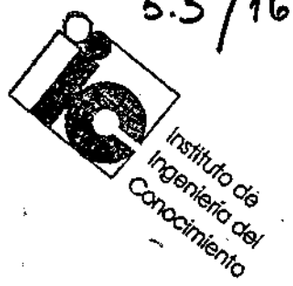

Fundamentación de un sistema para la resolución automática de problemas

TESIS DOCTORAL

AUTOR: Francisco Saiz López
DIRECTOR: Roberto Moriyón Salomón

Marzo 1994

Departamento de Ingeniería Informática

Memoria presentada para optar al grado de Doctor en Ingeniería Informática

Agradecimientos

Quiero expresar mi agradedimiento a Roberto Moriyón por el trabajo que ha dedicado en la dirección de esta tesis, y sobre todo por la entrega y empuje que me ha demostrado siempre. Así mismo, agradezco a Pablo Castells el compañerismo y ayuda que durante estos años dedicados al proyecto PRÓGENES he recibido de él. Igualmente expreso mi afecto a Javier Contreras por haberme introducido en estos temas, a José Antonio Castro por lo compartido en el Departamento de Matemáticas, a Julio Gonzalo, Joaquín Regodón, Julia Díaz y Pilar Rodríguez, que durante estos pasados años han participado conmigo en PRÓGENES y a todos los demás compañeros del IIC. Por último, quisiera resaltar el apoyo que he recibido de Merce que me ha animado en todo momento, así como el de mi familia.

Agradezco también al Instituto de Ingeniería del Conocimiento y al Departamento de Matemáticas la disposición de medios técnicos que han puesto a mi alcance para llevar a cabo este trabajo, y sin la cual no hubiera sido posible concluir esta tesis.

Contenido

Introducción

Esta tesis describe la fundamentación de un sistema, PRÓGENES, para la resolución automática de problemas científicos a un nivel de un primer curso de Universidad. Dichos fundamentos utilizan técnicas basadas en el paradigma de manipulación simbólica que provienen de dominios como Sistemas Expertos o Demostración Automática de Teoremas. Estas técnicas se pueden englobar dentro del marco de la Inteligencia Artificial.

Las Matemáticas son un dominio típico donde el hombre utiliza especialmente sus habilidades deductivas y, desde los inicios de la Inteligencia Artificial, ha sido un campo de experimentación para intentar resolver problemas crecientes en dificultad. A lo largo de la tesis se describe la resolución de diversos problemas de Matemáticas, así como algunos otros de Física, y no sería difícil adaptar las técnicas desarrolladas a otros dominios. Ello demuestra la generalidad de estas técnicas, que tienen interés desde el punto de vista cognitivo en general.

La principal aportación de este trabajo en el dominio de Resolución Automática de Problemas se refiere a la utilización combinada de métodos de naturaleza deductiva con métodos de cálculo, desarrollando una mayor potencia de resolución respecto a otros sistemas. Además, los mecanismos utilizados permiten la extensibilidad en la definición de conceptos y en la construcción de métodos de solución, permitiendo acumular conocimiento sobre áreas diversas.

Entre las heurísticas descritas destaca el razonamiento basado en el objetivo, que tiene un amplio espectro de aplicaciones. Esta heurística se aplica a problemas en los que se busca un objeto o se intenta demostrar su existencia. Consiste en suponer que dicho objeto existe y razonar a partir de él obteniendo condiciones necesarias para su existencia hasta determinarlo completamente comprobando a continuación que el objeto así hallado verifica las propiedades deseadas. También se ilustran aplicaciones de esta heurística, como es el algoritmo de expansión que hace razonamiento basado en el objetivo utilizando la estructura de los objetos que aparecen en un problema. El razonamiento basado en el objetivo ha sido utilizado con finalidades pedagógicas

por conocidos matemáticos, [Polya 65]. La incorporación del mismo en un contexto computacional y de Inteligencia Artificial es una aportación original de PRÓGENES.

El control del proceso de deducción mediante la asignación de prioridades a las diferentes reglas y tareas es esencial para la eficiencia de un sistema basado en conocimiento. En constraste con los mecanismos de control en Sistemas Expertos, que a menudo consisten en una simple asignación de prioridades a reglas individuales, PRÓGENES asigna preferencias a tareas y tipos genéricos de reglas, de modo que la aplicación de reglas de reescritura tiene prioridad máxima y el razonamiento en modo *forwards* [1] tiene prioridad sobre el razonamiento basado en el objetivo, que a su vez la tiene sobre el razonamiento en modo *backwards*.

El modelo de conocimiento utilizado en PRÓGENES pretende emular el comportamiento humano en Resolución de Problemas. De hecho una tendencia en el campo de Demostración Automática de Teoremas ha sido el desarrollo de métodos de deducción natural. Una razón fundamental acerca de esta elección es permitir la interacción entre el sistema demostrador y el hombre.

La descripción en PRÓGENES de los conceptos que aparecen en un problema se hace en unas Bases de Conocimiento. Estas describen un Lenguaje Formal en el que se enuncian los problemas, y contienen teoremas relacionados con el dominio. Así mismo definen una jerarquía de tipos asociada al conjunto de conceptos y un mecanismo que asocia un tipo a cada expresión del Lenguaje Formal. Las heurísticas generales utilizan las Bases de Conocimiento para llevar a cabo una búsqueda de la solución. Para ello se dispone de un motor de inferencia y de un intérprete para el Lenguaje Formal.

Cabe observar que el contenido de este trabajo se complementa con la tesis doctoral *Heurísticas y metaconocimiento en resolución automática de problemas*, de Pablo Castells, [Castells 94]. En esta tesis se ilustra la fundamentación del sistema, mientras que P. Castells ilustra técnicas más específicas del metaconocimiento usado en PRÓGENES. Ambas son el resultado del trabajo de equipo en el proyecto PRÓGENES, (PROof GENerator Expert System), coordinado por Roberto Moriyón, y 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 areas comunes. El sistema PRÓGENES ha sido descrito en [Castells et al. 91, a] [Castells et al. 92] [Castells, Moriyón, Saiz 93, a] [Castells, Moriyón, Saiz 94], aunque muchos aspectos estan más desarrollados en las dos tesis. En cuanto al estado de desarrollo de la implementación, PRÓGENES está en

[1] En todo el trabajo se utiliza la terminología inglesa para conceptos clásicos en Inteligencia Artificial, como ocurre en este caso.

su fase final, por lo que algunas de las técnicas que se describen aquí no están aún plenamente implementadas, pero estan diseñadas a un suficiente nivel de detalle como para que completar su implementación no plantee excesivos problemas.

Como referencia a lo largo de la tesis, se utilizan los siguientes problemas:

1. Demostrar que si se tiene $f, g : \mathbb{R} \to \mathbb{R}$, donde f es derivable, g es un difeomorfismo y x es un punto crítico de f, entonces $g(x)$ es un punto crítico de $g \circ f \circ g^{-1}$.

2. Demostrar que la funcion $f(x) = x^3 - 6x^2 + 15x - 2$ es creciente.

3. Demostrar que si $f, g : \mathbb{R} \to \mathbb{R}$, f es inyectiva y $g(x) = x^3 + 3x^2 + 3x + 2$, entonces $g \circ f$ es inyectiva.

4. Hallar la velocidad inicial de un proyectil que alcanza una altura máxima de 2 metros cuando su proyección sobre la horizontal está a una distancia de 1 metro del lugar del disparo.

5. Hallar la superficie de revolución alrededor del eje x cuya sección con el plano OYZ tiene área π, y el área de corte con cualquier otro plano paralelo a él es igual al volumen del sólido limitado por dicho plano y la superficie de revolución, en la dirección negativa del eje x.

6. Hallar el centro de la circunferencia que pasa por los puntos $(1,1)$, $(0,2)$ y $(\frac{1}{2}, \frac{\sqrt{3}+4}{2})$.

7. Encontrar el área de la circunferencia inscrita en el triángulo de vértices $(-1,0)$, $(0,2)$ y $(1,0)$.

8. Hallar el segmento cuyo punto medio es el punto $(0,1)$, tiene longitud 2 y es perpendicular al vector $(1,0)$.

9. Encontrar el plano que contiene a los puntos $(4,0,0)$ y $(0,8,0)$ y es tangente al elipsoide centrado en el origen cuyos ejes son paralelos a los ejes de coordenadas con longitudes 1, 2 y 1.

10. Sea un sistema mecánico unidimensional constituido por una partícula de masa 2 y carga 1, y por un campo de fuerzas formado a partir de las siguientes interacciones: un campo eléctrico de valor 5, el campo gravitatorio terrestre, una fuerza de fricción de coeficiente 4 y una fuerza elástica de constante de recuperación 3. Hallar la posición de la partícula en el instante 1 sabiendo que su posición y velocidad iniciales son nulas.

A lo largo de los siguientes capítulos se describen en detalle cual es el estado de antecedentes (capítulo 1), la descripción de las Bases de Conocimiento y del Lenguaje Formal (capítulo 2), el sistema deductivo en PRÓGENES y la heurística de razonamiento basado en el objetivo (capítulo 3), heurísticas específicas de resolución, como el algoritmo de expansion (capítulo 4) y unas conclusiones finales (capítulo 5). Al final también se incluye un Apéndice con Bases de Conocimiento y una Tabla de Símbolos utilizados a lo largo de la tesis.

Capítulo 1

Antecedentes

Quizá los primeros antecedentes relevantes en Deducción Automática haya que buscarlos en el mismo desarrollo de la Lógica de los últimos siglos, y especialmente a partir de los trabajos del siglo actual de J. Herbrand, [Herbrand 71], y K. Gödel, [Gödel 30]. La Lógica siempre ha sido el lenguaje de expresión de las Matemáticas, mediante el que se pueden expresar todos los conceptos. En este contexto, la innovación tecnológica incorporada mediante el tratamiento automático de la información ha venido a reforzar aquellos primeros esfuerzos. Así mismo, la utilización de lenguajes de programación declarativos y de sistemas de manipulación simbólica han permitido incorporar otras nuevas técnicas de tratamiento automático de objetos y reglas.

A continuación se examinan puntualmente cuales han sido los avances más significativos incorporados en distintos sistemas de Demostración y Resolución desde la perspectiva de su relevancia para esta tesis, especificando sus características fundamentales en cuanto a similitud con el proceso deductivo humano, su situación en marcos deductivos o procedurales, eficiencia, etc. Así mismo, se acompañan ejemplos que ilustran su comportamiento y que en capítulos sucesivos son de nuevo abordados desde la perspectiva de PRÓGENES. Igualmente se establecen marcos de carácter general acerca de lenguajes de programación declarativos, sistemas expertos, etc, que configuran el contexto de la Resolución Automática de Problemas.

1.1 Lenguajes de programación declarativos

El conocimiento acerca de un determinado dominio puede ser representado aceptablemente mediante estructuras como grafos. En este sentido resultan destacables las re-

presentaciones mediante redes semánticas de Sowa, [Sowa 84]. Así mismo, se establecen reglas de deducción en este tipo de estructuras para obtener nuevos grafos. Además, el conocimiento no se mantiene estático, sino que varía dinámicamente, y la forma en que varía se determina mediante metaconocimiento, que se expresa mediante nuevas reglas o funciones que crean grafos en base a otros grafos. Este conocimiento se puede representar en una lógica de predicados, la cual se formula de manera natural en forma de árboles, que son una clase específica de grafos. Igualmente, los *frames* son sistemas que representan el conocimiento en forma de árboles, [Minsky 75].

Los lenguajes de programación que permiten utilizar árboles como estructura de datos y que incorporan de manera natural algoritmos que los manejan son mas útiles para la representación del conocimiento y decimos que tienen un carácter declarativo.

Existen dos tipos de paradigmas declarativos dentro de la programación. Uno es el paradigma de la programación funcional, en donde la estructura de datos fundamental se representa mediante árboles. Estos, a su vez, pueden ser evaluados para producir otros datos. Lenguajes de programación funcional son LISP, [Winston, Horn 89], ML, [Harper, MacQueen, Milner 86], o MATHEMATICA, [Wolfram 91]. El diseño de LISP estuvo influenciado por el lambda-cálculo. El lambda-cálculo, [Church 51], permite definir funciones y evaluar el resultado al sustituir argumentos y, en general, fundamenta la teoría de funciones computables y recursivas.

El otro paradigma es el de la programación lógica, en donde la estructura de datos fundamental representa fórmulas en una lógica de predicados, existiendo un sistema de deducción en esta lógica. Un ejemplo es PROLOG, [Roussel 75]. PROLOG es un lenguaje que utiliza unificación y resolución sobre cláusulas de Horn para establecer deducción. Posee un motor de inferencia que utiliza deducción *backwards*. En este contexto, cabe destacar a MECHO, [Bundy 1979], que es un sistema implementado en PROLOG y que resuelve problemas de Mecánica elemental a partir de una descripción de objetos y leyes que rigen su comportamiento. Por otra parte, cabe señalar que dentro de los paradigmas de la programación funcional y lógica juegan un papel análogo los procesos de evaluación y deducción en donde se parte de estructuras de datos iniciales y se obtienen otros finales como resultado de tales procesos.

Realmente cualquier lenguaje de programación declarativa permite abordar estos dos tipos de paradigmas, aunque no los resuelve simultáneamente de manera natural y satisfactoria. Recientemente, algunos lenguajes como Babel, [Moreno, Rodríguez 88], permiten no privilegiar ninguno de estos paradigmas, estableciéndose esencialmente como lenguajes de programación declarativos.

Los lenguajes de programación declarativa permiten construir sistemas de mani-

pulación simbólica con facilidades algebraicas. En adelante se describen los aspectos esenciales relacionados con este tipo de tratamiento.

1.1.1 Sistemas de manipulación simbólica

Un sistema general de manipulación simbólica utiliza expresiones y patrones sobre los que se definen reglas de reescritura. Se construye un sistema de evaluación con un determinado criterio de parada, transformando un conjunto de expresiones según un determinado modelo. A cada patrón se le puede asignar un tipo y utilizar esta asociación en el proceso de evaluación.

MATHEMATICA es un sistema característico de manipulación simbólica especialmente adaptado para utilizar expresiones matemáticas. MATHEMATICA es un sistema de *software* para ordenador que incluye un lenguaje para aplicaciones matemáticas. Se puede utilizar como calculador simbólico y numérico, como lenguaje de programación de alto nivel, como sistema de representación del conocimiento para campos científicos, y también como un sistema de visualización para funciones y datos, [Wolfram 91].

MATHEMATICA utiliza conocimiento procedural y calculo simbólico, por ejemplo, en la resolución de ecuaciones en un conjunto de variables obteniendo soluciones tales como:

```
Solve[{x^2 + 2y == 3, y - x == -1}, {x,y}] :=

       -2 + Sqrt[24]        -4 + Sqrt[24]
{{x -> -------------, y -> -------------},
            2                    2

       -2 - Sqrt[24]        -4 - Sqrt[24]
 {x -> -------------, y -> -------------}}
            2                    2
```

También permite resolver un sistema de ecuaciones diferenciales como:

```
DSolve[{x'[t]== x[t] - y[t], y'[t] == x[t] + y[t]}, {x[t],y[t]}, t] :=

          (1 - I) t          2 I t                      2 I t
         E          (C[1] + E      C[1] - I C[2] + I E      C[2])
{{x[t] -> ----------------------------------------------------------,
                                  2

          (1 - I) t              2 I t                  2 I t
         E          (I C[1] - I E      C[1] + C[2] + E      C[2])
 {y[t] -> ----------------------------------------------------------}}
                                  2
```

Utiliza un lenguaje en donde cada expresión esta formada por una cabeza o constructor y por un conjunto de componentes o *slots*. En este lenguaje se pueden representar objetos y relaciones. Así, por ejemplo, se puede considerar la expresión MATHEMATICA `segmento[extremo1,extremo2]` cuyo constructor es `segmento` y tiene como componentes dos extremos. MATHEMATICA contiene un sistema de asignación de tipos a expresiones. En el caso anterior se asocia a aquella expresión el tipo segmento. El sistema de asociación de tipos de MATHEMATICA asigna como tipo a una expresión el constructor utilizado.

Así mismo, MATHEMATICA también permite la definición de reglas de reescritura aplicables a determinados patrones, con o sin restricciones de tipos. En el proceso de elaboración de reglas de reescritura, el conocimiento procedural se describe mediante técnicas de transformación para realizar cálculos. Así, por ejemplo, las técnicas de integración de funciones se pueden construir mediante este tipo de estrategias. En las demostraciones de la mayoría de los teoremas de Matemáticas se utilizan este tipo de procesos aplicando técnicas de manipulación simbólica a las que se aplican paralelamente procesos deductivos.

Estas reglas de reescritura elaboran un sistema de evaluación infinito de modo que a partir de una expresión de MATHEMATICA se obtiene otra expresión cuya evaluación es ella misma. Sin embargo, existen medios de controlar la evaluación de una expresión mediante la asignación de atributos a los argumentos de alguna de estas funciones.

Sin embargo, la aplicación indiscriminada de reglas de reescritura, que utilizan conocimiento procedural, no permite establecer procesos deductivos en donde se utilicen proclamaciones declarativas acerca de conocimiento deductivo de propiedades y teoremas. Por tanto, los sistemas de manipulación simbólica carecen de facilidades en el tratamiento de problemas en donde se necesitan procesos de deducción que comple-

menten las consideraciones simbólicas.

ANALYTICA, [Clarke 93], es un sistema reciente desarrollado en la línea de integrar facilidades simbólicas y deductivas. Este sistema es capaz de establecer deducciones para resolver problemas analíticos usando reglas de MATHEMATICA. De todos modos, existe aún un hueco entre lo que permite ANALYTICA y lo que un estudiante aprende en un curso avanzado de Cálculo o Geometría. La principal dificultad reside en el hecho de que algunos problemas se pueden resolver por cálculo directo, mientras que otros necesitan de manipulaciones previas y otros suponen razonamiento conceptual. Aunque el conocimiento incorporado por ANALYTICA es suficiente para resolver ciertas clases de problemas, se necesita otro tipo de tratamiento que tenga más en cuenta las actividades cognitivas en estos tipos de dominios.

1.2 Sistemas Expertos

Hacia principios de los años setenta fue desarrollándose paralelamente en el campo de la Inteligencia Artificial toda una corriente de los denominados Sistemas Expertos, [Nilsson 80]. Estos sistemas de *software* aplican en varias disciplinas, como Economía, Química, Medicina, etc, unos principios basados sobre todo en la utilización de técnicas del conocimiento y en el uso de reglas de producción.

Una regla de producción está formada por un conjunto de condiciones a verificar y por acciones a ejecutar en tal caso. Construida una base de reglas de producción, éstas se aplican a una base de hechos obteniéndose resultados adicionales. El proceso de elaboración de la base de reglas de producción necesita de unos conocimientos teóricos acerca del campo de estudio. Un experto en cada dominio suministra, por tanto, estos conocimientos. A su vez, otra persona se ocupa de construir un sistema experto que resuelva el problema en cuestión. Esta persona es el ingeniero del conocimiento que configura una base de conocimiento formada por una base de reglas de producción y una base de hechos.

En un Sistema Experto existe también un motor de inferencia que utiliza el conjunto de reglas de producción para establecer deducción a partir de una base de hechos. Lenguajes de programación declarativa como LISP o PROLOG resultan particularmente útiles en la construcción de motores de inferencia. Las reglas de producción, en estos contextos, se expresan mediante listas o expresiones declarativas que el motor de inferencia utiliza como patrones en el proceso deductivo.

Un motor de inferencia utiliza una serie de heurísticas en la búsqueda de soluciones

en el espacio de estados. En un sentido general, es posible un razonamiento *forwards* o *backwards*. En un proceso *forwards* se privilegia la dirección deductiva desde las hipótesis hacia las conclusiones, mientras que en uno *backwards* se utiliza la opción contraria. La utilización del conjunto de reglas de producción puede ser controlada mediante un sistema de reglas, denominadas metarreglas, que aplican conocimiento a un nivel superior. El sistema de metarreglas aplica metaconocimiento a la búsqueda en el espacio de estados. Son reglas cuyas condiciones se refieren a características específicas de otras reglas o cuyas acciones activan otras reglas, [Pitrat 90].

La Demostración y Resolución Automática ha utilizado técnicas de los Sistemas Expertos. En este sentido, en algunos sistemas se utiliza un motor de inferencia y se representa conocimiento deductivo en forma de reglas de producción. PRÓGENES contiene un motor de inferencia *forwards* y expresa teoremas mediante reglas de producción. Sin embargo, no sólo utiliza este tipo de razonamiento mediante reglas de producción. Problemas aritméticos elementales resultan difíciles de abordar desde este punto de vista. Se necesitan un elevado numero de reglas para establecer transformaciones en ecuaciones. Además no existe una clara direccionalidad en la aplicación de tales reglas produciéndose ciclos infinitos a causa de esta ambigüedad.

1.3 Principio de Resolución de Robinson

En 1965 apareció el Principio de Resolución de Robinson, [Robinson 65], estableciendo uno de los avances más importantes en Demostración Automática. Supuso un método innovador de tratar el cálculo de predicados, además de configurar un sistema deductivo completo, en el sentido de que si una fórmula es decicible, entonces es capaz de deducirlo en un tiempo finito.

El Principio de Resolución utiliza reducción al absurdo para la demostración de un determinado teorema. Para ello, se suponen ciertos los axiomas y la negación de la conclusión, se utiliza un proceso de reducción de fórmulas mediante la eliminación de conectivos como $\leftarrow$, $\leftrightarrow$, $\forall$ o $\exists$ y se consideran transformaciones de resolución. Estas transformaciones son procesos mediante los cuales se infieren nuevas fórmulas. El objetivo final es encontrar una fórmula contradictoria sencilla en la que aparezca expresada una afirmación y su negación.

La transformación de resolución a partir de dos cláusulas disyuntivas como $P \vee Q$ y $\neg P \vee R$ obtiene la cláusula $Q \vee R$. A partir de un conjunto general de cláusulas se considera resolución iterativa entre pares de cláusulas hasta obtener la cláusula vacía que es una contradicción. Desde un punto de vista deductivo se puede considerar

la demostración por resolución como un proceso de deducción *forwards* a partir de una base de claúsulas disyuntivas que se interpretan como reglas de todas las formas posibles. Asi, cada claúsula en el ejemplo anterior daría lugar a dos reglas como sigue:

$$\begin{array}{rcl} P \vee Q & : & (1)\ \neg P \rightarrow Q \quad (2)\ \neg Q \rightarrow P \\ \neg P \vee R & : & (3)\ P \rightarrow R \quad (4)\ \neg R \rightarrow \neg P \end{array}$$

Aplicando la regla (3) a la cláusula $P \vee Q$ se obtiene $R \vee Q$. Otra regla aplicable es (1) a $\neg P \vee R$ para obtener así mismo $Q \vee R$. De este modo aparece un nuevo hecho, $R \vee Q$, que aumenta la base de hechos que, a su vez, produce nuevas reglas. Sin embargo, el número de reglas que aparecen tiene un comportamiento exponencial.

Se ha conseguido demostrar que este sistema es completo, [Davis, Weyuker 83], para el cálculo de predicados de primer orden, pero el número de combinaciones entre cláusulas se hace tremendamente elevado produciéndose una explosión exponencial y, por consiguiente, necesitándose tiempos demasiado elevados para obtener la solución, a pesar de que el método es completo y se debe obtener la solución en tiempo finito para fórmulas decidibles.

Por otra parte, lenguajes de programación como PROLOG utilizan el Principio de Resolución en el proceso deductivo. Se trata por tanto de un método deductivo, pero realmente bastante innatural. Esta innaturalidad radica en el método utilizado que es básicamente la reducción al absurdo. A veces la reducción al absurdo es natural, pero se utiliza en casos muy especiales. La mayoría de los teoremas y problemas resueltos en Matemáticas se demuestran por otros métodos y son muy pocos los casos, quizás los irremediables, en los que se intenta demostrar un teorema por reducción al absurdo. Por otra parte, si se quiere construir un sistema de resolución de problemas, entonces la actitud constructivista debe tener un grado más intenso, precisamente para obtener soluciones concretas.

1.4 Heurísticas de Bledsoe en Deducción Natural

En Deducción Natural se utilizan heurísticas que emulan el comportamiento humano en procesos de inferencia. Una primera aproximación en este contexto fue el cálculo de *sequents*. El cálculo de *sequents* fue desarrollado hacia 1960 por H.Wang, [Wang 60]. Cada fórmula se representa mediante un *sequent* $A_1, \cdots, A_n \vdash B$ que significa que B se deduce de las hipótesis $\{A_1, \cdots, A_n\}$, y se establecen reglas inferencia entre *sequents* que emulan un proceso de deducción natural en el que se sustituye el objetivo de una

demostración por otro más sencillo. Así, por ejemplo, en un caso sencillo la regla de inferencia:

$$\frac{A \rightarrow B, H \vdash A}{A \rightarrow B, H \vdash B}$$

sustituye el objetivo de demostrar B a partir de H y $A \rightarrow B$, por el de demostrar A a partir de H. En este contexto, cada teorema se representa mediante un conjunto de *sequents* a modificar mediante adecuadas reglas hasta concluirlo.

Hacia finales de los años sesenta y principios de los setenta, se utilizaron las técnicas de resolución combinándolas con métodos de Deducción Natural en la construcción de demostradores. Así Bledsoe, [Bledsoe 71], utilizo resolución junto con el cálculo de *sequents* para construir un sistema de demostración. En primer lugar, utilizaba unas rutinas de reducción, expresadas mediante reglas de reescritura. Así, por ejemplo, las reglas más naturales del cálculo de *sequents* como:

$$\frac{H, P \vdash Q}{H \vdash P \rightarrow Q}$$

$$\frac{H \vdash P \quad H \vdash Q}{H \vdash P \wedge Q}$$

transforman el enunciado de un teorema en una forma canónica. Por otra parte, utilizaba otras reglas de reescritura para predicados. Y al final, después de todas estas simplificaciones, usaba resolución. Se experimentó en teoría de conjuntos y los resultados mejoraban los obtenidos únicamente por resolución.

Más adelante, Bledsoe obtuvo una nueva versión de su sistema, [Bledsoe 72], en la que sustituyó el método de resolución por un procedimiento en donde utilizaba un sistema de reglas privilegiando la implicación y haciendo una clara separación entre hipótesis y conclusión del teorema. Consideró demostraciones sobre propiedades del cálculo de límites, tales como la linealidad y su versión análoga para el producto, utilizando metaconocimiento en este área para establecer una heurística de límites.

En su heurística de límites, Bledsoe utilizó algunas técnicas específicas, como una representación de tipos para desigualdades entre números reales, de modo que ciertos predicados son transformados en asignaciones de tipo para variables. En este contexto, el programa no recibe todos los axiomas de una relación de orden total, sino que manipula directamente las asignaciones de tipo. Así mismo, también incorporó cálculos

sobre esta estructura de tipos en contextos como resolución de inecuaciones, determinando el tipo de alguna variable, o de ecuaciones despejando una incógnita en función de las demás. Su heurística de límites se adapta especialmente a demostraciones sobre propiedades del cálculo de límites. En esta heurística se utilizan reglas de reescritura para obtener una serie de acotaciones en función de las hipótesis.

En su siguiente versión, [Bledsoe, Bruell 74], aparece un nuevo tratamiento en el que se permite la interacción humana. Es fundamental para permitir este tipo de interacción que la formulación del teorema se presente de una forma natural para el usuario, para que éste pueda guiar al demostrador en el espacio de búsqueda. En este contexto, las heurísticas de Demostración Natural permiten este tipo de interacción. Este sistema llegó a demostrar difíciles teoremas de Topología Conjunstista.

Quizás una de las principales aportaciones de Bledsoe en el campo de la Demostración Automática de Teoremas fue el abandono a que fue sometiendo a los métodos de resolución retomando métodos de deducción natural frente a los primeros. En [Bledsoe 77] subraya este hecho junto con profundas reflexiones acerca de los retos humanos en este dominio de la Inteligencia Artificial. Según Bledsoe, resultan bastante comunes en todos los sistemas de Demostración Automática tanto los rápidos avances en los primeros instantes, como la aparición de obstáculos a un ritmo creciente. En [Bledsoe 77], Bledsoe propuso una serie de retos en ramas del Análisis Matemático, tales como las demostraciones de los teoremas de Rolle o Bolzano, que probablemente necesitarán de espectaculares avances para ser abordados.

1.5 Demostrador de Boyer y Moore

Hacia finales de los años setenta, y profundamente influenciado por los trabajos de Bledsoe, apareció el demostrador de Boyer y Moore. En su Lógica Computacional, [Boyer, Moore 79], se describe la arquitectura de un demostrador en donde se pueden definir recursivamente objetos y funciones y donde aparecen demostraciones por inducción. La Lógica Computacional de Boyer y Moore utiliza una serie de axiomas y reglas de inferencia que se obtienen a partir del cálculo proposicional con igualdad. La utilización de la igualdad se hace mediante reglas de reescritura. Estas reglas permiten obtener nuevas expresiones que reinterpretan objetos o expresiones de tipo booleano.

El comportamiento del demostrador está determinado por una base de reglas de inferencia. Esta base es construida por el sistema a partir de los axiomas, definiciones y teoremas dados por el usuario. Además, el sistema convierte los teoremas demostrados en reglas de inferencia que se guardan en la base de reglas.

La sintaxis que utiliza el sistema para construir el lenguaje formal es esencialmente análoga a las construcciones que se hacen en la programación funcional de LISP. El lenguaje contiene variables y nombres de funciones combinados en una notación prefija. Las constantes de la lógica son funciones sin argumentos. La lógica de Boyer y Moore es una lógica de primer orden con igualdad libre de cuantificación. Por tanto, se admiten variables libres en el lenguaje formal que se interpreta que están cuantificadas universalmente, mientras que las variables cuantificadas existencialmente se representan mediante constantes.

El demostrador de Boyer y Moore incluye comandos que permiten definir objetos y funciones. La definición funcional se establece mediante una igualdad con un código que configura unidireccionalmente una regla de reescritura. Así mismo, también se puede establecer una igualdad bidireccionalmente considerándola como una doble implicación e incluyéndola dentro de la Lógica. Este tipo de comandos también permite establecer interactivamente ciertos subteoremas a demostrar durante el proceso de demostración.

Las reglas de inferencia en esta Lógica Computacional son fundamentalmente cálculo proposicional, principio de inducción e instanciación. Por tanto, existe un motor de inferencia que utiliza la base de reglas para establecer las demostraciones. Así mismo, el demostrador proporciona una facilidad de suministrar interactivamente indicaciones desactivando ciertas definiciones, sugiriendo la utilización de determinados lemas, etc.

Por tanto, el sistema posee una sintaxis y unos principios para la construcción de datos, así como unas reglas de reescritura de los términos de esta Lógica. Junto a todos ellos, existen una serie de mecanismos de transformación aplicables a la representación de un determinado teorema y que conducen una demostración. Estas transformaciones se aplican recursivamente al enunciado de un teorema, colocado en una pila sobre la que se organiza el proceso, hasta obtener su demostración. Las principales transformaciones que se aplican, en orden de ejecución, son:

- Simplificación. Consiste en la aplicación de una serie de procedimientos de reducción y reescritura para obtener una expresión canónica. Estas técnicas son análogas a las empleadas por Bledsoe en sus heurísticas de Deducción Natural.
- Representación canónica de objetos.
- Eliminación de hipótesis de igualdad. Se produce una conversión en reglas de reescritura o en reglas deductivas.
- Generalización. Se establece un objetivo más general.
- Eliminación de hipótesis irrelevantes.

- Inducción. Se utiliza este método de demostración.

Los más destacados resultados teóricos obtenidos fueron las demostraciones de la existencia y unicidad de la factorización de primos, la ley de Gauss de reciprocidad cuadrática y la verificación interactiva del teorema de incompletitud de Gödel. Además este demostrador fue paulatinamente mejorado durante los años setenta hasta aparecer una nueva versión en 1988, [Boyer, Moore 88], en la que aparecían como novedades fundamentales la integración de un proceso de decisión aritmético-lineal y la mejora de facilidades interactivas. Otros campos de aplicación del demostrador de Boyer y Moore son la comprobación de *software* y *hardware*.

1.6 Editores de demostraciones

En la corriente de los sistemas de Deducción Natural se sitúa un conjunto de demostradores que permiten ediciones interactivas de los procesos de demostración. En este contexto se incriben los de la familia LCF (Lógica de Funciones Computables).

Los sistemas LCF se desarrollaron a partir de una lógica de funciones computables desarrollada por Scott en 1969, [Scott 70], en la que utiliza Teoría de Dominios. En la Teoría de Dominios se utiliza lambda-cálculo de tipos, desarrollando una semántica simple donde se puede identificar cada tipo con un conjunto y cada lambda-expresión como un función entre dos tipos (o conjuntos). En esta lógica de funciones computables se utiliza una teoría de tipos próxima a los planteamientos iniciales de Church, [Church 40]. La Teoría de Dominios se desarrolla en torno a campos de las Matemáticas como Topología o Teoría de Categorías.

Otra característica común a los sistemas LCF es la utilización de un metalenguaje, ML, que permite la interacción del usuario con el demostrador. Así mismo, el método de inferencia que utilizan es el cálculo de *sequents* y, en general, axiomas y reglas basados en el intuicionismo y constructivismo dentro del marco de la Deducción Natural. Además, el demostrador asegura coherencia. Esta es una característica común de los sistemas basados en Deducción Natural. No se consigue completitud en el sistema de demostración, como asegura por ejemplo el método de resolución de Robinson, pero se evitan inconsistencias durante el proceso de resolución.

Ediburgh LCF, [Gordon, Milner, Wadsworth 79], fue el primer sistema de esta familia, a partir del cual se fueron configurando nuevas variantes. En este sentido, Cambridge LCF, [Paulson 87], extendió la lógica de Edinburgh, LCF, con el tratamiento de más cláusulas lógicas, mejorando la eficiencia y añadiendo nuevos mecanismos de

inferencia. Se utilizó sobre todo en demostraciones sobre programación funcional. En esta misma corriente HOL, [Gordon 87], utiliza una lógica de orden superior y se ha utilizado principalmente en la comprobación de circuitos digitales y de demostraciones formales. NUPRL, [Constable et al. 86], es el más sofisticado de todos estos sistemas, permitiendo grandes facilidades de edición de demostraciones y consiguiendo una lógica más intrínsecamente constructivista que los demás.

Otro sistema editor de demostraciones que utiliza mecanismos de inferencia análogos a los de LCF es WATSON. Este sistema se desarrolló a partir de los trabajos de Francisco Corella [Corella 90], en donde se mecaniza una Teoría de Conjuntos. En estos trabajos incluye al sistema WATSON como un prototipo de sistema de demostración basado en la Teoría de Conjuntos de Zermelo-Fraenkel, [Jech 78], pero basado en una lógica de orden superior. En este sistema se tiene un sistema de traducción desde conceptos en Teoría de Conjuntos a expresiones formales en la lógica de orden superior para poder ser utilizadas mediante mecanismos de inferencia análogos a los utilizados por la familia LCF. Existe un proceso de conversión desde demostraciones de fórmulas conjuntistas en lógica de orden superior a demostraciones de Teoría de Conjuntos en lógica de primer orden que permite este tipo de traducción.

1.7 Muscadet

MUSCADET, [Pastre 89], es un demostrador de teoremas que funciona en campos de las Matemáticas que requieren el uso de metaconocimiento y de heurísticas de conocimiento profundo. Considera técnicas de Sistemas Expertos utilizando reglas de producción y un motor de inferencia. Estas técnicas se adecuan especialmente al tratamiento de dominios como Matemáticas. El metaconocimiento sobre este dominio se describe en forma de reglas más profundas denominadas metarreglas. El motor de inferencia utiliza una serie de reglas activas para llevar a cabo deducción. Por otro lado, contiene una base de conocimiento que describe conceptos matemáticos, estrategias de demostración y metaconocimiento para manipular expresiones formales.

Este demostrador se sitúa en la confluencia de las corrientes descritas hasta este punto, tales como Demostración Natural o Sistemas Expertos. En adelante se describen sus principales características que integran un sistema eminentemente deductivo. Los campos de aplicación se refieren sobre todo a aspectos conceptuales en Matemáticas, tales como Topología, Teoría de Conjuntos, etc.

En MUSCADET la acción del motor de inferencia utiliza la base de conocimiento para configurar una base de hechos y reglas activas que se actualizan dinámicamente durante

el proceso de deducción. La base de hechos contiene los objetos y expresiones booleanas a considerar en cada instante. Por otro lado, la base de reglas activas está constituida por reglas aplicables a la base de hechos. Las reglas activas se seleccionan desde una base general de reglas considerando aquellas que son locales a cada demostración. Así mismo, este proceso de selección de reglas se realiza mediante reglas específicas de activación.

Las heurísticas utilizadas en la demostración de un teorema se expresan, por tanto, en forma de reglas de producción. Cada regla contiene condiciones que expresan propiedades referidas a las bases de hechos y reglas y, así mismo, contiene acciones que pueden incluir, en el caso de las metarreglas, una heurística completa mediante la activación de determinadas reglas. Las acciones de las metarreglas, denominadas metaacciones, emulan el comportamiento de un sistema de Demostración Natural, al estilo de los métodos de Bledsoe, modificando dinámicamente la estructura de objetos y reglas.

El tratamiento de la igualdad en MUSCADET se realiza mediante unas reglas que reinterpretan dinámicamente la base de conocimiento. Así, en las reglas:

```
Regla =1
 Si x=y, Rxz son unas relaciones
 Entonces Ryz es una relacion

Regla =2
 Si x=y, Rzx son unas relaciones
 Entonces Rzy es una relacion
```

si dos objetos son iguales y se conocen ciertas propiedades acerca de uno de ellos, entonces se establecen las mismas para el otro en ambas direcciones según `Rule =1` y `Rule =2`. Las hipótesis existenciales se tratan específicamente construyendo una lista de objetos para evitar ciclos infinitos. Así mismo, existen metarreglas que construyen reglas a partir de definiciones, jerarquizando el orden de las mismas.

MUSCADET se ha experimentado dentro del dominio matemático de la Topología y, en particular, sobre espacios vectoriales topológicos. También se ha utilizado en Teoría de Conjuntos. En este último campo es capaz de resolver problemas como:

- Sea $f : A \rightarrow B$ una funcion inyectiva. Demostrar que $\forall\, X, Y \subset A$ se verifica que $f(X \cap Y) = f(X) \cap f(Y)$.

Para su resolución, se crea una base de hechos que contiene los conocimientos repartidos en módulos de objetos conocidos (OBJETOS), teorema a demostrar (CONCLUSION), hipótesis de las que se parten (HIPOTESIS), relaciones entre los objetos del problema (RELACIONES) y reglas activas en cada estado de la demostración (REGLAS-ACTIVAS). La evolución de la base de hechos viene determinada por la acción de las reglas que existen en REGLAS-ACTIVAS.

La base de conocimiento contiene definiciones de los conceptos que aparecen en el problema, tales como imagen o inyectiva. A partir de estas definiciones se crean automáticamente reglas de distinto carácter en función de datos conocidos o por conocer:

```
Regla IMAGEN-1
  Si W=IMAGEN(F,A) es una hipotesis
     y Y pertenece a W es una relacion
  Entonces existe X en A con Y=IMAGEN(F,X) es una hipotesis

Regla IMAGEN-2
  Si W=IMAGEN(F,A) es una hipotesis
     y X pertenece a A es una relacion
     y Y=IMAGEN(F,X) es una hipotesis
  Entonces Y pertenece a W es una hipotesis
```

También aparecen reglas basadas en Deducción Natural que reestructuran la base de hechos tales como:

```
Regla =>
  Si A->B es la conclusion
  Entonces A es la hipotesis y B es la conclusion
```

Son reglas que aplican metaconocimiento durante la resolución reestructurando datos y objetos para permitir aplicar nuevas heurísticas generales. Estos mecanismos son análogos a las técnicas de Deducción Natural de Bledsoe.

Todas estas reglas se utilizan junto con la base de hechos para irla modificando dinámicamente hasta obtener una CONCLUSION que sea trivialmente verdadera. La manera en que se van colocando reglas en REGLAS-ACTIVAS está controlada mediante la acción de metarreglas que contienen heurísticas de demostración que emulan el metaconocimiento matemático en Teoría de Conjuntos.

Por tanto, se trata de un potente sistema deductivo, aunque posee limitaciones en cuanto a facilidades de cálculo formal y simbólico. En el metaconocimiento humano se reúnen ambos tipos de concepciones. Existe un conocimiento deductivo que utiliza heurísticas generales de resolución, pero también están contenidas facilidades de cálculo formal y simbólico que pertenecen al aspecto más procedural del conocimiento que aplica técnicas de resolución directa.

1.8 Ontic

Hacia finales de los años ochenta apareció este sistema de demostración en el MIT, [McAllester 89]. Se trata de un demostrador que contiene una parte interactiva que lo relaciona con el usuario y que verifica demostraciones matemáticas, e incluso *software*. El lado interactivo lo recupera de los sistemas que anteriormente se describieron, como el de Bledsoe, [Bledsoe, Bruell 74], o LCF. Sin embargo, como veremos en lo que sigue, aunque utiliza métodos de deducción que aparecen en sistemas anteriores, introduce métodos originales como modulación semántica y generalización universal automática.

Este sistema utiliza inferencia orientada a objetos. Para ello desarrolla un mecanismo de deducción *forwards* pero a partir de unos objetos foco. En este sentido, considera hechos que tienen que ver con estos objetos a los que aplica mecanismos de inferencia. Además, el conjunto de objetos pueden ser considerado dentro de una jerarquía dinámica en la que se actualizan propiedades de todos los objetos según sea el estado de la demostración.

Por otro lado, en ONTIC la igualdad se trata desde un punto de vista innovador mediante un mecanismo de cierre de congruencia. Es decir, las reglas de reescritura a que dan lugar las igualdades, que por ejemplo el demostrador de Boyer y Moore trata en una dirección orientada por el usuario, aquí se implementan haciendo un cierre en una estructura de retículo que describe el estado de la demostración, completando dinámicamente la relación entre nodos. De este modo se implementan eficientemente las propiedades de una relación de equivalencia en un conjunto.

Además, existe un lenguaje formal formado por expresiones que pueden ser términos, fórmulas, expresiones funcionales, expresiones de tipos, etc. Existe una gran semejanza entre las expresiones formales y su formulación en lenguaje natural. De hecho, en este sistema se obtiene una relación con un factor cercano a uno entre número de palabras en lenguaje natural y formal. En este contexto se tendrían los siguientes enunciados, en lenguaje natural y formal sobre ONTIC, y utilizando traducción al castellano, sobre un problema acerca de retículos:

- Sea un conjunto parcialmente ordenado P en el que todo subconjunto tiene un supremo, entonces todo subconjunto también tiene un ínfimo.

```
(EN-CONTEXTO ((SEA P CONJUNTO-PARCIALMENTE-ORDENADO)
              (SUPONGAMOS (PARA-TODO (S (SUBCONJUNTO-DE P))
                                     (EXISTE (SUPREMO-DE S P))))
              (SEA H (SUBCONJUNTO-DE P)))
     (SE-CUMPLE (EXISTE (INFIMO-DE H P))))
```

En la expresión formal que representa a este problema aparecen suposiciones sobre objetos que se definen según un determinado contexto. Estas suposiciones se establecen definiendo objetos foco como P o H, y hechos acerca de los mismos que se utilizan en el proceso de deducción. Igualmente, la forma de contruir esta clase de expresiones formales utiliza características análogas a la estructura sintáctica utilizada en lenguaje natural.

La resolución de este problema se realiza estableciendo una red formada por nodos en los que se sitúan los objetos que aparecen y las expresiones booleanas que los relacionan. La componente interactiva del sistema permite para su resolución que se considere el conjunto formado por las cotas inferiores del conjunto. Entonces un método de inferencia *forwards* aplica la hipótesis de partida a este conjunto apareciendo el supremo de dicho conjunto. En un paso siguiente, se derivan hechos a partir de estos nuevos objetos, obteniendo precisamente el objetivo a demostrar de encontrar el ínfimo del conjunto H.

ONTIC crea una copia virtual de cada elemento genérico que aparece en un determinado problema y obtiene los hechos que se deducen a partir de el. Si no se ha hecho ninguna suposición acerca de este elemento, entonces se desencadena un mecanismo de generalización automática, para obtener una nueva expresión booleana con cuantificadores universales. Una vez obtenida esta generalización, se añade el booleano obtenido al conjunto de hechos, produciéndose una modificación dinámica de la red que describe el estado del problema. Esta es la forma en que ONTIC emula la acción de las reglas con carácter más marcadamente declarativo dentro de un sistema experto.

Capítulo 2

Descripción de Prógenes

En este capítulo se describe el marco en que se expresan los conceptos señalados inicialmente en la Introducción. El marco es el Sistema Formal PRÓGENES, [Castells et al. 91, a] [Castells, Moriyón, Saiz 94], que contiene unas Bases de Conocimiento y utiliza un Lenguaje Formal. Este Sistema Formal tiene dos aspectos característicos: uno es su carácter declarativo, definido por las Bases de Conocimiento y el Lenguaje Formal asociado, y otro es su carácter procedural, en el que caben destacar los procesos de traducción y metaevaluación de expresiones formales. En este sentido, el proceso de resolución integra ambos enfoques para establecer transformaciones en los estados parciales de resolución de un problema hasta obtener una solución final.

El campo de aplicación de PRÓGENES, y en particular de las heurísticas desarrolladas en esta tesis, son dominios como Matemáticas, Física y, en general, terrenos científicos similares. Las heurísticas se refieren a métodos de resolución en los que se utiliza deducción junto con manipulación algebraica. Una eficiente integración de deducción y cálculo permite resolver un extenso espectro de problemas en estos dominios. Así mismo, se trata de métodos que se pueden generalizar a otras disciplinas científicas en las que las estructuras de objetos y relaciones son análogas, permitiendo resolver un espectro equivalente de problemas.

Cabe señalar que el Lenguaje Formal PRÓGENES se genera a partir de los enunciados de los problemas en lenguaje natural con formato LaTeX, [Castells et al. 91, a] [Castells et al. 92]. El módulo de procesamiento del lenguaje natural utiliza un formalismo de gramática categorial de unificación, considerando un diccionario implementado en CLOS, (Common Lisp Object System), que permite las facilidades de representación de la Programación Orientada a Objetos.

En este capítulo, en relación con los aspectos declarativos dentro de la estructura

de PRÓGENES se describe, en primer lugar, la estructura de las Bases de Conocimiento utilizadas (2.1). A continuación, se especifica el Lenguaje Formal asociado a dicha estructura (2.2). Por otro lado, traducción y metaevaluación representan el carácter procedural en PRÓGENES. Se describe en el siguiente apartado el proceso de traducción de expresiones formales (2.3), el cual generaliza el proceso clásico de herencia en el contexto del Diseño Orientado a Objetos y, a continuación, se desarrolla el proceso de manipulación simbólica dentro de PRÓGENES, que se lleva a cabo mediante metaevaluación (2.4).

2.1 Bases de Conocimiento

Las Bases de Conocimiento contienen una descripción declarativa sobre objetos y su estructura jerárquica, así como de las relaciones que se pueden establecer entre ellos. PRÓGENES permite extensibilidad en la definición de conceptos de modo que en cada dominio científico especifico de aplicación se puede definir una Base de Conocimiento e integrar varias Bases en una estructura modular conjunta. Los conceptos que se definen en una Base de Conocimiento son:

- Objetos. Se define un objeto como un tipo o clase que verifica determinadas condiciones. A su vez, el conjunto de tipos tiene una estructura jerárquica.
- Metafunciones. Son funciones que, aplicadas a objetos, obtienen nuevos objetos. Tienen asociada una definición o semántica y unas condiciones para poder aplicar dicha semántica.
- Metapredicados. Son funciones que se aplican sobre objetos para obtener un valor booleano (Verdadero o Falso). Se definen mediante una semántica expresada en función de otros predicados más elementales y de unas condiciones de aplicación.
- Teoremas. Son afirmaciones generales en forma declarativa y que son aplicables a objetos que cumplen determinadas restricciones de tipo.

En los siguientes apartados se especifica la manera en que se definen estos conceptos en una Base de Conocimiento. Así mismo, el conjunto de Bases configuran un Lenguaje Formal propio, de modo que, según cada elemento en dicha Base, se pueden construir expresiones formales asociadas. Más adelante, (2.2), se describe este Lenguaje Formal al que dan lugar las Bases de Conocimiento.

2.1.1 Objetos

Un objeto se define en una Base de Conocimiento como un tipo. Así, por ejemplo, se define una circunferencia como el tipo `CIRCUNFERENCIA` descrito en un elemento de las Bases de Conocimiento. Así mismo, existe una estructura jerárquica entre tipos establecida mediante una relación de orden que relaciona determinados pares.

Para cada tipo, siempre existe una única definición *standard* en función de un patrón de construcción. Dependiendo de la forma en que se represente este patrón podemos hablar de objetos y tipos atómicos o compuestos. Es decir, se pueden definir objetos compuestos en función de otros que, a su vez, pueden ser atómicos o compuestos.

Los objetos atómicos están predefinidos mediante un predicado de evaluación directa. Así, por ejemplo, el elemento de las Bases de Conocimiento que define el tipo atómico `ENTERO` tiene la forma:

```
(atom es-entero ENTERO)
```

donde el predicado `es-entero` esta predefinido. De este modo se definen también los tipos `NATURAL`, `RACIONAL` y `COMPLEJO`.

Ejemplos de objetos atómicos son, por tanto, los números complejos, reales, racionales, enteros y naturales, y se utilizan predicados que reconocen directamente este tipo de objetos. Alternativamente, también es posible considerar como atómicos sólo algunas clases de números como los naturales y enteros, y definir los números racionales en función de los enteros, y los complejos en función de los racionales. Sin embargo, en el contexto de resolución de problemas de Matemáticas o Física, en donde la clase natural de problemas es la de los números reales (o complejos), resulta conveniente la consideración inicial acerca de objetos atómicos. Si las ecuaciones a tratar fueran de tipo diofántico, en donde aparecen soluciones y coeficientes enteros, se considerarían atómicos solo las clases de los números naturales y enteros.

Los objetos compuestos se pueden construir a partir de objetos atómicos y de otros objetos compuestos. A cada una de las componentes del objeto se le puede asignar opcionalmente un nombre, que es un símbolo. A su vez, se pueden exigir condiciones a sus elementos. Estas condiciones pueden ser predicados predefinidos directamente evaluables, o bien predicados más generales construidos sobre tipos.

Así, por ejemplo, se puede considerar el objeto compuesto `CIRCUNFERENCIA` construido a partir un centro (un punto en dos dimensiones) y de un radio (un numero real), con la condición de que el radio sea positivo (utilizando el símbolo `circunferencia`

como constructor para expresiones formales según este elemento de las Bases de Conocimiento). El elemento de las Bases de Conocimiento que lo define es:

```
(obj circunferencia ((centro (PUNTO 2)) (radio REAL))          (1)
     (> radio 0))
```

También se puede definir un tipo a partir de un numero indeterminado de componentes, como es el caso del tipo PUNTO formado por coordenadas de tipo REAL a las que no se les exige ninguna condición (condición V de Verdadero). Esta definición es:

```
(obj punto (&rest (coordenadas REAL))                          (2)
     V)
```

También es posible definir en las Bases de Conocimiento constantes que representan a objetos característicos, tales como e, pi, R (conjunto de números reales), RR (producto cartesiano), etc. Estas constantes se definen asignándolas a un tipo de objeto o a determinadas expresiones formales:

```
(cons e REAL)
(cons pi REAL)
(cons R (conjunto (<> ?x REAL) V))
(cons RR (conjunto ((<> ?x REAL) (<> ?y REAL)) V))
```

2.1.2 Metafunciones

A partir del conjunto de objetos se pueden establecer procedimientos para obtener un objeto a partir de otros mediante definiciones funcionales. Estos procedimientos se denominan metafunciones (se utiliza el prefijo meta para distinguirlas de las funciones matemáticas que se consideran como objetos, aparte de que llevan asociado un proceso de metaevaluación en su definición funcional). Además, una metafunción se puede definir de varias maneras en las Bases de Conocimiento dependiendo de los tipos de los argumentos. Cada definición de una metafunción contiene:

- Un nombre, que es un símbolo y actúa como constructor de expresiones.
- Unos argumentos, sobre los cuales se aplica, y de los que se especifican sus tipos.

- Un tipo, que es el objeto resultado de la aplicación de la metafunción a los argumentos.

- Una condición, la cual debe ser verificada por los argumentos.

- Una semántica, correspondiente a cada definición de una metafunción, y que consiste en una regla de reescritura que asocia a cada expresión cuyo primer elemento es el nombre de la metafunción un valor que es un objeto del tipo asociado a la metafunción.

- Una condición de aplicación, que es un argumento opcional que por defecto vale `F` (Falso), y que define la prioridad de aplicación de la semántica en el proceso de resolución (si es `V`, entonces se aplica inmediatamente).

Así, por ejemplo, en el campo de la Geometría tridimensional, si se consideran objetos tales como planos, superficies o puntos, se puede hablar de la metafunción `plano-tangente`. Esta metafunción utiliza como argumentos los tipos `SUPERFICIE`, definido mediante una función de tres variables con un valor real, y `(PUNTO 3)`, para obtener un objeto resultado de tipo `PLANO`. En este caso, la condición de la metafunción es el metapredicado que asegura que el punto pertenece a la superficie. Además, esta metafunción contiene una semántica que describe el procedimiento de obtener dicho plano: es un plano determinado por el punto inicial y por un vector normal a la superficie en dicho punto que se puede obtener considerando el gradiente de la función de tres variables que define a dicha superficie. Esta metafunción se define mediante:

```
(fun plano-tangente ((s SUPERFICIE) (p (PUNTO 3)))          (3)
     PLANO
     (pertenece p s)
     (plano (vector-normal s p) p)
     (condicion-aplicacion V))
```

En esta definición `vector-normal` es otra metafunción que utiliza en su semántica a la metafunción `derivada`. Es decir, en el proceso de aplicación de la semántica de una metafunción puede aparecer una cadena de nuevas metafunciones. En este caso, la cadena contiene a las metafunciones `plano-tangente`, `vector-normal` y `derivada`. De este modo, se establece una igualdad direccionada entre los estados intermedios obtenidos mediante la aplicación de las metafunciones en esta cadena, análogamente a la direccionalidad en el caso del demostrador de Boyer y Moore, [Boyer, Moore 88].

La aplicación de estas semánticas puede contener procedimientos de manipulación algebraica para el caso de metafunciones como derivada de una función. Así, el proceso de derivación de una función se realiza mediante métodos simbólicos, sin acudir a la estricta definición de derivada de una función en un punto. Este proceso de evaluación difiere de los utilizados en los sistemas descritos en el capítulo de antecedentes, tales como Muscadet, [Pastre 89], u Ontic, [McAllester 89], en donde no se permite el cálculo simbólico.

Es posible también tener una metafunción con mas de una definición. Así, por ejemplo, se puede definir la metafunción distancia actuando entre pares de puntos mediante la expresión euclídea, o bien entre subconjuntos del plano como el ínfimo de la distancia entre pares posibles de uno y otro conjunto, y obteniendo en ambos casos un objeto de tipo REAL como valor final. La condición para la metafunción distancia en ambos casos es que se aplique a subconjuntos de RR. Se tendrían las siguientes definiciones:

```
(fun distancia ((p1 PUNTO) (p2 PUNTO))                              (4)
     REAL
     (= (dimension p1) (dimension p2))
     (distancia-euclidea p1 p2)
     (condicion-aplicacion V))

(fun distancia ((s1 CONJUNTO) (s2 CONJUNTO))                        (5)
     REAL
     (y (contenido s1 RR) (contenido s2 RR))
     (infimo
       (funcion ((<> ?p1 (PUNTO 2)) (<> p2 (PUNTO 2)))
                (distancia ?p1 ?p2)
                :dominio
                  (conjunto
                    ((<> ?p1 (PUNTO 2)) (<> p2 (PUNTO 2)))
                    (y (pertenece ?p1 s1) (pertenece ?p2 s2))))))
```

en donde subexpresiones como (<> ?p1 PUNTO) son asignaciones de tipo que asocian a un símbolo como ?p1, denominado metavariable, un tipo objeto como PUNTO.

2.1.3 Metapredicados

También se pueden establecer relaciones entre varios tipos u objetos mediante metapredicados para obtener un valor Verdadero o Falso (`V` o `F`). Un metapredicado puede tener, análogamente al caso de las metafunciones, varias definiciones. Cada definición está formada por los siguientes elementos:

- Un nombre, que es un símbolo y actúa como constructor de expresiones.
- Unos argumentos, sobre los que actúa el metapredicado, y que son tipos que representan a objetos, o bien argumentos de tipo booleano (`BOOL`) en el caso de los constructores lógicos.
- Una condición, que deben cumplir los argumentos, que es a su vez otro metapredicado, o bien un predicado predefinido de evaluación directa.
- Una semántica, asociada al metapredicado que expresa una regla de reescritura para obtener el valor `V` o `F` de expresiones construidas a partir de este metapredicado. A su vez, la semántica es otro metapredicado o un predicado predefinido.
- Una condición de aplicación, que es opcional y tiene un comportamiento análogo al caso de las metafunciones,

Así, por ejemplo, se puede considerar el metapredicado `tangente` aplicado a dos superficies y un punto, con la condición de que el punto pertenezca a las dos superficies. La definición de dicho metapredicado puede ser expresada comprobando que los vectores normales a dichas superficie en el punto dado son paralelos. Este metapredicado se define en Las Bases de Conocimiento mediante:

```
(pred tangente ((s1 SUPERFICIE) (s2 SUPERFICIE) (p (PUNTO 3)))    (6)
      (y (pertenece p s1) (pertenece p s2))
      (paralelo (vector-normal s1 p) (vector-normal s2 p)))
```

Existe una cadena de metapredicados asociados a la aplicación de `tangente` hasta obtener un valor final, de modo que se establecen igualdades direccionadas a través de los metapredicados `tangente`, `paralelo` y `=`. En el proceso general de deducción, la aplicación de la semántica de un metapredicado es controlada mediante heurísticas generales de resolución de un problema. En este sentido, si un metapredicado aparece en una base de hechos conocidos y es también un objetivo en el problema, entonces ya está demostrado.

También pueden existir varias definiciones para un mismo metapredicado. Por ejemplo, se puede definir el metapredicado tangente para expresar el concepto de tangencia entre curvas en un punto, o bien se puede definir también para algún tipo determinado de superficies expresando otro metapredicado equivalente que sea más sencillo de comprobar. Por ejemplo, podemos expresar la tangencia entre esferas en un punto en función de que sean paralelos los vectores obtenidos uniendo dicho punto con los centros:

```
(pred tangente ((s1 ESFERA) (s2 ESFERA) (p (PUNTO 3)))          (7)
      (y (pertenece p s1) (pertenece p s2))
      (paralelo (- p (centro s1)) (- p (centro s2)))
      (condicion-aplicacion V))
```

Existen también metapredicados que se construyen a partir de otros metapredicados. Corresponden a los constructores lógicos $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$, $\forall$ y $\exists$. Entre ellos, solo $\leftrightarrow$ tiene una semántica que lo define, mientras que a los demás no se les asigna ninguna, y se consideran a nivel declarativo para ser tratados mediante el sistema de resolución de problemas. Así, se tienen definiciones de metapredicados (la aparición de la condición V indica que no se les exige ninguna condición) tales como:

```
(pred <-> ((b1 BOOL) (b2 BOOL))
      V
      (y (-> b1 b2) (-> b2 b1)))

(pred -> ((b1 BOOL) (b2 BOOL))
      V)
```

2.1.4 Jerarquía de tipos

La estructura de conceptos utilizados en el proceso de resolución se describe mediante un conjunto de tipos. En este conjunto se establece una jerarquía,
[Castells, Moriyón, Saiz 93, a], cumpliéndose que ciertos pares están relacionados siendo el primer elemento un subtipo del segundo. Se van a denotar los tipos mediante símbolos en mayúsculas, que a su vez pueden ser utilizados para construir, según unas determinadas reglas gramaticales, listas que denotan nuevos tipos.

El conjunto de tipos esta jerarquizado mediante una estructura de grafo orientado acíclico. Partes de este grafo tienen estructura de árbol. Sin embargo, en el grafo

completo se permiten casos como el de un tipo que tenga dos supertipos que no se puedan ser relacionados entre sí mediante la jerarquía.

La estructura de tipos se crea dinámicamente a partir de las Bases de Conocimiento. Existen tipos y relaciones que son universales para todas las Bases de Conocimiento. Así, se tiene un tipo universal EXPRESION-PROGENES. Subtipos de este tipo universal son OBJETO, BOOL y EXPR-ASIGN-TIPO (expresión de asignación de tipo). Este último tipo reúne expresiones en las que se asigna una metavariable a algún tipo que sea un subtipo del tipo OBJETO. Dentro del tipo OBJETO, un primer nivel de desdoblamiento aparece en la distinción entre OBJETO-ATOMICO y OBJETO-COMPUESTO. Subtipos de este último son CONJUNTO y FUNCION y, en general, objetos definidos en las Bases de Conocimiento a partir de subtipos del tipo OBJETO, tales como por ejemplo CIRCUNFERENCIA, ELIPSOIDE, etc. Cabe señalar que los tipos CONJUNTO y FUNCION se construyen a partir de elementos que son subtipos de EXPR-ASIGN-TIPO y BOOL, ya que necesitan metavariables con asignaciones. Gráficamente, se tiene la siguiente estructura universal de tipos:

```
-EXPRESION-PROGENES
  -OBJETO
    -OBJETO-ATOMICO
    -OBJETO-COMPUESTO
      -CONJUNTO
      -FUNCION
  -BOOL
  -EXPR-ASIGN-TIPO
```

A partir de esta arquitectura de tipos se definen en las Bases de Conocimiento tipos específicos a cada dominio que son subtipos de OBJETO-ATOMICO o de OBJETO-COMPUESTO. Así mismo, se repiten subestructuras jerárquicas, como es el caso para subtipos de EXPR-ASIGN-TIPO que se construyen a partir de subtipos de OBJETO. Por tanto, además de esta estructura de tipos para cualquier tipo de dominio, aparecen tipos específicos a disciplinas como Geometría, Mecánica Newtoniana, Calculo Infinitesimal, etc. Así, por ejemplo, subtipos de OBJETO-COMPUESTO son RECTA, ELIPSOIDE, PLANO o PARTICULA. Un tipo como PLANO además es subtipo del tipo CONJUNTO produciéndose una conexión entre distintas ramas en el grafo de tipos.

En las Bases de Conocimiento también se describen relaciones específicas entre dos tipos relacionándolos en la jerarquía, y permitiendo opcionalmente definir una función de traducción entre ambos que transforme objetos del primer tipo en objetos del segundo. Dos ejemplos de estas definiciones son:

```
(subtipo RACIONAL REAL)                                                    (8)

(subtipo (c1 CIRCUNFERENCIA) CONJUNTO                                      (9)
         (conjunto (<> ?p (PUNTO 2))
                   (= (distancia ?p (centro c1)) (radio c1))))
```

En el primer caso se afirma que RACIONAL es un subtipo de REAL y que no existe traducción específica de un objeto de tipo RACIONAL a otro de tipo REAL. En el segundo caso, se establece que CIRCUNFERENCIA es un subtipo de CONJUNTO y se da una traducción de un objeto de tipo CIRCUNFERENCIA al tipo CONJUNTO (se afirma que una circunferencia se expresa como el conjunto de puntos del plano cuya distancia al centro es igual al radio) estableciendo una igualdad que en el caso de la traducción está direccionada desde el tipo hasta el supertipo.

Aparte de la estructura jerárquica universal en el conjunto TIPOS, se pueden construir tipos mediante listas de subtipos estrictos de OBJETO tales como (ENTERO CONJUNTO) que siguen siendo subtipos de OBJETO heredando las propiedades jerárquicas de sus tipos componentes.

2.1.5 Teoremas

En las Bases de Conocimiento existe un módulo de conocimiento declarativo en forma de reglas que expresan teoremas y propiedades de uso común dentro de cada dominio. Este tipo de conocimiento puede ser completado interactivamente añadiendo resultados generales obtenidos tras la resolución de determinados problemas.

El conjunto de teoremas se puede utilizar como reglas de producción que se aplican mediante un mecanismo de inferencia *forwards*, o bien, en deducción *backwards* para simplificar la resolución de determinados subproblemas. En el capitulo 4 se describen los mecanismos generales de deducción a partir del conocimiento declarativo de los teoremas.

Determinados teoremas se pueden representar mediante una expresión booleana en la que aparecen cuantificadores universales, se generan metavariables a las que se les asignan tipos, y se utiliza una implicación que obtiene resultados generales a partir de las condiciones del teorema. Si utilizamos, por ejemplo, una Base de Conocimiento referida al Cálculo Infinitesimal, se puede considerar el teorema:

```
(teor (para-todo (<> ?f FUNCION-REAL)                                    (10)
        (-> (y (derivable ?f)
               (> (derivada ?f) 0))
            (estrict-creciente ?f))))
```

En este teorema se especifican condiciones suficientes para afirmar que una función sea estrictamente creciente. Es conocimiento declarativo en donde se considera una metavariable `?f` asignada al tipo `FUNCION-REAL` estableciendo una cuantificación universal en `?f`.

Por tanto, se tiene asociada al metapredicado `estrict-creciente` la semántica que expresa la monotonía estricta de la función (definida en las Bases de Conocimiento), y, por otro lado, teoremas como `(10)` que expresan condiciones suficientes, como es en este caso que la función sea derivable y tenga derivada positiva . Es decir, que en el contexto de las siguientes expresiones formales:

```
(> (derivada ?f) 0)                                                      (11)

(estrict-creciente ?f)                                                   (12)

(para-todo ((<> ?x REAL) (<> ?y REAL))                                   (13)
           (-> (< ?x ?y) (< (imagen ?f ?x) (imagen ?f ?y))))
```

la utilización de estos dos niveles de transformación en el proceso deductivo considera `(11)->(12)` con mayor prioridad que `(12)->(13)`, debido a que los teoremas proporcionan demostraciones alternativas mas sencillas en muchos problemas de Matemáticas. Así, por ejemplo, plantear demostrar que una función polinómica de tercer grado es creciente puede resultar intratable utilizando la definición exacta de función estrictamente creciente, mientras que considerando la derivada de la función, que es una expresión polinómica de segundo grado, resulta más sencillo de verificar.

Para tener en cuenta estas consideraciones acerca de la definición del metapredicado `estric-creciente`, se define la condición de aplicación en dicho metapredicado con valor `F` (Falso), que se le asigna por defecto. De este modo, se declara una baja prioridad de aplicación de la definición para `estric-creciente`. Por tanto, la definición en las Bases de Conocimiento de dicha metafunción sería la siguiente:

```
(pred estric-creciente ((?f FUNCION-REAL))                    (14)
      V
     (para-todo ((<> ?x REAL) (<> ?y REAL))
        (-> (< ?x ?y) (< (imagen ?f ?x) (imagen ?f ?y)))))
```

2.2 Lenguaje Formal

Las Bases de Conocimiento definen un Lenguaje Formal constituido por expresiones formales bien formadas. La sintaxis de estas expresiones se define a partir de reglas gramaticales asociadas a las Bases de Conocimiento que además asignan a cada expresión un tipo. Aparte de este carácter declarativo del Lenguaje Formal, las expresiones formales también pueden ser transformadas en nuevas expresiones mediante traducción (2.3) y metaevaluación (2.4). En lo que sigue se describen las clases de expresiones formales que constituyen el Lenguaje Formal.

Las expresiones formales se representan en forma de listas con notación prefija, o bien mediante expresiones atómicas utilizando símbolos o números. En el caso de considerar listas que representen a objetos compuestos, estas tienen la forma de:

```
(<constructor> <componente>+)
```

en donde `<constructor>` es un símbolo definido en las Bases de Conocimiento y `<componente>` una expresión formal que cumple las restricciones exigidas en la definición de `<constructor>`, en cuanto a que debe tener un tipo que sea subtipo del tipo asociado a los argumentos de `<constructor>`, y en cuanto a que debe verificar el metapredicado que aparece en la condición.

Para definir el Lenguaje Formal y relacionar expresiones formales con el conjunto de tipos, se utilizan las funciones `TIPO` y `SUBTIPO` y el conjunto `TIPOS`. La función `TIPO` asocia a una expresión formal su tipo. Por otro lado, la función `SUBTIPO` relaciona un par de tipos (`<tipo1>` y `<tipo2>`) obteniendo `V` si `<tipo1>` es un subtipo de `<tipo2>` y `F` en caso contrario. Se utiliza también la notación `<tipo1> < <tipo2>` si `<tipo1>` es un subtipo de `<tipo2>`. Cabe señalar también que la función `TIPO` admite como parámetro opcional unas asignaciones de tipo en el caso de que una expresión tenga metavariables libres. Es decir, se define el tipo del par formado por `<expresion>` y `<asignaciones>` como un elemento de `TIPOS`. Una expresión formal que no tiene variables libres se denomina canónica (se dice que en tal caso que `<asignaciones> := ()`).

2.2.1 Representación gramatical de objetos

Los objetos se representan mediante expresiones formales a las que se les asigna un tipo subtipo de OBJETO. Los elementos de las Bases de Conocimiento cuyo primer elemento es atom definen expresiones cuyo tipo es subtipo de OBJETO-ATOMICO, mientras que los que empiezan por obj definen otras expresiones con tipo subtipo de OBJETO-COMPUESTO. Además, se respeta la jerarquía de tipos en la asignación de tipos a expresiones. Es decir, si una expresión es de tipo atómico, entonces se le asigna como tipo el mínimo de los posibles tipos tales que dicho objeto verifica sus predicados característicos. Así, por ejemplo, el símbolo 4 es una expresión formal que verifica los predicados predefinidos para los tipos NATURAL y ENTERO, pero TIPO(4) := NATURAL porque es el mínimo de los dos en la estructura jerárquica de tipos.

Dentro del tipo OBJETO-COMPUESTO existen subtipos tales como CIRCUNFERENCIA, definido en las Bases de Conocimiento mediante (1), que tienen asociadas reglas gramaticales de construcción de expresiones formales. Así, por ejemplo, la regla asociada a la definición de (1) es:

```
Si <centro> cumple TIPO(<centro>) < (PUNTO 2)
   y <radio> verifica TIPO(<radio>) < REAL
   y (> ?radio 0)
Entonces
   TIPO(circunferencia <centro> <radio>) := CIRCUNFERENCIA
```

Por tanto, se cumple que TIPO(circunferencia (punto 2 4) 3) := CIRCUNFERENCIA puesto que se verifican las condiciones de esta regla considerando las asignaciones de <punto> a (punto 2 4) y de <radio> a 3:

```
TIPO(punto 2 4) := (PUNTO 2)                                    (15)
TIPO(3) := NATURAL < REAL
(> 3 0)
```

Igualmente, se verifica (15) a partir de la regla gramatical deducida a partir de la definición de expresiones de tipo PUNTO dada en (2), según la cual se asigna el tipo (PUNTO N) a una lista cuyo primer elemento es el símbolo punto y tiene N argumentos de tipo subtipo de REAL.

Subtipos del tipo OBJETO-COMPUESTO son tambien CONJUNTO y FUNCION. Sin embargo, poseen reglas gramaticales distintas puesto que sus componentes no deben tener

un tipo que sea subtipo del tipo OBJETO, como en el caso de CIRCUNFERENCIA, pudiendo tener como componentes expresiones de tipo EXPR-ASIGN-TIPO (denominadas expresiones de asignación de tipo) o BOOL (que se llaman expresiones booleanas).

Se denomina asignación de tipo a una expresión formal construida a partir del símbolo <>, de una metavariable (símbolo que empieza por el carácter ?), y por un tipo que sea subtipo del tipo OBJETO. Por ejemplo, (<> ?x REAL) sería un ejemplo de asignación de tipo REAL. A partir de asignaciones de tipo se pueden construir expresiones de asignación de tipo sustituyendo determinadas subexpresiones que representan a objetos por asignaciones de tipo compatibles. Las expresiones de asignación de tipo son expresiones formales con un tipo subtipo del tipo EXPR-ASIGN-TIPO. Así, por ejemplo, el tipo (EXPR-ASIGN-TIPO CIRCUNFERENCIA) se asigna a las siguientes expresiones de asignación de tipo:

```
TIPO(<> ?x CIRCUNFERENCIA) =
TIPO(circunferencia (<> ?p (PUNTO 2)) (<> ?r REAL)) :=
(EXPR-ASIGN-TIPO CIRCUNFERENCIA)
```

Por otra parte, las expresiones boolenas son expresiones formales con tipo BOOL, y se construyen a partir de metapredicados, como los definidos en 2.2.2. Un ejemplo sería la expresión (= 2 3) construida a partir del metapredicado =. La igualdad es, por tanto, una clase de metapredicado que se puede aplicar a diferentes tipos de argumentos. Así, se puede considerar igualdad entre números, funciones, conjuntos, etc.

Un conjunto se construye a partir de una expresión de asignación de tipo y de una expresión booleana. Por ejemplo, el conjunto de puntos de la recta $x + y = 0$ se representa mediante:

```
(conjunto (punto (<> ?x REAL) (<> ?y REAL)) (= (+ ?x ?y) 0))
```

que es una expresión de tipo CONJUNTO, verificándose que:

```
TIPO(punto (<> ?x REAL) (<> ?y REAL)) := (EXPR-ASIGN-TIPO (PUNTO 2))
TIPO((= (+ ?x ?y) 0), ((<> ?x REAL) (<> ?y REAL))) := BOOL
```

Análogamente, se puede construir una función real a partir de una expresión de asignación de tipo y una definición de la imagen. Así, por ejemplo, la función $f(x) = 2x + 2$ se representa mediante:

```
(funcion (<> ?x REAL) (+ (* 2 ?x) 2))
```

que es una expresión de tipo FUNCION-REAL. También se puede definir una función real a partir de una expresión de asignación de tipo con restricciones, es decir un conjunto, como por ejemplo la función logaritmo:

```
(funcion (<> ?x REAL) (log ?x)
         :dominio (conjunto (<> ?x REAL) (> ?x 0)))
```

que también es una expresión de tipo FUNCION-REAL. También se pueden considerar funciones definidas en otro tipo de dominios, como es el caso de la función área de una circunferencia, cuyo dominio es el conjunto de todas las circunferencias, que se representa por:

```
(funcion (<> ?c CIRCUNFERENCIA) (area ?c))
```

que es una expresión de tipo FUNCION cuya definición es la acción de la metafunción area aplicada a cada metavariable ?c.

Como se señala en 2.1, las Bases de Conocimiento definen en el Lenguaje Formal constantes que representan a otras expresiones formales fijas o a objetos con propiedades específicas. Estas constantes son símbolos tales como e, pi o R (conjunto de números reales), y cumplen igualdades como que TIPO(e) = TIPO(pi) := REAL y TIPO(R) := CONJUNTO.

Se pueden también considerar expresiones formadas sin constructor a partir de otras expresiones cuyos tipos sean subtipos de OBJETO. En tal caso dichas expresiones se asocian a otra clase de tipos construidos mediante listas a partir de los iniciales definidos en las Bases de Conocimiento. Por ejemplo, se tiene las siguientes identidades:

```
TIPO((punto 2 3) 3) := ((PUNTO 2) NATURAL)
TIPO((<> ?x REAL), (<> ?c CIRCUNFERENCIA)) :=
  (EXPR-ASIGN-TIPO (REAL CIRCUNFERENCIA))
```

2.2.2 Expresiones formales con metafunciones y metapredicados

Las metafunciones y metapredicados permiten construir expresiones formales de un tipo subtipo de OBJETO, en el caso de las metafunciones, y de tipo BOOL, en el caso de los metapredicados.

Así, por ejemplo, a partir de la definición (3), se construye la siguiente expresión de tipo PLANO:

```
(plano-tangente (esfera (punto 0 0 0) 3) (punto 3 0 0))
```

En este caso, las subexpresiones tienen tipos respectivos ESFERA y (PUNTO 3). En las Bases de Conocimiento se exige que sus componentes sean de tipos que sean subtipo de SUPERFICIE y (PUNTO 3), como efectivamente ocurre puesto que
ESFERA < SUPERFICIE.

En el caso de un metapredicado, se puede construir la siguiente expresión formal a partir de la definición (6):

```
(tangente (esfera (punto 0 0 0) 2)
          (plano 0 0 1 -2)
          (punto (2 0 0)))
```

que tiene tipo BOOL puesto que sus elementos son de tipos subtipos de los que aparecen en la definición de este metapredicado (se define el tipo PLANO en función los coeficientes A, B, C y D en la ecuación $Ax + By + Cz = D$ en coordenadas cartesianas), y porque se cumple la condición asociada a la definición del metapredicado tangente, ya que el punto pertenece al plano y a la esfera.

Se pueden tener expresiones en que el constructor, metafunción o metapredicado, sean el mismo pero se apliquen a distintas clases de objetos. Así, por ejemplo, la metafunción distancia se puede aplicar a pares de puntos o a pares de subconjuntos, con la misma dimensión, para obtener un numero real. Así, las expresiones:

```
(distancia (punto 1 2) (punto 3 4))
(distancia (conjunto (punto (<> ?x REAL) (<> ?y REAL))
                     (= (* ?x ?y) 1))
           (conjunto (punto (<> ?x REAL) (<> ?y REAL))
                     (= ?y (* -1 ?x))))
```

tienen tipo REAL y están construidas según las definiciones (4) y (5) de las Bases de Conocimiento para la metafunción distancia. Análogamente, se puede considerar, aparte de la metafunción tangente entre superficies, la expresión siguiente:

```
(tangente (curva (funcion ((<> ?x REAL) (<> ?y REAL))
                         (-- ?y (+ ?x 2))))
          (curva (funcion ((<> ?x REAL) (<> ?y REAL))
                         (++ ?y (sen ?x) 2)))
          (punto 0 0))
```

construida a partir de la definición en las Bases de Conocimiento para la metafunción tangente entre curvas.

Otro ejemplo de metapredicado con diferentes semánticas es =. La igualdad tiene distinto tratamiento según sean los objetos que se están relacionando. Así, podemos establecer diferentes expresiones formales que la involucran como:

```
(= 2 4)
(= (conjunto (<> ?x REAL) (= (sin ?x) 0))
   (conjunto (<> ?x REAL) (= (cos ?x) 0)))
```

Se pueden construir también expresiones formales a partir de los metapredicados lógicos. En este caso, las componentes de las expresiones no tienen tipo subtipo de OBJETO, sino que se permiten también tipos subtipos de EXPR-ASIGN-TIPO o BOOL. Son las análogas a los tipos como CONJUNTO o FUNCION en donde aparecen asignaciones de tipo. Ejemplos de expresiones formales de tipo BOOL son:

```
(y (paralelo (plano 1 2 -1 4) (plano 2 -1 3 2)) (par 3))
(para-todo (<> ?x ENTERO) (existe (<> ?y ENTERO) (= (* 2 ?x) ?y)))
(para-todo (<> ?x REAL) (-> (> ?x 1) (> (log ?x) 0)))
```

Cabe señalar que dentro de una misma expresión PRÓGENES pueden aparecer dos asignaciones de tipo distintas con la misma variable. Un ejemplo es la expresión siguiente construida a partir de la definición de la metafunción interseccion:

```
(interseccion (conjunto (<> ?x REAL) (> ?x 1))
              (conjunto (<> ?x REAL) (< ?x 3)))
```

que es una expresión de tipo CONJUNTO. Asociado a cada expresión de asignación de tipo existe un ámbito sobre el que tiene influencia. En este sentido, el ámbito de la primera asignación de ?x es el primer conjunto, por lo que se puede generar de nuevo la metavariable ?x para el segundo.

También se pueden construir expresiones mediante la metafunción el aplicada a una expresión de asignación de tipo y a una expresión booleana, representando un objeto desconocido definido por la propiedad que expresa un metapredicado. En este sentido, la expresión:

```
(el (<> ?x REAL) (y (= (sin ?x) 1/3) (> ?x 0) (< ?x 2)))
```

tiene tipo REAL, verificándose:

```
TIPO(<> ?x REAL) := (EXPR-ASIGN-TIPO REAL)
TIPO((y (= (sin ?x) 1/3) (> ?x 0) (< ?x 2)), (<> ?x REAL)) := BOOL
```

También se pueden construir expresiones formales a partir de las funciones de acceso a componente que se utilizan como nombre de componentes en definiciones de tipo OBJETO-COMPUESTO de las Bases de Conocimiento. Así, por ejemplo, a partir de la definición (1) de CIRCUNFERENCIA, en donde el nombre de la segunda componente es radio, se puede construir la expresión:

```
(radio (circunferencia (punto 0 2) 3))
```

que es de tipo REAL. En el caso de definiciones de objetos compuestos con un número indeterminado de componentes tales como (2) de tipo PUNTO, se pueden construir expresiones con propiedades tales como:

```
TIPO(coordenadas(punto -1 2)) := (ENTERO NATURAL)
```

2.2.3 Jerarquía de tipos y teoremas

La función SUBTIPO define la jerarquía de tipos a partir de las Bases de Conocimiento. Definiciones como (8) y (9), definen valor V para relaciones tales como:

```
SUBTIPO(ENTERO,REAL)
SUBTIPO(CIRCUNFERENCIA,CONJUNTO)
```

Para el caso de objetos compuestos por un número indeterminado de componentes, como es el caso de PUNTO, se cumplen propiedades como:

```
SUBTIPO((PUNTO 2), PUNTO)
```

Por otra parte, como se señalaba en 2.1.4, se pueden construir tipos mediante listas de subtipos de OBJETO, los cuales siguen siendo subtipos de OBJETO. Así, por ejemplo se tienen relaciones como:

```
SUBTIPO((ENTERO CONJUNTO), OBJETO)
SUBTIPO((NATURAL ESFERA), (ENTERO SUPERFICIE))
```

Además, los subtipos de OBJETO y de EXPR-ASIGN-TIPO tienen la misma estructura cumpliéndose propiedades como:

```
SUBTIPO((EXPR-ASIGN-TIPO (NATURAL ESFERA))
        (EXPR-ASIGN-TIPO (ENTERO SUPERFICIE)))
```

Por otra parte, los teoremas son expresiones formales de tipo BOOL construidas a partir de los cuantificadores para-todo y existe y de los constructores lógicos -> y y. Tienen un carácter declarativo y se utilizan en el proceso general de resolución integrando su aplicación junto con otros procesos de transformación en expresiones formales, tales como metaevaluación, traducción y expansión. Así, por ejemplo, la expresión de tipo BOOL:

```
(para-todo (<> ?f FUNCION)
   (-> (y (derivable ?f) (> (derivada ?f) 0))
       (estrict-creciente ?f)))
```

se utiliza como una regla para establecer razonamiento *forwards* o *backwards* en la resolución de un problema, permitiendo en ciertos casos ignorar la definición semántica exacta del metapredicado estrict-creciente.

La formulación de un teorema es universal en cuanto a la clase de objetos que aparecen. Cuando se lleva a cabo el proceso de deducción, sea *forwards* o *backwards*, se consideran las subexpresiones formales cuyos tipos sean compatibles con los expresados en los teoremas, y se utiliza deducción sobre ellos.

También se pueden expresar teoremas mediante cuantificadores universales sin que aparezca el metapredicado ->. O bien, puede aparece sólo el cuantificador existencial. Por ejemplo, la expresión siguiente es un teorema (la Ley de Newton):

```
(para-todo
   (<> ?s SISTEMA-MECANICO)
   (= (fuerza ?s)
      (* (masa (particula ?s)) (aceleracion (particula ?s)))))
```

que se aplica a cualquier objeto de tipo SISTEMA-MECANICO formado por una partícula y un campo de fuerzas. Igualmente, se pueden considerar teoremas a partir de cuantificación existencial en donde se introduzca un objeto con propiedades especiales, estableciendo axiomas que se pueden utilizar en un proceso deductivo.

Por otra parte, en PRÓGENES se pueden definir también teoremas en los que la conclusión es una igualdad que siempre se aplica en una única dirección. En tal caso, se permite utilizar el constructor :=, y gramaticalmente es equivalente a = pero define una regla de reescritura. Asi, por ejemplo, el teorema:

```
(teor (para-todo                                                    (16)
         ((<> ?f FUNCION) (<> ?g FUNCION) (<> ?a OBJETO))
         (-> (y (pertenece ?a (dominio ?g))
                (contenido (rango ?g) (dominio ?f)))
             (:= (imagen (composicion ?f ?g) ?a)
                 (imagen ?f (imagen ?g ?a))))))
```

expresa el tratamiento habitual utilizado en Matemáticas de aplicar la regla $(f \circ g)(a) := f(g(a))$ inmediatamente.

2.2.4 Tratamiento de conjuntos

Las Bases de Conocimiento PRÓGENES permiten hacer referencia a conjuntos, con lo que a su vez se permite el tratamiento de conceptos matemáticos basados en esta teoría axiomática como hace habitualmente el matemático. En PRÓGENES hay dos constructores de conjuntos: el-conjunto y conjunto. Se pueden construir conjuntos por extensión, como el conjunto $\{1, 2, 3\}$ que se representa como expresión formal como (el-conjunto 1 2 3). En general los conjuntos por extensión se definen a partir de expresiones de tipo OBJETO de acuerdo con la definicion siguiente:

```
(obj el-conjunto (&rest (objetos OBJETO)) V CONJUNTO)
```

También se pueden construir conjuntos por comprensión a partir del constructor conjunto, una expresión de asignación de tipo y un predicado, como por ejemplo:

```
(conjunto (circunferencia (punto 2 5) (<> ?r REAL)) (> ?r 1))
```

Solamente algunos tipos tienen conjuntos asociados. La lista de tipos maximales respecto a la relación SUBTIPO que tienen esta propiedad se denota por *TIPOS-CONJUNTO* (ningun supertipo del tipo CONJUNTO pertenece a esa lista), y tipo-conjunto-p denota el predicado que reconoce si un tipo define un conjunto. En general, un conjunto se puede definir por comprensión a partir de una expresión de asignación de un tipo que sea subtipo de algun elemento de *TIPOS-CONJUNTO*, y una expresion booleana arbitraria. Se tiene la definicion:

```
(obj conjunto ((expr-asign EXPR-ASIGN-TIPO) (cond BOOL))
     (tipo-conjunto-p (tipo-asignado (tipo expr-asign))))
```

Así mismo, se define el tipo FUNCION mediante:

```
(obj funcion ((expr-asign EXPR-ASIGN-TIPO) (def OBJETO)
              &optional
                (dominio CONJUNTO (conjunto expr-asign V))
                (rango CONJUNTO
                       (conjunto
                         (<> ?x (tipo def (asignaciones expr-asign)))
                         V)))
     (y (tipo-conjunto-p (tipo-asignado (tipo expr-asign)))
        (contenido dominio (conjunto expr-asign V))
        (contenido (conjunto-imagen :self dominio) rango)))
```

Ademas conceptos habituales en Teoría de Conjuntos tales como $\in$, *imagen*, $\subset$ y $=$ corresponden a los siguientes metapredicados y metafunciones:

```
(pred pertenece ((x OBJETO)
                 (conjunto (expr-asign EXPR-ASIGN-TIPO) (cond BOOL)))
      V
      (y (subtipo (tipo x) (tipo-asignado (tipo expr-asign)))
         (instancia x cond)))
```

```
(fun imagen ((funcion (expr-asign EXPR-ASIGN-TIPO) (def OBJETO))
             (x OBJETO))
      (subtipo (tipo x) (tipo-asignado (tipo expr-asign)))
      (instancia x def))

(pred contenido ((conj1 CONJUNTO) (conj2 CONJUNTO))
      V
      (para-todo (<> ?x OBJETO)
         (-> (pertenece ?x conj1) (pertenece ?x conj2))))

(pred = ((conj1 CONJUNTO) (conj2 CONJUNTO))                    (17)
      V
      (y (contenido conj1 conj2) (contenido conj2 conj1)))
```

Aparte de las definiciones anteriores hay otras expresiones definidas mediante el constructor conjunto y que en PRÓGENES se admiten como conjuntos. Son aquéllas de la forma:

```
(conjunto (<> ?Y CONJUNTO) (contenido ?Y ?X))
```

donde se cumple que (<> ?X CONJUNTO). La metafunción partes-conjunto se define considerando la expresión anterior como su semántica. Igualmente se puede considerar:

```
(conjunto (<> ?Y OBJETO)
          (existe (<> ?Z OBJETO)
                  (y (pertenece ?Z ?X) (pertenece ?Y ?Z))))
```

en donde (<> ?X CONJUNTO) tiene que ser un conjunto cuyos elementos son todos conjuntos, de modo que esta expresión es la semántica de la metafunción union. Por último tambien es una expresión construible:

```
(conjunto (<> ?Y OBJETO)
          (existe (<> ?Z OBJETO)
                  (y (pertenece ?Z ?X) (= (imagen ?F ?Z) ?Y))))
```

en donde se cumple que (<> ?X CONJUNTO), (<> ?F FUNCION) y (contenido ?X (rango ?F)). A partir de esta última expresión se define la metafunción conjunto-imagen.

Los constructores, metapredicados y metafunciones anteriores permiten hacer demostraciones compatibles con la teoría axiomática de Zermelo-Fraenkel, [Jech 78]. A continuación detallaremos la relación de los mismos con cada uno de los axiomas:

- *Axioma de Extensionalidad.*
  Si X e Y tiene los mismos elementos, entonces $X = Y$.
  Su aplicación en PRÓGENES se efectúa mediante la definición del metapredicado =.

- *Axioma de Apareamiento.*
  Para todo a y b existe un conjunto $\{a, b\}$ que contiene exactamente a a y a b.
  En PRÓGENES se aplica utilizando el constructor `el-conjunto`. Obsérvese que PRÓGENES distingue entre `CONJUNTO` y `OBJETO` (que es un supertipo de `CONJUNTO`). Desde el punto de vista axiomático todos los objetos serían conjuntos puesto que el constructor `el-conjunto` admite objetos como componentes. Sin embargo, el Lenguaje Formal PRÓGENES sólo permite referirse a los elementos de un objeto cuando éste tiene tipo `CONJUNTO`.

- *Axioma (esquema) de Separación.*
  Si ϕ es una propiedad (con parámetro p), entonces para todo X y p existe un conjunto $Y = \{u \in X : \phi(u, p)\}$ que contiene todos los $u \in X$ que cumplen la propiedad ϕ.
  Su aplicación en PRÓGENES se efectúa mediante el constructor `conjunto`. Además la condición impuesta a las expresiones de asignación de tipo que aparecen en un conjunto permiten excluir contradicciones como la paradoja de Russel, ya que `CONJUNTO` no es un subtipo de ninguno de los elementos de `*TIPOS-CONJUNTO*`.

- *Axioma de Unión.*
  Para todo X existe el conjunto $Y = \bigcup X$.
  Se aplica en PRÓGENES mediante la metafunción `union`. En la práctica se define otra metafunción en términos de ésta para representar la unión de una familia de conjuntos.

- *Axioma de Conjunto de Partes.*
  Para todo X existe el conjunto de las partes de X, $P(X)$.
  Se aplica en PRÓGENES mediante una metafunción, `partes-conjunto`, que construye el conjunto de los subconjuntos de X. Se aplica mediante la definición de la metafunción `partes-conjunto`.

- *Axioma (esquema) de Reemplazamiento.*
  Si F es una función, entonces para todo X existe el conjunto $Y = F(X) = \{F(x) : x \in X\}$.
  Se aplica utilizando la metafunción `imagen` que se puede aplicar a una función y un conjunto, y obtiene el conjunto imagen.
- Finalmente, el axioma de *Infinito* (Existe un conjunto infinito), el axioma de *regularidad* (Todo conjunto no vacío tiene un elemento $\in$*–minimal*) y el axioma de *elección* (Toda familia de conjuntos no vacíos tienen una función de elección) no se tratan en PRÓGENES.

2.3 Traducción de expresiones formales

Una de las transformaciones posibles de expresiones formales es la traducción, la cual generaliza el proceso clásico de herencia en el contexto del Diseño Orientado a Objetos, [Booch 91]. Se utilizan funciones de traducción entre determinados pares de tipos, permitiendo una mayor expresividad en la definición de metafunciones y metapredicados definidas sobre tipos generales. Por otra parte, la traducción se utiliza en el proceso de metaevaluación de expresiones (2.4). Además, permite un tratamiento alternativo de objetos que facilita la emulación de las actividades cognitivas utilizadas en algunas disciplinas científicas. Por otra parte, la traducción de expresiones establece una igualdad direccionada entre la representación de un objeto según patrones de diferentes tipos admisibles para dicho objeto.

En PRÓGENES, se utiliza la función `TRADUCE` para obtener transformaciones entre expresiones. `TRADUCE` parte de una expresión formal `<expr>` (de tipo `<tipo1>`) y de un tipo `<tipo2>` (que es un supertipo de `<tipo1>`) y obtiene como valor otra expresión de tipo `<tipo2>`, si existe una definición de una función de traducción para el par formado por `<tipo1>` y `<tipo2>`, o dejándola invariante en caso contrario. `TRADUCE` sólo se define entre pares de subtipos de `OBJETO`, modificando la representación de objetos según el patrón definido en las Bases de Conocimiento. Por otra parte, `TRADUCE` admite como argumento opcional `<asignaciones>` si la expresión inicial tiene variables libres, considerando `<asignaciones> := ()` para el caso de una expresión canónica.

2.3.1 Transformación de traducción en Prógenes

En las Bases de Conocimiento se describe la representación *standard* de objetos en función de componentes mas elementales y se establecen relaciones entre tipos en la

jerarquía, [Castells, Moriyón, Saiz 93, a]. La construcción específica de estas relaciones se elabora a partir del conocimiento científico en dominios como Geometría, Cálculo o Física. La representación de objetos ha ido variando a lo largo de la historia dependiendo de la evolución en las interpretaciones de determinados conceptos. Así se puede representar una circunferencia, concebida inicialmente como un objeto geométrico con un punto y un radio, como un conjunto de puntos cuyas coordenadas verifican una ecuación. Es en este contexto en donde se definen las funciones de traducción entre tipos, permitiendo obtener una representación múltiple de objetos. En el caso de una circunferencia, en PRÓGENES, la definicion (9) define la transformación de una expresión de tipo CIRCUNFERENCIA a una expresión de tipo CONJUNTO. Así, por ejemplo, se cumple que:

```
TRADUCE((circunferencia (punto 0 1) 3), CONJUNTO) :=                (18)
(conjunto (<> ?p (PUNTO 2))
   (= (distancia ?p (punto 0 1)) 3))
```

en donde se han utilizado simplificaciones de metaevaluación en subexpresiones de la expresión traducida tales como:

```
(centro (circunferencia (punto 0 1) 3)) := (punto 0 1)
(radio (circunferencia (punto 0 1) 3))  := 3
```

También es posible definir relaciones entre tipos, pero sin especificar una función de traducción. Así, por ejemplo, en (8) se define el tipo RACIONAL como un subtipo de REAL sin asociar ninguna función de traducción a dicha transformación. Por tanto, en este caso se verifica:

```
TRADUCE(4/3, REAL) := 4/3
```

en donde no se representa 4/3 de ninguna forma característica asociada al tipo REAL. Estos casos de invariancia de expresiones bajo transformación en la jerarquía de tipos reflejan la forma de trabajar en estos dominios científicos. Así, en Matemáticas, un número se representa de una única forma y se piensa en sus propiedades como número natural, entero, real o complejo en función del contexto en que se halla, y utilizando propiedades o teoremas relevantes para cada tipo de números.

La función TRADUCE respeta las propiedades de relación de orden de la jerarquía de tipos. Tal es el caso en la cadena de tipos:

```
ESFERA < SUPERFICIE < CONJUNTO
```

verificándose las siguientes igualdades que respetan las propiedades transitiva y reflexiva de la relación inferida por la función SUBTIPO en el conjunto TIPOS:

```
TRADUCE((esfera (punto 0 1 0) 4), CONJUNTO) :=
TRADUCE(TRADUCE((esfera (punto 0 1 0) 4), SUPERFICIE), CONJUNTO)

TRADUCE((elipsoide (punto 1 2 -1) 2 3 4), ELIPSOIDE) :=
(elipsoide (punto 1 2 -1) 2 3 4)
```

en donde (elipsoide (punto 1 2 -1) 2 3 4) representa al elipsoide centrado en $(1, 2, -1)$ con semiejes de longitudes 2, 3 y 4 en la dirección paralela a los ejes x, y y z.

También se pueden traducir expresiones con metavariables libres considerando en tal caso como asignaciones el conjunto de metavariables libres en la expresión inicial. Así, por ejemplo, se cumple que:

```
TRADUCE((elipsoide (punto (<> ?x REAL) (<> ?y REAL) (<> ?z REAL))
                   1 2 3),
        CONJUNTO) :=
(conjunto (punto (<> ?x1 REAL) (<> ?y1 REAL) (<> ?z1 REAL))
    (= (+ (/ (expt (- ?x ?x1) 2) 1)
          (/ (expt (- ?y ?y1) 2) 4)
          (/ (expt (- ?z ?z1) 2) 9))
       1))
```

interpretando que <asignaciones> :=
((<> ?x REAL) (<> ?y REAL) (<> ?z REAL)) como parámetro de TRADUCE.

Por último, cabe señalar que TRADUCE se puede aplicar también a expresiones múltiples (sin constructores) aplicándose a cada uno de los argumentos. Es decir, en el caso de dos parámetros se tendría:

```
Si <expr1> y <expr2> son expresiones formales
   con TIPO(<expr1>) < TIPO1
   y   TIPO(<expr2>) < TIPO2
Entonces
   TRADUCE((<expr1> <expr2>), (TIPO1 TIPO2)) :=
   (TRADUCE(<expr1>, TIPO1) TRADUCE(<expr2>, TIPO2))
```

2.3.2 Traducción en resolución de problemas científicos

El tratamiento clásico en Diseño Orientado a Objetos se establece considerando clases, en donde una clase está determinada por un patrón de representación para objetos y se define a partir de un conjunto de componentes, que a su vez pueden ser nuevas clases. A su vez, un objeto es la instanciación en este patrón de valores específicos para sus componentes. En el conjunto de clases se establece una jerarquía considerando para cada clase un conjunto de superclases de las que hereda propiedades. El mecanismo de herencia, en este contexto, se establece añadiendo determinadas componentes a cada clase en función de las superclases definidas para dicha clase. En este sentido, se pueden definir:

```
<clase1>:  componentes <a1> <a2> <a3>
<clase2>:  subclase de <clase1>
           componentes <b1> <b2> <b3> <b4>
```

en donde se afirma que <clase2> es una subclase de <clase1> y que los elementos de un objeto de <clase2> son los de <clase1> más otros nuevos. Es decir, se tienen las siguientes componentes para cada una de las clases del caso anterior:

```
<clase1>:  componentes <a1> <a2> <a3>
<clase2>:  componentes <a1> <a2> <a3> <b1> <b2> <b3> <b4>
```

El caso clásico de herencia puede ser desarrollado considerando una función de traducción entre dos tipos o clases consistente en suprimir determinadas componentes. Por ejemplo, en el caso anterior, si se tiene un objeto genérico perteneciente a <clase2>, su traducción a una expresión genérica de <clase1> vendría dada con la acción de una función TRADUCE, mediante la siguiente identidad:

```
TRADUCE((clase2 <a1> <a2> <a3> <b1> <b2> <b3> <b4>) <clase1>) :=
(clase1 <a1> <a2> <a3>)
```

El mecanismo de traducción que se define en PRÓGENES generaliza este tratamiento clásico permitiendo definir funciones de traducción arbitrariamente entre pares de tipos en función de las propiedades cognitivas de cada dominio. La naturaleza de los conceptos en los dominios tratados, como Matemáticas y Física, exige que la traducción de expresiones a superclases o supertipos se lleve a cabo mediante reinterpretaciones

de conceptos, lo que no siempre supone una información adicional sino una transformación de representaciones. Así, si consideramos la traducción (18), no se conservan componentes, sino que se transforman de acuerdo a la representación conjuntista de un objeto geométrico de clase circunferencia.

Por otra parte, la traducción entre expresiones establece una igualdad entre una expresión y su traducción. Es decir, si una expresión <expr> se traduce hasta el tipo <tipo> mediante la acción de TRADUCE, entonces se cumple que la expresión (= <expr> TRADUCE(<expr>, <tipo>)) tiene asignado un valor V en la semántica de PRÓGENES. Asi, en el ejemplo (18) se tiene la siguiente expresión de tipo BOOL con valor V:

```
(= (circunferencia (punto 0 1) 3)
   (conjunto (<> ?p (PUNTO 2))
     (= (distancia ?p (punto 0 1)) 3)))
```

Es una igualdad direccionada que se aplica siempre desde el tipo inferior hasta el supertipo. En el caso anterior, es natural el paso de considerar la transformación de una circunferencia expresada mediante un centro y un radio a un conjunto definido mediante una ecuación, mientras que el paso inverso resulta más artificial y solamente se aplica a casos muy específicos en resolución. En 2.4. se define el metapredicado = en PRÓGENES y su metaevaluación asociada que incluye este proceso de traducción según la jerarquía de tipos.

2.4 Metaevaluación

Otra transformación de expresiones formales es la metaevaluación. Este proceso utiliza las semánticas definidas para metafunciones y metapredicados para transformar cualquier expresión formal. Se necesita, por tanto, una forma de decidir la semántica a aplicar en el caso de que hayan varias definiciones para una misma metafunción (o metapredicado). En este sentido, la estructura jerárquica del conjunto TIPOS permite esta decisión aplicando la definición mas directamente aplicable para un caso específico de metaevaluación.

La metaevaluación de expresiones utiliza el proceso de traducción introducido en 2.3 promocionando, mediante la función TRADUCE, los argumentos hasta los tipos de la definición utilizada. Aparte de utilizar la jerarquía de tipos y la traducción de subexpresiones, la evaluación de expresiones se realiza utilizando una reescritura con estrategia

inner-most desde los niveles mas profundos de anidamiento, análogamente a sistemas como MATHEMATICA, [Wolfram 91], o como LISP, [Winston, Horn 89]. Es decir, la metaevaluación se trata de una proyección sobre un subespacio de expresiones más sencillas, en donde se definen específicos mecanismos de reconocimiento y resolución.

La decisión de utilizar metaevaluación en expresiones formales se realiza de acuerdo a las heurísticas generales de resolución descritas en el capítulo 3, en donde se integra metaevaluación junto con otras transformaciones generales en estados parciales de resolución de un problema. Por otra parte, las Bases de Conocimiento están estructuradas de modo que ocurre que una expresión tiene un nivel mayor de complejidad que su metaevaluación, de modo que se tiene una igualdad direccionada entre ambas, análogamente al caso de traducción, resultando innatural la sustitución inversa.

En PRÓGENES, la funcion METEVAL define esta transformación. METEVAL utiliza como argumento a una expresión <expr> y admite opcionalmente unas asignaciones de tipo <asigns>, cumpliéndose que <asigns> := () para el caso de expresiones sin metavariables libres. El resultado de esta acción es una nueva expresión METEVAL(<expr>, <asigns>) que, en general, tiene el mismo tipo que la expresión inicial, aunque puede ocurrir que tenga un tipo que sea un subtipo estricto del inicial.

2.4.1 Aplicación de semánticas de las Bases de Conocimiento

Existen metafunciones que se consideran predefinidas y cuya acción se define a nivel simbólico cuando utilizan como argumentos a metavariables. Así, METEVAL obtiene valores como:

```
METEVAL(exp (+ 2 3)) := (exp 5)
METEVAL((exp (+ 5 ?x)) (<> ?x REAL)) := (exp (+ 5 ?x))
```

Además, se define una metaevaluación recursiva en los argumentos que también se puede aplicar a expresiones que representan a objetos construidos según patrones para los tipos OBJETO-COMPUESTO. En tal caso, la semántica asociada es la función identidad aplicada sobre la acción recursiva de METEVAL en argumentos. Por ejemplo, se verifican las siguientes identidades:

```
METEVAL((punto (+ 3 4) (* (+ 4 -2) ?x)), (<> ?x REAL)) :=
(punto 7 (* 2 ?x))
```

Por otra parte, en las Bases de Conocimiento se definen específicamente otras metafunciones y metapredicados con una determinada semántica. En caso de que sólo haya una definición, tal como ocurre en (3), se utiliza la definición aplicada a la metaevaluación seguida de traducción en los argumentos. Así, se cumple:

```
METEVAL(plano-tangente
          (esfera (punto 0 0 0)
                  (distancia (punto 1 2 3) (punto 1 4 3)))
          (punto 2 0 0)) :=
(plano 2 0 0 -2)
```

en donde, en un primer paso, se ha aplicado METEVAL seguido de TRADUCE a los argumentos. Así, se producido la traducción de la subexpresión que representa a la esfera de centro $(0,0,0)$ y de radio 2, hasta el tipo SUPERFICIE, que es el argumento de la definición de plano-tangente en (3), mediante la acción de TRADUCE:

```
TRADUCE((esfera (punto 0 0 2) 2), SUPERFICIE)) :=
(superficie (funcion ((<> ?x REAL) (<> ?y REAL) (<> ?z REAL))
                     (+ (expt ?x 2) (expt ?y 2) (expt ?z 2) -4)))
```

y, en un segundo paso, se aplica la semántica de la metafunción plano-tangente que considera un plano definido por un punto y por el gradiente en dicho punto de la función asociada a la superficie. Es decir, se produce una metaevaluación de la expresión semántica asociada instanciando como argumentos los valores que han sido previamente metaevaluados y traducidos. En nuestro caso, la metaevaluación de la semántica asociada a plano-tangente obtiene el siguiente valor final:

```
METEVAL(plano
          (vector-normal
            (superficie
              (funcion ((<> ?x REAL) (<> ?y REAL) (<> ?z REAL))
                       (+ (expt ?x 2) (expt ?y 2) (expt ?z 2) -4))))
          (punto 2 0 0)) :=
(plano 2 0 0 -2)
```

Obsérvese que la definición *standard* del tipo PLANO se hace en las Bases de Conocimiento a partir de los cuatro coeficientes de la ecuación implícita asociada representando, por ejemplo, el plano $2x + 3y - z + 1 = 0$ mediante la expresión formal

(plano 2 3 -1 1). En el caso anterior se construye un plano a partir de un vector y un punto. Esto se debe a que en tal caso ocurre que plano es una metafunción. En este contexto, se verifican igualdades como:

```
METEVAL(plano (vector 1 2 3) (punto 0 1 -1)) :=
(plano 1 2 3 1)
```

permitiendo de este modo utilizar una representación múltiple de objetos, como tipos objeto o como metafunción, en la que solo una de ellas es la *standard* y corresponde al elemento de las Bases de Conocimiento cuyo primer elemento es obj y segundo elemento es el nombre de tal objeto.

Puede ocurrir que existan varias definiciones para una misma metafunción o metapredicado. En tal caso se aplica aquella que sea compatible con los tipos de los argumentos si sólo existe una posible. En el caso de que existan varias definiciones admisibles, se selecciona aquella que sea minimal con respecto a la jerarquía de tipos. Y, en caso de que puedan considerarse más de una definición minimal, entonces se toma la primera definición que aparece en las Bases de Conocimiento. En este último caso se supone que el diseñador de dichas Bases considera un orden de prioridades para la definición de metafunciones y metapredicados.

Así, en (4) y (5) se define la metafunción distancia aplicada a dos casos distintos para argumentos de tipo PUNTO o CONJUNTO, de modo que se utiliza la definición admisible en cada caso. Por ejemplo, se obtiene la identidad siguiente aplicando (4) para argumentos de tipo PUNTO:

```
METEVAL(distancia (punto 1 1) (punto 2 2)) := (sqrt 2)
```

El mismo mecanismo de metaevaluación se utiliza para aplicación de operaciones a distintas clases de argumentos. Así, se definen suma de números, de funciones, o de otro tipo de argumentos y se aplica la semántica admisible en cada caso. En este contexto, cabe señalar que la función METEVAL no conserva el tipo de la expresión inicial pudiendo descender en la jerarquía. Así, por ejemplo, si se define una suma general para números complejos para obtener expresiones de tipo COMPLEJO, el tipo de la expresión metaevaluada puede decrecer en la jerarquía de tipos. Así, se tienen las siguientes igualdades:

```
TIPO(- (* 2 i) (* (+ 1 1) i)) := COMPLEJO
METEVAL(- (* 2 i) (* (+ 1 1) i)) := 0
TIPO(0) := NATURAL
```

Como se decía anteriormente, en el caso de que existan varias definiciones admisibles para la metaevaluación de una expresión, se selecciona aquella que sea minimal con respecto a la jerarquía de tipos. Así, por ejemplo, para metaevaluar la expresión:

```
(tangente (esfera (punto 0 0 0) 1)
          (esfera (punto 3 0 0) 2)
          (punto 1 0 0))
```

se consideran las definiciones admisibles (6) y (7), que admiten como argumentos a los tipos SUPERFICIE y ESFERA y se selecciona (7) puesto que ESFERA < SUPERFICIE, de modo que el resultado de metaevaluar la siguiente expresión es V:

```
METEVAL(paralelo (- (punto 1 0 0) (centro (esfera (punto 0 0 0) 1)))
                 (- (punto 1 0 0) (centro (esfera (punto 3 0 0) 2))))
```

Por otra parte, también se puede aplicar metaevaluación en el caso de las funciones de acceso a componente que se realiza extrayendo la componente, si es posible. Así, por ejemplo, se tienen identidades tales como:

```
METEVAL((centro (elipse (punto 1 2) 3 4))) := (punto 1 2)
METEVAL((radio ?e), (<> ?e ESFERA)) := (radio ?e)
```

2.4.2 Tratamiento de la igualdad

La igualdad se define como un metapredicado que se puede aplicar a diferentes contextos considerando tipos específicos. Así, se puede definir una igualdad entre números complejos, entre funciones, entre conjuntos, etc. En general, la semántica asociada a la igualdad en tales casos refleja las actividades cognitivas utilizadas en cada dominio. Por ejemplo, la igualdad entre conjuntos se definió en (17) mediante:

```
(pred = ((conj1 CONJUNTO) (conj2 CONJUNTO))
      V
      (y (contenido conj1 conj2) (contenido conj2 conj1)))
```

y la igualdad de funciones en función del valor de las imágenes:

```
(pred = ((fun1 FUNCION) (fun2 FUNCION))
      (y (= (dominio fun1) (dominio fun2))
         (= (rango fun1) (rango fun2)))
      (para-todo (<> ?x OBJETO)
         (-> (pertenece ?x (dominio fun1))
             (= (imagen fun1 ?x) (imagen fun2 ?x)))))
```

que refleja la manera usual de demostrar propiedades en Teoría de Conjuntos. En este mismo contexto, se define un metapredicado de igualdad también para otros tipos como PLANO ya que la manera de representar un plano en el espacio mediante los cuatro coeficientes de una ecuación implícita no es única puesto que cualquier otra ecuación proporcional también representaría al mismo plano.

En PRÓGENES, se define una igualdad para determinados tipos como los señalados anteriormente considerando para el resto de los casos una igualdad recursiva en argumentos. Así, por ejemplo, se cumple la siguiente igualdad:

```
METEVAL((= (punto ?x 1) (punto ?y ?x)),
        ((<> ?x REAL) (<> ?y REAL))) :=
(y (= ?x ?y) (= 1 ?y))
```

METEVAL también utiliza a TRADUCE para aplicar la semántica de la igualdad a partir de dos expresiones formales. Por otra parte, METEVAL y TRADUCE establecen una igualdad entre una expresión y la acción de tales funciones, cumpliéndose para cualquier expresión <expr> que las siguientes expresiones tienen valor V:

```
METEVAL(= <expr> METEVAL(<expr>))
METEVAL(= <expr> TRADUCE(<expr>, <tipo>))
```

en donde <tipo> es un supertipo del tipo de <expr>.

Sin embargo, existe una direccionalidad, análoga a la que se considera en el demostrador de Boyer y Moore, [Boyer, Moore 88], entre una expresión y la acción mediante METEVAL o TRADUCE, resultando poco natural la transformación inversa. En PRÓGENES, esta direccionalidad se construye a partir de las Bases de Conocimiento especificando conceptos más complejos en función de otros elementales.

Por tanto, una vez desarrolladas las definiciones de semánticas en metafunciones y metapredicados, se puede considerar la igualdad como un metapredicado que relaciona expresiones formales mediante una acción no direccionada, estableciendo clases de

equivalencia entre expresiones relacionadas mediante igualdad. A estas expresiones se les puede aplicar reglas de inferencia específicas para el tratamiento de igualdad. Estas reglas pueden utilizar metaconocimiento, [Castells, Moriyón, Saiz 93, b], como ocurre en la aplicación de la regla de conservación de igualdad para condiciones específicas:

$$x = y \rightarrow A(x) = A(y)$$

o bien pueden utilizar reglas de cálculo que determinan el valor exacto, como ocurre a partir de un sistema de ecuaciones lineales con coeficientes complejos, en donde se utiliza una regla de cálculo que reconoce la naturaleza del sistema y calcula los valores de las incógnitas.

Capítulo 3

Heurísticas de resolución. Razonamiento basado en el objetivo.

En este capítulo se describe la heurística de razonamiento basado en el objetivo, que es aplicable en un contexto muy amplio en Resolución Automática de Problemas. Así mismo, se ilustran otras estrategias de resolución de problemas utilizadas en PRÓGENES, que hacen uso de técnicas de tipo deductivo y de cálculo. Se utilizan métodos que son usuales en dominios como Matemáticas y que se aplican a una gran variedad de problemas del estilo de los enunciados en la Introducción.

Los razonamientos en PRÓGENES utilizan teoremas, por un lado, como reglas de producción de sistemas expertos aplicándolas en modo *forwards*, o bien, se aplican en modo *backwards* intentando demostrar subobjetivos a partir de uno inicial. Se incorpora la novedad del razonamiento basado en el objetivo que se aplica a problemas en donde el objetivo es encontrar o demostrar la existencia de un objeto que verifica unas condiciones dadas, [Castells, Moriyón, Saiz 93, c]. Por otro lado, las técnicas de cálculo utilizadas en PRÓGENES representan conocimiento compilado en resolución de problemas que permite hallar el valor de unos objetos desconocidos cuando verifican unas condiciones predefinidas.

En lo que sigue se puntualizan todos estos aspectos. En primer lugar (3.1) se hace una descripción de las heurísticas de resolución utilizadas en PRÓGENES. En 3.2 se describe el formalismo de representación y transformación de estados construido a partir de tales heurísticas. Finalmente (3.3) se ilustra la heurística de razonamiento basado en el objetivo.

3.1 Heurísticas de resolución

En PRÓGENES se hace uso combinado de diversas heurísticas en el proceso de resolución. Fundamentalmente, estas técnicas son de naturaleza deductiva o de cálculo. Los procesos deductivos utilizan teoremas relevantes en el contexto de cada problema. Por otro lado, los procesos de cálculo utilizan heurísticas específicas para determinar el valor de objetos desconocidos sin necesidad de recurrir al uso de propiedades o teoremas.

Por otra parte, a partir de las definiciones de metafunciones y metapredicados, se construyen reglas de reescritura que transforman de manera inmediata expresiones formales en otras conceptuales mas simples. Por ejemplo, se tiene la siguiente regla de reescritura aplicable en el supuesto de tener dos circunferencias concéntricas:

```
Si (<> ?c1 CIRCUNFERENCIA) (<> ?c2 CIRCUNFERENCIA)
Entonces (concentricas ?c1 ?c2) := (= (centro ?c1) (centro ?c2))
```

Este proceso corresponde a la metaevaluación en PRÓGENES, que expresa igualdades direccionadas desde conceptos complejos hasta otros mas elementales. Por otro lado, en definiciones de funciones matemáticas es necesario utilizar cálculo marcadamente simbólico que simplifique subexpresiones siempre que sea posible. En este sentido, el proceso de simplificación es análogo al utilizado en MATHEMATICA, [Wolfram 91], en donde todas las subexpresiones se evalúan al máximo. Así, por ejemplo, se tiene la siguiente regla de reescritura asociada a la operación suma en un caso particular:

```
Si (<> ?a COMPLEJO) (<> ?b COMPLEJO)
Entonces (:= (* (+ ?a ?b (* 2 ?a)) (+ 1 1 (- i) (- i)))
             (+ (* (- 6 (* 6 i)) ?a) (* (- 2 (* 2 i)) ?b))
```

que representa el hecho de que $(a+b+2a)(1+1-i-i)$ se simplifica a $(6-6i)a+(2-2i)b$.

La aplicación de definiciones se realiza en primer lugar en determinados casos como los anteriores, mientras que en otros se utiliza un control sobre la aplicación de la definición semántica correspondiente de las Bases de Conocimiento. Esto se debe a que se utilizan en primer lugar otras heurísticas para determinar la solución, como es la aplicación de teoremas. En PRÓGENES, el control sobre la metaevaluación se realiza definiendo para una metafunción o metapredicado el argumento opcional `condicion-aplicacion` con valor `F` (que se toma por defecto), como en el caso de `(14)`, que retarda la aplicación de su semántica.

Como primera aproximación, se puede considerar un problema como una base de datos formada por unos hechos que se suponen conocidos (H) y por un objetivo o meta (M). Una base se puede representar en un formalismo de *sequents* como $H \vdash M$. A partir de una base se puede utilizar una regla de inferencia para obtener otra base. Una regla de inferencia tiene la siguiente forma:

$$\frac{H' \vdash M'}{H \vdash M}$$

y representa el hecho de que si se se resuelve $H' \vdash M'$, entonces se tiene resuelta la base inicial $H \vdash M$. Una regla de inferencia representa, por tanto, una transformación desde una base de datos hasta otra base que es una condición suficiente de demostración o de resolución. La aplicación de estas reglas de inferencia en PRÓGENES representa, por tanto, un proceso de búsqueda *bottom-up*, a diferencia del proceso *top-down* utilizado habitualmente en Lógica en donde se parte de unos axiomas y se aplican las reglas de inferencia en la dirección contraria, obteniendo hechos que se derivan a partir de los axiomas iniciales. En PRÓGENES, por tanto, se parte del enunciado del problema y se razona a partir de él de modo que se puede reducir el espacio de búsqueda de soluciones utilizando diferentes heurísticas según el tipo de problema.

Un teorema de la forma $A \rightarrow B$ se puede aplicar a una base $H \vdash M$ en sentido *forwards* si existe unificación entre H y A mediante unas asignaciones σ, añadiendo a dicha base los nuevos hechos definidos por B_σ, que resulta de la sustitución en B de las asignaciones σ. En este sentido, este razonamiento *forwards* a partir de hechos conocidos, (*Modus Ponens*), se expresa mediante la siguiente regla de inferencia:

$$\frac{A \rightarrow B, H, B_\sigma \vdash M}{A \rightarrow B, H \vdash M}$$

Cabe señalar que un teorema puede ser considerado como una regla de producción de un sistema experto, de modo que el razonamiento *forwards* corresponde a la acción de un motor de inferencia que se aplica a unos objetos y hechos iniciales. En PRÓGENES, las reglas que utiliza el motor de inferencia pueden ser también metarreglas que aplican metaconocimiento, [Castells, Moriyón, Saiz 93, b], acerca del dominio tratado, tal y como se describe en la tesis de Pablo Castells, [Castells 94].

Un teorema $A \rightarrow B$ también puede ser aplicado *backwards* a una base de datos $H \vdash M$. La aplicación de un teorema $A \rightarrow B$ se realiza en un sentido *backwards* cuando existe unificación entre B y M mediante unas asignaciones σ, de modo que se crea otra base de la forma $H \vdash A_\sigma$, siendo A_σ el resultado de la aplicación sobre A

de las asignaciones σ. En este sentido, el razonamiento *backwards* se puede expresar mediante la regla de inferencia:

$$\frac{A \rightarrow B, H \vdash A_\sigma}{A \rightarrow B, H \vdash M}$$

Además, cada vez que en PRÓGENES se dispone de un hecho
(= <expr> <expr-canonica>), se crea la regla de reescritura
(:= <expr> <expr-canonica>), sustituyéndose <expr> por <expr-canonica> en todas las expresiones. Así, por ejemplo, si (= ?x 5) es una expresión que aparece durante la resolución, entonces se transforma en (:= %x 5) y se sustituye ?x por 5 en todos lados.

El razonamiento *forwards* puede aplicarse para determinados problemas obteniendo la solución de una manera directa. Como sencilla ilustración, consideremos el problema 1 de la Introducción:

- Demostrar que si se tiene $f, g : \mathbb{R} \rightarrow \mathbb{R}$, donde f es derivable, g es un difeomorfismo y x es un punto crítico de f, entonces $g(x)$ es un punto crítico de $g \circ f \circ g^{-1}$.

cuya traducción en lenguaje formal PRÓGENES sería:

```
(demostrar
  (para-todo
    ((<> ?f FUNCION-REAL) (<> ?g DIFEOMORFISMO-REAL) (<> ?x REAL))
    (-> (y (derivable ?f)
           (punto-critico ?x ?f))
        (punto-critico (imagen ?g ?x)
                       (composicion ?g ?f (funcion-inversa ?g))))))
```

En este enunciado aparecen metapredicados como punto-critico y derivable, y metafunciones como composicion que tienen las semánticas usuales utilizadas en Matemáticas. Además, el tipo DIFEOMORFIMO-REAL representa a las funciones reales biyectivas que son derivables, así como su inversa.

A partir del enunciado del problema se consideran simplificaciones tales como la eliminación de la conjunción, y, en los hechos que se conocen, o de la implicación, ->, convirtiendo la parte izquierda en un hecho y la derecha en un objetivo. Estas

técnicas de simplificación se aplican a cualquier problema inmediatamente. Durante la resolución de este problema, se crea la base inicial siguiente:

```
H: (derivable ?f)
   (punto-critico ?x ?f)
M: (punto-critico (imagen ?g ?x)
                  (composicion ?g ?f (funcion-inversa ?g)))
```

Por otra parte, existen teoremas que se aplican en el cálculo de derivadas de funciones, y que expresan conocimiento direccionado utilizado por el matemático. Así, por ejemplo, si suponemos que $f, g, h : \mathbb{R} \rightarrow \mathbb{R}$ son funciones derivables y $a \in \mathbb{R}$, entonces se tienen las siguientes identidades:

$$\begin{aligned} (f \circ g)(a) &:= f(g(a)) \\ (f \circ g)'(a) &:= f'(g(a)).g'(a) \\ f \circ g \circ h &:= f \circ (g \circ h) \end{aligned}$$

que se utilizan como reglas de reescritura cuando se considera un valor al que se aplican, como en este caso es a (análogamente a como el matemático lo hace), y como igualdades no direccionadas cuando se escriben como identidades funcionales:

$$(f \circ g)' = (f' \circ g).g'$$

Si además g es un difeomorfismo, entonces:

$$\begin{aligned} (g^{-1})'(a) &:= (g'(g^{-1}(a)))^{-1} \\ (g^{-1})' &= \frac{1}{g'} \circ g^{-1} \\ g(g^{-1}(a)) &:= a \\ g^{-1}(g(a)) &:= a \end{aligned}$$

en donde la primera igualdad es el teorema de la función inversa. En PRÓGENES, estas propiedades se definen mediante los siguientes teoremas:

```
(teor (para-todo ((<> ?f FUNCION-REAL) (<> ?g FUNCION-REAL)
                  (<> ?a REAL))
        (-> (y (derivable ?f) (derivable ?g))
            (y (derivable (composicion ?f ?g))
               (:= (imagen (derivada (composicion ?f ?g)) ?a)
                   (* (imagen (derivada ?f) (imagen ?g ?x))
                      (imagen (derivada ?g) ?x))))
               (= (derivada (composicion ?f ?g))
                  (* (composicion (derivada ?f) ?g) (derivada ?g))))))

(teor (para-todo ((<> ?f FUNCION-REAL) (<> ?g FUNCION-REAL)
                  (<> ?h FUNCION-REAL))
        (:= (composicion ?f ?g ?h)
            (composicion ?f (composicion ?g ?h)))))

(teor (para-todo ((<> ?g DIFEOMORFISMO-REAL) (<> ?a REAL))
        (y (:= (imagen (derivada (funcion-inversa ?g)) ?a)
               (expt (imagen (derivada ?g)
                             (imagen (funcion-inversa ?g) ?a))
                     -1))
           (= (derivada (funcion-inversa ?g))
              (composicion (/ 1 (derivada ?g))
                           (funcion-inversa ?g))))))

(teor (para-todo ((<> ?g FUNCION-REAL) (<> ?a REAL))
        (-> (biyectiva ?g)
            (y (:= (imagen ?g (imagen (funcion-inversa ?g) ?a)) ?a)
               (:= (imagen (funcion-inversa ?g) (imagen ?g ?a))
                   ?a)))))
```

de modo que aplicándolos en modo *forwards*, junto con el teorema (16), a la base anterior dan lugar a:

```
H: (derivable ?f)
   (= (imagen (derivada ?f) ?x) 0)
M: (= (* (imagen (derivada ?g) (imagen ?f ?x))
         (imagen (derivada ?f) ?x)
         (/ 1 (imagen (derivada ?g) ?x)))
      0)
```

Finalmente, a partir del segundo hecho contenido en H, y de acuerdo al mecanismo de creación de reglas de reescritura explicado anteriormente se deduce la regla de reescritura (:= (imagen (derivada ?f) ?x) 0) que se aplica en el objetivo M obteniéndose (= 0 0) que obtiene V, con lo que el problema estaría resuelto.

Por otra parte, la eliminación de conectores lógicos para simplificar una base de datos da lugar a varias demostraciones alternativas en determinados casos tales como la eliminación de la conjunción, y, en las metas M. Asi, por ejemplo, si tenemos que demostrar que $A \rightarrow (B \wedge C)$, se puede conseguir demostrando $A \rightarrow B$ en primer lugar y $(A \wedge B) \rightarrow C$ a continuación, o bien, $A \rightarrow C$ primero y $(A \wedge C) \rightarrow B$ después. Entonces, si por ejemplo fuera cierto que $B \rightarrow C$, la primera alternativa se aplicaría directamente. En Matemáticas, este hecho se constata resultando a menudo más conveniente alguna de las dos posibilidades. De hecho, y para facilitar la tarea de resolución para el alumno, un profesor (razonable) podría proponer en un examen el problema anterior $A \rightarrow (B \wedge C)$ indicando que primero se demuestre B.

Una heurística importante en PRÓGENES es el razonamiento basado en el objetivo, que se explica en detalle más adelante, (3.3). Se aplica a problemas en los que se busca un objeto o se quiere demostrar su existencia. Esta heurística considera el objetivo de un determinado problema y razona a partir de el como si fuera un hecho conocido para derivar consecuencias que permitan determinar los valores posibles del objeto buscado. De hecho, el Algoritmo de Expansión en Resolución de Problemas (AERP), que se describe en 4.4, se puede considerar también como un caso particular de razonamiento basado en el objetivo.

Durante la resolución de un problema también se utilizan técnicas de cálculo que permiten resolver determinados subproblemas de acuerdo a patrones de solubilidad, como se describe mas adelante en 4.2. Estas técnicas son equivalentes a aplicar reglas del tipo siguiente, en donde se supone que todas las metavariables tienen tipo REAL y sólo ?x y ?y son desconocidas:

```
Si (= (+ (* ?a ?x) (?b ?y)) ?c)                                    (19)
   (= (+ (* ?d ?x) (?e ?y)) ?f)
   (no (= (- (* ?a ?e) (* ?b ?d)) 0))
Entonces
   (:= ?x (/ (- (* ?b ?f) (?c ?e)) (- (* ?a ?e) (* ?b ?d))))
   (:= ?y (/ (- (* ?a ?f) (?d ?c)) (- (* ?a ?e) (* ?b ?d))))
```

que representa el hecho de que el sistema de ecuaciones $ax + by = c, dx + ey = f$ con las incógnitas x e y, y bajo la condición $ae - bd \neq 0$ se simplifica a las igualdades

$x := \frac{df - ce}{ae - bd}, y := \frac{af - dc}{ae - bd}$.

En realidad estas reglas se aplican en primer lugar a cualquier conjunto de expresiones en donde se pueda determinar directamente el valor canónico de las metavariables involucradas.

Por otro lado, la negación en PRÓGENES se define mediante el metapredicado no. Se utilizan reglas sencillas como (no (no <expr>)) := <expr>, pero que en la práctica no se utilizan. En PRÓGENES no se ha tratado la negación a un nivel distinto del trivialmente simbólico porque no es necesario en la resolución de los problemas considerados. Evidentemente el matemático utiliza en ocasiones razonamiento por reducción al absurdo de forma heurística. El estudio de las condiciones y la forma en que se lleva a cabo este proceso sería sin duda de interés.

Análogamente, se considera el metapredicado o para definir la disyunción. Sin embargo, en PRÓGENES tampoco se ha tratado la disyunción por ser poco común en la resolución de problemas. Evidentemente se puede tratar creando bases hijas de una dada cuando el objetivo tiene la forma $p \vee q$ y demostrar, alternativamente, p suponiendo cierto $\neg p$, o bien q suponiendo cierto $\neg q$. Así mismo, si $p \vee q$ es una hipótesis, se podrían crear dos subestados suponiendo p cierto en el primero y $\neg p \wedge q$ en el segundo, y demostrarlos ambos; alternativamente se puede invertir el papel de p y q.

Finalmente, la inducción no ha sido tratada en PRÓGENES. Sin embargo se verifican los axiomas de Peano, [Ebbinghaus, Flum, Thomas 84]: el primer axioma, $0 \in \mathbb{N}$, se cumple porque existe la expresión 0 con tipo NATURAL; el segundo, $\forall n \in \mathbb{N}, n + 1 \neq 0$ es consecuencia de los axiomas de ordenación de NATURAL; el tercero, $\forall n, m \in \mathbb{N}$, $n + 1 = m + 1 \rightarrow n = m$ se demuestra automáticamente en PRÓGENES; finalmente el axioma de inducción se podría incluir en la Base de Conocimiento correspondiente de la forma:

```
(teor (-> (y (sust ?m 0 ?prop)
             (para-todo (<> ?m NATURAL)
                (-> ?prop (sust ?m (+ ?m 1) ?p))))
          (para-todo (<> ?m NATURAL) ?prop)))
```

donde (sust ?k ?l ?prop) sustituye ?k por ?l en ?prop. Obsérvese que aunque la expresión anterior no es una expresión PRÓGENES, se puede extender el Lenguaje Formal para incluirla y el mecanismo de unificación también admite una extensión trivial

que permite su tratamiento por el motor de inferencia, permitiendo hacer deducción mediante predicados de orden superior.

3.2 Estados. Mecanismo de resolución.

La aplicación de heurísticas a bases de datos mediante reglas de inferencia como las descritas en la sección anterior da lugar, por tanto, a otras nuevas bases o modifican las iniciales. Se produce modificación cuando se añade información, mientras que si se deben sustituir datos, entonces se crea una base nueva. El proceso de resolución consiste en una búsqueda a través de las bases de datos obtenidas de este modo de una base que contenga como objetivo `M: V`. Surge, por tanto, la necesidad de considerar una estructura de árbol de bases en el que se permita *backtracking* en la búsqueda en los casos en los que sean aplicables varias heurísticas.

Por tanto, el estado de resolución de un problema es un árbol, y esta formado por nodos que son bases de datos PRÓGENES. Por otra parte, un estado siempre tiene asociada una base activa, que está en una hoja del árbol. Así mismo, cada base contiene expresiones formales PRÓGENES representadas de forma modular en una serie de componentes. Esta representación se realiza al estilo de MUSCADET, [Pastre 89], en donde las relaciones y objetos se organizan en una estructura de módulos. A continuación se describen todos estos procesos en PRÓGENES.

3.2.1 Problema 2

Como ilustración del tipo de transformaciones que se llevan a cabo en estados y de la estructura modular de las bases en el proceso de resolución, se describe la evolución que experimenta la base activa en el caso particular del problema 2 de la Introducción:

- Demostrar que la función $f(x) = x^3 - 6x^2 + 15x - 2$ es creciente.

cuya expresión formal asociada es:

```
(demostrar
  (creciente (funcion (<> ?x REAL)
                (+ (expt ?x 3) (* -6 (expt ?x 2)) (* 15 ?x) -2))))
```

Inicialmente se crea un estado con una sola base, la cual esta formada por un solo elemento MM (módulo de metas), que representa al objetivo a demostrar. En realidad, esta base también contiene un módulo de objetos buscados, que denotaremos por MOB, y que en este caso contiene a la lista () porque es una demostración y no se busca ningún objeto explícitamente. En el caso de un problema de búsqueda de un objeto, el valor de MOB es la lista de las expresiones formales que se buscan. En general, una base esta formada por varios módulos, como se describe mas adelante en 3.2.2. La base inicial que se obtiene, aparte del módulo MOB: (), contiene:

```
MM: (creciente (funcion (<> ?x REAL)
                  (+ (expt ?x 3) (* -6 (expt ?x 2)) (* 15 ?x) -2)))
```

Existen dos teoremas cuya aplicación iterada *backwards* establece una condición suficiente para resolver el problema. Uno de ellos afirma que una función que sea estrictamente creciente, en particular es creciente, y otro afirma que una condición suficiente para que una función sea estrictamente creciente (si la función es derivable) es que la derivada sea positiva:

```
(teor (para-todo (<> ?f FUNCION-REAL)
        (-> (estric-creciente ?f) (creciente ?f))))

(teor (para-todo (<> ?f FUNCION-REAL)
        (-> (y (derivable ?f)
               (> (derivada ?f) 0))
            (estrict-creciente ?f))))
```

La aplicación de estos dos teoremas mediante razonamiento *backwards* según la regla de inferencia descrita en la sección anterior crea una base hija y una nieta de la inicial en el árbol que representa al estado parcial de resolución. Por tanto, la base activa tras esta doble transformación de estados, denominada RAZONA-B (razona *backwards*), tiene la forma:

```
MM: (y (derivable (funcion (<> ?x REAL)
                     (+ (expt ?x 3) (* -6 (expt ?x 2)) (* 15 ?x) -2)))
       (> (derivada
            (funcion (<> ?x REAL)
              (+ (expt ?x 3) (* -6 (expt ?x 2)) (* 15 ?x) -2)))
          0))
```

Son posibles varias transformaciones aplicadas a bases (ver 3.2.3). Sólo modifican la estructura geométrica de árbol de un estado aquellas que crean una casuística de resolución, es decir en los casos en que la heurística da lugar a varias posibles demostraciones. Las demás transformaciones solo modifican la base activa. En este sentido, existe una transformación denominada `CANONIZA` que comprende varias simplificaciones tales como eliminación de determinados conectores lógicos y creación de constantes, o aplicación de las reglas de reescritura derivadas de la metaevaluación en metafunciones y metapredicados.

En el caso anterior, la acción de `CANONIZA` transforma la base de datos en la siguiente base (en donde `MO`, módulo de objetos, contiene objetos que aparecen durante la resolución y que se pueden considerar como constantes):

```
MO: (<> ?x REAL)
MM: (> (+ (* 3 (expt ?x 2) (* -12 ?x) 15)) 0)
```

puesto que mediante metaevaluación se verifica:

```
(derivable
  (funcion (<> ?x REAL)
    (+ (expt ?x 3) (* -6 (expt ?x 2)) (* 15 ?x) -2))) := V

(> (derivada (funcion (<> ?x REAL)
                (+ (expt ?x 3) (* -6 (expt ?x 2)) (* 15 ?x) -2)))
   0) :=
(para-todo (<> ?x REAL) (> (+ (* 3 (expt ?x 2) (* -12 ?x) 15)) 0))
```

y además `CANONIZA` ha eliminado la conjunción, y, de `MM` puesto que su primer argumento sería `V` tras aplicar la regla asociada al metapredicado `derivable`. Otra acción de `CANONIZA` es la creación de constantes para el caso en que el objetivo sea una expresión construida mediante el constructor lógico `para-todo`. Así, en el caso anterior se ha creado la constante `(<> ?x REAL)` que se considera como un objeto conocido en la base de datos.

Obsérvese que la aplicación de semánticas en algunos metapredicados como `derivable` se realiza de manera inmediata. Sin embargo, en otros casos, como el de `creciente` o `estric-creciente`, sólo se aplica con posterioridad a la aplicación de teoremas, para analizar las soluciones que éstos proporcionan en primer lugar. Esto se

debe a que en las Bases de Conocimiento, derivable tiene `condicion-aplicacion V`, mientras que para `creciente` y `estric-creciente` su valor es `F`.

Finalmente, se pueden aplicar patrones de solubilidad a la base activa anterior mediante una función de reducción o simplificación denominada `REDUCE`, descrita en 4.2.3, que encuentra que el objetivo `MM` se puede reducir a `(pertenece ?x R)` (el objetivo es una inecuación con un polinomio de segundo grado con raíces complejas). Como sabemos que `(<> ?x REAL)`, el motor de inferencia crea el hecho `(pertenece ?x R)` en `MO`, que es el objetivo, con lo que se sustituye la meta por `V`.

3.2.2 Representación de estados

Una base de datos PRÓGENES, o simplemente base, es un conjunto de módulos que contienen objetos y relaciones expresados en Lenguaje Formal PRÓGENES. Existen módulos que son globales durante la resolución de un problema, tales como el módulo de teoremas activos (*MT*), y cuyo valor se puede utilizar para otros problemas que utilicen conceptos similares. *MT* se construye mediante una activación de los teoremas de las Bases de Conocimiento que involucran a los tipos y relaciones que aparecen en un problema [1]. Otro módulo global es el de las definiciones activas de las Bases de Conocimiento (*MD*) que se seleccionan de una forma similar al caso anterior. Otros módulos son locales a las bases de datos.

Una base esta formada por los siguientes módulos locales:

- Módulo de Objetos (MO), que representa a objetos que aparecen y se consideran constantes lógicas durante la resolución. Son objetos canónicos (sin metavariables libres) o expresiones de asignación de tipo.
- Módulo de Hechos (MH), el cual contiene hechos conocidos y expresa relaciones entre objetos de MO. En realidad los hechos conocidos en cualquier instante son MH y *MT* y ambos se utilizan para hacer deducción mediante el motor de inferencia a partir de MO. Una parte de los hechos MH es utilizada por el motor de inferencia como reglas de deducción y el resto se consideran como hechos para comprobar las condiciones de las reglas.
- Módulo de Objetos Buscados (MOB), que contiene los objetos que se quieren encontrar. Por tanto, si el problema es la búsqueda de un objeto, entonces MOB está formado por expresiones de asignación de tipo que contienen las metavariables cuyo

[1] En realidad *MT* contiene también metarreglas que describen metaconocimiento en Resolución de Problemas.

valor hay que calcular. Si el problema es una demostración entonces MOB tiene valor (), aunque si es una demostración existencial, entonces MOB vale (existe <expresion-asignacion-tipo>), en donde <expresion-asignacion-tipo> contiene los objetos de los que se va a demostrar la existencia.

- Módulo de Metas (MM), que contiene el objetivo de una base de datos. Esta constituido por una expresión formal de tipo BOOL construida a partir de objetos de MO y de MOB.

- Módulo de Objetos Pistas (MOP), que está formado por objetos con valor desconocido que aparecen durante la resolución de un problema. Son expresiones de asignación de tipo obtenidas a partir de MOB y aparecen en la aplicación del algoritmo de expansión AERP, como se ilustra en 4.4.

- Módulo de Pistas (MP), el cual contiene condiciones necesarias para que se verifique el objetivo MM. Este módulo contiene expresiones formales de tipo BOOL que resultan de la aplicación de la heurística de razonamiento basado en el objetivo, como se describe en 3.3.

Un estado de resolución es un árbol de bases en donde existe una base activa en un extremo del árbol. Cabe señalar que en un estado el parentesco entre una base y sus hijas puede ser de tipo AND o OR. Se tiene un parentesco de tipo AND cuando para resolver una base hay que resolver todas las hijas (ordenadamente desde la primera hasta la última), y de tipo OR cuando basta con resolver alguna de ellas.

Si <expr> representa el enunciado formal de un problema, el estado inicial esta formado por una sola base con un solo módulo (MM) y que tiene la forma MM: <expr>. A partir de este estado, y como se describe en el siguiente apartado, se aplican transformaciones que generan nuevos estados hasta encontrar uno que esté resuelto. Un estado está resuelto cuando la base activa contiene MM: V y tiene parentesco OR con su base padre, o cuando todas las hijas con parentesco AND con la base padre están resueltas.

3.2.3 Funciones de transformación de estados

Los estados se modifican mediante funciones de transformación aplicadas a la base activa. Las transformaciones de bases utilizan reglas de inferencia para modificar un estado. Estas reglas de inferencia son las ilustradas en 3.1 y pueden modificar un estado cambiando el contenido de la base activa, creando varias bases hijas y activando una de ellas o suprimiendo la base activa y activando una nueva base en la estructura de

árbol del estado, considerando prioritariamente bases hermanas y, si no existen, la base padre.

Las transformaciones que se realizan pueden ser estáticas o dinámicas. Si no modifican la geometría de un estado, sino que simplemente cambian el valor de la base activa, se tienen transformaciones estáticas. En estas se añade información y corresponden a transformaciones que se realizan siempre que es posible; en caso contrario se habla de transformaciones dinámicas, en las que se debe sustituir información potencialmente útil por otra nueva creándose nuevas bases. En los casos en los que se tenga una casuística de resolución con necesidad de tanteo de posibilidades se utilizan las transformaciones dinámicas.

Las transformaciones estáticas en estados son:

- `SIMPLIFICA-LOGICA`. Canoniza la estructura de una base eliminando conectores lógicos que aparecen como primer elemento de expresiones formales en determinados módulos. Así, suprime el `para-todo` en `MM` creando un nuevo objeto en `MO` y suprime el `existe` en `MH` y `MM` creando objetos en `MO` y `MOB` respectivamente. Estas técnicas son análogas a las utilizadas por Bledsoe en sus heurísticas de *split-reduce* [Bledsoe 71]. En PRÓGENES esta simplificación parte de un cálculo de *sequents* con una semántica mas rica.

- `METEVAL`. Esta transformación aplica en una base metaevaluación a las expresiones formales de `MH`, `MM` y `MP`. Es decir, hace que actúe sobre ellas la función `METEVAL`, descrita en 2.4.1, considerando las definiciones activas en metafunciones y metapredicados (son las que tienen `condicion-aplicacion V`)

- `REDUCE`. Esta función, a partir de una base de datos, reduce y simplifica conjuntos de expresiones de `MH`, `MM` y `MP` de acuerdo a patrones de solubilidad, descritos en 4.2, que definen las soluciones a determinados subproblemas de resolución directa, tales como soluciones a ecuaciones, inecuaciones, etc. Así, por ejemplo, sustituye un sistema de dos ecuaciones con dos incógnitas por las reglas de reescritura que dan las soluciones, como ocurría en `(19)`.

- `AMPLIA-MH`. Esta transformación aplica razonamiento *forwards* utilizando como argumentos del motor de inferencia a las reglas de `MH` y `*MT*`, y a los objetos `MO` junto con el resto de hechos de `MH`, y añade las expresiones resultantes a `MH`. El motor de inferencia construido en PRÓGENES utiliza unificación y se aplica a objetos teniendo en cuenta la jerarquía de tipos.

La función que aplica todas las transformaciones estáticas a una base se denomina

CANONIZA. Si consideramos dos bases B1 y B2 tales que CANONIZA(B1) := B2, entonces se verifica que:

```
SIMPLIFICA-LOGICA(B2)=METEVAL(B2)=REDUCE(B2)=AMPLIA-MH(B2) := B2
```

es decir, CANONIZA integra a todas las transformaciones estaticas. Así, por ejemplo, el problema 1 se resuelve únicamente aplicando las transformaciones que contiene CANONIZA. Por otro lado, cabe señalar que otra transformación que se realiza inmediatamente consiste en la aplicación de la heurística de razonamiento basado en el objetivo mediante la transformación RAZONA-BO, como se ilustra en 3.3.3.

Por otra parte, las transformaciones dinámicas modifican la estructura geométrica de un estado creando bases hijas a partir de la base activa. Son las siguientes:

- RAZONA-B. Utiliza razonamiento *backwards* en el objetivo MM a partir de teoremas. Así, en el problema 2 se utiliza esta transformación. Crea una base hija que contiene como MM las condiciones suficientes expresadas por el teorema para obtener el objetivo inicial.

- AERP. Aplica el algoritmo de expansión a los objetos buscados MOB de una base de datos. Esta expansión se realiza de acuerdo a las definiciones de los tipos OBJETO definidos en las Bases de Conocimiento. La aplicación de esta transformación se describe en 4.4.

- ELIMINA-Y. Se aplica cuando el objetivo es una conjunción. Así, si
  MM: (y <expr1> <expr2>), intenta demostrar una de las dos siguientes alternativas:

  - B1:(MM:<expr1>) y B2:(MH:<expr1>, MM:<expr2>)
  - B2:(MM:<expr2>) y B2:(MH:<expr2>, MM:<expr1>)

3.2.4 Esquema de flujo en resolución

Partiendo de las descripciones de estados y transformaciones de los dos apartados anteriores, en este apartado se describe el esquema general de flujo de resolución en PRÓGENES. Para ilustrar estos conceptos, se utiliza como ejemplo el problema 3 de la Introducción:

- Demostrar que si $f, g : \mathbb{R} \rightarrow \mathbb{R}$, f es inyectiva y $g(x) = x^3 + 3x^2 + 3x + 2$, entonces $g \circ f$ es inyectiva.

cuya traducción en lenguaje formal es:

```
(demostrar
  (para-todo ((<> ?f FUNCION-REAL) (<> ?g FUNCION-REAL))
     (-> (y (inyectiva ?f)
            (:= ?g (funcion (<> ?x REAL)
                            (+ (expt ?x 3) (* 3 (expt ?x 2)) (* 3 ?x) 2))))
         (inyectiva (composicion ?g ?f)))))
```

Para resolver un problema, en PRÓGENES se utilizan las funciones de transformación estáticas aplicándolas a las base activa de un estado. Una vez aplicadas las estáticas, las dinámicas generan la estructura de árbol creando a partir de una base otras hijas y modificando la posición de la base activa inicial. Cada vez que se utiliza una transformación dinámica, se considera como nueva base activa la primera resultante de tal transformación. Por otra parte, la elección de transformación se realiza según un orden que privilegia determinados métodos de inferencia sobre otros. Así, por ejemplo, en un problema de búsqueda del valor de un objeto, AERP se activa antes que RAZONA-B, mientras que en un problema de demostración primero se aplican teoremas mediante RAZONA-B.

En el ejemplo anterior, las transformaciones estáticas dan lugar a un estado formado por una sola base B que tiene la forma:

```
MO:  (<> ?f FUNCION-REAL)
MH:  (inyectiva ?f)
MOB: ()
MM:  (inyectiva
        (composicion
           (funcion (<> ?x REAL)
                    (+ (expt ?x 3) (* 3 (expt ?x 2)) (* 3 ?x) 2)))
           ?f))
```

en donde se ha sustituido ?g por su valor canónico aplicando la regla de reescritura del enunciado. Esta base B verifica, por tanto, que CANONIZA(B) := B. Obsérvese que se ha utilizado SIMPLIFICA-LOGICA y REDUCE mientras que AMPLIA-H no se ha

podido utilizar como ocurría en el problema 1. A continuación se puede aplicar la función de transformación dinámica RAZONA-B, en el supuesto que exista en las Bases de Conocimiento el teorema que afirma que la composición de dos funciones inyectivas es también inyectiva:

```
(teor (para-todo ((<> ?h FUNCION) (<> ?k FUNCION))
        (-> (y (inyectiva ?h) (inyectiva ?k))
            (inyectiva (composicion ?h ?k)))))
```

La aplicación del razonamiento *backwards* se hace considerando las asignaciones σ(?g/?h, ?f/?k) puesto que en la jerarquía de tipos PRÓGENES se tiene que FUNCION-REAL < FUNCION, y se obtiene un estado formado por dos bases: la anterior y su hija que queda activada y se transforma de nuevo mediante CANONIZA en la base siguiente (en donde se omite el resto de módulos por no ser relevantes para lo que sigue):

```
MOB: ()
MM:  (inyectiva (funcion (<> ?x REAL)
                  (+ (expt ?x 3) (* 3 (expt ?x 2)) (* 3 ?x) 2)))
```

En esta base, se utiliza el teorema según el cual toda función estrictamente creciente es inyectiva:

```
(teor (para-todo (<> ?f FUNCION-REAL)
        (-> (estric-creciente ?f) (inyectiva ?f))))
```

y se procede análogamente al problema 2 utilizando el hecho de que esta función tiene derivada positiva.

Otra manera de resolver el problema, en el caso de que las Bases de Conocimiento no contengan los teoremas anteriores, con lo que no se podría aplicar RAZONA-B, consiste en considerar la base inicial y aplicar la definición de inyectiva en MM. El metapredicado inyectiva tiene condicion-aplicacion F, con lo que su metaevaluación en MM no se realiza de manera inmediata y sólo cuando no sea posible aplicar transformaciones dinámicas (que en este caso se cumple puesto que RAZONA-B no puede utilizar los teoremas anteriores y AERP no se aplica para el caso de problemas de demostración). Por tanto, la base inicial se sustituye por:

```
MO:  (<> ?f FUNCION-REAL) (<> ?x REAL) (<> ?y REAL)
MH:  (para-todo ((<> ?x REAL) (<> ?y REAL))
       (-> (= (imagen ?f ?x) (imagen ?f ?y)) (= ?x ?y)))
     (= (+ (expt (imagen ?f ?x) 3) (* 3 (expt (imagen ?f ?x) 2))
           (* 3 (imagen ?f ?x)) 2)
        (+ (expt (imagen ?f ?y) 3) (* 3 (expt (imagen ?f ?y) 2))
           (* 3 (imagen ?f ?y)) 2))
MOB: ()
MM:  (= ?x ?y)
```

A esta base se le puede aplicar RAZONA-B puesto que el objetivo MM unifica con la conclusión del segundo hecho de MH, con lo que se crea una base hija que contiene como objetivo (= (imagen ?f ?x) (imagen ?f?y)). Obsérvese que en este caso ha sido uno de los hechos de MH el que se ha utilizado en el razonamiento *backwards*, a diferencia de los utilizados hasta ahora que utilizaban teoremas como regla de deducción. En la selección de reglas de deducción para aplicar razonamiento *backwards* se eligen en primer lugar las de los hechos MH. Además, en este caso, por razones de eficiencia no se consideran todos los teoremas cuya conclusión es de la forma (= ?x ?y), sino sólo las reglas de MH.

A continuación, la función REDUCE, según las técnicas de solubilidad que se describen en 4.2.3, se da cuenta de que el tercer hecho de MH, que es una ecuación polinómica de grado 3, es una ecuación resoluble en (imagen ?f ?x), que es uno de los miembros de la igualdad del objetivo, y la resuelve obteniendo que
(= (imagen ?f?x) (imagen ?f ?y)) que es el objetivo MM. Por tanto, modifica en la base activa el objetivo a MM: V, de modo que el problema queda resuelto con valor V.

Por tanto la generación de la estructura de árbol de estados se efectúa, tras la generación del estado inicial, ejecutando en el orden que se indica las siguientes tareas:

```
ESTADO <- CANONIZA(ESTADO)
ESTADO <- RAZONA-BO(ESTADO)
ESTADO <- AERP(ESTADO)
ESTADO <- ELIMINA-Y(ESTADO)
ESTADO <- RAZONA-B(ESTADO)
ACTIVA-DEFS
```

en donde RAZONA-BO y AERP aplican la heurística de razonamiento basado en el objetivo y el algoritmo de expansión, respectivamente, y se describen en 3.3 y 4.4.

Por otro lado, ACTIVA-DEFS añade reglas de reescritura que provienen de las definiciones de metafunciones y metapredicados que no se utilizaron en primer lugar porque tenían `condicion-aplicacion F`. Cabe señalar que parte de las definiciones se activan inmediatamente al cargarse las Bases de Conocimiento y el resto es activado por ACTIVA-DEFS pero su aplicación es llevada a cabo mediante la transformación METEVAL.

Cada vez que alguna de ellas efectúa algun cambio en el estado global, se vuelve a ejecutar la primera tarea. De esta manera el proceso se lleva a cabo de acuerdo con un criterio de asignación de prioridades a reglas que las clasifica por grupos y se actualizan automáticamente al variar las Bases de Conocimiento.

3.3 Razonamiento basado en el objetivo

En esta sección se describe la heurística de razonamiento basado en el objetivo que es aplicable en PRÓGENES a un amplio espectro de problemas,
[Castells, Moriyón, Saiz 93, c]. Son problemas en que se busca un objeto o se quiere demostrar su existencia. La heurística consiste en asumir las condiciones que el objeto buscado debe cumplir en un problema y, de acuerdo a ellas, extraer condiciones necesarias que debe verificar dicho objeto, de modo que se restringen los valores posibles que puede tomar. De este modo, si se puede probar que sólo determinados valores son posibles, basta con determinar cuáles de ellos verifican las condiciones requeridas.

En los siguientes apartados se ilustra esta heurística en un contexto general de resolución de problemas (3.3.1), se introducen ejemplos de su aplicación en PRÓGENES (3.3.2 y 3.3.4) y se describe su funcionamiento en el marco de PRÓGENES.

3.3.1 Descripción de la heurística

Esta heurística se basa en ideas de resolución muy comunes entre matemáticos,
[Polya 65]. La principal característica de esta heurística es actuar como si se conociese el objeto que se esta buscando y razonar sobre el hasta que se deduce su valor. Por tanto, se consideran las propiedades que el objeto buscado debe satisfacer como si fueran hechos conocidos y se hacen deducciones a partir de estos hechos y otros conocidos aplicando reglas de inferencia.

En Matemáticas existen tipos de problemas en los que se puede aplicar este razonamiento para determinar los valores de incógnitas. Así, por ejemplo, en problemas de maximización de Cálculo Infinitesimal se determinan los puntos críticos como can-

didatos y después se comprueba cuales de ellos tienen la segunda derivada negativa. En un contexto mas general, también se puede aplicar a problemas de maximización condicionada utilizando los multiplicadores de Lagrange.

Así mismo, esta heurística corresponde a la manera de encontrar soluciones a problemas en donde aparecen determinadas ecuaciones que se pueden resolver, bien sean numéricas, diferenciales o en derivadas parciales. Así, por ejemplo, supongamos una cuerda infinita con una posición y velocidad iniciales. Si el problema fuera calcular la posición de dicha cuerda en cualquier instante, este problema se podría resolver considerando que toda cuerda tiene una función de elongación asociada que verifica la ecuación asociada a la cuerda, que es la ecuación de ondas $\frac{\partial^2 u}{\partial t^2}(x,t) = \frac{\partial^2 u}{\partial x^2}(x,t)$ y resolverla aplicando razonamiento basado en el objetivo. Es decir, se tendría un problema $P1$ consistente en encontrar una función $u(x,t) \in C^\infty(\mathbb{R}^2)$ que verifica las condiciones:

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2}(x,t) &= \frac{\partial^2 u}{\partial x^2}(x,t) \\ u(x,0) &= u_0(x) \\ \frac{\partial u}{\partial t}(x,0) &= u_1(x) \end{aligned}$$

en donde $u_0(x)$ y $u_1(x)$ son unas funciones dadas que son $C^\infty(\mathbb{R})$ y representan la posición y velocidad inicial respectivamente de la cuerda.

Este problema puede ser resuelto suponiendo que existe la solución $u(x,t)$ y derivando hechos a partir de ello. Así, un teorema asegura que si $u(x,t) \in C^\infty(\mathbb{R}^2)$ y verifica la ecuación de ondas $\frac{\partial^2 u}{\partial t^2}(x,t) = \frac{\partial^2 u}{\partial x^2}(x,t)$, entonces $\exists v_1, v_2 \in C^\infty(\mathbb{R}^2)$ tales que $u(x,t) = v_1(x+t) + v_2(x-t)$. Por tanto, si suponemos que $u(x,t)$ verifica el problema $P1$, entonces se deduce aplicando el teorema anterior que:

$$\begin{aligned} v_1(x) + v_2(x) &= u_0(x) \\ v_1'(x) - v_2'(x) &= u_1(x) \\ u(x,t) &= v_1(x+t) + v_2(x-t) \end{aligned}$$

y resolviendo la ecuación diferencial ordinaria que resulta al despejar v_1 en la primera ecuación, se obtiene que:

$$u(x,t) = \frac{1}{2}(u_0(x+t) + u_0(x-t)) + \frac{1}{2}\int_{x-t}^{x+t} u_1(x)dx$$

De este modo, la solución, en caso de existir, debe verificar esta fórmula. El siguiente paso en la heurística consiste en comprobar que la función $u(x,t)$ definida de esta forma verifica el problema $P1$. Por tanto, se comprueba que la función así definida es $C^\infty(\mathbb{R}^2)$ y que verifica la ecuación de ondas y las condiciones iniciales.

Otro ejemplo de aplicación de esta heurística es la resolución del problema de encontrar la temperatura en función del tiempo de un anillo circular con una temperatura inicial. Este problema tiene asociada una ecuación en derivadas parciales que es la ecuación del calor $\frac{\partial u}{\partial t}(x,t) = \frac{\partial^2 u}{\partial x^2}(x,t)$. Por tanto, la resolución de este problema consiste en resolver el problema $P2$ de encontrar una función $u(x,t) \in C^\infty(\mathbb{R}^2)$ que sea periódica en x, por ejemplo de periodo 1, y que verifique las condiciones:

$$\begin{aligned} \frac{\partial u}{\partial t}(x,t) &= \frac{\partial^2 u}{\partial x^2}(x,t) \\ u(x,0) &= u_0(x) \end{aligned}$$

en donde $u_0(x) \in C^\infty(\mathbb{R})$ es periódica de periodo 1.

Para resolver este problema se puede utilizar la descomposición de $u(x,t)$ y $u_0(x)$ en sus series de Fourier. Es decir se utiliza el teorema que asegura que si $u(x) \in C^\infty(\mathbb{R})$ tiene periodo 1, entonces $\exists\{u_j\}_{j\in\mathbb{Z}}$, tales que $\forall N \in \mathbb{N}, \exists C_N \in \mathbb{R}$ tal que $\forall j \in \mathbb{Z}$, se cumple que $\mid u_j \mid \leq \frac{C_N}{(1+|j|)^N}$ y $\forall x \in \mathbb{R}, u(x) = \sum_{j\in\mathbb{Z}} u_j e^{2\pi i j x}$. Por tanto, si existe una $u(x,t)$ solución de $P2$, entonces se cumple que:

$$\begin{aligned} u(x,t) &= \sum_{j\in\mathbb{Z}} u_j(t) e^{2\pi i j x} \\ u_0(x) &= \sum_{j\in\mathbb{Z}} u_0^j(t) e^{2\pi i j x} \end{aligned}$$

por lo que $\forall j \in \mathbb{Z}$:

$$\begin{aligned} u_j'(t) &= -4\pi^2 j^2 u_j(t) \\ u_j(0) &= u_0^j \end{aligned}$$

que para cada j es una ecuación diferencial ordinaria. Por tanto, al sustituir todas las soluciones en $u(x,t)$, se obtiene que $u(x,t)$ debe verificar la fórmula:

$$u(x,t) = \sum_{j\in\mathbb{Z}} u_0^j e^{2\pi i j x - 4\pi^2 j^2 t}$$

Por tanto, si la solución existe, entonces debe verificar esta fórmula. Análogamente al caso anterior, la heurística finalmente comprueba que una función definida mediante esta forma verifica $P2$. Para ello se comprueba, utilizando el teorema anterior, que $u(x,t) \in C^{\infty}(\mathbb{R}^2)$ y que verifica la ecuación y la condición inicial. Cabe señalar además que técnicas como la descomposición de Fourier se pueden aplicar también a otros problemas, como por ejemplo el de la cuerda vibrante ilustrado anteriormente.

Cabe observar que las dos demostraciones anteriores se adaptan al esquema formal de resolución que el matemático utiliza y en PRÓGENES no sería difícil construir una Base de Conocimiento que permitiera resolver dichos problemas. La comprobación en el segundo problema de que $u(x,t) \in C^{\infty}(\mathbb{R}^2)$ es más delicada pero sería posible dar heurísticas generales como las contenidas en la tesis de Pablo Castells, [Castells 94], que permitirían efectuarla.

3.3.2 Problema 4

Como una primera ilustración de la heurística de razonamiento basado en el objetivo en el contexto de PRÓGENES, consideremos el problema 4 de la Introducción:

- Hallar la velocidad inicial de un proyectil que alcanza una altura máxima de 2 metros cuando su proyección sobre la horizontal esta a una distancia de 1 metro del lugar del disparo.

para comprender el mecanismo de resolución. La expresión formal que representa el enunciado del problema sería:

```
(hallar (velocidad-inicial (<> ?p PROYECTIL))
        (y (alcanza-maximo (trayectoria ?p) 1)
           (= (imagen (trayectoria ?p) 1) 2)))
```

Un proyectil es una masa sometida a la fuerza de la gravedad, que se dispara con una velocidad inicial y un ángulo de tiro, y que describe una trayectoria parabólica. Las definiciones relevantes en este problema son las del tipo PROYECTIL y la meta-función `trayectoria`, que se definen mediante los siguientes elementos de las Bases de Conocimiento [2]:

[2]Se supone que la aceleración de la gravedad es 10.

```
(obj proyectil ((angulo-tiro REAL) (velocidad-inicial REAL))
     (y (> velocidad-inicial 0)
        (< 0 angulo-tiro (/ pi 2))))
```

```
(fun trayectoria ((proyectil PROYECTIL))                    (20)
     FUNCION-REAL
     V
     (el (<> ?f FUNCION-REAL)
         (= (imagen ?f 0) 0)
         (= (imagen (derivada ?f) 0) (tan (angulo-tiro proyectil)))
         (= (derivada ?f 2)
            (/ (* -10 (+ 1 (expt (tan (angulo-tiro proyectil)) 2)))
               (expt (velocidad-inicial proyectil) 2)))))
```

Aparte de esta definición de trayectoria aplicada al tipo PROYECTIL, se puede definir la trayectoria aplicada al tipo PARTICULA definido en (26) (suponiendo las restricciones de que la partícula se mueva en el plano y que se pueda despejar $y(x)$, es decir que por ejemplo trayectorias de tipo espiral no estarían permitidas). En la jerarquía de tipos se verifica que PROYECTIL < PARTICULA y, por tanto, para aplicar `trayectoria` a una expresión de tipo PROYECTIL se puede equivalentemente utilizar en primer lugar la traducción de la expresión de tipo PROYECTIL a otra expresión de tipo PARTICULA y, a continuación, aplicar la semántica correspondiente. Por simplicidad, en la descripción de este problema se utiliza la definición anterior (20).

En este problema se utiliza razonamiento basado en el objetivo como sigue: por la definición del objeto PROYECTIL, si ?p es un proyectil, entonces existen metavariables ?a y ?v de tipo REAL tales que (= ?p (proyectil ?a ?v)), por lo que se pueden añadir al módulo de objetos pistas MOP. Este tipo de razonamiento es muy frecuente y se puede sistematizar de forma algorítmica mediante el algoritmo de expansión AERP, que se describe en 4.4. La transformación AERP transforma la asignación de tipo
(<> ?p PROYECTIL) en la expresión de asignación de tipo
(proyectil (<> ?a REAL) (<> ?v REAL)), ya que buscar un proyectil equivale a buscar su velocidad inicial y su ángulo de tiro, obteniendo una base hija que se activa.

Por otra parte, la aplicación de la definición de la metafunción `trayectoria` (que supone aplicar la metafunción `el` que resuelve una ecuación diferencial de segundo orden con condición inicial en la función y en su derivada de acuerdo a los patrones de solubilidad ilustrados en 4.2) obtiene como valor la función:

```
(funcion (<> ?x REAL)
         (- (* (tan ?a) ?x)
            (/ (* 5 (+ 1 (expt (tan ?a) 2)) (pexpt ?x 2))
               (* (expt ?v 2)))))
```

La función RAZONA-BO (razona basado en el objetivo) añade en el módulo de pistas, MP, el objetivo, MM, y expresiones formales que son condiciones necesarias para obtener MP. Si ocurre, como en este caso, que el objetivo MM inicial es una conjunción de expresiones formales, entonces añade a MP las expresiones que aparecen como cláusulas conjuntivas en MM. Por tanto, mediante la acción de CANONIZA y AERP se obtiene la siguiente base de datos en donde aparecen objetos buscados como pistas (MOP), módulo de objetos pistas, y las pistas MP:

```
MOP: (<> ?a REAL) (<> ?v REAL)
MP:  (alcanza-maximo
       (funcion (<> ?x REAL)
         (- (* (tan ?a) ?x)
            (/ (* 5 (+ 1 (expt (tan ?a) 2)) (pexpt ?x 2))
               (* (expt ?v 2)))))
       1)
     (= (- (tan ?a) (/ (* 5 (+ 1 (expt (tan ?a) 2))) (expt ?v 2))) 2)
```

Existen dos teoremas relevantes relativos a Cálculo Infinitesimal que se usan en la resolución de este problema. Son los que afirman que si una función derivable alcanza un máximo local en un punto, entonces dicho punto es crítico, y que si una función derivable dos veces tiene un punto crítico con segunda derivada negativa, entonces alcanza un máximo local en dicho punto. Estos se expresan en las Bases de Conocimiento mediante los siguientes elementos [3]:

```
(teor (para-todo ((<> ?f FUNCION-REAL) (<> ?a REAL))
         (-> (y (derivable ?f) (alcanza-maximo ?f ?a))
             (punto-critico ?a ?f))))
```

[3]Suponemos que el metapredicado alcanza-maximo se interpreta, en realidad, como una propiedad de máximo local.

```
(teor (para-todo ((<> ?f FUNCION-REAL) (<> ?a REAL))
        (-> (y (derivable ?f 2)
               (punto-critico ?a ?f)
               (< (imagen (derivada ?f 2) ?a) 0))
            (alcanza-maximo ?f ?a))))
```

Además, RAZONA-BO añade a MP los resultados de aplicar razonamiento *forwards* a las pistas MP a partir de los objetos de MOP. En nuestro caso, RAZONA-BO añade pistas derivadas de la aplicación del primer teorema que exige que la función trayectoria tenga un punto crítico cuando ?x vale 1 [4]. Por tanto, el valor de este módulo es:

```
MP: (alcanza-maximo
      (funcion (<> ?x REAL)
        (- (* (tan ?a) ?x)
           (/ (* 5 (+ 1 (expt (tan ?a) 2)) (pexpt ?x 2))
              (* (expt ?v 2)))))
      1)
    (= (- (tan ?a) (/ (* 10 (+ 1 (expt (tan ?a) 2))) (expt ?v 2))) 0)
    (= (- (tan ?a) (/ (* 5 (+ 1 (expt (tan ?a) 2))) (expt ?v 2))) 2)
```

En este punto, la función RAZONA-BO utiliza a la función REDUCE que considera el subproblema formado por las dos ecuaciones con dos incógnitas y determina exactamente el valor de las metavariables. REDUCE se aplica por tanto en este caso equivalentemente a la siguiente regla de cálculo:

```
Si (= (- (tan ?a) (/ (* 10 (+ 1 (expt (tan ?a) 2))) (expt ?v 2))) 0)
   (= (- (tan ?a) (/ (* 5 (+ 1 (expt (tan ?a) 2))) (expt ?v 2))) 2)
Entonces
   (:= ?v (sqrt (/ 5 32))) o bien (:= ?v (- sqrt (/ 5 32)))
   (:= ?a (atan 4))
```

PRÓGENES parte de unos patrones de solubilidad, como se describe en 4.2, y a partir de ellos actúa como si la anterior regla fuera una regla del sistema. A continuación, la función RAZONA-BO modifica la base activa inicial sustituyendo los valores de las metavariables ?a y ?v, e ignorando el valor negativo de la velocidad debido a la condición de la definición del tipo PROYECTIL. La base que aparece como resultado de esta acción es la siguiente:

[4]Para poder aplicar dicho teorema se aplica la semántica de la función derivable a un polinomio de segundo grado obteniendo V.

```
MOB: (sqrt (/ 5 32))
MM:  (alcanza-maximo
       (funcion (<> ?x REAL)
                (- (* 4 ?x) (* 2 (expt ?x 2))))
       1)
```

A partir de esta base se crea mediante RAZONA-B una base hija en la que se utiliza por razonamiento *backwards* el segundo de los teoremas anteriormente mencionados. Por tanto, la base hija que aparece activada es:

```
MOB: (sqrt (/ 5 32))
MM:  (y (derivable-
          (funcion (<> ?x REAL) (- (* 4 ?x) (* 2 (expt ?x 2)))))
        (punto-critico
          1
          (funcion (<> ?x REAL) (- (* 4 ?x) (* 2 (expt ?x 2)))))
        (< (imagen
             (derivada
               (funcion (<> ?x REAL)
                        (- (* 4 ?x) (* 2 (expt ?x 2))))
               2)
             1)
           0))
```

y la funcion CANONIZA aplica a esta base metaevaluación obteniendo una base que está resuelta con resultado final (sqrt (/ 5 32)):

```
MOB: (sqrt (/ 5 32))
MM:  V
```

3.3.3 La heurística en Prógenes

La heurística de razonamiento basado en el objetivo se puede usar en un sistema lógico como el usado en PRÓGENES que contenga los mecanismos de razonamiento *forwards* basados en la regla de inferencia de *Modus Ponens*, $((A \rightarrow B) \wedge A) \rightarrow B$, y de reescritura basados en beta-reduccion (que afirma la igualdad de una función aplicada a un objeto y la correspondiente sustitución del objeto en la definición de la función). Para describir

la heurística partimos de una lógica basada en el cálculo de *sequents*, como por ejemplo en [Paulson 87], en la que se utilizan *sequents* extendidos. Un *sequent* extendido es un *sequent* que tiene una lista adicional de fórmulas, llamadas pistas (P). Las pistas P se escriben debajo del símbolo de un *sequent* $\vdash$. Todas las reglas de la lógica de *sequents* se pueden extender a *sequents* extendidos ignorando la existencia de pistas.

Existen tres reglas de inferencia para la heurística:

$$\frac{\Gamma \underset{\Omega}{\vdash} \exists\Psi.\Omega}{\Gamma \vdash \exists\Psi.\Omega}$$

$$\frac{\Gamma,\ A \longrightarrow B \quad \underset{\Phi,\ C,\ B_\sigma}{\vdash} \quad \exists\Psi.\Omega}{\Gamma,\ A \longrightarrow B \quad \underset{\Phi,\ C}{\vdash} \quad \exists\Psi.\Omega}$$

si Ψ es no vacío y C unifica con A mediante σ, y:

$$\frac{\Gamma \underset{\Phi_B^A}{\vdash} \exists(\Psi,\ V)\ .\ \Omega_B^A}{\Gamma \underset{\Phi,\ A:=B}{\vdash} \exists(\Psi,\ A)\ .\ \Omega}$$

donde V denota todas las variables libres de B y Ω_B^A denota Ω con A sustituido por B. Además, estas reglas, al igual que en el caso del razonamiento *forwards*, se aplican *bottom-up* a partir del enunciado.

Así, por ejemplo, suponiendo que la función p tiene asociada la regla de reescritura $p(c) := q(c)$, y que c es una constante, entonces el teorema siguiente:

$$\forall x.\ (p(x) \longrightarrow x := c) \wedge q(c) \longrightarrow \exists x\ .\ p(x)$$

se resolvería de la siguiente forma:

El *sequent* inicial sería:

$$\forall x.(p(x) \longrightarrow x := c) \wedge q(c) \vdash \exists x.p(x)$$

Aplicando la primera regla de inferencia de la heurística seguida de la segunda, se obtiene el *sequent* extendido:

$$\forall x.(p(x) \longrightarrow x := c) \wedge q(c) \underset{p(x)\wedge x:=c}{\vdash} \exists x.p(x)$$

Por otra parte, por la tercera regla de la heurística, se obtiene:

$$\forall x.(p(x) \longrightarrow x := c) \wedge q(c) \underset{p(c)}{\vdash} \exists x.p(c)$$

de modo que si se aplica beta-reduccion a la meta, de acuerdo a la regla $p(c) := q(c)$, se obtiene el *sequent*:

$$\forall x.(p(x) \longrightarrow x := c) \wedge q(c) \underset{q(c)}{\vdash} \exists x.q(c)$$

que se demuestra satisfactoriamente ya que la meta afirma la existencia de una variable que es un hecho conocido.

En PRÓGENES, la función RAZONA-BO (razona basada en el objetivo) aplica esta heurística utilizando las tres reglas anteriores. La primera regla de la heurística corresponde a copiar el objetivo MM en el módulo de pistas MP. La segunda aplica razonamiento *forwards generalizado* a partir de MP, y añadiendo los hechos obtenidos a MP. Este razonamiento es equivalente al realizado por las funciones REDUCE y AMPLIA-MH a partir de los hechos conocidos MH (pero aplicado a las pistas), y por tanto se utiliza al motor de inferencia y a la función REDUCE (ver 4.2.3), que aplica patrones de solubilidad para crear reglas de reescritura cuando puede aplicar métodos de resolución directa para determinados subproblemas. Por último, la tercera regla de la heurística aplica en el objetivo MM las reglas de reescritura que existen en el módulo de pistas.

Esta heurística corresponde en el aspecto cognitivo de resolución de problemas a transformaciones que se aplican inmediatamente en cualquier instante de la resolución de un problema. En el contexto de PRÓGENES, resulta ser, por tanto, una transformación estática que se aplica inmediatamente sin crear alternativas ya que para encontrar la solución final basta con comprobar que los valores candidatos verifican todas las condiciones del problema.

3.3.4 Problema 5

El razonamiento basado en el objetivo se puede aplicar también utilizando metarreglas que tengan como referencia a MP. En particular, estas metarreglas pueden aplicar operadores relevantes a determinadas expresiones formales. Así, por ejemplo, en determinadas ecuaciones funcionales resulta particularmente apropiado considerar la derivación en ambos miembros de la igualdad. Así mismo, en Cálculo Infinitesimal se toman límites en determinadas inecuaciones entre sucesiones aplicando la propiedad de conservación de la monotonía al tomar límites. Sin embargo, se considera un control sobre el uso de metarreglas como las anteriores, ya que su uso automático puede producir ciclos infinitos. Estas heurísticas están descritas en la tesis de Pablo Castells, [Castells 94].

Consideremos un último ejemplo en el que se utiliza la heurística de razonamiento basado en el objetivo utilizando una metarregla de conservación de la igualdad del estilo introducido en el párrafo previo. El problema a tratar es el número 5 de la Introducción:

- Hallar la superficie de revolución alrededor del eje x cuya sección con el plano OYZ tiene área π, y el área de corte con cualquier otro plano paralelo a el es igual al volumen del sólido limitado por dicho plano y la superficie de revolución, en la dirección negativa del eje x.

cuyo enunciado en Lenguaje Formal es el siguiente:

```
(hallar
  (<> ?s SUPERFICIE-REVOLUCION)
  (y (= (eje-revolucion ?s) (eje :x))
     (= (area-transversal 0 ?s) pi)
     (para-todo (<> ?x REAL)
        (= (area-transversal ?x ?s)
           (volumen-revolucion (:-infinito ?x) ?s)))))
```

en donde `area-transversal` es la metafunción cuya semántica es el área de la sección transversal a la superficie de revolución mediante el plano perpendicular al eje en `?x`, y `volumen-revolucion` es otra metafunción con semántica el volumen del solido limitado por la superficie de revolución y por dos planos perpendiculares al eje de giro. Por otro lado, el tipo `SUPERFICIE-REVOLUCION` está definido mediante:

```
(obj superficie-revolucion
     ((eje-revolucion EJE) (funcion-generatriz FUNCION-REAL))
     (> funcion-generatriz 0))
```

En este problema se utiliza razonamiento basado en el objetivo aplicando la definición del tipo SUPERFICIE-REVOLUCION y deduciendo:

```
Si (<> ?s SUPERFICIE-REVOLUCION)
Entonces, existen (<> ?e EJE), (<> ?f FUNCION-REAL)
             (y (= ?s (superficie-revolucion ?e ?f)) (> ?f 0))
```

por lo que se pueden añadir al módulo de objetos pistas MOP. Como se ha dicho anteriormente, este tipo de razonamiento esta sistematizado algorítmicamente mediante AERP, como se verá más adelante en 4.4. Si se aplica la transformación AERP a la asignación de tipo (<> ?s SUPERFICIE-REVOLUCION), se obtiene:

```
(superficie-revolucion (<> ?e EJE) (<>?f FUNCION-REAL))
```

Por otra parte, el tipo EJE representa a los ejes de coordenadas, de modo que expresiones como (eje :x) o (eje :z) tienen tipo EJE. Además la función CANONIZA modulariza y simplifica la base inicial obteniéndose la siguiente base activa:

```
MOP: (superficie-revolucion (eje :x) (<> ?f FUNCION-REAL))
MP:  (= pi (* pi (expt (imagen ?f 0) 2)))
     (= (* pi (expt ?f 2))
        (* pi (integral :-infinito ?x (expt ?f 2))))
```

A partir de esta base, se sigue aplicando el razonamiento basado en el objetivo. La segunda expresión del módulo de pistas relaciona la función con su integral indefinida. Existe una metarregla que afirma que si una pista es una ecuación integral, entonces se añade como pista la ecuación que resulta al derivar ambos miembros. Se trata de una manera heurística de aplicar la regla $f(x) = g(x) \rightarrow f'(x) = g'(x)$, pero evitando que se aplique indefinidamente y considerándose solo en caso de que haya garantías de utilidad por introducir una ecuación diferencial que se puede intentar resolver. Por tanto, se obtiene un módulo de pistas MP que contiene una ecuación diferencial lineal de primer orden junto con una condición inicial:

```
MP: (= (* 2 pi ?f (derivada ?f)) (* pi (expt ?f 2)))
    (= pi (* pi (expt (imagen ?f 0) 2)))
```

La funcion REDUCE se aplica entonces al módulo de pistas MP calculando la solución de la ecuación diferencial con condición inicial, y obteniéndose para el módulo de objetos pistas MOP:

```
MOP: (superficie-revolucion
       (eje :x)
       (funcion (<> ?x REAL)(* 1/2 (exp (* 1/2 ?x)))))
```

Finalmente se aplica la sustitución en MOB de la base de datos inicial y se comprueba que existe la integral indefinida y que se verifica el objetivo, es decir M: V, con lo que el problema queda resuelto.

Capítulo 4

Heurísticas específicas

En este capítulo se describen en el marco de PRÓGENES un conjunto de heurísticas específicas utilizadas en PRÓGENES que aplican conocimiento procedural en resolución de problemas. Pese a no tener la generalidad de las reglas de deducción introducidas en el capítulo anterior, estas heurísticas se utilizan muy frecuentemente durante la resolución de problemas.

Por un lado, se describe la manera de aplicar transformaciones de reducción que simplifican y canonizan el estado parcial de un problema sustituyendo un conjunto de expresiones por otro representado de una manera más simple. Para aplicar estas transformaciones se definen unos patrones de solubilidad que representan el conocimiento procedural utilizado en estos procesos. Además se definen técnicas de instanciación para los valores obtenidos mediante aplicación de reducción, que corresponden al proceso de sustitución en el objetivo de un problema de los valores obtenidos por reducción.

Por otra parte, se describe una heurística que utiliza conocimiento procedural para resolver problemas en un amplio espectro de dominios. Esta heurística es el Algoritmo de Expansión en Resolución de Problemas (AERP). En relación con el capítulo anterior, cabe señalar que AERP es un tipo particular de razonamiento basado en el objetivo. Se utiliza en problemas en los que se busca un objeto y, para encontrarlo, se supone que existe dicho objeto y se razona a partir de el expresándolo en función de otros objetos más elementales. En AERP se utilizan los axiomas de expansión derivados de la definición de objetos compuestos en las Bases de Conocimiento. Es decir, se utiliza que:

```
Si (<> ?x TIPO), donde TIPO es compuesto
Entonces, existen (<> ?x1 TIPO1) ... (<> ?xn TIPON) tales que
            (= ?x (tipo ?x1 ... ?xn))
```

en donde tipo es el constructor definido en las Bases de Conocimiento de TIPO.

Así, por ejemplo, a partir de la definición del tipo PUNTO, si se aplica el axioma de expansión a un punto representado por la expresión de asignación de tipo (<> ?p (PUNTO 2)), entonces se obtiene que existen (<> ?x REAL) y (<> ?y REAL) tales que (= ?p (punto ?x ?y)).

Sin embargo, el tratamiento de este axioma debe controlarse mediante alguna heurística puesto que su aplicación automática introduce nuevos objetos e igualdades. En AERP se desarrolla una heurística que utiliza razonamiento basado en el objetivo y aplica el axioma de expansión para encontrar el objeto buscado. En este sentido, las técnicas utilizadas permiten simplificar la búsqueda del objeto aplicando un algoritmo de expansión sencillo que transforma el estado inicial de un problema en nuevos estados a los que se les aplica una transformación de reducción que reconoce y resuelve casos sencillos, como por ejemplo ecuaciones numéricas lineales o ecuaciones diferenciales.

En el primer apartado se analiza desde un punto de vista cognitivo cómo es el conocimiento procedural disponible durante la resolución de un problema. En la segunda sección se definen los patrones de solubilidad que representan el conocimiento procedural utilizado en aquellos casos en que se saben resolver determinados subproblemas. A continuación, se describe algorítmicamente la expansión de objetos en PRÓGENES. Por fin, en el cuarto apartado se analiza el comportamiento algorítmico de AERP en el proceso de resolución.

4.1 Conocimiento procedural

La forma de resolver un problema en dominios como Matemáticas depende fundamentalmente de la práctica adquirida durante el proceso de aprendizaje. Parte de lo aprendido corresponde a conocimiento procedural o compilado acerca de patrones de problemas en donde no se aplican reglas de deducción sino métodos de resolución directa. En este contexto, AERP parte de un problema e identifica la posibilidad de aplicar conocimiento procedural, obteniendo en caso satisfactorio una solución directa. Para problemas de carácter marcadamente deductivo, como ya hemos visto, se utilizan sin embargo teoremas para llevar a cabo una búsqueda de la solución a través del espacio de estados.

4.1.1 Problema 6

Como ilustración sencilla de la aplicación de conocimiento procedural y compilado que se utiliza en AERP, se considera en este apartado un problema sencillo de Geometría en el plano y una traza de su resolución en PRÓGENES. A lo largo del capítulo se describen con mayor detalle las transformaciones que se utilizan para aplicar esta heurística. Se considera el problema 6 de la Introducción:

- Hallar el centro de la circunferencia que pasa por los puntos (1,1), (0,2) y $(\frac{1}{2}, \frac{\sqrt{3}+4}{2})$.

cuyo enunciado formal es el siguiente:

```
(hallar (centro (<> ?c CIRCUNFERENCIA))
        (y (pertenece (punto 1 1) ?c)
           (pertenece (punto 0 2) ?c)
           (pertenece (punto 1/2 (/ (+ (sqrt 3) 4) 2)) ?c)))
```

Inicialmente se crea una base de datos que contiene los siguientes módulos:

```
MOB: (centro (<> ?c CIRCUNFERENCIA))
MM:  (y (pertenece (punto 1 1) ?c)
        (pertenece (punto 0 2) ?c)
        (pertenece (punto 1/2 (/ (+ (sqrt 3) 4) 2)) ?c))
```

A partir de esta base, de acuerdo a la heurística del razonamiento basado en el objetivo, se crea el módulo de objetos pistas, MOP, y el módulo de pistas, MP, a los que se añaden los módulos anteriores, MOB y MM, respectivamente. Sin embargo, el subproblema planteado no se sabe resolver en la metavariable ?c.

Si se aplica expansión al objeto buscado, que está representado por la asignación de tipo (<> ?c CIRCUNFERENCIA), entonces se obtiene el desarrollo en función del centro y el radio:

```
(circunferencia (<> ?p (PUNTO 2) (<> ?r REAL))
```

También se deduce el hecho (> ?r 0) de la definición del tipo CIRCUNFERENCIA. De nuevo, no se puede resolver el problema en las metavariables ?p y ?r, pero se puede aplicar otra vez expansión en la metavariable ?p en función de sus dos coordenadas:

```
(punto (<> ?x REAL) (<> ?y REAL))
```

Al sustituir esta aplicación recursiva de expansión en la metavariable ?c de MP y MM, se obtiene una base de datos nieta de la inicial con un módulo de objetos pistas, MOP, y unas pistas MP de la forma:

```
MOP: (centro
       (circunferencia (punto (<> ?x REAL) (<> ?y REAL))
                       (<> ?r REAL)))
MP:  (pertenece (punto 1 1) (circunferencia (punto ?x ?y) ?r))
     (pertenece (punto 0 2) (circunferencia (punto ?x ?y) ?r)
     (pertenece (punto 1/2 (/ (+ (sqrt 3) 4) 2))
                (circunferencia (punto ?x ?y) ?r))
     (> ?r 0)
```

En realidad, se utilizan directamente las reglas de reescritura que se derivan de la aplicación de metaevaluación, de modo que la base de datos anterior se escribe en la forma:

```
MOP: (punto (<> ?x REAL) (<> ?y REAL))
     (<> ?r REAL)
MP:  (= (+ (expt ?x 2) (expt (- ?y 2) 2)) (expt ?r 2))
     (= (+ (expt (- ?x 1) 2) (expt (- ?y 1) 2)) (expt ?r 2))
     (= (+ (expt (- ?x (/ 1 2)) 2)
           (expt (- ?y (/ (+ (sqrt 3) 4) 2)) 2))
        (expt ?r 2))
     (> ?r 0)
```

Ahora, como se ilustra mas adelante en 4.4, AERP encarga a REDUCE que actúe seleccionando a partir de los módulos anteriores las tres ecuaciones que aparecen en MP y los objetos que relacionan, que son ?x, ?y y ?r (en general, siempre considera un conjunto maximal de relaciones reducibles). REDUCE también considera la inecuación (> ?r 0) pero la deja invariante pues no admite simplificación. Obsérvese que en el caso anterior de la base de datos sin expandir, REDUCE no pudo reconocer ningún conjunto de expresiones solubles. La función REDUCE indica qué conocimiento procedural es necesario para resolver el sistema de 3 ecuaciones con tres incógnitas, puesto que existen métodos de resolución directa que se derivan de la aplicación de ciertos operadores a ecuaciones numéricas que permiten obtener la solución. En este caso, se trata

de ecuaciones cuadráticas pero que se pueden resolver aplicando métodos de reducción y sustitución. En este caso, la función REDUCE obtiene las soluciones:

```
(or (y (:= ?x 1) (:= ?y 2) (:= ?r 1))
    (y (:= ?x 1) (:= ?y 2) (:= ?r -1)))
```

A continuación, como parte de la dinámica del razonamiento basado en el objetivo, se sustituyen estas soluciones en el objetivo y se obtiene `MM: V`, con lo que se cumple el objetivo y el resultado es el valor de `MOB` que es `(punto 1 2)`, que es la solución final al problema.

4.1.2 Dualidad procedural-deductiva

Durante el proceso de resolución automática, el conocimiento deductivo por un lado, y el procedural por otro, perfilan la búsqueda de soluciones. En analogía con el proceso humano de resolución, corresponden a la dualidad entre el reconocimiento inmediato de problemas resolubles y la puesta en funcionamiento de estrategias deductivas.

Anderson, en su teoría cognitiva ACT, [Anderson 83], considera tambien esta dualidad. Establece una conversión de conocimiento declarativo o deductivo en procedural. Según Anderson, durante el proceso de aprendizaje de resolución se produce una conversión en un conocimiento compilado, reforzándose paulatinamente la utilización de determinadas reglas hasta configurar un método de resolución. Esta conversión permite el reconocimiento del modelo de problema, además del almacenamiento de específicas reglas de resolución en cada caso particular.

Así, la manera en que se resuelve un sistema de ecuaciones lineales puede consistir en aplicar operadores de conservación de la igualdad en las ecuaciones hasta despejar cada una de las incógnitas. Por tanto, se utilizan reglas de deducción del estilo de $A = B \rightarrow f(A) = f(B)$, o bien, en general, $A_1 = B_1, \ldots, A_n = B_n \rightarrow f(A_1, \ldots, A_n) = f(B_1, \ldots, B_n)$ manteniendo como objetivo aislar una sola variable en un lado de cada igualdad, con lo que estarían determinadas completamente. Sin embargo, considerando la conversión procedural, la resolución alternativa es la utilización de un método de reconocimiento de un sistema de ecuaciones lineales y su resolución mediante técnicas de sustitución, reducción o bien métodos como los de Gauss o la regla de Cramer.

Por otra parte, el grado de compilación de conocimiento deductivo depende del nivel de conocimiento que se establece en un determinado problema, que viene definido en las Bases de Conocimiento. Es un aspecto análogo al proceso humano de resolución, en

donde ciertos subproblemas se consideran de resolución innmediata dependiendo de la persona que los resuelve. En el caso de las ecuaciones lineales, se puede considerar una resolución a partir de axiomas elementales mediante el método de sustitución para un primer nivel de iniciación, o alternativamente utilizar la regla de Cramer como rutina o máquina calculadora para niveles superiores, u otros métodos más eficientes.

Ambos tipos de conocimiento, procedural y deductivo, deben incorporarse por tanto en un sistema de resolución. Deben existir módulos de razonamiento y módulos de resolución directa que sean dirigidos desde un nivel superior durante todo el proceso. Dichos conocimientos pueden ser integrados evolucionando el estado parcial de un problema a través de transformaciones generales que emulan la aplicación de dichos conocimientos. Asociado al tipo de conocimiento procedural disponible para el sistema, existe un reconocedor de este tipo de condiciones, mientras que, por ejemplo, en el segundo caso un motor de inferencia produce transformaciones sobre cada estado. Dentro de PRÓGENES, esta bipolaridad viene representada por heurísticas como AERP, para aplicar conocimiento procedural, o por el razonamiento que se lleva a cabo mediante el motor de inferencia.

Cabe observar que, en realidad, los problemas resueltos utilizando conocimiento procedural se pueden resolver también mediante conocimiento deductivo. Pero el conocimiento procedural es la manera de utilizar el deductivo de una manera más eficiente. En PRÓGENES, AERP utiliza un mecanismo de reducción aplicándolo a cada estado parcial de un problema, así como un mecanismo de búsqueda a través del espacio de estados. En este sentido, intenta aplicar conocimiento procedural en el proceso de resolución. En la sección 4.4 se describe AERP en el contexto de PRÓGENES y, en concreto, cuáles son sus mecanismos de reducción de estados parciales de resolución a través de patrones de solubilidad, así como el proceso algorítmico de búsqueda asociado.

4.1.3 Resolución procedural

Se necesita, por tanto, una aplicación de métodos directos para calcular la solución a determinados subproblemas debido a que la aplicación de métodos deductivos es más ineficiente. De particular complejidad resulta la aplicación de propiedades algebraicas como la conmutatividad, asociatividad y distributividad de operaciones como la suma y el producto. Este tipo de complejidad se elimina, por tanto, aplicando métodos de resolución directa.

La naturaleza de un problema se reconoce mediante un predicado de solubilidad y se obtiene una solución, si es posible, aplicando métodos asociados a la clase de

problema que representa. Este conocimiento se puede describir mediante un conjunto de patrones de solubilidad. Cada patrón contiene el tipo de objetos al que se aplica, la clase de relación que se tiene entre ellos y un método de solución que determina el valor de dichos objetos. El predicado de solubilidad considera un estado de un problema, extrae los objetos indeterminados y compara la clase de problema que representa con cada patrón de solubilidad. Una vez reconocido el problema, se obtiene la solución directamente. Así, por ejemplo, supongamos que un subproblema es $x - y = 2$ y $x^2 + xy = 0$, es decir:

```
(y (= (- ?x ?y) 2) (= (+ (expt ?x 2) (* ?x ?y)) 0))
```

en donde `(<> ?x REAL)` y `(<> ?y REAL)`. Entonces, dicho subproblema se reconoce como soluble mediante un patrón de solubilidad para ecuaciones cuadráticas que da un método de solución aplicando sustitución de variables, ver (22). Y la aplicación de dicho patrón al subproblema anterior obtiene las soluciones:

```
(or (y (:= ?x 0) (:= ?y -2))
    (y (:= ?x -2) (:= y -4)))
```

Obsérvese que la aplicación indiscriminada de propiedades como la suma y el producto de números reales para el caso anterior no habría obtenido la solución de una manera directa, pues existirían varias formas de aplicarlas.

Existen tipos de problemas comunes que se resuelven mediante técnicas predefinidas. En particular, considerando ecuaciones, se tienen diferentes clases de problemas:

- Sistemas de ecuaciones lineales. Resolución por métodos de sustitución o reducción, o aplicando el método de Gauss o la regla de Cramer.
- Sistemas de ecuaciones no lineales. Resolución aplicando métodos de sustitución para reducirlos a ecuaciones polinómicas de grado no superior a 4, que se pueden resolver, o bien métodos aproximados como el de Newton.
- Sistemas de ecuaciones diferenciales. Resolución reduciendo el orden de las ecuaciones hasta primer orden y hallando la exponencial de la matriz asociada, para el caso de ecuaciones lineales, o aplicando métodos alternativos para ecuaciones no lineales.
- Ecuaciones en derivadas parciales. Hacer cambios de variables a coordenadas esféricas o cilíndricas en casos específicos con simetría, para reducir el problema al caso anterior, o aplicar técnicas como transformadas de Fourier en otros casos.

Cabe señalar que MATHEMATICA, [Wolfram 91], es un sistema capaz de resolver subproblemas como los anteriores. De hecho, PRÓGENES lo utiliza en parte de su implementación.

Estos métodos de resolución simbólica son comunes a diferentes dominios teóricos. En estos dominios los problemas que se plantean no son directamente expresables mediante problemas de resolución algebraica. Sin embargo, existen transformaciones que expresan el problema inicial como un nuevo problema soluble, como es el caso de las ecuaciones anteriores. En PRÓGENES, AERP se encarga de producir este tipo de transformaciones, como se describe en 4.4.

La característica común a dominios como Matemáticas, Física, Química, etc, en Resolución de Problemas es que una gran cantidad de problemas se resuelven reinterpretándolos como nuevos problemas tales como sistemas de ecuaciones diferenciales, búsqueda de autovalores de un determinado operador (por ejemplo en el caso de la resolución de la ecuación de Schrödinger), etc.

A pesar de que los problemas de resolución por expansión representan a una gran parte de los problemas que se pueden plantear en estos dominios, existen también otros tipos de problemas en los que se utilizan otra clase de técnicas propias del dominio en cuestión, o de otras técnicas de resolución. Resulta, en este sentido, destacable la generalidad que proporcionan métodos de resolución matemática al conocimiento humano, permitiéndole expresar problemas de muy diversa naturaleza en el marco de la resolución algebraica de las Matemáticas.

4.2 Patrones de solubilidad

En esta sección se describen los patrones de solubilidad en PRÓGENES. Estos patrones contienen conocimiento procedural o compilado sobre determinados subproblemas que pueden ser resueltos mediante técnicas de aplicación directa. Emulan la compilación de conocimiento que se realiza en el aprendizaje de la resolución de problemas en un dominio como Matemáticas o Física. Cada patrón contiene unos objetos (las incógnitas), unas relaciones a las que se les aplica, una condición para poder aplicarlo y un método de solución para determinar tales objetos, si es posible. En adelante se describen en detalle estos patrones, así como la manera en que a partir de ellos se aplica la función REDUCE.

4.2.1 Introducción de patrones

Tanto conocimiento procedural como deductivo se pueden representar en forma de reglas de deducción. Así, por ejemplo, en el problema de determinar la circunferencia inscrita en un triángulo dado, si es conocido el hecho de que la circunferencia se construye considerando como centro el lugar donde se cortan las bisectrices (el incentro del triángulo) y como radio la distancia del incentro a uno de los lados del triángulo, este hecho se puede expresar mediante el teorema:

```
(para-todo (triangulo (<> ?p1 (PUNTO 2))
                      (<> ?p2 (PUNTO 2))
                      (<> ?p3 (PUNTO 2)))
  (inscrito (circunferencia (incentro (triangulo ?p1 ?p2 ?p3))
                            (distancia
                               (incentro (triangulo ?p1 ?p2 ?p3))
                               (recta ?p1 ?p2)))
            (triangulo ?p1 ?p2 ?p3)))
```

Sin embargo, este conocimiento se expresa mas eficientemente en un patrón de solubilidad que en un teorema, puesto que si se incluyera este teorema en las Bases de Conocimiento, entonces el motor de inferencia lo aplicaría inmediatamente para cualquier triángulo, lo que en general no es útil. Por tanto, en PRÓGENES, se introducen patrones de solubilidad para que una función, REDUCE los pueda aplicar cuando son útiles. En este caso, el conocimiento expresado en el anterior teorema se expresa mediante el siguiente patrón de solubilidad:

```
(patron-sol                                                    (21)
   (objetos (<> ?c CIRCUNFERENCIA))
   (relaciones (inscrito ?c (triangulo (<> ?p1 (PUNTO 2))
                                       (<> ?p2 (PUNTO 2))
                                       (<> ?p3 (PUNTO 2)))))
   (condicion V)
   (metodo-solucion
       (:= ?c (circunferencia
                  (incentro (triangulo ?p1 ?p2 ?p3))
                  (distancia (incentro (triangulo ?p1 ?p2 ?p3))
                             (recta (?p1 ?p2)))))))
```

en donde se ha utilizado una notación para definir un patrón de solubilidad considerando asignaciones de tipo en objetos y relaciones. Obsérvese además que en este caso no se exige ningún tipo de condición puesto que este método da la solución para cualquier tipo de triángulo.

Por otra parte, no siempre es posible dar un patrón de solubilidad definido mediante patrones clásicos, como el caso anterior. En el apartado siguiente se describe la forma general en que se pueden definir estos patrones.

4.2.2 Estructura de los patrones

Durante el proceso de resolución de un problema se utiliza conocimiento previo sobre resolución de determinados subproblemas para utilizarlo en determinados pasos de la resolución. Para ello, se utiliza la función REDUCE que se aplica a un estado parcial de resolución. Así, por ejemplo, si el objetivo es un conjunto de ecuaciones, REDUCE las contempla como tales y obtiene su solución a partir de métodos conocidos. Si no es posible encontrar ningún patrón de solubilidad que obtenga la solución, entonces se utilizan otras transformaciones, como se describe en el capítulo 3.

Supongamos, en primer lugar, un caso sencillo en donde la lista de objetos que aparecen como pistas contiene una sublista, <obj>, formada por unas variables que representan a números reales y como sublista del módulo de pistas, <relac>, unas ecuaciones lineales que relacionan unas con otras. Por ejemplo, supongamos que buscamos números reales x e y tales que $2x + y = 1$ y $x - 2y = -2$, es decir que:

```
<obj>:    (<> ?x REAL), (<> ?y REAL)
<relac>:  (= (+ (* 2 ?x) (* 3 ?y)) -1), (= (- ?x (* 2 ?y)) -2)
```

Para resolver este subproblema, se considera un patrón de solubilidad (PS) que puede ser aplicado a unos objetos y relaciones genéricas y que representa el subproblema de resolver un sistema de ecuaciones lineales dando un método de resolución, como puede ser la resolución por determinantes en la regla de Cramer. Este patrón es el siguiente:

```
(patron-sol
  (objetos (:todos REAL))
  (relaciones (:todos (= REAL REAL)))
  (condicion (:todos relaciones (ecuac-lineal-p relaciones objetos)))
  (metodo-solucion (resuelve-Cramer relaciones objetos)))
```

en donde se utiliza una notación más general que en el caso del apartado anterior, (21).

Este patrón expresa que el problema de resolver unas ecuaciones lineales con coeficiente reales es soluble aplicando un método de solución que es la regla de Cramer. En general, un patrón se aplica a unos objetos, expresados mediante asignaciones de tipo, y a unas relaciones, que son expresiones booleanas, obteniendo una solución directa para los objetos si se cumplen las restricciones expresadas en cada uno de sus elementos. En este caso, PS1 es aplicable a <obj> y <relac> puesto que se cumple que todas las asignaciones son de tipo REAL y las relaciones son ecuaciones lineales.

La función REDUCE aplica cada patrón de solubilidad a unos objetos (que son metavariables que representan a objetos conocidos en MO o a buscados en MOP) y unas relaciones iniciales (que son hechos que aparecen en el módulo de pistas MP y que son seleccionados considerando el conjunto maximal de expresiones con el mismo constructor), y determina el valor de dichos objetos, si es aplicable, y obtiene F, en caso contrario. En el caso anterior:

```
REDUCE(PS, <relac>, <obj>) := (y (:= ?x 4) (:= ?y 3))
```

En problemas de Geometría aparecen con mucha frecuencia sistemas de ecuaciones con términos cuadráticos. Los típicos casos aparecen en el caso de superficies como esferas o elipsoides, que se pueden representar mediante ecuaciones de segundo grado. También es posible definir un patrón para tales casos, en donde el método de resolución utiliza otro tipo de técnicas de manipulación algebraica, como sustitución y reducción. En tal caso, obtienen la solución cuando al aplicar sustitución y despejar el valor de cada una de ellas no aparecen ecuaciones polinómicas en una variable de grado superior a 4; si no es posible resolverlas, las deja invariantes. El patrón de solubilidad para el caso cuadrático sería:

```
(patron-sol                                                    (22)
  (objetos (:todos REAL))
  (relaciones (:todos (= REAL REAL)))
  (condicion (:todos (ecuac-polinomial-p relaciones objetos)))
  (metodo-solucion (intenta-resolver-sust-red relaciones objetos)))
```

Este patrón es aplicable también a sistemas de ecuaciones lineales, aunque es menos eficiente y directo. Por tanto, la resolución mediante el método de Cramer se aplica antes que mediante sustitución y reducción. En el mecanismo de búsqueda de patrones

de solubilidad aplicables para cada par de relaciones y objetos se comprueba por tanto antes la linealidad del sistema de ecuaciones. Se supone que la elección del orden en que aparecen los patrones es responsabilidad de la persona que diseña las Bases de Conocimiento.

Otro ejemplo de patrón de solubilidad es el que reconoce inecuaciones en una variable y las resuelve, si es posible, obteniendo como resultado una relación canónica a una unión de intervalos disjuntos. Este patrón tiene la forma:

```
(patron-sol
  (objetos REAL)
  (relaciones (:todos (:o (> REAL REAL) (< REAL REAL))))
  (condicion
      (:todos relaciones (inecuac-polinomial-p relaciones objetos)))
  (metodo-solucion (intenta-resolver-inec relaciones objeto)))
```

Análogamente al caso de ecuaciones de números reales es posible obtener, en casos como sistemas mecánicos newtonianos, sistemas de ecuaciones diferenciales que relacionan funciones de una variable, o incluso, en casos como mecánica cuántica en varias dimensiones, ecuaciones en derivadas parciales. Para tales casos se considera un patrón de solubilidad que reconozca un sistema de ecuaciones diferenciales ordinarias con condiciones iniciales y obtenga las soluciones, si es posible. En tal caso, el patrón de solubilidad sería:

```
(patron-sol                                                    (23)
  (objetos (:todos FUNCION-REAL))
  (relaciones (:varios edos (= FUNCION-REAL FUNCION-REAL))
              (:resto cond-inics (= REAL REAL)))
  (condicion (y (: todos edos (ecua-dif-ordin-p edos objetos))
                (condiciones-iniciales-p cond-inics objs)))
  (metodo-solucion (intenta-resolver-edos edos cond-inics)))
```

Por otra parte, para subproblemas en los que se tiene una ecuación diferencial en una función de cualquier orden con condiciones iniciales en la función y en sus derivadas, se utiliza un patrón de solubilidad que transforma este subproblema inicial en un sistema de ecuaciones diferenciales ordinarias, de modo que la función REDUCE aplicaría a continuación el módulo ilustrado anteriormente. En este caso el patrón es:

```
(patron-sol                                                    (24)
  (objetos FUNCION-REAL)
  (relaciones (ec-dif (= FUNCION-REAL FUNCION-REAL)))
              (:resto cond-inics (= REAL REAL)))
  (condicion (y (ecuac-difer-p objetos)
                (cond-inicial-derivadas-p cond-inics)))
  (metodo-solucion (cambia-variables-ecuac-difer ec-dif cond-inics)))
```

4.2.3 Función REDUCE

Dentro de PRÓGENES, los patrones de solubilidad están contenidos en las Bases de Conocimiento y añaden otro tipo de conocimiento procedural utilizado para la resolución directa de determinados subproblemas. La estructura de cada patrón es, por tanto:

```
(patron-sol
  (objetos {<TIPO1>* | (:varios <TIPO2>)}+ [:resto <TIPO3>] |
           (:todos <TIPO>))
  (relaciones {<expr1>* | (:varios <expr2>)}+ [:resto <expr3>] |
              (:todos <expr>))
  (condicion  {<expr1>* | (:varios <expr2>)}+ [:resto <expr3>] |
              (:todos <expr>))
  (metodo-solucion (<funcion-solucion> <sub-relac> <sub-obj>)))
```

en donde se puede opcionalmente poner nombre a subexpresiones de las componentes `objetos`, `relaciones` y `condicion` y considerar a partir de ellas `<sub-relac>` y `<sub-obj>` como argumentos de `<funcion-solucion>`.

Cada patrón se aplica al par formado por `<relaciones>` y `<objetos>` obtenidos a partir de un estado parcial de resolución, en donde `<objetos>` son asignaciones de tipo y `<relaciones>` son expresiones formales PRÓGENES de tipo BOOL. Para seleccionar `<relaciones>` de un módulo de una base hechos, se considera una familia maximal de expresiones reconocidas por el mismo patrón, para que se pueda comparar con módulos de solubilidad como los descritos en el apartado anterior.

La función REDUCE se aplica a un `<patron>`, a unas `<relaciones>` y `<objetos>` dados y, si es aplicable dicho patrón al par (`<relaciones>`, `<objetos>`), lo transforma en una forma canónica cuando puede aplicar el `<metodo-solucion>` y a continuación

lo sustituye en la base inicial, o bien deja <objetos> y <relaciones> invariantes si no es aplicable. Es decir se tiene:

```
REDUCE(<patron> <relaciones> <objetos>) :=
  (<metodo-solucion> <sub-relaciones> <sub-objetos>)
```

Se dice que un patrón es aplicable a dicho par si se cumplen las restricciones que se especifican en cada uno de sus elementos. En primer lugar, los tipos de los objetos deben ser subtipos de los especificados en la componente objetos. En el caso de que aparezca :todos, se debe cumplir para cada uno de ellos (si se define con :varios y :resto, es análogo a :todos salvo que se permite flexibilizar el número de expresiones). Análogamente, las restricciones expresadas en la segunda componente relaciones definen mediante patrones clásicos cómo deben ser las expresiones de <relaciones>. En tercer lugar, se exigen las condiciones expresadas en el elemento condiciones. Por último, en el cuarto elemento se da un método de solución, que puede utilizar a subexpresiones definidas anteriormente en el patrón como argumentos del método de solución, y que obtiene la solución cuando es posible.

Por otra parte, cabe señalar que el conjunto de patrones de solubilidad tiene una estructura lineal, de modo que REDUCE considera en primer lugar a los métodos más directos, como ocurre en los ejemplos anteriores en que para resolver un sistema de ecuaciones lineales, se utilizan métodos de resolución directa como la regla de Cramer, en vez de métodos mas generales que pueden aplicarse, por ejemplo, a ecuaciones cuadráticas.

Por último, cabe señalar la posibilidad de que los patrones se pudieran modificar interactivamente mediante resultados obtenidos en determinados problemas. De este modo se podría considerar que se pueden almacenar conocimientos ya deducidos para utilizarlos en nuevos problemas. De todos modos, en este proceso interactivo, el experto controlaría el conocimiento que considerara directo para que el sistema general de resolución lo utilizara para resolver problemas transformables en este tipo de patrones.

4.2.4 Heurísticas de instanciación

La función REDUCE utiliza patrones de instanciación para obtener una solución canónica a un subproblema. Si esta solución es una regla de reescritura, entonces se instancia en la base de datos y se comprueba si se verifica el objetivo. Sin embargo, no siempre está claro cómo se debe instanciar el valor obtenido por la función REDUCE. Así, por ejemplo, supongamos el problema sencillo:

- Demostrar que $\exists x \in I\!R$ tal que $x^2 - 4x + 3 < 0$.

Este problema se resolvería en PRÓGENES aplicando el razonamiento basado en el objetivo y creando una base de datos que contiene a los módulos:

```
MOP: (<> ?x REAL)
MP:  (< (+ (expt ?x 2) (* -4 ?x) 3) 0)
```

de modo que si se aplica la funcion REDUCE, se modifica dicha base conteniendo MP: (< 1 ?x 3). De acuerdo al razonamiento basado en el objetivo, se debería instanciar el valor obtenido por REDUCE en el objetivo y comprobar que se verifica. Sin embargo, no es posible ya que (< 1 ?x 3) no da un único valor para ?x. Por tanto, se necesita definir un patrón de instanciación que permita sustituir un valor para la ?x, que puede ser por ejemplo 2 (en general sería el punto medio del intervalo en cuestión). Además, obsérvese que MP no debe contener mas expresiones, puesto que si existiera otro predicado P(?x), entonces no necesariamente se cumpliría el objetivo. Para definir este proceso de instanciación se define el patrón de instanciación:

```
(patron-inst
  (objetos (<> ?x REAL))
  (relaciones (y (:o (< (<> ?a REAL) ?x (<> ?b REAL))
                     (> (<> ?a REAL) ?x (<> ?b REAL)))
                 :nada-mas))
  (condicion V)
  (metodo-solucion (:= ?x (/ (+ ?a ?b) 2))))
```

en donde :ninguna-mas exige la restricción de que no hayan más relaciones en MP. Obsérvese que de este modo los patrones de instanciación tienen en cuenta no sólo conjuntos de expresiones sino también el contenido global de una base de datos.

4.3 Expansión de objetos

La estructura recursiva de definición de objetos en PRÓGENES permite representar conceptos compuestos en función de otros mas elementales. Así, por ejemplo, a partir de la definición:

```
(obj elipsoide ((centro (PUNTO 3)) (a REAL) (b REAL) (c REAL))
     (y (> a 0) (> b 0) (> c 0)))
```

del tipo ELIPSOIDE en función de su centro y las longitudes de sus semiejes, se construye el axioma de expansión para objetos de tipo ELIPSOIDE. Por tanto, el mecanismo de expansión dentro de resolución se deriva de la construcción recursiva de objetos y utiliza conocimiento acerca del procedimiento utilizado para definir dichos objetos. En nuestro caso, la regla que se deriva de dicho axioma es:

```
Si (<> ?e ELIPSOIDE)
Entonces, existen (<> ?c (PUNTO 3)), (<> ?a REAL),
                  (<> ?b REAL), (<> ?c REAL) tales que
            (= ?e (elipsoide ?c ?a ?b ?c)) y (> ?a 0)(> ?b 0)(> ?c 0)
```

En esta sección se describe el proceso algorítmico de expansión en objetos utilizado en PRÓGENES. La función EXPANDE utiliza las Bases de Conocimiento para producir transformaciones en expresiones de asignación de tipos compuestos aplicando el axioma de expansión. Así, en el caso anterior, se verifica que:

```
EXPANDE(<> ?c ELIPSOIDE) :=
  (elipsoide (<> ?c (PUNTO 3)) (<> ?a REAL)(<> ?b REAL)(<> ?c REAL))
```

Por otro lado, en el caso de objetos atómicos, tales como por ejemplo números enteros o reales, o de objetos que no se expresan en función de otros de una manera predefinida, como es el caso en conjuntos o funciones, EXPANDE los deja invariantes pues no puede aplicar conocimiento acerca de la composición de tales objetos y, por tanto, no los puede expandir en función de otros más elementales. Por ejemplo, se verifica:

```
EXPANDE(<> ?x REAL) := (<> ?x REAL)
EXPANDE(<> ?c CONJUNTO) := (<> ?c CONJUNTO)
```

4.3.1 Expansión de expresiones de asignación de tipo

En el proceso de resolución de un problema aparecen objetos que se suponen conocidos (MO), que verifican unas relaciones (MH), y objetos buscados (MOB), que deben cumplir un determinado objetivo (MM) y unas pistas MP. Los objetos buscados se representan mediante expresiones de asignación de tipo que contienen un conjunto de metavariables

que representan incógnitas del problema. Un problema se considera resuelto cuando se determina el valor de todas estas metavariables. Así, por ejemplo, si el objetivo de un problema es calcular el volumen de un elipsoide que verifica un objetivo, el objeto buscado se expresa mediante la expresión de asignación de tipo:

```
MOB: (volumen (<> ?e ELIPSOIDE))
```

De este modo, las metavariables que aparecen en MOB (en el caso anterior ?e), sobre las cuales se aplican transformaciones para determinar completamente su valor, aparecen como subexpresiones de las expresiones de asignación de tipo que forman MOB. Por tanto, es equivalente encontrar MOB a determinar el valor de dichas metavariables. El mecanismo de expansión se aplica a expresiones de asignación de tipo, en donde aparecen una o varias metavariables asignadas a tipos que representan objetos. Estas expresiones aparecen de forma natural en la formulación de problemas como el anterior.

En este problema, el proceso de expansión en expresiones de asignación de tipo se aplica a la asignación de tipo (<> ?e ELIPSOIDE) que representa al objeto buscado. Se produce, por tanto, dinámicamente una reinterpretación de los objetos MOB que se introducen como pistas en en módulo de objetos pistas MOP. También se reescriben las relaciones iniciales del objetivo MM, siendo añadidas como pistas en el módulo de pistas MP. En general, se desarrollan conceptos en función de otros más elementales sobre los que resulta mas tratable su estudio. En el caso anterior, la expresión inicial de MOB se transforma mediante expansión obteniéndose los tres estados intermedios siguientes obtenidos por la acción recursiva de la función EXPANDE:

```
(volumen (<> ?e ELIPSOIDE))
(volumen (elipsoide
           (<> ?c (PUNTO 3))
           (<> ?a REAL)(<> ?b REAL)(<> ?c REAL)))
(volumen (elipsoide
           (punto (<> ?x REAL) (<> ?y REAL) (<> ?z REAL))
           (<> ?a REAL)(<> ?b REAL)(<> ?c REAL)))
```

El proceso de definición recursiva de objetos en Geometría se generaliza a otros dominios tales como Física. En este sentido, se pueden definir conceptos tales como sistema mecánico o partícula, y expresar leyes que regulen su comportamiento. Se puede considerar un sistema mecánico formado por una o varias partículas y por uno o varios campos de fuerza que actúan sobre dichas partículas. Como se describe en (26),

se definen tipos como SISTEMA-MECANICO o PARTICULA en las Bases de Conocimiento. Así, en el caso de una sola partícula unidimensional, que se define en función de su masa, su carga y su posición, aparecerían las siguientes transformaciones de expansión:

```
EXPANDE(<> ?s SISTEMA-MECANICO) :=
(sistema-mecanico (<> ?p PARTICULA) (<> ?f (FUNCION-REAL 5 1)))

EXPANDE(sistema-mecanico (<> ?p PARTICULA)
                         (<> ?f (FUNCION-REAL 5 1))) :=
(sistema-mecanico
    (particula (<> ?m REAL) (<> ?q REAL) (<> ?x (FUNCION-REAL 1 3)))
    (<> ?f (FUNCION-REAL 5 1)))
```

en donde el tipo (FUNCION-REAL N M) representa a las funciones definidas de $\mathbb{R}^N$ en $\mathbb{R}^M$. Los tipos REAL y (FUNCION-REAL N M) no pueden expandirse de nuevo, con lo que, si se aplica otra vez EXPANDE, no varía la expresión anterior:

```
EXPANDE(sistema-mecanico
          (particula (<> ?m REAL) (<> ?q REAL)
                     (<> ?x (FUNCION-REAL 1 3)))
          (<> ?f (FUNCION-REAL 5 1))) :=
  (sistema-mecanico
      (particula (<> ?m REAL) (<> ?q REAL)
                 (<> ?x (FUNCION-REAL 1 3)))
      (<> ?f (FUNCION-REAL 5 1)))
```

Este proceso algorítmico se realiza en varias etapas, obteniendo en cada una el desarrollo mediante expansión de cada una de las asignaciones de tipo expandibles. En general, se tiene, por tanto, una estructura de grafo acíclico orientado. En realidad, en la implementación se generan los nodos del grafo en forma de árbol, [Nilsson 80], por lo que a partir de ahora se habla sólo de árboles. Por tanto, se tiene un árbol generado a partir de una expresión de asignación de tipo inicial, y en donde los hijos de cada expresión se generan expandiendo cada una de las asignaciones de tipo expandibles. Además, se evitan en este árbol repeticiones de expresiones, de manera que si en la generación del árbol, alguna de ellas ya ha aparecido, no se considera.

El mecanismo de expansión finaliza, por tanto, para asignaciones de tipos atómicos que han sido definidos mediante predicados de evaluación directa, o bien para tipos

tales como CONJUNTO o FUNCION. En el ejemplo anterior, las siguientes asignaciones de tipo no son expandibles:

```
(<> ?m REAL) (<> ?q REAL)
(<> ?x (FUNCION-REAL 1 3)) (<> ?f (FUNCION-REAL 5 1))
```

En el caso de utilizar expansión en varias etapas hasta conseguir asignaciones de tipo que no se pueden volver a expandir, aparecen todos los estados intermedios distintos utilizando la expansión en cada una de las asignaciones de tipo posibles de expandir. Así, por ejemplo, supongamos el caso de un segmento definido en la Base de Conocimiento a partir de dos puntos distintos en el plano. Entonces, en cada nivel de expansión aparecerían a partir de cada expresión de asignación de tipo otras nuevas, según una estructura de árbol. Se tendría la siguiente estructura de estados generados por expansión a partir de la asignación de tipo inicial (<> ?s SEGMENTO):

```
                        (<> ?s SEGMENTO)                                    (25)

                              ||

      (segmento (<> ?p1 (PUNTO 2)) (<> ?p2 (PUNTO 2)))

                        //          \\

(segmento (punto (<> ?x1 REAL)     (segmento (<> ?p1 (PUNTO 2))
                 (<> ?y1 REAL))               (punto (<> ?x2 REAL
          (<> ?p2 (PUNTO 2)))                         (<> ?y2 REAL)))

                        \\          //

                (segmento (punto (<> ?x1 REAL)
                                 (<> ?y1 REAL))
                          (punto (<> ?x2 REAL)
                                 (<> ?y2 REAL)))
```

En este grafo, cada nodo representa un estado intermedio de expansión. Además las hojas mas profundas, que en este caso es la expresión:

```
(segmento (punto (<> ?x1 REAL) (<> ?y1 REAL))
          (punto (<> ?x2 REAL) (<> ?y2 REAL)))
```

son expresiones de asignación no expandibles (las asignaciones de tipo que las constituyen también lo son).

En la resolución de un problema es necesario considerar todos los estados intermedios de expansión para utilizar posibles métodos de solución para cada estado particular. Por tanto, no basta con considerar la expresión de asignación de tipo inicial y la final obtenida por iterativos pasos de expansión, sino tener en cuenta también nodos intermedios en el árbol para utilizar el conocimiento disponible para los objetos considerados. Así, por ejemplo, objetos como triángulos, cuadrados, circunferencias, etc, pueden ser hallados directamente si se conocen métodos directos para calcularlos, sin necesidad de volver a expresarlos en función de componentes como vértices o coordenadas.

El grado de expansión necesario para resolver un subproblema asociado al problema inicial, que se puede medir mediante la profundidad de desarrollo de expansión en el árbol que la representa, depende del grado de conocimiento que se dispone en los patrones de solubilidad. Cuanto más avanzado es el nivel de resolución, menos desarrollo de la expansión es necesario permitiendo resolver mas eficazmente un problema. Así, por ejemplo, si se dispone de un conocimiento sobre Geometría que asegura que el centro de la circunferencia inscrita a un triángulo es el incentro (y de un procedimiento de cálculo hallando la intersección de las bisectrices), entonces no se necesitan realizar más cálculos sobre las coordenadas del centro y el radio de dicha circunferencia para obtener la circunferencia inscrita a un triángulo dado.

Así mismo, el grado de expansión depende también de la definición que se hace en las Bases de Conocimiento de los tipos atómicos, puesto que estos son los nodos terminales en el proceso de expansión. Así, por ejemplo, alternativamente se podría expresar un número racional desarrollándolo mediante el cociente de dos enteros para determinados problemas en donde se sepan resolver cuestiones sobre números enteros. En este sentido, otro factor que determina el grado de expansión es el desarrollo que se hace de las Bases de Conocimiento por el experto que depende del tipo de estrategias que construye sobre subproblemas en los patrones de solubilidad.

4.3.2 Función EXPANDE

En este apartado se describe en detalle el comportamiento de EXPANDE. Dentro de PRÓGENES se define la función EXPANDE que utiliza a las Bases de Conocimiento para definir un proceso recursivo de expansión en expresiones de asignación de tipo. La función EXPANDE se aplica a una expresión de asignación de tipo para obtener como

resultado varias expresiones de asignación de tipo construidas según se especifica más adelante.

EXPANDE hereda el carácter recursivo de la función TIPO para construir objetos a partir de otros más elementales cuyos tipos son subtipos de los tipos especificados como argumentos en las definiciones de las Bases de Conocimiento para objetos compuestos. En este contexto, utiliza este conocimiento inverso para expandir asignaciones de tipo referidas a tales objetos. Por ejemplo, en el caso sencillo de un objeto construido a partir de otros dos más elementales:

```
(obj <tipo> ((<nombre1> <TIPO1>) (<nombre2> <TIPO2>))
     <condicion>)
```

se verifica que:

```
EXPANDE(<> ?c <TIPO>) :=
   (<tipo> (<> ?x <TIPO1>) (<> ?y <TIPO2))
```

además de obtener que <condicion> se debe verificar. Obsérvese que EXPANDE utiliza el axioma de expansión:

```
Si (<> ?c <TIPO>)
Entonces existen (<> ?x <TIPO1>), (<> ?y <TIPO2>) tales que
        (y (= ?c (<tipo> ?x ?y))
           <condicion>)
```

En el caso de partir de otro tipo de asignaciones como las de tipos atómicos, que no se pueden desarrollar en función de otros más elementales, o de tipos como CONJUNTO y FUNCION, en donde no existe una representación predefinida en función de otros objetos, la función EXPANDE deja invariante dichas asignaciones de tipo. En este caso, decimos que estas asignaciones de tipo son no expandibles y, si suponemos una asignación genérica (<> ?x <TIPO>), se verifica:

```
EXPANDE(<> ?x <TIPO>) := (<> ?x <TIPO>)
```

En el caso de que se utilice una expresión general de asignación de tipo, se aplican las definiciones anteriores de EXPANDE para las asignaciones de tipo expandibles que

aparecen como subexpresiones de dicha expresión. Por tanto, EXPANDE da como resultado tantas expresiones de asignación de tipo como asignaciones de tipo expandibles posee la inicial, y cada una de ellas se obtiene sustituyendo en la expresión inicial la aplicación de EXPANDE a dicha asignación de tipo expandible.

La transformación de EXPANDE se realiza a un nivel obteniendo varias expresiones de asignación de tipo a partir de una inicial. Si a cada una de estas expresiones resultantes se le vuelve a aplicar EXPANDE, se obtienen nuevas expresiones. De este modo, en el proceso de resolución, se obtiene un árbol en donde en cada nodo existiría una expresión de asignación de tipo. El conjunto de nodos son los estados intermedios de expansión, distintos entre sí, de la expresión de asignación de tipo inicial. En AERP se utilizan estos estados intermedios para utilizar, si es posible, métodos de resolución directa según el conocimiento compilado que existe en los patrones de solubilidad.

4.4 Descripción de AERP

En esta sección se describe el Algoritmo de Expansión en Resolución de Problemas (AERP) dentro de PRÓGENES. AERP utiliza a la función EXPANDE y a los patrones de solubilidad introducidos en las secciones precedentes para definir un método de búsqueda de la solución para un problema en que hay que determinar un objeto. AERP es un caso especial de razonamiento basado en el objetivo que utiliza el axioma de expansión para definir una heurística de búsqueda de un objeto.

En los siguientes apartados se describe en detalle el funcionamiento de AERP. En primer lugar, se da un ejemplo en donde se ilustran estas técnicas, 4.4.1. Más adelante, se describe el funcionamiento general de AERP, 4.4.2, y finalmente se dan mas ejemplos de aplicación, 4.4.3 y 4.4.4.

4.4.1 Problema 7

Como ilustración de la forma en que AERP resuelve un problema, consideremos el problema 7 de la Introducción:

- Encontrar el área de la circunferencia inscrita en el triángulo de vértices $(-1,0)$, $(0,2)$ y $(1,0)$.

cuya traducción a lenguaje formal PRÓGENES es la siguiente:

```
(hallar (area (<> ?c CIRCUNFERENCIA))
        (inscrito ?c
                  (triangulo (punto -1 0) (punto 0 2) (punto 1 0))))
```

Este problema puede ser resuelto desde distintos enfoques dependiendo del grado de conocimiento utilizado. AERP no necesita utilizar conocimiento específico, expresado en forma de teoremas, sobre Geometría para resolverlo sino que aplica las definiciones de las Bases de Conocimiento para objetos como CIRCUNFERENCIA o PUNTO, y para metapredicados como inscrito.

En primer lugar, se inicializa el valor de una base de datos que está formada por un módulo de objetos buscados (MOB) y por un objetivo (MM):

```
MOB: (area (<> ?c CIRCUNFERENCIA))
MM:  (inscrito ?c (triangulo (punto -1 0) (punto 0 2) (punto 1 0)))
```

En MOB aparece una expresión de asignación de tipo que contiene la metavariable cuyo valor hay que determinar, que es ?c. A partir de esta base se crea el módulo de pistas MP, que contiene a MM, en donde se utiliza a la semántica de inscrito que afirma que la circunferencia es tangente a cada uno de los lados:

```
(inscrito ?c (triangulo (punto -1 0) (punto 0 2) (punto 1 0))) :=
(y (tangente (segmento (punto -1 0) (punto 0 2)) ?c)
   (tangente (segmento (punto 0 2) (punto 1 0)) ?c)
   (tangente (segmento (punto 1 0) (punto -1 0)) ?c))
```

A su vez, la definición de tangente se aplica obteniendo para cada uno de los lados que existe un punto en dicho lado que pertenece a la circunferencia y que, además, verifica que el segmento que lo une con el centro de la circunferencia es perpendicular al lado. Por simplicidad, se incluye solamente el caso del primer lado, obteniendo:

```
(tangente (segmento (punto -1 0) (punto 0 2)) ?c) :=
(existe (<> ?p1 (PUNTO 2))
  (y (pertenece ?p1 (segmento (punto -1 0) (punto 0 2)))
     (pertenece ?p1 ?c)
     (perpendicular (- (punto 0 2) (punto -1 0))
                    (- ?p1 (centro ?c)))))
```

Por otra parte, se crea el módulo de objetos pistas MOP que inicialmente vale MOP: (area (<> ?c CIRCUNFERENCIA)). Al aplicar las anteriores definiciones, se obtiene en el módulo de pistas una expresión boolena con constructor **existe**, con lo que se crea un objeto, ?p1, en MOP:

```
MOP: (area (<> ?c CIRCUNFERENCIA))
     (<> ?p1 (PUNTO 2))
MP:  (pertenece ?p1 (segmento (punto -1 0) (punto 0 2)))
     (pertenece ?p1 ?c)
     (perpendicular (- (punto 0 2) (punto -1 0))
                    (- ?p1 (centro ?c)))
```

Al considerar los <objetos> y <relaciones> de MOP y MP para aplicárselos a los patrones de solubilidad, no se encuentra ninguno aplicable. Aplicando la expansión en ?c, como se hizo en 4.1.1, en función de su centro ?c y su radio ?r, tampoco se consigue reducir a un problema soluble. Lo mismo ocurre al expandir (<> ?p1 (PUNTO 2)) obteniendo:

```
(punto (<> ?x1 REAL) (<> ?x2 REAL))
```

Sin embargo, al aplicar de nuevo expansión de ?c y aplicar las definiciones anteriores, se obtiene una nueva base (que es descendiente de la base inicial en la estructura de árbol de bases obtenida por expansión), que contiene a:

```
MOP: (area (circunferencia (punto (<> ?x0 REAL) (<> ?y0 REAL))
                           (<> ?r REAL)))
     (punto (<> ?x1 REAL) (<> ?y1 REAL))
     (punto (<> ?x2 REAL) (<> ?y2 REAL))
     (punto (<> ?x3 REAL) (<> ?y3 REAL))
MP:  (= (+ (- ?x1 ?x0) (* 2 (?y1 ?y0))) 0)
     (= (+ (expt (- ?x1 ?x0) 2) (expt (- ?y1 ?y0) 2)) (expt ?r 2))
     (= ?y (+ (* 2 ?x) 2))
     (= (+ (* -1 (- ?x2 ?x0)) (* -2 (?y2 ?y0))) 0)
     (= (+ (expt (- ?x2 ?x0) 2) (expt (- ?y2 ?y0) 2)) (expt ?r 2))
     (= (?y (+ (* -2 ?x) 2)))
     (= (* -2 (- ?x3 ?x0)) 0)
     (= (+ (expt (- ?x3 ?x0) 2) (expt (- ?y3 ?y0) 2)) (expt ?r 2))
     (= ?y3 0)
```

en donde en MOP aparecen ?x1, ?y1, ?x2, ?y2, ?x3 y ?y3, que son objetos que se buscan y que aparecen como consecuencia de aplicar el razonamiento basado en el objetivo que obtiene que, en caso de que la circunferencia esté inscrita en el triángulo, entonces existen los puntos de tangencia.

A partir de MOP se generan los <objetos>, y a partir de MP las <relaciones> para buscar en los patrones de solubilidad si el problema sería soluble. Por tanto, asociado a este conjunto de módulos existen los objetos siguientes:

```
<objetos>: (<> ?x0 REAL) (<> ?y0 REAL) (<> ?r REAL)
           (<> ?x1 REAL) (<> ?y1 REAL)
           (<> ?x2 REAL) (<> ?y2 REAL)
           (<> ?x3 REAL) (<> ?y3 REAL)
```

y las <relaciones> son las 9 ecuaciones anteriores que aparecen en el módulo de pistas. En este estado, el par de <objetos> y <relaciones> tiene un patrón de solubilidad aplicable puesto que <relaciones> es un sistema de ecuaciones de números reales con términos cuadráticos que se puede resolver mediante técnicas como reducción y sustitución.

El método de solución en este patrón de solubilidad obtiene los valores de estas metavariables. Una vez hallados los valores de las 9 metavariables, se comprueba que verifican el problema. Para ello, se sustituyen en el objetivo MM y se comprueba que se obtiene V. Por tanto, se sustituye en MOB los valores de todas las metavariables, obteniendo el objeto buscado como expresión canónica:

```
(area (circunferencia (punto 0 (/ (- (sqrt 5) 1) 2))
                      (/ (- (sqrt 5) 1) 2)))
```

con lo que se obtiene el resultado final (* pi (/ (- 3 (sqrt 5)) 2)).

4.4.2 Comportamiento algorítmico de AERP

La acción de AERP en PRÓGENES se desarrolla fundamentalmente en problemas de búsqueda de objetos, aunque se puede aplicar también a problemas en cuyo objetivo aparecen cuantificadores existenciales.

Dentro del esquema general de resolución descrito en 3.2.4, AERP se aplica a una base de datos a la que previamente se le han aplicado todas las transformaciones de

reducción que lleva a cabo la función CANONIZA. Cabe señalar que AERP es un tipo especial de razonamiento basado en el objetivo, pero que no se aplica en cualquier tipo de problema (ya que otras estrategias pueden tener éxito antes), sino que, si no es posible encontrar el valor de un objeto, entonces se realiza backtracking para aplicar otro tipo de estrategias.

AERP considera las expresiones de asignación de tipo que contiene el módulo de objetos pistas MOP y utiliza a la función EXPANDE para formar un árbol de expresiones de asignación de tipo distintas que resultan por expansión a partir de la inicial. Por otra parte, AERP modifica a partir de cada una de estas expansiones el módulo de pistas MP que contiene condiciones necesarias para obtener el objetivo. Además, compara cada uno de estos estados con los patrones de solubilidad y, en caso de que la función REDUCE obtenga una solución, lo sustituye en el objetivo para ver si se verifica. La acción de AERP acaba en fracaso si no se reconoce como soluble ninguno de los estados expandidos. Así, en un caso sencillo como en el problema 8 de la Introducción:

- Hallar el segmento cuyo punto medio es el punto $(0,1)$, tiene longitud 2 y es perpendicular al vector $(1,0)$.

con enunciado formal:

```
(hallar (<> ?s SEGMENTO)
        (y (= (punto-medio ?s) (punto 0 1))
           (= (longitud ?s) 2)
           (perpendicular ?s (vector 1 0))))
```

se obtienen los siguientes 5 estados posibles de expansión, de acuerdo a la estructura de árbol ilustrada en (25):

```
MOP: (<> ?s SEGMENTO)
MP:  (= (punto-medio ?s) (punto 0 1))
     (= (longitud ?s) 2)
     (perpendicular (segmento ?p1 ?p2) (vector 1 0))

MOP: (segmento (<> ?p1 (PUNTO 2)) (<> ?p2 (PUNTO 2)))
MP:  (= (/ (+ ?p1 ?p2) 2) (punto 0 1))
     (= (longitud (segmento ?p1 ?p2)) 2)
     (perpendicular (- ?p1 ?p2) (vector 1 0))

MOP: (segmento (punto (<> ?x1 REAL) (<> ?y1 REAL)) (<> ?p2 (PUNTO 2)))
MP:  (= (/ (+ (punto ?x1 ?y1) ?p2) 2) (punto 0 1))
     (= (longitud (segmento (punto ?x1 ?y1) ?p2)) 2)
     (perpendicular (- (punto ?x1 ?y1) ?p2) (vector 1 0))

MOP: (segmento (<> ?p1 (PUNTO 2)) (punto (<> ?x2 REAL) (<> ?y2 REAL)))
MP:  (= (/ (+ ?p1 (punto ?x2 ?y2)) 2) (punto 0 1))
     (= (longitud (segmento ?p1 (punto ?x2 ?y2))) 2)
     (perpendicular (- ?p1 (punto ?x2 ?y2)) (vector 1 0))

MOP: (segmento (punto (<> ?x1 REAL) (<> ?y1 REAL))
               (punto (<> ?x2 REAL) (<> ?y2 REAL)))
MP:  (= (/ (+ ?x1 ?x2) 2) 0)
     (= (/ (+ ?y1 ?y2) 2) 1)
     (= (sqrt (+ (expt (- ?x1 ?x2) 2) (expt (- ?y1 ?y2) 2))) 2)
     (= (- ?x1 ?x2) 0)
```

para los cuales sólo el quinto puede ser resuelto por REDUCE, puesto que se trata de un sistema de ecuaciones con términos cuadráticos que se puede resolver.

En los siguientes apartados se describe en detalle la forma en la que se resuelven más problemas aplicando AERP.

4.4.3 Problema 9

Aplicando AERP se pueden resolver gran cantidad de problemas en un dominio como Geometría. Consideremos el ejemplo 9 de la Introducción:

- Encontrar el plano que contiene a los puntos $(4,0,0)$ y $(0,8,0)$ y es tangente al elipsoide centrado en el origen cuyos ejes son paralelos a los ejes de coordenadas con longitudes 1, 2 y 1.

cuyo enunciado en lenguaje formal es:

```
(hallar (<> ?p PLANO)
        (y (pertenece (punto 4 0 0) ?p)
           (pertenece (punto 0 8 0) ?p))
           (tangente ?p (elipsoide (punto 0 0 0) 1 2 1)))
```

En este caso, la acción de `CANONIZA` obtiene la siguiente base de datos inicial:

```
MOB: (<> ?p plano)
MM:  (y (tangente ?p (elipsoide (punto 0 0 0) 1 2 1))
        (pertenece (punto 4 0 0) ?p)
        (pertenece (punto 0 8 0) ?p))
```

Por otra parte, se utiliza la definición del metapredicado **tangente**, que tiene `condicion-aplicacion F`, que produce la siguiente transformación:

```
(tangente ?p (elipsoide (punto 0 0 0) 1 2 1)) :=
(existe (<> ?q (PUNTO 3))
  (y (pertenece ?q ?p)
     (pertenece ?q (elipsoide 1 2 1))
     (perpendicular (vector-normal
                        (elipsoide (punto 0 0 0) 1 2 1) ?q)
                    ?p)))
```

Además utilizando razonamiento basado en el objetivo se crean los módulos `MOP` y `MP`, que tiene la forma:

```
MOP: (<> ?p PLANO)
     (<> ?q (PUNTO 3))
MP:  (pertenece (punto 4 0 0) ?p)
     (pertenece (punto 0 8 0) ?p))
     (pertenece ?q ?p)
     (pertenece ?q (elipsoide 1 2 1))
     (perpendicular (vector-normal (elipsoide 1 2 1) ?q) ?p))
```

La función EXPANDE se utiliza para expandir las metavariables ?p y ?q [1]:

```
EXPANDE(<> ?p PLANO) :=
  (plano (<> ?a REAL) (<> ?b REAL) (<> ?c REAL) (<> ?d REAL))
EXPANDE(<> ?q (PUNTO 3)) :=
  (punto (<> ?x REAL) (<> ?y REAL) (<> ?z REAL))
```

Por otro lado, también se aplica metaevaluación obteniendo las siguientes transformaciones:

```
(pertenece (punto 4 0 0) (plano ?a ?b ?c ?d)) :=
    (= (+ (* 4 ?a) ?d) 0)
(pertenece (punto 0 8 0) (plano ?a ?b ?c ?d)) :=
    (= (+ (* 8 ?b) ?d) 0)
(pertenece (punto ?x ?y ?z) (plano ?a ?b ?c ?d)) :=
    (= (+ (* ?x ?a) (* ?y ?b) (* ?z ?c) ?d) 0)
(pertenece (punto ?x ?y ?z) (elipsoide 1 2 1)) :=
    (= (+ (expt ?x 2) (/ (expt ?y 2) 4) (expt ?z 2)) 1)
(perpendicular (vector-normal (elipsoide 1 2 1) (punto ?x ?y ?z))
               (plano ?a ?b ?c ?d)) :=
    (y (= (* 4 ?x ?b) (* ?y ?a))
       (= (* ?y ?c) (* 4 ?z ?b)))
```

Por tanto, la expansión completa del problema da lugar a una base descendiente de la inicial, y la aplicación de metavaluacion permite obtener un sistema de ecuaciones que se pueden resolver. En este caso, se ha creado un módulo de pistas MP que contiene seis ecuaciones con siete incógnitas, que aparecen en MOP, y en donde aparecen términos cuadráticos. Por tanto, se resuelve el sistema aplicando el correspondiente patrón de solubilidad al par siguiente:

```
<objetos>:    (<> ?a REAL) (<> ?b REAL) (<> ?c REAL) (<> ?d REAL)
              (<> ?x REAL) (<> ?y REAL) (<> ?z REAL)
<relaciones>: (= (+ (* 8 ?b) ?d) 0)
              (= (+ (* 4 ?a) ?d) 0)
              (= (+ (expt ?x 2) (/ (expt ?y 2) 4) (expt ?z 2)) 1)
              (= (+ (* ?x ?a) (* ?y ?b) (* ?z ?c) ?d) 0)
              (= (* 4 ?x ?b) (* ?y ?a))
              (= (* ?y ?c) (* 4 ?z ?b))
```

[1] La expresión (plano a b c d) representa al plano con ecuación implícita $ax + by + cz + d = 0$.

En este caso, se obtienen dos soluciones y una dependencia en función de un parámetro, (<> ?d REAL), obteniendo las soluciones siguientes:

```
(plano (/ ?d -4) (/ ?d -8) (* (sqrt (/ 7 8)) ?d) ?d)
(plano (/ ?d -4) (/ ?d -8) (* (- (sqrt (/ 7 8))) ?d) ?d)
```

que corresponden a los planos $-\frac{d}{4}x - \frac{d}{8}y \pm \sqrt{\frac{7}{8}}dz + d = 0$

4.4.4 Problema 10

El algoritmo de expansión también se puede aplicar a problemas de Física considerando patrones de solubilidad correspondientes a ecuaciones numéricas o diferenciales. Así, por ejemplo, consideremos el problema 10 de la introducción:

- Sea un sistema mecánico unidimensional constituido por una partícula de masa 2 y carga 1, y por un campo de fuerzas formado a partir de las siguientes interacciones: un campo eléctrico de valor 5, el campo gravitatorio terrestre, una fuerza de fricción de coeficiente 4 y una fuerza elástica de constante de recuperación 3. Hallar la posición de la partícula en el instante 1 sabiendo que su posición y velocidad iniciales son nulas.

cuya traducción a Lenguaje Formal es la siguiente:

```
(hallar
   (imagen (posicion (particula (<> ?s SISTEMA-MECANICO))) 1)
   (y (= (masa (particula ?s)) 2)
      (= (carga (particula ?s) 1))
      (= (campo-fuerzas ?s)
         (funcion ((<> ?mm REAL) (<> ?qq REAL) (<> ?tt REAL)
                   (<> ?xx REAL) (<> ?xx' REAL))
           (+ (* -3 ?xx) (* -4 ?xx') (* 10 ?mm) (* 5 ?qq))))
      (= (imagen (posicion (particula ?s)) 0) 0)
      (= (imagen (velocidad (particula ?s)))0) 0))
```

En este problema se consideran tipos como SISTEMA-MECANICO y PARTICULA, definidos en la Base de Conocimiento mediante:

```
(obj sistema-mecanico ((particula PARTICULA)                    (26)
                       (campo-fuerzas (FUNCION-REAL 5 1)))
     V)

(obj particula ((masa REAL) (carga REAL)
                (posicion FUNCION-REAL))
     (> masa 0))
```

Por otra parte, asociado a un sistema mecánico existe un teorema (la ley de Newton):

```
(teor (para-todo (<> ?s SISTEMA-MECANICO)
         (= (fuerza ?s)
            (* (masa (particula ?s)) (aceleracion (particula ?s))))))
```

en donde la aceleración y la fuerza están definidas como sigue:

```
(aceleracion (<> ?p PARTICULA)) := (derivada (posicion ?p) 2)
```

y

```
(fuerza (sistema-mecanico
          (particula (<> ?m REAL) (<> ?q REAL) (<> ?x FUNCION-REAL))
          (<> ?f (FUNCION-REAL 5 1)))) :=
  (composicion ?f
               (funcion (<> ?t REAL)
                 (?t ?m ?q (imagen ?x ?t) (imagen (derivada ?x) ?t))))
```

En este problema, a partir de las definiciones de objetos de las Bases de Conocimiento anteriormente introducidas, se genera una base de datos nieta de la inicial aplicando expansión, de modo que el módulo de objetos pistas tiene la siguiente expresión antes de ser metaevaluada:

```
MOP: (imagen
       (posicion
         (particula
           (sistema-mecanico
             (particula
               (<> ?m REAL) (<> ?q REAL) (<> ?x FUNCION-REAL))
             (<> ?f (FUNCION-REAL 5 1)))))
       1)
```

Por otro lado, se aplican las definiciones anteriores para metafunciones y se utiliza razonamiento basado en el objetivo, de modo que se crea en el módulo de pistas la ecuación que se obtiene el aplicar la ley de Newton, además de utilizar las demás ecuaciones del enunciado, con lo que el par de objetos y relaciones que se obtiene a partir de MOP y MP es:

```
<objetos>:     (<> ?x FUNCION-REAL)
<relaciones>: (= (+ (* -3 ?x) (* -4 (derivada ?x)) 25)
                 (* 2 (derivada ?x 2)))
               (= (imagen ?x 0) 0)
               (= (imagen (derivada ?x) 0) 0)
```

El patrón de solubilidad (24) es aplicable al par anterior de modo que esta ecuación diferencial de segundo orden con dos condiciones iniciales se transforma en un sistema de dos ecuaciones diferenciales ordinarias lineales. De nuevo, se puede aplicar el otro patrón de solubilidad (23) al valor obtenido por la función REDUCE (según el patrón anterior) y se resuelve el sistema anterior obteniendo como solución una función que al sustituir en el instante 1 da:

```
(+ (/ (- (* 25 (sqrt 2))
         (* (* 25 (sqrt 2)) (exp (- -1 (/ i (sqrt 2))))))
      (+ (* -4 i) (expt 2 3/2)))
   (/ (- (* 25 (sqrt 2))
         (* (* 25 (sqrt 2)) (exp (+ -1 (/ i (sqrt 2))))))
      (+ (* 4 i) (expt 2 3/2))))
```

que corresponde a:

$$\frac{25\sqrt{2}-25\sqrt{2}\exp(-1-\frac{i}{\sqrt{2}})}{-4i+2^{\frac{3}{2}}}+\frac{25\sqrt{2}-25\sqrt{2}\exp(-1+\frac{i}{\sqrt{2}})}{4i+2^{\frac{3}{2}}}$$

Capítulo 5

Conclusiones

En este capítulo se hace una recapitulación acerca del tipo de técnicas desarrolladas, relacionándolas con las utilizadas en otros sistemas previos e ilustrando las principales aportaciones de PRÓGENES en el campo de la Resolución Automática de Problemas.

PRÓGENES integra muchas de las capacidades de deducción de otros sistemas de Demostración Automática, por un lado, y de manipulación simbólica y de cálculo, por otro. Esta integración permite desarrollar heurísticas para la resolución de problemas que estan fuera del alcance de todos estos sistemas.

PRÓGENES, inspirado en ideas de MUSCADET, [Pastre 89], organiza los datos en una estructura de módulos y aplica metarreglas mediante un motor de inferencia *forwards*. Además utiliza, al igual que MUSCADET las heurísticas de Bledsoe de preprocesamiento de fórmulas logicas, [Bledsoe 71]. Por otro lado, PRÓGENES, análogamente a ONTIC, [McAllester 89], utiliza los objetos relevantes a un problema para controlar y dirigir la búsqueda de una solución. Por último, en PROGENES parte del conocimiento se representa mediante reglas de reescritura del tipo de las usadas en sistemas como MATHEMATICA, [Wolfram 91]. Este mismo mecanismo se utiliza para definir transformaciones de reducción y de simplificación en conjuntos de expresiones.

En resumen, PRÓGENES se adecua especialmente para integrar métodos deductivos junto con técnicas de cálculo y de manipulación algebraica permitiendo resolver un amplio espectro de los problemas que se resuelven, por ejemplo, en un primer curso de una carrera de Ciencias.

PRÓGENES utiliza gran parte del conocimiento utilizado a este nivel por un estudiante en estos dominios, de modo que resulta altamente eficiente para ser aplicado en la resolución de un gran porcentaje de problemas. Tras efectuar un análisis de los

problemas de Cálculo Infinitesimal correspondientes a algunos capítulos representativos de un libro de texto típico utilizado en un primer curso de estudios universitarios (*CALCULUS, de una y varias variables*, [Salas, Hille 82]), (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), se observa que el porcentaje de problemas tratables por los métodos descritos en esta tesis es, en promedio, de un setenta y cinco por ciento.

Además cabe señalar que los sistemas de Demostración Automática de Teoremas resolverían un porcentaje muy pequeño de estos problemas, y que los sistemas de manipulación simbólica y algebraica, como MATHEMATICA [Wolfram 91], podrían resolver sólo aquéllos en donde se aplica un cálculo directo.

Además, las téecnicas desarrolladas en PRÓGENES permiten una resolución eficiente de los problemas debido a que se lleva a cabo un control del proceso de deducción mediante la asignación de prioridades a grupos de reglas y tareas. Así, la aplicación de reglas de reescritura tiene prioridad máxima y el razonamiento en modo *forwards* tiene prioridad sobre el razonamiento basado en el objetivo, que a su vez la tiene sobre el razonamiento en modo *backwards*.

Por último, el tipo de conocimiento utilizado en PRÓGENES, construido a partir del utilizado por una persona en este tipo de dominios puede utilizarse como punto de partida para el desarrollo de un sistema interactivo de enseñanza en materias científicas, [Bazin, Castells, Moriyón, Saiz 93], [Castells, Moriyón, Saiz, Villa 93, a], [Castells, Moriyón, Saiz, Villa 93, b].

Referencias

[Anderson 83] Anderson J. R., *The Architecture of Cognition*, Harvard University Press, Cambridge, Massachusets, 1983.

[Bazin, Castells, Moriyón, Saiz 93] Bazin J.M., Castells P., Moriyón R. and Saiz F., *A Knowledge Based Problem Solver Conceived for Intelligent Tutoring Applications*, Proceedings ICCTE'93, V. Petrushin and A. Dovgiallo (Eds.), Kiev (Ukraine), 1993, pp. 79-80, 1993.

[Bledsoe 71] Bledsoe W. W., *Splitting and Reduction Heuristics in Automatic Theorem Proving* Artificial Intelligence 2, 1971, pp. 55-77.

[Bledsoe 72] Bledsoe W.W., Boyer R.S. and Henneman W.H., *Computer Proofs of Limit Theorems*, Artificial Intelligence 3, 1972, pp. 27-60.

[Bledsoe, Bruell 74] Bledsoe W.W. and Bruell P., *A Man-Machine Theorem-Proving System*, Artificial Intelligence 5, 1974, p.p. 51-72.

[Bledsoe 77] Bledsoe W.W., *Non-resolution Theorem Proving*, Artificial Intelligence 9, 1977, pp. 1-35.

[Booch 91] Booch G., *Object Oriented Design with applications*, Publishing Company, Inc., 1991.

[Boyer, Moore 79] Boyer R.S. and Moore J.S., *A Computational Logic*, Academic Press, Inc., 1979.

[Boyer, Moore 88] Boyer R.S. and Moore J.S., *A Computational Logic Handbook*, Academic Press, Inc., 1988.

[Bundy 1979] Bundy A., Byrd L., Luger G., Mellish C. and Palmer M.. MECHO: a program to solve mechanics problems, Working paper N.50, Dept. of Artificial Intelligence, Edinburgh, 1979.

[Castells et al. 91, a] Castells P., Díaz J., Gonzalo J., Moriyón R., Rodríguez-Marín P., Saiz F. and Tobar M.J., *A Scientific Problem Solver with a Natural Language Interface*, Proceedings ISCIS'92, Presses EHEI, Paris, 1991, pp.237-243.

[Castells et al. 91, b] Castells P., Díaz J., Moriyón R., Rodríguez-Marin P. y Saiz F., *Progenes: Una Base de Conocimiento para la resolución de problemas de Cálculo Infinitesimal*, Actas AEPIA'91, Madrid, 1991, pp. 17-30.

[Castells et al. 92] Castells P., Díaz J., Gonzalo J.,Moriyón R., Rodríguez-Marin P., Saiz F. y Tobar M.J., *Un sistema para la resolución de problemas enunciados en Lenguaje Natural* Actas PRODE'92, Facultad de Informática, UPM, Madrid, 1992, pp. 232-245,

[Castells, Moriyón, Saiz, Villa 93, a] Castells P., Moriyón R., Saiz F. and Villa E., *A Model of Knowledge in Elementary Mechanics*, Proceedings ICCE'93, Taiwan, December 1993, pp. 319-321.

[Castells, Moriyón, Saiz, Villa 93, b] Castells P., Moriyón R., Saiz F., and Villa E., *Application of Techiques of Automatic Problem Solving to Computer Intelligent Tutoring.* Proceedings ICCTE'93, V. Petrushin and A. Dovgiallo (Eds.), Kiev, Ukraine, September 1993, pp. 82-83.

[Castells, Moriyón, Saiz 93, a] Castells P., Moriyón R. y Saiz F., *Jerarquías de objetos en la resolución de problemas científicos*, Actas CAEPIA'93, Madrid, November 1993, pp. 89-99.

[Castells, Moriyón, Saiz 93, b] Castells P., Moriyón R., and Saiz F., *PROGENES: Using Metaknowledge to Solve Scientific Problems*, Technical Report IIC B06/93, 1993.

[Castells, Moriyón, Saiz 93, c] Castells P., Moriyón R., and Saiz F. *A goal-based reasoning heuristic in Automatic Problem Solving*, Technical Report IIC B05/93, 1993.

[Castells, Moriyón, Saiz 94] Castells P., Moriyón R., and Saiz F. *PROGENES: An Automatic Problem Solver based on Mathematica*, Technical Report IIC B02/94, 1994.

[Castells 94] Castells P., *Heurísticas y Metaconocimiento en Resolución Automática de Problemas*, Tesis Doctoral, Dep. Ingeniería Informática (UAM), 1994.

[Church 40] Church A., *A formulation of the simple theory of types*, Journal of Simbolic Logic, 5(1) pp. 56-68, 1940.

[Church 51] Church A., *The calculi of lambda-conversion*, Annals of Mathematics Studies, No. 6. Princeton University Press, Princeton, NJ, 1951.

[Clarke 93] Clarke E. and Zhao X., *ANALYTICA: A Theorem Prover for Mathematica*, The Mathematica Journal, Vol. 3, Issue 1, Winter 1993, pp. 56-71.

[Constable et al. 86] Constable R.L. et al., *Implementing Mathematics with the Nuprl Proof Development System*, Prentice-Hall, Inc., 1986.

[Corella 90] Corella F., *Mechanizing set theory*, PhD thesis, University of Cambridge, 1990.

[Davis, Weyuker 83] Davis M.D. and Weyuker E.J., *Computability, complexity and languages*, Academic Press, Inc., 1983.

[Ebbinghaus, Flum, Thomas 84] Ebbinghaus H.D., Flum J. and Thomas W., *Mathematical Logic*, Springer-Verlag, 1984.

[Gödel 30] Godel K., *Die Vollstandigkeit der Axiome des Logischen Funktionenkalkuls*, Monatsh. Math. Phys. 37 (1930), 349-360.

[Gordon, Milner, Wadsworth 79] Gordon M.J.C., Milner R. and Wadsworth C.P., *Edinburgh LCF: A Mechanised Logic of Computation*, Springer LNCS 78, 1979.

[Gordon 87] Gordon M.J.C., *HOL: A proof generating system for higher-order logic*, Report 103, Computer Laboratory, University of Cambridge, 1987.

[Harper, MacQueen, Milner 86] Harper R., MacQueen B. and Milner R., *Standard ML*, Report ECS-LFCS-86-2, Laboratory of Computer Science, University of Edinburgh, 1986.

[Herbrand 71] Herbrand J., *Logical Writings*, (Warren D. Goldfarb, ed.) Harvard University Press, 1971.

[Jech 78] Jech T., *Set Theory*, Academic Press, Inc., 1978.

[McAllester 89] McAllester D.A., *Ontic: a knowledge representation system for Mathematics*, Massachussets Institute of Technology, 1989.

[Minsky 75] Minsky M., *A framework for representing knowledge*, Reprinted in Haugeland, 95-128, 1981.

[Moreno, Rodríguez 88] Moreno J.J and Rodríguez-Artalejo M., *BABEL: A Functional and Logic Programming Language based on a constructor discipline and narrowing*, in I. Grabowski, P. Lescanne and W. Wechler, editors, Algebraic and Logic Programming, volume 343 of Lecture Notes in Computer Science, pp. 223-232. Springer-Verlag, Berlin, 1988.

[Nilsson 80] Nilsson N.J., *Principles of Artificial Intelligence*, Tioga Publishing Co., 1980.

[Pastre 89] Pastre D., *An Automatic Theorem Proving System Using Knowledge and Metaknowledge*, Artificial Intelligence 38, 1989, pp. 257-318.

[Paulson 87] Paulson L.C., *Logic and computation. Interactive proof with Cambridge LCF*, Cambridge University Press, 1987.

[Pitrat 90] Pitrat J., *Metaconnaissance, futur de l'intelligence artificielle*, Hermes, Paris, 1990.

[Polya 65] Polya G., *Mathematical discovery* John Wiley and Sons, Inc., 1965.

[Robinson 65] Robinson J.A., *A machine-oriented logic based on the resolution principle*, J. ACM 12, 1965, pp.23-41.

[Roussel 75] Roussel Ph., *PROLOG: Manuel de reference et d'utilisation*, Groupe d'Intelligence Artificielle, Marseille-Lumigny, 1975.

[Salas, Hille 82] Salas S.L. y Hille E., *CALCULUS, de una y varias variables*, Ed. Reverte, 1982.

[Scott 70] Scott D., *Constructive validity*, in Symposium on Automatic Demostration, Lecture Notes in Mathematics, Vol. 125. Springer-Verlag, New York, 1970, pages 237-275.

[Sowa 84] Sowa J., *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, Reading, Massachusets.

[Winston, Horn 89] Winston P.H. and Horn B.K.P., *LISP, 3rd Edition*, Addison-Wesley Publishing Company, Inc., 1989.

[Wang 60] Wang H., *Towards mechanical mathematics*, IBM J. Res. Develop. 4, 1960, 2-22.

[Wolfram 91] Wolfram S., *Mathematica, A System for Doing Mathematics by Computer*, Wolfram Research, Inc., 1991.

Apéndice

```
(cons e REAL)

(cons pi REAL)

(cons i COMPLEJO)

(cons R (conjunto (<> ?x REAL) V))

(cons RR (conjunto ((<> ?x REAL) (<> ?y REAL)) V))

(atom es-natural NATURAL)

(atom es-entero ENTERO)

(atom es-racional RACIONAL)

(atom es-real REAL)

(atom es-complejo COMPLEJO)

(obj punto (&rest (coordenadas REAL))
     V)

(obj vector (&rest (coordenadas REAL))
     V)

(obj segmento ((punto-1 (PUNTO 2) (punto-2 (PUNTO 2))))
     (no (= punto-1 punto-2)))
```

```
(obj circunferencia ((centro (PUNTO 2)) (radio REAL))
     (> radio 0))

(obj elipse ((centro (PUNTO 2)) (a REAL) (b REAL))
     (y (> a 0) (> b 0)))

(obj curva (funcion (FUNCION-REAL 2 1))
     (para-todo (<> ?x REAL)
        (no (= 0 (imagen (gradiente funcion) ?x)))))

(obj esfera ((centro (PUNTO 3)) (radio REAL))
     (> (radio 0)))

(obj elipsoide ((centro (PUNTO 3)) (a REAL) (b REAL) (c REAL))
     (y (> a 0) (> b 0) (> c 0)))

(obj plano ((a REAL) (b REAL) (c REAL) (d REAL))
     (no (= (a b c) (0 0 0))))

(obj superficie-revolucion
     ((eje-revolucion EJE) (funcion-generatriz FUNCION-REAL))
     (> funcion-generatriz 0))

(obj superficie (funcion (FUNCION-REAL 3 1))
     (para-todo (<> ?p (PUNTO 3))
        (no (= 0 (imagen (gradiente funcion) ?p)))))

(obj proyectil ((angulo-tiro REAL) (velocidad-inicial REAL))
     (y (> velocidad-inicial 0)
        (< 0 angulo-tiro (/ pi 2)))
     V)

(obj particula ((masa REAL) (carga REAL) (posicion FUNCION-REAL))
     (> masa 0))
```

```
(obj sistema-mecanico ((particula PARTICULA)
                       (campo-fuerzas (FUNCION-REAL 5 1)))
     V)

(obj el-conjunto (&rest (objetos OBJETO)) V CONJUNTO)

(obj conjunto ((expr-asign EXPR-ASIGN-TIPO) (cond BOOL))
     (tipo-conjunto-p (tipo-asignado (tipo expr-asign))))

(obj funcion ((expr-asign EXPR-ASIGN-TIPO) (def OBJETO)
              &optional
                (dominio CONJUNTO (conjunto expr-asign V))
                (rango CONJUNTO
                       (conjunto
                         (<> ?x (tipo def (asignaciones expr-asign)))
                         V)))
     (y (tipo-conjunto-p (tipo-asignado (tipo expr-asign)))
        (contenido dominio (conjunto expr-asign V))
        (contenido (conjunto-imagen :self dominio) rango)))

(fun plano-tangente ((s SUPERFICIE) (p (PUNTO 3)))
     PLANO
     (pertenece p s)
     (plano (vector-normal s p) p)
     (condicion-aplicacion V))

(fun vector-normal ((superficie SUPERFICIE) (punto (PUNTO 3)))
     (VECTOR 3)
     (pertenece punto superficie)
     (/ (imagen (gradiente superficie) punto)
        (modulo (imagen (gradiente superficie) punto)))
     (condicion-aplicacion V))

(fun distancia ((p1 PUNTO) (p2 PUNTO))
     REAL
     (= (dimension p1) (dimension p2))
     (distancia-euclidea p1 p2)
     (condicion-aplicacion V))
```

```
(fun distancia ((s1 CONJUNTO) (s2 CONJUNTO))
     REAL
     (y (contenido s1 RR) (contenido s2 RR))
     (infimo
       (funcion ((<> ?p1 (PUNTO 2)) (<> p2 (PUNTO 2)))
                (distancia ?p1 ?p2)
                :dominio
                  (conjunto
                    ((<> ?p1 (PUNTO 2)) (<> p2 (PUNTO 2)))
                    (y (pertenece ?p1 s1) (pertenece ?p2 s2))))))

(fun punto-medio ((segmento SEGMENTO))
     (PUNTO 2)
     V
     (/ (+ (punto-1 segmento) (punto-2 segmento)) 2)
     (condicion-aplicacion V))

(fun longitud ((segmento SEGMENTO))
     REAL
     V
     (sqrt (+ (expt (- (coordenada-1 (punto-1 segmento))
                       (coordenada-1 (punto-2 segmento)))
                    2)
              (expt (- (coordenada-2 (punto-1 segmento))
                       (coordenada-2 (punto-2 segmento)))
                    2)))
     (condicion-aplicacion V))
```

```
(fun trayectoria ((proyectil PROYECTIL))
     FUNCION-REAL
     V
     (el (<> ?f FUNCION-REAL)
         (= (imagen ?f 0) 0)
         (= (imagen (derivada ?f) 0) (tan (angulo-tiro proyectil)))
         (= (derivada ?f 2)
            (/ (* -10 (+ 1 (expt (tan (angulo-tiro proyectil)) 2)))
               (expt (velocidad-inicial proyectil) 2)))))

(fun fuerza ((sist SISTEMA-MECANICO))
     FUNCION-REAL
     V
     (composicion
       (campo-fuerzas sist)
       (funcion (<> ?t REAL)
                (?t (masa (particula sist)) (carga (particula sist))
                 (imagen (posicion (particula sist)) ?t)
                 (imagen (velocidad (particula sist)) ?t))))
     (condicion-aplicacion V))

(fun aceleracion ((part PARTICULA))
     FUNCION-REAL
     V
     (derivada (posicion part) 2)
     (condicion-aplicacion V))

(fun velocidad ((part PARTICULA))
     FUNCION-REAL
     V
     (derivada (posicion part))
     (condicion-aplicacion V))
```

```
(fun incentro ((triang TRIANGULO))
     (PUNTO 2)
     V
     (punto-interseccion (bisectrices triang))
     (condicion-aplicacion V))

(fun area-transversal ((x REAL) (sup SUPERFICIE-REVOLUCION))
     V
     (* pi (expt (imagen (funcion-generatriz sup) x) 2))
     (condicion-aplicacion V))

(fun volumen-revolucion ((intervalo (REAL REAL))
                         (sup SUPERFICIE-REVOLUCION))
     V
     (integral
        intervalo
        (funcion (<> ?x real)
           (* pi (expt (imagen (funcion-generatriz sup) ?x) 2))))
     (condicion-aplicacion V))

(fun imagen ((funcion (expr-asign EXPR-ASIGN-TIPO) (def OBJETO))
             (x OBJETO))
      (subtipo (tipo x) (tipo-asignado (tipo expr-asign)))
      (instancia x def))

(pred tangente ((s1 SUPERFICIE) (s2 SUPERFICIE) (p (PUNTO 3)))
      (y (pertenece p s1) (pertenece p s2))
      (paralelo (vector-normal s1 p) (vector-normal s2 p)))

(pred tangente ((s1 ESFERA) (s2 ESFERA) (p (PUNTO 3)))
      (y (pertenece p s1) (pertenece p s2))
      (paralelo (- p (centro s1)) (- p (centro s2)))
      (condicion-aplicacion V))
```

```
(pred tangente ((plano PLANO) (elipsoide ELIPSOIDE))
      V
     (existe (<> ?q (PUNTO 3))
       (y (pertenece ?q plano)
          (pertenece ?q elipsoide)
          (perpendicular (vector-normal elipsoide ?q) plano)))
     (condicion-aplicacion V))

(pred tangente ((seg SEGMENTO) (circ CIRCUNFERENCIA))
      V
      (existe (<> ?p1 (PUNTO 2))
        (y (pertenece ?p1 seg)
           (pertenece ?p1 circ)
           (perpendicular (- (punto-1 seg) (punto-2 seg))
                          (- ?p1 (centro circ)))))
      (condicion-aplicacion V))

(pred perpendicular ((vec1 VECTOR) (vec2 VECTOR))
      (= (dimension vec1) (dimension vec2))
      (= 0 (* vec1 vec2))
      (condicion-aplicacion V))

(pred concentricas ((c1 CIRCUNFERENCIA) (c2 CIRCUNFERENCIA))
      V
      (= (centro c1) (centro c2))
      (condicion-aplicacion V))

(pred inscrito ((circ CIRCUNFERENCIA) (triang TRIANGULO))
      V
      (y (tangente (segmento (vertice-1 triang) (vertice-2 triang)))
         (tangente (segmento (vertice-2 triang) (vertice-3 triang)))
         (tangente (segmento (vertice-3 triang) (vertice-1 triang))))
      (condicion-aplicacion V))
```

```
(pred inyectiva ((func FUNCION))
      V
      (para-todo ((<> ?x OBJETO) (<> ?y OBJETO))
        (-> (= (imagen fun ?x) (imagen fun ?y))
            (= ?x ?y))))

(pred estric-creciente ((f FUNCION-REAL))
      V
     (para-todo ((<> ?x REAL) (<> ?y REAL))
        (-> (< ?x ?y) (< (imagen fun ?x) (imagen fun ?y)))))

(pred punto-critico ((punto REAL) (funcion FUNCION-REAL))
      (derivable funcion)
      (= 0 (imagen (derivada funcion) punto))
      (condicion-aplicacion V))

(pred <-> ((b1 BOOL) (b2 BOOL))
      V
      (y (-> b1 b2) (-> b2 b1)))

(pred -> ((b1 BOOL) (b2 BOOL))
      V)

(pred pertenece ((x OBJETO)
                 (conjunto (expr-asign EXPR-ASIGN-TIPO) (cond BOOL)))
      V
      (y (subtipo (tipo x) (tipo-asignado (tipo expr-asign)))
         (instancia x cond)))

(pred contenido ((conj1 CONJUNTO) (conj2 CONJUNTO))
      V
      (para-todo (<> ?x OBJETO)
         (-> (pertenece ?x conj1) (pertenece ?x conj2))))

(pred = ((conj1 CONJUNTO) (conj2 CONJUNTO))
      V
      (y (contenido conj1 conj2) (contenido conj2 conj1)))
```

```
(pred = ((fun1 FUNCION) (fun2 FUNCION))
      (y (= (dominio fun1) (dominio fun2))
         (= (rango fun1) (rango fun2)))
      (para-todo (<> ?x OBJETO)
         (-> (pertenece ?x (dominio fun1))
             (= (imagen fun1 ?x) (imagen fun2 ?x)))))

(subtipo NATURAL ENTERO)

(subtipo ENTERO RACIONAL)

(subtipo RACIONAL REAL)

(subtipo REAL COMPLEJO)

(subtipo (c1 CIRCUNFERENCIA)
         CONJUNTO
         (conjunto (<> ?p (PUNTO 2))
                   (= (distancia ?p (centro c1)) (radio c1))))

(subtipo (plano (a REAL) (b REAL) (c REAL) (d REAL))
         SUPERFICIE
         (superficie
            (funcion ((<> ?x REAL) (<> ?y REAL) (<> ?z REAL))
                     (+ (* a ?x) (* b ?y) (* c ?z) d))))

(subtipo (curva (funcion (FUNCION-REAL 2 1)))
         CONJUNTO
         (conjunto (<> ?p (PUNTO 2))
                   ((= (imagen funcion (?x ?y)) 0))))

(subtipo (superficie (funcion (FUNCION-REAL 3 1)))
         CONJUNTO
         (conjunto (<> ?p (PUNTO 2)) (= (imagen funcion ?p) 0)))

(subtipo (e ESFERA) ELIPSOIDE
         (elipsoide (centro e) (radio e) (radio e) (radio e)))
```

```
(subtipo (elipsoide
           (punto (x0 REAL) (y0 REAL) (z0 REAL))
           (a REAL) (b REAL) (c REAL))
         SUPERFICIE
         (superficie
            (funcion ((<> ?x REAL) (<> ?y REAL) (<> ?z REAL))
                 (- (+ (/ (expt (- ?x x0) 2) (expt a 2))
                       (/ (expt (- ?y y0) 2) (expt b 2))
                       (/ (expt (- ?z z0) 2) (expt c 2)))
                    1))))

(subtipo FUNCION-REAL FUNCION)

(teor (para-todo (<> ?f FUNCION-REAL)
         (-> (y (derivable ?f) (> (derivada ?f) 0))
             (estrict-creciente ?f))))

(teor (para-todo (<> ?f FUNCION-REAL)
         (-> (estric-creciente ?f) (creciente ?f))))

(teor (para-todo (<> ?s SISTEMA-MECANICO)
         (= (fuerza ?s)
         (* (masa (particula ?s)) (aceleracion (particula ?s))))))

(teor (para-todo ((<> ?f FUNCION-REAL) (<> ?a REAL))
         (-> (y (derivable ?f) (alcanza-maximo ?f ?a))
                (punto-critico ?a ?f))))

(teor (para-todo ((<> ?f FUNCION-REAL) (<> ?a REAL))
         (-> (y (derivable ?f 2)
                (punto-critico ?a ?f)
                (< (imagen (derivada ?f 2) ?a) 0))
             (alcanza-maximo ?f ?a))))
```

```
(teor (para-todo
        ((<> ?f FUNCION) (<> ?g FUNCION) (<> ?a OBJETO))
        (-> (y (pertenece ?a (dominio ?g))
               (contenido (rango ?g) (dominio ?f)))
            (:= (imagen (composicion ?f ?g) ?a)
                (imagen ?f (imagen ?g ?a))))))

(teor (para-todo ((<> ?f FUNCION-REAL) (<> ?g FUNCION-REAL)
                  (<> ?a REAL))
        (-> (y (derivable ?f) (derivable ?g))
            (y (derivable (composicion ?f ?g))
               (:= (imagen (derivada (composicion ?f ?g)) ?a)
                   (* (imagen (derivada ?f) (imagen ?g ?x))
                      (imagen (derivada ?g) ?x))))
               (= (derivada (composicion ?f ?g))
                  (* (composicion (derivada ?f) ?g)
                     (derivada ?g)))))))

(teor (para-todo ((<> ?f FUNCION-REAL) (<> ?g FUNCION-REAL)
                  (<> ?h FUNCION-REAL))
        (:= (composicion ?f ?g ?h)
            (composicion ?f (composicion ?g ?h)))))

(teor (para-todo ((<> ?g DIFEOMORFISMO-REAL) (<> ?a REAL))
        (y (:= (imagen (derivada (funcion-inversa ?g)) ?a)
               (expt (imagen (derivada ?g)
                             (imagen (funcion-inversa ?g) ?a))
                     -1))
           (= (derivada (funcion-inversa ?g))
              (composicion (/ 1 (derivada ?g))
                           (funcion-inversa ?g)))))))

(teor (para-todo ((<> ?g FUNCION-REAL) (<> ?a REAL))
        (-> (biyectiva ?g)
            (y (:= (imagen ?g (imagen (funcion-inversa ?g) ?a)) ?a)
               (:= (imagen (funcion-inversa ?g) (imagen ?g ?a))
                   ?a))))))
```

```
(patron-sol
  (objetos (:todos REAL))
  (relaciones (:todos (= REAL REAL)))
  (condicion (:todos relaciones (ecuac-lineal-p relaciones objetos)))
  (metodo-solucion (resuelve-Cramer relaciones objetos)))

(patron-sol
  (objetos (:todos REAL))
  (relaciones (:todos (= REAL REAL)))
  (condicion (:todos (ecuac-polinomial-p relaciones objetos)))
  (metodo-solucion (intenta-resolver-sust-red relaciones objetos)))

(patron-sol
  (objetos REAL)
  (relaciones (:todos (:o (> REAL REAL) (< REAL REAL))))
  (condicion
      (:todos relaciones (inecuac-polinomial-p relaciones objetos)))
  (metodo-solucion (intenta-resolver-inec relaciones objeto)))

(patron-sol
  (objetos (:todos FUNCION-REAL))
  (relaciones (:varios edos (= FUNCION-REAL FUNCION-REAL))
              (:resto cond-inics (= REAL REAL)))
  (condicion (y (: todos edos (ecua-dif-ordin-p edos objetos))
                (condiciones-iniciales-p cond-inics objs)))
  (metodo-solucion (intenta-resolver-edos edos cond-inics)))

(patron-sol
  (objetos FUNCION-REAL)
  (relaciones (ec-dif (= FUNCION-REAL FUNCION-REAL)))
              (:resto cond-inics (= REAL REAL)))
  (condicion (y (ecuac-difer-p objetos)
                (cond-inicial-derivadas-p cond-inics)))
  (metodo-solucion (cambia-variables-ecuac-difer ec-dif cond-inics)))
```

```
(patron-inst
  (objetos (<> ?x REAL))
  (relaciones (y (:o (< (<> ?a REAL) ?x (<> ?b REAL))
                     (> (<> ?a REAL) ?x (<> ?b REAL)))
                 :nada-mas))
  (condicion V)
  (metodo-solucion (:= ?x (/ (+ ?a ?b) 2))))
```

Tabla de símbolos

AERP	Algoritmo de Expansion en Resolucion de Problemas
AMPLIA-MH	Funcion que Amplia Hechos en MH aplicando forwards
CANONIZA	Funcion que aplica SIMPLIFICA-LOGICA, METEVAL, REDUCE y AMPLIA-H
ELIMINA-Y	Elimina el conector logico Y en el objetivo
METEVAL	Funcion de metaevaluacion de una expresion. Cuando se aplica a una base actua sobre todas las expresiones de los modulos
MT	Modulo de Teoremas
MD	Modulo de Definiciones
MH	Modulo de Hechos
MO	Modulo de Objetos
MM	Modulo de Metas
MOB	Modulo de Objetos Buscados
MOP	Modulo de Objetos Pistas
MP	Modulo de Pistas
RAZONA-B	Funcion que razona backwards
RAZONA-BO	Funcion que aplica razonamiento basado en el objetivo
REDUCE	Obtiene la solucion a un subproblema aplicando patrones de solubilidad
SIMPLIFICA- ..... LOGICA	Funcion que elimina conectores logicos en una base de datos
SUBTIPO	Predicado aplicable a dos tipos
TIPO	Funcion que calcula el tipo de una expresion
TIPOS	Conjunto de tipos
TRADUCE	Funcion de traduccion de una expresion a un tipo