palabras (véase Fellbaum, C. [1998], capítulo 8). Este tema es fundamental en el ámbito de la recuperación de información, ya que, tratar de recuperar los documentos

buscados entre un conjunto heterogéneo y grande de ellos requiere que la coincidencia entre las palabras de la pregunta y las de los documentos sea cuanto más exitosa mejor. La desambiguación influye de manera directa en la consecución de este éxito.

Entre 1996 y 1999 se ha desarrollado en Europa el proyecto EuroWordNet, cuyo objetivo es la elaboración de una base de datos con redes de palabras para distintas lenguas europeas como el inglés, holandés, español, italiano, francés, alemán, checo y estonio (véase Vossen, P. [1998]). Cada una de estas redes de palabras está formada por *synsets* (conjuntos de sinónimos) entre los que existen distintas relaciones semánticas (sinonimia, hiponimia, hiperonimia, antonimia, etc.) y están estructuradas de la misma forma que el WordNet para el inglés americano creado por la Universidad de Princeton. La coordinación del proyecto EuroWordNet corrió a cargo de la Universidad de Amsterdam y en él han participado varias universidades de distintos países europeos que han tratado de proporcionar la versión de WordNet en su idioma correspondiente. Cada una de estas versiones están ligadas a un índice interlingüístico basado en la versión de Princeton. Por medio de este índice, cada lenguaje queda intercontectado con cualquiera de los otros. La versión española fue desarrollada por la Fundación Universidad Empresa, en la que intervienen la Universidad Nacional de Educación a Distancia (UNED), la Politécnica de Cataluña y la Universidad de Barcelona.

La versión española de WordNet está construida de tal forma que casi todos los *synsets* de nombres y verbos están conectados con los correspondientes en la versión 1.5 del inglés. Además, la estructura jerárquica del WordNet español está muy próxima, al menos en los niveles más altos, a la estructura del WordNet 1.5. El procedimiento de elaboración de la versión española (véase Rodríguez, H., Climent, S., Vossen, P., Bloksma, L., Peters, W., Alonge, A., Bertagna, F., Roventini, A. [1998]), fue llevado a cabo en dos fases: la primera comienza por la traducción manual de todas las variantes de los nombres que figuraban en los dos niveles más altos del WordNet 1.5. Posteriormente, y mediante procedimientos automáticos se han traducido al español buena parte de las redes de nombres y verbos del WordNet 1.5, lo que conllevaba el refinamiento manual posterior para conseguir los *synsets* correctos del español. Esta primera fase se caracteriza por el uso de procedimientos automáticos para la adquisición de conocimiento léxico y su principal consecuencia es el aumento del volumen de la base de datos del WordNet español prácticamente hasta las dimensiones estimadas para su versión final. En una segunda fase se opta por los procedimientos manuales encaminados no tanto al aumento del volumen de la base de datos, sino más bien a la mejora del solapamiento con las redes de palabras de otros idiomas y al incremento de la calidad de la versión española. Una de las tareas llevadas a cabo fue la detección de huecos en las cadenas de hiponimia. Dichos huecos fueron rellenados manualmente, lo que provocaba el incremento de la compatibilidad con las otras redes de palabras. Para mejorar la

calidad de la versión española se procedió a la verificación manual de *synsets* que contenían variantes generadas automáticamente, la adición de vocabulario nuevo considerado como relevante, con la consiguiente creación de nuevos *synsets*, la adición, mediante procedimientos semi-automáticos y manuales, de relaciones distintas de la sinonimia y la hiponimia en la red semántica de los nombres y entre las redes semánticas de los nombres y los verbos.

La red semántica de los adjetivos fue añadida vinculándola con las redes de nombres y verbos. El primer paso para ello fue el análisis de un diccionario monolingüe español con el fin de extraer información sobre las relaciones adjetivo-adjetivo, adjetivo-nombre y adjetivo-verbo. De este análisis se han extraído una serie de reglas que permitieron la generación automática de adjetivos que posteriormente fueron clasificados de acuerdo con el sufijo con el que habían sido generados. Dado un sufijo, se comprobó manualmente si la relación adjetivo-verbo o adjetivo-nombre era válida para todos los sentidos del nombre o del verbo, seleccionándose aquellos sufijos que cumplían mayoritariamente esta condición y eliminando los adjetivos que eran polisémicos en el diccionario empleado.

Una de las principales diferencias de la versión española de WordNet con respecto al DES de este trabajo tiene que ver con la estrategia de elaboración. Se trata del hecho de haber empleado como base de la investigación un diccionario elaborado para otra lengua como el inglés. Ello supone, como se ha podido comprobar en el proceso de elaboración del WordNet español descrito líneas

atrás, una costosa labor de traducción y adaptación de una lengua a otra que dificulta la fiabilidad del resultado final. El DES presentado carece de esta problemática por el hecho de utilizar como base de datos para el análisis de la sinonimia un diccionario de sinónimos en español elaborado por lexicógrafos cuya competencia lingüística en lengua española es indudable.

El siguiente cuadro resume las coincidencias y diferencias entre WordNet y el DES propuesto aquí:

	WORDNET	**DES**
Se considera como un experimento, no como un producto	**Sí**	**Sí**
Elaboración de la base de datos	**Manual**	**Automática**
La palabra es la unidad básica	**Sí**	**Sí**
Figuran expresiones complejas	**Sí**	**Sí**
Organiza la información según distintas relaciones semánticas	**Sí**	**No, sólo sinonimia**
Figura información sintáctica	**No**	**Sí**
Consta de una única base de datos para nombres, verbos, adjetivos y adverbios	**No**	**Sí**
Vincula palabras de distintas categorías sintácticas	**No**	**Sí**
Incluye conocimiento enciclopédico	**No**	**Sí**
Organización de la información por …	**Propiedades semánticas**	**Orden alfabético**
Se representan conceptos por medio de conjuntos de sinónimos	**Sí**	**Sí**
Contiene información sobre el grado de sinonimia	**No**	**Sí**
Contiene definiciones como las de los diccionarios tradicionales	**Sí**	**No**
Incluye información sobre palabras morfológicamente relacionadas	**Sí**	**No**
Distingue relaciones semánticas y léxicas	**Sí**	**No**
Da cuenta de la semántica a nivel de discurso	**No**	**No**
Contiene información sobre el contexto de uso de las palabras y acepciones	**No**	**Sí**
Detección de las acepciones de las palabras que proporciona como respuesta	**No**	**Sí**

Versión española generada a partir de un diccionario español	**No**	**Sí**

5.- Utilidad del diccionario electrónico de sinónimos

Una pregunta interesante en el ámbito del procesamiento del lenguaje natural con respecto a los diccionarios es si la información que estos proporcionan puede ser de ayuda para el procesamiento y el "entendimiento" de los lenguajes naturales humanos por parte de las computadoras. Muchos investigadores en inteligencia artificial han trabajado en esta empresa de hacer que una máquina "entienda" un texto. Bien es verdad que, en un principio, la mayoría de los sistemas diseñados sólo funcionaban para fragmentos muy pequeños del lenguaje natural. Los intentos de remediar esta situación han apostado, entre otros, por el uso de métodos empíricos, el uso de redes neuronales o el uso de métodos estadísticos para el análisis y "entendimiento" automático de textos, así como también, y esto es lo que resulta interesante con respecto al tema de este trabajo, el incremento del volumen de los lexicones de los sistemas computacionales mediante la extracción del significado a partir de diccionarios electrónicos.

Generalmente se afirma que los diccionarios contienen una gran cantidad de conocimiento sobre el mundo y que la distinción entre conocimiento del lenguaje y conocimiento del mundo es, cuando menos, dudosa. La cuestión es si la información que puede ser extraída de un diccionario, de forma manual por un

humano o de forma automática por un ordenador, resulta interesante para los propósitos de la inteligencia artificial.

El uso de diccionarios electrónicos para la construcción de lexicones computacionales se debe a que aquellos contienen información sobre las palabras que estos necesitan. Si se pueden desarrollar procedimientos automáticos para la extracción y formalización de información léxica sobre corpus lingüísticos grandes, los sistemas de procesamiento del lenguaje natural se verían beneficiados en buena medida. Éste es uno de los motivos por los que se ha propuesto, en diversas ocasiones, la automatización de diccionarios ya publicados, tal y como se ha hecho en este capítulo para un diccionario de sinónimos en lengua española. El motivo de haber seleccionado un diccionario de sinónimos y no un diccionario tradicional es que éste define términos mientras que aquél organiza unas expresiones con respecto a otras. Esto hace a los primeros más atractivos para el procesamiento del lenguaje natural debido a la mayor comodidad de su tratamiento computacional. Sin embargo, la propuesta aquí presentada no sólo consiste en la implementación de una base de datos de palabras de una forma trivial aunque útil por la comodidad de manejo del diccionario electrónico resultante, sino que también permite obtener información extra adquirida mediante procedimientos automáticos no triviales, como por ejemplo, el grado de sinonimia, la detección automática de las acepciones de las expresiones que el sistema da como respuesta, la obtención de palabras relacionadas que no figuran como sinónimas en el diccionario impreso, etc.

Ya se ha indicado que el diccionario de sinónimos empleado como base de datos para la herramienta descrita ha sido implementado sin intervenir en el orden ni en la estructura que poseía en su versión original, excepto en la inclusión del caso trivial de que una palabra figure como sinónima de sí misma. Todo trabajo lexicográfico que contribuya a mejorar la base de datos repercutirá en una mejora del DES propuesto. Sin embargo, el camino inverso también se da, ya que éste puede ser de ayuda para un lexicógrafo por el hecho de proporcionar una herramienta que le permite tener un control de aquellas expresiones que figuran como sinónimas de cualquier entrada del diccionario de sinónimos impreso, de aquellas que figuran como sinónimas pero no son entradas y de aquellas que, aun no figurando como sinónimas, podrían estar relacionadas con la entrada. El lexicógrafo puede decidir incorporar como entradas del diccionario algunas o todas las expresiones que figuran como sinónimas pero no son entradas, o podría incorporar como sinónimas algunas de las expresiones relacionadas.

Hasta el momento se ha indicado cómo un DES como éste es aplicable al campo del procesamiento del lenguaje natural y al campo de la lexicografía. Falta referirse finalmente a su utilidad en el marco de la recuperación de documentos, ámbito de vital importancia desde el surgimiento de Internet, en el que la eficiencia de los buscadores de información tiene una relevancia excepcional.

La recuperación de textos o de documentos, consiste en la localización de aquellos textos, entre un conjunto amplio de ellos, escritos en lenguaje natural,

cuyo contenido satisface la consulta de un usuario. Una consulta es un enunciado que se pretende que coincida con algunas partes del documento; usualmente el título, resumen o palabras clave utilizados para representar el contenido del texto.

Mediante procesos como la denominada *indexación automática*, se consigue la asignación de términos a determinados textos. Estos términos se pretende que expresen el contenido del texto. Lo usual en este proceso es concebir una palabra como una cadena de caracteres entre dos espacios en blanco. Eliminando todas aquellas palabras frecuentes como preposiciones, artículos, pronombres, etc., así como los sufijos de aquellas palabras que los posean, quedando sólo con los temas de las palabras, se obtienen los términos correspondientes al texto. A estos términos se les asocian pesos que reflejan la importancia de cada uno de estos dentro del documento. Lo usual es hacer que este peso sea directamente proporcional al número de veces que el término aparece en el documento e inversamente proporcional al número de documentos en que aparece el término. Cuando un usuario realiza una consulta, el sistema de recuperación de información proporciona una lista de documentos ordenados por similaridad con respecto a la consulta.

Ya en 1964, Sparck Jones (véase Sparck Jones, K. [1986]) indicaba la importancia que tenían los tesauros en el campo de la recuperación de información. Ella misma los calificaba como recursos indexadores para la caracterización de tópicos flexibles y como recursos de búsqueda dirigidos a la

mecanización a través de coincidencia completa o parcial de especificaciones de términos.

Para comprobar la efectividad de un sistema de recuperación de documentos se suele utilizar un conjunto de textos y una serie de preguntas a cada una de las cuales se le asocia la lista de documentos que son relevantes para ella, es decir, la lista de aquellos documentos que el sistema debería recuperar. Existen diversas maneras de medir la efectividad de un sistema de recuperación de información. La más común es la de controlar la *precisión* y la *cobertura* (*recall*) del sistema. La precisión consiste en la proporción de los documentos que son relevantes entre aquellos que son recuperados. La cobertura es la proporción de documentos que son recuperados entre aquellos que son relevantes. La polisemia entre palabras, es decir, el hecho de que una palabra pueda tener varios significados o acepciones, hace disminuir la precisión del sistema ya que se pueden realizar coincidencias erróneas entre los términos de la consulta y los términos de los documentos. Esto trae como consecuencia que se recuperen documentos que contienen determinadas palabras de la consulta pero en los que se utilizan dichas palabras en una acepción distinta a la empleada en la consulta. Por su parte, la sinonimia hace disminuir la cobertura, ya que pueden no realizarse coincidencias que se deberían haber realizado entre los términos de la pregunta y los términos de algún documento. Esto trae como consecuencia que no se recuperen documentos que contienen palabras que, aunque no figuran en la pregunta, son sinónimas de las de la pregunta.

Un DES como el presentado aquí, puede contribuir a una mejora en la eficiencia del sistema de recuperación de información al aumentar la cobertura mediante la ampliación de la consulta con los sinónimos de las palabras que figuran en ella. Este procedimiento es conocido como *expansión de la pregunta* (*query expansion*) y generalmente tiene como consecuencia la recuperación de un mayor número de documentos, algunos de los cuales serán relevantes. El problema de este procedimiento es que así como hace aumentar la cobertura puede hacer disminuir la precisión debido a las siguientes razones:

1. Dado que en el DES presentado se considera el caso de que las palabras pueden tener varias acepciones, se recuperarán documentos que posean palabras sinónimas de las de la consulta pero en acepciones distintas, disminuyendo así la precisión del sistema. Al comienzo de este capítulo se ha comentado la problemática de fijar las acepciones de las palabras cuando aparecen en una oración. El empleo de criterios del estilo de los presentados en el tercer capítulo tienen un coste excesivamente alto y una escasa fiabilidad. Una posibilidad factible es la de instar al usuario que realiza la consulta a seleccionar manualmente las acepciones concretas de las palabras polisémicas que utiliza en la misma de modo análogo a como sucede cuando un usuario del DES selecciona una acepción de la palabra que proporciona como entrada. Aunque esto no es una solución al problema, constituye, por el momento, una estrategia eficiente para evitar que aparezca.

2. Debido al carácter gradual de la sinonimia, otro de los problemas que surgen es que los sinónimos con los que se expande la pregunta pueden no significar exactamente lo mismo que las palabras utilizadas en la pregunta, con lo que, una vez más, disminuirá la precisión al recuperarse documentos con términos que no significan exactamente lo que el usuario busca. El DES presentado aquí, tiene una ventaja al respecto. El hecho de proporcionar información sobre el grado de sinonimia entre las expresiones y posibilitar el establecimiento de un umbral de sinonimia puede permitir el ajuste adecuado en la expansión de la pregunta, de tal forma que ésta se expanda con aquellos sinónimos cuyo grado de sinonimia no supere determinado umbral por debajo del cual, la precisión disminuye en exceso, lo que va en detrimento de la eficiencia del sistema de recuperación. De esta forma, la cobertura aumentará por el hecho de expandir la pregunta y el establecimiento de un umbral de sinonimia hará que la precisión disminuya de una forma controlada con el fin de evitar un descenso excesivo de la eficiencia.

CONCLUSIONES

El objetivo de este trabajo fue el tratamiento automático de la sinonimia con Prolog y estuvo caracterizado por dos enfoques: el interlingüístico, que nos situó ante el problema de la traducción y el intralingüístico, en el que se trató la sinonimia entre palabras y entre oraciones. Para ello se comenzó presentando el lenguaje de programación Prolog describiendo la parte de la lógica de primer orden que subyace a éste y las estrategias comunes de la programación en dicho lenguaje, entre las que se puede destacar el procesamiento de listas y gramáticas, aspectos que ponen de relieve su adecuación para el procesamiento del lenguaje natural. El paso siguiente fue realizar un recorrido por distintos modos en cómo fue tratada históricamente la sinonimia. Entre las diversas concepciones de la sinonimia podían vislumbrarse, de modo general, dos tendencias: la que persigue su definición y la que trata de describirla tal y como es usada en la práctica

lingüística. La primera ha resultado interesante en ámbitos como el de la filosofía analítica, pero en general resulta deficiente si se ejemplifica con fragmentos del lenguaje natural en los que no se utilizan términos propios de las ciencias exactas. La segunda está más próxima a la propuesta de este trabajo. La ventaja de dicha concepción es que no se ve afectada por los problemas que la definición tiene a la hora de ser aplicada en la práctica; el inconveniente es que se corre el riesgo de quedar en una mera descripción de un fenómeno sin que ello contribuya a tener un mejor conocimiento teórico del mismo.

La sinonimia concebida desde un punto de vista interlingüístico podría caracterizarse de igual modo que la traducción diciendo que dos expresiones de dos lenguajes diferentes son sinónimas si su significado es el mismo. Esto no constituye una definición de la traducción, dado que se recurre a un concepto como el de significado cuya caracterización no está clara. A pesar de todo, éste concepto ha sido tratado, en el presente contexto, como usualmente se hace en el ámbito del Prolog. Para ello se ha recurrido al procesamiento de gramáticas. Una gramática puede asociar, a cada una de las oraciones que genera, un significado, valor semántico o estructura semántica (cualquiera de los tres fueron términos empleados en el trabajo para denotar lo mismo) en función de los valores semánticos asociados a cada uno de los componentes de dichas oraciones. Construyendo gramáticas que asocien el mismo significado para expresiones de distintos idiomas, se pueden traducir las oraciones correctas de cada uno de ellos en los demás. La dificultad radica en la elaboración de estas gramáticas, ya que la

complejidad y el carácter impreciso de las reglas del lenguaje natural convierten esta labor en una tarea impracticable si se pretende aplicar de un modo general. Sin embargo, la idea resulta útil si se restringe el alcance a contextos concretos de comunicación persona-máquina, como en el caso del horario de trenes presentado, donde la gramática constituye un interfaz entre un usuario que se comunica en lenguaje natural y la sintaxis propia de las consultas en Prolog.

Respecto a la sinonimia intralingüística entre palabras, se obtuvo un procedimiento para el cálculo del grado de sinonimia de las entradas de un diccionario de sinónimos. La ventaja de esta propuesta es que, a diferencia de los demás intentos de este trabajo, resulta generalizable al diccionario completo sin que ello suponga un coste excesivo. Esto posibilitó el diseño del diccionario electrónico presentado en el último capítulo.

Aunque no fue el propósito de este trabajo facilitar una definición de sinonimia, basándonos en el recorrido realizado por sus distintas concepciones, se podrían indicar diversos criterios que subyacen a la propuesta establecida aquí. Estos no proporcionan una caracterización completamente fiel de la sinonimia, pero son capaces de marcar indicios de su existencia en la práctica. En el trabajo se han manejado algunos de estos criterios, como por ejemplo:

1. Dos expresiones son sinónimas si tienen el mismo o parecido significado.
2. Dos expresiones son sinónimas si son utilizadas como sinónimas.

3. Dos expresiones son sinónimas si figuran como tales en un diccionario de sinónimos.

El primero de ellos apela al concepto de significado, y éste es, cuando menos, tan poco claro como el de sinonimia. Sin embargo, ha sido empleado, en cierto modo, en el tratamiento de la sinonimia entre palabras de este trabajo tras considerar al conjunto de sinónimos de una expresión como una forma de representar el significado de ésta. Ello permitió comparar los significados de dos expresiones según su parecido o similaridad dando lugar al cálculo del grado de sinonimia llevado a cabo por el diccionario electrónico de sinónimos recién descrito. Aunque concebir el significado como el conjunto de sinónimos de una expresión no constituye una definición del mismo, sí es una forma computacionalmente factible de representarlo.

Pareció plausible pensar que un diccionario de sinónimos puede constituir una muestra razonablemente fiel del uso de la sinonimia en la práctica. En ese caso, los criterios segundo y tercero darían cuenta casi de la misma idea y estarían muy próximos al primero si se utiliza, como se acaba de señalar, el conjunto de sinónimos de una expresión para representar su significado, ya que el diccionario proporcionará tal conjunto.

Conscientes de la problemática que conlleva su definición, se ha perseguido la descripción del comportamiento de la sinonimia dentro un diccionario de sinónimos concreto. Esto reflejó, por una parte, su carácter gradual, conllevando que la similaridad fuese un modelo más apropiado para

tratarla que la equivalencia, alternativa que históricamente se había pensado como adecuada para este propósito en algunos ámbitos. Por otra parte, también reveló su carácter contextual, que se manifiesta en dos aspectos:

1. El contexto puede influir en el umbral mínimo exigido para que dos expresiones sean consideradas como sinónimas. Esto tiene como consecuencia que existan contextos donde determinadas expresiones se utilicen como sinónimas y otros donde esas mismas expresiones no se utilicen como tales.
2. El contexto puede contribuir a detectar los significados correctos de las expresiones cuando éstas son polisémicas. Esto supone que un par de expresiones pueden ser sinónimas si se emplean los significados que las hacen sinónimas y ese mismo par no serlo si no se emplean tales significados cuando al menos uno de sus elementos es polisémico.

El tratamiento de la sinonimia intralingüística entre oraciones fue llevado a cabo en este trabajo desde dos puntos de vista: el gradual y el preciso. Con respecto al primero, los criterios seguidos podrían establecerse de la siguiente forma:

1. Dos expresiones son sinónimas si son intersustituibles en una oración sin que cambie su valor de verdad, su significado, etc.
2. Dos oraciones son sinónimas si sus palabras coinciden o son sinónimas respectivamente.

3. Dos oraciones son sinónimas si una de ellas es el resultado de sustituir en la otra algunas palabras por sus sinónimas.

A pesar de que se ha indicado que la intersustituibilidad no siempre es condición suficiente ni condición necesaria para la existencia de sinonimia, el primero de los criterios ha sido empleado de todas formas debido a que muestra una de las principales características del uso de un diccionario de sinónimos: la sustitución de unas palabras por otras dentro de una oración. No se planteó el problema de cómo afecta la sustitución al valor de verdad de la oración o a su significado, sino que el procedimiento consistía en dar cuenta de cómo influye en el grado de sinonimia de ambas oraciones el hecho de intercambiar unas palabras por otras que poseen cierto grado de sinonimia respecto a las sustituidas. Sin embargo, es intuitivamente plausible que si se concibe la sinonimia como una cuestión gradual, la sustitución puede establecerse entre dos expresiones cuyo significado, aunque parecido, no sea exactamente el mismo, lo que debería alterar, en cierto modo, el valor de verdad o el significado de la oración completa.

El caso de la sustituibilidad condujo al análisis de otra cuestión que tiene su origen en el carácter polisémico de las palabras: el problema de la desambiguación. Aunque una palabra puede tener varios significados, lo usual es que se esté utilizando solamente uno de estos cuando es empleada en una oración. Averiguar cuál de ellos se está empleando, resulta fundamental para poder sustituir dicha palabra por sus sinónimos, ya que estos variarán según el

significado utilizado. Se ha podido comprobar que el tratamiento computacional del problema de la desambiguación resulta excesivamente complejo. Detectar la acepción que se está empleando de determinada palabra en el contexto oracional es una labor que los humanos realizan generalmente con cierta solvencia y eficacia, sin embargo, una máquina necesitaría mucha cantidad de información relativa al gran número de excepciones que poseen las reglas semánticas del lenguaje y aun así no se podría tener la completa seguridad de que la acepción detectada fuese la correcta.

Respecto al segundo punto de vista para la sinonimia entre oraciones, el que la concibe de un modo preciso, se ha analizado el tema de las oraciones en pasiva y de las que contienen términos inversos. Su tratamiento computacional ha requerido, una vez más, el empleo del procesamiento de gramáticas y el manejo del significado con Prolog. El método empleado es análogo al de la traducción, con la diferencia de que, en este caso, las gramáticas diseñadas son relativas a un único idioma. Como también acontecía allí, la generalización de los resultados es dificultosa y sólo resulta interesante si se restringe a contextos concretos.

Pero sin duda la principal aportación del trabajo la constituye el diccionario electrónico de sinónimos descrito en el último capítulo. Este diccionario ha sido implementado en Visual Prolog y consta de tres partes:

1. La base de datos, que es un diccionario de sinónimos convencional.

2. El programa que calcula el grado, basado en el tratamiento de la sinonimia entre palabras propuesto.
3. El interfaz, que es un cuadro de diálogo donde se emplean las técnicas comunes en programación visual.

El cálculo del grado de sinonimia proporciona al diccionario electrónico unas características que mejoran sus prestaciones con respecto a los diccionarios de sinónimos convencionales, como por ejemplo:

1. La detección automática de las acepciones de los sinónimos obtenidos como respuesta.
2. La detección automática de los índices de homógrafa de las palabras proporcionadas como respuesta.
3. El ordenamiento de los sinónimos de mayor a menor grado de sinonimia respecto a la entrada.
4. La ampliación del conjunto de sinónimos con aquellos que, aun no figurando como tales en la base de datos, podrían serlo por el hecho de poseer un grado de sinonimia distinto de cero.
5. La reducción de las palabras de la respuesta mediante el establecimiento de umbrales de sinonimia.

Para el diseño del diccionario electrónico se partió de la descripción del comportamiento de la sinonimia en el diccionario de sinónimos utilizado como base de datos. Ello mostró que la relación de sinonimia no se comporta de un modo regular; satisface la propiedad irreflexiva pero no satisface las propiedades

simétrica ni transitiva. Se ha tomado la decisión desde un principio de implementar la base de datos tal y como aparece en su versión publicada sin realizar ningún tipo de manipulación a no ser que ello supusiese una mejora que afectase de modo genérico a todo el resultado. Un ejemplo de esto tuvo lugar cuando se decidió considerar a una entrada como sinónima de sí misma, algo que, como se ha dicho en su momento, fue beneficioso para la eficiencia del algoritmo que calcula el grado.

Por otra parte, en el diccionario electrónico de sinónimos la sinonimia se comporta como una relación reflexiva, simétrica y no transitiva. El hecho de que no coincida en la propiedad reflexiva respecto a la base de datos no parece especialmente relevante: que en ésta no se incluya el caso de que una expresión sea sinónima de sí misma se debe a cuestiones de economía a la hora de elaborar el diccionario. Que una palabra figure como sinónima de sí misma en un diccionario de sinónimos no resulta informativo y por eso no se incluye. Lo que parece más relevante es la falta de coincidencia en la propiedad simétrica. Se han analizado algunos casos en la base de datos como las palabras a las que se incorporan sufijos, la existencia de sinónimos que no figuran como entrada o el hecho de que el diccionario dé cuenta de otras relaciones semánticas como la hiponimia o la hiperonimia. Todas ellas son cuestiones que provocan que la relación entre las expresiones del diccionario no satisfaga la propiedad simétrica. Se han apuntado dos modos de tratar este problema. Uno de ellos consiste en la modificación de la base de datos con el fin de incorporar los casos simétricos; sin

embargo, se ha indicado que esta tarea requiere cierta labor lexicográfica que está fuera del alcance de este trabajo. El otro consiste en utilizar medidas de similaridad que no satisfagan la propiedad simétrica como la de Tversky, el problema es que ésta requiere el establecimiento de pesos numéricos que, sin una investigación más exhaustiva, parecen no poder ser fijados más que de un modo arbitrario.

Con respecto a los usuarios potenciales del diccionario electrónico, éste no ha sido pensado para un público concreto. Se puede decir que contribuye, en cierto modo, a mejorar el diccionario impreso, por lo que un lexicógrafo podría utilizarlo como herramienta de trabajo. En el contexto de la recuperación de información, las ventajas que han sido indicadas respecto al uso de umbrales para expandir la pregunta de una forma controlada evitando una pérdida excesiva de precisión, convierte en usuarios potenciales a los que trabajan en este campo. Por otra parte, cualquier usuario del diccionario impreso puede serlo del electrónico. Quizás en este caso la información explícita sobre el grado de sinonimia de dos palabras sea poco informativa y la selección de una medida de similaridad puede llevar a confusión cuando el usuario no tiene siquiera una idea intuitiva de cómo funcionan y para qué sirven dichas medidas en el diccionario. Quizás una propuesta adecuada sería la de diseñar distintos interfaces según el perfil del usuario, de tal forma que a unos se les oculte determinada información que puede resultar innecesaria u oscura.

A todo este trabajo subyace la idea de que el tratamiento automático de cuestiones relativas al lenguaje natural es una tarea de gran complejidad en la que no se pueden establecer reglas excesivamente estrictas y de la que no cabe esperar la ausencia de problemas ante cualquier resultado. En ningún momento se ha pretendido afirmar que el diccionario electrónico de sinónimos recién presentado realice una descripción infalible del fenómeno de la sinonimia, simplemente constituye una herramienta relativamente fiable que podría ser útil en campos como el del procesamiento del lenguaje natural, la lexicografía o la recuperación de información y que entre sus usuarios podrían verse beneficiados los lexicógrafos, informáticos, teóricos de la sinonimia y usuarios de la lengua en general. Se ha procurado ser lo más cuidadoso posible respecto a la fiabilidad de los resultados siempre que la complejidad del lenguaje lo permitiera. Toda la problemática encontrada en el camino que se acaba de recorrer no hace más que poner de manifiesto que las máquinas todavía no son capaces de superar a los humanos en su capacidad de uso del lenguaje natural, lo que significa que aún hay muchos problemas por resolver en el campo apasionante de su procesamiento automático.

REFERENCIAS

Ajdukiewicz, K. [1931] 'On the meaning of expressions' en Giedymin, J. [1978] (ed.) *Kazimierz Ajdukiewicz: The scientific world-perspective and other essays 1931-1963*, Reidel.

Ajdukiewicz, K. [1934] 'Language and meaning' en Giedymin, J. [1978] (ed.) *Kazimierz Ajdukiewicz: The scientific world-perspective and other essays 1931-1963*, Reidel.

Aristóteles *Categorías*. (Traducción de Valdés Villanueva, L. [1983], Cuadernos Teorema).

Aristóteles *Retórica*. (Traducción de Racionero, Q. [1990], Gredos).

Austin, J. L. [1961] *Philosophical Papers*, Oxford University Press.

Bergman, G. [1964] *Logic and Reality*, University of Wisconssin Press.

Bréal, M. [1924] *Essai de sémantique. Science des significations*, Hachette.

Carnap, R. [1947-56] *Meaning and Necessity: A Study in Semantics and Modal Logic*, University of Chicago Press.

Carnap, R. [1955] 'Meaning and synonymy in natural languages' en *Philosophical Studies*, Nº 7, pp. 33-47 (incorporado también en la segunda edición de Carnap, R. [1947-56]).

Cases, B. [1996] 'From Synonymy to Self-Modifying Automata: Q-Diam Language' en Dassow, J., Rozenberg, G., Salomaa, A. (eds.) *Developments in Language Theory II: At the Crossroads of Mathematics, Computer Science and Biology*, World Scientific.

Blecua J. M. [1997] (dir.) *Diccionario avanzado de sinónimos y antónimos de la lengua española*, Bibliograf.

Fellbaum, C. [1998] (ed.) *WordNet: An Electronic Lexical Database*, MIT Press.

Foxley, E., Gwei, G. M. [1989] 'Synonymy and Contextual Disambiguation of Words' en *International Journal of Lexicography*, Vol. 2, Nº 2, pp. 111-131.

Goodman, N. [1949] 'On Likeness of Meaning' en *Analysis*, Vol. 10 (1), pp. 61-70.

Lewis, C. I. [1943] 'The Modes of Meaning' en *Philosophy and Phenomenological Research*, Vol. 4 (2), pp. 236-250.

Liao, T. W., Zhang Z. M., Mount, C. R. [1998] 'Similarity Measures for Retrieval in Case-Based Reasoning Systems' en *Applied Artificial Intelligence*, Vol. 12, pp. 267-288.

Lyons, J. [1995] *Linguistic Semantics. An Introduction*, Cambridge University Press.

Masterman, M. [1957] 'The Thesaurus in Syntax and Semantics' en *Mechanical Translation*, Vol. 4, pp. 1-2.

Mates, B. [1950] 'Synonymity' en *University of California Publications in Philosophy*, Vol. 25.

Naess, A. [1949] 'Toward a Theory of Interpretation and Preciseness' en *Theoria*, Vol. 15.

Peregrin, J. [1998] 'Linguistics and Philosophy' en *Theoretical Linguistics*, Vol. 24, Nº 2-3, pp. 245-264.

Quine, W. O. [1951] 'Two Dogmas of Empiricism' en *The Philosophical Review*. (Incluido en Quine, W. O. [1953])

Quine, W. O. [1953] *From a Logical Point of View*, Harvard University Press.

Quine, W. O. [1960] *Word and Object*, MIT Press.

Rodríguez, H., Climent, S., Vossen, P., Bloksma, L., Peters, W., Alonge, A., Bertagna, F., Roventini, A. [1998] 'The Top-Down Strategy for Building EuroWordNet: Vocabulary Coverage, Base Concepts and Top Ontology' en *Computers and the Humanities*, Vol. 32, pp. 117-152.

Rorty, R. [1967] *The Linguistic Turn: Recent Essays in Philosophical Method*, University of Chicago Press.

Shöning, U. [1989] *Logic for Computer Scientists*, Birkhäuser.

Sparck Jones, K. [1986] *Synonymy and Semantic Classification*, Edinbürg University Press.

Trillas, E., Alsina, C., Terricabras, J. M. [1995] *Introducción a la lógica borrosa*, Ariel.

Tversky, A. [1977] 'Features of Similarity' en *Psychological Review*, Vol. 84, Nº 4, pp. 327-352.

Ullmann, S. [1962] *Semantics: An Introduction to the Science of Meaning*, Oxford University Press.

Vossen, P. [1998] (ed.) *EuroWordNet: A Multilingual Database with Lexical Semantic Networks*, Kluwer A. P.

White, M. G. [1950] 'The Analytic and the Synthetic: an Untenable Dualism' en Hook, S. (ed.) *John Dewey: Philosopher of Science and Freedom*, The Dial Press, pp. 316-330.

Wilks, Y. A., Slator, B. M., Guthrie, L. M. [1996] *Electric Words: Dictionaries, Computers, and Meanings*, MIT Press.

Zadeh, L. A. [1965] 'Fuzzy sets', *Information and Control*, Vol. 8, pp. 338-353.

Bibliografía Adicional

Bezdek, J. C., Biswas, G., Huang, L. [1986] 'Transitive closures of fuzzy thesauri for information-retrieval systems', *International Journal Man-Machine Studies*, Vol. 25, pp. 343-356.

Bratko, I. [1990] *Prolog Programming for Artificial Intelligence*, Addison Wesley.

Burke, E., Foxley, E. [1996] *Logic and its Applications*, Prentice Hall.

Carnap, R. [1949] 'A reply to Leonard Linsky', *Philosophy of Science*, Vol. 16, Nº 4, pp. 347-350.

Carnap, R. [1950] *Logical Foundations of Probability*, University of Chicago Press.

Chaffin, R., Glass, A. [1990] 'A comparison of hyponym and synonym decisions', *Journal of Psycholinguistic Research*, Vol. 19, Nº 4, pp. 265-280.

Chierchia, G., McConnell-Ginet [1990] *Meaning and grammar. An Introduction to Semantics*, MIT Press.

Chugur, I., Peñas, A., Gonzalo, J., Verdejo, M. F. [2000] 'Incorporación de adjetivos al WordNet español' en *Revista de la Sociedad Española para el Procesamiento del Lenguaje Natural (SEPLN)*, Nº 24.

Covington, M. A. [1994] *Natural Language Processing for Prolog Programmers*, Prentice-Hall.

Cruse, D. A. [1997] *Lexical Semantics*, Cambridge University Press.

De Bouvère, K. L. [1965] 'Logical synonymity', *Indagatione Mathematicae*, Vol. 27, pp. 622-627.

Espasa-Calpe [1997] (ed.) *Diccionario Espasa sinónimos y antónimos*, Espasa-Calpe.

Fernández Lanza, S. [2000] 'La sinonimia como relación precisa en el procesamiento del lenguaje natural', *Actas del III Congreso de la Sociedad de Lógica, Metodología y Filosofía de la Ciencia en España*, pp. 135-146.

Fernández Lanza, S. [2000] 'Prolog for automatic processing of synonymy', *Logica Trianguli*, Nº 4, 2000.

Fernández Lanza, S., Sobrino, A. [1998] 'Análisis del grado de sinonimia en un diccionario de sinónimos', *Actas del VIII Congreso Español sobre Tecnologías y Lógica Fuzzy*, pp.117 - 124.

Fernández Lanza, S., Sobrino, A. [1999] 'Algunos aspectos filosóficos y computacionales de la sinonimia', *Actas del Congreso Internacional la Filosofía Analítica en el Cambio de Milenio*, pp. 469-482.

Fernández Lanza, S., Sobrino, A. [2000] 'Hacia un tratamiento computacional de la sinonimia', *Revista de la Sociedad Española para el Procesamiento del Lenguaje Natural*, Nº 24, pp. 89-95.

García-Hernández, B. [1997] 'Sinonimia y diferencia de significado' en *Revista española de lingüística*, Vol. 27-1, pp. 1-31.

García-Hernández, B. [1997] 'Sinonimia, relación onomasiológica en la antesala de la semántica' en *Revista española de lingüística*, Vol. 27-2, pp. 381-407.

Gili Gaya, S. [1973-75] *Diccionario de sinónimos*, Bibliograf.

González Martínez, J. M. [1988-89] 'La sinonimia. Problema metalingüístico', *Anales de filología hispánica*, Vol. 4, pp. 193-210

González Pérez, R. [1994] 'Sinonimia y teoría semántica en diccionarios de sinónimos de los siglos XVIII y XIX', *Revista española de lingüística*, Vol. 24, Nº 1, pp. 39-48.

Gonzalo, J., Verdejo, F., Chugur, I. [1999] 'Using EuroWordNet in a Concept-Based Approach to Cross-Language Text Retrieval', *Applied Artificial Intelligence*, Vol. 13, pp. 647-678.

Green, S. J. [1998] 'Automated link generation: Can we do better than term Repetition?', *Computer Networks and ISDN Systems*, Vol. 30, pp. 75-84.

Gwei, G. M., Foxley, E. [1987] 'A flexible synonym interface with application examples in CAL and help environments', *The Computer Journal*, Vol. 30, Nº 6, pp. 551-557.

Hernández, H. [1994] (coord.) *Aspectos de lexicografía contemporánea*, Vox.

Herrmann, D. J. [1978] 'An old problem for the new psychosemantics: Synonymity', *Psychological Bulletin*, Vol. 85, Nº 3, pp. 490-512.

Iwańska, £. M., Shapiro, S. C. [2000] *Natural Language Processing and Kowledge Representation*, MIT Press.

Lewandowska, B., Tomaszczyk, J. [1990] 'Meaning, Synonymy, and dictionary', en Lewandowska, B., Tomaszczyk, J. (ed.) *Meaning and Lexicography*, Benjamin.

Linsky, L. [1952] *Semantics and the Philosophy of Language*, University of Illinois Press.

Lopez de la Huerta, J. [1799] *Examen de la posibilidad de fixar la significación de los sinónimos de la lengua castellana*, Imprenta Real.

López de Mántaras, R., Trillas, E. [1984] 'Towards a measure of the degree of synonymy', en Sánchez, E. (ed.) *Fuzzy Information, Knowledge Representation and Decision Analysis*, Pergamon Press.

Martínez Hernández, M. [1984] 'Para una historia de los diccionarios de sinónimos del griego antiguo', en Bernabé, A., de Cuenca, L. A., Gangutia, E., López Facal, J. (eds.) *Athlon. Satura Grammatica in Honorem Francisci R. Andrados*, Gredos.

Miller, J. [1989] 'Participant roles, synonymy, and truth conditions', en Arnold, D., Atkinson, M., Durand, J., Grover, C., Sadler, L. (eds) *Essays on Grammatical Theory and Universal Grammar*, Clarendon Press.

Moro Simpson, T. (comp.) [1973] *Semántica filosófica: Problemas y discusiones*, Siglo XXI.

Murphy, G. L., Andrew, J. M. [1993] 'The conceptual basis of antonymy and synonymy adjectives', *Journal of memory and language*, Vol. 32, pp. 301-319.

Neus Vila, M. y otros (eds.) [1999] *Así son los diccionarios*, Edicions de la Universitat de Lleida.

Nilsson, U., Małuszyński, J. [1995] *Logic, Programming and Prolog*, John Wiley.

Ovchinnikov, S. V. [1981] 'Representation of synonymy and antonymy by automorphisms in fuzzy sets theory', *Stochastica*, Vol. 5, Nº 2, pp. 95-107.

Owens, J. [1986] 'Synonymy and the nonindividualistic model of the mental', *Synthese*, Vol. 66, pp. 361-382.

Peregrin, J. [1997] 'Structure and Meaning', *Semiotica*, Vol. 113-1/2, pp. 71-88.

Putnam, H. [1954] 'Synonymy and the analysis of belief sentences', *Analysis*, Vol. 14, pp. 114-122.

Recarte, M. A., Peraita, H. [1988] 'Tiempos de verificación de la sinonimia: Implicaciones para las teorías de la memoria semántica', *Cognitiva*, Vol. 1 (3), pp. 229-243.

Roelofs, A. [1997] 'A case for nondecomposition in conceptually driven word retrieval', *Journal of Psycholinguistic Research*, Vol. 26, Nº 1, pp. 33-67.

Scheffler, I. [1965] 'On synonymy and indirect discourse', *Philosophy of Science*, Vol. 22, Nº 1, pp.39-44.

Sobrino, A. [1991] 'Regimentación y sinonimia en Carnap. Dos problemas relacionados con la formalización de la vaguedad del lenguaje natural', *Encuentro de Lógica y Filosofía de la Ciencia*, pp. 437-444.

Sobrino, A., Fernández Lanza, S. [2000] 'Synonymy from a Computational and Philosophical Perspective', *Proceedings of the eighth International Conference IPMU'2000*, Vol. II, pp. 784-791.

Tarán, L. [1978] 'Speusippus and Aristotle on Homonymy and Synonymy', *Hermes*, Vol. 106, pp. 73-99.

Tokarz, M. [1987] 'Synonymy in sentencial languages: a pragmatic view', *Studia Logica*, Vol. 48, Nº 2, pp. 93-97.

Trillas, E. [1993] 'Apunte sobre la indistinguibilidad', *Theoria: Segunda época*, Vol. 8, Nº 19, pp. 23-49.

Trillas, E., Riera, T. [1981] 'Towards a representation of "synonyms" and "antonyms" by fuzzy sets', *Busefal*, Vol. 5, pp. 42-68.

Vilares Ferro, M., Alonso Pardo, M., Valderruten, A. [1994] *Programación lógica*, Tórculo.

Weaver, G. E. [1994] 'Syntactic features and synonymy relations: a unified treatment of some proofs of the compactness and interpolación theorems', *Studia Logica*, Vol. 53, pp. 325-342.

Wilks, Y. [1996] 'Natural language processing', *Communications of the ACM*, Vol. 39, Nº 1.

APÉNDICE

1.- Programa que calcula el grado de sinonimia

```
include "C:\\VIP\\BIN\\WIN\\32\\DICCIONARIO\\ORDENAR.PRO"

DOMAINS
   list_grado = real*

PREDICATES
  max(list,string,list,string,list_grado,real)
  union(list,list,list)
  inter(list,list,list)
  dif(list,list,list)
  num(list,integer)
nondeterm sin_dic(string,list,string,string)
nondeterm min(integer,integer,integer)
nondeterm
sinonimo(string,string,string,string,real,real,string,string,
     string)
nondeterm comparable(string,string,string,string)
nondeterm miembro(string,list)
nondeterm
sin_acep(string,string,string,string,real,real,string,string,
     string)
nondeterm
sin_acep2(string,string,string,string,real,real,string,string,
     string)
nondeterm dic2(string,string,string,string)
```

```
CLAUSES

%INCLUSIÓN DEL CASO REFLEXIVO EN LA BASE DE DATOS
sin_dic(A,[A|B],C,D):-
      dic(A,B,C,D).

%SINONIMIA ENTRE PALABRAS
sinonimo(A,X,A,X,1.0,_,_,HomA,HomA):-
      sin_dic(A,_,X,HomA).

sinonimo(A,AcepA,B,AcepB,GRS_MAX,Umbral,MedSim,HomA,HomB):-
      comparable(A,B,AcepA,HomA),
      not(A=B),
      findall(GRS,sin_acep(A,AcepA,B,_,GRS,Umbral,MedSim,HomA,_),
            Lista1),
      findall(Acep,sin_acep(A,AcepA,B,Acep,_,Umbral,MedSim,HomA,
            _),Lista2),
      findall(Hom,sin_acep(A,AcepA,B,_,_,Umbral,MedSim,HomA,Hom),
            Lista3),
      max(Lista3,HomB,Lista2,AcepB,Lista1,GRS_MAX).

%PREDICADO PARA SINÓNIMOS QUE NO SON ENTRADAS
dic2(Palabra,AcepA,HomA,Sino):-
      dic(Palabra,LSIN,AcepA,HomA),
      miembro(Sino,Lsin),
      not(dic(Sino,_,_,_)).

%PREDICADO PARA LAS PALABRAS RELACIONADAS
sin_acep2(A,AcepA,B,AcepB,GRS,Umbral,MS,HomA,HomB):-
      sin_acep(A,AcepA,B,AcepB,GRS,Umbral,MS,HomA,HomB),
      not(GRS = 0),
      not(sinonimo(A,_,B,_,_,Umbral,MS,_,_)).

%COMPARABILIDAD ENTRE PALABRAS
comparable(A,B,AcepA,HomA):-
      sin_dic(A,X,AcepA,HomA),
      miembro(B,X).

%MÁXIMO DE LOS GRADOS DE UNA LISTA, ACEPCIÓN, ÍNDICE DE
HOMÓGRAFA Y DERIVACIÓN CORRESPONDIENTES A ESE GRADO MÁXIMO
max([C],C,[B],B,[A],A).
max([G,_|H],Z,[D,_|F],Y,[A,B|C],X):-
      A >= B,
      max([G|H],Z,[D|F],Y,[A|C],X),
      !.
max([_,G|H],Z,[_,E|F],Y,[_,B|C],X):-
      max([G|H],Z,[E|F],Y,[B|C],X).

%SINONIMIA ENTRE ACEPCIONES
%Medida de similaridad: Intersección partido por la unión
sin_acep(A,AcepA,B,AcepB,GRS,Umbral,"Coeficiente jaccard",HomA,
HomB):-
      sin_dic(A,X,AcepA,HomA),
```

```
      sin_dic(B,Y,AcepB,HomB),
      not(A=B),
      union(X,Y,U),
      inter(X,Y,I),
      num(U,NU),
      num(I,NI),
      GRS = NI/NU,
      GRS >= Umbral.

%Medida de similaridad: intersección partido por el minimo de
las cardinalidades
sin_acep(A,AcepA,B,AcepB,GRS,Umbral,"Coeficiente de
solapamiento",HomA,HomB):-
      sin_dic(A,X,AcepA,HomA),
      sin_dic(B,Y,AcepB,HomB),
      not(A=B),
      inter(X,Y,I),
      num(I,NI),
      num(X,NX),
      num(Y,NY),
      min(NX,NY,Min),
      GRS = NI/Min,
      GRS >= Umbral.

%Medida de similaridad: intersección partido por la media
aritmética de las cardinalidades
sin_acep(A,AcepA,B,AcepB,GRS,Umbral,"Coeficiente del dado",HomA,
HomB):-
      sin_dic(A,X,AcepA,HomA),
      sin_dic(B,Y,AcepB,HomB),
      not(A=B),
      inter(X,Y,I),
      num(I,NI),
      num(X,NX),
      num(Y,NY),
      GRS = (2*NI)/(NX+NY),
      GRS >= Umbral.

%Medida de similaridad: intersección partido por la media
geométrica de las cardinalidades
sin_acep(A,AcepA,B,AcepB,GRS,Umbral,"Coeficiente del coseno",
HomA,HomB):-
      sin_dic(A,X,AcepA,HomA),
      sin_dic(B,Y,AcepB,HomB),
      not(A=B),
      inter(X,Y,I),
      num(I,NI),
      num(X,NX),
      num(Y,NY),
      SQRX = sqrt(NX),
      SQRY = sqrt(NY),
      GRS = NI/(SQRX*SQRY),
      GRS >= Umbral.
```

```
%Medida de similaridad: intersección partido por la media
armónica de las cardinalidades
sin_acep(A,AcepA,B,AcepB,GRS,Umbral,"Coeficiente de similaridad
mutua",HomA,HomB):-
      sin_dic(A,X,AcepA,HomA),
      sin_dic(B,Y,AcepB,HomB),
      not(A=B),
      inter(X,Y,I),
      num(I,NI),
      num(X,NX),
      num(Y,NY),
      SLR = NI/NX,
      SRL = NI/NY,
      GRS = (SLR+SRL)/2,
      GRS >= Umbral.

%Medida de similaridad de Tversky
sin_acep(A,AcepA,B,AcepB,GRS,Umbral,"Medida de Tversky",HomA,
HomB):-
      sin_dic(A,X,AcepA,HomA),
      sin_dic(B,Y,AcepB,HomB),
      miembro(A,Y),
      not(miembro(B,X)),
      not(A=B),
      inter(X,Y,I),
      dif(X,Y,DXY),
      dif(Y,X,DYX),
      num(I,NI),
      num(DXY,NDXY),
      num(DYX,NDYX),
      GRS = NI/(NI+NDXY+2*NDYX),
      GRS >= Umbral.
sin_acep(A,AcepA,B,AcepB,GRS,Umbral,"Medida de Tversky",HomA,
HomB):-
      sin_dic(A,X,AcepA,HomA),
      sin_dic(B,Y,AcepB,HomB),
      not(miembro(A,Y)),
      miembro(B,X),
      not(A=B),
      inter(X,Y,I),
      dif(X,Y,DXY),
      dif(Y,X,DYX),
      num(I,NI),
      num(DXY,NDXY),
      num(DYX,NDYX),
      GRS = NI/(NI+NDXY+2*NDYX),
      GRS >= Umbral.
sin_acep(A,AcepA,B,AcepB,GRS,Umbral,"Medida de Tversky",HomA,
HomB):-
      sin_dic(A,X,AcepA,HomA),
      sin_dic(B,Y,AcepB,HomB),
      not(miembro(A,Y)),
```

```
      not(miembro(B,X)),
      not(A=B),
      inter(X,Y,I),
      dif(X,Y,DXY),
      dif(Y,X,DYX),
      num(I,NI),
      num(DXY,NDXY),
      num(DYX,NDYX),
      GRS = NI/(NI+NDXY+NDYX),
      GRS >= Umbral.
sin_acep(A,AcepA,B,AcepB,GRS,Umbral,"Medida de Tversky",HomA,
HomB):-
      sin_dic(A,X,AcepA,HomA),
      sin_dic(B,Y,AcepB,HomB),
      miembro(A,Y),
      miembro(B,X),
      not(A=B),
      inter(X,Y,I),
      dif(X,Y,DXY),
      dif(Y,X,DYX),
      num(I,NI),
      num(DXY,NDXY),
      num(DYX,NDYX),
      GRS = NI/(NI+NDXY+NDYX),
      GRS >= Umbral.

%UNIÓN DE LISTAS
union([],X,X).
union([A|B],Y,Z):-
      miembro(A,Y),!,
      union(B,Y,Z).
union([A|B],Y,[A|Z]):-
      union(B,Y,Z).

%INTERSECCIÓN DE LISTAS
inter([],_,[]).
inter([A|B],Y,[A|Z]):-
      miembro(A,Y),!,
      inter(B,Y,Z).
inter([_|B],Y,Z):-
      inter(B,Y,Z).

%DIFERENCIA DE LISTAS
dif([],_,[]).
dif([A|B],C,[A|D]):-
      not(miembro(A,C)),
      dif(B,C,D),
      !.
dif([_|B],C,D):-
      dif(B,C,D).

%NÚMERO DE ELEMENTOS DE UNA LISTA
num([],0).
```

```
num([_|B],C):-
      num(B,E),
      C = E + 1.0.

%MÍNIMO DE DOS NÚMEROS
min(A,B,A):-A<=B.
min(A,B,B):-A>B.

%PERTENENCIA A UNA LISTA
miembro(A,[A|_]).
miembro(A,[_|C]):-
      miembro(A,C).
```

2.- Programa que ordena los sinónimos según el grado

```
DOMAINS
   lista_grado = real*
   lista_int = integer*

PREDICATES

  maxi(list,string,list,string,list,string,lista_grado,real)
nondeterm
ordenar(list,list,list,list,list,list,lista_grado,lista_grado)
nondeterm  bor(lista_int,list,list)
nondeterm  bor(lista_int,lista_grado,lista_grado)
nondeterm  pos(string,list,lista_int)
nondeterm  pos(real,lista_grado,lista_int)

CLAUSES
%PREDICADO QUE ORDENA LAS LISTAS DE SINÓNIMOS, ACEPCIONES,
INDICES DE HOMÓGRAFAS, DERIVACIONES Y GRADOS
ordenar([A],[A],[B],[B],[C],[C],[E],[E]).
ordenar([A1|B1],[C1|D1],[E1|F1],[G1|H1],[A|B],[C|D],[I|J],
[K|L]):-
      maxi([A1|B1],C1,[E1|F1],G1,[A|B],C,[I|J],K),
      pos(C1,[A1|B1],Pos),
      pos(G1,[E1|F1],Pos),
      pos(C,[A|B],Pos),
      pos(K,[I|J],Pos),
      bor(Pos,[A1|B1],X1),
      bor(Pos,[E1|F1],Y1),
      bor(Pos,[A|B],X),
      bor(Pos,[I|J],Z),
      ordenar(X1,D1,Y1,H1,X,D,Z,L),
      !.
```

```
%PREDICADO QUE CALCULA EL GRADO MÁXIMO DE UNA LISTA, CON SUS
SINÓNIMOS, ACEPCIONES, INDICES DE HOMÓGRAFAS Y DERIVACIONES
CORRESPONDIENTES
maxi([E],E,[D],D,[C],C,[A],A).
maxi([L,_|M],U,[J,_|K],W,[G,_|I],Z,[A,B|C],X):-
      A >= B,
      maxi([L|M],U,[J|K],W,[G|I],Z,[A|C],X),
      !.
maxi([_,L|M],U,[_,J|K],W,[_,H|I],Z,[_,B|C],X):-
      maxi([L|M],U,[J|K],W,[H|I],Z,[B|C],X).

%BORRA EL ELEMENTO QUE FIGURA EN DETERMINADA POSICIÓN DE UNA
LISTA
bor([],[],[]).
bor([1],[_|D],D):-
      bor([],D,D).
bor([_|B],[A|D],[A|F]):-
      bor(B,D,F),
      !.
bor([],[A|D],[A|D]):-
      bor([],D,D).

%INDICA LA POSICIÓN QUE TIENE UN ELEMENTO EN UNA LISTA
pos(X,[X|_],[1]).
pos(X,[_|Z],[1|Y]):-
      pos(X,Z,Y).
```

3.- Programa que clasifica la información adicional

```
PREDICATES

   nondeterm flexion(string)
   nondeterm flex(string,string,string,string)
   nondeterm gramatical(string,string)
   nondeterm gram(string,string,string,string)
   nondeterm tecnicismo(string,string)
   nondeterm tecn(string,string,string,string)
   nondeterm dialectalismo(string,string)
   nondeterm dial(string,string,string,string)
   nondeterm prestamo(string,string)
   nondeterm prest(string,string,string,string)
   nondeterm pragmatica(string,string)
   nondeterm prag(string,string,string,string)
   nondeterm diacronico(string,string)
   nondeterm diac(string,string,string,string)
   nondeterm otros_datos(string,string,string,string)

CLAUSES
```

```
%BASE DE DATOS CON SUFIJOS FLEXIVOS
flexion("-triz").
flexion("-gria").
flexion("-bo").
flexion("-nas").
flexion("-bas").
flexion("-plia").
flexion("-gla").
flexion("-ros").
flexion("-ata").
flexion("-ar").
flexion("-dua").
flexion("-sas").
flexion("-llona").
flexion("-as").
flexion("-gua").
flexion("-pa").
flexion("-rra").
flexion("-nua").
flexion("-xa").
flexion("-bra").
flexion("-chas").
flexion("-mia").
flexion("-bla").
flexion("-sia").
flexion("-ina").
flexion("-tua").
flexion("-tia").
flexion("-tisa").
flexion("-esa").
flexion("-pua").
flexion("-via").
flexion("-pia").
flexion("-ía").
flexion("-gia").
flexion("-dia").
flexion("-osa").
flexion("-nia").
flexion("-tra").
flexion("-cha").
flexion("-bria").
flexion("-pla").
flexion("-gra").
flexion("-flua").
flexion("-fa").
flexion("-lia").
flexion("-ba").
flexion("-cra").
flexion("-ya").
flexion("-dra").
flexion("-ana").
flexion("-bia").
```

```
flexion("-a").
flexion("-cua").
flexion("-das").
flexion("-rias").
flexion("-lla").
flexion("-ga").
flexion("-ja").
flexion("-na").
flexion("-cia").
flexion("-ea").
flexion("-ria").
flexion("-va").
flexion("-ma").
flexion("-fia").
flexion("-la").
flexion("-ña").
flexion("-ra").
flexion("-sa").
flexion("-ta").
flexion("-za").
flexion("-ca").
flexion("-ona").
flexion("-da").

%BASE DE DATOS CON ABREVIATURAS DE CUESTIONES GRAMATICALES Y LAS
EXPRESIONES CORRESPONDIENTES A DICHAS ABREVIATURAS
gramatical("adj","Adjetivo").
gramatical("adv","Adverbio").
gramatical("adv c","Adverbio de cantidad").
gramatical("adv d","Adverbio de duda").
gramatical("adv l","Adverbio de lugar").
gramatical("adv m","Adverbio de modo").
gramatical("adv t","Adverbio de tiempo").
gramatical("amb","Ambiguo de género").
gramatical("com","Común de género").
gramatical("conj","Conjunción").
gramatical("conj adv","Conjunción adversativa").
gramatical("conj causal","Conjunción causal").
gramatical("conj conces","Conjunción concesiva").
gramatical("conj consec","Conjunción consecutiva").
gramatical("conj final","Conjunción final").
gramatical("f","Femenino").
gramatical("impers","Verbo impersonal").
gramatical("interj","Interjección").
gramatical("intr","Verbo intransitivo").
gramatical("loc","Locución").
gramatical("loc adj","Locución adjetiva").
gramatical("loc adv","Locución adverbial").
gramatical("loc conj","Locución conjuntiva").
gramatical("loc lat","Locución latina").
gramatical("loc prep","Locución prepositiva").
gramatical("m","Masculino").
gramatical("pers","Persona o personal").
```

```
gramatical("pl","Plural").
gramatical("pp","Participio de pasado").
gramatical("prep","Preposición").
gramatical("prnl","Pronominal").
gramatical("pron","Pronombre").
gramatical("pron indef","Pronombre indefinido").
gramatical("pron pers","Pronombre personal").
gramatical("pron rel","Pronombre relativo").
gramatical("s","Sustantivo").
gramatical("tr","Verbo transitivo").
gramatical("voz adv lat","Voz adverbial latina").
gramatical("voz verb lat","Voz verbal latina").

%BASE DE DATOS CON ABREVIATURAS DE TECNICISMOS Y LAS EXPRESIONES
CORRESPONDIENTES A DICHAS ABREVIATURAS
tecnicismo("aeron", "Aeronáutica").
tecnicismo("agr", "Agricultura").
tecnicismo("albañ", "Albañilería").
tecnicismo("anat", "Anatomía").
tecnicismo("arq", "Arqueología").
tecnicismo("astron", "Astronomía").
tecnicismo("bib", "Biblia").
tecnicismo("biol", "Biología").
tecnicismo("blas", "Blasonería").
tecnicismo("bot", "Botánica").
tecnicismo("carp", "Carpintería").
tecnicismo("cientif", "Científico").
tecnicismo("cinem", "Cinematografía").
tecnicismo("cir", "Cirugía").
tecnicismo("comerc", "Comercio").
tecnicismo("constr", "Construcción").
tecnicismo("dep", "Deportes").
tecnicismo("der", "Derecho").
tecnicismo("electr", "Electricidad, electrónica").
tecnicismo("esgr", "Esgrima").
tecnicismo("farm", "Farmacia").
tecnicismo("fil", "Filosofía").
tecnicismo("filol", "Filología").
tecnicismo("fis", "Física").
tecnicismo("fisiol", "Fisiología").
tecnicismo("fot", "Fotografía").
tecnicismo("geogr", "Geografía").
tecnicismo("geol", "Geología").
tecnicismo("geom", "Geometría").
tecnicismo("gram", "Gramática").
tecnicismo("h nat", "Historia natural").
tecnicismo("impr", "Imprenta").
tecnicismo("inform", "Informática").
tecnicismo("ling", "Lingüística").
tecnicismo("liturg", "Liturgia").
tecnicismo("log", "Lógica").
tecnicismo("mar", "Marina").
tecnicismo("mat", "Matemáticas").
```

```
tecnicismo("mec", "Mecánica").
tecnicismo("med", "Medicina").
tecnicismo("metal", "Metalurgia").
tecnicismo("meteor", "Meteorología").
tecnicismo("mil", "Milicia").
tecnicismo("min", "Minería").
tecnicismo("mineral", "Mineralogía").
tecnicismo("mit", "Mitología").
tecnicismo("mont", "Montería").
tecnicismo("mus", "Música").
tecnicismo("pint", "Pintura").
tecnicismo("quim", "Química").
tecnicismo("ret", "Retórica").
tecnicismo("taurom", "Tauromaquia").
tecnicismo("tecn", "Tecnicismo").
tecnicismo("tecnol", "Tecnología").
tecnicismo("teol", "Teología").
tecnicismo("top", "Topografía").
tecnicismo("veter", "Veterinaria").
tecnicismo("zool", "Zoología").

%BASE DE DATOS CON ABREVIATURAS DE DIALECTALISMOS Y LAS
EXPRESIONES CORRESPONDIENTES A DICHAS ABREVIATURAS
dialectalismo("al", "Álava").
dialectalismo("amer", "América").
dialectalismo("amer central", "América Central").
dialectalismo("amer merid", "América Meridional").
dialectalismo("and", "Andalucía").
dialectalismo("antill", "Antillas").
dialectalismo("ar", "Aragón").
dialectalismo("argent", "Argentina").
dialectalismo("ast", "Asturias").
dialectalismo("bol", "Bolivia").
dialectalismo("burg", "Burgos").
dialectalismo("can", "Canarias").
dialectalismo("cast", "Castilla").
dialectalismo("cat", "Cataluña").
dialectalismo("colomb", "colombia").
dialectalismo("c rica", "Costa Rica").
dialectalismo("dial", "dialectal").
dialectalismo("ecuad", "Ecuador").
dialectalismo("filip", "Filipinas").
dialectalismo("gal", "Galicia").
dialectalismo("gran", "Granada").
dialectalismo("hond", "Honduras").
dialectalismo("lev", "Levante").
dialectalismo("logr", "Logroño").
dialectalismo("mex", "México").
dialectalismo("murc", "Murcia").
dialectalismo("nav", "Navarra").
dialectalismo("nicar", "Nicaragua").
dialectalismo("par", "Paraguay").
dialectalismo("p rico", "Puerto Rico").
```

```
dialectalismo("sant", "Santander").
dialectalismo("urug", "Uruguay").
dialectalismo("val", "Valencia").
dialectalismo("venez", "Venezuela").
dialectalismo("Cuba", "Cuba").
dialectalismo("Chile", "Chile").
dialectalismo("Perú", "Perú").
dialectalismo("Madrid", "Madrid").
dialectalismo("País Vasco", "País Vasco").
dialectalismo("Brasil", "Brasil").
dialectalismo("La Mancha", "La Mancha").

%BASE DE DATOS CON ABREVIATURAS DE PRÉSTAMOS LINGÜÍSTICOS Y LAS
EXPRESIONES CORRESPONDIENTES A DICHAS ABREVIATURAS
prestamo("anglic", "Anglicismo").
prestamo("angl", "Anglicismo").
prestamo("galic", "Galicismo").
prestamo("italian", "Italianismo").
prestamo("lat", "Latinismo").

%BASE DE DATOS CON ABREVIATURAS DE CUESTIONES PRAGMÁTICAS Y LAS
EXPRESIONES CORRESPONDIENTES A DICHAS ABREVIATURAS
pragmatica("burl", "Burlesco").
pragmatica("col", "Coloquial").
pragmatica("cult", "Cultismo").
pragmatica("desp", "Despectivo").
pragmatica("dim", "Diminutivo").
pragmatica("eufem", "Eufemismo").
pragmatica("fam", "Familiar").
pragmatica("fig", "Sentido figurado").
pragmatica("hum", "Humorístico").
pragmatica("humor", "Humorístico").
pragmatica("intens", "Intensivo").
pragmatica("iron", "Irónico").
pragmatica("lit", "Literario").
pragmatica("pleb", "Plebeyo").
pragmatica("poet", "Poético").
pragmatica("rust", "Rústico").
pragmatica("vulg", "Vulgar").

%BASE DE DATOS CON ABREVIATURAS DE CUESTIONES DIACRÓNICAS Y LAS
EXPRESIONES CORRESPONDIENTES A DICHAS ABREVIATURAS
diacronico("ant", "Anticuado o antiguo").
diacronico("antic", "Anticuado o antiguo").
diacronico("desus", "Desusado").
diacronico("neol", "Neologismo").
diacronico("p us", "Poco usado").

%INDICA LA FLEXIÓN PARA UNA ENTRADA, ACEPCIÓN Y HOMÓGRAFA
flex(Entr,Acep,Hom,Flex):-
      ant(Entr,Acep,Hom,Lista,_),
      miembro(Flex,Lista),
      flexion(Flex).
```

```
%INDICA CUESTIONES GRAMATICALES PARA UNA ENTRADA, ACEPCIÓN Y
HOMÓGRAFA
gram(Entr,Acep,Hom,Gram):-
      ant(Entr,Acep,Hom,Lista,_),
      miembro(Item,Lista),
      gramatical(Item,Gram).

%INDICA TECNICISMOS PARA UNA ENTRADA, ACEPCIÓN Y HOMÓGRAFA
tecn(Entr,Acep,Hom,Tecn):-
      ant(Entr,Acep,Hom,Lista,_),
      miembro(Item,Lista),
      tecnicismo(Item,Tecn).

%INDICA DIALECTALISMOS PARA UNA ENTRADA, ACEPCIÓN Y HOMÓGRAFA
dial(Entr,Acep,Hom,Dial):-
      ant(Entr,Acep,Hom,Lista,_),
      miembro(Item,Lista),
      dialectalismo(Item,Dial).

%INDICA PRESTAMOS LINGÜÍSTICOS PARA UNA ENTRADA, ACEPCIÓN Y
HOMÓGRAFA
prest(Entr,Acep,Hom,Prest):-
      ant(Entr,Acep,Hom,Lista,_),
      miembro(Item,Lista),
      prestamo(Item,Prest).

%INDICA CUESTIONES PRAGMÁTICAS PARA UNA ENTRADA, ACEPCIÓN Y
HOMÓGRAFA
prag(Entr,Acep,Hom,Prag):-
      ant(Entr,Acep,Hom,Lista,_),
      miembro(Item,Lista),
      pragmatica(Item,Prag).

%INDICA CUESTIONES DIACRÓNICAS PARA UNA ENTRADA, ACEPCIÓN Y
HOMÓGRAFA
diac(Entr,Acep,Hom,Diac):-
      ant(Entr,Acep,Hom,Lista,_),
      miembro(Item,Lista),
      diacronico(Item,Diac).

%INDICA OTROS DATOS PARA UNA ENTRADA, ACEPCIÓN Y HOMÓGRAFA
otros_datos(Entr,Acep,Hom,Item):-
      ant(Entr,Acep,Hom,Lista,_),
      miembro(Item,Lista),
      not(flexion(Item)),
      not(gramatical(Item,_)),
      not(tecnicismo(Item,_)),
      not(dialectalismo(Item,_)),
      not(prestamo(Item,_)),
      not(pragmatica(Item,_)),
      not(diacronico(Item,_)).
```

4.- Código del interfaz

```
/*************************************************************

Copyright (c) Santy 2000 Universidad de Santiago de Compostela

 Project:  DICCIONARIO DE SINÓNIMOS
 FileName: PALABRAS.PRO
 Purpose: No description
 Written by: Santiago Fernández Lanza
 Comments:
*************************************************************/
include "global.inc"
include "dicciona.inc"
include "dicciona.con"
include "hlptopic.con"
include "codigo.pro"
include "anadir.pro"
include "mas_dic.pro"

%BEGIN_DLG Diccionario de Sinónimos
/*************************************************************
Creation and event handling for dialog: Diccionario de Sinónimos
*************************************************************/

CONSTANTS

%BEGIN Diccionario de Sinónimos, CreateParms, 14:06:57-
19.1.2001, Code automatically updated!
dlg_diccionario_de_sinónimos_ResID=idd_diccionario_de_sinónimos
dlg_diccionario_de_sinónimos_DlgType = wd_Modal
dlg_diccionario_de_sinónimos_Help = contents
%END Diccionario de Sinónimos, CreateParms

PREDICATES

dlg_diccionario_de_sinónimos_eh : EHANDLER
dlg_diccionario_de_sinónimos_handle_answer(INTEGER EndButton,
     DIALOG_VAL_LIST)
dlg_diccionario_de_sinónimos_update(DIALOG_VAL_LIST)

CLAUSES

  dlg_diccionario_de_sinónimos_Create(Parent):-

%MARK Diccionario de Sinónimos, new variables

     dialog_CreateModal(Parent,dlg_diccionario_de_sinónimos_
```

```
          ResID,"",
          [
%BEGIN Diccionario de Sinónimos, ControlList, 14:06:57-
19.1.2001, Code automatically updated!
          df(idc_edit,editstr("",[]),nopr),
          df(lista_hom1,listbox([],[0]),nopr),
          df(lista_acep1,listbox([],[0]),nopr),
          df(lista_palabras2,listbox([],[0]),nopr),
          df(medidas_de_similaridad,listbutton(["Coeficiente
jaccard","Coeficiente del dado","Coeficiente del
coseno","Coeficiente de similaridad mutua","Coeficiente de
solapamiento","Medida de Tversky"],0),nopr),
          df(umbral,scrollbar(1,10,0,100,0),nopr),
          df(expresiones_relacionadas,listbox([],[0]),nopr),
          df(lista_flex,listbutton([],0),nopr),
          df(lista_gram,listbutton([],0),nopr),
          df(lista_tecn,listbutton([],0),nopr),
          df(lista_dial,listbutton([],0),nopr),
          df(lista_prest,listbutton([],0),nopr),
          df(lista_prag,listbutton([],0),nopr),
          df(lista_diac,listbutton([],0),nopr),

     df(idc_diccionario_de_sinónimos_65,listbutton([],0),nopr),

     df(idc_diccionario_de_sinónimos_67,listbutton([],0),nopr),

     df(idc_diccionario_de_sinónimos_69,listbutton([],0),nopr),

     df(idc_diccionario_de_sinónimos_71,listbutton([],0),nopr),

     df(idc_diccionario_de_sinónimos_73,listbutton([],0),nopr),

     df(idc_diccionario_de_sinónimos_75,listbutton([],0),nopr),

     df(idc_diccionario_de_sinónimos_77,listbutton([],0),nopr),
          df(lista_otros,listbutton([],0),nopr),

     df(idc_diccionario_de_sinónimos_80,listbutton([],0),nopr),

     df(idc_diccionario_de_sinónimos_7,listbutton([],0),nopr),

     df(idc_diccionario_de_sinónimos_30,listbutton([],0),nopr),

     df(idc_diccionario_de_sinónimos_83,listbutton([],0),nopr),

     df(idc_diccionario_de_sinónimos_85,listbutton([],0),nopr),

     df(idc_diccionario_de_sinónimos_87,listbutton([],0),nopr),

     df(idc_diccionario_de_sinónimos_89,listbutton([],0),nopr),

     df(idc_diccionario_de_sinónimos_91,listbutton([],0),nopr),
```

```
      df(idc_diccionario_de_sinónimos_93,listbutton([],0),nopr),
           df(lista_acepciones2,listbox([],[0]),nopr),
           df(lista_hom2,listbox([],[0]),nopr),
           df(lista_acep3,listbox([],[0]),nopr),
           df(lista_acep4,listbox([],[0]),nopr),

      df(idc_diccionario_de_sinónimos_48,listbox([],[0]),nopr),

      df(barra_grado_sinonimas,scrollbar(0,0,0,100,100),nopr),

      df(barra_grado_relacionadas,scrollbar(0,0,0,100,100),nopr)
%END Diccionario de Sinónimos, ControlList
           ],
           dlg_diccionario_de_sinónimos_eh,0,VALLIST,ANSWER),
      dlg_diccionario_de_sinónimos_handle_answer(ANSWER,VALLIST).

  dlg_diccionario_de_sinónimos_handle_answer(idc_ok,VALLIST):-!,
      dlg_diccionario_de_sinónimos_update(VALLIST).
  dlg_diccionario_de_sinónimos_handle_answer(idc_cancel,_):-!.
% Handle Esc and Cancel here
  dlg_diccionario_de_sinónimos_handle_answer(_,_):-
      errorexit().

  dlg_diccionario_de_sinónimos_update(_VALLIST):-
%BEGIN Diccionario de Sinónimos, Update controls, 14:06:57-
19.1.2001, Code automatically updated!
      dialog_VLGetListButton(medidas_de_similaridad,_VALLIST,_MED
IDAS_DE_SIMILARIDAD_ITEMLIST,_MEDIDAS_DE_SIMILARIDAD_SELECT),
      _UMBRAL_POS = dialog_VLGetScrollBar(umbral,_VALLIST),
      _IDC_EDIT_VALUE = dialog_VLGetstr(idc_edit,_VALLIST),
      dialog_VLGetListBox(lista_hom1,_VALLIST,_LISTA_HOM1_ITEMLIS
T,_LISTA_HOM1_SELECT),
      dialog_VLGetListBox(lista_acep1,_VALLIST,_LISTA_ACEP1_ITEML
IST,_LISTA_ACEP1_SELECT),
      dialog_VLGetListBox(lista_palabras2,_VALLIST,_LISTA_PALABRA
S2_ITEMLIST,_LISTA_PALABRAS2_SELECT),
      dialog_VLGetListBox(lista_acepciones2,_VALLIST,_LISTA_ACEPC
IONES2_ITEMLIST,_LISTA_ACEPCIONES2_SELECT),
      dialog_VLGetListBox(lista_hom2,_VALLIST,_LISTA_HOM2_ITEMLIS
T,_LISTA_HOM2_SELECT),
      dialog_VLGetListBox(expresiones_relacionadas,_VALLIST,_EXPR
ESIONES_RELACIONADAS_ITEMLIST,_EXPRESIONES_RELACIONADAS_SELECT),
      dialog_VLGetListBox(lista_acep3,_VALLIST,_LISTA_ACEP3_ITEML
IST,_LISTA_ACEP3_SELECT),
      dialog_VLGetListBox(lista_acep4,_VALLIST,_LISTA_ACEP4_ITEML
IST,_LISTA_ACEP4_SELECT),
      dialog_VLGetListBox(idc_diccionario_de_sinónimos_48,_VALLIS
T,_IDC_DICCIONARIO_DE_SINÓNIMOS_48_ITEMLIST,_IDC_DICCIONARIO_DE_
SINÓNIMOS_48_SELECT),
      dialog_VLGetListButton(lista_flex,_VALLIST,_LISTA_FLEX_ITEM
LIST,_LISTA_FLEX_SELECT),
```

```
     dialog_VLGetListButton(lista_gram,_VALLIST,_LISTA_GRAM_ITEM
LIST,_LISTA_GRAM_SELECT),
     dialog_VLGetListButton(lista_tecn,_VALLIST,_LISTA_TECN_ITEM
LIST,_LISTA_TECN_SELECT),
     dialog_VLGetListButton(lista_dial,_VALLIST,_LISTA_DIAL_ITEM
LIST,_LISTA_DIAL_SELECT),
     dialog_VLGetListButton(lista_prest,_VALLIST,_LISTA_PREST_IT
EMLIST,_LISTA_PREST_SELECT),
     dialog_VLGetListButton(lista_prag,_VALLIST,_LISTA_PRAG_ITEM
LIST,_LISTA_PRAG_SELECT),
     dialog_VLGetListButton(lista_diac,_VALLIST,_LISTA_DIAC_ITEM
LIST,_LISTA_DIAC_SELECT),
     dialog_VLGetListButton(idc_diccionario_de_sinónimos_65,_VAL
LIST,_IDC_DICCIONARIO_DE_SINÓNIMOS_65_ITEMLIST,_IDC_DICCIONARIO_
DE_SINÓNIMOS_65_SELECT),
     dialog_VLGetListButton(idc_diccionario_de_sinónimos_67,_VAL
LIST,_IDC_DICCIONARIO_DE_SINÓNIMOS_67_ITEMLIST,_IDC_DICCIONARIO_
DE_SINÓNIMOS_67_SELECT),
     dialog_VLGetListButton(idc_diccionario_de_sinónimos_69,_VAL
LIST,_IDC_DICCIONARIO_DE_SINÓNIMOS_69_ITEMLIST,_IDC_DICCIONARIO_
DE_SINÓNIMOS_69_SELECT),
     dialog_VLGetListButton(idc_diccionario_de_sinónimos_71,_VAL
LIST,_IDC_DICCIONARIO_DE_SINÓNIMOS_71_ITEMLIST,_IDC_DICCIONARIO_
DE_SINÓNIMOS_71_SELECT),
     dialog_VLGetListButton(idc_diccionario_de_sinónimos_73,_VAL
LIST,_IDC_DICCIONARIO_DE_SINÓNIMOS_73_ITEMLIST,_IDC_DICCIONARIO_
DE_SINÓNIMOS_73_SELECT),
     dialog_VLGetListButton(idc_diccionario_de_sinónimos_75,_VAL
LIST,_IDC_DICCIONARIO_DE_SINÓNIMOS_75_ITEMLIST,_IDC_DICCIONARIO_
DE_SINÓNIMOS_75_SELECT),
     dialog_VLGetListButton(idc_diccionario_de_sinónimos_77,_VAL
LIST,_IDC_DICCIONARIO_DE_SINÓNIMOS_77_ITEMLIST,_IDC_DICCIONARIO_
DE_SINÓNIMOS_77_SELECT),
     dialog_VLGetListButton(lista_otros,_VALLIST,_LISTA_OTROS_IT
EMLIST,_LISTA_OTROS_SELECT),
     dialog_VLGetListButton(idc_diccionario_de_sinónimos_80,_VAL
LIST,_IDC_DICCIONARIO_DE_SINÓNIMOS_80_ITEMLIST,_IDC_DICCIONARIO_
DE_SINÓNIMOS_80_SELECT),
     dialog_VLGetListButton(idc_diccionario_de_sinónimos_7,_VALL
IST,_IDC_DICCIONARIO_DE_SINÓNIMOS_7_ITEMLIST,_IDC_DICCIONARIO_DE
_SINÓNIMOS_7_SELECT),
     dialog_VLGetListButton(idc_diccionario_de_sinónimos_30,_VAL
LIST,_IDC_DICCIONARIO_DE_SINÓNIMOS_30_ITEMLIST,_IDC_DICCIONARIO_
DE_SINÓNIMOS_30_SELECT),
     dialog_VLGetListButton(idc_diccionario_de_sinónimos_83,_VAL
LIST,_IDC_DICCIONARIO_DE_SINÓNIMOS_83_ITEMLIST,_IDC_DICCIONARIO_
DE_SINÓNIMOS_83_SELECT),
     dialog_VLGetListButton(idc_diccionario_de_sinónimos_85,_VAL
LIST,_IDC_DICCIONARIO_DE_SINÓNIMOS_85_ITEMLIST,_IDC_DICCIONARIO_
DE_SINÓNIMOS_85_SELECT),
     dialog_VLGetListButton(idc_diccionario_de_sinónimos_87,_VAL
LIST,_IDC_DICCIONARIO_DE_SINÓNIMOS_87_ITEMLIST,_IDC_DICCIONARIO_
DE_SINÓNIMOS_87_SELECT),
```

```
      dialog_VLGetListButton(idc_diccionario_de_sinónimos_89,_VAL
LIST,_IDC_DICCIONARIO_DE_SINÓNIMOS_89_ITEMLIST,_IDC_DICCIONARIO_
DE_SINÓNIMOS_89_SELECT),
      dialog_VLGetListButton(idc_diccionario_de_sinónimos_91,_VAL
LIST,_IDC_DICCIONARIO_DE_SINÓNIMOS_91_ITEMLIST,_IDC_DICCIONARIO_
DE_SINÓNIMOS_91_SELECT),
      dialog_VLGetListButton(idc_diccionario_de_sinónimos_93,_VAL
LIST,_IDC_DICCIONARIO_DE_SINÓNIMOS_93_ITEMLIST,_IDC_DICCIONARIO_
DE_SINÓNIMOS_93_SELECT),
      _BARRA_GRADO_SINONIMAS_POS=dialog_VLGetScrollBar(barra_
grado_sinonimas,_VALLIST),
      _BARRA_GRADO_RELACIONADAS_POS=dialog_VLGetScrollBar(barra_
grado_relacionadas,_VALLIST),
%END Diccionario de Sinónimos, Update controls
      true.

%MARK Diccionario de Sinónimos, new events

%BEGIN Diccionario de Sinónimos, idc_más _CtlInfo
  dlg_diccionario_de_sinónimos_eh(_Win,e_Control(idc_más,
_CtrlType,_CtrlWin,_CtlInfo),0):-!,
      BOTONMENOS = win_GetCtlHandle(_Win, idc_menos),
      BOTONMAS = win_GetCtlHandle(_Win, idc_más),
      OuterRCT = win_GetOuterRect(_Win),
      Wsflags = win_GetState(_Win),
      ClientRCT = rect_GetClient(WsFlags,0,OuterRct),
      ClientRCT = rct(X,Y,Z,W),
      R = W+200,
      win_Move(_Win,rct(X,Y,Z,R)),
      win_SetState(BOTONMENOS, [wsf_visible]),
      win_SetState(BOTONMAS, [wsf_Invisible]),
      !.
%END Diccionario de Sinónimos, idc_más _CtlInfo

%BEGIN Diccionario de Sinónimos, idc_menos _CtlInfo
  dlg_diccionario_de_sinónimos_eh(_Win,e_Control(idc_menos,
_CtrlType,_CtrlWin,_CtlInfo),0):-!,
      BOTONMENOS = win_GetCtlHandle(_Win, idc_menos),
      BOTONMAS = win_GetCtlHandle(_Win, idc_más),
      OuterRCT = win_GetOuterRect(_Win),
      Wsflags = win_GetState(_Win),
      ClientRCT = rect_GetClient(WsFlags,0,OuterRct),
      ClientRCT = rct(X,Y,Z,W),
      R = W-200,
      win_Move(_Win,rct(X,Y,Z,R)),
      win_SetState(BOTONMENOS, [wsf_Invisible]),
      win_SetState(BOTONMAS, [wsf_visible]),
      !.
%END Diccionario de Sinónimos, idc_menos _CtlInfo

%BEGIN Diccionario de Sinónimos, idc_diccionario_de_sinónimos_48
activated
```

```
dlg_diccionario_de_sinónimos_eh(_Win,e_Control(idc_diccionario_
de_sinónimos_48,_CtrlType,_CtrlWin,activated),0):-!,
     dlg_Note("Sinonimos que no son entradas","Estas expresiones
no figuran como entrada en el diccionario."),
     !.
%END Diccionario de Sinónimos, idc_diccionario_de_sinónimos_48
activated

%BEGIN Diccionario de Sinónimos, expresiones_relacionadas
activated

dlg_diccionario_de_sinónimos_eh(_Win,e_Control(expresiones_relac
ionadas,_CtrlType,_CtrlWin,activated),0):-!,
     LISTARELA=win_GetCtlHandle(_Win, expresiones_relacionadas),
     ENTRADA = win_GetCtlHandle(_Win, idc_edit),
     LISTASIN=win_GetCtlHandle(_Win, expresiones_relacionadas),
     LISTAHOM1 = win_GetCtlHandle(_Win, lista_hom1),
     LISTAACEP1 = win_GetCtlHandle(_Win, lista_acep1),
     LISTAACEP2 = win_GetCtlHandle(_Win, lista_acep3),
     LISTAHOM2 = win_GetCtlHandle(_Win, lista_acep4),
     BOTONRELA=win_GetCtlHandle(_Win,idc_expresiones_
          relacionadas),
     IndexSin = lbox_GetSelIndex(LISTARELA),
     Palabra = lbox_GetItem(LISTASIN, IndexSin),
     IndexAcep = lbox_GetSelIndex(LISTAACEP2),
     Item2 = lbox_GetItem(LISTAACEP2, IndexAcep),
     IndexHom = lbox_GetSelIndex(LISTAHOM2),
     Item1 = lbox_GetItem(LISTAHOM2, IndexHom),
     str_int(Item1,HOMINT),
     str_int(Item2,AcepINT),
     H = HOMINT - 1,
     A = AcepINT - 1,
     win_SetText(ENTRADA, Palabra),
     lbox_SetSel(LISTAHOM1, H, 1),
     dlg_diccionario_de_sinónimos_eh(_Win,e_Control(lista_hom1,
          _CtrlType,_CtrlWin,selchanged)),
     lbox_SetSel(LISTAACEP1, A, 1),
     dlg_diccionario_de_sinónimos_eh(_Win,e_Control(lista_acep1,
          _CtrlType,_CtrlWin,selchanged)),
     dlg_diccionario_de_sinónimos_eh(_Win,e_Control(idc_expre
          siones_relacionadas,wc_LBox,BOTONRELA,activated)),
     !.
%END Diccionario de Sinónimos, expresiones_relacionadas
activated

%BEGIN Diccionario de Sinónimos, idc_ordenar_por_similaridad1
_CtlInfo

dlg_diccionario_de_sinónimos_eh(_Win,e_Control(idc_ordenar_por_
similaridad1,_CtrlType,_CtrlWin,_CtlInfo),0):-!,
     dlg_procesando_Create(_Win),
     MEDIDAS = win_GetCtlHandle(_Win, medidas_de_similaridad),
```

```
UMBRAL = win_GetCtlHandle(_Win, idct_0),
ENTRADA = win_GetCtlHandle(_Win, idc_edit),
LISTAHOM1 = win_GetCtlHandle(_Win, lista_hom1),
LISTAACEP1 = win_GetCtlHandle(_Win, lista_acep1),
TEXTOINFO=win_GetCtlHandle(_Win,idct_diccionario_de_
     sinónimos_1),
LISTARELA=win_GetCtlHandle(_Win, expresiones_relacionadas),
LISTAACEP3 = win_GetCtlHandle(_Win, lista_acep3),
LISTAHOM3 = win_GetCtlHandle(_Win, lista_acep4),
TXGRADOR=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_45),
TXVERB = win_GetCtlHandle(_Win, barra_grado_relacionadas),
LISTAFLEX3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_93),
LISTAGRAM3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_91),
LISTATECN3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_89),
LISTADIAL3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_87),
LISTAPREST3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_85),
LISTAPRAG3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_83),
LISTADIAC3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_30),
LISTAOTROS3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_7),
lbox_clear(LISTAFLEX3),
lbox_clear(LISTAGRAM3),
lbox_clear(LISTATECN3),
lbox_clear(LISTADIAL3),
lbox_clear(LISTAPREST3),
lbox_clear(LISTAPRAG3),
lbox_clear(LISTADIAC3),
lbox_clear(LISTAOTROS3),
lbox_clear(LISTARELA),
lbox_clear(LISTAACEP3),
lbox_clear(LISTAHOM3),
win_SetText(TXGRADOR, "0"),
win_SetScrollPos(TXVERB, 2, 100),
IndexMed = lbox_GetSelIndex(MEDIDAS),
Med = lbox_GetItem(MEDIDAS, IndexMed),
Umbstr = win_GetText(UMBRAL),
str_real(Umbstr, Umb),
Palabra = win_GetText(ENTRADA),
Index = lbox_GetSelIndex(LISTAHOM1),
Item1 = lbox_GetItem(LISTAHOM1, Index),
Index2 = lbox_GetSelIndex(LISTAACEP1),
Item2 = lbox_GetItem(LISTAACEP1, Index2),
findall(Sin,sin_acep2(Palabra,Item2,Sin,_,_,Umb,Med,Item1,
     _),SinDes),
findall(Acep2,sin_acep2(Palabra,Item2,_,Acep2,_,Umb,Med,
```

```
          Item1,_),AcepDes),
     findall(Hom2,sin_acep2(Palabra,Item2,_,_,_,Umb,Med,Item1,
          Hom2),HomDes),
     findall(GR,sin_acep2(Palabra,Item2,_,_,GR,Umb,Med,Item1,_),
          GraDes),
     ordenar(SinDes,SinOrd,AcepDes,AcepOrd,HomDes,HomOrd,GraDes,
          _),
     anadir(LISTARELA, SinOrd),
     anadir(LISTAACEP3, AcepOrd),
     anadir(LISTAHOM3, HomOrd),
     win_SetText(TEXTOINFO, "Seleccione una palabra relacionada
          o haga doble clic para incorporarla como entrada."),
     Proc = win_GetActiveWindow(),
     win_Destroy(Proc),
     !.
%END Diccionario de Sinónimos, idc_ordenar_por_similaridad1
_CtlInfo

%BEGIN Diccionario de Sinónimos, expresiones_relacionadas
selchanged

dlg_diccionario_de_sinónimos_eh(_Win,e_Control(expresiones_relac
ionadas,_CtrlType,_CtrlWin,selchanged),0):-!,
     MEDIDAS = win_GetCtlHandle(_Win, medidas_de_similaridad),
     UMBRAL = win_GetCtlHandle(_Win, idct_0),
     ENTRADA = win_GetCtlHandle(_Win, idc_edit),
     LISTASIN=win_GetCtlHandle(_Win, expresiones_relacionadas),
     LISTAHOM1 = win_GetCtlHandle(_Win, lista_hom1),
     LISTAACEP1 = win_GetCtlHandle(_Win, lista_acep1),
     LISTAACEP2 = win_GetCtlHandle(_Win, lista_acep3),
     LISTAHOM2 = win_GetCtlHandle(_Win, lista_acep4),
     LISTAFLEX=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_93),
     LISTAGRAM=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_91),
     LISTATECN=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_89),
     LISTADIAL=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_87),
     LISTAPREST=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_85),
     LISTAPRAG=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_83),
     LISTADIAC=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_30),
     LISTAOTROS=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_7),
     TXGRADO=win_GetCtlHandle(_Win,idc_diccionario_de_sinónimos_
          45),
     TXVERB = win_GetCtlHandle(_Win, barra_grado_relacionadas),
     win_SetText(TXGRADO, ""),
     lbox_clear(LISTAFLEX),
     lbox_clear(LISTAGRAM),
```

```
lbox_clear(LISTATECN),
lbox_clear(LISTADIAL),
lbox_clear(LISTAPREST),
lbox_clear(LISTAPRAG),
lbox_clear(LISTADIAC),
lbox_clear(LISTAOTROS),
IndexMed = lbox_GetSelIndex(MEDIDAS),
Med = lbox_GetItem(MEDIDAS, IndexMed),
Umbstr = win_GetText(UMBRAL),
str_real(Umbstr, Umb),
Palabra = win_GetText(ENTRADA),
Index = lbox_GetSelIndex(LISTAHOM1),
Item1 = lbox_GetItem(LISTAHOM1, Index),
Index2 = lbox_GetSelIndex(LISTAACEP1),
Item2 = lbox_GetItem(LISTAACEP1, Index2),
IndexPal = lbox_GetSelIndex(LISTASIN),
lbox_SetSel(LISTAACEP2, IndexPal, 1),
lbox_SetSel(LISTAHOM2, IndexPal, 1),
Pal2 = lbox_GetItem(LISTASIN, IndexPal),
Acep2 = lbox_GetItem(LISTAACEP2, IndexPal),
Hom2 = lbox_GetItem(LISTAHOM2, IndexPal),
sin_acep2(Palabra,Item2,Pal2,Acep2,Grado,Umb,Med,Item1,
     Hom2),
str_real(GradoStr,Grado),
win_SetText(TXGRADO, GradoStr),
Pos = 100 - round(Grado * 100),
Posicion = val(integer,Pos),
win_SetScrollPos(TXVERB, 2, Posicion),
findall(Flex,flex(Pal2,Acep2,Hom2,Flex),LFlex),
lbox_Add(LISTAFLEX, 0, LFlex),
lbox_SetSel(LISTAFLEX, 0, 1),
findall(Gram,gram(Pal2,Acep2,Hom2,Gram),LGram),
lbox_Add(LISTAGRAM, 0, LGram),
lbox_SetSel(LISTAGRAM, 0, 1),
findall(Tecn,tecn(Pal2,Acep2,Hom2,Tecn),LTecn),
lbox_Add(LISTATECN, 0, LTecn),
lbox_SetSel(LISTATECN, 0, 1),
findall(Dial,dial(Pal2,Acep2,Hom2,Dial),LDial),
lbox_Add(LISTADIAL, 0, LDial),
lbox_SetSel(LISTADIAL, 0, 1),
findall(Prest,prest(Pal2,Acep2,Hom2,Prest),LPrest),
lbox_Add(LISTAPREST, 0, LPrest),
lbox_SetSel(LISTAPREST, 0, 1),
findall(Prag,prag(Pal2,Acep2,Hom2,Prag),LPrag),
lbox_Add(LISTAPRAG, 0, LPrag),
lbox_SetSel(LISTAPRAG, 0, 1),
findall(Diac,diac(Pal2,Acep2,Hom2,Diac),LDiac),
lbox_Add(LISTADIAC, 0, LDiac),
lbox_SetSel(LISTADIAC, 0, 1),
findall(Otros,otros_datos(Pal2,Acep2,Hom2,Otros),LOtros),
lbox_Add(LISTAOTROS, 0, LOtros),
lbox_SetSel(LISTAOTROS, 0, 1),
!.
```

```
%END Diccionario de Sinónimos, expresiones_relacionadas
selchanged

%BEGIN Diccionario de Sinónimos, e_Destroy
  dlg_diccionario_de_sinónimos_eh(_Win,e_Destroy,0):-!,
      retractall(dic(_,_,_,_)),
      retractall(ant(_,_,_,_,_)),
      !.
%END Diccionario de Sinónimos, e_Destroy


%BEGIN Diccionario de Sinónimos, e_Create

dlg_diccionario_de_sinónimos_eh(_Win,e_Create(_CreationData),0):
-!,
      consult("C:\\VIP\\BIN\\WIN\\32\\DICCIONARIO\\DIC.PRO"),
      consult("C:\\VIP\\BIN\\WIN\\32\\Diccionario\\ANT.PRO"),
      !.
%END Diccionario de Sinónimos, e_Create

%BEGIN Diccionario de Sinónimos, idc_expresiones_relacionadas
_CtlInfo

dlg_diccionario_de_sinónimos_eh(_Win,e_Control(idc_expresiones_
relacionadas,_CtrlType,_CtrlWin,_CtlInfo),0):-!,
      dlg_procesando_Create(_Win),
      MEDIDAS = win_GetCtlHandle(_Win, medidas_de_similaridad),
      UMBRAL = win_GetCtlHandle(_Win, idct_0),
      ENTRADA = win_GetCtlHandle(_Win, idc_edit),
      LISTAHOM1 = win_GetCtlHandle(_Win, lista_hom1),
      LISTAACEP1 = win_GetCtlHandle(_Win, lista_acep1),
      LISTARELA=win_GetCtlHandle(_Win, expresiones_relacionadas),
      LISTAACEP3 = win_GetCtlHandle(_Win, lista_acep3),
      LISTAHOM3 = win_GetCtlHandle(_Win, lista_acep4),
      TXGRADOR=win_GetCtlHandle(_Win,idc_diccionario_de_sinónimos
            _45),
      TXVERB = win_GetCtlHandle(_Win, barra_grado_relacionadas),
      LISTAFLEX=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_93),
      LISTAGRAM=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_91),
      LISTATECN=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_89),
      LISTADIAL=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_87),
      LISTAPREST=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_85),
      LISTAPRAG=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_83),
      LISTADIAC=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_30),
      LISTAOTROS=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_7),
```

```
      lbox_clear(LISTAFLEX),
      lbox_clear(LISTAGRAM),
      lbox_clear(LISTATECN),
      lbox_clear(LISTADIAL),
      lbox_clear(LISTAPREST),
      lbox_clear(LISTAPRAG),
      lbox_clear(LISTADIAC),
      lbox_clear(LISTAOTROS),
      lbox_clear(LISTARELA),
      lbox_clear(LISTAACEP3),
      lbox_clear(LISTAHOM3),
      win_SetText(TXGRADOR, "0"),
      win_SetScrollPos(TXVERB, 2, 100),
      IndexMed = lbox_GetSelIndex(MEDIDAS),
      Med = lbox_GetItem(MEDIDAS, IndexMed),
      Umbstr = win_GetText(UMBRAL),
      str_real(Umbstr, Umb),
      Palabra = win_GetText(ENTRADA),
      Index = lbox_GetSelIndex(LISTAHOM1),
      Item1 = lbox_GetItem(LISTAHOM1, Index),
      Index2 = lbox_GetSelIndex(LISTAACEP1),
      Item2 = lbox_GetItem(LISTAACEP1, Index2),
      findall(Pal2,sin_acep2(Palabra,Item2,Pal2,_,_,Umb,Med,
           Item1,_),LPal2),
      findall(Acep2,sin_acep2(Palabra,Item2,_,Acep2,_,Umb,Med,
           Item1,_),LAcep2),
      findall(Hom2,sin_acep2(Palabra,Item2,_,_,_,Umb,Med,Item1,
           Hom2),LHom2),
      lbox_Add(LISTARELA, 0, LPal2),
      lbox_Add(LISTAACEP3, 0, LAcep2),
      lbox_Add(LISTAHOM3, 0, LHom2),
      Proc = win_GetActiveWindow(),
      win_Destroy(Proc),
      !.
%END Diccionario de Sinónimos, idc_expresiones_relacionadas
_CtlInfo

%BEGIN Diccionario de Sinónimos, idc_ordenar_por_similaridad
_CtlInfo

dlg_diccionario_de_sinónimos_eh(_Win,e_Control(idc_ordenar_por_
similaridad,_CtrlType,_CtrlWin,_CtlInfo),0):-!,
      dlg_procesando_Create(_Win),
      MEDIDAS = win_GetCtlHandle(_Win, medidas_de_similaridad),
      UMBRAL = win_GetCtlHandle(_Win, idct_0),
      ENTRADA = win_GetCtlHandle(_Win, idc_edit),
      LISTASIN = win_GetCtlHandle(_Win, lista_palabras2),
      LISTAHOM1 = win_GetCtlHandle(_Win, lista_hom1),
      LISTAACEP1 = win_GetCtlHandle(_Win, lista_acep1),
      LISTAACEP2 = win_GetCtlHandle(_Win, lista_acepciones2),
      LISTAHOM2 = win_GetCtlHandle(_Win, lista_hom2),
      TXGRADO=win_GetCtlHandle(_Win,idct_diccionario_de_sinónimos
           _2),
```

```
TXVERB = win_GetCtlHandle(_Win, barra_grado_sinonimas),
LISTAFLEX2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_65),
LISTAGRAM2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_67),
LISTATECN2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_69),
LISTADIAL2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_71),
LISTAPREST2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_73),
LISTAPRAG2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_75),
LISTADIAC2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_77),
LISTAOTROS2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_80),
lbox_clear(LISTAFLEX2),
lbox_clear(LISTAGRAM2),
lbox_clear(LISTATECN2),
lbox_clear(LISTADIAL2),
lbox_clear(LISTAPREST2),
lbox_clear(LISTAPRAG2),
lbox_clear(LISTADIAC2),
lbox_clear(LISTAOTROS2),
lbox_clear(LISTASIN),
lbox_clear(LISTAACEP2),
lbox_clear(LISTAHOM2),
win_SetText(TXGRADO, "0"),
win_SetScrollPos(TXVERB, 2, 100),
IndexMed = lbox_GetSelIndex(MEDIDAS),
Med = lbox_GetItem(MEDIDAS, IndexMed),
Umbstr = win_GetText(UMBRAL),
str_real(Umbstr, Umb),
Palabra = win_GetText(ENTRADA),
Index = lbox_GetSelIndex(LISTAHOM1),
Item1 = lbox_GetItem(LISTAHOM1, Index),
Index2 = lbox_GetSelIndex(LISTAACEP1),
Item2 = lbox_GetItem(LISTAACEP1, Index2),
findall(Sin,sinonimo(Palabra,Item2,Sin,_,_,Umb,Med,Item1,
     _),SinDes),
findall(Acep2,sinonimo(Palabra,Item2,_,Acep2,_,Umb,Med,
     Item1,_),AcepDes),
findall(Hom2,sinonimo(Palabra,Item2,_,_,_,Umb,Med,Item1,
     Hom2),HomDes),
findall(GR,sinonimo(Palabra,Item2,_,_,GR,Umb,Med,Item1,_),
     GraDes),
ordenar(SinDes,SinOrd,AcepDes,AcepOrd,HomDes,HomOrd,GraDes,
     _),
anadir(LISTASIN, SinOrd),
anadir(LISTAACEP2, AcepOrd),
anadir(LISTAHOM2, HomOrd),
Proc = win_GetActiveWindow(),
```

```
      win_Destroy(Proc),
      !.
%END Diccionario de Sinónimos, idc_ordenar_por_similaridad
_CtlInfo

%BEGIN Diccionario de Sinónimos, lista_palabras2 selchanged

dlg_diccionario_de_sinónimos_eh(_Win,e_Control(lista_palabras2,
_CtrlType,_CtrlWin,selchanged),0):-!,
      MEDIDAS = win_GetCtlHandle(_Win, medidas_de_similaridad),
      UMBRAL = win_GetCtlHandle(_Win, idct_0),
      ENTRADA = win_GetCtlHandle(_Win, idc_edit),
      LISTASIN = win_GetCtlHandle(_Win, lista_palabras2),
      LISTAHOM1 = win_GetCtlHandle(_Win, lista_hom1),
      LISTAACEP1 = win_GetCtlHandle(_Win, lista_acep1),
      LISTAACEP2 = win_GetCtlHandle(_Win, lista_acepciones2),
      LISTAHOM2 = win_GetCtlHandle(_Win, lista_hom2),
      LISTAFLEX=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_65),
      LISTAGRAM=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_67),
      LISTATECN=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_69),
      LISTADIAL=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_71),
      LISTAPREST=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_73),
      LISTAPRAG=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_75),
      LISTADIAC=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_77),
      LISTAOTROS=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_80),
      TXGRADO=win_GetCtlHandle(_Win,idct_diccionario_de_
            sinónimos_2),
      TXVERB = win_GetCtlHandle(_Win, barra_grado_sinonimas),
      lbox_clear(LISTAFLEX),
      lbox_clear(LISTAGRAM),
      lbox_clear(LISTATECN),
      lbox_clear(LISTADIAL),
      lbox_clear(LISTAPREST),
      lbox_clear(LISTAPRAG),
      lbox_clear(LISTADIAC),
      lbox_clear(LISTAOTROS),
      win_SetText(TXGRADO, ""),
      IndexMed = lbox_GetSelIndex(MEDIDAS),
      Med = lbox_GetItem(MEDIDAS, IndexMed),
      Umbstr = win_GetText(UMBRAL),
      str_real(Umbstr, Umb),
      Palabra = win_GetText(ENTRADA),
      Index = lbox_GetSelIndex(LISTAHOM1),
      Item1 = lbox_GetItem(LISTAHOM1, Index),
      Index2 = lbox_GetSelIndex(LISTAACEP1),
```

```
      Item2 = lbox_GetItem(LISTAACEP1, Index2),
      IndexPal = lbox_GetSelIndex(LISTASIN),
      lbox_SetSel(LISTAACEP2, IndexPal, 1),
      lbox_SetSel(LISTAHOM2, IndexPal, 1),
      Pal2 = lbox_GetItem(LISTASIN, IndexPal),
      Acep2 = lbox_GetItem(LISTAACEP2, IndexPal),
      Hom2 = lbox_GetItem(LISTAHOM2, IndexPal),
      sinonimo(Palabra,Item2,Pal2,Acep2,Grado,Umb,Med,Item1,
           Hom2),
      str_real(GradoStr,Grado),
      win_SetText(TXGRADO, GradoStr),
      Pos = 100 - (Grado * 100),
      Posicion = val(integer,Pos),
      win_SetScrollPos(TXVERB, 2, Posicion),
      findall(Flex,flex(Pal2,Acep2,Hom2,Flex),LFlex),
      lbox_Add(LISTAFLEX, 0, LFlex),
      lbox_SetSel(LISTAFLEX, 0, 1),
      findall(Gram,gram(Pal2,Acep2,Hom2,Gram),LGram),
      lbox_Add(LISTAGRAM, 0, LGram),
      lbox_SetSel(LISTAGRAM, 0, 1),
      findall(Tecn,tecn(Pal2,Acep2,Hom2,Tecn),LTecn),
      lbox_Add(LISTATECN, 0, LTecn),
      lbox_SetSel(LISTATECN, 0, 1),
      findall(Dial,dial(Pal2,Acep2,Hom2,Dial),LDial),
      lbox_Add(LISTADIAL, 0, LDial),
      lbox_SetSel(LISTADIAL, 0, 1),
      findall(Prest,prest(Pal2,Acep2,Hom2,Prest),LPrest),
      lbox_Add(LISTAPREST, 0, LPrest),
      lbox_SetSel(LISTAPREST, 0, 1),
      findall(Prag,prag(Pal2,Acep2,Hom2,Prag),LPrag),
      lbox_Add(LISTAPRAG, 0, LPrag),
      lbox_SetSel(LISTAPRAG, 0, 1),
      findall(Diac,diac(Pal2,Acep2,Hom2,Diac),LDiac),
      lbox_Add(LISTADIAC, 0, LDiac),
      lbox_SetSel(LISTADIAC, 0, 1),
      findall(Otros,otros_datos(Pal2,Acep2,Hom2,Otros),LOtros),
      lbox_Add(LISTAOTROS, 0, LOtros),
      lbox_SetSel(LISTAOTROS, 0, 1),
      !.
%END Diccionario de Sinónimos, lista_palabras2 selchanged

%BEGIN Diccionario de Sinónimos, lista_palabras2 activated

dlg_diccionario_de_sinónimos_eh(_Win,e_Control(lista_palabras2,_
CtrlType,_CtrlWin,activated),0):-!,
      ENTRADA = win_GetCtlHandle(_Win, idc_edit),
      LISTASIN = win_GetCtlHandle(_Win, lista_palabras2),
      LISTAHOM1 = win_GetCtlHandle(_Win, lista_hom1),
      LISTAACEP1 = win_GetCtlHandle(_Win, lista_acep1),
      LISTAACEP2 = win_GetCtlHandle(_Win, lista_acepciones2),
      LISTAHOM2 = win_GetCtlHandle(_Win, lista_hom2),
      IndexSin = lbox_GetSelIndex(LISTASIN),
      Palabra = lbox_GetItem(LISTASIN, IndexSin),
```

```
      IndexAcep = lbox_GetSelIndex(LISTAACEP2),
      Item2 = lbox_GetItem(LISTAACEP2, IndexAcep),
      IndexHom = lbox_GetSelIndex(LISTAHOM2),
      Item1 = lbox_GetItem(LISTAHOM2, IndexHom),
      str_int(Item1,HOMINT),
      str_int(Item2,AcepINT),
      H = HOMINT - 1,
      A = AcepINT - 1,
      win_SetText(ENTRADA, Palabra),
      lbox_SetSel(LISTAHOM1, H, 1),
      dlg_diccionario_de_sinónimos_eh(_Win,e_Control(lista_hom1,
            _CtrlType,_CtrlWin,selchanged)),
      lbox_SetSel(LISTAACEP1, A, 1),
      dlg_diccionario_de_sinónimos_eh(_Win,e_Control(lista_acep1,
            _CtrlType,_CtrlWin,selchanged)),
      !.
%END Diccionario de Sinónimos, lista_palabras2 activated

%BEGIN Diccionario de Sinónimos, lista_acep1 selchanged
  dlg_diccionario_de_sinónimos_eh(_Win,e_Control(lista_acep1,
_CtrlType,_CtrlWin,selchanged),0):-!,
      dlg_procesando_Create(_Win),
      MEDIDAS = win_GetCtlHandle(_Win, medidas_de_similaridad),
      UMBRAL = win_GetCtlHandle(_Win, idct_0),
      ENTRADA = win_GetCtlHandle(_Win, idc_edit),
      LISTASIN = win_GetCtlHandle(_Win, lista_palabras2),
      LISTAHOM1 = win_GetCtlHandle(_Win, lista_hom1),
      LISTAACEP1 = win_GetCtlHandle(_Win, lista_acep1),
      LISTAACEP2 = win_GetCtlHandle(_Win, lista_acepciones2),
      LISTAHOM2 = win_GetCtlHandle(_Win, lista_hom2),
      TEXTOINFO=win_GetCtlHandle(_Win,idct_diccionario_de_
            sinónimos_1),
      LISTASINNOENT=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_48),
      LISTARELA=win_GetCtlHandle(_Win, expresiones_relacionadas),
      LISTAACEP3 = win_GetCtlHandle(_Win, lista_acep3),
      LISTAHOM3 = win_GetCtlHandle(_Win, lista_acep4),
      BOTONORD2=win_GetCtlHandle(_Win,idc_ordenar_por_
            similaridad),
      BOTONPR=win_GetCtlHandle(_Win,idc_expresiones_
            relacionadas),
      BOTONORD3=win_GetCtlHandle(_Win,idc_ordenar_por_
            similaridad1),
      TXGRADO=win_GetCtlHandle(_Win,idct_diccionario_de_
            sinónimos_2),
      TXGRADOR=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_45),
      TXVERB = win_GetCtlHandle(_Win, barra_grado_sinonimas),
      TXVERBR = win_GetCtlHandle(_Win, barra_grado_relacionadas),
      LISTAFLEX = win_GetCtlHandle(_Win, lista_flex),
      LISTAGRAM = win_GetCtlHandle(_Win, lista_gram),
      LISTATECN = win_GetCtlHandle(_Win, lista_tecn),
      LISTADIAL = win_GetCtlHandle(_Win, lista_dial),
```

```
LISTAPREST = win_GetCtlHandle(_Win, lista_prest),
LISTAPRAG = win_GetCtlHandle(_Win, lista_prag),
LISTADIAC = win_GetCtlHandle(_Win, lista_diac),
LISTAOTROS = win_GetCtlHandle(_Win, lista_otros),
LISTAFLEX2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_65),
LISTAGRAM2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_67),
LISTATECN2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_69),
LISTADIAL2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_71),
LISTAPREST2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_73),
LISTAPRAG2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_75),
LISTADIAC2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_77),
LISTAOTROS2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_80),
LISTAFLEX3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_93),
LISTAGRAM3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_91),
LISTATECN3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_89),
LISTADIAL3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_87),
LISTAPREST3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_85),
LISTAPRAG3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_83),
LISTADIAC3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_30),
LISTAOTROS3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_7),
lbox_clear(LISTAFLEX2),
lbox_clear(LISTAGRAM2),
lbox_clear(LISTATECN2),
lbox_clear(LISTADIAL2),
lbox_clear(LISTAPREST2),
lbox_clear(LISTAPRAG2),
lbox_clear(LISTADIAC2),
lbox_clear(LISTAOTROS2),
lbox_clear(LISTAFLEX3),
lbox_clear(LISTAGRAM3),
lbox_clear(LISTATECN3),
lbox_clear(LISTADIAL3),
lbox_clear(LISTAPREST3),
lbox_clear(LISTAPRAG3),
lbox_clear(LISTADIAC3),
lbox_clear(LISTAOTROS3),
lbox_clear(LISTASIN),
```

```
lbox_clear(LISTAACEP2),
lbox_clear(LISTAHOM2),
lbox_clear(LISTASINNOENT),
lbox_clear(LISTARELA),
lbox_clear(LISTAACEP3),
lbox_clear(LISTAHOM3),
lbox_clear(LISTAFLEX),
lbox_clear(LISTAGRAM),
lbox_clear(LISTATECN),
lbox_clear(LISTADIAL),
lbox_clear(LISTAPREST),
lbox_clear(LISTAPRAG),
lbox_clear(LISTADIAC),
lbox_clear(LISTAOTROS),
win_SetText(TXGRADO, "0"),
win_SetText(TXGRADOR, "0"),
win_SetScrollPos(TXVERB, 2, 100),
win_SetScrollPos(TXVERBR, 2, 100),
IndexMed = lbox_GetSelIndex(MEDIDAS),
Med = lbox_GetItem(MEDIDAS, IndexMed),
Umbstr = win_GetText(UMBRAL),
str_real(Umbstr, Umb),
Palabra = win_GetText(ENTRADA),
Index = lbox_GetSelIndex(LISTAHOM1),
Item1 = lbox_GetItem(LISTAHOM1, Index),
Index2 = lbox_GetSelIndex(LISTAACEP1),
Item2 = lbox_GetItem(LISTAACEP1, Index2),
findall(Pal2,sinonimo(Palabra,Item2,Pal2,_,_,Umb,Med,Item1,
     _),LPal2),
findall(Acep2,sinonimo(Palabra,Item2,_,Acep2,_,Umb,Med,
     Item1,_),LAcep2),
findall(Hom2,sinonimo(Palabra,Item2,_,_,_,Umb,Med,Item1,
     Hom2),LHom2),
lbox_Add(LISTASIN, 0, LPal2),
lbox_Add(LISTAACEP2, 0, LAcep2),
lbox_Add(LISTAHOM2, 0, LHom2),
findall(Palsne,dic2(Palabra,Item2,Item1,Palsne),LPalsne),
lbox_Add(LISTASINNOENT, 0, LPalsne),
findall(Flex,flex(Palabra,Item2,Item1,Flex),LFlex),
lbox_Add(LISTAFLEX, 0, LFlex),
lbox_SetSel(LISTAFLEX, 0, 1),
findall(Gram,gram(Palabra,Item2,Item1,Gram),LGram),
lbox_Add(LISTAGRAM, 0, LGram),
lbox_SetSel(LISTAGRAM, 0, 1),
findall(Tecn,tecn(Palabra,Item2,Item1,Tecn),LTecn),
lbox_Add(LISTATECN, 0, LTecn),
lbox_SetSel(LISTATECN, 0, 1),
findall(Dial,dial(Palabra,Item2,Item1,Dial),LDial),
lbox_Add(LISTADIAL, 0, LDial),
lbox_SetSel(LISTADIAL, 0, 1),
findall(Prest,prest(Palabra,Item2,Item1,Prest),LPrest),
lbox_Add(LISTAPREST, 0, LPrest),
lbox_SetSel(LISTAPREST, 0, 1),
```

```
      findall(Prag,prag(Palabra,Item2,Item1,Prag),LPrag),
      lbox_Add(LISTAPRAG, 0, LPrag),
      lbox_SetSel(LISTAPRAG, 0, 1),
      findall(Diac,diac(Palabra,Item2,Item1,Diac),LDiac),
      lbox_Add(LISTADIAC, 0, LDiac),
      lbox_SetSel(LISTADIAC, 0, 1),
      findall(Otros,otros_datos(Palabra,Item2,Item1,Otros),
           LOtros),
      lbox_Add(LISTAOTROS, 0, LOtros),
      lbox_SetSel(LISTAOTROS, 0, 1),
      win_SetState(BOTONORD2, [wsf_Enabled]),
      win_SetState(BOTONPR, [wsf_Enabled]),
      win_SetState(BOTONORD3, [wsf_Enabled]),
      win_SetText(TEXTOINFO, "Seleccione otra acepción, haga
           doble clic en una palabra para incorporarla como
           entrada o pinche en el botón ordenar por
           similaridad."),
      Proc = win_GetActiveWindow(),
      win_Destroy(Proc),
      !.
%END Diccionario de Sinónimos, lista_acep1 selchanged

%BEGIN Diccionario de Sinónimos, lista_hom1 selchanged
  dlg_diccionario_de_sinónimos_eh(_Win,e_Control(lista_hom1,
_CtrlType,_CtrlWin,selchanged),0):-!,
      ENTRADA = win_GetCtlHandle(_Win, idc_edit),
      LISTASIN = win_GetCtlHandle(_Win, lista_palabras2),
      LISTAHOM1 = win_GetCtlHandle(_Win, lista_hom1),
      LISTAACEP1 = win_GetCtlHandle(_Win, lista_acep1),
      LISTAACEP2 = win_GetCtlHandle(_Win, lista_acepciones2),
      LISTAHOM2 = win_GetCtlHandle(_Win, lista_hom2),
      TEXTOINFO=win_GetCtlHandle(_Win,idct_diccionario_de_
           sinónimos_1),
      LISTASINNOENT=win_GetCtlHandle(_Win,idc_diccionario_de_
           sinónimos_48),
      LISTARELA=win_GetCtlHandle(_Win, expresiones_relacionadas),
      LISTAACEP3 = win_GetCtlHandle(_Win, lista_acep3),
      LISTAHOM3 = win_GetCtlHandle(_Win, lista_acep4),
      BOTONORD2=win_GetCtlHandle(_Win,idc_ordenar_por_
           similaridad),
      BOTONPR=win_GetCtlHandle(_Win,idc_expresiones_
           relacionadas),
      BOTONORD3=win_GetCtlHandle(_Win,idc_ordenar_por_
           similaridad1),
      TXGRADO=win_GetCtlHandle(_Win,idct_diccionario_de_
           sinónimos_2),
      TXGRADO=win_GetCtlHandle(_Win,idct_diccionario_de_
           sinónimos_2),
      TXGRADOR=win_GetCtlHandle(_Win,idc_diccionario_de_sinónimos
           _45),
      TXVERB = win_GetCtlHandle(_Win, barra_grado_sinonimas),
      TXVERBR = win_GetCtlHandle(_Win, barra_grado_relacionadas),
      LISTAFLEX = win_GetCtlHandle(_Win, lista_flex),
```

```
LISTAGRAM = win_GetCtlHandle(_Win, lista_gram),
LISTATECN = win_GetCtlHandle(_Win, lista_tecn),
LISTADIAL = win_GetCtlHandle(_Win, lista_dial),
LISTAPREST = win_GetCtlHandle(_Win, lista_prest),
LISTAPRAG = win_GetCtlHandle(_Win, lista_prag),
LISTADIAC = win_GetCtlHandle(_Win, lista_diac),
LISTAOTROS = win_GetCtlHandle(_Win, lista_otros),
LISTAFLEX2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_65),
LISTAGRAM2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_67),
LISTATECN2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_69),
LISTADIAL2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_71),
LISTAPREST2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_73),
LISTAPRAG2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_75),
LISTADIAC2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_77),
LISTAOTROS2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_80),
LISTAFLEX3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_93),
LISTAGRAM3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_91),
LISTATECN3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_89),
LISTADIAL3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_87),
LISTAPREST3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_85),
LISTAPRAG3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_83),
LISTADIAC3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_30),
LISTAOTROS3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_7),
lbox_clear(LISTAFLEX),
lbox_clear(LISTAGRAM),
lbox_clear(LISTATECN),
lbox_clear(LISTADIAL),
lbox_clear(LISTAPREST),
lbox_clear(LISTAPRAG),
lbox_clear(LISTADIAC),
lbox_clear(LISTAOTROS),
lbox_clear(LISTAFLEX2),
lbox_clear(LISTAGRAM2),
lbox_clear(LISTATECN2),
lbox_clear(LISTADIAL2),
lbox_clear(LISTAPREST2),
lbox_clear(LISTAPRAG2),
```

```
      lbox_clear(LISTADIAC2),
      lbox_clear(LISTAOTROS2),
      lbox_clear(LISTAFLEX3),
      lbox_clear(LISTAGRAM3),
      lbox_clear(LISTATECN3),
      lbox_clear(LISTADIAL3),
      lbox_clear(LISTAPREST3),
      lbox_clear(LISTAPRAG3),
      lbox_clear(LISTADIAC3),
      lbox_clear(LISTAOTROS3),
      win_SetState(BOTONORD2, [wsf_Disabled]),
      win_SetState(BOTONPR, [wsf_Disabled]),
      win_SetState(BOTONORD3, [wsf_Disabled]),
      lbox_clear(LISTASINNOENT),
      lbox_clear(LISTAACEP1),
      lbox_clear(LISTASIN),
      lbox_clear(LISTAACEP2),
      lbox_clear(LISTAHOM2),
      lbox_clear(LISTARELA),
      lbox_clear(LISTAACEP3),
      lbox_clear(LISTAHOM3),
      win_SetText(TXGRADO, "0"),
      win_SetText(TXGRADOR, "0"),
      win_SetScrollPos(TXVERB, 2, 100),
      win_SetScrollPos(TXVERBR, 2, 100),
      Palabra = win_GetText(ENTRADA),
      Index = lbox_GetSelIndex(LISTAHOM1),
      Item = lbox_GetItem(LISTAHOM1, Index),
      findall(Acep,sin_dic(Palabra,_,Acep,Item),LAcep),
      lbox_Add(LISTAACEP1, 0, LAcep),
      sin_dic(Palabra,_,_,Item),
      win_SetText(TEXTOINFO, "Seleccione una acepción."),
      !.
%END Diccionario de Sinónimos, lista_hom1 selchanged

%BEGIN Diccionario de Sinónimos, idc_edit modified
  dlg_diccionario_de_sinónimos_eh(_Win,e_Control(idc_edit,
_CtrlType,_CtrlWin,modified),0):-!,
      ENTRADA = win_GetCtlHandle(_Win, idc_edit),
      LISTASIN = win_GetCtlHandle(_Win, lista_palabras2),
      LISTAHOM1 = win_GetCtlHandle(_Win, lista_hom1),
      LISTAACEP1 = win_GetCtlHandle(_Win, lista_acep1),
      LISTAACEP2 = win_GetCtlHandle(_Win, lista_acepciones2),
      LISTAHOM2 = win_GetCtlHandle(_Win, lista_hom2),
      TEXTOINFO=win_GetCtlHandle(_Win,idct_diccionario_de_
            sinónimos_1),
      LISTASINNOENT=win_GetCtlHandle(_Win,idc_diccionario_de_
            sinónimos_48),
      LISTARELA=win_GetCtlHandle(_Win, expresiones_relacionadas),
      LISTAACEP3 = win_GetCtlHandle(_Win, lista_acep3),
      LISTAHOM3 = win_GetCtlHandle(_Win, lista_acep4),
      BOTONORD2=win_GetCtlHandle(_Win,idc_ordenar_por_
            similaridad),
```

```
BOTONPR=win_GetCtlHandle(_Win,idc_expresiones_
     relacionadas),
BOTONORD3=win_GetCtlHandle(_Win,idc_ordenar_por_
     similaridad1),
TXGRADO=win_GetCtlHandle(_Win,idct_diccionario_de_
     sinónimos_2),
TXGRADO=win_GetCtlHandle(_Win,idct_diccionario_de_
     sinónimos_2),
TXGRADOR=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_45),
TXVERB = win_GetCtlHandle(_Win, barra_grado_sinonimas),
TXVERBR = win_GetCtlHandle(_Win, barra_grado_relacionadas),
LISTAFLEX = win_GetCtlHandle(_Win, lista_flex),
LISTAGRAM = win_GetCtlHandle(_Win, lista_gram),
LISTATECN = win_GetCtlHandle(_Win, lista_tecn),
LISTADIAL = win_GetCtlHandle(_Win, lista_dial),
LISTAPREST = win_GetCtlHandle(_Win, lista_prest),
LISTAPRAG = win_GetCtlHandle(_Win, lista_prag),
LISTADIAC = win_GetCtlHandle(_Win, lista_diac),
LISTAOTROS = win_GetCtlHandle(_Win, lista_otros),
LISTAFLEX2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_65),
LISTAGRAM2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_67),
LISTATECN2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_69),
LISTADIAL2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_71),
LISTAPREST2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_73),
LISTAPRAG2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_75),
LISTADIAC2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_77),
LISTAOTROS2=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_80),
LISTAFLEX3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_93),
LISTAGRAM3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_91),
LISTATECN3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_89),
LISTADIAL3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_87),
LISTAPREST3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_85),
LISTAPRAG3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_83),
LISTADIAC3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_30),
LISTAOTROS3=win_GetCtlHandle(_Win,idc_diccionario_de_
     sinónimos_7),
lbox_clear(LISTAFLEX),
```

```
        lbox_clear(LISTAGRAM),
        lbox_clear(LISTATECN),
        lbox_clear(LISTADIAL),
        lbox_clear(LISTAPREST),
        lbox_clear(LISTAPRAG),
        lbox_clear(LISTADIAC),
        lbox_clear(LISTAOTROS),
        lbox_clear(LISTAFLEX2),
        lbox_clear(LISTAGRAM2),
        lbox_clear(LISTATECN2),
        lbox_clear(LISTADIAL2),
        lbox_clear(LISTAPREST2),
        lbox_clear(LISTAPRAG2),
        lbox_clear(LISTADIAC2),
        lbox_clear(LISTAOTROS2),
        lbox_clear(LISTAFLEX3),
        lbox_clear(LISTAGRAM3),
        lbox_clear(LISTATECN3),
        lbox_clear(LISTADIAL3),
        lbox_clear(LISTAPREST3),
        lbox_clear(LISTAPRAG3),
        lbox_clear(LISTADIAC3),
        lbox_clear(LISTAOTROS3),
        win_SetText(TEXTOINFO, "Introduzca una palabra."),
        win_SetState(BOTONORD2, [wsf_Disabled]),
        win_SetState(BOTONPR, [wsf_Disabled]),
        win_SetState(BOTONORD3, [wsf_Disabled]),
        lbox_clear(LISTAHOM1),
        lbox_clear(LISTAACEP1),
        lbox_clear(LISTASIN),
        lbox_clear(LISTAACEP2),
        lbox_clear(LISTAHOM2),
        lbox_clear(LISTASINNOENT),
        lbox_clear(LISTARELA),
        lbox_clear(LISTAACEP3),
        lbox_clear(LISTAHOM3),
        win_SetText(TXGRADO, "0"),
        win_SetText(TXGRADOR, "0"),
        win_SetScrollPos(TXVERB, 2, 100),
        win_SetScrollPos(TXVERBR, 2, 100),
        Palabra = win_GetText(ENTRADA),
        sin_dic(Palabra,_,_,_),
        findall(Homo,sin_dic(Palabra,_,"1",Homo),LHomo),
        lbox_Add(LISTAHOM1, -1, LHomo),
        lbox_SetSel(LISTAHOM1, 0, 1),
        Item = lbox_GetItem(LISTAHOM1, 0),
        findall(Acep,sin_dic(Palabra,_,Acep,Item),LAcep),
        lbox_Add(LISTAACEP1, -1, LAcep),
        win_SetText(TEXTOINFO, "Seleccione una palabra homógrafa o
             una acepción."),
        !.
%END Diccionario de Sinónimos, idc_edit modified
```

```
%BEGIN Diccionario de Sinónimos, medidas_de_similaridad
selchanged
  dlg_diccionario_de_sinónimos_eh(_Win,e_Control(medidas_de_
similaridad,_CtrlType,_CtrlWin,selchanged),0):-!,
     LISTASIN = win_GetCtlHandle(_Win, lista_palabras2),
     LISTAACEP2 = win_GetCtlHandle(_Win, lista_acepciones2),
     LISTAHOM2 = win_GetCtlHandle(_Win, lista_hom2),
     TEXTOINFO=win_GetCtlHandle(_Win,idct_diccionario_de_
          sinónimos_1),
     LISTASINNOENT=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_48),
     LISTARELA=win_GetCtlHandle(_Win, expresiones_relacionadas),
     LISTAACEP3 = win_GetCtlHandle(_Win, lista_acep3),
     LISTAHOM3 = win_GetCtlHandle(_Win, lista_acep4),
     TXGRADO=win_GetCtlHandle(_Win,idct_diccionario_de_sinónimos
          _2),
     TXGRADOR=win_GetCtlHandle(_Win,idc_diccionario_de_sinónimos
          _45),
     TXVERB = win_GetCtlHandle(_Win, barra_grado_sinonimas),
     TXVERBR = win_GetCtlHandle(_Win, barra_grado_relacionadas),
     LISTAFLEX2=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_65),
     LISTAGRAM2=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_67),
     LISTATECN2=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_69),
     LISTADIAL2=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_71),
     LISTAPREST2=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_73),
     LISTAPRAG2=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_75),
     LISTADIAC2=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_77),
     LISTAOTROS2=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_80),
     LISTAFLEX3=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_93),
     LISTAGRAM3=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_91),
     LISTATECN3=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_89),
     LISTADIAL3=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_87),
     LISTAPREST3=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_85),
     LISTAPRAG3=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_83),
     LISTADIAC3=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_30),
     LISTAOTROS3=win_GetCtlHandle(_Win,idc_diccionario_de_
          sinónimos_7),
     lbox_clear(LISTAFLEX2),
```

```
      lbox_clear(LISTAGRAM2),
      lbox_clear(LISTATECN2),
      lbox_clear(LISTADIAL2),
      lbox_clear(LISTAPREST2),
      lbox_clear(LISTAPRAG2),
      lbox_clear(LISTADIAC2),
      lbox_clear(LISTAOTROS2),
      lbox_clear(LISTAFLEX3),
      lbox_clear(LISTAGRAM3),
      lbox_clear(LISTATECN3),
      lbox_clear(LISTADIAL3),
      lbox_clear(LISTAPREST3),
      lbox_clear(LISTAPRAG3),
      lbox_clear(LISTADIAC3),
      lbox_clear(LISTAOTROS3),
      win_SetText(TXGRADO, "0"),
      win_SetText(TXGRADOR, "0"),
      win_SetScrollPos(TXVERB, 2, 100),
      win_SetScrollPos(TXVERBR, 2, 100),
      win_SetText(TEXTOINFO, "Introduzca una palabra o selecione
           items en las listas Hom o Acep."),
      lbox_clear(LISTASIN),
      lbox_clear(LISTAACEP2),
      lbox_clear(LISTAHOM2),
      lbox_clear(LISTASINNOENT),
      lbox_clear(LISTARELA),
      lbox_clear(LISTAACEP3),
      lbox_clear(LISTAHOM3),
      !.
%END Diccionario de Sinónimos, medidas_de_similaridad selchanged


%BEGIN Diccionario de Sinónimos, umbral _CtlInfo

dlg_diccionario_de_sinónimos_eh(_Win,e_Control(umbral,_CtrlType,
_CtrlWin,_CtlInfo),0):-!,
      BARRA = win_GetCtlHandle(_Win, umbral),
      TEXTO = win_GetCtlHandle(_Win, idct_0),
      LISTASIN = win_GetCtlHandle(_Win, lista_palabras2),
      LISTAACEP2 = win_GetCtlHandle(_Win, lista_acepciones2),
      LISTAHOM2 = win_GetCtlHandle(_Win, lista_hom2),
      TEXTOINFO=win_GetCtlHandle(_Win,idct_diccionario_de_
           sinónimos_1),
      LISTASINNOENT=win_GetCtlHandle(_Win,idc_diccionario_de_
           sinónimos_48),
      LISTARELA=win_GetCtlHandle(_Win, expresiones_relacionadas),
      LISTAACEP3 = win_GetCtlHandle(_Win, lista_acep3),
      TXGRADO=win_GetCtlHandle(_Win,idct_diccionario_de_
           sinónimos_2),
      LISTAHOM3 = win_GetCtlHandle(_Win, lista_acep4),
      TXGRADO=win_GetCtlHandle(_Win,idct_diccionario_de_sinónimos
           _2),
      TXGRADOR=win_GetCtlHandle(_Win,idc_diccionario_de_sinónimos
```

```
      _45),
TXVERB = win_GetCtlHandle(_Win, barra_grado_sinonimas),
TXVERBR = win_GetCtlHandle(_Win, barra_grado_relacionadas),
LISTAFLEX2=win_GetCtlHandle(_Win,idc_diccionario_de_
      sinónimos_65),
LISTAGRAM2=win_GetCtlHandle(_Win,idc_diccionario_de_
      sinónimos_67),
LISTATECN2=win_GetCtlHandle(_Win,idc_diccionario_de_
      sinónimos_69),
LISTADIAL2=win_GetCtlHandle(_Win,idc_diccionario_de_
      sinónimos_71),
LISTAPREST2=win_GetCtlHandle(_Win,idc_diccionario_de_
      sinónimos_73),
LISTAPRAG2=win_GetCtlHandle(_Win,idc_diccionario_de_
      sinónimos_75),
LISTADIAC2=win_GetCtlHandle(_Win,idc_diccionario_de_
      sinónimos_77),
LISTAOTROS2=win_GetCtlHandle(_Win,idc_diccionario_de_
      sinónimos_80),
LISTAFLEX3=win_GetCtlHandle(_Win,idc_diccionario_de_
      sinónimos_93),
LISTAGRAM3=win_GetCtlHandle(_Win,idc_diccionario_de_
      sinónimos_91),
LISTATECN3=win_GetCtlHandle(_Win,idc_diccionario_de_
      sinónimos_89),
LISTADIAL3=win_GetCtlHandle(_Win,idc_diccionario_de_
      sinónimos_87),
LISTAPREST3=win_GetCtlHandle(_Win,idc_diccionario_de_
      sinónimos_85),
LISTAPRAG3=win_GetCtlHandle(_Win,idc_diccionario_de_
      sinónimos_83),
LISTADIAC3=win_GetCtlHandle(_Win,idc_diccionario_de_
      sinónimos_30),
LISTAOTROS3=win_GetCtlHandle(_Win,idc_diccionario_de_
      sinónimos_7),
lbox_clear(LISTAFLEX2),
lbox_clear(LISTAGRAM2),
lbox_clear(LISTATECN2),
lbox_clear(LISTADIAL2),
lbox_clear(LISTAPREST2),
lbox_clear(LISTAPRAG2),
lbox_clear(LISTADIAC2),
lbox_clear(LISTAOTROS2),
lbox_clear(LISTAFLEX3),
lbox_clear(LISTAGRAM3),
lbox_clear(LISTATECN3),
lbox_clear(LISTADIAL3),
lbox_clear(LISTAPREST3),
lbox_clear(LISTAPRAG3),
lbox_clear(LISTADIAC3),
lbox_clear(LISTAOTROS3),
win_SetText(TXGRADO, "0"),
win_SetText(TXGRADOR, "0"),
```

```
      win_SetScrollPos(TXVERB, 2, 100),
      win_SetScrollPos(TXVERBR, 2, 100),
      win_SetText(TEXTOINFO, "Introduzca una palabra o selecione
           items en las listas Hom o Acep."),
      Pos = win_GetScrollPos(BARRA,sb_Ctl),
      str_int(X,POS),
      str_real(X,Rpos),
      Resul = Rpos/100,
      str_real(Strpos,Resul),
      win_SetText(TEXTO, Strpos),
      lbox_clear(LISTASIN),
      lbox_clear(LISTAACEP2),
      lbox_clear(LISTAHOM2),
      lbox_clear(LISTASINNOENT),
      lbox_clear(LISTARELA),
      lbox_clear(LISTAACEP3),
      lbox_clear(LISTAHOM3),
      !.
%END Diccionario de Sinónimos, umbral _CtlInfo


  dlg_diccionario_de_sinónimos_eh(_,_,_):-!,fail.

%END_DLG Diccionario de Sinónimos
```

Código del cuadro de diálogo "buscando datos"

```
%BEGIN_DLG Procesando
/************************************************************
***********
     Creation and event handling for dialog: Procesando
************************************************************
**********/

CONSTANTS

%BEGIN   Procesando,   CreateParms,   21:49:11-14.12.2000,   Code
automatically updated!
  dlg_procesando_ResID = idd_procesando
  dlg_procesando_DlgType = wd_Modeless
  dlg_procesando_Help = contents
%END Procesando, CreateParms

PREDICATES

  dlg_procesando_eh : EHANDLER

CLAUSES

  dlg_procesando_Create(Parent):-
```

```
%MARK Procesando, new variables

     dialog_CreateModeless(Parent,dlg_procesando_ResID,"",
          [
%BEGIN   Procesando,   ControlList,   21:49:11-14.12.2000,   Code
automatically updated!
%END Procesando, ControlList
          ],
          dlg_procesando_eh,0).

%MARK Procesando, new events

  dlg_procesando_eh(_,_,_):-!,fail.

%END_DLG Procesando
```