Fundamentos lógicos de la Programación Declarativa

José F. Quesada Moreno

El objetivo del artículo es el estudio de los fundamentos lógicos que están en la base de gran cantidad de sistemas de razonamiento automático, y que suponen una estrategia computacionalmente viable para la demostración de teoremas o el diseño de motores inferenciales (Prolog). En concreto se presenta la forma clausal, en tanto que formalismo para la representación de conocimiento, y el principio de resolución, como mecanismo inferencial que asegura la completud y corrección lógicas.

Lógica y Programación

La *programación de ordenadores* incluye un amplio espectro de técnicas, metodologías, etc., centradas en torno al problema que supone la implementación computacional de sistemas que permitan la resolución automática de determinados tipos de problemas. Esta concepción nos retrotrae inmediatamente hasta el concepto de *algoritmo*, entendido como la especificación formal del proceso que dirige la resolución y las estructuras de datos que intervienen en ella. A partir de estos conceptos se puede entender un *lenguaje de programación* como un formalismo que rige las reglas sintácticas y semánticas necesarias para la implementación computacional de algoritmos, o de forma similar, y siguiendo el modelo propuesto en [DERSHEM-90], un lenguaje de programación es un lenguaje cuya gramática ha sido diseñada específicamente para que una persona pueda expresar formalmente el proceso que un ordenador deberá seguir para resolver un problema.

Esta definición pone de manifiesto la intervención de múltiples factores en el proceso de diseño de un lenguaje de programación, entre los que destacan: *la persona*, que idea el algoritmo y lo codifica; *el ordenador*, en tanto que plataforma física y lógica que ejecuta automáticamente las órdenes programadas; *el problema*, cuya arquitectura conceptual puede verse favorecida o incluso requerir determinados aspectos del formalismo para su implementación; y por último, *el proceso* mismo de resolución. La prioridad de cada uno de estos elementos en el diseño de un lenguaje determina otras tantas cuatro metodologías o paradigmas básicos de la programación, como son la *programación procedimental* o imperativa (cuando se da mayor importancia a la perspectiva de la arquitectura del ordenador), la *orientación lógica* o programación declarativa (que pondera la perspectiva de la persona), la *programación funcional* (que asume el punto de vista del proceso) y la *orientación a objetos* (donde el foco de interés es el problema real).

Este enfoque nos permite caracterizar a los lenguajes declarativos por su interés en la dotación al programador de un entorno lo más cercano posible a los esquemas de razonamiento que son usados comúnmente por los hombres en la solución de problemas. A otro nivel, y por comparación con el paradigma imperativo, la programación declarativa se interesa por el *qué hacer* y no por el *cómo hacer*.

La perspectiva que caracteriza a la programación declarativa y por extensión a los denominados lenguajes lógicos de programación[1] genera interesantes consecuencias tanto a nivel computacional como de aplica-

[1] Los lenguajes lógicos de programación son una subclase de los lenguajes declarativos basados en la lógica simbólica, dirigidos por el objetivo, aún no alcanzado, de programar en lógica «pura».

ción, destacando ámbitos como la investigación en torno a la especificación de lenguajes formales, el procesamiento del lenguaje natural, el estudio de teorías a un nivel formal (coherencia, consistencia, demostrabilidad, etc.), la simulación de procesos regidos por leyes, etc. Asimismo es destacable el enorme potencial de aplicabilidad didáctica de estos entornos, pues la misma concepción de los mismos permite una aproximación lógica y argumentativa a los problemas, con una sintaxis y una semántica muy próximas a los mismos campos de estudio, de forma que en la implementación computacional el principal factor de esfuerzo se centra en el estudio del problema a resolver más que en el mismo proceso de codificación del algoritmo o en el entorno para su implementación.

A pesar de su interés, el objetivo básico de este artículo no lo constituye la aplicabilidad de la orientación lógica de los lenguajes de programación, sino el estudio de los fundamentos lógicos de la programación declarativa, es decir, de un conjunto de técnicas lógicas que permiten el desarrollo de sistemas capaces de realizar razonamientos (y por tanto, argumentaciones y demostraciones) de una forma *automática*. El estudio de los fundamentos teóricos nos permitirá una visión más fundada acerca del funcionamiento de estos sistemas, y consecuentemente, de sus posibilidades y sus limitaciones.

En la próxima sección abordaremos el estudio del conocimiento declarativo a nivel computacional, lo que nos permitirá obtener un modelo básico de la orientación lógica a partir de la diferenciación entre motores de inferencias y bases de conocimiento. En la sección 3 se hará un repaso de la lógica de primer orden, cuya sintaxis se utilizará como fundamento conceptual y notacional y cuyo cálculo deductivo será el substrato para el estudio de la potencia argumentativa de los lenguajes lógicamente orientados. La sección 4 aborda la traducción de las fórmulas de la lógica de primer orden a la *forma clausal*, un formalismo que facilita su manipulación automática por parte del algoritmo inferencial que se estudia en la sección 5: el *principio de resolución*. La consecuencia lógica más importante de la forma clausal (en tanto que formalismo para la especificación del conocimiento) y el principio de resolución (como algoritmo deductivo) es que aseguran la *completud y corrección lógicas* a nivel de insatisfacibilidad. Es decir, todo conjunto de fórmulas lógicas bien formadas puede ser representado en forma clausal (de forma automática), y además, si en lógica de primer orden se puede demostrar un teorema a partir de un conjunto de axiomas, el principio de resolución asegurará que en un número finito de pasos se podrá demostrar la insatisfacibilidad entre el conjunto de axiomas y la negación del teorema (representados en forma clausal)[2].

Conocimiento Declarativo

Una de las características de la historia de la informática es la tendencia a la explicitación del conocimiento. Tendencia de la que dejan constancia el mismo paradigma de la orientación lógica en la programación, la aparición de disciplinas como la Ingeniería del Conocimiento o el basar la quinta generación de lenguajes de programación en el concepto de sistemas basados en el conocimiento.

En [GENESERETH-88] se propone una ya común distinción entre dos planteamientos generales de representación y utilización del conocimiento desde un punto de vista computacional: el conocimiento procedural o implícito almacenado en la misma secuencia de operaciones que constituyen un programa (Figura 1a) y el conocimiento declarativo o explícito que aparece en los programas como sentencias o *declaraciones explícitas de conocimiento*, con una clara e intuitiva interpretación semántica (Figura 1b).

```
InvierteMatriz (m, d)
int *m, d;
{
        int c1, c2, i;
        for (c1 = 0; c1 <d; c1++);
            for (c2=c1+1; c2<d; c2++);
            {
                i=m [c1*d+c2];
                m[c1*d+c2] *m[c2*d+c1];
                m[c2*d+c1] = i;
            }
}
```

Figura 1a

```
derivada (X,X,1) :- !.
derivada (C,X,0) :- atomic (C).
derivada (U+V, X, DU+DV) :-
            derivada (U,X,DU) ,
            derivada (V,X,DV) .
derivada (U*V,X,DU*V+U*DV) :-
            derivada (U,X,DU) ,
            derivada (V,X,DV) .
```

Figura 1b

[2] Siempre dentro de los límites que la tesis de Church impone a la decidibilidad de la lógica de primer orden.

A este nivel, como en la mayoría, la exclusividad suele ser perniciosa, y no se puede defender sin más ninguna de las alternativas. En general la metodología declarativa aumenta la flexibilidad y generalidad del planteamiento del problema, pero a costa de la eficiencia, que suele ser mayor en un planteamiento procedimental.

No obstante, para determinados ámbitos de problemas (y de forma específica para la Inteligencia Artificial) la orientación declarativa supone una serie de ventajas, como son la reutilizabilidad del conocimiento, la posibilidad de realizar inferencias sobre él, la facilidad de modificación, etc.

Las bases de conocimiento declarativas requieren, para su manipulación, sofisticados sistemas, computacionalmente complejos, que permitan realizar argumentaciones (obtención de conocimiento) a partir de la información disponible. Estos sistemas se denominan *motores de inferencia* y se basan en la implementación de una parte del cálculo deductivo que asegure la corrección y completud lógicas.

Estas consideraciones nos permiten obtener un primer esquema de un entorno típico de programación declarativa, formado por: (a) Un lenguaje formal desde el que se lleva a cabo la especificación de las bases de conocimiento; y (b) Un motor inferencial que puede manipular la base de conocimiento para distintos fines, pero que en última instancia se basa en su potencia deductiva o de razonamiento.

Formalismos de la Lógica de Primer Orden

Nuestro interés por la lógica se centrará en el estudio de los formalismos, en tanto que conjuntos de signos y combinaciones correctas de éstos. Más específicamente nos centraremos en los formalismos de primer orden, caracterizados por la restricción de la cuantificación a los términos.

Un formalismo no es más que un mero juego de signos susceptible de múltiples interpretaciones, pero que en sí mismo no incluye ninguna interpretación; estas mismas son parte de la semántica, y su inclusión convierte al formalismo en un lenguaje formal. En nuestro caso, y por motivos de brevedad, presentaremos un formalismo para la lógica de primer orden junto con la interpretación común. Un tratamiento más exhaustivo del tema se puede encontrar en [CUENA-85] y [MOSTERÍN-83].

Alfabeto de los formalismos

Un formalismo lógico es en realidad un lenguaje, y por tanto su especificación requiere en primer lugar la explicitación del alfabeto. Para los formalismos de la lógica de primer orden se distinguen dos tipos de signos: los comunes a todos los formalismos y los peculiares de cada uno.

Son símbolos comunes las variables y los signos lógicos. Las variables constituyen un conjunto recursivamente numerable de símbolos distintos; para referirnos a ellas utilizaremos las últimas letras del alfabeto en minúsculas: *x, y, z, ...* Respecto al conjunto de signos lógicos es necesario tener en cuenta que no tiene por qué ser fijo, sino que, de hecho, distintos conjuntos pueden asegurar la misma potencia expresiva, aunque no las mismas economía y concisión. En nuestro caso, utilizaremos los cinco conectores ya clásicos (negador: $\neg$, conjuntor: $\wedge$, disyuntor: $\vee$, condicionador: $\rightarrow$, y bicondicionador: $\leftrightarrow$) y los dos cuantificadores (generalizador: $\forall$ y particularizador: $\exists$). La misma potencia expresiva se puede conseguir sólo con el operador NAND (negación del conjuntor) y el generalizador.

Serán específicos de cada formalismo las constantes individuales, los functores y los relatores, pudiendo existir o no, y en caso de existir pudiendo ser su número variable, lo que dependerá del formalismo en sí.

Las constantes individuales se utilizarán para referirse a objetos del mundo (teniendo en cuenta que lo que se entiende por objeto es dependiente de la conceptualización). Las representaremos con las primeras letras del alfabeto en minúsculas, numeradas si es necesario: *a, b, c, ...*

Los functores equivalen formalmente al concepto matemático de función, mediante el que se designa unívocamente un objeto a partir de un conjunto de términos (por ejemplo, *la suma de 3 y 4* designa el número 7). La aridad de un functor indica el número de términos que requiere para completar su designación. Usualmente utilizaremos las letras *f, g, h, ...* como functores, con superíndices indicando su aridad si resulta necesario par eliminar ambigüedades.

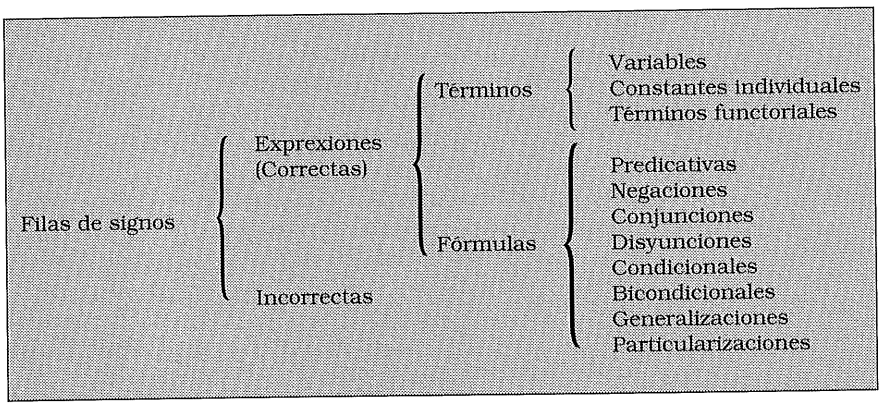
Los relatores o predicados combinan términos designando relaciones entre ellos que podrán ser verdaderas o falsas (por ejemplo, *4 es mayor que 3* es una relación verdadera, es decir, se satisface por la interpretación matemática usual de todos los conceptos implicados). De forma similar a los functores se define la aridad de cada relator; para representarlos utilizare-

mos las letras *P, Q, R, ...*, provistas si es necesario de subíndices de diferenciación y superíndices designando su aridad.

Gramática de los formalismos

No todas las combinaciones posibles de los signos de un formalismo serán correctas. En concreto, se considerarán expresiones (filas de signos correctas) las que se pueden formar de acuerdo con las siguientes reglas:

1.- Cualquier variable es un término (variable).
2.- Cualquier constante individual es un término (constante individual).
3.- Si $t_1,...,t_n$ son términos y f^n un functor n-ario, entonces $f^n t_1,...,t_n$ será un término (término functorial).
4.- Si $t_1,...,t_n$ son términos y P^n es un relator, entonces $P^n t_1,...,t_n$ será una fórmula (fórmula atómica o predicativa).
5.- Si α es una fórmula, entonces $-\alpha$ es una fórmula (negación).
6.- Si α y β son fórmulas, entonces $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \rightarrow \beta$, $\alpha \leftrightarrow \beta$ son también fórmulas (respectivamente, conjunción, disyunción, condicional y bicondicional).
7.- Si α es una fórmula, entonces para toda variable x, $\forall x\ \alpha$ y $\exists x\ \alpha$ son fórmulas (generalización y particularización, respectivamente).



Cálculo deductivo

Dentro de un formalismo se pueden introducir determinadas relaciones entre sus fórmulas. No obstante, a la lógica sólo le interesan algunas de estas relaciones, en concreto, las que aseguran la preservación de la verdad, es decir, las relaciones de deducibilidad.

El conjunto de estas reglas de inferencia o deducción forman un cálculo deductivo (junto con las reglas que rigen la construcción de las deducciones). Obviamente existen muchos cálculos deductivos que aseguran la misma capacidad inferencial. Básicamente se distinguen dos modelos: los basados en reglas de inferencia o sistemas de deducción natural, introducidos por Gerhardt Gentzen en los años 30 y los sistemas axiomáticos, en los que a partir de un número muy reducido de reglas de inferencia (incluso en ocasiones una sola, como puede ser *modus ponens* o eliminación del disyuntor) aseguran la misma potencia deductiva mediante la inclusión de esquemas axiomáticos que de alguna forma traducen el resto de las reglas de inferencia.

Lo más importante para un cálculo deductivo es que asegure, dentro de un formalismo (y una teoría es un formalismo), dos propiedades fundamentales: (a) *Corrección*: toda fórmula (teorema) deducida a partir de axiomas verdaderos debe ser verdadera; y (b) *Completud*: toda fórmula o teorema verdadero dentro del formalismo debe poder obtenerse mediante el cálculo deductivo, es decir, debe ser posible obtener una deducción del teorema utilizando el aparato argumentativo o inferencial del cálculo.

En esta sección se ha presentado sólo la parte de la lógica de primer orden que interesa a nuestros objetivos; en las siguientes secciones se abordarán los problemas que surgen durante la implementación computacional de sistemas inferenciales. El primer y quizás principal punto a tener en cuenta es la complejidad que supondría la simulación de un cálculo deductivo con múltiples reglas; asimismo la sintaxis presentada no puede ser manipulada fácilmente por un ordenador. Por lo tanto, los objetivos primordiales serán la obtención de un formalismo para la representación de conocimiento y un cálculo deductivo lo más simplificados posibles pero que aseguren la potencia expresiva de los formalismos, por un lado, y la corrección y completud del sistema, por otro.

Una posibilidad lo constituyen la forma clausal, en tanto que formalismo para la representación, y el principio de resolución, que permite realizar inferencias sobre cláusulas basándose en la determinación de su contradicción o insatisfacibilidad.

Forma clausal

Forma clausal de las sentencias de la lógica de primer orden

La representación clausal de una fórmula está constituida por una o más cláusulas, cada una de las cuales es un elemento de la *forma normal conjuntiva*, es decir, la forma clausal es una conjunción de cláusulas, cada una de las cuales es una disyunción de fórmulas atómicas (predicativas) o negaciones de fórmulas atómicas.

Todas las variables que aparezcan se sobreentenderá que están dentro del alcance de un cuantificador universal, de forma que la eliminación de cuantificadores existenciales requerirá la utilización de constantes y/o funciones de Skolem.

A continuación se presentará un algoritmo, que puede ser implementado computacionalmente, y que permite la traducción de cualquier fórmula correcta de la lógica de primer orden a su forma clausal.

Se utilizarán dos ejemplos para ilustrar el proceso: la fórmula α conceptualiza la univocidad de la relación de maternidad (a es madre de b se escribirá como Mab); la fórmula β es una definición de la propiedad simétrica para relaciones (el par ordenado formado por los elementos a y b se construirá mediante el functor $< , >$: $<a,b>$).

$$\alpha \equiv \forall xyz\ (Mxz \wedge Myz \rightarrow x = y)$$
$$\beta \equiv \forall r\ (Sim\ r \leftrightarrow \forall xy\ (<x,y> \in r \rightarrow <y,x> \in r))$$

a) ***Eliminación de condicionales y bicondicionales***: Se eliminarán estos conectores lógicos teniendo en cuenta que, para cualesquiera dos fórmulas Φ y Ψ:

$\Phi \rightarrow \Psi$ se puede reescribir como $\neg\Phi \vee \Psi$
$\Phi \leftrightarrow \Psi$ se puede reescribir como $(\neg\Phi \vee \Psi) \wedge (\Phi \vee \neg\Psi)$

$$\alpha \equiv \forall xyz\ (\neg(Mxz \wedge Myz) \vee x = y)$$
$$\beta \equiv \forall r\ (\neg Sim\ r \vee \forall xy\ (<x,y> \notin r \vee <y,x> \in r)) \wedge (Sim\ r \vee \neg\forall xy\ (<x,y> \notin r \vee <y,x> \in r)))$$

(b) ***Eliminación del negador sobre fórmulas no atómicas***, basándose en las siguientes reglas: para cualesquiera fórmulas Φ y Ψ y cualquier variable x:

$\neg(\Phi \wedge \Psi)$	se puede reescribir como	$\neg\Phi \vee \neg\Psi$
$\neg(\Phi \vee \Psi)$	se puede reescribir como	$\neg\Phi \wedge \neg\Psi$
$\neg\forall x \Phi)$	se puede reescribir como	$\exists x \neg\Phi$
$\neg\exists x \Phi$	se puede reescribir como	$\forall x \neg\Phi$
$\neg\Phi$	se puede reescribir como	Φ

$$\alpha \equiv \forall xyz\ (\neg Mxz \vee \neg Myz \vee x = y)$$
$$\beta \equiv \forall r\ (\neg Sim\ r \vee \forall xy\ (<x,y> \notin r \vee <y,x> \in r)) \wedge (Sim\ r \vee \exists xy\ (<x,y> \in r \wedge <y,x> \notin r)))$$

(c) ***Estandarización de variables***, de forma que una misma variable no aparezca en varios cuantificadores dentro de la misma fórmula.

$$\alpha \equiv \forall xyz\ (\neg Mxz \vee \neg Myz) \vee x = y)$$
$$\beta \equiv \forall r\ (\neg Sim\ r \vee \forall xy\ (<x,y> \notin r \vee <y,x> \in r)) \wedge (Sim\ r \vee \exists uv\ (<u,v> \in r \wedge <v,u> \notin r)))$$

d) ***Eliminación de los cuantificadores existenciales***. Desde un punto de vista lógico, la presencia de un cuantificador existencial ($\exists x\ \Phi$) asegura la existencia de al menos un objeto que cumple la condición expresada por Φ. Para eliminar estas cuantificaciones se utilizará el procedimiento de *skolemización*, distinguiéndose dos casos:

En primer lugar, si el cuantificador existencial no se encuentra dentro del alcance de ningún generalizador se puede sustituir la variable por una constante (constante de Skolem) y prescindir del particularizador, lo que nos permite pasar de $\exists x(Px \wedge Qx)$ a $(Pa \wedge Qa)$, siempre que a no haya sido utilizada previamente.

En segundo lugar, si el particularizador está dentro del alcance de uno o más generalizadores, la variable cuantificada existencialmente se sustituirá por un término functorial, donde el functor será uno nuevo (función de Skolem) previamente no utilizado, cuya aridad coincidirá con el número de generalizadores que le afectan y los términos sobre los que se aplica el functor serán las mismas variables universalmente cuantificadas. Por ejemplo, si R es el predicado *ser un número real*, tendrá sentido reescribir

$$\forall xy\ (Rx \wedge Ry \wedge x<y \rightarrow \exists z\ (Rz \wedge z>x \wedge z<y))$$

como

$$\forall xy\ (Rx \wedge Ry \wedge x<y \rightarrow (Rfxy \wedge fxy>x \wedge fxy<y))$$

Es decir, podemos construir para cada par de números reales x e y un nuevo número fxy que es menor que x y mayor que y. El nuevo functor de Skolem f designa a partir de cada par de números ese otro intermedio.

$$\alpha \equiv \forall xyz\ (\neg Mxz \vee \neg Myz) \vee x = y)$$
$$\beta \equiv \forall r\ (\ (\neg Sim\ r \vee \forall xy\ (<x,y> \notin r \vee <y,x> \in r)) \wedge (Sim\ r \vee (<f_1 r, f_2 r> \in r \wedge <f_2 r, f_1 r> \notin r)))$$

donde f_1 y f_2 son dos nuevos functores introducidos para eliminar respectivamente las variables u y v, cuantificadas existencialmente.

(e) ***Eliminación de los cuantificadores universales***: simplemente se eliminarán todos los cuantificadores universales dando por sobreentendido que todas las variables que aparecen en forma clausal están cuantificadas universalmente.

$$\alpha \equiv \neg Mxz \vee \neg Myz \vee x = y$$
$$\beta \equiv (\neg Sim\ r \vee (<x,y> \notin r \vee <y,x> \in r) \wedge (Sim\ r \vee <f_1 r, f_2 r> \in r \wedge <f_2 r, f_1 r> \notin r)$$

(f) ***Forma normal conjuntiva***: se trata de obtener la fórmula como conjunción de disyunciones, para lo que se aplica la siguiente regla

$\Phi \vee (\Psi \wedge \Omega)$ se reescribe como $(\Phi \vee \Psi) \wedge (\Phi \vee \Omega)$

$$\alpha \equiv \neg Mxz \vee \neg Myz \vee x = y$$
$$\beta \equiv (\neg Sim\ r \vee (<x,y> \notin r \vee <y,x> \in r) \wedge (Sim\ r \vee <f_1 r, f_2 r> \in r) \wedge (Sim\ r \vee <f_2 r, f_1 r> \notin r)$$

(g) ***Representación en forma clausal***. Cada una de las conjunciones obtenida será una cláusula construida como el conjunto (desordenado) de sus disyuntores. Así por ejemplo, a partir de $(\Phi \vee \Psi) \wedge (\Phi \vee \Omega)$ se obtendrían las dos cláusulas $\{\Phi,\Psi\}$ y $\{\Phi,\Omega\}$.

$$\alpha \equiv \{\neg Mxz \vee \neg Myz, x = y\}$$
$$\beta \equiv \{\neg Sim\ r, <x,y> \notin r, <y,x> \in r\}$$
$$\{Sim\ r, <f_1 r, f_2 r> \in r\}$$
$$\{Sim\ r, <f_2 r, f_1 r> \notin r\}$$

Los elementos o disyuntores de una cláusula serán siempre fórmulas atómicas o negaciones de éstas. Para referirse a ambos tipos de fórmulas se utiliza el término *literales*.

Algunos autores añaden un paso adicional mediante el que se asegura que cada variable aparezca en una sóla cláusula, es decir, se evita la aparición de una misma variable en más de una cláusula. El añadir este paso simplifica la definición del algoritmo de resolución.

Interpretación de las cláusulas

Cada una de las cláusulas se define como una disyunción de literales. La principal consecuencia lógica es que una cláusula será verdadera si al menos alguno de los literales lo es, o de forma equivalente, no puede ocurrir que todos los literales de una cláusula verdadera sean falsos.

Esto nos permite obtener una interpretación bastante intuitiva del comportamiento lógico de una cláusula: la negación de todos los literales menos uno de una cláusula verdadera asegura la verdad del literal no negado.

Por ejemplo, si partimos de la cláusula $\{\neg Mxz, \neg Myz, x=y\}$ y si negamos los dos primeros literales, es decir, afirmamos Mxz y Myz, entonces forzosamente deberá cumplirse $x=y$. De forma similar, si negamos el primer y tercer literales: Mxz y $x \neq y$, podemos concluir que $\neg Myz$.

El principio de resolución

Este principio fue introducido inicialmente por Robinson en 1965 [ROBINSON-65, ROBINSON-68, KOWALSKY-79]. Está caracterizado básicamente como una regla universal de resolución con unificación, y pretende dar cuenta de la derivación automática de teoremas de la lógica de primer orden, dentro de las limitaciones que impone la indecidibilidad en este ámbito.

Unificación

Una unificación es un conjunto de asociaciones entre variables y términos que permite la identificación entre dos expresiones. Así por ejemplo las dos expre-

siones *Mxz* y *Mab* son unificables (identificables) si sobre la primera se aplica la unificación $\{x/a, z/b\}$. El resultado de aplicar una sustitución λ sobre una expresión Φ se denomina *factor* de Φ y se escribe como $\Phi\lambda$.

La decisión acerca de si dos expresiones son unificables y en caso de que sí lo sean, la obtención de unificadores, son procesos relativamente sencillos, para los que se han definido algoritmos implementables en ordenador. En concreto se han diseñado procedimientos para la obtención automática del unificador más general de dos expresiones. Por cuestiones de espacio no se presenta ninguno de estos algoritmos y se remite al lector interesado a [CUENA-85] y [GENESERETH-88] donde se presentan en detalle.

El principio de resolución

Este principio se basa en la regla lógica de eliminación del disyuntor:

$$\frac{\Phi \vee \Psi \qquad \{\Phi,\Psi\}}{\neg\Phi \qquad \{\neg\Phi\}} \qquad \frac{}{\Psi \qquad \{\Psi\}}$$

Formalmente, el principio de resolución afirma que si tenemos dos cláusulas Φ y Ψ, y existe un literal α en un factor Φ' de Φ y un literal $\neg\beta$ en algún factor Ψ' de Ψ, y entre α y β se puede obtener un unificador (y por tanto el unificador más general λ) entonces podremos obtener una nueva cláusula de la siguiente forma:

$$\frac{\Phi \quad (\text{donde } \alpha \in \Phi') \qquad \Psi \quad (\text{donde } \beta \in \Psi' \text{ y } \alpha\lambda = \neg\beta\lambda)}{((\Phi'-\{\alpha\}) \cup (\Psi'-\{\beta\}))\lambda}$$

El algoritmo de resolución necesita hacer uso de un mecanismo de *simplificación* que se encargue de eliminar de una cláusula dos literales idénticos, o identificables mediante algún unificador.

Razonamiento mediante el principio de resolución

¿Qué sucedería si mediante el principio de resolución obtuviéramos una cláusula vacía? Si volvemos a la interpretación lógica de las cláusulas, sólo podemos obtener una cláusula vacía si podemos negar todos los elementos de la cláusula, lo cual está en contradicción con la definición dada para las cláusulas.

La consecuencia lógica de este planteamiento es que si un conjunto de cláusulas puede generar mediante resolución una cláusula vacía entonces el conjunto original de cláusulas era autocontradictorio. La aplicación de este planteamiento para el razonamiento es inmediato: si queremos demostrar un teorema a partir de un conjunto de axiomas, representaremos en forma clausal los axiomas y la negación del teorema e intentaremos obtener la contradicción en la forma de una cláusula vacía, proceso que es denominado *refutación por resolución.*

Este enfoque permite asegurar la completud y corrección lógicas del principio de resolución en función de la insatisfacibilidad:

(a) *Corrección*: si existe una deducción por resolución de una cláusula a partir de un conjunto de cláusulas, entonces este conjunto de cláusulas implica lógicamente la cláusula inicial.

(b) *Completud*: si un conjunto de cláusulas es insatisfacible, entonces existe una deducción por resolución de la cláusula vacía a partir del conjunto inicial de cláusulas.

Un ejemplo para la teoría de conjuntos

A continuación intentaremos demostrar el teorema

$$\beta \equiv \forall r \ (Sim\ r \wedge Tras\ r \rightarrow Refl\ r)$$

según el cual toda relación simétrica y transitiva es reflexiva.

Para ello partiremos de los siguientes axiomas

$$\alpha_1 \equiv \forall r\ (Refl\ r \leftrightarrow \forall x\,(x \in dr \rightarrow \rightarrow \langle x,x\rangle \in r))$$
$$\alpha_2 \equiv \forall r\ (Sim\ r \leftrightarrow \forall xy\,(\langle x,y\rangle \in \in d \rightarrow \langle y,x\rangle \in r))$$
$$\alpha_3 \equiv \forall r\ (Tras\ r \leftrightarrow \forall xyz\,(\langle x,y\rangle \in \in r \wedge \langle y,z\rangle \in r \rightarrow \langle x,z\rangle \in r))$$

En α_1, *dr* representa el conjunto de todos los elementos que pertenecen al dominio o codominio de la relación *r*. Para que podamos manipular este axioma será necesario introducir uno nuevo:

$$\alpha_4 \equiv \forall r\, x(\, x \in dr \leftrightarrow \exists y\, (\langle x,y\rangle \in r \vee \langle y,x\rangle \in r))$$

En primer lugar llevaremos a cabo la transcripción clausal de los axiomas y la negación del teorema a demostrar, numerando cada una de las cláusulas obtenidas e indicando para cada una de ellas su procedencia. Los functores f_n corresponden a las diferentes funciones de Skolem introducidas, y *a* es una constante de Skolem introducida en la transcripción de la negación del teorema a demostrar.

1	$\{\neg Refl\ r,\ x \notin dr,\ \langle x,x\rangle \in r\}$	(α_1)
2	$\{Refl\ r,\ f_1 r \in dr\}$	(α_1)
3	$\{Refl\ r,\ \langle f_1 r, f_1 r\rangle \notin r\}$	(α_1)
4	$\{\neg Sim\ r,\ \langle x,y\rangle \notin r,\ \langle y,x\rangle \in r\}$	(α_2)
5	$\{Sim\ r,\ \langle f_2 r, f_3 r\rangle \in r\}$	(α_2)
6	$\{Sim\ r,\ \langle f_3 r, f_2 r\rangle \notin r\}$	(α_2)
7	$\{\neg Trans\ r,\ \langle x,y\rangle \notin r,\ \langle y,z\rangle \notin r,\ \langle x,z\rangle \in r\}$	(α_3)
8	$\{Trans\ r,\ \langle f_4 r, f_5 r\rangle \in r\}$	(α_3)
9	$\{Trans\ r,\ \langle f_5 r, f_6 r\rangle \in r\}$	(α_3)
10	$\{Trans\ r,\ \langle f_4 r, f_6 r\rangle \notin r\}$	(α_3)
11	$\{x \notin dr,\ \langle x, f_7 x\rangle \in r,\ \langle f_7 x, x\rangle \in r\}$	(α_4)
12	$\{x \in dr,\ \langle x,u\rangle \notin r\}$	(α_4)
13	$\{x \in dr,\ \langle u,x\rangle \notin r\}$	(α_4)
14	$\{Sim\ a\}$	$(\neg\beta)$
15	$\{Trans\ a\}$	$(\neg\beta)$
16	$\{\neg Refl\ a\}$	$(\neg\beta)$

Y a continuación aparece una demostración por refutación:

17	$\{\langle x,y\rangle \notin a,\ \langle y,x\rangle \in a\}$	(4,14) {r/a}
18	$\{\langle x,y\rangle \notin a,\ \langle y,z\rangle \notin a,\ \langle x,z\rangle \in a\}$	(7,15) {r/a}
19	$\{f_1 a \in da\}$	(2,16) {r/a}
20	$\{\langle f_1 a, f_1 a\rangle \notin a\}$	(3,16) {r/a}
21	$\{\langle f_1 a, f_1 f_1 a\rangle \in a,\ \langle f_2 f_1 a, f_1 a\rangle \in a\}$	(11,19) {x/f_1a,r/a}
22	$\{\langle f_1 a, f_1 f_1 a\rangle \in a\}$	(17,21) {x/$f_2 f_1$a,y/f_1a}
23	$\{\langle f_1 f_1 a, f_1 a\rangle \in a\}$	(12,22) {x/f_1a y/$f_2 f_1$a}*
24	$\{\langle f_1 f_1 a, z\rangle \notin a,\ \langle f_1 a, z\rangle \in a\}$	(18,22) {x/f_1a y/$f_2 f_1$a}
25	$\{\langle f_1 a, f_1 a\rangle \in a\}$	(23,24) {z/f_1a}
26	$\{\ \}$	(20,25)

* Se aplica simplificación, pues el resultado de la unificación son dos literales idénticos.

Aplicación para la búsqueda de soluciones

La capacidad de demostración de teoremas puede ser utilizada para encontrar soluciones que cumplen una serie de condiciones.

Supongamos, por ejemplo, que los relatores *M* y *A* representan, respectivamente, las relaciones *ser madre de* y *ser abuela de*, y que dispone de la siguiente información (axiomas):

$$\alpha_1 \equiv \forall xyz\ (Mxy \wedge Myz) \rightarrow Axz)$$
$$\alpha_2 \equiv Mab$$
$$\alpha_3 \equiv Mbc$$

y nos interesa conocer *los nietos* de *a*, es decir, nuestra pregunta será *Aau*, donde *u* es la variable que referencia la solución.

En primer lugar obtendremos la representación clausal de los axiomas, y añadiremos como teorema a demostrar el presentado en la línea 4. A continuación aparece el desarrollo del algoritmo de resolución, que concluye presentando *c* como solución a la interrogación planteada:

1	$\{\neg Mxy, \neg Myz, Axz\}$	(α_1)
2	$\{Mab\}$	(α_2)
3	$\{Mbc\}$	(α_3)
4	$\{\neg Aau, Su\}$	(teorema a demostrar)
5	$\{\neg Mbz, Aaz\}$	(1,2)
6	$\{Aac\}$	(2,5)
7	$\{Sc\}$	(4,6)

En este caso, el procedimiento de demostración no termina con la cláusula vacía sino con una cláusula que contenga un único literal formado por el relator *S*.

Estrategias de resolución

Desde el punto de vista computacional el principal problema que aparece es la definición de una estrategia que dirija el proceso de resolución.

En definitiva el principio de resolución nos permite recorrer todo el espacio del problema (el grafo generado) y lo que nos interesará será disponer de algún control que nos asegure la mayor rapidez en la obtención de la demostración.

De esta forma convertimos la demostración en un claro problema de búsqueda (véase [BARR-81]) para el

que por tanto son aplicables los numerosos sistemas o algoritmos diseñados, destacando entre otras las estrategias de mayor profundidad, de menor distancia, la resolución simple, la resolución lineal, la resolución dirigida (basada en la utilización de heurísticas) o la satisfacción de restricciones.

Bibliografía

[BARR-81] BARR, A. y FEIGENBAUM, E.A. (eds) (1981): **The Handbook of Artificial Intelligence**, Volumen 1. Addison-Wesley Publishing Company, Inc.

[CUENA-85] CUENA, J. (1985): **Lógica Informática**. Madrid, Alianza Informática.

[DERSHEM-90] DERSHEM, H.L. y JIPPING, M.J. (1990): **Programming Languajes: Structures and Models**. Belmont, California, Wadsworth Publishing Company.

[GENESERETH-88] GENESERETH, M.R. y NILSSON, N.J. (1988): **Logical Foundations of Artificial Intelligence**. Morgan Kaufmann Publishers Inc.

[KOWALSKY-79] KOWALSKI, R. (1979): **Logic for Problem Solving**. Elsevier Science Publishing Co. [Existe traducción al español: **Lógica, Programación e Inteligencia Artificial**. Díaz de Santos, 1986]

[MOSTERÍN-83] MOSTERÍN, J. (1983): **Lógica de primer orden**. Barcelona, Ariel.

[ROBINSON-65] ROBINSON, J.A. (1965): **A Machine Oriented Logic Based on Resolution Principle**. J. ACM 12 (Enero 1965), pp. 23-41

[ROBINSON-68] ROBINSON, J.A. (1968): **The Generalised Resolution Principle**. En DALE y MICHIE (edts): **Machine Intelligence 3**, Oliver and Boyd, Edinburg, 1968.

José F. Quesada Moreno
C.I.C.A. Sevilla