Un formalismo para la extracción de información semántica en textos matemáticos

Año 1995

Autor: Julia Díaz García

Director: Pilar Rodríguez Marín

Departamento de Ingeniería Informática
Universidad Autónoma de Madrid

Memoria presentada para optar al grado de Doctor en Ingeniería Informática

Agradecimientos

Deseo expresar mi agradecimiento en primer lugar a Pilar Rodríguez Marín, por el trabajo y apoyo que ha dedicado al desarrollo de esta tesis como directora de la misma, y a Roberto Moriyón por su ayuda constante y sus valiosos consejos a lo largo del desarrollo del proyecto Prógenes.

También quiero agradecer su ayuda a mis compañeros de Prógenes, especialmente a Julio Gonzalo. Sin las provechosas discusiones que hemos tenido, su labor técnica y buen hacer no hubiera sido posible realizar este trabajo. Gracias también a Pablo Castells, Joaquín Regodón, Francisco Saiz y Montse Serrano.

Y, muy especialmente, gracias a Chelo Rodríguez por haberme ayudado a poner los puntos sobre las íes y a mejorar la redacción de esta tesis, a Luis Sopeña por sus valiosos comentarios y a José María Villanueva y Pedro Pascual por haberme soportado con infinita paciencia y haber insistido tanto en la realización de este trabajo. Quiero resaltar también el apoyo que he recibido de Juan y de mis padres.

Mis agradecimientos finales al Instituto de Ingeniería del Conocimiento, por la disposición de medios técnicos que han puesto a mi alcance, y a todos mis compañeros. Que nadie se sienta olvidado.

Contenido

Introducción 1

Capítulo 1 5

1.0 Representación de información semántica 7
1.1 Objetivos de las teorías semánticas 8
1.2 Interpretación semántica e Inteligencia Artificial 14
1.2.1 Representaciones lógicas 16
1.2.2 Estructuras del conocimiento 24
1.3 Recapitulación 29

Capítulo 2 31

2.0 Marco de la investigación: Proyecto Prógenes 33
2.1 Prógenes: Descripción del Proyecto 35
2.2 Sistemas de referencia: CAPRATE y MECHO 38
2.3 Características del dominio 41
2.4 La interfaz. Estrategia lingüística 45
2.4.1 La gramática 45
2.4.2 El parser 49
2.4.3 El diccionario 52
2.5 Implementación de la interfaz. CLOS 55
2.5.1 Herencia y gramática 57
2.5.2 Uso de clases en Prógenes. Herencia 60
2.5.3 Uso de métodos en Prógenes. Gramática 63

Capítulo 3 67

3.0 Semántica léxica en Prógenes 69
3.1 Elementos de la Base de Conocimientos 70
3.1.1 Objetos 70
3.1.2 Funciones 74
3.1.3 Casos particulares: funciones de asignación 77
3.2 Jerarquía de tipos 80
3.2.1 Jerarquía de tipos: formalización algebraica 82
3.2.2 Jerarquía de tipos: formalización basada en el orden 83
3.2.3 Jerarquía de tipos: completitud del conjunto 87
3.3 Descripción del lenguaje formal Prógenes 88
3.3.1 Sintaxis general sin tipos 88
3.3.2 Sintaxis general con tipos explícitos 91
3.4 Las expresiones semánticas 94
3.5 Relación entre la BC y las Expresiones Semánticas 96

Capítulo 4 107

4.0 Semántica composicional en Prógenes 109
4.1 Mecanismos de combinación semántica 110
4.1.1 Reglas semánticas en Prógenes. 110
4.2 Tratamiento de la anáfora en Prógenes 113
4.2.1 Resolución de referentes en Prógenes: el determinante 116
4.3 Ejemplo de funcionamiento: Problema 1 119
4.3.1 Léxico estático 119
4.3.2 Léxico dinámico 123
4.3.3 Análisis parciales 123
4.3.4 Otras reglas en Prógenes 128

Conclusiones 129

Apéndices 135

5.0 Apéndice A. Ejemplos del léxico 137

6.0 Apéndice B. Ejemplos de enunciados 145

7.0 Apéndice C. Jerarquía de tipos 157

Referencias 159

Bibliografía 169

Introducción

El trabajo de investigación que se presenta ha sido desarrollado en el marco del proyecto Prógenes [(Castells et al., 1991), (Castells et al., 1992), (Castells et al., 1992b)].[1]

Prógenes (**Proof generator expert system**) es un proyecto de investigación básica dedicado al desarrollo de un sistema capaz de resolver problemas enunciados en lenguaje natural. El tipo de problemas que se pueden resolver es muy general, sin limitación en cuanto a la temática, con la condición de que se puedan reducir a ecuaciones. Esta es una condición muy general que cumplen una gran parte de los problemas que se plantean a matemáticos, físicos e ingenieros. Entre ellos podemos citar los relacionados con objetos geométricos, funciones, cálculo diferencial, así como otros que involucran conceptos más sofisticados relacionados con estos temas. También cumplen esta condición algunos tipos de problemas de mecánica y termodinámica, que implican un conocimiento profundo del dominio.

En un principio se podría suponer que la entrada del sistema fuesen los enunciados de los problemas expresados en un lenguaje formal determinado, pero el objetivo es más ambicioso, ya que se trata de porporcionarle exclusivamente la misma información, y en la misma forma, que a cualquier persona que tuviera que resolver el problema. Por lo tanto se ha incorporado al sistema una interfaz [(Díaz y Rodríguez-Marín, 1992)] capaz de entender el texto de los enunciados expresados en lenguaje natural, español, e interpretarlos según una semántica [(Díaz et al., 1993), (Díaz et al., 1993b), (Gonzalo, Rodríguez-Marín y Díaz, 1992)] que refleja conceptos y relaciones existentes en esta área de conocimientos.

[1] Financiado por el Plan Nacional de Investigación, Programa Nacional de Tecnología de Información y de las Comunicaciones, Proyecto 352.90.

Con este planteamiento, la interfaz se convierte en un elemento clave para el buen fin del sistema, ya que la resolución de los problemas [(Castells, 1994), (Saiz, 1994)] tiene lugar a partir de la expresión formal que se le suministra y que, a su vez, es el resultado del análisis semántico del texto de origen.

El objetivo de esta tesis ha sido *diseñar e implementar un formalismo semántico que represente el texto inicial de los enunciados, utilizando mecanismos composicionales generales, resolviendo ambigüedades léxicas y efectuando un tratamiento de la complejidad semántica.*

Sin embargo, el propósito de este trabajo no ha sido que el formalismo sea general para todos los dominios posibles (con cualquier lenguaje natural), sino que en todo momento se han tenido en cuenta las características que ofrecía el dominio restringido de Prógenes. La principal de ellas es que el conocimiento subyacente se puede representar mediante una taxonomía jerarquizada de los objetos involucrados y de sus relaciones.

Estos objetos y relaciones se han almacenado en una Base de Conocimiento (BC), que se ha definido como el nexo de unión entre el módulo que resuelve los enunciados y la interfaz que los procesa. Dicha BC ha sido uno de los pilares de este trabajo, puesto que (1) sobre ella se ha definido un lenguaje formal que ha sido empleado para establecer el contenido semántico de cada palabra de forma aislada, de porciones de texto o de un enunciado completo, (2) ha permitido clasificar los objetos del dominio mediante una tipología que respeta una estructura de retículo completo y (3) ha permitido resolver los problemas de equivalencia semántica que aparecen en los textos.

Para efectuar la interpretación semántica de los textos, se han abordado problemas de polisemia, palabras que tienen varios significados, homonimia, palabras que se escriben o pronuncian como otras teniendo distinto sentido, y ambigüedad sintáctica, palabras o porciones de texto que pueden tener distintos papeles sintácticos. Para ello hemos hecho uso del análisis del contexo, de indicaciones locales dentro de la propia oración, de la jerarquía de los objetos y las relaciones definidas por su estructura, y de atributos multivaluados para la definición semántica de los elementos del léxico, entendiendo por léxico las palabras que puedan aparecer en los enunciados o los análisis parciales o completos.

Los mecanismos semánticos implementados sobre la semántica léxica de Prógenes se basan en la composicionalidad. Estos mecanismos nos han permitido

obtener, de forma elegante, expresiones sobre las que hemos resuelto algunas complejidades semánticas, como la adecuada resolución de referentes.

Cabe observar que el contenido de este trabajo se complementa con la tesis doctoral *Una aproximación orientada a objetos para el tratamiento de excepciones en gramáticas*, de Julio Gonzalo (Gonzalo, en preparación), en la que se describe la teoría de funciones genéricas para tratamiento de excepciones empleando técnicas de programación orientada a objetos. Ambas son el resultado del trabajo en equipo del proyecto Prógenes, desarrollado en el Instituto de Ingeniería del Conocimiento de la Universidad Autónoma de Madrid. Aunque cada una de ellas trata en profundidad aspectos distintos del sistema, ambas comparten inevitablemente áreas comunes.

Aunque se mencionarán diversos ejemplos a lo largo de esta memoria, se utilizarán como referencia los siguientes problemas:

Problema 1:

> "Escribir la ecuación de la recta paralela a 3x-5y+8=0 y que pasa por el punto (-3,2)."

Problema 2:

> "Decimos que una función f:R → R es par si f(x) = f(-x) para todo x ∈R. Demostrar que si una función f es par f'(0) =0."

Problema 3:

> "Hallar el punto donde se intersectan las rectas 2x+7y+31=0 y x-2y+7=0, y determinar su ángulo de intersección."

Problema 4

> "Sean a y b números no negativos. Demostrar que si $a^2 =< b^2$ entonces a =< b. INDICACION: $b^2-a^2=(b+a)(b-a)$."

El resto de este trabajo ha sido organizado como sigue. En el capítulo **1**, *"Representación de información semántica"*, se examinan, centrándonos en el dominio de Prógenes, el tipo de problemas o complejidades que se deben tener en cuenta para realizar una adecuada representación semántica y los mecanismos más adecuados

para abordar este problema a partir de elementos aislados y posteriormente generalizando a todo el texto. Se estudian también algunos de los formalismos existentes.

En el capítulo 2, *"Marco de la investigación: Proyecto Prógenes"*, se describen las características generales del sistema, centrándonos en los aspectos relacionados con la interfaz para el procesamiento del texto de los problemas. También mencionaremos los proyectos CAPRATE y MECHO, que han abordado problemas similares incorporando interfaces a sistemas de enseñanza o demostración.

En el capítulo 3, *"Semántica léxica en Prógenes"*, se determinará la forma de las expresiones semánticas, la caracterización formal de los objetos que sirven como representaciones semánticas y el tipo de asociación que se puede establecer entre las expresiones con sus representaciones correspondientes.

En el capítulo 4, *"Semántica composicional en Prógenes"* se describen los mecanismos composicionales del proceso de combinación semántica y el tratamiento que se ha dado a algunas complejidades semánticas, como es la anáfora.

Se presentan finalmente las conclusiones a nuestro trabajo, algunos apéndices que contienen información y datos de interés, las referencias y bibliografía auxiliar que también ha sido de gran utilidad en la elaboración de este trabajo.

En esta memoria se ha utilizado la notación que habitualmente se emplea en la materia. La gran mayoría de los términos han sido traducidos al español, pero hay otros que se ha preferido mantener en el original (inglés), por razones de claridad. Pedimos disculpas por ello.

Capítulo 1

1.0 Representación de información semántica

Una teoría semántica se pronuncia, implícita o explícitamente, respecto de la naturaleza fundamental de las representaciones semánticas y de sus relaciones con aquello sobre lo que la lengua nos permite hablar. Podemos considerar que las representaciones semánticas están directamente relacionadas con aspectos del mundo, las cosas sobre las que hablamos. Como se ha comentado en la Introducción, en Prógenes las representaciones semánticas y sus relaciones se definen a partir de una base de conocimiento en la que subyacen todos los conceptos y relaciones del mundo, en este caso el definido por problemas que se pueden reducir a ecuaciones. Las interpretaciones semánticas deben por tanto representar, de forma unívoca e independiente del lenguaje natural utilizado, las relaciones que se circunscriben al ámbito de los conceptos matemáticos.

Antes de decidir qué tipo de representación semántica sería la más adecuada, es preceptivo examinar el tipo de problemas o complejidades que se deben tener en cuenta. Una vez analizado el problema, deberá ser abordado primero viendo cómo se resolvería a partir de elementos aislados, y posteriormente esbozando las representaciones subyacentes en la estructura profunda de la oración. Es entonces cuando se puede hablar de cómo resolverlo.

En la primera sección de este capítulo se verán cuáles son los objetivos de una teoría semántica, se esbozarán el tipo de problemas que se deben abordar teniendo en cuenta siempre los textos utilizados en Prógenes. En la segunda sección se resumirán algunas de las teorías más representativas.

1.1 Objetivos de las teorías semánticas

Normalmente, el término semántica cubre un amplio abanico de cuestiones relacionadas con el significado, la significación, la interpretación y la comprensión del lenguaje. Sin embargo, en este apartado no es nuestra intención examinar en profundidad todos los aspectos y características de las teorías semánticas, sino sólo revisar aquellas que resultan relevantes para el propósito de nuestro trabajo, que se centra en la extracción de información de textos de problemas matemáticos.

Se puede considerar que el principal objetivo descriptivo de una teoría semántica es *dar cuenta de la estructura semántica de una lengua*, de las propiedades y relaciones que mantienen las expresiones en virtud de su significado (Ladusaw, 1990). Por analogía con la teoría sintáctica, es un estudio de parte de la competencia lingüística de un hablante nativo, básicamente el conocimiento que subyace a la *competencia semántica*. Como tal, presupone un estudio de la sintaxis de una lengua y predice juicios basados en el significado sobre las relaciones semánticas entre sus expresiones.

Entre las relaciones de las que hay que dar cuenta destacan la **equivalencia semántica** (o paráfrasis) y la **consecuencia semántica** (o entrañamiento). Por ejemplo, una teoría semántica adecuada para el dominio de Prógenes explicaría el hecho de que las oraciones en (1.1) son paráfrasis la una de la otra, de que las oraciones de (1.2) son lógicamente equivalentes y de que las oraciones (a) en (1.3) y (1.4) tienen a las oraciones (b) como consecuencias semánticas:

(1.1) a. El punto (2,1,1) pertenece al plano 2x-5y+3z=2.

b. El plano 2x-5y+3z=2 pasa por el punto (2,1,1).

(1.2) a. Las funciones f y g no son inyectivas ni biyectivas.

b. Las funciones f y g no son inyectivas y no son biyectivas.

(1.3) a. Las funciones f y g son continuas.

b. La función f es continua.

(1.4) a. La siguiente es la ecuación canónica de ...

b. La siguiente es la ecuación de ...

Es evidente que las relaciones de equivalencia semántica y de consecuencia no pueden definirse directamente sobre sus estructuras sintácticas (superficiales). Por ejemplo, aunque los pares de oraciones en (1.5) - (1.8) sean, de alguna manera, sintácticamente paralelas a las de (1.1) - (1.4), las relaciones semánticas entre ellas son diferentes:

(1.5) a. Ningún punto pertenece al plano 2x-5y+3z=2.

b. Algún plano pasa por algún punto.

(1.6) a. Las funciones f y g son inyectivas o biyectivas.

b. Las funciones f y g son inyectivas y son biyectivas.

(1.7) a. Las rectas r y s son paralelas.

b. La recta r es paralela.

(1.8) a. El siguiente es el punto de intersección de las rectas r y s.

b. El siguiente es el punto de las rectas r y s.

Para dar cuenta de estas relaciones, la teoría semántica debe proporcionar la proyección de expresiones sobre objetos que les sirvan como interpretaciones. A estas interpretaciones se las denomina **representaciones semánticas**. Las relaciones de equivalencia y consecuencia se definen formalmente sobre esas representaciones. Así, dos expresiones de la lengua son equivalentes si sus representaciones semánticas son equivalentes.

La equivalencia semántica de expresiones se puede representar de dos formas: (1) permitiendo que la proyección de la sintaxis les asigne la misma representación semántica o (2) dada una relación de equivalencia en las representaciones semánticas, asignándoles representaciones semánticas diferentes pero equivalentes. La manera de representar nuestras intuiciones sobre la equivalencia semántica, cuáles son los puntos de contacto entre la sintaxis y la semántica, es un eje importante a lo largo del cual difieren unas teorías y otras. En concreto, suelen diferir respecto a la prominencia concedida a aspectos de las representaciones semánticas y a la relación de consecuencia que se define sobre ellas. Por ejemplo, si las representaciones semánticas son fórmulas de un lenguaje lógico con una interpretación explícita como la lógica de predicados, entonces la equivalencia de la represen-

tación semántica es equivalencia lógica. Dos oraciones son equivalentes sólo si las fórmulas que les sirven de representaciones semánticas tienen siempre el mismo valor de verdad en cualquier circunstancia dada. Así, por ejemplo, se podría predecir que las oraciones de (1.9) son equivalentes si se les asigna las dos fórmulas de (1.10) como representaciones semánticas, ignorando la semántica interna del sintagma verbal:

(1.9) a. No todas las aplicaciones son inyectivas.

b. Algunas aplicaciones no son inyectivas.

(1.10) a. $\neg$ ($\forall$ x) [aplicacion (x) -> inyectiva (x)]

b. ($\exists$ x) [aplicacion (x) $\wedge$ $\neg$ inyectiva (x)]

El presupuesto de que las representaciones semánticas son fórmulas lógicas de una lógica definida independientemente permite a la teoría incorporar todas las equivalencias lógicas conocidas que garantizan que las dos fórmulas de (1.10) sean equivalentes. No hay necesidad de restringir la proyección sintaxis-semántica como para proyectar exactamente (1.9a) y (1.9b) en la misma fórmula. Su equivalencia, la equivalencia de las oraciones de (1.2) y la no equivalencia de las oraciones de (1.6) se deriva inmediatamente de conocidas leyes de la lógica.

Otro caso diferente presentan las oraciones de (1.1), que son representativas de aquellas equivalencias que se derivan de principios de la estructura de la oración. Una opción para representar esa equivalencia es proporcionar representaciones semánticas que, aún siendo distintas, sean equivalentes, proporcionando fórmulas como las de (1.11a) y (1.11b), respectivamente:

(1.11) a. Pertenecer (Punto, Plano).

b. Pasar-por (Plano, Punto).

La equivalencia entre (1.1a) y (1.1b) se reduce así a la equivalencia entre (1.11a) y (1.11b), garantizada por el requisito de que *"Pertenecer"* vale para el par ordenado *"(x , y)"* si y sólo si *"Pasar-por"* vale para *"(y, x)"*. Esta estrategia mantiene una

relación bastante directa entre la estructura sintáctica de la oración y su representación semántica.

Otra posibilidad sería proyectar ambas oraciones en una única expresión semántica, es decir que ambas se proyectaran directamente en (1.14a). Esta estrategia es adoptada frecuentemente por los análisis de oraciones que sólo se diferencian en la forma en la que los *argumentos lógicos* de sus predicados se relacionan con las funciones gramaticales desempeñadas por los sintagmas nominales que les sirven de expresión. Sin embargo, expresar cómo se asocian los argumentos lógicos de un predicado con los complementos de un verbo en una oración requiere una forma gramaticalmente neutra de nombrar a los argumentos lógicos. El modo más común de hacerlo es asumir un conjunto de *papeles semánticos* o *papeles temáticos*. En el ejemplo anterior, (1.1), podemos afirmar que ambas oraciones describen el mismo hecho, que involucrará a dos elementos: *plano* y *punto*, atribuyéndoles el mismo papel en el suceso. La interpretación de estas oraciones garantizaría que, a pesar de la diferencia de función gramatical de los dos sintagmas nominales, los papeles desempeñados en el suceso por los referentes fueran los mismos.

Por otra parte, y dado que las lenguas son conjuntos infinitos de expresiones, la proyección de las oraciones sobre sus representaciones semánticas no puede ser un mero listado de los elementos que intervienen, sin relacionarlos de forma alguna. Mientras que la conexión entre palabras como *"recta"*, *"plano"* y *"la(el)"* y sus significaciones es arbitraria, las interpretaciones de los sintagmas nominales *"la recta"* y *"el plano"* no lo son. Este hecho se refleja en el presupuesto de un principio de *composicionalidad* que afirma que el significado de una expresión completa está determinado por el significado de sus constituyentes y de la forma en que estos se combinan. Así como en sintaxis se distingue el léxico de los principios que determinan la estructura de frases y oraciones, también se distingue en semántica entre **semántica léxica** y **semántica composicional**. La primera describe las representaciones semánticas de las palabras y de otros elementos sintácticos básicos. La segunda proporciona los principios que determinan cómo se combinan las interpretaciones léxicas para dar como resultado representaciones semánticas de expresiones sintácticamente complejas.

Existen sin embargo alternativas a la semántica composicional. La primera es que el significado del todo sea función no sólo de las partes, sino también de la situación en la que la oración es enunciada. Esta alternativa es muy útil a la hora de efectuar una correcta resolución de referentes, como en el caso de algunos pronombres:

(1.12) Hallar el punto donde se intersectan las rectas y=3x-2 y y=x-7 y determinar su ángulo de intersección.

También existe una segunda alternativa no composicional, en la que el significado global no es una función sistemática de las partes en ningún sentido razonable, sino que el significado de una palabra aislada varía con las otras palabras en la misma oración.

Mientras que la aplicación de las semánticas no composicionales suele resultar muy compleja, mantener la composicionalidad permite aplicar las técnicas semánticas, siempre que sea razonable hacerlo, de forma recursiva en una oración y con cada uno de sus componentes, de forma uniforme.

No es tampoco estrictamente necesario que una teoría sea capaz de interpretar la primera parte de la oración antes de tener información del resto, aunque este planteamiento concuerda con nuestra intuición acerca de cómo los individuos entienden las oraciones.

Por ejemplo, se puede pensar que el resultado de leer o escuchar (1.13) es alguna clase de estructura mental que representa el proceso de recopilar los datos suficientes para poder *"escribir"* la fórmula pedida. Es más, si el enunciado no fuese finalizado, el lector (u oyente) podría resolverlo en cuanto se proporcionaran algunos datos acerca del objeto para el que se pide la ecuación.

(1.13) Escribir una ecuación para ...

Por lo tanto es deseable que nuestro sistema sea capaz de proporcionar una interpretación de un componente tan pronto como esté completo, especialmente si debe utilizar el contexto de la primera parte de la oración para resolver ambigüedades en el resto de ésta.

Desde nuestro punto de vista, también sería conveniente que el intérprete **trabajase en paralelo con el parser** para ser así capaz de proporcionarnos la información necesaria para la desambiguación de las estructuras. Para ello, la representación de la oración parcialmente interpretada deberá ser siempre un

objeto semántico enunciado de forma adecuada, para que de este modo pueda ser utilizado.

El intérprete también debe ser capaz de **resolver la ambigüedad léxica** así como dar cuenta de otras propiedades semánticas como son la anomalía de las oraciones (1.14), la contradicción de (1.15) y que (1.16) sea necesariamente verdadera atendiendo al significado de sus partes:

(1.14) a. El punto que sumado al plano es perpendicular al conjunto de números naturales.

b. El conjunto de números naturales es perpendicular.

(1.15) El plano Π paralelo al plano Ψ es perpendicular al plano Ψ.

(1.16) El eje X es una recta.

Por último, otra de las características de la interpretación semántica es que la **implementación de reglas no sea ad hoc**, entendiéndose bajo este término que sean mecanismos generales no diseñados de forma puntual para cada palabra o porción de texto. Las reglas semánticas o el formalismo deberían ser capaces de manipular objetos semánticos y construir los nuevos, pero no deberían poder *destruir* los objetos semánticos (poniendo en peligro la composicionalidad); es más, cada uno debería ser general y adecuadamente motivado. Las reglas también deben ser capaces de tener en cuenta la contribución de la estructura sintáctica de la oración al significado.

Existen, por supuesto, otros muchos problemas a los que un intérprete semántico se puede enfrentar como son las intensiones, contextos opacos, genéricos, cuantificación compleja, etc. Estos puntos, sin embargo, no han aparecido en nuestro dominio y no serán considerados en los siguientes apartados.

1.2 Interpretación semántica e Inteligencia Artificial

Por interpretación semántica se entiende el proceso de *"representar"* un texto en lenguaje natural en una aclaración o traducción de su significado libre de ambigüedades e independiente de la lengua que se interpreta. La entrada de un intérprete semántico puede estar formada por un árbol de análisis, subárboles tales como sintagmas nominales o bien palabras aisladas. La salida de un intérprete semántico será el significado del texto de entrada, utilizando una representación adecuada.

La representación semántica se puede realizar de forma que excluya todos los aspectos del análisis sintáctico o interactuando con él. En la segunda aproximación, los análisis sintáctico y semántico están integrados, como se describe en (Riesbeck y Schank, 1978), (Cater, 1982). En la primera aproximación, como la que se describe en (Hirst 1989), se da por supuesta la existencia de un *"parser"* que efectúe un análisis sintáctico (y morfológico) antes de entrar en el intérprete semántico. Sin embargo el *parsing* no es posible sin ayuda semántica, por ejemplo para la resolución de referentes en sintagmas preposicionales y oraciones de relativo.

Por su parte, el desarrollo de la Inteligencia Artificial (IA) necesariamente ha incluido gran parte de la investigación en formalismos y sistemas para la representación del conocimiento. Principalmente se han utilizado dos clases de representaciones:

- *Representaciones lógicas:* lógica de predicados, distintas formas de lógica intensional, lógicas de órdenes superiores, etc. En este tipo de representación, una base de conocimiento está formada por un conjunto de asignaciones, aserciones o sentencias lógicas que cumplen ciertas leyes.
- *Estructuras del conocimiento:* redes semánticas, marcos, *scripts*, etc. En este caso, una base de conocimiento está formada por un conjunto de objetos (datos) estructurados mediante relaciones entre los objetos.

Las dos clases no son antitéticas; de hecho, el cálculo de predicados es isomorfo a una sencilla red semántica o sistema de marcos.

Independientemente de la representaciones utilizadas, se puede pensar que una **base de conocimiento** es un modelo de un mundo exactamente en el sentido utilizado en las semánticas basadas en modelos. Es decir, proporciona un modo para

decidir la certeza de una declaración acerca del mundo que representa: una declaración verdadera es aquella que está representada en la base de conocimiento o se puede probar a partir de ella, y es falsa aquella cuya negación es verdadera. Esto no quiere decir que la base de conocimiento encuentre necesariamente los requisitos formales de un modelo para cualquier sistema modelo-teórico particular.

Se debe hacer énfasis en el concepto que tenemos de la semántica que, como en (Winograd, 1984), consiste en el estudio del significado abstracto y su relación con el lenguaje y el mundo, así como acerca de la manera en que los agentes comprenden el significado de una declaración lingüística en el mundo. En este caso, un mundo restringido al dominio de Prógenes.

A continuación se van a examinar brevemente algunos formalismos semánticos y de interpretación semántica en la lingüística y en sistemas del procesamiento del lenguaje natural. La descripción de los trabajos que se aporta trata de responder a algunas de las siguientes preguntas:

- ¿Qué clase de formalismo, representación o modelo puede capturar adecuadamente la semántica de una declaración en lenguaje natural?; es decir, ¿qué es un objeto semántico?
- ¿Qué clase de proceso puede proyectar una declaración en lenguaje natural en estos objetos semánticos?
- ¿Pueden este tipo de objetos semánticos ser utilizados en aplicaciones de Inteligencia Artificial? Una vez que una oración ha sido interpretada, ¿cómo puede ser aplicado su significado por el sistema que lo recibía como entrada?

Todas estas preguntas son de interés para los investigadores en Inteligencia Artificial, pero históricamente los lingüistas se ocupan principalmente de las dos primeras y los filósofos de la primera. Parece, por tanto, razonable el hecho de que al estudiar el trabajo sobre semántica, fuera del ámbito de la Inteligencia Artificial, encontremos que estas interpretaciones no han sido utilizadas en sistemas reales. Opuestamente se podrían encontrar teorías semánticas adecuadas para la Inteligencia Artificial que no satisfacen a lingüistas ni a filósofos.

1.2.1 Representaciones lógicas

1.2.1.1 Semántica basada en modelos y condiciones de verdad

En su bien conocido artículo sobre *"PTQ"* (Proper Treatment of Quantification), Richard Montague (Montague, 1974) presentó la sintaxis y semántica completa de un pequeño fragmento de inglés. Aunque era limitado en vocabulario y complejidad sintáctica, el fragmento de Montague tenía en cuenta problemas semánticos como las intensiones, los contextos opacos, distintos tipos de "predicación" con el verbo *"be"*, así como la posibilidad de impedir que de oraciones del tipo:

```
"La temperatura es noventa y la temperatura está aumentando."
```

se pueda inferir que

```
"90 está aumentando."
```

El formalismo de Montague es bastante complejo, y no va a ser descrito en el presente trabajo, sólo se van a presentar sus importantes propiedades teóricas. Más detalles sobre este formalismo semántico se pueden encontrar en (Moreno, 1985) y (Dowty et al., 1981).

La teoría de Montague está basada en condiciones veritativas y en modelos. El que esté basada en condiciones veritativas quiere decir que el significado de una oración es el conjunto de condiciones necesarias y suficientes para que la oración sea cierta; es decir, que se corresponda con un conjunto de hechos en el mundo. El que esté basada en modelos quiere decir que la teoría utiliza un modelo matemático y formal del mundo, necesario para establecer las relaciones entre elementos lingüísticos y sus significados. Por lo tanto, los objetos semánticos serán entidades en este modelo. Puesto que las oraciones no son enunciados limitados sobre el mundo, Montague emplea un conjunto de *mundos posibles*; la certeza de una oración es entonces relativa a un mundo posible y un punto en el tiempo. Un par formado por el *mundo-tiempo* se denomina **índice.**

Montague considera que la palabra es la unidad básica del significado, suponiendo que para cada índice existe una entidad en el modelo por cada palabra del lenguaje. Por supuesto la misma entidad podría ser representada por más de una palabra; de aquí que, para algún índice, dos palabras distintas pudieran denotar el mismo conjunto de individuos y, en particular, puedan denotar el conjunto vacío. También puede ocurrir lo opuesto: una palabra ambigua que representa distintas entidades en contextos lingüísticos diferentes, pero con el mismo índice, no estaría permitida en el formalismo de Montague.

Por lo tanto, para Montague los objetos semánticos y los resultados de las traducciones semánticas son *unidades individuales* en el modelo del mundo, *conceptos individuales* (que son funciones del conjunto de unidades individuales en el conjunto de índices), propiedades de conceptos individuales, y funciones de funciones de funciones. Inicialmente, el significado de una oración es una condición de certeza relativa a un índice. Estos objetos semánticos son representados mediante expresiones de una lógica intensional; es decir, en vez de traducir directamente del lenguaje natural a estos objetos, cualquier oración se traduce primero a una expresión de la lógica intensional para la cual existe una interpretación en el modelo en términos de estos objetos semánticos.

Montague define una fuerte *teoría de tipos* para sus objetos semánticos, tipos que se corresponden con tipos de los constituyentes sintácticos. Por lo tanto, dada una categoría gramatical particular como nombre propio o adverbio, Montague es capaz de decir que el significado del constituyente de esa categoría es un objeto semántico determinando su tipo.[2] El sistema de tipos de Montague es definido recursivamente, con unidades individuales, valores de certeza, e intensiones como primitivas, y otros tipos definidos como funciones de un tipo en otro de tal forma que si la categoría sintáctica *X* es creada componiendo la categoría sintáctica *Y* con la categoría sintáctica *Z*, entonces el tipo correspondiente a *Z* es una función del tipo de *Y* al tipo de *X*. Por ejemplo, el tipo semántico de las preposiciones es la función que aplica el tipo semántico de los sintagmas nominales en el tipo semántico de los sintagmas preposicionales.

El sistema de Montague contiene un conjunto de reglas sintácticas y un conjunto de reglas semánticas, y entre las dos existe una correspondencia biunívoca. Cada vez que una regla sintáctica es aplicada, también lo hace la regla semántica; mien-

2 Para ser exactos, el tipo semántico de un nombre propio es el conjunto de propiedades de los conceptos individuales y el de un adverbio es la función entre los conjuntos de conceptos individuales.

tras que la una opera sobre algunos elementos sintácticos para crear un nuevo elemento, las otras operan sobre los correspondientes objetos semánticos para crear un nuevo objeto correspondiente al nuevo elemento sintáctico.

Las reglas sintácticas están formalizadas mediante una sencilla gramática categorial (Dowty et al., 1981). Reglas típicas son (1.17), la regla que permite combinar un determinante y un nombre para crear un sintagma nominal, y (1.18), la regla que permite crear sintagmas verbales intransitivos:[3]

1.17 Si ξ es un nombre, entonces todo ξ, el ξ y un ξ son sintagmas nominales.

1.18 Si δ es un sintagma verbal que necesita una oración, y β es una oración, entonces δβ es un sintagma verbal intransitivo.

Por su parte, existen tres tipos de reglas semánticas. El primer tipo está formado por las reglas básicas que sólo proveen traducciones para las palabras individuales. El segundo tipo traduce constituyentes como los expresados en la regla 1.17 del sintagma nominal. Por ejemplo, la regla semántica que corresponde a la del sintagma nominal es:

1.17' Si ξ se traduce como ξ', entonces todo ξ se traduce como:

$\lambda P\ [\forall x[\xi'(x) \Rightarrow P\{x\}]]$;

el ξ se traduce como:

$\lambda P\ [\exists y[\forall x[\xi'(x) \Leftrightarrow x=y] \wedge P\{y\}]]$;

un ξ se traduce como:

$\lambda P[\exists x[\xi'(x) \wedge P\{x\}]]$;

donde P{x} representa la aplicación de la extensión de P a x.

[3] En el ejemplo se está abreviando la regla al no considerar las flexiones del género y número.

Lo que esta regla pretende recalcar es que la traducción de un sintagma nominal es una función, i.e., una expresión en un λ-calculus. En particular, es una función de la lógica intensional que incluye la traducción del nombre.

El tercer tipo de regla semántica está compuesto por aquellas reglas de aplicación funcional: la traducción de nuevos constituyentes se obtiene a partir de la aplicación funcional de la traducción de uno de sus componentes en otro. La regla que corresponde a 1.18 es:

1.18' Si la traducción de δ es la función δ' y la de β es β', entonces la traducción de δβ es δ'(^β'), donde ^β' representa la intensión de β'.

La semántica de Montague es composicional debido, entre otros, a:

- el hecho de que la operación fundamental de combinación semántica sea la aplicación funcional,
- el fuerte tipado,
- el hecho de que las reglas semánticas operan a continuación de las sintácticas sobre dos tipos no básicos,
- el hecho de que el resultado de una regla semántica es siempre función del dato de entrada.

A pesar de los excelentes principios de la semántica de Montague, que habitualmente se consideran como punto de partida en la composición de formas lógicas, no es posible implementarla directamente en un sistema práctico, principalmente por dos razones. La primera es que la semántica de Montague tal y como está formulada, crea enormes conjuntos, infinitos objetos, funciones de funciones y gran número de mundos posibles. En Friedman (Friedman et al., 1978), el más pequeño de los posibles sistemas de Montague, formado por dos entidades y dos puntos de referencia, contiene $2^{(2^{522})}$ elementos en la clase de las posibles denotaciones de preposiciones, donde cada elemento es un conjunto que contiene 2^{512} pares ordenados.

La segunda razón por la que no podemos utilizar la semántica de Montague es porque no estamos interesados en ver si un conjunto de hechos es o no cierto en algún mundo posible, sino en el propio estado de los hechos, de aquí que en Inte-

ligencia Artificial se utilicen formalismos semánticos basados en Bases de Conocimiento, en los que un objeto semántico tiende a convertirse en símbolos o expresiones en un sistema de representación del conocimiento declarativo o procedural. Es más, la semántica basada en condiciones de certeza realmente sólo tiene que ver con oraciones declarativas (aunque existen algunos trabajos que han extendido el trabajo de Montague a otro tipo de oraciones). En la práctica, los sistemas para el procesamiento del lenguaje natural deben ser capaces de interpretar comandos, preguntas, así como oraciones declarativas.

Sin embargo, existen algunos intentos de usar la lógica intensional que Montague utiliza como un paso intermedio en las traducciones para dar una nueva interpretación en términos de objetos semánticos válidos en IA (Hobbs y Rosenschein, 1978).

1.2.1.2 Representaciones del discurso

(Kamp y Reyle, 1993) proponen teorías de la interpretación que complementan una teoría semántica basada en las condiciones de verdad en un modelo con una teoría de la representación del discurso, y ofrecen soluciones a algunos problemas relacionados con la referencia de los sintagmas nominales definidos e indefinidos y de los pronombres. Estas teorías utilizan un nivel de representación semántica que media entre las representaciones sintácticas de las oraciones y la interpretación basada en los valores de verdad. Las *representaciones del discurso* elaboradas como parte de la interpretación de los discursos son conjuntos de entidades, básicamente los referentes del discurso necesarios para la interpretación de los pronombres y las condiciones especificadas por el discurso sobre estos. La representación del discurso de (1.19a) contendría, por ejemplo, la información de (1.19b): dos objetos identificados como *"u"* y *"v"*, donde el primero es paralelo al segundo.

(1.19) a. Π es paralelo a Π'

b. •u •v

u = Π
v = Π'
u paralelo v

(1.20) a. Π es paralelo a un plano

b. •u •v

u = Π
plano (v)
u paralelo v

Mientras que (1.19b) puede simplemente considerarse como una particular representación gráfica de la información contenida en una traducción a la lógica de primer orden, la innovación de este enfoque para la interpretación puede verse en (1.20b), la representación del discurso construida para (1.20a), que contiene un sintagma nominal indefinido. Los sintagmas nominales indefinidos se consideran términos referenciales más que expresiones cuantificadoras; su interpretación introduce una entidad en la representación del discurso que le sirve como referente. Por tanto, las condiciones asociadas con el sintagma nominal *"un plano"* en (1.20a) activan la introducción del referente *"v"* y el requisito de que *"v"* sea *"un plano"*.

Los principios de construcción de una representación del discurso unifican la información contenida en oraciones sucesivas, de tal forma que permiten que la referencia pronominal intra e inter-oracional se represente de modo unificado. La interpretación de la oración "Π *no lo corta*", siguiendo (1.19a) o (1.20a), implica expandir las representaciones para incluir más información sobre las entidades que contienen. Uno de los medios por los que (1.19b) y (1.20b) podrían expandirse se deriva de la elección de enlazar los pronombres con las entidades presentes ya en la representación del discurso, dando lugar a las representaciones de (1.21) y (1.22), respectivamente.

(1.21) a. Π es paralelo a Π'

Π no lo corta.

b. •u •v

u = Π
v = Π'
u paralelo v
u no-corta v

(1.22) a. Π es paralelo a un plano

Π no lo corta.

b. •u •v

u = Π
plano (v)
u paralelo v
u no-corta v

Obsérvese que la conexión entre las expresiones y las denotaciones basadas en modelos es indirecta, ya que intervienen las representaciones del discurso. Interpretar las expresiones consiste en construir y actualizar esas representaciones del discurso; las expresiones interpretadas pueden guiar este proceso codificando instrucciones sobre cómo tienen que actualizarse las representaciones. Por ejemplo, la diferencia entre sintagmas nominales indefinidos y definidos se expresa en términos del cambio en la representación del discurso resultante de su evaluación: los indefinidos introducen nuevos referentes del discurso y los definidos no.

Los principios de la teoría de representación del discurso no sólo permiten determinar cuándo una oración es cierta o no en un mundo posible, sino que también esboza las bases para efectuar una adecuada representación del significado. Teniendo en cuenta estos principios, faltaría por decidir cuál es la representación más adecuada para interactuar con el sistema que tomará esta representación como entrada.

1.2.1.3 Semántica de situaciones

(Barwise y Perry, 1992) desarrollan la *semántica de situaciones* como alternativa a la semántica teórica basada en modelos y en mundos posibles. Esta propuesta comparte algunos de los objetivos y técnicas de la descrita por Montague.

Según la concepción de la semántica de la situación, el mundo consiste en situaciones que contienen individuos reales, propiedades, relaciones y ubicaciones en el espacio y en el tiempo. Las situaciones son construcciones conjuntísticas,

pero las propiedades y las relaciones son conceptos primitivos no construidos a partir de individuos y puntos de referencia. Las situaciones en el mundo se clasifican mediante tipos de situación, que son conjuntos de secuencias de individuos, ubicaciones espacio-temporales, relaciones y valores de polaridad; todos ellos básicos desde el punto de vista ontológico. Por ejemplo, la siguiente secuencia representa una situación en la que "Q" pertenece a "R" y "R" no pertenece a "Q", donde "Q" y "R" representan el conjunto de los números racionales y reales respectivamente.

```
<1, pertenecer, Q, R, 1>

<1, pertenecer, R, Q, 0>
```

Secuencias como ésta pueden construirse como interpretaciones de las oraciones combinando los tipos de situación asociados con partes de la oración para presentar una explicación composicional de la interpretación de la oración. Las interpretaciones de las oraciones, combinadas con una teoría de restricciones, permiten definir las relaciones de equivalencia y de consecuencia.

La estructura adicional de las situaciones, comparadas con las proposiciones, ofrece posibilidades de nuevas soluciones al problema de las actitudes proposicionales y sugiere interesantes explicaciones de la semántica de ciertas construcciones. La investigación en la semántica de la situación, sin embargo, no está exclusivamente dedicada a la semántica lingüística, sino que es un intento de elaborar una concepción relacional general del significado. El significado lingüístico se considera como un tipo particular de significado, semejante al sentido reflejado en afirmaciones como *"Toda función continua es derivable"*. El significado es una relación entre diferentes tipos de situaciones y la relación entre las situaciones que implican eventos lingüísticos. No se conciben otros tipos de situaciones de un tipo diferente a las relaciones entre situaciones no lingüísticas. En los artículos de *Linguistics and Philosophy 8, 1* se puede encontrar una discusión y crítica de la investigación en la semántica de situaciones.

1.2.2 Estructuras del conocimiento

1.2.2.1 Semántica decomposicional

La teoría de la semántica decomposicional trata de representar el significado de cada palabra descomponiéndolo en un conjunto de primitivas semánticas. Por ejemplo, la palabra *"silla"* podía ser representada como se muestra:

```
(Objeto), (Físico), (No-animado), (Artefacto), (Mueble),
(Portable), (Algo con patas), (Algo con respaldo),
(Algo con un asiento), (Asiento unitario)
```

Por supuesto, en la práctica, un programa que descomponga de esta forma los conceptos es problemático. Es extremadamente difícil, o quizá imposible, encontrar un conjunto de elementos semánticos primitivos, lingüísticamente universal, en el que todas las palabras sean descompuestas en sus propiedades necesarias. Por ejemplo, no todas las propiedades enumeradas definen atributos de una silla, o están presentes en todas las sillas modernas, y aún así parecen necesarios para distinguir las sillas de otros utensilios para sentarse. Aun las palabras aisladas más simples son extremadamente difíciles de caracterizar.

La semántica decomposicional resulta también problemática en la representación de una oración. No se puede, simplemente, manejar las representaciones de las palabras individuales desde el árbol de análisis de una oración y pensar que el resultado es el significado, puesto que la menor variación en la estructura sintáctica cambia el significado de la oración.

Pero un formalismo menos estructurado tampoco funciona adecuadamente. Tanto Katz (Katz, 1980) como Fodor (Fodor, 1985) simplemente ensartaron en orden las representaciones de las palabras, obteniendo la siguiente representación:

```
El hombre golpea la pelota de color.
[Algo contextualmente definido] → [Objeto
físico] → (Humano) → (Adulto) →
(Hombre) → (Acción) → (Instancia) →
(Intensidad) → [Choca con un impacto] →
[Algo contextualmente definido] → [Objeto
físico] → [[Superficie globular]
[Abundante en contraste y variedad de colores]]
```

Desde luego no está claro qué se puede hacer con una representación como ésta, pero, obviamente, no muestra la contribución del significado de la estructura sintáctica de una oración, requisito necesario.

El representar una palabra como un conjunto de primitivas no resulta muy útil si la teoría no puede proveer ni una estructura adecuada para unirlos ni métodos para efectuar una inferencia profunda en el contexto a partir de las estructuras resultantes, como puede verse en (Winograd, 1984), (Tarnawsky, 1982). Sin embargo, se han efectuado algunos intentos en Inteligencia Artificial para realizar semántica decomposicional útiles al añadir a la teoría los elementos necesarios. Entre ellos cabe destacar el sistema de Preferencia Semántica (Wilks, 1989) y las representaciones de Dependencias Conceptuales (Schank, 1982).

1.2.2.2 Las "estructuras conceptuales" de Jackendoff

Para (Jackendoff, 1990), las representaciones semánticas son totalmente psicológicas. Sus representaciones semánticas son *estructuras conceptuales* concebidas como objetos mentales. Jackendoff complementa los objetivos más clásicos de la teoría semántica con dos restricciones: la *restricción gramatical* y la *restricción cognitiva*. La restricción gramatical señala que, en igualdad de condiciones, debe preferirse una teoría semántica que explique lo que de otra forma serían generalizaciones arbitrarias sobre la sintaxis o el léxico. La restricción cognitiva adopta la forma de una hipótesis, en el sentido de que las estructuras conceptuales son las representaciones utilizadas por el sistema sensitivo así como por el sistema lingüístico.

La notación empleada para las estructuras conceptuales son diagramas de atribución de valores. Los constituyentes de estos diagramas se asignan a una de las siguientes categorías ontológicas: *COSAS*, *LUGARES*, *DIRECCIONES*, *ACCIONES*,

SUCESOS, *MODOS* y *CANTIDADES*. Estas estructuras conceptuales se asocian con oraciones mediante *reglas de correspondencia* que preservan parte de la estructura de constituyentes de las oraciones en la representación conceptual. Por ejemplo, la estructura conceptual asociada con la oración (1.23a) es (1.23b).

(1.23) a. Aplicar el Teorema del valor medio a f.

b.
```
[ ACCION
  APLICAR ( [ COSA                          ] , [ COSA ] )
            [ EL TEOREMA DEL VALOR MEDIO    ]   [ F    ]
]
```

Dada la restricción gramatical, estas representaciones, que mantienen la estructura de constituyentes superficial de la oración, son preferibles a, por ejemplo, traducciones en lógica de primer orden que no contienen constituyentes correspondientes a los sintagmas nominales. Estas estructuras conceptuales son las representaciones sobre las que se formulan reglas de inferencia como transformaciones y presentan, así, las definiciones deseadas de equivalencia y consecuencia.

La idea de representar los constituyentes del discurso mediante categorías ontológicas es de gran utilidad. Sin embargo, y dadas las características del dominio de Prógenes, parece más adecuado utilizar categorías, o *tipos*, que describan el dominio de forma más concreta. Así el constituyente *"F"* no tendría tipo *"COSA"* sino *FUNCION*, y así ocurriría con todas las palabras que intervienen en el texto. Falta por estudiar cómo se almacenan los constituyentes y cuáles son realmente las reglas de correspondencia que relacionan estos elementos.

1.2.2.3 Semántica basada en el conocimiento

Históricamente, la perspectiva de la semántica centrada en bases de conocimiento utilizada en el ámbito de la Inteligencia Artificial fue denominada *conocimiento semántico* (Tarnawsky, 1982). Este paradigma propone que el significado de una oración depende del conocimiento del intérprete, y representa dicho significado mediante proposiciones, posiblemente un número infinito, vinculadas por la oración con respecto a ese conocimiento. En Inteligencia Artificial el número de inferencias que resulta de la suma de una oración a la base de conocimiento es finita. Es más, los sistemas restringen al máximo las inferencias que pueden ser efectuadas.

Tarnawsky formalizó una teoría semántica en la que la base de conocimiento desempeñaba un papel central. En su teoría, las reglas transformacionales aplicaban los datos de entrada en lógica de predicados de órdenes superiores, que era añadida a la base de conocimiento.

Existía una base de datos en la que se almacenaba el conocimiento del sistema. En esta teoría, el significado de una oración consistía en añadir la declaración inicial a la base de conocimientos, y las consecuencias que este hecho producía. El proceso semántico que se realiza es composicional, al nivel de manipulación simbólica; es decir, este tipo de paradigma no construye el significado final a partir de la representación de cada uno de los elementos. Esto es debido a que la base de conocimiento provee un modelo semántico para las oraciones, pero no para los componentes de las oraciones. Por ejemplo (Tarnawsky, 1982):

```
The man ate the kutya.

ate (the (man), the (kutya))
```

Como se puede ver en el ejemplo, la representación semántica está formada por las mismas palabras que componen la oración inicial; ha sido reestructurada aunque los símbolos por sí solos no han sido interpretados de ninguna forma. La base de conocimiento puede o no saber qué significa *"kutya"*, y puede o no contener otros hechos sobre dicho concepto, pero no contiene ningún elemento que represente *"kutya"* per se. Por otra parte, los sintagmas preposicionales no pueden ser representados. Puesto que los símbolos no tienen una representación directa en la base de conocimiento, la resolución de referentes dentro de las declaraciones

se consigue mediante mecanismos de inferencia, que aumentan su número de forma considerable teniendo en cuenta las distintas formas en las que los referentes pueden ser combinados.

En resumen, Tarnawsky utiliza lógica de predicados de órdenes superiores para representar el significado de las oraciones, pero no las entidades a las que se refiere.

Para resolver algunas de las deficiencias de este paradigma se puede utilizar una estructura del conocimiento más adecuada, como pueden ser las redes semánticas, los marcos o representaciones del conocimiento similares, como la que se describirá en los siguientes capítulos. Los símbolos, que eran interpretados, pueden ahora ser referencias a la estructura del conocimiento que sea adecuada. Es decir, cada concepto o palabra dentro de una oración puede ahora ser reemplazado por un puntero a un elemento de la base de conocimiento. El léxico aplica las palabras y las oraciones en sus representaciones, y los sinónimos tienen la misma representación. Un concepto puede existir dentro de la base de conocimiento con el mismo nombre, pero esto no es más que una coincidencia.

1.3 Recapitulación

Cada uno de los enfoques teóricos mencionados aquí tiene sus propios problemas internos pendientes de futuras investigaciones. Cada uno tiene áreas en las que ofrece atractivos análisis. Para concluir este capítulo se enumeran algunas cuestiones que, sin ningún género de dudas, siguen siendo motivo de discusión.

La cuestión más espinosa es la de saber qué son realmente las representaciones semánticas. La aplicación de una semántica basada en modelos a la semántica computacional proporcionó un inmediato arsenal de técnicas y una precisión en la formulación que ha estimulado gran cantidad de trabajos. Aunque otros enfoques hayan intentado evitar los problemas de las actitudes proposicionales o hayan derivado hacia nociones más psicológicas del significado, todavía no han llegado al nivel de la semántica de Montague. A medida que van ampliando su alcance, los análisis contrastivos van arrojando más luz sobre las cuestiones fundamentales de las técnicas empleadas en la teoría semántica.

La proyección entre sintaxis y semántica es otro problema, dada su dependencia de asunciones previas sobre su aducto y su objetivo. La variedad de enfoques actuales sobre la teoría sintáctica y la correspondiente variedad de asunciones sobre el alcance de la proyección interpretativa originan muchas y diferentes propuestas sobre la naturaleza de los puntos de contacto entre la sintaxis y la semántica. Estas diferencias mantienen abierta la cuestión de la medida en la que las lenguas son composicionales.

Otro importante grupo de cuestiones en las que se ha logrado algún avance tiene que ver con la relación entre la semántica, tal y como se ha caracterizado en este capítulo, y la pragmática. La semántica ha sido introducida como una teoría de la interpretación de tipos de oraciones, no de instancias de oración; de abstracciones lingüísticas y no de enunciados específicos. Sólo los enunciados son interpretados y transmiten o piden información. La teoría de la pragmática toma los enunciados como objeto de estudio y la descripción de lo transmitido por el uso lingüístico como meta. Si consideramos la semántica como descripción de la competencia lingüística, esta abstracción parece, entonces, razonable, dado que la distinción entre semántica y pragmática se corresponde, así, con la de competencia y actuación.

Capítulo 2

2.0 Marco de la investigación: Proyecto Prógenes

A lo largo de este capítulo describiremos el marco de la investigación en la que se ha llevado a cabo el trabajo que se presenta, el proyecto Prógenes: **Proof Generator Expert System**. Dicho proyecto se ha realizado en el Instituto de Ingeniería del Conocimiento (IIC), con financiación del Plan Nacional de Investigación, Programa Nacional de Tecnología de Información y de las Comunicaciones, PRONTYC, con el número de proyecto 352.90.

Prógenes es un proyecto de investigación básica dedicado al desarrollo de un *sistema capaz de resolver problemas enunciados en lenguaje natural, español* [(Castells et al., 1991), (Castells et al., 1992), (Castells et al., 1992b), (Castells, 1994), (Saiz, 1994)].

En un principio los problemas que se planteaban se circunscribían a algunas áreas del campo de las Matemáticas, aquellas en las que resolver dichos problemas se redujera, en la última fase, a resolver un conjunto de ecuaciones. Sin embargo, a medida que avanzaba el proyecto, se fue haciendo cada vez más evidente que la condición impuesta era de índole suficientemente general como para que pudiéramos plantearnos la resolución de problemas relacionados con otros dominios como, por ejemplo, mecánica o termodinámica.

Por otra parte, desde el inicio del proyecto, uno de los requerimientos impuestos para la resolución de los problemas era ser capaces de proporcionar al sistema exclusivamente la misma información, y en la misma forma, que a cualquier persona que tuviera que resolverlos. Por lo tanto se planificó la incorporación de una interfaz capaz de entender el texto de los enunciados expresados en español, e interpretarlos según una semántica que reflejara los conceptos y las relaciones existentes entre las distintas entidades que intervienen en cada una de las posibles áreas de conocimiento.

Mediante un lenguaje formal específicamente diseñado en el transcurso del proyecto se podría relacionar los enunciados de los problemas, el texto, con los módulos de deducción automática que los resolverían.

El objetivo de este capítulo es describir las características generales del sistema, centrándonos en los aspectos relacionados con la interfaz para el procesamiento del texto de los problemas. En este sentido, es importante analizar las características que posee el dominio en el que estamos trabajando, características que justifican muchas de las decisiones tomadas en el desarrollo de nuestro trabajo.

También mencionaremos los proyectos CAPRATE y MECHO, que han abordado problemas similares, incorporando sendas interfaces a sistemas de enseñanza o demostración.

2.1 Prógenes: Descripción del Proyecto

Prógenes es capaz de resolver problemas como los siguientes:

```
Problema 1:

    "Escribir la ecuación de la recta paralela a 3x-5y+8=0 y
     que pasa por el punto (-3,2)."

Problema 2:

    "Decimos que una función f:R → R es par si
     f(x) = f(-x) para todo x ∈R. Demostrar que
     si una función f es par f'(0) =0."
```

Como hemos mencionado, se ha definido un lenguaje formal al que son traducidos los enunciados propuestos originalmente en lenguaje natural, y que constituye el punto de partida del proceso de resolución de los problemas. Dicho lenguaje formal es equivalente al utilizado en una lógica de primer orden, con tipos que reflejan las características de los conceptos que forman parte del dominio. A este lenguaje formal, que sirve de comunicación entre la interfaz y el módulo de resolución de problemas lo denominaremos a partir de ahora **lenguaje formal Prógenes.**

Los problemas anteriores se expresan en este lenguaje de la forma siguiente:

```
Problema 1:

    (hallar (ecuacion (el (%r recta)
                          (y (paralela (recta (== (-- (++ (** 2 x) y) 7) 0))
                                       %r)
                             (en (punto 3 2) %r)))))

Problema 2:

    (demostrar (=> (par ?F) (== ((D ?F) 0) 0))
               (<=> (par ?F)
                    (y (funcion? ?F $RR $RR)
                       (paratodo (?X $RR) (== (?F ?X) (?F (- ?X)))))))
```

Para que se puedan resolver problemas de un tipo concreto, por ejemplo problemas sobre el plano euclídeo como en el Problema 2, el módulo de resolución automática hace uso de una *base de conocimientos* en la que se describen los objetos que puedan aparecer en el problema.

Actualmente se han diseñado distintas bases de conocimiento y, aunque se pueden utilizar varias de ellas simultáneamente, cada una es específica para un dominio determinado. En ellas se incluye la descripción de una jerarquía de objetos y de las relaciones y acciones que se pueden dar entre ellos. El sistema trabaja sobre los problemas expresados en términos de ese conocimiento mediante expresiones en el *lenguaje formal Prógenes.*

Dichas expresiones constituyen la entrada del módulo de resolución, que a su vez utiliza un programa de manipulación simbólica de alto nivel, Mathematica, para efectuar los cálculos simbólicos que se requieran en los problemas. El motor de inferencia de este módulo deduce en qué casos la utilización de Mathematica puede ayudar a la resolución del problema para resolver ecuaciones algebraicas o diferenciales. Prógenes dispone de un módulo que, en esos casos, efectúa los cambios de formato necesarios para plantear el cálculo requerido a Mathematica, solicitar que se realice e incorporar el resultado a los datos del problema.

En cuanto a la interfaz para el procesamiento de los textos de los problemas, el sistema Prógenes cuenta con un módulo específico para la entrada y comprensión de los mismos, cuya finalidad última es traducir el contenido del texto que se presente como entrada al lenguaje formal Prógenes.

El módulo encargado de la comprensión de los enunciados se ciñe al dominio específico de la(s) base(s) de conocimiento, circunstancia que se aprovecha tanto en lo relativo al léxico requerido como por lo que respecta a la estructuración del módulo para conseguir una mayor eficiencia. Las tareas de este módulo incluyen el tratamiento sintáctico y semántico de las oraciones de entrada, hasta conseguir la representación del enunciado de forma unívoca.

La interfaz utiliza algunos fundamentos de la familia de las gramáticas categoriales [(Oehrle et al., 1988), (Zeevat et al., 1987)] y de la teoría de representación del discurso (Kamp y Reyle, 1993). El resultado es un sistema muy apropiado para ser implementado.

El procesamiento del texto se apoya en un diccionario que contiene los rasgos sintácticos y semánticos. Como veremos, los elementos del diccionario se repre-

sentan mediante objetos generales que sirven para representar no sólo las palabras aisladas (nombres, verbos, etc.) sino también análisis parciales, fórmulas o toda la oración. El conjunto de las palabras del diccionario, análisis parciales, análisis globales y fórmulas componen el *léxico* Prógenes. Cada elemento del léxico está implementado como una estructura tipada de pares atributo-valor (Emele y Zajac, 1990) y jerarquizado en una red de herencia por defecto.

La gramática utilizada está concebida como un conjunto de reglas libre de contexto especificadas sobre las clases de la red de herencia. El hecho de que la aplicación de estas reglas esté condicionada a clases, y no a constituyentes convencionales, permite considerar su grado de especificidad. En la gramática existen reglas que pueden aplicarse siempre y otras más específicas que anulan a las anteriores cuando las clases de los objetos lingüísticos involucrados en el análisis así lo requieren.

Sobre el léxico se aplica un analizador capaz de llevar a cabo el análisis del problema, según los datos aportados por el diccionario y utilizando las reglas de la gramática. El analizador utiliza simultáneamente los aspectos sintácticos y semánticos para guiar el proceso de análisis. Este mecanismo presenta dos importantes ventajas como son la rapidez del proceso, al resultar una búsqueda dirigida apoyada en la especificidad de las reglas, y la coherencia de la representación final de la información, garantizada a través de las reglas semánticas.

El analizador utiliza ideas y técnicas de programación orientada a objetos. Estos conceptos son utilizados no sólo en los aspectos de implementación, sino también en la concepción del formalismo lingüístico en el que se basa la interfaz. La mayoría de los formalismos lingüísticos actuales que incluyen redes de herencia son marcadamente lexicalistas y dejan un papel marginal a las reglas gramaticales. En Prógenes, la gramática también hace uso de esas redes de herencia y de otros conceptos de POO. Las *"reglas orientadas a objetos"* pueden ser vistas como reglas de una gramática libre de contexto en la que los elementos no terminales son sustituidos por clases de una jerarquía a la que pertenecen los objetos lingüísticos.

2.2 Sistemas de referencia: CAPRATE y MECHO

Los sistemas para el procesamiento del lenguaje natural pueden dividirse en dos clases principales (Grosz et al., 1986): sistemas interactivos y sistemas para el procesamiento de textos.

Los *sistemas de procesamiento de textos* son aquellos en los que los textos en lenguaje natural son los principales objetos de interés. El propósito de estos sistemas es analizar un texto en lenguaje natural en una forma que sea más apropiada que el propio texto de entrada, para procesamientos posteriores. El procesamiento de textos ofrece la posibilidad de automatizar la creación de bases de conocimiento, así como efectuar traducciones en textos, integrar mensajes, etc.

Los *sistemas interactivos* son aquellos en los que el lenguaje natural es el principal modo de interacción con un sistema diseñado para hacer otras cosas. Este tipo de sistemas ha sido muy utilizado para acceder a sistemas de bases de datos. También han sido utilizados para interactuar con software convencional, como los sistemas estándar de petición de ayuda, así como con sistemas inteligentes, tales como sistemas expertos, robots, tutores inteligentes.

A continuación describiremos dos ejemplos de sistemas interactivos que han coordinado el procesamiento del lenguaje natural con la enseñanza inteligente asistida por ordenador (CAPRATE) y con la resolución de problemas de mecánica (MECHO).

2.2.1.1 Sistema CAPRATE

El objetivo del sistema CAPRATE (Sarasola, 1988) era la comprensión automática de problemas de programación enunciados en lenguaje natural. El sistema se realizó en el marco del proyecto CAPRA (Garijo et al., 1987), proyecto de Enseñanza Inteligente Asistida por Ordenador definido para el dominio de la Programación elemental.

CAPRATE efectúa un tratamiento semántico de los textos que no se ejecuta después del sintáctico sino que ambos se intercalan permitiendo eliminar lo antes posible aquellas posibilidades que son sintácticamente correctas pero que no lo son semánticamente.

En cuanto a la estrategia lingüística adoptada, este sistema fue implementado con ATNs complementadas con un tratamiento semántico. La representación de los objetos aparecidos en el enunciado consta de un conjunto de aserciones que se crean secuencialmente. Su coherencia y completitud sólo es verificada después del análisis del texto completo.

En cuanto al léxico, en CAPRATE sólo algunas palabras poseen una interpretación semántica que tenga influencia en las aserciones.

La interfaz para el procesamiento del lenguaje natural de Prógenes es similar a CAPRATE en tanto que no efectúa el análisis semántico con posterioridad al sintáctico. Aunque en Prógenes el proceso de obtención de expresiones también es secuencial, se diferencia con este sistema, en que la coherencia de una expresión se chequea en el momento en que se genera mediante la comprobación de los tipos de los argumentos. De esta forma, al final del análisis sólo se debe tener en cuenta la completitud del enunciado.

El léxico Prógenes presenta también algunas diferencias. Aparte de las producidas por el uso de distintos formalismos sintácticos, el tipo de representación semántica también afecta a las definiciones de las palabras. En Prógenes, como veremos, todas las palabras poseen definición semántica, aunque sólo algunas tienen interpretación.

2.2.1.2 Sistema MECHO

MECHO (Bundy et al., 1979) es un sistema que resuelve problemas de mecánica enunciados en lenguaje natural (inglés) o en aserciones del cálculo de predicados que describen una situación física, todo ello implementado en Prolog.

Este sistema, al igual que CAPRATE, realiza el análisis semántico tan pronto como sea posible, descartando así ramas sintácticas que pueden conducir a representaciones semánticas erróneas.

MECHO utiliza una representación de los conceptos basada en una jerarquía de tipos. El lenguaje utilizado incluye funciones y predicados, que se aplican a objetos de tipos determinados. Los conceptos pueden tener asociadas ciertas restricciones, que pueden dar lugar a hechos que se dan por ciertos, o a condiciones que hay que comprobar dependiendo del contexto en que aparece el concepto. La represen-

tación está formada por un conjunto de aserciones a elementos contenidos en una base de datos intermedia.

El sistema es capaz de reconocer configuraciones globales a partir de las relaciones entre los objetos del problema (cuerdas, barras, poleas, masas puntuales, etc.), reconstruyendo así la disposición de los mismos y utilizando información por defecto para completar la información que no se indica.

Además de compaginar los análisis sintácticos y semánticos, otra característica común a Prógenes y MECHO es el uso de una jerarquía de tipos para representar conceptos. Sin embargo en Prógenes estos tipos son utilizados también para guiar el análisis semántico. El resultado de este análisis, en el caso de Prógenes, está formado por una expresión en lenguaje formal con el contenido de todo el enunciado.

2.3 Características del dominio

El dominio en el que nos movemos durante el proceso es, en principio, restringido, centrándose en el lenguaje con el que se plantean los problemas que se pueden reducir a ecuaciones. Como cualquier lenguaje especializado, este tipo de textos presenta unas características peculiares cuyo análisis determina algunos problemas a los que hay que prestar gran atención, dado que partimos de la base de que el resultado final de este proceso es una expresión formal del contenido del enunciado absolutamente libre de ambigüedades. Igualmente existen otros problemas que podemos obviar dada la restricción del dominio.

Para poder determinar las características del dominio, se eligió un libro de texto típico de un primer curso de estudios universitarios, *"CALCULUS, de una y varias variables"* (Salas y Hille, 1982), del que se seleccionaron algunos capítulos representativos:

1. Introducción.
2. Límites y Continuidad.
3. Diferenciación.
4. El Teorema del valor medio y aplicaciones.
5. Integración.
6. La función logarítmica y exponencial.
9. Las secciones cónicas.
11. Coordenadas polares; ecuaciones paramétricas.
14. Vectores.
17. Gradientes; valores extremos, diferenciales.

Observamos que eran necesarias únicamente unas doscientas palabras para describir todos los enunciados. Estas palabras constituyen el diccionario propiamente dicho que forma el conjunto del léxico que denominamos **léxico estático.**

En dichos enunciados, aparte de las palabras del diccionario, aparecen también numerosas expresiones a las que llamaremos genéricamente *"fórmulas"* como pueden ser *"y=x+1"*, *"(3,2)"* o *"f(x,y)"*. Estas palabras, que no pueden estar conte-

nidas en el diccionario puesto que no son conocidas a priori, forman lo que denominamos **léxico dinámico.**

Como ya se ha mencionado, la entrada al sistema es un enunciado igual al que resolvería un alumno de primer año de una carrera de ciencias, pero con una diferencia: Prógenes recibe el enunciado antes de que sea impreso, es decir, directamente en TeX. Esta diferencia conlleva la necesidad de un módulo que elimine aquellos *tags* que no aportan ningún significado a los textos, y que reconozca de forma dinámica las fórmulas por el hecho de ir entre signos de *$*. De esta forma los anteriores ejemplos de fórmulas aparecerían en el texto como *"$y=x+1$"*, *"$(3,2)$"* y *"$f(x,y)$"* respectivamente.

Hemos omitido el análisis morfológico puesto que el objetivo de la interfaz para el procesamiento de los enunciados de Prógenes es extraer la semántica de un texto que se puede suponer correcto. Esta decisión fue tomada tras comprobar empíricamente que el análisis morfológico no servía para podar el árbol de análisis sintáctico ni aportaba información al análisis semántico. Es más, la rica interpretación semántica proporcionada por el lenguaje formal Prógenes permite tomar decisiones de desambiguación en enunciados confusos como el obtenido al variar el número de algunas de las palabras del Problema 1, *"Escribir las ecuaciones para las rectas* **paralelas a 3x-5y+8=0 y que pasan por el punto (-3,2)***"*. Un alumno que resolviera este problema se daría cuenta de que sólo hay una recta que cumpla la condición en negrita, lo que resulta confuso a partir del enunciado en el que se pide *"las ecuaciones de las rectas"*, y probablemente preguntaría al profesor. Prógenes no dudaría en resolver este enunciado dando la ecuación única de la recta pedida. Lo mismo ocurriría con variantes de este enunciado, como las siguientes, en las cuales la solución del sistema vuelve a ser única y totalmente libre de ambigüedades: *"Escribir* **una** *ecuación para la recta paralela a 3x-5y+8=0 y que pasa por el punto (-3,2)"*, en la que parece que puede haber más de una ecuación, *"Escribir* **ecuaciones para las rectas paralelas** *a 3x-5y+8=0 y que pasen por el punto (-3,2)"*, en el que también parece que hay más de una ecuación.

Dada la restricción del lenguaje, no se han encontrado grandes problemas de ambigüedad léxica, sin embargo, en lo relativo al discurso la situación es muy diferente. Este problema, en lo que se refiere a la resolución de referentes, es siempre muy importante, pero en algunas aplicaciones se evita a menudo, dando por supuesto que la indeterminación que se introduce no es crítica de cara al resultado final. Nos encontramos con esta situación, por ejemplo, en algunas aplicaciones de traducción automática.

En Prógenes no podemos dar por supuesta esta indeterminación, ya que para la correcta formalización del enunciado han de considerarse todos los datos que éste especifique. El enunciado puede constar de una o varias líneas, y puede incluir subapartados, notas u otros tipos de descripciones. De manera que es necesario resolver, por una parte, las referencias dentro del alcance de cada oración aislada (pronombres, subordinadas, etc.) y, por la otra, las referencias que se den en un contexto formado por varias oraciones.

Este último punto incluye tanto las referencias que se realizan mediante la utilización explícita de pronombres, como la interpretación conjunta de las fracciones de información que se encuentran en cada una de las oraciones del enunciado:

```
Problema 3:

    "Hallar el punto donde se intersectan las rectas 2x+7y+31=0 y
     x-2y+7=0, y determinar su ángulo de intersección."

Problema 4

    "Sean a y b números no negativos. Demostrar que si a² =< b²
     entonces a =< b.
     INDICACION: b²-a²=(b+a)(b-a)."
```

Por ejemplo en el Problema 3 es absolutamente necesario determinar que *"su ángulo de intersección"* se refiere a *"las rectas"*, evitando relacionarlo con *"el punto"*. A su vez, el enunciado del Problema 4 consta de tres oraciones que proporcionan los datos y plantean el problema. En ambos casos la interpretación del texto debe tener en cuenta todas las relaciones existentes: la referencia a la recta en el Problema 3 y los términos *"a"* y *"b"* que hacen referencia a las mismas entidades en las distintas porciones del Problema 4.

En este sentido, también es crítico para el sistema el problema de la determinación del nivel de modificación al que se refieren los complementos preposicionales, aún siendo relativamente tolerable en otras aplicaciones. Son frecuentes situaciones como la del ejemplo siguiente, donde es absolutamente necesario determinar la correcta asociación de los complementos preposicionales:

2.1 "Hallar la intersección de la recta R con el eje X."

Para la asociación del primero de los dos grupos preposicionales que aparecen *"de la recta R"* no existe ningún problema, ya que sólo es posible que dependa de *"intersección"* (la dependencia de *"Hallar"* es fácilmente descartable debido, por ejemplo, a sus categorías sintácticas). Igualmente, *"con el eje X"* tiene que asociarse a *"intersección"* y no a *"recta"*:

(2.1-a) "la intersección de A con B"

*(2.1-b) "la recta con C"

(2.1-a) tiene sentido, siempre y cuando se cumplan algunas restricciones sobre el *tipo* de A y de B que, como veremos a continuación, también hay que considerar en el sistema. Concretamente, A y B han de ser de tipo *curva*, circunstancia que sí se cumple en la oración 2.1. (2.1-b) pudiera tener algún sentido, pero no de forma evidente en nuestro dominio.

2.4 La interfaz. Estrategia lingüística

Como hemos dicho, la entrada al sistema está constituida por enunciados de problemas como los expresados en el apartado 2.1, que se analizan teniendo en cuenta los criterios sintácticos habituales y considerando las restricciones representadas en la descripción semántica de las palabras. Durante el proceso de análisis se obtiene una estructura que incluye la expresión semántica del enunciado, expresión que como veremos es convertida de forma casi inmediata al lenguaje formal Prógenes.

En la fase de diseño de la interfaz se decidió abordar el problema del procesamiento del lenguaje con el que nos enfrentábamos mediante un formalismo de gramática categorial sobre estructuras de rasgos limitadas, después de barajar otras posibilidades, como ATNs, gramáticas basadas en reglas y gramáticas categoriales de unificación.

Esta decisión está apoyada, en primer lugar, en la idea de que es conveniente hacer recaer la mayor parte del peso del proceso sobre el léxico. En ese sentido las aproximaciones basadas en gramáticas categoriales sobre estructuras de rasgos se apoyan fundamentalmente en la descripción de cada ocurrencia de las palabras, descripciones que incluso incluyen la categoría gramatical de análisis parciales. Por otra parte, la implementación de este tipo de formalismos es adecuada desde el punto de vista computacional, obteniéndose buenos resultados en cuanto a la rapidez del proceso (Wittenburg y Wall, 1990) (Yampol y Karttunen, 1990). Por último nos decidió la gran importancia que tiene la extracción semántica en nuestra aplicación, extracción que es inmediata en este tipo de aproximaciones.

2.4.1 La gramática

Dadas las características de nuestra aproximación, la mayor parte de la información se incluye en el diccionario, utilizándose muy pocas reglas combinatorias que, además, sólo afectan a las categorías asignadas en el léxico. En parte por lo anterior, las expresiones sintácticas y semánticas asociadas a las palabras están muy relacionadas.

En este marco, el formalismo adoptado combina los puntos de vista sintácticos de las gramáticas categoriales con los puntos de vista semánticos de la teoría de la

representación del discurso, siendo la potencia del conjunto aún mayor al utilizarse mecanismos de unificación que permiten relacionar diferentes niveles lingüísticos en la descripción de las palabras.

Clásicamente, en gramáticas categoriales se emplean tres categorías primitivas: N, S y NP. Estas representan a los nombres o núcleos de los sintagmas nominales, a las oraciones y a los sintagmas nominales respectivamente. En nuestra implementación hasta ahora sólo hemos utilizado las dos últimas, no siéndonos necesario, por el momento, considerar *N* como categoría primitiva.

El resto de las categorías se atiene a los siguientes principios:

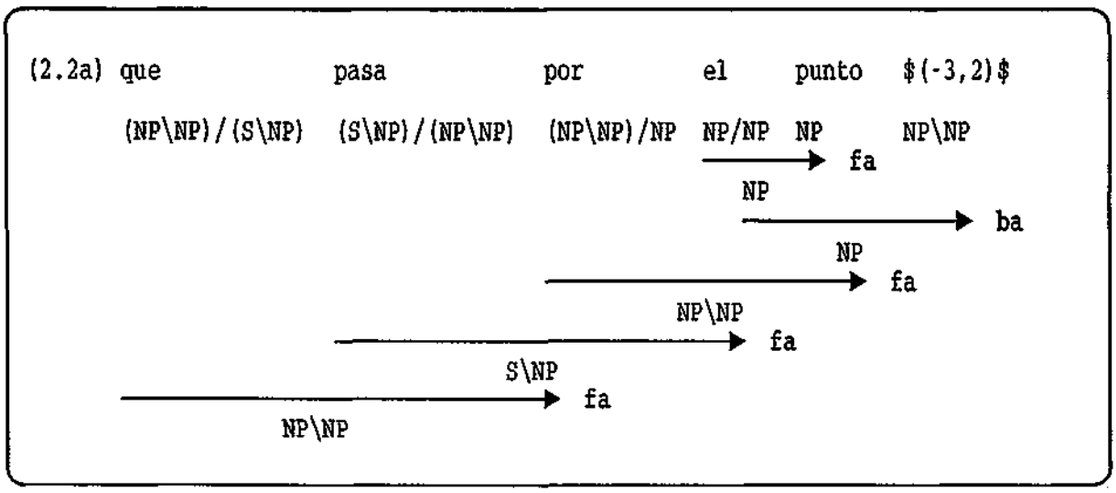
1. Cualquier categoría básica es una categoría.
2. Si *A* y *B* son categorías, también lo es *A / B*, *A \ B* y *A | B*.

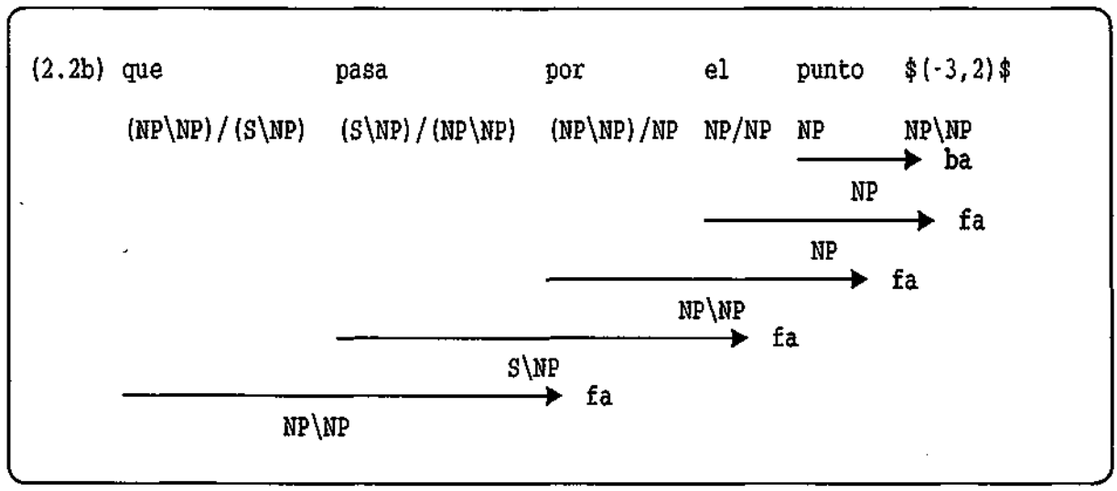
siendo / el operador binario que efectúa una búsqueda hacia la derecha, \ el que la efectúa hacia la izquierda y | el operador que une las dos funciones anteriores.

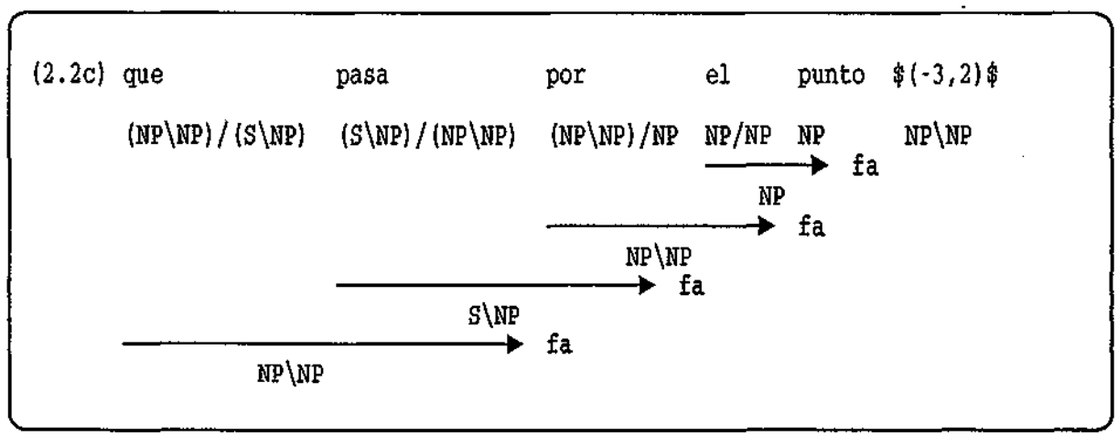
Sobre estas categorías se definen las operaciones clásicas que constituyen la gramática categorial básica, usando la notación de Steedman (Wittenburg y Wall, 1990), (Moreno-Torres y Solias, 1991):

```
X/Y     Y     =>     X        forward application (fa)
Y       X\Y   =>     X        backward application (ba)
```

A continuación se describe en detalle el análisis sintáctico de un fragmento del enunciado del Problema 1, para ilustrar la aplicación de la gramática sobre la descripción de las palabras. La categoría sintáctica de este fragmento de la oración inicial es *NP\NP*. La aplicación de las dos reglas sintácticas **fa** y **ba** da lugar a los tres árboles sintácticos siguientes:


(2.2a) que pasa por el punto $(-3,2)$
(NP\NP)/(S\NP) (S\NP)/(NP\NP) (NP\NP)/NP NP/NP NP NP\NP
fa
NP
ba
NP
fa
NP\NP
fa
S\NP
fa
NP\NP


(2.2b) que pasa por el punto $(-3,2)$
(NP\NP)/(S\NP) (S\NP)/(NP\NP) (NP\NP)/NP NP/NP NP NP\NP
ba
NP
fa
NP
fa
NP\NP
fa
S\NP
fa
NP\NP


(2.2c) que pasa por el punto $(-3,2)$
(NP\NP)/(S\NP) (S\NP)/(NP\NP) (NP\NP)/NP NP/NP NP NP\NP
fa
NP
fa
NP\NP
fa
S\NP
fa
NP\NP

De estos, el árbol (2.2c) no representaría un árbol completo, puesto que al final se obtienen dos categorías sobre las que no se puede aplicar ninguna de las reglas sintácticas. Los árboles (2.2a) y (2.2b) son prácticamente equivalentes, de hecho la semántica de ambas subexpresiones sería la misma. Sin embargo siempre que se tenga un determinante seguido de un nombre y de una fórmula, es preferible aplicar **ba** y **fa** como en (2.2b) realizando así una poda temprana en el árbol sintáctico.

2.4.2 El parser

Una vez tomada la decisión de utilizar un formalismo de gramática categorial, podemos utilizar dos tipos de parser: bottom-up con backtracking o top-down con backtracking. El más utilizado por motivos tanto de eficiencia como conceptuales es el primero (Wesche y König, 1990), y es el que nosotros hemos elegido.

En el caso de gramáticas categoriales básicas, la definición de un algoritmo de parsing es bastante sencillo debido a que los pasos para la eliminación del operador *slash* (/ o \) corresponden a las reglas de la aplicación funcional:

```
X/Y     Y     =>     X        forward application (fa)

Y       X\Y   =>     X        backward application (ba)
```

El análisis bottom-up con backtracking, también llamado análisis shift-reduce, es debido a (Ades y Steedman, 1982). Las operaciones *shift-* y *reduce-* son definidas con respecto a una pila de símbolos de categorías. En la siguiente definición del parser, el signo · separa la categoría que se encuentra más arriba en la pila del primer lexema del resto del argumento inicial. La regla (*shift*) reemplaza la siguiente palabra del argumento de entrada por una de las categorías asignadas por el léxico. Las reglas de reducción implementan las reglas de la aplicación funcional.

Definición (Shift-reduce-parser para gramáticas categoriales básicas)

Sea L el léxico, g la categoría meta, w el lexema, U la secuencia de categorías, α el argumento de entrada, z := g.

(axioma) $x \cdot \vdash x$

(shift) $$\frac{x \in L(w) \quad U,x \cdot \alpha \vdash z}{U \cdot w, \alpha \vdash z}$$

(reduce\) $$\frac{U,x \cdot \alpha \vdash z}{U,y,x\backslash y \cdot \alpha \vdash z}$$

$$\text{(reduce/)} \quad \frac{U, x \cdot \alpha \rightarrow z}{U, x/y, y \cdot \alpha \rightarrow z}$$

Como ilustración, veamos el mismo ejemplo del apartado anterior tomado del Problema 1, i. e., el análisis de *"que pasa por el punto (-3,2)"*. En este caso la categoría meta *z* es *NP\NP* y el léxico *L* contiene las categorías sintácticas del texto que va a ser analizado.

(2.2a)

(L = {<a, {(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP, NP/NP, NP, NP\NP}>}.)

•a,a,a,a,a,a → NP\NP	(shift)
(NP\NP)/(S\NP)•a,a,a→NP\NP	(shift)
(NP\NP)/(S\NP), (S\NP)/(NP\NP)•a,a,a→NP\NP	(shift)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP•a,a,a→NP\NP	(shift)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP, NP/NP•a,a,a→NP\NP	(shift)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP, NP/NP, NP•a,a,a→NP\NP	(reduce/)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP, NP•a,a→NP\NP	(shift)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP, NP, NP\NP• →NP\NP	(reduce\)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP, NP • →NP\NP	(reduce/)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP) • →NP\NP	(reduce/)
(NP\NP)/(S\NP), (S\NP) • →NP\NP	(reduce/)
(NP\NP) • →NP\NP	(axioma)

Este ejemplo corresponde a la formalización del árbol 2.2a. En este análisis se ha aplicado *shift* hasta que en la secuencia de categorías *U* se encuentran de forma consecutiva *NP/NP* y *NP*. En ese momento se aplica *reduce/*, que equivale al análisis de *"el punto"*. Con *reduce* se obtiene la categoría sintáctica de *"el punto $(-3,2)$"*. Con la aplicación consecutiva de *reduce/* se obtiene el análisis de *"por el*

punto $(-3,2)$", *"pasa por el punto $(-3,2)$"* y *"que pasa por el punto $(-3,2)$"* respectivamente.

(2.2b)

(L = {<a, {(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP, NP/NP, NP, NP\NP}>}.)

•a,a,a,a,a,a → NP\NP	(shift)
(NP\NP)/(S\NP)•a,a→NP\NP	(shift)
(NP\NP)/(S\NP), (S\NP)/(NP\NP)•a,a→NP\NP	(shift)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP•a,a→NP\NP	(shift)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP, NP/NP•a,a→NP\NP	(shift)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP, NP/NP, NP•a,a→NP\NP	(shift)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP, NP/NP, NP, NP\NP•a→NP\NP	(reduce\)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP, NP/NP, NP • →NP\NP	(reduce/)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP, NP • →NP\NP	(reduce/)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP) • →NP\NP	(reduce/)
(NP\NP)/(S\NP), (S\NP) • →NP\NP	(reduce/)
(NP\NP) • →NP\NP	(axioma)

Este ejemplo corresponde a la formalización del árbol 2.2b. En este análisis se ha aplicado *shift* hasta que en la secuencia de categorías *U* se encuentran de forma consecutiva *NP* y *NP\NP*. En ese momento se aplica *reduce*, que equivale al análisis de *"punto $(-3,2)$"*. Con *reduce/* se obtiene la categoría sintáctica de *"el punto $(-3,2)$"*. El resto del proceso es similar al anterior.

(2.2c)

(L = {<a, {(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP, NP/NP, NP, NP\NP}>}.)

•a,a,a,a,a,a → NP\NP	(shift)
(NP\NP)/(S\NP)•a,a→NP\NP	(shift)
(NP\NP)/(S\NP), (S\NP)/(NP\NP)•a,a→NP\NP	(shift)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP•a,a→NP\NP	(shift)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP, NP/NP•a,a→NP\NP	(shift)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP, NP/NP, NP•a,a→NP\NP	(reduce/)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)/NP, NP•a,a→NP\NP	(reduce\)
(NP\NP)/(S\NP), (S\NP)/(NP\NP), (NP\NP)•a,a→NP\NP	(reduce/)
(NP\NP)/(S\NP), (S\NP)•a,a→NP\NP	(reduce/)
(NP\NP)•a,a→NP\NP	(shift)
(NP\NP), NP\NP• →NP\NP	

Este ejemplo corresponde a la formalización del árbol 2.2c. En este análisis se ha aplicado *shift* hasta que en la secuencia de categorías *U* se encuentran de forma consecutiva *NP/NP* y *NP*. En ese momento se aplica *reduce/*, que equivale al análisis de *"el punto"*. Con *reduce* se obtiene la categoría sintáctica de *"por el punto"*. Aplicando *reduce/* se obtiene el análisis de *"pasa por el punto"* y *"que pasa por el punto"*.

2.4.3 El diccionario

El diccionario se construye a partir de la definición formal de los conceptos y el léxico propiamente dicho que describe el comportamiento de las palabras. Sistematizando su creación, se han definido un conjunto de rasgos básicos formados por pares atributo-valor para describir cada entrada: su **spelling**, su **categoría** sintáctica asociada y su expresión **semántica**.

2.4.3.1 Spelling

El valor de este atributo puede ser uno de los siguientes:

- Las palabras tal cual aparecen en los enunciados, i.e., los verbos, nombres, adjetivos y determinantes en su forma flexionada, con sus rasgos de género y número. Las preposiciones, adverbios y conjunciones en su forma canónica. Algunos ejemplos de los valores de este atributo en nuestro léxico serían los siguientes: *"hallar"*, *"una"*, *"ecuación"*, *"paralela"*, *"y"*.
- La composición de las formas anteriores, que consiste en la secuencia de éstas separadas por el signo *"+"*. Se obtiene durante el análisis cuando los métodos sintácticos y semánticos han sido aplicados con éxito. Es entonces cuando se crea un nuevo elemento del léxico donde el valor de su spelling es la concatenación de las de los elementos que han intervenido en el análisis. Algunos ejemplos son *"la+recta"* y *"por+el+punto+(3, 0)"*. El primer ejemplo es el resultado de analizar *"la"* y *"recta"* mientras que el segundo corresponde al análisis de *"por"* y *"el punto (3, 0)"*. En ningún caso correspondería al análisis de *"por el"* con *"punto (3, 0)"* ni al de *"por el punto"* con *"(3, 0)"*.

2.4.3.2 Categoría sintáctica

En este atributo se codifica la categoría sintáctica asociada. Expresada en términos de una gramática categorial básica. En virtud de esta gramática, los valores de este atributo pueden ser:

- categorías simples, como es el caso de *"punto"*, cuyo valor sintáctico es *NP*.
- categorías compuestas, obtenidas a partir de las simples utilizando los operadores binarios / y \. Un ejemplo sería la definición del determinante cuyo atributo sintáctico tiene como valor *NP / NP*.

La mayor parte de las categorías son "representaciones" lógicas del papel que desempeñan en la oración; así, si suponemos que todos los nombres tienen categoría *NP*, los determinantes son *aquellas palabras que preceden al nombre dentro de un sintagma nominal*, que en términos de una gramática categorial se codifica como *NP / NP*, y así sucesivamente.

Sin embargo no todas las definiciones de las palabras tienen una representación única. Uno de los ejemplos más comunes de disyunciones sintácticas en nuestro dominio está formado por las fórmulas. Éstas pueden aparecer desempeñando dis-

tintas funciones gramaticales como ocurre con las dos que aparecen en el enunciado del Problema 1, que escribimos señalando en **negrita** las fórmulas:

```
"Escribir una ecuación para la recta paralela a 3x-5y+8=0 y que
 pasa por el punto (-3,2)."
```

La primera, *"3x-5y+8=0"*, desempeña un papel similar al de un sintagma nominal, mientras que la segunda, *"(-3,2)"*, modifica a un sintagma nominal. Por tanto las fórmulas pueden tener muchas funciones sintácticas y semánticas diferentes.

Otro ejemplo que se debe mencionar es el de la conjunción *"y"* o *"e"*. Estas partes invariables del discurso unen dos palabras o dos proposiciones, siempre que estas desempeñen el mismo papel sintáctico. Por tanto, la conjunción *"y"* del enunciado anterior no puede unir en ningún caso las palabras *"3x-5y+8=0"* y *"que"*, ni las proposiciones *"una ecuación para la recta paralela a 3x-5y+8=0"* y *"que pasa por el punto (-3,2)"*.

Por otra parte, una vez efectuada la coordinación de constituyentes, el papel sintáctico de la coordinación es el de cualquiera de las proposiciones. Por tanto, en nuestro diccionario, la categoría de las conjunciones es $(X / X)\backslash X$, donde X puede ser cualquier categoría ya sea simple o compuesta.[4]

2.4.3.3 Expresión semántica

El valor del atributo semántico está formado por una expresión tipada del lenguaje formal. Las fuentes por las que se consigue el valor de este atributo serán descritas en el capítulo 3 *"Semántica léxica en Prógenes"*. Algunos de los mecanismos que permiten la combinación semántica de dos elementos serán descritos en el capítulo 4 *"Semántica Composicional en Prógenes"*.

[4] Esta categoría no puede, en principio, ser tratada con las reglas gramaticales expuestas. Su comportamiento específico será tratado en el apartado 2.5.2.

2.5 Implementación de la interfaz. CLOS

La programación orientada a objetos es una disciplina cada vez más popular gracias a las ventajas que ofrece en el desarrollo del software así como en el mantenimiento del código (Brooks, 1990) (Cox, 1990). Dichas ventajas son presentadas como corolarios del intensivo uso que se hace de la encapsulación de datos y las facilidades de reutilización de código que ofrecen los lenguajes de programación orientada a objetos y los sistemas creados con estas técnicas (Cox, 1986) (Stroustrup, 1992). La orientación a objetos es de gran utilidad no sólo en el desarrollo, sino también en la especificación del software y en el diseño (Booch, 1990), (Niertrasz, 1989) (Sernadas, 1993).

Para el diseño e implementación del interfaz de usuario de Prógenes se ha utilizado CLOS (Common Lisp Object System). CLOS es una extensión orientada a objetos del Common Lisp. Está basada en funciones genéricas, herencia múltiple, combinación de métodos declarativos y un protocolo de meta-objetos. Los objetos fundamentales de CLOS son las clases, instancias, funciones genéricas y métodos.

Una *clase* es un objeto que determina la estructura y el comportamiento de un conjunto de otros objetos, llamados *instancias.*

Una clase puede heredar la estructura y el comportamiento de otras clases. Una clase cuya definición hace referencia a otras clases, heredando de ellas, se denomina *subclase* de cada una de las clases de las que herede. Las clases diseñadas para que otras hereden de ellas se denominan *superclases* de dichas subclases.

Cada clase tiene una lista de precedencia de clases, que es un orden total sobre el conjunto de la clase dada y sus superclases. Dicho orden es expresado como una lista cuyos elementos son ordenados desde los más específicos a los menos específicos. La lista de precedencia de clases es utilizada de muchas formas. En general, las clases más específicas pueden ocultar y reescribir atributos que, de otra forma, serían heredados de las clases menos específicas. El proceso de combinación de métodos, así como la selección de estos, utiliza la lista de precedencia para ordenar los métodos por su grado de especificidad.

Las clases se organizan en forma de un grafo acíclico orientado. Existen dos clases que cumplen la función de supremo e ínfimo del conjunto de clases. Dicho conjunto, con el orden de precedencia definido, forman un conjunto parcialmente ordenado.

Las clases son representadas mediante objetos que a su vez son instancias de clases. El término *metaclase* hace referencia a aquellas clases cuyas instancias son a su vez clases. Cada clase puede tener cero o más slots univaluados, que serán heredados por las subclases.

Una *función genérica* es una función cuyo comportamiento depende de las clases o identidades de sus argumentos. Los *métodos* definen un comportamiento específico a la clase, así como operaciones sobre la función genérica. El comportamiento de los objetos *función genérica* es similar al de las funciones estándar de LISP: toman argumentos, realizan una serie de operaciones y devuelven, siempre que sea necesario, valores de utilidad. A diferencia de las funciones LISP, en las que el cuerpo del código es siempre ejecutado, en las funciones genéricas sólo una parte del código es seleccionada para ser ejecutada, dependiendo de la clase de los argumentos.

Los métodos no son funciones y por lo tanto no pueden ser invocados como tal. Los métodos son invocados desde las funciones genéricas, dependiendo siempre del tipo de los argumentos; en ese caso se ejecuta el código del cuerpo del método. Las ambigüedades que puedan surgir al invocar métodos con el mismo nombre sobre la misma clase de argumentos se resuelven mediante el tipo de los argumentos asociados a dichos métodos y las palabras clave asociadas (:primary, :before, :around, etc). El funcionamiento de las funciones genéricas se ha adaptado al sistema de gramáticas no monótonas (Gonzalo, en preparación) que hemos experimentado en este sistema.

A continuación veremos cómo se utilizan los objetos y los métodos, basándonos en los módulos descritos en el apartado anterior, a saber diccionario, gramática y parser. Con ello tendremos una descripción del entorno en el que se deben integrar los métodos encargados del análisis semántico y el conocimiento del dominio, aunque no es nuestra intención explicar exhaustivamente todos los detalles de implementación de la interfaz para el procesamiento de los enunciados de Prógenes.

2.5.1 Herencia y gramática

La mayoría de los formalismos lingüísticos objeto de estudio en la actualidad son orientados a objetos en uno u otro sentido. Todos ellos se centran en la representación de los objetos lingüísticos básicos: palabras y signos en general. Desde que se desarrolló el concepto de *hierarchical lexicon* (Flickinger et al., 1985), una de las prioridades de la investigación ha sido la de desarrollar y estudiar los marcos formales para representar objetos lingüísticos y las relaciones entre ellos.

No sorprende, pues, encontrar modelos complejos de representación lingüística que descansan esencialmente sobre principios adoptados en POO (el mejor ejemplo es HPSG (Head-Driven Phrase Structure Grammar) (Pollard y Sag, 1993)). Estos principios están relacionados esencialmente con el concepto de *herencia*. La *herencia por defecto* y la *herencia múltiple*, como sistemas de organizar la información en el léxico, son consideradas como una forma elegante de evitar la información redundante, de organizar el conocimiento lingüístico y de restringir el espacio de descripciones admisibles (ver (Daelemans et al., 1992)).

El conocimiento lingüístico está distribuido entre las reglas gramaticales y el léxico. En formalismos basados en la herencia suele ser el léxico el que alberga casi toda la información, y las reglas gramaticales juegan un papel marginal (Flickinger and Nerbonne, 1992). De hecho, la tendencia es la de reducir las reglas gramaticales a unos pocos principios generales. Es el caso de las reglas de las Gramáticas Categoriales, de los principios de HPSG o del concepto de Unificación en otros formalismos.

Uno de los propósitos de la interfaz de Lenguaje Natural de Prógenes es mostrar que un léxico jerarquizado mediante una red de herencia permite el desarrollo de un conjunto de reglas organizadas también de acuerdo con ella. Al contrario que las reglas de una Gramática Libre de Contexto (CFG), una gramática orientada a objetos (la llamaremos OOG) puede ser modular y de tamaño restringido. La principal diferencia entre una CFG y una OOG es que una regla CFG se especifica sobre *no terminales*, que en un contexto lingüístico coinciden con constituyentes. Sin embargo, una regla OOG se especifica sobre *clases* definidas previamente en la red de herencia. Definiendo *reglas por defecto* sobre las clases superiores de la jerarquía debe reducirse drásticamente el tamaño de las gramáticas. Definiendo reglas específicas (sobre las clases inferiores de la jerarquía) podemos resolver situaciones excepcionales de una forma modular: en una CFG es difícil tratar este tipo de situaciones sin que las reglas generadas entorpezcan el análisis y produzcan efectos laterales.

En un sistema así, el conocimiento empleado para construir la jerarquía de clases es compartido por el léxico y la gramática.

Todos los conceptos necesarios para adaptar las reglas gramaticales a una red de herencia se encuentran en los principios de Programación Orientada a Objetos (Booch, 1990). Dentro de este paradigma, los objetos se organizan jerárquicamente de la misma forma en la que los signos lingüísticos se organizan en un léxico jerarquizado. Se pueden definir funciones que actúan sobre estos objetos. Pero, al contrario que en programación convencional, estas funciones están formadas por *métodos.* Los métodos se diferencian entre sí por las clases de los argumentos sobre los que se aplican. Se definen independientemente unos de otros, y nunca son llamados directamente. Cuando se llama a una función, las clases de los argumentos, junto con la red de herencia, deciden qué método será usado. El conjunto de los métodos aplicables está formado por todos los métodos definidos sobre clases compatibles con las de los argumentos. Dentro de este conjunto, el método usado finalmente es el más específico (a este proceso se le llama *dynamic binding* en POO).[5]

En la interfaz de Prógenes, una regla gramatical es el equivalente lingüístico de un método. En particular, al concepto de método con las restricciones y características propias de CLOS (Steele, 1990), que es el lenguaje usado para su implementación. Definimos reglas gramaticales que son operativas sobre ciertas clases de signos. Estas reglas se agrupan en *reglas genéricas.* Cuando se aplica una regla genérica sobre dos signos, sus clases determinan qué regla será usada. De esta forma no hay ya necesidad de definir reglas sobre cada par posible de clases. El diseño jerárquico nos permite definir reglas por defecto que sean siempre, o casi siempre, aplicables. Reglas específicas sustituirán a las anteriores en situaciones excepcionales.

Existen, además, motivaciones computacionales para adoptar este sistema. Los fenómenos complejos del lenguaje natural, como la coordinación o las dependencias a larga distancia, son tratados dentro de las implementaciones mediante dispositivos extragramaticales. Cierto número de ellos es *demon-based* (Haugeneder, 1992). En (Huang, 1983), por ejemplo, una subgramática específica de la coordinación se dispara sólo cuando se la requiere. Dentro de una aproximación orientada a objetos, cada regla puede ser vista como un *demon.* Al ser definida, cada

5 Decidir qué método es el más específico no es siempre una tarea simple. Las dificultades son similares a aquellas que se encuentran al manejar redes con herencia múltiple o herencia por defecto.

regla adquiere automáticamente el grado de especificidad adecuado. Además, cualquier procedimiento particular puede ser integrado dentro de la gramática de una forma totalmente modular. Finalmente, los lenguajes de programación orientados a objeto permiten implementar este tipo de gramáticas de forma bastante sencilla.

2.5.2 Uso de clases en Prógenes. Herencia

La reunión de todos los pares atributo-valor asociados a cada palabra constituye asimismo una estructura, que se unifica con las de su entorno durante la fase de análisis, hasta dar lugar a la estructura representativa del enunciado.

Todos los objetos involucrados en el análisis pertenecen, inicialmente, a la clase principal *signo*. Generalizando a partir de los rasgos definidos en el diccionario, podemos definir la clase general cuyos atributos serían, inicialmente, los ya descritos anteriormente: su spelling, categoría sintáctica asociada y su expresión semántica.

Dependiendo del modo en el que se implementen las reglas de la gramática, la categoría sintáctica se puede codificar de distintas formas. Nosotros hemos optado por representar cada uno de los papeles sintácticos en un único atributo que contiene, en forma de lista, la categoría de la palabra. El primer elemento de la lista sería la categoría principal de la palabra, el segundo sería el operador y el tercero la categoría buscada por el operador. A su vez, el primer y tercer elemento de esta lista pueden ser listas. Un ejemplo sería el pronombre *"que"* cuyo papel sintáctico se representaría con la lista *"((NP \ NP) / (S \ NP))"*.

Resumiendo lo expuesto, la clase signo se codificaría en CLOS como se ilustra a continuación.

```
(defclass signo ()
  ((spelling :initarg :spelling :initform nil            :accessor spelling)
   (cat      :initarg :cat      :initform nil            :accessor cat)
   (sem      :initarg :sem      :initform '((OBJECT @o)) :accessor sem)
  ))
```

En todos los especificadores de slot, la unión de los argumentos de inicialización declarados en la etiqueta *":initarg"* especifica el conjunto de argumentos que inicializan un slot dado.

En la etiqueta *":initform"* se asignan los valores por defecto; en esta definición el único significativo es el semántico *((OBJECT @o))*.

En la etiqueta *":accessor"* se asigna el nombre de la función que nos permite extraer, de cada una de las instancias del objeto, el valor de un atributo. Estos nombres de funciones son heredadas por todas las instancias.

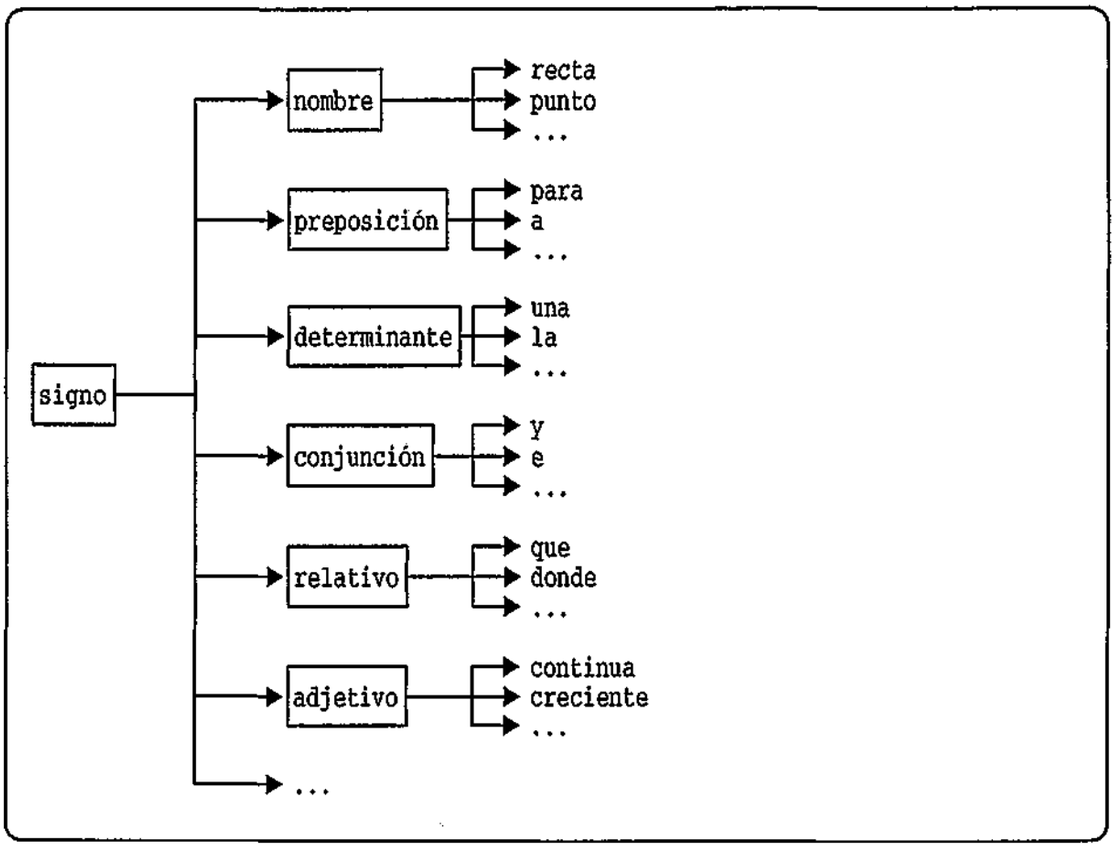
Todo el léxico podría ser definido como instancias de esta clase general *signo,* sin embargo, las palabras se pueden agrupar en subclases, teniendo en cuenta su papel sintáctico. Estructurando de esta forma el léxico, podemos utilizar la herencia que nos proporciona CLOS, evitando así la repetición de los valores de los atributos sintácticos en todos los determinantes, nombres, verbos, etc. El propósito de estas subclases es definir valores que serán heredados por las instancias comunes. Algunos ejemplos son los siguientes:

```
(defclass nombre (signo)
   ((cat :initform '(NP))))

(defclass preposicion (signo)
   ((cat :initform '((NP \ NP) / NP))))

(defclass determinante (signo)
  ((cat :initform '(NP / NP))))

(defclass conjuncion (signo)
  ((cat :initform '((X / X) \ X))))

(defclass relativo (signo)
  ((cat :initform '((NP \ NP) / (S \ NP)))))

(defclass adjetivo (signo)
  ((cat :initform '(NP \ NP))))

(defclass verbo-transitivo (verbo)
  ((main-cat :initform '((S \ NP) / NP))))

...
```

De esta forma, los elementos del léxico son instancias de las subclases anteriores con valores específicos en los atributos *spelling* y *sem* siempre que sea necesario.

Estas instancias pueden representar tres tipos de elementos del discurso distintos:

- **Léxico permanente** que, como hemos visto, está dividido en subclases. En la siguiente figura, se ilustra la jerarquía de este tipo de objetos.



- **Léxico dinámico:** Cuando el analizador encuentra una fórmula, como las ya tratadas en ejemplos anteriores, se crean sendas instancias de *nombre*, NP, y *adjetivo*, NP\NP, con dichas fórmulas. Su semántica tiene valor aunque no completo hasta que el proceso de análisis lo resuelva.
- **Análisis parcial:** Cada éxito parcial durante el análisis produce un objeto de la clase *signo*.

En el Apéndice A se muestran algunos ejemplos de la representación del léxico permanente (o estático) así como de los análisis parciales y del léxico dinámico que pueden ocurrir durante el proceso.

2.5.3 Uso de métodos en Prógenes. Gramática

Las reglas sintácticas del sistema se aplican sobre un par contiguo y ordenado de objetos para crear otro objeto del que son sus hijos. El objetivo de este apartado es esbozar la forma de estos métodos, de las funciones genéricas que los engloban y el modo en el que funcionan. Nos centraremos en los métodos sintácticos y en la implementación de reglas gramaticales aunque sin efectuar una enumeración minuciosa de todas las funciones genéricas que se han implementado para tratar excepciones gramaticales. Unicamente se desea demostrar la sencillez de su código basándonos, principalmente, en la particular implementación que se ha dado a *forward application*, *backward application*, y a la conjunción.

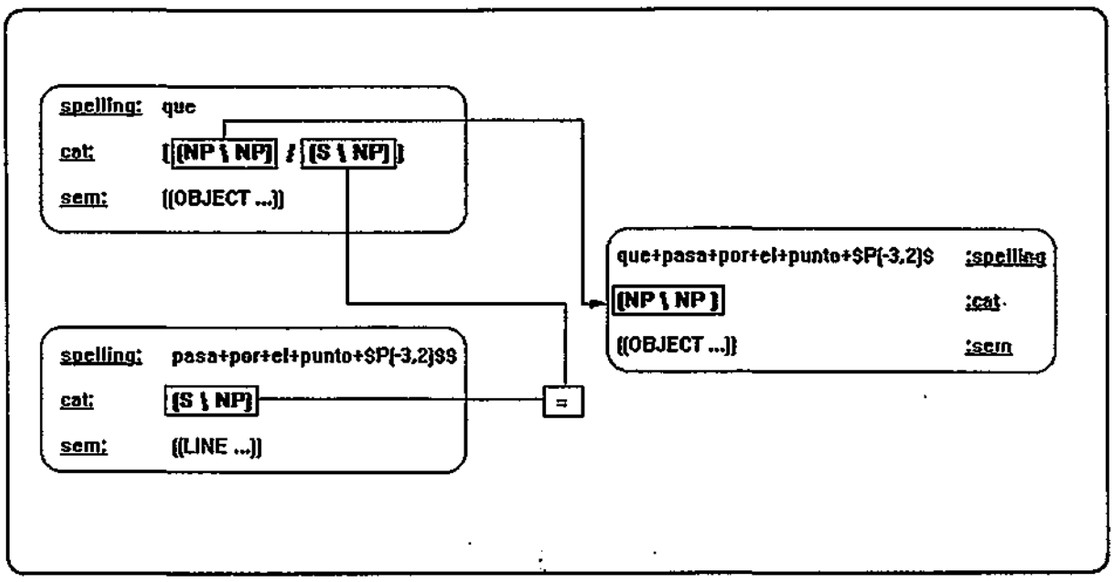
Aunque las reglas semánticas están directamente relacionadas con las sintácticas, su descripción se efectuará en el capítulo 4, una vez que se haya definido el lenguaje formal Prógenes.

2.5.3.1 Forward application

Como ya hemos dicho, tal y como está definida la categoría gramatical, los métodos sintácticos sólo tienen que "comparar", de forma adecuada, los valores de la lista *cat* y asignar los nuevos cuando proceda. El código de la función genérica que efectua la regla *"forward"* es el que se ilustra a continuación:

```
(defmethod FORWARD-APPLICATION ((w1 signo) (w2 signo))
  (when (and
          (oper-included '/ (oper w1))
          (equal (oper-cat w1) (cat w2)))
        (NCAT (main-cat w1))))
```

La función *NCAT* se encarga de asignar la categoría principal de la primera palabra, i.e., el primer elemento de la lista, al atributo *cat* del nuevo objeto generado. La función *oper-included* chequea que sus argumentos sean iguales, o bien que el operador sea |. Las funciones *main-cat*, *oper* y *oper-cat* chequean el primero, segundo y tercer elemento de la lista que forma la categoría. En la siguiente figura se ilustra su funcionamiento.

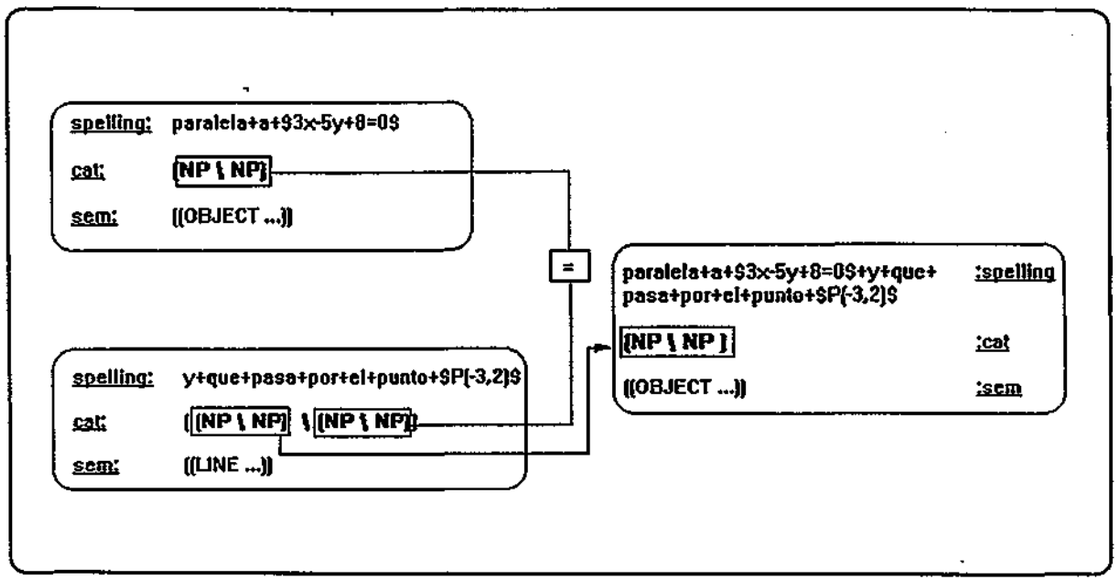


2.5.3.2 Backward application

Esta regla tiene, al igual que la anterior, dos argumentos de tipo signo, siendo su código asociado:

```
(defmethod BACKWARD-APPLICATION ((w1 signo) (w2 signo))
  (when (and
          (oper-included '\\ (oper w2))
          (equal (oper-cat w2) (cat w1)))
        (NCAT (main-cat w2))))
```

En este caso, es el papel sintáctico de la segunda palabra el que predomina sobre el de la primera, como se puede observar en el siguiente ejemplo:



2.5.3.3 Coordinación de constituyentes

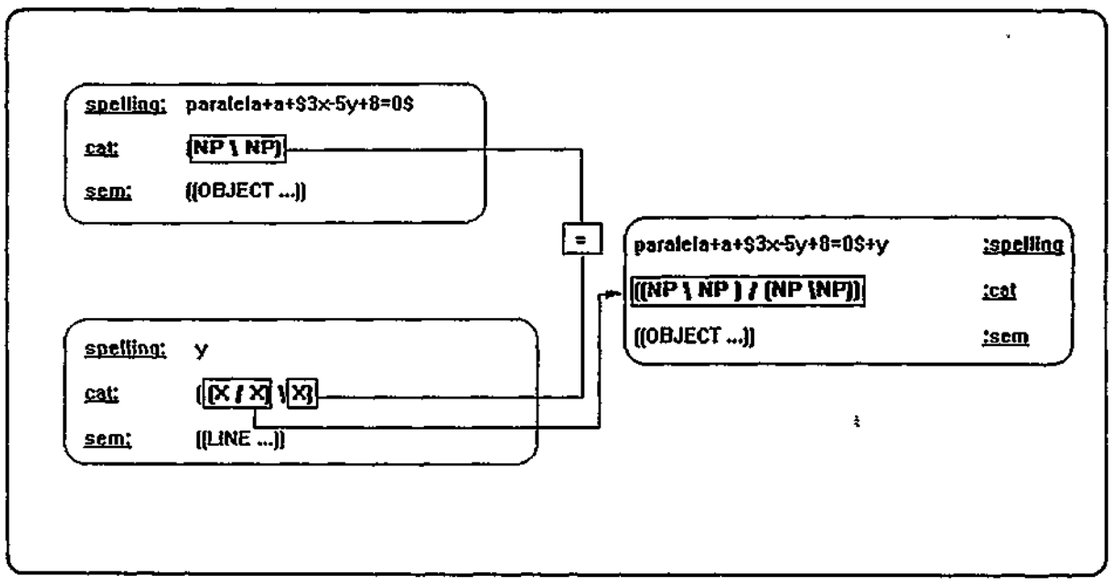
Como ya mencionamos en el apartado 2.4, el papel sintáctico de las conjunciones *"y"* y *"e"* es representado por *(X / X) \ X*, donde *X* puede ser cualquier categoría, ya sea simple o compuesta. Las conjunciones pueden ser detectadas de distintas formas. Una de ellas consiste en examinar el valor del atributo *base*, que debe ser *"y"*. Otra es, simplemente, viendo que es una instancia del tipo conjunción. El código de la regla que nos permite analizar estas partes del discurso es el siguiente:

```
(defmethod BACKWARD-APPLICATION ((w1 signo) (w2 conjuncion))
   (NCAT (list (cat w1) '/ (cat w1))))
```

En él se detecta que el segundo objeto pertenece al tipo *conjunción*, y codifica el funcionamiento de la siguiente regla, en la que *CAT* puede ser cualquier categoría.

```
CAT  ((X / X) \ X) → CAT / CAT
```

El funcionamiento sintáctico de esta regla se ilustra en el siguiente ejemplo.



Capítulo 3

3.0 Semántica léxica en Prógenes

Como ya se ha mencionado en el Capítulo 1, un formalismo semántico, en este caso aplicado a Prógenes, se pronuncia respecto de la naturaleza de las representaciones semánticas y de sus relaciones con aquello sobre lo que la lengua nos permite hablar. En este capítulo se determinará la forma de las expresiones semánticas, la caracterización formal de los objetos que sirven como representaciones semánticas y el tipo de asociación que se puede establecer entre las expresiones con sus representaciones correspondientes.

Para ello se va a describir el lenguaje formal Prógenes. Comenzaremos determinando el tipo de elementos que aparecen en las bases de conocimiento, su función y su sintaxis general. Sin embargo, no se va a tratar el fundamento de estas bases de conocimiento, que es ampliamente descrito en (Castells, 1994) y (Saiz, 1994). A continuación se formalizará la jerarquía de tipos, junto a las estructuras matemáticas que cumple. En los últimos apartados se explicará la descripción y extracción del lenguaje formal a partir de la Base de Conocimientos.

3.1 Elementos de la Base de Conocimientos

En la Base de Conocimientos (BC) se encuentran todos los objetos del dominio junto con las relaciones que se pueden establecer entre ellos. Se encuentra, por tanto, la mayor parte del conocimiento declarativo del sistema, así como el procedural, por contener también las acciones, procedimientos y teoremas del módulo de resolución.

La base de conocimientos está formada por un conjunto de subbases en las que se encuentra el conocimiento preceptivo para la interpretación y resolución de los enunciados en cada área (conocimiento general, geometría euclídea, álgebra, etc.). En general, los elementos que podemos encontrar en la base de conocimientos son:

- **Objetos:** entidades estructuradas que tienen asignado un tipo y pueden incluir uno o varios argumentos (*slots*), que se entienden como elementos descriptores de la entidad de que se trate.
- **Funciones:** aplicaciones cuyo origen e imagen es el conjunto de objetos de la BC.

En los siguientes apartados veremos cómo se construyen estos objetos.

3.1.1 Objetos

Los objetos de la BC pueden ser definidos de distintas formas; concretamente pueden ser *átomos*, *objetos compuestos* o *constantes*, siendo representados en la BC como *term*, *obj* y *const* respectivamente. La definición de las características de cada objeto de la BC realiza una definición de su tipo.

Los objetos atómicos son tipos simples predefinidos por el intérprete LISP. Ejemplos de este tipo de elementos son *STRING*, *NUM* y *BOOL*.

Los objetos compuestos son entidades complejas estructuradas, que tienen asignado un tipo. En su definición pueden intervenir objetos simples o compuestos, respetando siempre la siguiente sintaxis:

```
(obj  <name> ({<compulsory slot>}* [<key-list> | <rest-list>])
      <type>
      <code>)

<name> := Nombre del objeto.

<type> := Alguno de los tipos válidos de la jerarquía.

<compulsory slot> := (<name> | (<name> {<usual-name>}*)
                      <type> [<expansion code>])

<expansion code> := una única expresión LISP

<key-list> := &key {<optional slot>}+

<optional slot> := (<name> <type> [ <slot code> [<default>]])

<slot code> := Una única expresión LISP
               (puede contener referencias al constructor mediante
               <name>)

<default> := Una única expresión LISP
             (puede contener referencias al constructor mediante
             <name>)

<rest-list> := &rest <rest slot>

<rest-slot> := (<name> <type>)

<code> := Código LISP con un "progn" implícito
          (puede contener referencias a los slots y al constructor)
```

Obsérvese que el número de elementos que lo definen puede estar determinado o no. Veamos algunos ejemplos pertenecientes a distintos dominios:

```
(obj muelle ((rigidez NUM)
             (posicion-agarre POINT)
             (tension NUM))
     SPRING
     ())
```

Para este objeto, las tres primeras líneas definen no sólo el tipo de los tres argumentos, sino también la definición implícita de otros conceptos como la *rigidez*, la *posición de agarre* y la *tensión* de un muelle. La cuarta línea denota el tipo asignado al concepto, que en este caso coincide con su nombre. A partir de la quinta, aparecería la información propia del módulo de resolución, en aquellos casos en los que sea necesario. La palabra siguiente a *obj*, **muelle**, representa la traducción de la palabra enunciada en lenguaje natural al lenguaje formal, que en este caso coincide.

La definición del objeto compuesto *parábola* presenta un caso diferente:

```
(obj parabola ((ecuacion EQUATION)
               &optional (rotacion MATRIX)
                         (translacion VECTOR)
                         (distancia-focal REAL)
               ))
    PARABOLA
    (parabolic ecuacion))
```

Aquí aparece un único argumento obligatorio, *ecuación*, y tres argumentos opcionales, *rotación*, *translación* y *distancia focal*. Su tipo vuelve a coincidir con el nombre del objeto compuesto y en la última línea se puede observar el código del módulo de demostración, que en este caso aplicaría la restricción *(parabolic ecuacion)* sin la cual la definición del objeto *parabola* no sería correcta.

Veamos ahora otro ejemplo, el objeto *vector*, cuya definición en la BC es:

```
(obj vector (&rest (coordenadas REAL))
     VECTOR
     (= *ambdim* (length (cdr coordenadas)))
)
```

En este caso el objeto compuesto *vector* está formado por un número indefinido de coordenadas; se consigue así que la misma definición sea válida si estamos en el plano bidimensional o tridimensional, en los que el número de *coordenadas* serían 2 y 3 respectivamente.

Los ejemplos de objetos compuestos descritos hasta el momento son definiciones declarativas explícitas. Sin embargo, en ellas también podemos encontrar, de forma implícita, la definición de nuevos elementos. Es el caso de la *rigidez* de un *muelle*, la *ecuación* de una *parábola* y las *coordenadas* de un *vector*. La definición desarrollada de estos conceptos sería la siguiente:

```
(obj rigidez (muelle SPRING)
     NUM
     ())

(obj ecuacion (parabola PARABOLA)
     ECUATION
     ())

(obj coordenadas (vector VECTOR)
     REAL
     ())
```

Obsérvese que la definición de *rigidez* y *coordenadas* ilustra definiciones de objetos en los que su tipo no coincide con su nombre.

Para justificar el uso de las definiciones implícitas de objetos compuestos, veamos la definición del objeto *punto*:

```
(obj punto (&rest (coordenadas REAL))
     POINT
     (= *ambdim* (length (cdr coordenadas)))
)
```

En esta definición se encuentra también, de forma implícita, la del objeto *coordenadas* de un *punto*, cuya expansión es:

```
(obj coordenadas (punto POINT)
     REAL
     ())
```

Que coincide con la definición de las coordenadas de un vector en el tipo, pero no en sus argumentos. Puesto que la mayoría de los objetos compuestos contienen definiciones implícitas, si en la BC se incluyeran sus declaraciones expandidas, el código crecería considerablemente perdiendo elegancia, además de dar lugar a posibles problemas de mantenimiento.

El último ejemplo de objeto de la BC está formado por el grupo de las constantes cuya sintaxis general es la siguiente: .

```
(constant <name> <type>)
<name> := Nombre de la constante.
<type> := Alguno de los tipos válidos de la jerarquía.
```

Ejemplos de estas constantes son *infinito* y *conjunto vacío*, definidos en la BC como sigue:

```
(constant infinito REAL)
(constant conjunto-vacio SET)
```

3.1.2 Funciones

Como ya hemos visto, las funciones del sistema denotan las aplicaciones de términos y objetos. Su sintaxis general sería la siguiente:

```
(fun <name> ([<argument>]*
             [&optional [<optional argument>]*]
             [&rest <argument>])
     <type>
     <conditions>
     <code>)

<name> := Nombre de la función.

<argument> := (<name> <type>)

<type> := Alguno de los tipos válidos de la jerarquía.

<optional argument> := (<name> | (<name> <default>) <type>)

<default> := Una expresión LISP

<code> := Un grupo de expresiones LISP unidas por progn
```

Es decir, para definir una *función* es necesario especificar su nombre, los argumentos sobre los que se aplica, su tipo de retorno, las condiciones que deben cumplir los argumentos y el código asociado al cuerpo de dicha función, necesario sólo en el módulo de resolución, y que consiste en el encadenamiento de funciones del intérprete LISP o de las del dominio Prógenes.

Veamos algunos ejemplos. El primero describe la *directriz* de una *parábola*. Esta función tiene un único argumento, *p*, de tipo *PARABOLA*, y su tipo global es *RECTA*. En este caso el argumento no debe cumplir ninguna condición particular y, por último, el código que la define es el que puede observarse a partir de la tercera línea.

```
(fun directriz ((p PARABOLA))
     LINE
     ()
     (let ((m (rotacion-parabola p)))
          (make-line
            (matriz** m (p+v (punto (/ (dist-focal-parabola p) -2) 0)
                             (translacion-parabola p)))
            (column 2 m))))
```

Veamos a continuación la definición de la función *hallar*, que también es válida para las palabras del lenguaje natural *"escribir"* y *"calcular"*, entre otras. Esta función es de tipo *OBJECT*, y se puede aplicar sobre cualquier elemento de este tipo, satisfaciendo un conjunto arbitrario de condiciones booleanas. Para no hacer la explicación de los ejemplos innecesariamente complicada y, puesto que para la formulación del lenguaje formal no es necesario conocer el cuerpo de las funciones, a partir de ahora la mayor parte del código será sustituido por *"..."*

```
(fun hallar ((obj OBJECT) &rest (satisface BOOL))
     OBJECT
     ()
     (cond ((and (listp obj) (eq 'a (car obj))) (a (hallar (cadr obj))))
           (t (let ((variables (vars obj '%)))
                (...))))
```

El elemento de la BC representa aquellos verbos del lenguaje natural sinónimos de *"probar"* o *"demostrar"*. Su tipo es *booleano*, i.e. verdadero o falso, y se obtiene al aplicar la función *probar* sobre un *objetivo* también de tipo *booleano*. De forma opcional, a esta función se le pueden proporcionar dos argumentos más, la *hipótesis* y las indicaciones necesarias. De hecho, atendiendo a la definición de argumentos opcionales, puede aparecer uno, los dos o ninguno de ellos.

```
(fun probar ((objetivo BOOL) &optional (hipotesis BOOL) (usando BOOL))
     BOOL
     ()
     (if hipotesis (condicion hipotesis))
     (if usando (add *hints* usando))
     (cond ((atom objetivo) (...)))
)
```

Al igual que ocurría con los objetos, la definición de las funciones no es única, aunque, a diferencia de estos, en la BC puede aparecer una definición distinta para cada una de ellas. Este es el caso de la palabra del lenguaje natural *"intersección"*, que se define de forma distinta según el tipo de sus argumentos. Existe, por tanto, una definición para la intersección de rectas, de curvas o de conjuntos. Las tres definiciones tienen la misma aridad, sin embargo, sus tipos, los de sus argumentos, y su código asociado son totalmente distintos. Mientras que los tipos de la primera

y tercera definición son terminales, en el segundo aparece una disyunción. Este hecho quiere decir que la intersección de dos curvas puede ser un *punto* o una *lista de puntos*, dependiendo de la dimensión de los espacios en los que hayan sido definidas las curvas.

```
(fun inter ((line1 LINE) (line2 LINE))
     POINT
     (not (paralelo line1 line2))
     (let* ((...))))

(fun inter ((c1 CURVE) (c2 CURVE))
     (type-or POINT (type-list POINT))
     ()
     ())

(fun inter ((set1 SET) (set2 SET))
     SET
     ()
     (declare (special set1 set2))
     (mcase set1 ...)
)
```

3.1.3 Casos particulares: funciones de asignación

En la BC existe un reducido grupo de funciones que, por el particular papel que desempeñan, justifican un tratamiento por separado. Este grupo está formado por aquellas funciones en las que, de forma implícita, se efectúa una asignación de tipo sobre un objeto. En la BC estas funciones son denominadas *bndfun* (**bind**ing **fun**ctions). En general, en este grupo se engloban aquellas funciones que realizan labores de cuantificación, como es el caso del determinante *"el"*, así como otros cuantificadores más específicos como *"para-todo"* y *"existe"*. Su sintaxis general es similar a la de las funciones, aunque en este caso el terminal *bndfun* encabeza la definición:

```
(bndfun <name> ({<argument>}*
                [&optional {<optional argument>}*]
                [&rest <argument>])
        <type>
        <conditions>
        <code>)

<name> := Nombre de la función de asignación.

<argument> := (<name> <type>)

<type> := Alguno de los tipos válidos de la jerarquía.

<optional argument> := (<name> | (<name> <default>) <type>)

<default> := Una expresión LISP

<code> := Un grupo de expresiones LISP unidas por progn
```

Veamos dos ejemplos representativos: el determinante *"el"* y el cuantificador *"para todo"* [6].

```
(bndfun el ((obj OBJECT) &rest (bool BOOL))
        OBJECT
        ()
        (if bool (...)))

(bndfun para-todo ((obj OBJECT) (bool BOOL))
        BOOL
        ()
        (cond (...)))
```

En la BC, el determinante es una función que se aplica sobre un elemento de tipo *OBJECT* y un conjunto arbitrario de condiciones booleanas. Por ser una función de cuantificación general se deduce que el tipo de su primer argumento será asignado a una variable del dominio. La expresión que efectúa esta *asignación de tipo* se hace explícita en el lenguaje formal, como veremos en el apartado 3.5.

6 El cuantificador *"existe"* es igual a *"para todo"* en lo referente al tipo y al número y clase de argumentos.

El cuantificador específico *para todo* tiene también aridad dos, aunque su tipo es *booleano*.

3.2 Jerarquía de tipos

El álgebra es una importante rama de las Matemáticas puras y quizá sea la que mejor ilustra el hecho de que las Matemáticas estudian las estructuras. Los resultados de esta rama han sido aplicados en disciplinas como la Cristalografía, Ciencias de la Computación y Mecánica Cuántica, entre otras. También se pueden encontrar descripciones de la aplicación de las ideas algebraicas en el estudio de la semántica de los plurales [(Loning, 1989) (Blackburn, 1990)].

Un álgebra es simplemente un conjunto, por ejemplo T, con un conjunto de operaciones definidas. Una operación *n-aria* sobre un conjunto T es una función de T^n en T. Un caso particular lo forman las operaciones binarias, es decir, aquellas que toman dos elementos del conjunto inicial para hallar un tercero. Este tipo de operaciones son especialmente importantes en el álgebra. En la actualidad se están estudiando álgebras cuyas operaciones cumplan ciertas restricciones. Las más comunes en la literatura del procesamiento del lenguaje natural son los retículos.

En nuestro caso las ideas algebraicas se han utilizado para formalizar la jerarquía de tipos del sistema. Estos tipos identifican cada uno de los conceptos definidos en la Base de Conocimientos. El que este conjunto respete las restricciones de un álgebra, concretamente de un retículo, ofrece ventajas a nivel computacional y conceptual. En esta sección veremos la forma en la que los tipos se relacionan y las operaciones binarias definidas sobre los elementos del conjunto de tipos.

La idea clave de aplicar estas estructuras a conjuntos es que los dominios de cuantificación algebraicamente estructurados permiten aproximaciones más reales a la ontología que presupone que el uso del lenguaje natural puede ser fácilmente construido.

La primera relación que se puede establecer entre los elementos del conjunto de tipos es mediante una relación de orden parcial entre pares, enunciando que el primero es un subtipo del segundo o, lo que es igual, el primero es más específico que el segundo. Pero, como veremos, esta relación no es la única que puede establecerse.

Dentro del conjunto de los tipos de Prógenes cabe destacar los siguientes:

- **OBJECT:** Es el supremo, o menor cota superior, del conjunto. Está definido como un supertipo cuyo valor puede ser cualquier otro más específico. Muy

pocos objetos de la BC son de tipo *OBJECT*; la mayoría son identificados por un tipo más concreto.

- **NIL:** Es el ínfimo del conjunto, es decir, es el subtipo más específico de todos.
- **FORMULA:** En los enunciados de nuestro dominio aparecen fórmulas de las que, de forma aislada, sólo se sabe con seguridad su expresión. Sin embargo sólo se puede conocer su tipo específico una vez analizado el contexto. Como vimos en el apartado 2.4, para resolver este problema se le asigna un tipo general, *FORMULA*, que inicialmente sería subtipo de tipos como *LINEAR-EQUATION*, *NUM* o *INTERVAL* y dependiendo del contexto su valor cambiará por el que corresponda.

La relación de orden *ser subtipo de* es utilizada para dar al conjunto de los tipos del dominio la estructura de **retículo**. A lo largo de estos apartados formalizaremos estas estructuras y diseñaremos el diagrama de Hasse correspondiente a este conjunto. Para facilitar la explicación, al conjunto de tipos correspondiente al dominio del primer curso de Matemáticas (de carreras de Ciencias) le denominaremos **T**.

3.2.1 Jerarquía de tipos: formalización algebraica

Definición (Retículos: definición algebraica)

Un retículo es un conjunto no vacío L junto con dos operaciones binarias $\vee$ *(unión) y* $\wedge$ *(intersección) tal que para todos los elementos 'x', 'y' y 'z' en L:*

L1.	$x \vee y = y \vee x$	$x \wedge y = y \wedge x$	(Conmutatividad)
L2.	$x \vee (y \vee z) = (x \vee y) \vee z$	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$	(Asociatividad)
L3.	$x \vee x = x$	$x \wedge x = x$	(Idempotencia)
L4.	$x = x \vee (x \wedge y)$	$x = x \wedge (x \vee y)$	(Absorción)

■

Si en T suponemos que $\vee$ y $\wedge$ son las operaciones binarias que consisten en *buscar el mínimo supertipo común* y *buscar el máximo subtipo común* respectivamente, es fácil ver que las propiedades de la definición se cumplen. En el peor de los casos el resultado de $\vee$ sería el tipo *OBJECT*, y el de la operación $\wedge$ *NIL*.

Los retículos se suelen representar como ternas ordenadas de la forma $<L, \vee, \wedge>$. En particular, la terna correspondiente a T sería < T, *buscar-el-mínimo-supertipo-común, buscar-el-maximo-subtipo-común*>

Un subconjunto de estas estructuras está formado por los subretículos.

Definición (Subretículos)

Si $<L, \vee, \wedge>$ *es un retículo y* $L' \neq \emptyset$ *es un subconjunto de L tal que* $\forall a, b \in L'$ *tanto* $a \vee b$ *como* $a \wedge b \in L'$ *(i.e., L' es cerrado bajo* $\vee$ *y* $\wedge$*), entonces* $<L', \vee', \wedge'>$*, donde* $\vee'$ *y* $\wedge'$ *son* $\vee$ *y* $\wedge$ *restringidas a L', es un subretículo de* $<L, \vee, \wedge>$*.*

■

Veamos algunos ejemplos de subretículos de T. Supongamos los siguientes subconjuntos del conjunto de tipos T:

```
T' = {ECUATION, LINE, CURVE, SUBVAR, SET, OBJECT}

T' = {PLANE, CUADRICA, CURVE, SET, OBJECT}

T' = {MATRIX, OBJECT}

T' = {SEGMENT, SET, OBJECT}

T' = {LEMNISCATA, CURVE, SET, OBJECT}
```

Si $\vee'$ y $\wedge'$ son *buscar-el-mínimo-supertipo-comun* y *buscar-el-máximo-subtipo-comun* en T' respectivamente, de la definición se deduce que cualquiera de los subconjuntos de la figura es un subretículo y por lo tanto un retículo. Por supuesto todo subretículo es a su vez un retículo. El punto crucial de la demostración está en que, tal y como se indica en el enunciado, L' es cerrado bajo $\vee$ y $\wedge$, lo que quiere decir que sus restricciones realmente son operaciones sobre L'. Faltaría por comprobar las condiciones L1 - L4, pero en realidad estas son heredadas de las operaciones originales, teniendo en cuenta que son operaciones restringidas al conjunto L'.

3.2.2 Jerarquía de tipos: formalización basada en el orden

Los retículos también pueden ser definidos teniendo en cuenta órdenes parciales. Para efectuar la definición de retículo atendiendo al orden, es necesario antes introducir algunos conceptos, como son los órdenes parciales, los conjuntos parcialmente ordenados, y su correspondiente representación utilizando diagramas de Hasse.

Definición (Órdenes parciales.)

Una relación binaria $\leq$ sobre un conjunto no vacío P es un orden parcial sobre ese conjunto sii para todos los elementos a y b de P:

1. $a \leq a$ (Reflexividad)
2. $a \leq b$ y $b \leq a \Rightarrow a = b$ (Antisimetría)
3. $a \leq b$ y $b \leq c \Rightarrow a \leq c$ (Transitividad)

Un conjunto no vacío con un orden parcial definido se denomina **conjunto parcialmente ordenado.**

■

Por ejemplo, sea *A* cualquier tipo de conjunto. Definamos *Pot(A)* = $\{B:B \subseteq A\}$, i.e., todos los subconjuntos de *A*. La operación $\subseteq$ es un orden parcial sobre *A*.

Los conjuntos parcialmente ordenados se pueden escribir como pares de la forma $<P,\leq>$. Por tanto, el ejemplo anterior se representaría como $<Pot(A), \subseteq>$. Esta representación también sería válida en nuestro dominio, definiendo $\subseteq$ como *ser subtipo de.*

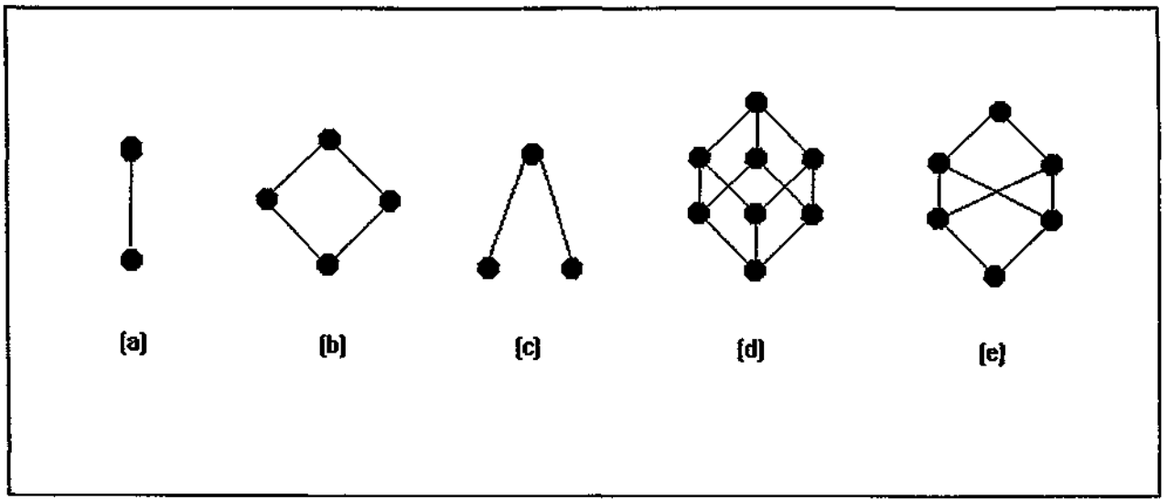
Por otra parte, los conjuntos parcialmente ordenados que sean finitos pueden ser representados gráficamente mediante diagramas de Hasse. Para ello veamos primero algunas definiciones preliminares.

Definición

Dados dos elementos a *y* b *de un conjunto parcialmente ordenado, decimos que* 'b' **cubre** *a* 'a', *o que* 'a' **es cubierto por** 'b', *i. e.* $a \leq b$, *sii* $a < b$, *y para cualquier* $c \in P$ *tal que* $a \leq c \leq b$ *entonces* $c = a$ *o* $c = b$.

■

Para dibujar Diagramas de Hasse de conjuntos parcialmente ordenados finitos, cada elemento de dicho conjunto es representado por un pequeño círculo, y si $a \leq b$ dibujamos el círculo correspondiente a *"b"* por debajo del círculo correspondiente a *"a"*. En la siguiente figura se pueden ver algunos ejemplos.



En (Davey y Priestley, 1990) se puede encontrar más información sobre este tipo de diagramas.

Definición

Sea A un subconjunto de un conjunto parcialmente ordenado P. Un elemento $p \in P$ es una cota superior de A sii $\forall a \in A$, $a \leq p$. Un elemento $p \in P$ es el supremo de A (sup A), o la menor cota superior de A, sii p es una cota superior de a y para cualquier elemento $b \in P$ tal que b es también una cota superior de a, $p \leq b$.
Las cotas inferiores de A y el ínfimo, o mayor cota inferior de A (inf A), se definen de forma análoga.

■

Utilicemos ahora este conocimiento para enunciar una nueva definición de retículos.

Definición (Retículo: definición de orden)

Un conjunto parcialmente ordenado L es un retículo sii para cualesquiera $a,b \in L$, existen (en L) tanto el sup{a,b} como el inf{a,b}.

■

Nótese que tanto el ejemplo de conjunto parcialmente ordenado $<Pot(A),\subseteq>$ como $< T,\subseteq>$, donde $\subseteq$= *'ser subtipo de'*, son ejemplos de retículos de acuerdo con la definición anterior. En cambio, de los conjuntos parcialmente ordenados representados en los diagramas de la figura anterior, sólo *(a)*, *(b)* y *(d)* son retículos.

Las dos definiciones de retículos anteriores son equivalentes en lo que se explica a continuación. Como veremos, dado un retículo algebraico $<L,\vee,\wedge>$ existe una forma uniforme de construir otro $<L,\leq>$ ambos sobre el mismo conjunto *L* y viceversa. Es más, ambas construcciones son inversas, por lo que cualquier retículo algebraico puede ser representado como un retículo basado en orden y viceversa.

Teorema

Si $<L,\vee,\wedge>$ es un retículo definido de forma algebraica, se define en L la relación binaria $\leq$ como sigue:

$$a \leq b \text{ si y sólo si } a = a \wedge b, \text{ para todo } a, b \in L.$$

Entonces $<L, \leq>$ es un retículo de acuerdo con la definición basada en el orden.

■

En el caso de T, recordemos que $\wedge$ = *'ser subtipo de'*.

Teorema

Si $<L,\leq>$ es un retículo definido de forma algebraica, se definen en L dos operaciones binarias $\vee$ y $\wedge$ mediante $a \vee b = sup\{a,b\}$ y $a \wedge b = inf\{a,b\}$. Entonces $<L,\vee,\wedge>$ es un retículo de acuerdo a la definición algebraica.

■

En resumen, cualquier retículo algebraico es equivalente a un retículo basado en el orden. Por lo tanto, a partir de este momento, se puede asumir que cualquier *retículo algebraico* tiene un retículo basado en orden definido, y que cualquier *retículo basado en un orden teórico* tiene definidas la unión y la intersección. Se

tienen, por tanto, dos perspectivas de los retículos, ambas comprobadas en el conjunto de tipos del dominio T.

3.2.3 Jerarquía de tipos: completitud del conjunto

Hasta ahora se ha demostrado que el conjunto de tipos del dominio, T, es un retículo tanto desde un punto de vista algebraico como atendiendo al orden parcial definido sobre dicho conjunto. Hemos visto también que está formado por un conjunto de subretículos, que a su vez son retículos. Para finalizar la sección, veamos que es un retículo completo.

Supongamos que tenemos un retículo $<L,\vee,\wedge>$. Sabemos que cualquier par de elementos tiene un supremo (y un ínfimo), pero ¿se puede afirmar lo mismo de cualquier subconjunto L' de L?.

Obsérvese que todos los subconjuntos de L finitos y no vacíos, L', tienen supremo e ínfimo, puesto que si el conjunto está formado por n elementos del tipo $\{a_1, ..., a_n\}$, entonces la unión $a_1 \vee ... \vee a_n$ de los n elementos es el supremo de dicho conjunto. Este hecho puede ser demostrado por inducción. De igual forma, el ínfimo de este conjunto es $a_1 \wedge ... \wedge a_n$.

Esto sólo es cierto para subconjuntos finitos L', y en general no es cierto que conjuntos arbitrarios de retículos tengan siempre supremo e ínfimo.

Estos retículos especiales en los que cualquier subconjunto tiene un supremo y un ínfimo son llamados **retículos completos.** Los retículos finitos son ejemplos de retículos completos.

En el Apéndice C: "Jerarquía de tipos" se ilustran la mayor parte de los elementos y subconjuntos de la jerarquía de tipos; en dicho apéndice se puede observar que el conjunto global T es completo. Aunque el conjunto sigue aumentando en número a medida que se va ampliando el contenido de la base de conocimientos, los criterios seguidos para relacionar los conceptos son los mismos, por lo que T seguirá siendo un retículo completo cuyo cardinal será cada vez mayor.

3.3 Descripción del lenguaje formal Prógenes

En este apartado se formalizará el lenguaje formal Prógenes. Aunque el fondo es siempre el mismo, debemos diferenciar entre el lenguaje formal utilizado por el módulo de resolución y el utilizado por la interfaz en lenguaje natural. Ambos son iguales en contenido. Sin embargo, en el utilizado por la interfaz, los tipos aparecen de forma explícita en la expresión, mientras que en el del módulo de resolución están implícitos en la expresión. El motivo de la aparición de los tipos se verá en el capítulo 4, en el que se describirá el proceso composicional por el que se obtienen las expresiones semánticas.

Comencemos describiendo la sintaxis del lenguaje formal sin tipos explícitos y, a partir de esta, se generalizará a las utilizadas por la interfaz.

3.3.1 Sintaxis general sin tipos

Las expresiones del lenguaje formal se generan a partir de los conceptos existentes en la Base de Conocimiento. Estas expresiones son definidas a partir de reglas asociadas a la BC que obtienen, a partir de cualquier objeto, no sólo la expresión formal asociada sino también su tipo.

Las expresiones del lenguaje formal son expresiones LISP, i.e. un árbol implementado como una lista anidada, con notación prefija, cuyo primer valor es un símbolo. Los componentes de cualquier expresión son los elementos que forman dicha lista. Existe una forma natural de explorar las subexpresiones de una expresión dada, que concuerda con el bucle seguido por el intérprete Lisp: primeramente en profundidad y de izquierda a derecha. De acuerdo con esta definición se puede hablar de subexpresiones que siguen o preceden a una subexpresión dada, y también acerca de los componentes que lo siguen o preceden. De la misma forma también hablaremos de subexpresiones que contienen a una dada.

Una expresión formal Prógenes es aquella que satisface la siguiente sintaxis general:

```
(<nucleo> {<argumentos>}+)

<nucleo> := <objeto> ¦ <funcion> ¦ <asignacion>

<objeto> := El nombre de cualquier objeto válido de la BC.

<funcion> := El nombre de cualquier función general o de cuantificación de la BC.

<asignacion> := Función que asigna un tipo a una metavariable.
                Se representa por el símbolo terminal :.

<argumentos> := {<terminales>}* ¦ {(<nucleo> {<argumentos>}+)}*

<terminales> := <matematicos> ¦ <objeto> ¦ <metavariables>

<matematicos> := <numero> ¦ <variables>

<numero> := Los números naturales o reales.

<variables> := Cualquiera de las variables utilizadas en los
enunciados de los problemas, como pueden ser "x", "y", "z".

<metavariable> := Cualquier variable que comience por %.
```

El prefijo *meta*, en metavariable, es utilizado para no confundir este concepto con el objeto *variable* propio del dominio. Las metavariables se crean de forma dinámica, comenzando siempre por %, mientras que las variables son cualquiera de las variables utilizadas en los enunciados de los problemas, como pueden ser *"x"*, *"y"* o *"z"*.

Según el gráfico anterior, las ramas del árbol Lisp pueden ser *núcleos* o *argumentos* admisibles. Los núcleos admisibles sólo pueden aparecer al principio de las subexpresiones y los argumentos como el resto de las subexpresiones.

Los núcleos admisibles pueden ser nombres de objetos válidos, como por ejemplo *muelle*, *parábola*, *vector*, *punto*; nombres de funciones, como *directriz*, *hallar*, *probar*, *inter*, = = , + + ; funciones de cuantificación, como *el* y *para-todo*; o el signo de asignación de tipo *":"*.

Los argumentos admisibles pueden ser a su vez símbolos terminales admisibles, por ejemplo números, metavariables como *%x*, *%y23*; o nombres de tipos como *EQUATION*, *LINE*, *CURVE*. Como se ha visto en los ejemplos, los nombres de tipos

siempre empiezan por mayúsculas, para ser distinguidos de los nombres de objetos, con los que pueden coincidir.

Veamos como ejemplo la expresión formal que representa a uno de los enunciados de nuestro corpus, junto a las subexpresiones particulares que intervienen en la expresión total:

```
(3.1)  Escribir la ecuación de la recta paralela a 3x-5y+8=0
       y que pasa por el punto (-3,2).

(3.2)  (hallar
         (ecuacion
           (el (: %r recta)
               (y (paralela (== (++ (-- (** 3 x) (** 5 y)) 8) 0) %r)
                  (en (punto (-3 2) %r))))))

(3.3)  (ecuacion
         (el (: %r recta)
             (y (paralela (== (++ (-- (** 3 x) (** 5 y)) 8) 0) %r)
                (en (punto (-3 2) %r)))))

(3.4)  (el (: %r recta)
           (y (paralela (== (++ (-- (** 3 x) (** 5 y)) 8) 0) %r)
              (en (punto (-3 2) %r))))

(3.5)  (: %r recta)

(3.6)  (y (paralela (== (++ (-- (** 3 x) (** 5 y)) 8) 0) %r)
          (en (punto (-3 2) %r)))

(3.7)  (paralela (== (++ (-- (** 3 x) (** 5 y)) 8) 0) %r)

(3.8)  (== (++ (-- (** 3 x) (** 5 y)) 8) 0)

(3.9)  (en (punto (-3 2)) %r)

(3.10) (punto (-3 2))
```

Tanto la expresión global (3.2) como las subexpresiones (3.3), ..., (3.10) cumplen la sintaxis general anterior. Las expresiones (3.2), (3.6), (3.7), (3.8) y (3.9) tienen como núcleo de la expresión una función. El núcleo de (3.4) es una función de cuantificación, el determinante. En (3.5) podemos ver una asignación del tipo *LINE* a la metavariable *%r*; la subexpresión está encabezada por la función de asignación

:. Por último, las subexpresiones (3.3) y (3.10) representan la definición de dos objetos: una ecuación y un punto.

3.3.2 Sintaxis general con tipos explícitos

Como hemos visto, las expresiones formales del apartado anterior están constituidas por distintos tipos de objetos. Para que a partir de las subexpresiones más simples se formen otras más complejas es necesario aplicar las relaciones entre los objetos. Estas relaciones determinan cuándo una subexpresión puede formar parte de otra y cuándo no.

Por otra parte, en la sección anterior hemos visto que las relaciones entre los objetos forman un conjunto jerarquizado de tipos. Puesto que el objetivo principal de la interfaz es hallar la representación formal de un enunciado a partir de las expresiones formales de sus elementos más simples, es necesario conocer en todo momento las relaciones entre estos objetos, dirigiendo así la composicionalidad del proceso.

Es por ello que en las representaciones semánticas se deben incluir los tipos para dar cuenta de dichas relaciones. La sintaxis de las expresiones formales al incluir de forma explícita los tipos, es igual a la ya definida añadiendo los siguientes cambios:

```
(<tipo> (<nucleo> {<argumentos>}+))

<tipo> := Cualquier elemento del conjunto de tipos T.

<argumentos> := (<tipo> {<terminales>}*) |
                {(<tipo> (<nucleo> {<argumentos>}+))}*
```

Obsérvese que la única diferencia es que todas las subexpresiones han sido incluidas en otra cuyo primer elemento es el tipo de la expresión.

El enunciado del ejemplo anterior se convierte ahora en:

```
3.1  Escribir la ecuación de la recta paralela a 3x-5y+8=0 y
     que pasa por el punto (-3,2).

3.2  (EQUATION (hallar
       (EQUATION (ecuacion
         (LINE (el (LINE (: (VAR %r) (LINE recta)))
                   (BOOL (y (BOOL (paralela (LINE (...)) (VAR %r)))
                            (BOOL (en (POINT (punto (NUM (-3 2))))
                                      (LINE %r)))
                   ))))))))


3.3  (EQUATION (ecuacion
       (LINE (el (LINE (: (VAR %r) (LINE recta)))
                 (BOOL (y (BOOL (paralela (LINE (...)) (VAR %r)))
                          (BOOL (en (POINT (punto (NUM (-3 2))))
                                    (LINE %r)))
                      ))))))

3.4  (LINE (el (LINE (: (VAR %r) (LINE recta)))
               (BOOL (y (BOOL (paralela (LINE (...)) (VAR %r)))
                        (BOOL (en (POINT (punto (NUM -3 2))))
                                  (LINE %r)))
               ))))

3.5  (LINE (: (VAR %r) (LINE recta)))

3.6  (BOOL (y (BOOL (paralela (LINE (...)) (VAR %r)))
              (BOOL (en (POINT (punto (NUM (-3 2))))
                        (LINE %r)))
         ))

3.7  (BOOL (paralela (LINE (...)) (VAR %r)))

3.8  (LINE (= (...) (NUM 0)))

3.9  (BOOL (en (POINT (punto (NUM (-3 2)))) (LINE %r)))

3.10 (POINT (punto (NUM (-3 2))))
```

Este ejemplo se ha desarrollado a partir de un enunciado completo, i.e., con todas las variables instanciadas. Es por ello que la expresión (3.9) tiene como segundo argumento de *"en"* una expresión de tipo *LINE* que es más específico que el tipo *SUBVAR* que, como veremos en el capítulo 4, aparece en su definición inicial.

Si observamos las subexpresiones en orden inverso, comenzando por la (3.10), el proceso es claramente composicional, aunque este hecho será más evidente en el próximo capítulo.

En el apartado 3.1, *"Elementos de la Base de Conocimientos"* se presentaron las definiciones de algunas de las palabras que intervienen en este enunciado. Concretamente *punto*, *hallar* y *el*. El objeto *punto* tenía un número indeterminado de argumentos de tipo *REAL*, sin embargo en (3.10) el tipo de los argumentos de este objeto es *NUM*. Ambas posibilidades son válidas, puesto que en la jerarquía de tipos se hace explícita la relación entre números y reales; de hecho *NUM* es un subtipo de *REAL*, por tanto es más específico.

La función *hallar* tiene un argumento obligatorio de tipo *OBJETO* y un conjunto arbitrario de argumentos booleanos. En el caso del enunciado, *hallar* sólo tiene un argumento de tipo *EQUATION*. Nuevamente esto es posible teniendo en cuenta la relación entre dos clases de objetos, en este caso los de tipo *EQUATION* y los de tipo *OBJECT*.

El determinante realiza labores de cuantificación, por tanto efectúa una asignación del tipo *OBJECT* a una metavariable del sistema. En el caso del ejemplo la metavariable se denomina *%r* y a ésta se la asigna el tipo *LINE*. Nuevamente esto es posible porque los objetos de tipo *LINE* están relacionados con los de tipo *OBJECT*, son elementos del mismo subretículo. En la asignación de tipo no podría intervenir en ningún caso el tipo *BOOL* puesto que entre los elementos de este tipo y los de tipo *OBJECT* no existe relación directa alguna. Aunque tienen el mismo supremo e ínfimo pertenecen a subretículos distintos.

3.4 Las expresiones semánticas

Las expresiones Prógenes son muy parecidas a las utilizadas por otros formalismos basados en representaciones lógicas como el de Montague o la Teoría de Representación del Discurso expuestos en el Capítulo 1. Al igual que en formalismos basados en teorías veritativas, cada expresión semántica ha sido dotada de un tipo, que en nuestro caso se corresponde con la taxonomía del mundo que se está describiendo. A diferencia de otros formalismos, la información semántica no comparte estructuras con el resto de la información gramatical.

```
({<subexpresiones>}+)

<subexpresiones> := (<tipo> <descriptor>) |
                    (<tipo> ({<descriptor> (<tipo> <variable-semantica>)}+))

<tipo> := Cualquier elemento del conjunto de tipos T

<descriptor> := Nombres de los elementos descritos en la BC, ya sean objetos
                o funciones.

<variable-semantica> := Cualquier variable que comience por @.
```

Los valores de los atributos semánticos de nuestro léxico son denominados *expresiones semánticas*. La figura anterior ilustra su sintaxis, donde:

Tipo: Puede representar bien el tipo de un descriptor o los tipos de los argumentos. Estos tipos, como ya hemos visto, respetan un orden parcial, están estructurados en una jerarquía como grafos acíclicos orientados y reflejan estructuras matemáticas. En cualquier momento del análisis debemos saber no sólo el tipo de la expresión completa sino también los de los argumentos relacionados.

Descriptor: Representa la posible traducción de una palabra escrita en lenguaje natural al lenguaje formal de Prógenes, pero también denota los nombres de las funciones utilizadas en el módulo de resolución.

Variable-semántica: Generalmente, está formada por el símbolo @ seguido de cualquier letra. No deben ser confundidas con las metavariables del sistema, que son aquellas que intervienen en asignaciones de tipo y que comienzan por % seguida de cualquier letra. Las variables semánticas son instanciadas durante el

análisis y su valor puede ser otra expresión semántica tan compleja como sea necesario o una metavariable. Existe dos tipos especiales de variables semánticas. La primera está formada por los símbolos *@!*, seguida de cualquier letra, que representa los argumentos opcionales de una expresión. La segunda está formada por la cadena *@rest* y representa las listas con un número indefinido de argumentos.

La sintaxis descrita es general y puede representar:

- El valor de los atributos semánticos en el diccionario. Por ejemplo para las palabras *hallar* o *escribir* es:

```
((OBJECT (hallar (OBJECT @o) (BOOL @!b))))
```

- Análisis semánticos parciales, como en *"... pasa por el punto P(3,1) ..."*, representado por la siguiente expresión formal:

```
((BOOL (en (POINT (punto (NUM (3 1)))) (SUBVAR @s))))
```

- El significado de la oración completa. Así, la oración *"Escribir la ecuación de la recta que pasa por el punto* **P(-3,2)** *y que es paralela a* **3x-5y+8=0"** es representada mediante la siguiente expresión formal:

```
((EQUATION
   (hallar
     (EQUATION
       (ecuacion
         (LINE
           (el (LINE (: (VAR %x) (LINE recta)))
               (BOOL (y (BOOL (en (POINT (punto (NUM (-3 2))))
                                  (SUBVAR @s)))
                        (BOOL (paralelo
                                (LINE (== (++ (-- (** 3 x) (** 5 y))
                                              8) 0))
                                (LINE @q)))))))))))
```

3.5 Relación entre la BC y las Expresiones Semánticas

El valor del atributo semántico del léxico está formado por una expresión tipada del lenguaje formal. Las fuentes por las que se consigue el valor de este atributo depende de cada caso:

- Las palabras del diccionario que **tienen significado en el** dominio se obtienen de la base de conocimiento, como ocurre con *"el"* y *"punto"*.

  Sin embargo, desde el punto de vista del procesamiento del lenguaje natural, la base de conocimientos que contiene los conceptos matemáticos no es directamente asimilable a un diccionario semántico. Antes de poder ser utilizada es necesario establecer una equivalencia entre las palabras del léxico y los conceptos que significan. En algunos casos, esta equivalencia es de muchos a uno ya que, a pesar de lo restringido del entorno, se utilizan varias expresiones diferentes para significar la misma idea. Un ejemplo lo constituyen las palabras *"escribir"* o *"hallar"* que, en este dominio, tienen la misma definición en el lenguaje formal Prógenes:

```
"Escribir la ecuación ..." → (hallar ...)
"Hallar la ecuación ..."   → (hallar ...)
```

También se da el caso de equivalencias de uno a muchos, como sucede con la palabra *"ecuación"*, que se puede referir a la definición de recta pero también se puede obtener a partir de la de plano y de la de otros muchos objetos matemáticos, determinándose el sentido correcto durante el análisis. Obsérvese que las expresiones semánticas obtenidas (3.11-b) y (3.12-b) difieren únicamente en sus argumentos.

```
(3.11-a)  (obj recta ((ecuacion (modelo-ecuacion 'recta
                                                 (forma-ecuacion recta)
                                                 *dimamb*)
                                LINEAR-EQUATION))
            LINE
            (...))

(3.11-b)  (LINEAR-EQUATION (ecuacion (LINE @r)))
```

```
(3.12-a)  (obj plano ((ecuacion (modelo-ecuacion 'plano
                                                 (forma-ecuacion plano)
                                                 3)
                                LINEAR-EQUATION))
          PLANE
          (...))

(3.12-b)  (LINEAR-EQUATION (ecuacion (PLANE @p)))
```

- La BC no contiene la representación de todas las palabras que se necesitan en el lenguaje natural para expresar un enunciado, sólo de aquellas que tienen significado en el dominio. Si la palabra **no tiene significado en el dominio** el atributo semántico tendrá un valor por defecto *(OBJECT @t)*, donde *OBJECT* es el supremo de la jerarquía de tipos. Los ejemplos más claros de palabras que, en nuestro dominio, no tienen semántica explícita son las preposiciones, adverbios y conjunciones. Únicamente durante el análisis semántico se instancian con valores específicos.
- Existe un tipo de elementos cuya definición semántica no se encuentra en la Base de Conocimiento, aun teniendo significado en el dominio. Se trata de las **fórmulas**, cuya expresión aparece de forma dinámica en los enunciados. La semántica de esta clase de elementos varía dependiendo del contexto; así, la misma expresión puede tener distintos significados obteniendo los ejemplos más representativos de palabras polisémicas en el dominio. A las fórmulas se les asigna inicialmente una semántica por defecto *(FORMULA @f)*, siendo *FORMULA* un subtipo común a todos los tipos que pueden representar una fórmula, como son los tipos de las ecuaciones lineales, los puntos, los intervalos, etc. Durante el análisis, este subtipo es instanciado, determinando así el contexto y el tipo de tratamiento que se debe dar a la fórmula para obtener su expresión en lenguaje formal. En la siguiente figura se pueden ver las traducciones de la fórmula *"(3,4)"* en dos enunciados distintos.

```
Escribir la ecuación de la recta que pasa por (3,4) y ...

(3,4) => (POINT (punto (REAL 3) (REAL 4)))

... Demostrar que f es continua en (3,4).

(3,4) => (INTERVAL (intervalo (TAG :open)
                              (REAL 3)
                              (TAG :open)
                              (REAL 4)))
```

En el apartado anterior describimos el aspecto y la sintaxis de nuestro lenguaje formal. A continuación veremos cómo se construyen expresiones bien enunciadas teniendo en cuenta la información de la BC y las restricciones impuestas por el lenguaje formal.

Como hemos mencionado, nuestra BC es el nexo entre la interfaz en lenguaje natural y el módulo de resolución, puesto que contiene el conocimiento de los conceptos matemáticos involucrados y el metaconocimiento requerido para resolver los problemas. Como veremos, también contiene, de forma implícita, las restricciones necesarias para enunciar expresiones correctas. Otro tipo de información contenida en la BC son las relaciones entre objetos, representadas en forma declarativa o procedural. Veamos con un poco más de detalle cómo extraer las expresiones semánticas asociadas a algunos tipos de palabras según su categoría sintáctica. Queremos hacer especial énfasis en el hecho de que la extracción que se va a describir se realiza de forma automática para la mayoría de los nombres, adjetivos y verbos propios de cada dominio. Las expresiones formales de un dominio se obtienen añadiendo, a las obtenidas de la BC, las expresiones semánticas de los determinantes, preposiciones, adverbios, etc.

Nombres y adjetivos: Estas palabras tienen significado en el dominio y se corresponden con los objetos en la BC y con algunas metafunciones. Los objetos están constituidos por entidades complejas y estructuradas. Todos tienen un tipo característico y un número arbitrario de atributos que describen los elementos del concepto. Por ejemplo, la definición en nuestra BC del nombre *muelle* es:

```
(obj muelle ((rigidez NUM)
             (posicion-agarre POINT)
             (tension NUM))
     SPRING
     ())
```

Como ya se ha dicho, las tres primeras líneas definen no sólo el tipo de los argumentos, sino también la definición implícita de otros conceptos como la *rigidez*, la *posición de agarre* y la *tensión* de un muelle. La cuarta línea denota el tipo asignado al concepto y, a partir de la quinta, información propia del módulo de resolución en aquellos casos en los que sea necesario. La palabra siguiente a *obj*, en este caso *muelle*, representa la traducción de la palabra enunciada en lenguaje natural al lenguaje formal. Aunque en este caso coincide, no siempre es así.

La extracción de la definición semántica de *muelle* es general, automática y respeta la sintaxis enunciada en la sección anterior:

```
((SPRING (muelle (NUM @n) (POINT @p)
                 (NUM @n))))
```

Del *objeto* anterior se extraen también las definiciones de la rigidez, tensión y posición de agarre de un muelle:

```
((NUM (rigidez (SPRING @s))))

((POINT (posicion-agarre (SPRING @s))))

((NUM (tension (SPRING @s))))
```

La definición del objeto compuesto *parábola* presenta un caso diferente:

```
(obj parabola ((ecuacion EQUATION)
               &optional (rotacion MATRIX)
                         (translacion VECTOR)
                         (distancia-focal REAL)
               ))
    PARABOLA
    (parabolic ecuacion))
```

En esta definición existe un único argumento obligatorio y tres opcionales. Su expresión semántica y las de las definiciones implícitas existentes son las siguientes:

```
((PARABOLA (parabola (EQUATION @e)
                     (MATRIX @!m)
                     (VECTOR @!v)
                     (REAL @!r))))

((EQUATION (ecuacion (PARABOLA @p))))

((MATRIX (rotacion (PARABOLA @p))))

((VECTOR (translacion (PARABOLA @p))))

((REAL (distancia-focal (PARABOLA @p))))
```

Veamos ahora otro ejemplo, el objeto *vector*, cuya definición en la BC es:

```
(obj vector (&rest (coordenadas REAL))
     VECTOR
     (= *ambdim* (length (cdr coordenadas)))
)
```

En este tipo de definiciones, los argumentos *&rest* pueden ser obligatorios, si aparecen como único argumento en la definición, u opcionales, si aparecen después de otro argumento obligatorio aunque no esten precedidos de *&optional.* La definición del objeto *vector* es un ejemplo de argumentos *&rest* obligatorios, siendo representada su expresión semántica como se ilustra en la siguiente figura:

```
((VECTOR (vector (REAL @rest))))

((REAL (coordenadas (VECTOR @v))))
```

Aunque la mayoría de los nombres y adjetivos se corresponden con los objetos de la BC, existen también algunas correspondencias con las metafunciones, como ocurre en el siguiente ejemplo:

```
(fun directriz ((p PARABOLA))
     LINE
     ()
     (let ((m (rotacion-parabola p)))
          (make-line
            (matriz** m (p+v (punto (/ (dist-focal-parabola p) -2) 0)
                             (translacion-parabola p)))
            (column 2 m))))
```

La expresión semántica que se obtiene es la siguiente:

```
((LINE (directriz (PARABOLA @p))))
```

Aunque la sintaxis de las *metafunciones* es similar a la de los *objetos*, en su representación formal no existen definiciones implícitas. Es decir, si los argumentos de la función varían, entonces se crea una nueva definición en la BC, como ocurre en el caso de *intersección*, cuyas definiciones en la BC son las siguientes:

```
(fun inter ((line1 LINE) (line2 LINE))
     POINT
     (not (paralelo line1 line2))
     (let* ((...))))

(fun inter ((c1 CURVE) (c2 CURVE))
     (type-or POINT (type-list POINT))
     ()
     ())

(fun inter ((set1 SET) (set2 SET))
     SET
     ()
     (declare (special set1 set2))
     (mcase set1 ...)
)
```

Estas definiciones dan lugar a la expresión semántica de *"intersección"*:

```
((POINT (inter (LINE @l1) (LINE @l2)))
 (POINT (inter (CURVE @c1) (CURVE @c2)))
 (SET (inter (SET @s1) (SET @s2))))
```

Los **verbos** son también palabras con significado que normalmente desempeñan acciones especificadas en las meta**fun**ciones de nuestra BC. Las metafunciones son aplicaciones del conjunto de objetos de la BC en sí mismo. La siguiente es la definición del verbo **hallar** en la BC:

```
(fun hallar ((obj OBJECT) &rest (satisfaciendo BOOL))
     OBJECT
     ()
     (cond ((and (listp obj) (eq 'a (car obj))) (a (pfind (cadr obj))))
           (t (let ((variables (vars obj '%)))
                (...))))
```

Su representación en lenguaje formal sería el que se presenta a continuación. Obsérvese que en este caso el argumento *&rest* se ha tratado como opcional, puesto que aparece a continuación de uno obligatorio.

```
((OBJECT (hallar (OBJECT @t) (BOOL @!b))))
```

En la definición del verbo *probar* aparecen dos argumentos opcionales:

```
(fun probar ((objetivo BOOL) &optional (hipotesis BOOL) (usando BOOL))
     BOOL
     ()
     (if hipotesis (condicion hipotesis))
     (if usando (add *hints* usando))
     (cond ((atom objetivo) (...)))
)
```

y su expresión semántica es la siguiente:

```
((BOOL (probar (BOOL @b)
               (BOOL @!b)
               (BOOL @!b))))
```

Los **determinantes** tienen una semántica muy específica que nos permite resolver adecuadamente los complementos preposicionales así como resolver referentes dentro y fuera de la oración. Dichos determinantes son representados en la BC como **bind**ing **func**tions. Este grupo también engloba todas aquellas funciones que realicen asignación de tipo, propiedad utilizada para la resolución de referentes. La definición en la BC de la *binding function "el"* es:

```
(bndfun el ((obj OBJECT) (bool BOOL))
       OBJECT
       ()
       (if bool (condition (eval bool))
           (eval (move-= = = obj))))
```

y la expresión semántica representada por este cuantificador:

```
(OBJECT (el (OBJECT (: (VAR %X) (OBJECT @t)))
            (BOOL @b)))
```

Obsérvese que el código escrito en negrita no puede ser deducido a partir de la BC utilizando el mecanismo natural utilizado con los objetos y metafunciones, pero es necesario para que se pueda ejecutar la acción de los determinantes, que se describirá en el capítulo 4.

El objetivo de la subexpresión en negrita es asignar a la metavariable *%X* el valor que tenga la variable semántica *@t* una vez instanciada. El tipo *OBJECT* será también instanciado por otro más específico en toda la expresión.

Las **preposiciones** y los **adverbios** sólo desempeñan un papel sintáctico durante el análisis y su semántica depende del contexto. En nuestro lenguaje formal este caso es representado como *(OBJECT @o)* donde *OBJECT* es el supremo de nuestra jerarquía, a partir de la cual heredan el resto de los tipos.

Otro elemento que aparece numerosas veces en los textos del dominio son las **fórmulas**. Su proceso semántico pasa por distintas etapas. En la fase inicial adquiere la semántica por defecto *(FORMULA formula)*, donde *FORMULA* es un tipo vacío que hereda de todos los conceptos que aparecen en el texto como fórmulas TeX: ecuaciones, líneas, pares de números, intervalos, conjuntos... En las fases siguientes, un módulo traductor instancia el valor de este tipo y convierte la fórmula inicial en una expresión simbólica en notación prefija. Por ejemplo, después de todo el proceso, el valor semántico de *3x-5y+8=0* es

```
(LINEAR-EQUATION
  (== (++ (-- (** 3 x) (** 5 y)) 8) 0))
```

Los operadores + + , --, **, y = = se comportan como los operadores aritméticos normales cuando sus argumentos son números, pero además permiten expresiones simbólicas no instanciadas. La descripción del módulo traductor puede verse en (Gonzalo, en preparación).

Capítulo 4

4.0 Semántica composicional en Prógenes

En el capítulo 1 se enumeraron las propiedades que serían deseables en un formalismo semántico. Algunas de estas propiedades ya han sido tratadas en capítulos anteriores, concretamente el uso de objetos semánticos en el capítulo 3 y la realimentación para el parser en el capítulo 2.

En este capítulo se describirá la composicionalidad del proceso, la implementación de reglas, el tratamiento de la ambigüedad semántica y el tratamiento de algunas complejidades semánticas, como es la anáfora.

4.1 Mecanismos de combinación semántica

Hasta ahora, se ha descrito cómo se relaciona la representación semántica con la base de conocimientos, el lenguaje formal y el léxico. A continuación mostraremos cuáles son los mecanismos composicionales que permiten combinar expresiones semánticas. Para ello se utiliza el concepto de regla orientada a objetos ya mencionado en el capítulo 2, y relacionado estrechamente con nuestro léxico jerarquizado.

4.1.1 Reglas semánticas en Prógenes.

Como ya se ha visto, la clase superior de la jerarquía de objetos es denominada *signo*. Por herencia, todas las entradas del léxico y todos los análisis parciales pertenecen a esta clase. Recordemos que sus atributos principales son: spelling, cat (papel sintáctico) y semántica (sem). El valor semántico del objeto respeta la sintaxis descrita en el capítulo anterior.

Las reglas sintácticas, semánticas y globales del sistema se aplican sobre un par contiguo y ordenado de objetos para dar otro objeto, del que son sus hijos.

A continuación describiremos algunas de las reglas utilizadas en la combinación semántica. Recordemos que algunas de las utilizadas en el análisis sintáctico fueron descritas en 2.5.

4.1.1.1 Regla por defecto

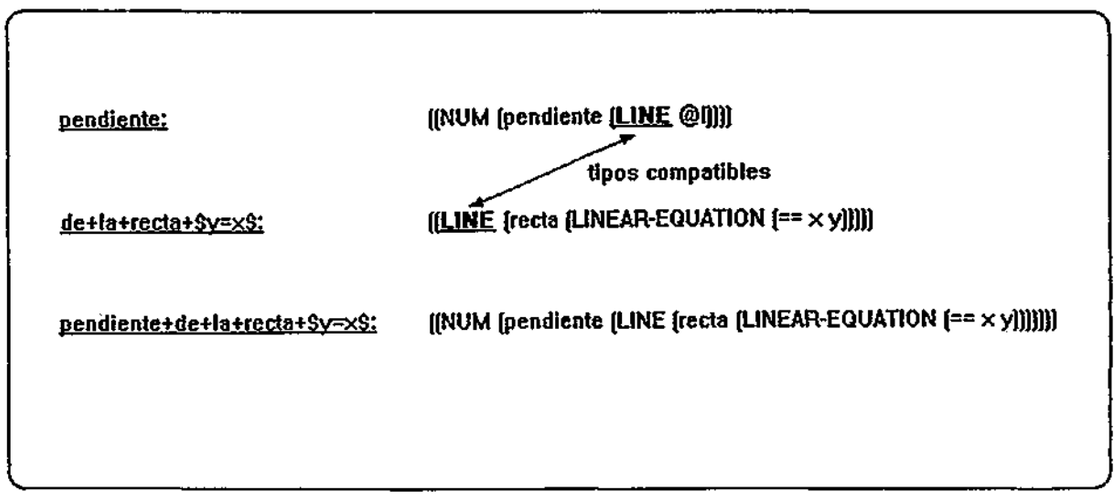
Dentro de las reglas semánticas, una de ellas se puede aplicar siempre que ambos objetos pertenezcan a la clase *signo*. Como siempre se da este extremo, la regla es siempre aplicable. Sólo cuando existan reglas más específicas no será disparada. En nuestro entorno, juega el papel que la unificación tiene en las gramáticas categoriales de unificación (UCG) (Zeevat et al., 1987). Pero ese es el único mecanismo del que dispone UCG, lo que dificulta en ese formalismo el tratamiento de fenómenos complejos o específicos. Nuestras reglas por defecto tienen también cierto paralelismo con cierto tipo de reglas en HPSG, ya que ambas se definen sobre estructuras tipadas de pares atributo-valor. Una diferencia menor es que en

HPSG el orden se especifica de forma distinta (mediante reglas de "Linear Precedence"). Una diferencia mayor es que las reglas de HPSG, expresadas en forma de restricciones, deben cumplirse siempre; no se contempla en la teoría la posibilidad de que sean incumplidas por el hecho de que otras reglas más específicas tomen prioridad sobre éstas.

La regla para dos objetos de tipo *signo* es la siguiente:

> *La semántica global de dos objetos s1 y s2 de tipo signo se obtiene reemplazando por el valor semántico de s2 la variable semántica más cercana cuyo tipo sea compatible con el de s2.*

Por ejemplo, la combinación de *"pendiente"* con *"de+la+recta+$y=x$"* da lugar a la expresión semántica de *"pendiente+de+la+recta+$y=x$"* como se observa a continuación:



4.1.1.2 Determinantes

La acción del determinante no puede ser descrita mediante el algoritmo general. Un determinante puede tener dos funciones distintas sobre un nombre, según su alcance. En una de ellas coinciden denotación e intensión, como en 4.1a, y en la otra son distintas, como en 4.1b:

```
(4.1a) La+recta+$y=x$ :

       ((LINE (recta (LINEAR-EQUATION (== y x)))))

(4.1b) La+recta+que+pasa+por+p(3,2)+y+q(4,1):

       ((LINE (el (LINE (: (LINE %X) (LINE LINE)))
                  (BOOL (y (BOOL (en (POINT (3 2))
                                     (LINE %X)))
                           (BOOL (en (POINT (4 1))
                                     (LINE %X))))))))
```

Por ello existe un método binario sobre objetos de tipo *determinante* y *nombre* que da dos semánticas posibles para el análisis de, vg., *"la"* y *"recta"*:

```
la+recta:

((LINE (recta (LINEAR-EQUATION @1)))
 (LINE (el (LINE (: (LINE %X) (LINE recta)))
           (BOOL @b))))
```

El método semántico general también podría dar lugar a la siguiente expresión:

```
la+recta:

((LINE (el (LINE (: (LINE %X) (LINE (recta (LINEAR-EQUATION @1)))))
           (BOOL @b))))
```

que no es considerada, aunque respeta la sintaxis general descrita anteriormente. El motivo es que la función *":"* asigna a una variable, en el presente ejemplo *"(LINE %X)"*, el nombre de un objeto, *(LINE recta)*, que es una expresión sin argumentos. Se permiten por tanto expresiones como *(LINE (: (LINE %X) (LINE recta)))*, que denota una variable de tipo recta. La expresión *(LINE (recta (LINEAR-EQUATION @l)))* representa un objeto con sus argumentos, por lo que no puesde ser utilizada por *":"*

4.2 Tratamiento de la anáfora en Prógenes

La anáfora es un tipo de deixis que desempeñan ciertas palabras para asumir el significado de una parte del discurso ya emitida. Para identificar su significado completo se debe efectuar la completa resolución de todos los referentes existentes en la oración.

En algunas aplicaciones el problema de la anáfora se evita a menudo asumiendo que la indeterminación que se introduce no es crítica de cara al resultado final.

En Prógenes sin embargo, no podemos asumir esta indeterminación ya que, para la correcta formalización del enunciado, han de considerarse todos los datos que éste especifique. También se debe tener en cuenta que los enunciados pueden constar de una o varias líneas, e incluir subapartados, notas u otros tipos de descripciones, como se observa en los textos de los Problemas 1, 2, 3 y 4 enunciados al principio de este trabajo. Por este motivo es necesario resolver no sólo las referencias dentro del alcance de cada oración aislada, como puede ser el pronombre del Problema 3, subordinadas, como la del Problema 1, etc., sino también las referencias que se den en un contexto formado por varias oraciones, como los problemas 2 y 4.

```
Problema 1:

    "Escribir la ecuación de la recta paralela a 3x-5y+8=0 y
     que pasa por el punto (-3,2)."

Problema 2:

    "Decimos que una función f:R → R es par si
     f(x) = f(-x) para todo x ∈R. Demostrar que
     si una función f es par f'(0) =0."

Problema 3:

    "Hallar el punto donde se intersectan las rectas 2x+7y+31=0 y
     x-2y+7=0, y determinar su ángulo de intersección."

Problema 4

    "Sean a y b números no negativos. Demostrar que si a² =< b²
     entonces a =< b.
     INDICACION: b²-a²=(b+a)(b-a)."
```

Como ya vimos, para la correcta interpretación del Problema 3 se debe evidenciar el hecho de que *"su ángulo de intersección"* se refiere a *"las rectas"* y, en el Problema 4, la interpretación del texto debe tener en cuenta que los términos "a" y "b" hacen referencia a las mismas entidades en las distintas porciones del texto del enunciado.

También se debe determinar adecuadamente el nivel de modificación al que se refieren los complementos preposicionales, puesto que son frecuentes situaciones como la del siguiente ejemplo ya mencionado en anteriores apartados:

```
Hallar la intersección de la recta R con el eje X.
```

Para la asociación del primero de los dos grupos preposicionales que aparecen no existe ningún problema, ya que sólo es posible que dependa de *"intersección"* (la dependencia de *"Hallar"* se puede descartar teniendo en cuenta las categorías sintácticas de este verbo y del complemento preposicional).

Igualmente, *"con el eje X"* tiene que asociarse a *"intersección"* y no a *"recta"*, lo que se asegura teniendo en cuenta el tipo de los argumentos. Utilizando las traducciones de estas proposiciones en las expresiones semánticas del lenguaje formal Prógenes, estas afirmaciones acerca de la dependencia de *"con el eje X"*, *"intersección"* y *"recta"* son evidentes.

```
intersección:

((SET (inter (SET @s1)
             (SET @s2))))

la recta:

((LINE (recta (LINEAR-EQUATION @l))))

con el eje X:

((LINE (eje (FORMULA X))))

intersección de la recta con el eje X:

((SET (inter (LINE (recta (LINEAR-EQUATION @l)))
             (LINE (eje (FORMULA X))))))
```

Para que se diera lugar al análisis de *"la recta R con el eje X"*, el tipo de *"con el eje X"*, *LINE*, debería ser compatible con *LINEAR-EQUATION*, circunstancia que no ocurre en nuestra jerarquía de tipos.

El algoritmo utilizado para la resolución de la anáfora en Prógenes se basa en criterios de proximidad, como ocurre en otros algoritmos como el descrito en (Lappin et al., 1990). A diferencia del algoritmo descrito en la referencia anterior, en el utilizado en Prógenes se necesita, como veremos a continuación, conocimiento semántico. Otra diferencia es que en el algoritmo anterior la metáfora se resuelve dando un papel primordial a los pronombres, en cambio en Prógenes no se utilizan sólo los pronombres, sino también los determinantes, en los que nos centraremos a continuación.

Los elementos del algoritmo son los siguientes:

- Un filtro sintáctico dentro de la oración para averiguar el entorno anáforico del determinante en un sintagma nominal.
- Un procedimiento sintáctico para identificar determinantes pleonásticos.
- Un algoritmo de acotación de la anáfora para identificar el posible antecedente de una anáfora léxica dentro de la misma oración.

- Un procedimiento para asignar valores semánticos a aquellos argumentos de la expresión que son referenciados de forma implícita y que, por tanto, componen la anáfora.

Con todo ello pueden aparecer referencias entre oraciones, al mismo tiempo que referencias dentro de la oración. En ese caso se utilizarán en primer lugar las *intraoracionales* y en segundo lugar las *interoracionales.*

4.2.1 Resolución de referentes en Prógenes: el determinante

Para poder resolver adecuadamente dicha referencia se define un nivel conceptual de denotación llamado *entorno.* Este entorno es definido por la semántica de ciertas palabras del dominio. Las más comunes son los determinantes, que pueden estar anidados.

En el proceso de resolución de referentes se obtiene una expresión semántica completa, en la que todas las variables semánticas están instanciadas. Para la explicación del proceso utilizaremos la expresión formal del Problema 1:

```
((EQUATION
   (hallar
     (EQUATION
       (ecuacion
         (LINE
           (el (LINE (: (VAR %x) (LINE recta)))
               (BOOL (y (BOOL (en (POINT (punto (NUM (-3 2))))
                                  (SUBVAR @s)))
                        (BOOL (paralelo
                                (LINE (== (++ (-- (** 3 x)
                                                  (** 5 y))
                                              8) 0))
                                (LINE @q)))))))))
     (BOOL @!b))))
```

En ella se deben eliminar los argumentos opcionales, que son aquellos cuya variable semántica está formada por *"@!"* seguida de cualquier variable semántica. La expresión semántica del ejemplo se convertiría en la siguiente:

```
((EQUATION
   (hallar
     (EQUATION
       (ecuacion
         (LINE
           (el (LINE (: (VAR %x) (LINE recta)))
               (BOOL (y (BOOL (en (POINT (punto (NUM (-3 2))))
                                  (SUBVAR @s)))
                        (BOOL (paralelo
                                (LINE (== (++ (-- (** 3 x)
                                                  (** 5 y))
                                              8) 0))
                                (LINE @q)))))))))))
```

El filtro sintáctico en la oración para averiguar el entorno anafórico del determinante no sólo nos devolvería la subexpresión que se ilustra a continuación, sino la variable asignada y su tipo, en este caso *"(LINE %X)"*.

```
(el (LINE (: (VAR %x) (LINE recta)))
    (BOOL (y (BOOL (en (POINT (punto (NUM (-3 2))))
                       (SUBVAR @s)))
             (BOOL (paralelo
                     (LINE (== (++ (-- (** 3 x)
                                       (** 5 y))
                                   8) 0))
                     (LINE @q))))))
```

Las referencias implícitas vienen representadas por los pares *(SUBVAR @s)* y *(LINE @l)*. Si no hubiera ninguna variable libre, la expresión semántica sería correcta, puesto que se habría formado, de forma composicional y mediante las reglas de construcción, a partir de expresiones bien formadas.

Para resolver estas referencias, basta con comprobar la compatibilidad de los tipos asignados a las variables y los tipos de las variables semánticas sin instanciar, en este caso dada por la compatibilidad de *LINE* y *SUBVAR*, y sustituir el tipo de la variable sin instanciar por el más específico de los dos, y la variable *"@"* por la referenciada en la variable *"%"*, quedando finalmente la expresión:

```
((EQUATION
   (hallar
     (EQUATION
       (ecuacion
         (LINE
           (el (LINE (: (VAR %x) (LINE recta)))
               (BOOL (y (BOOL (en (POINT (punto (NUM (-3 2))))
                                  (LINE %x)))
                        (BOOL (paralelo
                                (LINE (== (++ (-- (** 3 x)
                                                  (** 5 y))
                                              8) 0))
                                (LINE %x)))))))))))
```

4.3 Ejemplo de funcionamiento: Problema 1

Para ilustrar de forma detallada, tanto la forma de los elementos estáticos del diccionario como de los creados de forma dinámica, a continuación se muestra la representación de la mayor parte de los objetos léxicos involucrados en el análisis del Problema 1. En esta exposición sólo se van a ilustrar aquellos objetos que representan análisis parciales correctos o los objetos que intervienen en estos análisis, sin exponer las posibles ambigüedades sintácticas. Así las fórmulas aparecerán con categoría sintáctica *NP* o *NP* \ *NP* según corresponda.

4.3.1 Léxico estático

Obsérvese que la categoría sintáctica del primer objeto léxico *"escribir"*, es *"S/NP"*. Esta definición es motivada por el hecho de que el objetivo del análisis sintáctico es analizar toda la oración, lo que es equivalente, sin tener en cuenta el discurso, a obtener una categoría *"S"*. A este hecho se le une el que normalmente se "escribe un objeto", es decir, el verbo escribir espera encontrar a continuación un sintagma nominal.

```
escribir
[ spelling: escribir
  cat:      (S / NP)
  sem:      ((OBJECT (hallar (OBJECT @p)
                             (BOOL @!b))))  ]
```

```
una
[ spelling: una
  cat:      (NP / NP)
  sem:      ((OBJECT (el (OBJECT (<> (VAR %x) (OBJECT @o)))
                         (BOOL @b)))
             (OBJECT @o))  ]
```

```
ecuación
[ spelling: ecuación
  cat:      (NP)
  sem:      ((EQUATION (ecuacion (LINE @p))))  ]
```

```
para
[ spelling: para
  cat:      ((NP \ NP) / NP)
  sem:      ((OBJECT @o))  ]
```

```
la
[ spelling: la
  cat:      (NP / NP)
  sem:      ((OBJECT (el (OBJECT (<> (VAR %x) (OBJECT @o)))
                         (BOOL @b)))
              (OBJECT @o))  ]
```

```
recta
[ spelling: recta
  cat:      (NP)
  sem:      ((LINE recta)
              (LINE (recta (LINEAR-EQUATION @e))))  ]
```

```
paralela
[ spelling: paralela
  cat:      ((NP \ NP) / (NP \ NP))
  sem:      ((BOOL (paralelo (LINE @p) (LINE @q))))
]
```

```
a
[ spelling: a
  cat:      ((NP \ NP) / NP)
  sem:      ((OBJECT @o))
]
```

```
y
[ spelling: y
  cat:      ((X / X) \ X)
  sem:      ((BOOL (y (BOOL @p2) (BOOL @p1)))
             (ACTION (pvalues (ACTION @a2) (ACTION @a1))))
]
```

```
que
[ spelling: que
  cat:      ((NP \ NP) / (S \ NP))
  sem:      ((OBJECT @o))
]
```

pasa

```
spelling: pasa
cat:      ((S \ NP) / (NP \ NP))
sem:      ((BOOL (en (POINT @p) (SUBVAR @s))))
```

por

```
spelling: por
cat:      ((NP \ NP) / NP)
sem:      ((OBJECT @o))
```

el

```
spelling: el
cat:      (NP / NP)
sem:      ((OBJECT (el (OBJECT (<> (VAR %x) (OBJECT @o)))
           (OBJECT @o))
```

punto

```
spelling: punto
cat:      (NP)
sem:      ((POINT (punto (NUM @n))))
```

4.3.2 Léxico dinámico

El léxico dinámico del Problema 1 está formado por dos elementos: las fórmulas *"$3x-5y+8=0$"* y *"$P(-3,2)$"*. Ambas expresiones podrían ser, en principio, un sintagma nominal, NP, o un complemento preposicional, NP\NP. En ambos casos su semántica sería la misma, por lo que únicamente se ilustra la categoría sintáctica que conduce al análisis correcto.

```
$3x-5y+8=0$
[ spelling: $3x-5y+8=0$
  cat:      (NP)
  sem:      ((FORMULA (== (++ (-- (** 3 x) (** 5 y)) 8) 0))) ]
```

```
$P(-3,2)$
[ spelling: $P(-3,2)$
  cat:      (NP \ NP)
  sem:      ((FORMULA (-3 2))) ]
```

4.3.3 Análisis parciales

Los análisis parciales se obtienen aplicando *fa* y *ba*, junto a la regla semántica por defecto. Como ya hemos dicho se están obviando aquellos subárboles que, aunque en principio pudieran ser factibles, no conducen al análisis final.

```
una ecuación
[ spelling: una+ecuacion
  cat:      (NP)
  sem:      ((EQUATION (ecuacion 'LINE @p))) ]
```

```
la recta
[ spelling: la+recta
  cat:      (NP)
  sem:      ((LINE (el (LINE (<> (VAR %x) (LINE recta)))
                       (BOOL @b)))
             (LINE recta)
             (LINE (recta (LINEAR-EQUATION @e))))                    ]
```

```
punto $P(-3,2)$
[ spelling: punto+$P(-3,2)$
  cat:      (NP)
  sem:      ((POINT (punto (NUM (-3 2)))))                           ]
```

```
el punto $P(-3,2)$
[ spelling: el + punto + $P(-3,2)$
  cat:      (NP)
  sem:      ((POINT (punto (NUM (-3 2)))))                           ]
```

```
para la recta
[ spelling: para + la + recta
  cat:      (NP \ NP)
  sem:      ((LINE (el (LINE (<> (VAR %x) (LINE recta)))
                       (BOOL @b)))
             (LINE recta)
             (LINE (recta (LINEAR-EQUATION @e))))                    ]
```

```
por el punto $P(-3,2)$
[ spelling: por + el + punto + $P(-3,2)$
  cat:      (NP \ NP)
  sem:      ((POINT (punto (NUM (-3 2)))))]
```

```
una ecuación para la recta
[ spelling: una+ecuacion+para+la+recta
  cat:      (NP)
  sem:      ((EQUATION (ecuacion (LINE (LINE (el (LINE
                                                  (: (VAR %x)
                                                     (LINE recta)))
                                                 (BOOL @b))))))
              (EQUATION (ecuacion (LINE recta)))
              (EQUATION (ecuacion (LINE (recta (LINEAR-EQUATION @e))))))]
```

```
pasa por el punto $P(-3,2)$
[ spelling: pasa+por+el+punto+$P(-3,2)$
  cat:      (S \ NP)
  sem:      ((BOOL (en (POINT (punto (NUM (-3 2))))
                       (SUBVAR @s))))]
```

```
que pasa por el punto $P(-3,2)$
[ spelling: que+pasa+por+el+punto+$P(-3,2)$
  cat:      (NP \ NP)
  sem:      ((BOOL (en (POINT (punto (NUM (-3 2))))
                       (SUBVAR @s))))]
```

```
a $3x-5y+8=9$
[ spelling: a + $3x-5y+8=0$
  cat:      (NP \ NP)
  sem:      ((FORMULA (== (++ (-- (** 3 x) (** 5 y)) 8) 0))) ]
```

```
paralela a $3x-5y+8=9$
[ spelling: paralela+a+$3x-5y+8=0$
  cat:      (NP \ NP)
  sem:  ((BOOL (paralelo (LINE (== (++ (-- (** 3 x) (** 5 y)) 8)
                                   0))
                         (LINE @q)))) ]
```

```
y que pasa por el punto $P(-3,2)$
[ spelling: y+que+pasa+por+el+punto+$P(-3,2)$
  cat:      ((NP \ NP) \ (NP \ NP))
  sem:      ((BOOL (y (BOOL (en (POINT (punto (NUM (-3 2))))
                                (SUBVAR @s)))
                      (BOOL @p1)))) ]
```

```
paralela a $3x-5y+8=9$ y que pasa por el punto $P(-3,2)$
[ spelling: paralela+a+$3x-5y+8=0$+y+que+pasa+por+el+punto+$P(-3,2)$
  cat:      (NP \ NP)
  sem:      ((BOOL (y (BOOL (en (POINT (punto (NUM (-3 2))))
                                (SUBVAR @s)))
                      (BOOL (paralelo
                              (LINE (== (++ (-- (** 3 x) (** 5 y)) 8)
                                        0))
                              (LINE @q)))))) ]
```

una ecuación para la recta paralela a $3x-5y+8=9$ y que pasa por el punto $P(-3,2)$

```
spelling: una+ecuacion+para+la+recta+
          paralela+a+$3x-5y+8=0$+y+que+pasa+por+el+punto+$P(-3,2)$
cat:      (NP)
sem:      ((EQUATION
            (ecuacion
              (LINE
                (el (LINE (: (VAR %x) (LINE recta)))
                    (BOOL (y (BOOL (en (POINT (punto (NUM (-3 2))))
                                       (SUBVAR @s)))
                             (BOOL (paralelo
                                     (LINE (== (++ (-- (** 3 x)
                                                       (** 5 y))
                                                   8) 0))
                                     (LINE @q)))))))))))
```

escribir una ecuación para la recta paralela a $3x-5y+8=9$ y que pasa por el punto $P(-3,2)$

```
spelling: escribir+una+ecuacion+para+la+recta+
          paralela+a+$3x-5y+8=0$+y+que+pasa+por+el+punto+$P(-3,2)$
cat:      (S)
sem:      ((EQUATION
            (hallar
              (EQUATION
                (ecuacion
                  (LINE
                    (el (LINE (: (VAR %x) (LINE recta)))
                        (BOOL (y (BOOL (en (POINT (punto (NUM (-3 2))))
                                           (SUBVAR @s)))
                                 (BOOL (paralelo
                                         (LINE (== (++ (-- (** 3 x)
                                                           (** 5 y))
                                                       8) 0))
                                         (LINE @q)))))))))))
            (BOOL @!b))))
```

4.3.4 Otras reglas en Prógenes

Con estos mecanismos básicos, heredados de la aplicación funcional, queda diseñado el corazón del sistema y se obtienen análisis correctos para un buen número de enunciados.

Una de las líneas abiertas en Prógenes (Gonzalo, en preparación) tiene como objetivo el aumentar la cobertura lingüística del analizador para incluir referencias intersentenciales, fenómenos de no-constituencia (como la coordinación de no constituyentes y el vaciado), etc., mediante la incorporación de reglas que resuelven coordinación de constituyentes, otras referencias pronominales, etc.

En el trabajo citado se describen en profundidad el tratamiento de las fórmulas y de los pronombres posesivos.

Conclusiones

Como ya comentamos en la introducción, el objetivo de la tesis ha sido diseñar e implementar un formalismo semántico que represente el texto inicial, utilizando reglas no ad hoc, resolviendo ambigüedades léxicas y efectuando un tratamiento de la complejidad semántica.

Una vez expuesto en detalle el trabajo desarrollado, podemos extraer las siguientes conclusiones:

1. Se ha demostrado que una base de conocimientos adecuada puede simplificar extraordinariamente la tarea de construir una interfaz de lenguaje natural con el sistema que hace uso de esa base de conocimientos. Una base de conocimientos adecuada quiere decir, en este contexto, aquella que incluye una taxonomía jerarquizada de los objetos del dominio, y sobre la que se ha definido un lenguaje formal.

2. Con el uso de esta base de conocimientos, los problemas de equivalencia semántica quedan reducidos a efectuar una adecuada representación de los objetos y relaciones del dominio en dicha BC. Así el explicar el hecho de que las oraciones "El punto (2,1,1) pertenece al plano 2x-5y+3z=2" y "El plano 2x-5y+3z=2 pasa por el punto (2,1,1)" sean paráfrasis la una de la otra se resuelve atendiendo a la adecuada definición de las porciones de texto "pertenecer" y "pasar por", en la BC, y sus relaciones con otros objetos.

3. En cuanto al formalismo semántico desarrollado, por un lado, la sintaxis adoptada para el lenguaje formal permite abordar complejidades semánticas como la correcta resolución de los referentes dentro de un texto en lenguaje natural, haciendo uso de la mera presencia de cuantificadores como los determinantes y la explicitación de su entorno de actuación.

4. Por otro lado la jerarquía de tipos semánticos establecida en el lenguaje formal permite no sólo guiar adecuadamente los análisis sintáctico y semántico sino facilitar las desambiguaciones.

5. Dicho lenguaje formal proporciona una representación única y completa del texto original en lenguaje natural.

   Esto ofrece ventajas evidentes de cara a la posterior utilización de la representación formal, como se ha comprobado en Prógenes.

6. El mecanismo de combinación semántica, mediante reglas que hacen uso del lenguaje formal desarrollado, resuelve de manera sencilla ambigüedades de significado y sintácticas y proporcionan interpretaciones semánticas de forma composicional de parte o todo el texto original.
7. El lenguaje formal desarrollado, y en particular su jerarquía de tipos, permite dar contenido semántico a las entradas del diccionario de forma que realizar el análisis semántico del texto original en base al contenido semántico de cada uno de los elementos, sea una tarea relativamente sencilla.

   Esto prueba la adecuación de una teoría lexicalista para la aproximación al procesamiento del lenguaje natural de un lenguaje tan fuertemente dependiente del dominio como el contemplado.
8. A lo largo de la exposición queda implícito que el uso de un lenguaje como LISP, en el que cualquier expresión válida en el entorno puede ser tratada como declarativa o procedural en cualquier momento, simplifica enormemente la implementación del lenguaje formal diseñado.
9. Pese a que el lenguaje formal refleja la lógica de predicados de primer orden presente en la Base de Conocimientos de Prógenes, no queda limitada su reusabilidad por otro tipo de Bases de Conocimiento, como se ha demostrado en el desarrollo del sistema experto GANESH: Sistema Experto para el Diseño y Valoración de Puestos de Trabajo (Zaccagnini, 1993).
10. Por último, una aproximación lexicalista como la adoptada hace necesaria la participación de expertos en la materia que se quiere tratar, personas capaces de determinar los conceptos fundamentales: su tipo y sus descriptores.

La validación del formalismo diseñado se ha llevado a cabo en el marco de investigación del proyecto Prógenes que, como ya se ha mencionado, es un proyecto de investigación básica dedicado al desarrollo de una base de conocimiento en el área del Cálculo Infinitesimal. El sistema final es capaz de resolver problemas formulados en lenguaje natural al nivel de un curso del primer año de estudios científicos universitarios.

En este momento se encuentran en fase de desarrollo las siguientes líneas de trabajo:

1. Aumentar la cobertura lingüística del analizador para incluir referencias intersentenciales, fenómenos de no-constituencia (como la coordinación de no constituyentes y el vaciado), etc.

2. Ampliar los tipos de error que el sistema entiende a nivel gramatical.

   Por ejemplo, el sistema entiende exactamente igual "Escribir una ecuación para la recta ..." que "Escribir ecuación para la recta ...", aunque no ocurre lo mismo con "Escribir una ecuación para recta ...". Todo ello es debido, como hemos visto, a la aplicación que se ha dado a la semántica léxica y composicional en este formalismo.

En cuanto a futuras líneas de trabajo en este área, podríamos enumerar las siguientes:

1. El estudio de la viabilidad de otro tipo de bases de conocimiento como fuentes de información semántica, asi como la validez general de los procesos de combinación semántica y el lenguaje formal en otros dominios. Un punto clave será la jerarquización de las bases de conocimiento, ya que el grado de refinamiento de una red de herencia con objetos matemáticos es difícil de alcanzar en dominios más generales.
2. La creación de interfaces en lenguaje natural para otros sistemas expertos.
3. Las relativas a la generación de lenguaje a partir de un lenguaje de representación intermedio.

Para finalizar, este tipo de trabajos facilita el abordar problemas relacionados con el procesamiento del lenguaje natural como el acceso a sistemas de bases de datos (Díaz y Rodríguez-Marín, 1994), la interacción con sistemas inteligentes tales como sistemas expertos, robots, tutores inteligentes, algunas de cuyas aproximaciones pueden verse en [(Díaz et al. 1992) y (Díaz et al., 1995)], etc..

Apéndices

5.0 Apéndice A. Ejemplos del léxico

A continuación se ilustran algunos de los elementos del léxico estático y la mayor parte de los componentes del léxico dinámico y análisis parciales del Problema 2. Recordemos que el diccionario contiene unas doscientas entradas y que todos los objetos del léxico son instancias de alguna de las subclases mencionadas en el Capítulo 2.

```
(setf recta
  (make-instance 'nombre
    :spelling    'recta
    :sem         '((LINE (recta (LINEAR-EQUATION @e))))
))
```

```
(setf punto
  (make-instance 'nombre
    :spelling    'punto
    :sem         '((POINT (punto (NUM @N))))
  ))
```

```
(setf a
  (make-instance 'preposicion
    :spelling    'a
  ))
```

```
(setf una
    :spelling    'una
    :sem         '((OBJECT (el (OBJECT (: (VAR %x) (OBJECT @o)))
                               (BOOL @b)))
                   (OBJECT @o))
  ))
```

```
(setf la
  (make-instance 'determinante
    :spelling    'la
    :sem         '((OBJECT (el (OBJECT (: (VAR %x) (OBJECT @o)))
                               (BOOL @b)))
                   (OBJECT @o))
  ))
```

```
(setf y
  (make-instance 'conjuncion
    :spelling    'y
    :sem         '((bool (y (bool @p2) (bool @p1)))
                   (action (pvalues (action @a2) (action @a1))))
  ))
```

```
(setf e
  (make-instance 'conjuncion
    :spelling    'y
    :sem         '(sem y)
  ))
```

```
(setf que
  (make-instance 'relativo
    :spelling    'que
  ))
```

```
(setf donde
  (make-instance 'relativo
    :spelling    'DONDE
  ))
```

```
(setf continua
  (make-instance 'adjetivo
    :spelling    'continua
    :sem         '((BOOL (continua (REALFUN @f) (REAL @r))))
  ))
```

```
(setf creciente
  (make-instance 'adjetivo
    :spelling    'creciente
    :sem         '((BOOL (creciendo (REALFUN @f) (INTERVAL @i))))
  ))
```

```
(setf una+ecuacion
  (make-instance 'signo
    :spelling    'una+ecuacion
    :cat         '(NP)
    :sem         '((EQUATION (ecuacion (LINE @p))))
  ))
```

```
(setf la+recta
  (make-instance 'signo
    :spelling    'la+recta
    :cat         '(NP)
    :sem         '((LINE (el (LINE (<> (VAR %x) (LINE recta)))
                             (BOOL @b)))
                   (LINE recta)
                   (LINE (recta (LINEAR-EQUATION @e))))
  ))
```

```
(setf punto+$P(-3,2)$
  (make-instance 'signo
    :spelling    'punto+$P(-3,2)$
    :cat         '(NP)
    :sem         '((POINT (punto (NUM (-3 2))))))
  ))
```

```
(setf el+punto+$P(-3,2)$
  (make-instance 'signo
    :spelling    'el+punto+$P(-3,2)$
    :cat         '(NP)
    :sem         '((POINT (punto (NUM (-3 2))))))
  ))
```

```
(setf para+la+recta
  (make-instance 'signo
    :spelling    'para+la+recta
    :cat         '(NP \  NP)
    :sem         '((LINE (el (LINE (<> (VAR %x) (LINE recta)))
                             (BOOL @b)))
                   (LINE recta)
                   (LINE (recta (LINEAR-EQUATION @e))))
  ))
```

```
(setf por+el+punto+$P(-3,2)$
  (make-instance 'signo
    :spelling    'por+el+punto+$P(-3,2)$
    :cat         '(NP \ NP)
    :sem         '((POINT (punto (NUM (-3 2))))))
  ))
```

```
(setf una+ecuación+para+la+recta
  (make-instance 'signo
    :spelling    'una+ecuacion+para+la+recta
    :cat         '(NP)
    :sem         '((EQUATION (ecuacion (LINE (LINE (el (LINE
                                                    (: (VAR %x)
                                                       (LINE recta)))
                                                  (BOOL @b))))))
                   (EQUATION (ecuacion (LINE recta)))
                   (EQUATION (ecuacion (LINE (recta (LINEAR-EQUATION @e)))))))
  ))
```

```
(setf pasa+por+el+punto+$P(-3,2)$
  (make-instance 'signo
    :spelling    'pasa+por+el+punto+$P(-3,2)$
    :cat         '(S \ NP)
    :sem         '((BOOL (en (POINT (punto (NUM (-3 2))))
                             (SUBVAR @s))))
  ))
```

```
(setf que+pasa+por+el+punto+$P(-3,2)$
  (make-instance 'signo
    :spelling    'que+pasa+por+el+punto+$P(-3,2)$
    :cat         '(NP \  NP)
    :sem         '((BOOL (en (POINT (punto (NUM (-3 2))))
                             (SUBVAR @s))))
```

```
(setf a+$3x-5y+8=9$
  (make-instance 'signo
    :spelling    'a+$3x-5y+8=9$
    :cat         '(NP \ NP)
    :sem         '((FORMULA (== (++ (-- (** 3 x) (** 5 y)) 8) 0)))
  ))
```

```
(setf paralela+a+$3x-5y+8=9$
  (make-instance 'signo
    :spelling    'paralela+a+$3x-5y+8=9$
    :cat         '(NP \ NP)
    :sem         '((BOOL (paralelo (LINE (== (++ (-- (** 3 x) (** 5 y)) 8) 0))
                                   (LINE @q))))
  ))
```

```
(setf y+que+pasa+por+el+punto+$P(-3,2)$
  (make-instance 'signo
    :spelling    'y+que+pasa+por+el+punto+$P(-3,2)$
    :cat         '((NP \ NP) \ (NP \ NP))
    :sem         '((BOOL (y (BOOL (en (POINT (punto (NUM (-3 2))))
                                      (SUBVAR @s)))
                            (BOOL @p1))))
  ))
```

```
(setf paralela+a+$3x-5y+8=9$+y+que+pasa+por+el+punto+$P(-3,2)$
  (make-instance 'signo
    :spelling    'paralela+a+$3x-5y+8=9$+y+que+pasa+por+el+punto+$P(-3,2)$
    :cat         '(NP \ (NP \ NP))
    :sem         '((BOOL (y (BOOL (en (POINT (punto (NUM (-3 2))))
                                      (SUBVAR @s)))
                            (BOOL (paralelo
                                    (LINE (== (++ (-- (** 3 x) (** 5 y)) 8)
                                              0))
                                    (LINE @q))))))
  ))
```

```
(setf una+ecuación+para+la+recta+paralela+a+$3x-5y+8=9$+y+que+
      pasa+por+el+punto+$P(-3,2)$
  (make-instance 'signo
    :spelling    'una+ecuación+para+la+recta+paralela+a+$3x-5y+8=9$+y+que+
                  pasa+por+el+punto+$P(-3,2)$
    :cat         '(NP)
    :sem         '((EQUATION
                     (ecuacion
                       (LINE
                         (el (LINE (: (VAR %x) (LINE recta)))
                             (BOOL (y (BOOL (en (POINT (punto (NUM (-3 2))))
                                                (SUBVAR @s)))
                                      (BOOL (paralelo
                                              (LINE (== (++ (-- (** 3 x)
                                                                (** 5 y))
                                                            8) 0))
                                              (LINE @q)))))))))))
  ))
```

```
(setf escribir+una+ecuacion+para+la+recta+paralela+a+$3x-5y+8=9$+
      y+que+pasa+por+el+punto+$P(-3,2)$
  (make-instance 'signo
    :spelling    'escribir+una+ecuacion+para+la+recta+paralela+a+$3x-5y+8=9$+
                  y+que+pasa+por+el+punto+$P(-3,2)$
    :cat         '(S)
    :sem         '((EQUATION
                     (hallar
                       (EQUATION
                         (ecuacion
                           (LINE
                             (el (LINE (: (VAR %x) (LINE recta)))
                                 (BOOL (y (BOOL (en (POINT (punto (NUM (-3 2))))
                                                    (SUBVAR @s)))
                                          (BOOL (paralelo
                                                  (LINE (== (++ (-- (** 3 x)
                                                                    (** 5 y))
                                                                8) 0))
                                                  (LINE @q)))))))))))))
  ))
```

6.0 Apéndice B. Ejemplos de enunciados

Los siguientes son algunos ejemplos de enunciados de PRÓGENES y sus traducciones. Las representaciones en lenguaje formal Prógenes de algunos de los textos presentes en este apéndice no pueden ser obtenidas con las reglas presentadas en este trabajo, sino haciendo uso de las expuestas en (Gonzalo, en preparación).

```
Calcular $\integral \frac{x^3}{\sqrt{a^2-x^2}}dx$.

A.$\sqrt{a^2-x^2}\left( -\frac{2a^2+x^2}3 \right)$.
B.$\frac{\sqrt{a^2-x^2}}4 (x^2+a^2)$.
C.$\frac{\sqrt{a^2-x^2}}3 (3a^2-x^2)$.
D.$\frac{\sqrt{a^2-x^2}}3 (2a^2-x^2)$.
E.$\frac{\sqrt{a^2-x^2}}2 (x^2+ax+a^2)$.

(define ((%a :REAL))
 (yields
   (integral (realfun (lambda (x)
                        (// (expt x 3) (sqrt (-- (expt a 2) (expt x 2))) x))))
   (realfun (lambda (x)
              (** (sqrt (-- (expt a 2) (expt x 2)))
                  (-- (// (++ (** 2 (expt a 2)) (expt x 2)) 3)))))
   (realfun (lambda (x)
             (** (// (sqrt (-- (expt a 2) (expt x 2))) 4)
                 (++ (expt x 2) (expt a 2)))))
   (realfun (lambda (x)
             (** (// (sqrt (-- (expt a 2) (expt x 2))) 3)
                 (-- (** 3 (expt a 2)) (expt x 2)))))
   (realfun (lambda (x)
             (** (// (sqrt (-- (expt a 2) (expt x 2))) 3)
                 (-- (** 2 (expt a 2)) (expt x 2)))))
   (realfun (lambda (x)
             (** (// (sqrt (-- (expt a 2) (expt x 2))) 2)
                 (++ (expt x 2) (** a x) (expt a 2)))))))
```

```
Hallar $\integral \sin(6x)\;\sin(4x)dx$.

A. $-\frac{\sin (10x)}{20}+\frac{\cos (2x)}{4}+C$.
B. $-\frac{\sin (10x)}{20}+\frac{\sin (2x)}{4}+C$.
C. $\frac{\sin (10x)}{20}+\frac{\sin (2x)}{4}+C$.
D. $-\frac{\cos (10x)}{20}-\frac{\sin (2x)}{4}+C$.
E. $-\frac{\cos (10x)}{20}+\frac{\cos (2x)}{4}+C$.

(yields (integral (realfun (lambda (x) (* (sin (* 6 $x)) (sin (* 4 $x))) x)))
   (realfun (lambda (x) (++ (-- (// (sin (** 10 %x)) 20))
                            (// (cos (** 2 %x)) 4) c)))
   (realfun (lambda (x) (++ (-- (// (sin (** 10 %x)) 20))
                            (// (sin (** 2 %x)) 4) c)))
   (realfun (lambda (x) (++ (// (sin (** 10 %x)) 20)
                            (// (cos (** 2 %x)) 4) c)))
   (realfun (lambda (x) (++ (-- (// (cos (** 10 %x)) 20))
                            (-- (// (sin (** 2 %x)) 4)) c)))
   (realfun (lambda (x) (++ (-- (// (cos (** 10 %x)) 20))
                            (// (cos (** 2 %x)) 4) c))))
```

```
Hallar la función que pasa por $(\frac 12,0)$ y tiene derivada:
$ f'(x) = x \sqrt{1-x^2}.$

A.  $ -\frac 13 \sqrt{(1-x^2)^3} + \frac {\sqrt 3}8 $.
B.  $ \frac 23 \sqrt {1-x^2}$.
C.  $ -\frac 12 \sqrt{(1-x^2)^3} - \frac 38$.
D.  $ \sqrt{\dfrac{1-x^2}x} - \dfrac 54$.
E.  $ \sqrt{1-x^2} - \dfrac{x^2}{\sqrt {1-x^2}} + 2$.

(yields
 (pfind
  (pthe (<> %f REALFUN)
    (pand (== (img %f 1/2) 0)
          (== (der %f)
              (realfun (lambda (x) (** x (pqsrt (-- 1 (pexpt x 2))))))))))
  (realfun (lambda (x) (++ (** -1/3 (pqsrt (pexpt (-- 1 (pexpt x 2)) 3)))
                           (// (pqsrt 3) 8))))
  (realfun (lambda (x) (** 2/3 (pqsrt (-- 1 (pexpt x 2))))))
  (realfun (lambda (x) (++ (** -1/2 (pqsrt (pexpt (-- 1 (pexpt x 2)) 3)))
                           (// -3 8))))
  (realfun (lambda (x) (-- (pqsrt (// (-- 1 (pexpt x 2)) x)) 5/4)))
  (realfun (pqsrt (++ (-- 1 (pexpt x 2))
                      (// (-- (expt x 2)) (pqsrt (-- 1 (pexpt x 2))))
                      2))))
```

```
Hallar los valores de $x$ para los que la recta normal a
$f(x) = \sqrt {x^3 +1} $ pasa por el origen.

A. ${1}$.
B. ${-1, -\frac 23}$.
C. $ {0,-\frac 23}$.
D. ${0}$.
E. Para ningun $x$.
F. ${1,-1}$.

(define ((%f (realfun (lambda (x) (psqrt (++ (pexpt x 3) 1))))))
  (yields
     (pfind (pthe (<> %x :real)
                  (in (point 0 0) (normal-line %f %x))))
     (the-set 1)
     (the-set -1 -2/3)
     (the-set 0 -2/3)
     (the-set 0)
     :empty-set
     (the-set 1 -1))
```

```
Calcular la ecuación de la recta tangente a la curva definida por:
$3xy^2 + y = e^x $ en el punto $(0,1)$.

A. $x=0$.
B. $y=-2x + 1$.
C. $x=-2y+2$.
D. $y=1$.
E. $y= \frac x2 +1$.
F. $x= \frac y2-1$.

(yields (pfind
        (equation (pthe (<> %l line)
                        (tangent %l
                                 (curve (== (++ (** 3 !x (pexpt !y 2)) !y)
                                            (pexp !x)))
                                 (point 0 1)))))
     (== !x 0)
     (== !y (-- (** 2 !x)))
     (== !x (++ (-- (** 2 !y)) 2))
     (== !y 1)
     (== !y (++ (// !x 2) 1))
     (== !x (-- (// !y 2) 1)))
```

```
Si $\quad f(x)=|x^2-9|\quad $, entonces $\quad f\quad $ es creciente en
el conjunto:


A. $(-\infty ,\infty )$
B. $ [-3,0]\cup [3,\infty )$
C. $[-3,3]$
D. $ \bold{R} \setminus [-3,3]$
E. $(-\infty ,0]$
F. $[0,\infty )$

(define ((%f REALFUN (realfun (lambda (x) (pabs (-- (pexp x 2) 9))))))
  (test
    (increasing %f :REAL)
    (increasing %f (punion (interval :closed -3 :closed 0)
                           (interval :closed 3 :open :infinity)))
    (increasing %f (interval :closed -3 :closed 3))
    (increasing %f (-- :REAL (interval :closed -3 :closed 3)))
    (increasing %f (interval :open :-infinity :closed 0)))
```

```
Si la función $\quad f:\,\,\bold {R}\to \bold {R}\quad $
viene definida por

$
\aligned
& f(x)=x^2+3      \quad \text {si}      \quad  x<-1,
\quad f(x)=|x-6| \quad \text {si}      \quad -1\leq x<6,\\
& f(x)=(x-6)(x^2+3)     \quad \text {si}  \quad  6<x<10
\quad \text {y} \quad f(x)=1 \quad \text {si}  \quad  10\leq x
\endaligned
$

los puntos de discontinuidad de $\quad f\quad $ son:

A. ${-1,6,10}$
B. no hay
C. ${6,10}$
D. ${-1,10}$
E. ${-1,6}$
F. ${-1}$

(define ((%f REALFUN (realfun (lambda (x) (pcond ((<< x -1) (++ (pexpt x 2) 3))
                                ((<=< -1 x 6) (pabs (-- x 6)))
                                ((<< 6 x 10)
                                 (** (-- x 6) (++ (pexpt x 2) 3)))
                                ((<<== 10 x) 1))))))
  (yields
    (disc-points %f)
    (the-set -1 6 10)
    :empty-set
    (the-set 6 10)
    (the-set -1 10)
    (the-set -1 6)
    (the-set -1)))
```

```
La integral $\int _0^{\pi /3} sen \, 3t \, dt$ vale:

A. $5/3$
B. $-1$
C. $0$
D. $2/3$
E. $sen \pi$
F. $-5/3$

(yields
  (integral (realfun (lambda (x) (psin (** 3 t))) 0 (// :pi 3)))
  5/3 -1 0 2/3 (psin :pi) -5/3)
```

```
Hallar el coeficiente de  $ x^5 $  en el desarrollo de Taylor de
$  f(x) = x\,e^{\tsize -x^2}  $ .


A. $0$
B. $1$
C. $1/2$
D. $-2$
E. $1/6$
F. $-1$

(yields
  (coef-taylor (realfun (lambda (x) (** x (pexp (-- (pexpt x 2))))))) 5)
  0 1 1/2 -2 1/6 -1)
```

```
El valor maximo de la función $x^4-x^2$ en el intervalo $[-2,2]$ es
A. $0$.
B. $-\frac 12$.
C. $12$.
D. $\frac 1{\sqrt 2}$.
E. $1$.
F. $-1$.

(yields (abs-max-value (realfun (lambda (x) (-- (pexpt x 4) (pexpt x 2))))
                       (interval :close -2 :close 2))
        0 -1/2 12 (// 1 (psqrt 2)) 1 -1)
```

```
Los puntos criticos de la función $F(x)=\integral_1^{x^2}(t^2-1)dt$ son
A. ${2,0}$.
B. ${1,0}$.
C. ${0,-1}$.
D. ${-1,0,1}$.
E. ${-1,1}$.
F. ${0}$.

(define ((%f realfun (realfun
                      (lambda (x)
                        (definite-integral 0
                                           (pexpt x 2)
                                           (realfun (t) (-- (pexpt t 2)
                                                            1)))))))
  (yields (critical-points %f)
          (the-set 2 0)
          (the-set 1 0)
          (the-set 0 -1)
          (the-set -1 0 1)
          (the-set -1 1)
          (the-set 0)))
```

```
Las ecuaciones de las tangentes en los puntos de inflexión de
$y=x^4-6x^3+12x^2-3x$ son
A. $y=3x-2$ y $y=0$.
B. $y=2x-3$ y $y=3$.
C. $y=2x-3$ y $y=0$.
D. $y=5x-7$ y $y=0$.
E. $y=3x-8$ y $y=x+2$.

(define ((%y CURVE (curve (== y (++ (expt x 4)
                                    (** -6 (expt x 3))
                                    (** 12 (expt x 2))
                                    (** -8 x))))))
  (yields (equations (tangent %y (inflection-points %y)))
          (plist (== (++ (** 3 !x) (** -1 !y) -2) 0) (== !y 0))
          (plist (== (++ (** 2 !x) (** -1 !y) -3) 0) (== !y 3))
          (plist (== (++ (** 2 !x) (** -1 !y) -3) 0) (== !y 0))
          (plist (== (++ (** 5 !x) (** -1 !y) -7) 0) (== !y 0))
          (plist (== (++ (** 3 !x) (** -1 !y) -8) 0) (== (++ !x
                                                              (** -1 !y)
                                                              -2) 0)))))
```

```
Hallar los puntos de la parabola $x = y^2$ más proximos al punto $(0, 3)$.

A. $(4, 2)$
B. $(4, -2)$
C. $(0, 0)$
D. $(1, 1)$
E. $(9, 3)$

(yields
  (nearest-point (parabola (== !x (pexpt !y 2))) (point 0 3))
  (point 4 2)
  (point 4 -2)
  (point 0 0)
  (point 1 1)
  (point 9 3))
```

```
Sea $f(x)$ la función $f(x) = \cases
   sen x  \quad & x \le \pi \\
   ax+b  \quad & x \gt \pi.
   \endcases$.
   Calcular $a$ y $b$ para que $f'(x)$ exista:

A.$ a = \pi^2$, $b = 2$
B.$ a = -1$, $b = \pi$
C.$ a = -2\pi$, $b = 3$
D.$ a = 2$, $b = 1$
E.$ a = \pi$, $b = -\pi$

(define ((%f (realfun (lambda (x)
                        (pcond ((<<= :-infinity x :pi) (++ (** %a x) %b))
                               ((<< :pi x :infinity)  (psin x))))))))
  (yields
    (pfind
       (pthe (plist (<> %a REAL) (<> %b REAL))
             (derivable %f :pi)))
    (plist (pexpt :pi 2) 2)
    (plist -1 :pi)
    (plist (* -2 :pi) 3)
    (plist 2 1)
    (plist :pi (-- :pi))))
```

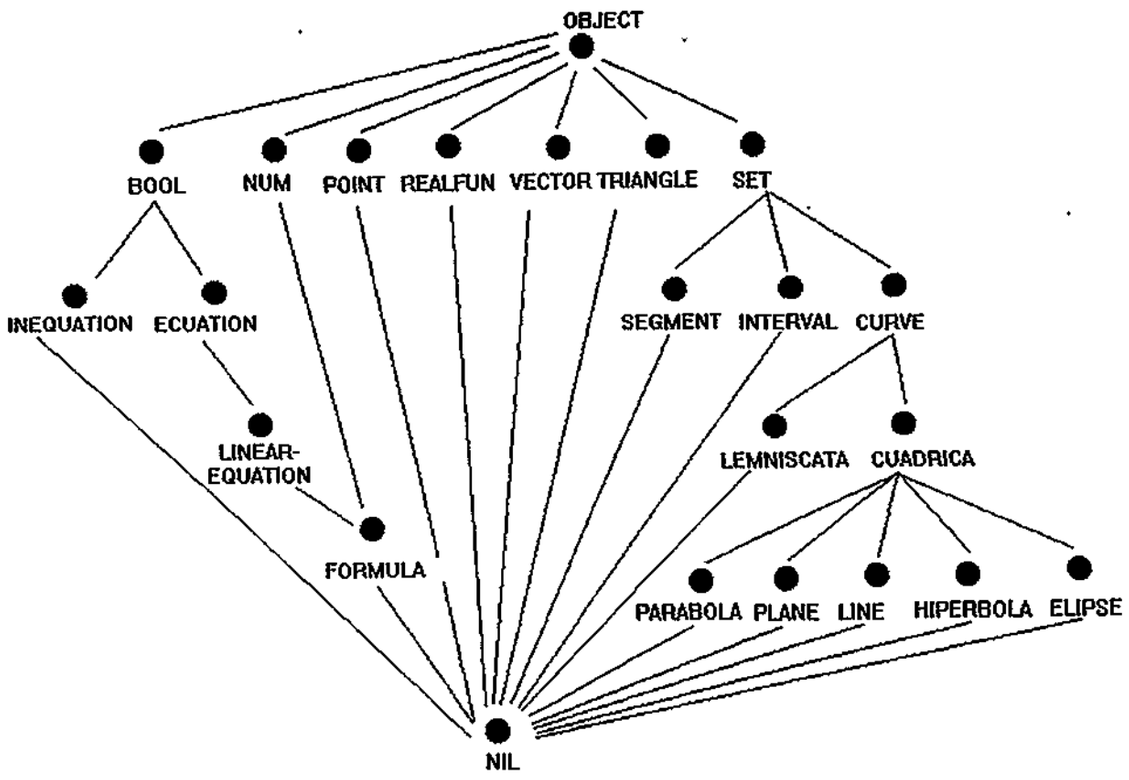
```
Determinar los puntos de tangente vertical de la curva
$ x(t) = 1 + \frac{\sqrt 2}{2} \cos t - \frac{\sqrt 2}{2} {\rm sen}\;
t, \;\;\;\;\; y(t) = 2 + \sqrt 2 {\rm sen}\; t + \sqrt 2 \cos t $.

A.  $(0,0)$, $(1,1)$ y $(2,2)$
B.  $(0,2)$, $(1,2)$ y $(2,2)$
C.  $(1,0)$ y $(1,4)$
D.  $(0,0)$
E.  No tiene ninguno.
F.  $(2,2)$ y $(0,2)$

(yields
  (pthe (<> %p PUNTO)
    (vertical
      (tangent-line
        (curve (plist (== !x (++ 1 (** (psqrt 2) 1/2 (pcos !t))
                                   (** (psqrt 2) 1/2 (psin !t))))
                      (== !y (++ 2 (** (psqrt 2) (psin !t))))))
        %p)))
  (the-set (point 0 0) (point 1 1) (point 2 2))
  (the-set (point 0 2) (point 1 2) (point 2 2))
  (the-set (point 1 0) (point 1 4))
  (the-set (point 0 0))
  :empty-set
  (the-set (point 2 2) (point 0 2)))
```

```
Sabiendo que $f$ es derivable en $0$ y que $f'(0)=2$,
calcular $\lim_{h \to 0}\frac1h[f(2h)-f(h^2)]$.

A. $1$
B. $4$
C. $3/2$
D. $e^2$
E. $\log\, 2$
F. $2h$

(define ((<> %f REALFUN))
  (yields
    (pfind (lim (realfun (lambda (h)
                           (-- (img %f (** 2 h)) (img %f (pexpt h 2))))))
                0)
           (== (img (der %f) 0) 2))
    1
    4
    3/2
    (pexp 2)
    (plog 2)))
```

```
Hallar el mínimo valor que toma la función $f(x)=2x-\log x$.
A. $1+\log 2$.
B. $\log (1+\sqrt 2)$.
C. $2-\log 2$.
D. $-3e^2$.
E. $5\log 7$.
F. $2+e^2$.

(define ((%f REALFUN (realfun (lambda (x) (-- (** 2 x) (plog x))))))
  (yields (abs-min-value %f)
          (++ 1 (plog 2))
          (plog (++ 1 (psqrt 2)))
          (-- 2 (plog 2))
          (** -3 (pexp 2))
          (** 5 (plog 7))
          (++ 2 (pexp 2))))
```

7.0 Apéndice C. Jerarquía de tipos

El siguiente es el Diagrama de Hasse de la jerarquía de tipos para el dominio de Matemáticas. En él están ilustrados la mayor parte de los tipos que intervienen.



Referencias

1.
Ades, A.E. y Steedman, M.J.: "On the order of words", en *Linguistics and Philosophy, 4*, págs: 517-558, 1982.

2.
Barwise, J. y Perry, J.: "Situaciones y actitudes", Madrid, Visor, 1992.

3.
Blackburn, P.: "Introduction to lattices, algebras and their applications in formal semantics", en: *Lectures notes for the Second European Summer School in Language, Logic and Information*, 1990.

4.
Booch, G.: "Object Oriented Design With Applications", Benjamin/Cummings 1990.

5.
Brooks, F.: "No Silver Bullet" en T. de Marco y T. Lister (eds), *Software State-Of-The-Art: Selected Papers*, Dorset House Publishing, 1990, págs: 14-29.

6.
Bundy, A., Byrd, L., Luger, G., Mellish, C., Milne, R., Palmer, M.: "MECHO: A program to solve mechanics problems". Ed.: Department of Artificial Intelligence. University of Edimburgh. Working Paper n° 50, 1979.

7.
Carlson, G. N.: "Natural Kinds and Common Nouns" En A. von Stechow y D. Wunderlich (eds.) 1991.

8.
Castells, P.: "Heurísticas y Metaconocimiento en Resolución Automática de Problemas de Matemáticas". Tesis Doctoral. Facultad de Ciencias. Universidad Autónoma de Madrid. 1994.

9.
Castells, P., Díaz, J., Gonzalo, J., Moriyón, R., Rodríguez-Marín, P., Saiz, F., Tobar, M.J.: "Un sistema para la resolución de problemas enunciados en len-

guaje natural". *Proceedings del Primer Congreso Nacional de Programación Declarativa: Prode.92*, Madrid, Septiembre de 1992.

10. Castells, P., Díaz, J., Gonzalo, J., Moriyón, R., Rodríguez-Marín, P., Saiz, F., Tobar, M.J.: "A Scientific Problem Solver with a Natural Language Interface." *The Seventh International Symposium on Computer and Information Sciences: ISCIS VII*, Turquía, Noviembre 1992.

11. Castells, P., Díaz, J., Moriyón, R., Rodríguez-Marín, P., Saiz, F.: "PROGENES: Una base de conocimiento para la resolución de problemas de cálculo infinitesimal." *AEPIA 91*, Octubre de 1991.

12. Cater, A.W.S.: "Request-based parsing with low level syntactic recognition." En: *"Spark and Jones and Wilks"* págs: 141-147, 1982.

13. Chierchia, G. y McConnel-Ginet, S.: "Lambda abstraction" en Chierchia y S. McConnel-Ginet (eds.), 1990.

14. Cox, B.: "Object-Oriented Programming - An Evolutionary Approach", Addison-Wesley, 1986.

15. Daelemans, W., De Smedt, K., Gazdar, G.: "Inheritance in Natural Language Processing", en *1992 Association for Computational Linguistics*, págs: 205-218.

16. Davey, B.A. y Priestley, H.A.: "Introduction to Lattices and Order". Ed.: Cambridge University Press, 1990.

17. Díaz, J.; Cuezva, F.; Ibáñez, C.; Rodríguez, F.; Zaccagnini, J.L.; De Pablo, E.: *FORMIP: Herramienta para la formación de personal en el control de procesos* Boletín n° 8 de la Asociación para el Desarrollo de la Informática Educativa, 1992.

18. Díaz, J.; Gonzalo, J.; Rodríguez, P.: *El formalismo semántico en la Interfaz de Lenguaje Natural de Prógenes.* Boletín n° 14 de la Sociedad Española para el Procesamiento del Lenguaje Natural, 1993.

19.

Díaz, J.; Gonzalo, J.; Rodríguez, P.: *Semantic Analysis of Mathematical Texts.* Inf. Téc. del IIC, A01/93, 1993.

20. Díaz, J.; Rodríguez-Marín, P.: "PROGENES: La interfaz en Lenguaje Natural". *Boletín n° 11 de la Sociedad Española para el Procesamiento del Lenguaje Natural*, Diciembre de 1992.

21. Díaz, J.; Pascual, P.; Sigüenza, J.A.; López-Fando, M.P.: *Aplicación de la Tecnología Multimedia al desarrollo de Software Educativo* RED, (previsto 1995).

22. Díaz, J.; Rodríguez-Marín, P.: "Recuperación Automática de Información Textual", Inf. Téc. del IIC, B02/91, 1994.

23. Dowty, D.: "Type Raising, Functional Composition, and Non-Constituent Conjunction" en *Categorial Grammars and Natural Language Structures.* Ed.: Deirdre Publishing Company, 1989.

24. Dowty, D., Wall, R. y Peters, S.: "Introduction to Montague semantics (Synthese language library 11). Dordrecht: D. Reidel, 1981.

25. Emele, M.C. y Zajac, R.: "Typed unification grammars", en: *COLING 90*, vol.2, Helsinki 1990.

26. Flickinger, D., Pollard, C. y Wasow, T.: "Structure-sharing in Lexical Representation". En: *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics* 262-267, University of Chicago, Chicago, 1985.

27. Flickinger, D., Nerbonne, J.: "Inheritance and Complementation: A case Study of Easy Adjetives and Related Nouns". En: *1992 Association for Computational Linguistics*, págs: 269-309.

28. Fodor, J.D.: "Semántica: Teorías del significado en la gramática generativa", Madrid, Cátedra, 1985.

29.

Friedman, J., Moran, D. y Warren, D.: "Evaluating English sentences in a logical model: A process version of Montague grammar". En: *Proceedings of the 7th International Conference on Computational Linguistics*, Bergen, Norway, August, 1978.

30. Garijo, F.J., Verdejo, M.F., Díaz, A., Fernández, I., Sarasola, K.: "CAPRA: An intelligent System to Teach Novice Programmers". En: *Actas del II Congreso Mundial Vasco*, 1987.

31. Gonzalo, J.: "Una aproximación orientada a objetos para el tratamiento de excepciones en gramáticas". Tesis Doctoral. Facultad de Ciencias. Universidad Autónoma de Madrid. En preparación.

32. Gonzalo, J., Rodríguez-Marín, P., y Díaz, J.: "Formal Representation of Mathematical Texts." En *Les Chemins du texte*. Ed.: F. Poswick y G. Bernard, Slatkine-Champion, Genève-Paris, 1992.

33. Grosz, B.J., Sparck Jones, K., Webber, B.L.: "Readings in Natural Language Processing." Ed.: Barbara J. Grosz, Karen Sparck Jones, Bonnie Lynn Webber. Morgan Kaufmann Publishers, 1986.

34. Haugeneder, H.: "A computational Model for processing coordinate structures: Parsing Coordination without grammatical specification", en: *ECAI 92*. Ed.: Wiley & Sons, 1992.

35. Hirst, G.: "Semantic interpretation and the resolution of ambiguity". Ed.: Cambridge University Press, 1989.

36. Hobbs, J.R. y Rosenchein, S.J.: "Making Computational Sense of Montague's Intensional Logic", en *Artificial Intelligence 7*, págs: 287 - 306. 1978.

37. Huang, X.: "Dealing with conjunction in a machine translation environment", en Proc. *1st meeting of the European Chapter of the ACL*, 1983.

38. Jackendoff, R.: "Semantic Structures", The MIT Press, 1990.

39.
Kamp, H. y Reyle, U.: "From Discourse to Logic. Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory", Dordrecht, Kluwer, 1993.

40.
Katz, J.J.: "Teoría Semántica", Madrid, Aguilar, 1980.

41.
Ladusaw, W. A.: "Teoría Semántica". En *Panorama de la Lingüística moderna de la Universidad de Cambridge. I. Teoría Lingüística: Fundamentos.* Visor Distribuciones S.A., 1990.

42.
Lappin, S.: "The syntax and Semantics of NPs." Ed.: MOUTON DE GRUYTER, Amsterdam, 1988 Volumen 26-6.

43.
Lappin, S. and Mccord, M.: "Anaphora Resolution in Slot Grammar", 1990.

44.
Lonning, J.T.: "Some Aspects of the Logic of Plural Noun Phrases", COSMOS Report number 11, Department of Mathematics, University of Oslo, P.O. Box 1053, Blindern, 0316 Oslo 3, Norway, 1989.

45.
Meyer,B.: "Object-Oriented Software Construction", Prentice Hall, New York 1988.

46.
Montague, R.: "Formal Philosophy". Ed.: R.H. Thomason, Yale University Press, New Haven, 1974.

47.
Moreno, J.C.: "Lógica Formal y Lingüística. Una introducción a la gramática de R. Montague". Ed.: Ediciones de la Universidad Autónoma de Madrid, 1985.

48.
Moreno-Torres, I. y Solias, M.T.: "Un analizador con 'chart' para gramáticas categoriales de unificación". En: *Procesamiento del Lenguaje Natural*, Boletín n°9, pp 207-225. Ed.: Sociedad española para el procesamiento del Lenguaje Natural, 1991.

49.

Nierstrasz, O.: "A survey of Object-Oriented Concepts", en W.Kim y F.Lochovski (eds), *Object-Oriented Concepts, Databases and Applications*, ACM Press, 1989, págs: 3-22.

50. Oehrle, R., Bach, E. y Wheeler, D.: "Categorial Grammars and Natural Language Structures", Reidel 1988.

51. Pollard, C.J. y Sag,I.A.: "Head-Driven Phrase Structure Grammar" University of Chicago Press and CSLI publications, 1993.

52. Riesbeck, C. y Schank, R.: "Comprenhension by computer: Expectation-based analysis of sentences in context.", en *Studies in the perception of language*. Ed.: LEVELT, Willen J.M. and FLORES D'ARCAIS, G.B.. New York, 1978.

53. Salas, S.L.. y Hille, E.: "CALCULUS, de una y varias variables". Ed.: Reverte, 1982.

54. Saiz, F.: "Fundamentación de un sistema para la resolución automática de problemas". Tesis Doctoral. Facultad de Ciencias. Universidad Autónoma de Madrid. 1994.

55. Sarasola, K.: "CAPRATE, un sistema de interpretación de problemas en Lenguaje Natural". Tesis Doctoral. Facultad de Informática. Universidad del Pais Vasco, 1988.

56. Schank, R.G.: "Dynamis Memory: A theory of reminding and learning in people and computers". Cambridge University Press, 1982.

57. Sernadas, A.: "Object-Oriented Programming". *Fifth European Summer School in logic language and information*, Facultade de Letras. Universidade de Lisboa. Portugal, 1993.

58. Steele, G.L.: "Common Lisp.". Ed.: Digital Press, 1990.

59.

Stroustrup, B.: "The C + + programming language", segunda edición. Addison-Wesley, 1992.

60. Tarnawsky, G.O.: "Knowledge Semantics". Doctoral dissertation, Department od Linguistics, New York University, 1982.

61. Wesche, B. y König, E.: "Introduction to Categorial Grammar", en: *Second European Summer School in Language, Logic and Information.* Katholieke Universiteit Leuven, Belgium, 1990.

62. Wilks, Y.: "Theoretical Issues in Natural Language Processing". Ed.: Lawrence Erlbaum Associates, 1989.

63. Winograd, T.: "Understanding Natural Language". Ed.: Academic Press, New York, 1984.

64. Wittenburg, K. y Wall, R.E.: "Parsing with Categorial Grammar in Predictive Normal Form". En: *Inheritance hierarchies in Knowledge representation and programming languages.* Ed.: Wiley, 1990.

65. Yampol, T. y Karttunen, L.: "Efficient Implementation of PATR for Categorial Unification Grammar". En: *COLING 90* Vol 2. Ed.: Hans Karlgren.

66. Zaccagnini, J.L.; García, A.; Abasolo, J.M.; García, J.: "El Sistema Experto GANESH de valoración de puestos de trabajo. Un caso de aplicación de la Inteligencia Artificial en el entorno de la gestión empresarial." Actas de la CAEPIA, 93, págs: 493-500. Madrid, Noviembre de 1993.

67. Zeevat, H., Klein, E. and Calder, J.: "An introduction to Unification Categorial Grammar" en Haddock,N.J. Klein,E. and Morrill,G. (eds.): *Edinburgh Working Papers in Cognitive Science, vol. 1: Categorial Grammar, Unification Grammar and Parsing.* 1987.

Bibliografía

1.
Akama, S.: "Methodology and Verifiability in Montague Gramar", en: *COLING 86*, 1986.

2.
Amori, R.: "Criteria for Succesful Natural Language Interfaces", en: *The Fourth International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems*, Proceedings, volumen II, págs: 562-569. Hawaii, 1991.

3.
Barwise, J. y Perry, J.: "Situations and attitudes", Cambridge, MIT Press, 1983.

4.
Block, H.U., Haugenader, H.: "An Efficiency-oriented LFG PARSER". En: *Readings in NLP*. Ed.: B.J. Grosz, K. Sparck Jones & B.L. Webber, Morgan Kaufmann Publishers, Inc., 1986.

5.
Boguraev, B., Briscoe, T.: "Computational Lexicography for Natural Language Processing". Ed.: Longman Group, 1990.

6.
Bouma, G.: "Defaults in Unification Grammar", en: *28th Annual Meeting of the ACL*, Pittsburgh 1990.

7.
Bouma, G.: "Efficient processing of flexible categorial grammar", en: *Fourth Conference of the European Chapter of the Association for Computational Linguistics*, Manchester 1989.

8.
Boyer, R., Moore, J.: "A Computational Logic Handbook". Ed.: Academic Press, Inc., 1988.

9.
Brachman, R.J.: "The future of knowledge representation", en *AAAI-90*, págs: 1082- 1092.

10.
Burris, S. y Sankappanavar, H.P.: "A course inUniversal Algebra, Graduate Texts in Mathematics". Ed.: Springer Verlag, 1981.

11.
Calder, J., Klein, E., y Zeevat, H.: "Unification Categorial Grammar: A concise, extendable grammar for Natural Language Processing", *COLING 88*, Budapest 1988.

12.
Carbonell, J.G. y Hayes, P.J.: "Robust Parsing using Multiple Construction Specific Strategies". En: *Natural Language Parsing Systems*, Springer Verlag 1987.

13.
Casadio, C.: "Semantic Categories and the development of Categorial Grammars", en: *Categorial Grammars and Natural Language Structures*. Ed.: Reidel Publishing Company, 1988.

14.
Chi, M., Feltovich, P. y Glaser, R.: "Categorization and representations of physics problems by experts and novices", en *Cognitive Science 5*, 1981.

15.
Constable, R.L., et al.: "Implementing Mathematics with the Nuprl Proof Development System". Ed.: Prentice Hall, 1986.

16.
Cox, B.: "There is a Silver Bullet", *Byte*, 1990, 209-218.

17.
Dahl, V., Saint-Dizier, P.: "Natural Language Understanding and Logic Programming, II". Ed.: North-Holland, 1988.

18.
Davidson, D.: "The logical form of action sentences", en: *The logic of decision and action*. Ed.: University of Pittsburgh Press, Pittsburgh 1967.

19.
Earley, J.: "An Efficient Context-Free Parsing Algorithm". En: *Readings in NLP*. Ed.: B.J. Grosz, K. Sparck Jones & B.L. Webber, Morgan Kaufmann Publishers, Inc., 1986.

20.

Elhadal, M.: "Types in Functional Unification Grammars", en: *28th Annual Meeting of the ACL*, Pittsburgh 1990.

21. Findler, N.V.: "An Artificial Intelligence Technique for Information and Fact Retrieval". Ed.: MIT Press, 1991.

22. Gallin, D.: "Intensional and Higher-Order Modal Logic with Application to Montague Semantics". Ed.: North Holland, Amsterdam, 1975.

23. Gazdar, G.: "Phrase Structure Grammars and Natural Languages", en: *8th IJCAI*, Karlsruhe 1983.

24. Gazdar, G.: "Applicability of Indexed Grammars to Natural Languages", en: *Natural Language Parsing and Linguistic Theories*, pp 69-94, Reidel Publishing Company, 1988.

25. Genthial, D., Courtin, J. y Kowalsky, I.: "Contribution of a category hierarchy to the robustness of syntactic parsing", en: *COLING 90*, Helsinki, 1990.

26. Gerdemann, D. y Hinrichs, E.W.: "Functor driven natural language generation with categorial unification grammars", en: *COLING 90* vol 2, Helsinki 1990.

27. Grätzer, G.: "Lattice Theory: First concepts and distributive lattices". Ed.: W.H. Freeman and Company, San Francisco, 1971.

28. Grishman, R., Ksiezyk, K.: "Causal and Temporal Text Analysis: The role of the Domain Model". En: *COLING 90*, Vol. 3, pp 126-131.

29. Grishman, R., Sterling, J.: "Information Extraction and Semantic Constraints". En: *COLING 90*, Vol. 3, pp 355-357.

30. Heim, I.: "The semantics of definite and indefinite noun phrases" Tesis Doctoral, University of Massachusetts at Amherst. Publicado por la University of Massachusetts Graduate Linguistics Association, 1982.

31.
Hepple, M.: "Normal form theorem proving for the lambeck calculus", en: *COLING 90*, Helsinki, 1990.

32.
Hepple, M.: "Chart parsing Lambeck grammars: modal extensions and incrementality", en: *COLING 90*, Nantes 1990.

33.
Hepple, M. y Morrill, G.: "Parsing and derivational equivalence", en: *IV Conference of the European Chapter of the ACL*, Manchester, 1989.

34.
Hopcroft, J.E. y Ullman, J.D.: "Introduction to Automata Theory, Languages and Computation". Ed.: Addison Wesley, 1979.

35.
Jacobs, P.S.: "PHRED: A generator for Natural Language Interfaces", en: *Natural Language Generator Systems.* Ed.: Springer-Verlag, New York, 1988.

36.
Kamp, H.: "A theory of truth and semantic representation". En Groenendijk et al., 1981.

37.
Kaplan, R.M. y Maxwell III, J.T.: "Constituent coordination in Lexical Functional Grammar", en: COLING 88, Budapest 1988.

38.
Kay, M.: "Parsing in functional unification grammar". En: *Readings in NLP.* Ed.: B.J. Grosz, K. Sparck Jones & B.L. Webber, Morgan Kaufmann Publishers, Inc., 1986.

39.
Keenan, E. y Faltz, L.: "Boolean semantics for Natural Language". Dordretch: Foris, 1985.

40.
Kindermann, J. y Meier, J.: " An extension of LR-Parsing for Lexical Functional Grammar". En: *Natural Language Parsing and Linguistic Theories*, pp 131-148. Ed.: D. Reidel Publishing Company, 1988.

41.
Klein, E.: "VP ellipsis in DR theory". En Groenendijk y Stockhoff, 1990.

42.
Klein, E. y Johnson, M.: "Discourse, anaphora and parsing". En *Proceedings of the 11th International Conference on Computational Linguistics and the 24th Annual Meeting of the Association for Computational Linguistics* Institut fuer Kommunicationsforschung und Phonetik, Bonn University, 1986.

43.
König, E.: "The complexity of parsing with extended categorial grammars", en: *COLING 90*, vol. 2, Helsinki, 1990.

44.
Larkin, J., Mc Dermott, J., Simon, D., y Simon, H.: "Models of Competence in solving physics problems" en: *Cognitive Science 4*, 1980.

45.
Lavelli, A. y Stock, O.: "When something is missing: ellipsis, coordination and the chart", en: *COLING 90*, Helsinki, 1990.

46.
Lytinen, S.L.: "The organization of knowledge in a multi-lingual, integrated parser". Tesis Doctoral, Department of Computer Science, Universidad de Yale, 1984.

47.
Mauldin, M.L.: "Information Retrieval by Text Skimming". Ed.: CMU-CMT-90-122.

48.
Meadow, C.T.: "Text Information Retrieval Systems". Ed.: Academic Press, Inc., 1992.

49.
Milne, R.W..: "Resolving lexical ambiguity in a deterministic parser". En: *Computational Linguistics, 12*(1), Enero-Marzo de 1986, págs. 1-12.

50.
Milward, D.: "Coordination in an axiomatic grammar". En: *COLING 90*, vol. 3, Helsinki, 1990.

51.
Moens, M., Calder, J., Klein, E., Reape, M. y Zeevat, H.: "Expressing generalizations in unification-based grammar formalisms", en: *IV Conference of the European Chapter of the ACL*, Manchester, 1989.

52.

Moreno, J.C.: "Curso Universitario de Lingüística General. Tomo II: Semántica, pragmática, morfología y fonología". Ed.: Síntesis, 1994.

53. Newmeyer, F.J.: "Panorama de la lingüística moderna. I.Teoría Lingüística: Fundamentos". Ed.: Visor Distribuciones, S.A., 1990.

54. Numazaki, H. y Tanaka, H.: "A new parallel algorithm for generalized LR parsing", en: *COLING 90*, vol. 2, Helsinki, 1990.

55. Pareschi, R.: "A definitive clause version of Categorial Grammar", en: 26th Annual Meeting of the ACL, Búfalo, 1988.

56. Pastre, D.: "An Automatic Theorem Proving Using Knowledge and Metaknowledge in Mathematics". En *Artificial Intelligence 38* (1989), pp 257-318. Ed.: Elsevier Science Publishers B.V. (North-Holland), 1989.

57. Pollard, C.J.: "Categorial Grammar and Phrase Structure Grammar: an excursion on the syntax-semantics frontier", en: *Categorial Grammars and Natural Language structures*, Ed.: Reidel Publishing Company, Dordrecht, 1988.

58. Reyle, U.: " Compositional Semantics for LFG". En: *Natural Language Parsing and Linguistic Theories*, pp 448-474, Ed.: D. Reidel Publishing Company, 1988.

59. Salton, G.: "Automatic Text Processing: the transformation, analysis, and retrival of information by computer". Ed.: Addison-Wesley Publishing Company, 1989.

60. Schubert, L.. y Pelletier, F.J.: "From English to Logic: Context-Free Computation of 'Conventional' Logical Translations", University of Alberta, 1981.

61. Shieber, S.M.: "An Introduction to Unification-Based approaches to Grammar", Ed.: CSLI, Ventura Hall. Standford University, 1986.

62. Steedman, M.: "Combinators and Categorial Grammars.". En *Categorial Grammars and Natural Language Structures* Ed.: Richard T. Oehrle et al., D. Reidel Publishing Company, 1988.

63.
Stottler, R., Henke, A., King, J.: "Rapid Retrieval Algorithms for Case-Based Reasoning", en: *IJCAI 89*, pp: 233-237, 1989.

64.
Teil, G.: "CANDIDE: Software for Mapping Lexical Networks based on Large Amounts of Data". Ed.: Centre de Sociologie de l'Innovation, París, 1990.

65.
Thomason, R.H.: "A model Theory for Propositional Attitudes", en: *Linguistics and Philosophy 4*, págs: 47-70, 1980.

66.
Tomita, M. y Carbonell, J.: "The universal parser architecture for knowledge based machine translation", en: *IJCAI 87*, Milano, 1987.

67.
Tomita, M.: "The generalized LR parser/compilae V8-4: A compiler package for practical NL projects", en *COLING 90*, Helsinki 1990.

68.
Turner, R.: "Montague Semantics, Nominalization and Scott's Domains", en *Linguistics and Philosophy 6*, págs:259-288, 1983.

69.
Tyler, L. y Marslen-Wilson, W.: "Speech comprenhension processes." En Mehler, Walker, y Garrett 1982.

70.
Van Leeuwen, J.: "Formal Models and Semantics", Ed.: Elsevier, Amsterdam, 1990.

71.
Woods, W.: "Semantics for a question-answering machine.". Ed.: Garland Publishing, 1967.