REPRESENTACION DE HECHOS EN LOGICA DE PREDICADOS

Pero en seguida nos tropezamos con las limitaciones de la lógica proposicional, y se pasó a utilizar la de predicados como medio de representar el conocimiento y que encima nos proporciona como efecto lateral, una manera de razonar con ese conocimiento.

Las fórmulas bien construidas del cálculo de predicados de primer orden proporcionan una manera de describir objetos y relaciones en el dominio del problema, mientras que las reglas de inferencia permiten extraer nuevo conocimiento de estas descripciones.

A primera vista esta solución es perfecta, pero el Cálculo de Predicados no tiene un proceso de decisión, es decir, no hay ningún procedimiento que encuentre una demostración de una fórmula si dicha fórmula no es un teorema. Además, no hay una aplicación única entre una frase del lenguaje natural y una fórmula del cálculo de predicados.

Consideremos las siguientes sentencias:

1.- Marco era un hombre.
2.- Marco era de Pompeya.
3.- Todos los habitantes de Pompeya eran romanos.
4.- Cesar era gobernante.
5.- Todos los romanos o eran leales al Cesar o le odiaban.
6.- Todo el mundo es leal a alguien.
7.- La gente intenta asesinar a los gobernantes a los que nos les son leales.
8.- Marco intentó asesinar a Cesar.

Los hechos descritos por estas sentencias se pueden representar en la lógica de predicados de la siguiente manera:

1.- hombre (marco)
2.- pompeya (marco)
3.- $\forall x$ [pompeya(x) $\rightarrow$ romano(x)]
4.- gobernante (cesar)
5.- $\forall x$ [romano (x) $\rightarrow$ (leal(x, cesar) $\vee$ odiar(x, cesar))]
6.- $\forall x \forall y$ leal(x, y)
7.- $\forall x \forall y$ [persona(x) $\wedge$ gobernante(y) $\wedge$ asesina(x, y) $\rightarrow \sim$ leal (x, y)]
8.- asesina (marco, cesar)

Supongamos que tenemos todo este conocimiento representado en la máquina en LISP, PROLOG o cualquier otro lenguaje de ordenador y queremos que el sistema nos responda a la pregunta: ¿Era Marco leal al Cesar? que en forma lógica sería: **leal(marco, cesar)**. Nos bastaría demostrar que: $\sim$ **leal (marco, cesar)**.

leal (marco, cesar)

(7)
persona (marco) ∧ gobernante (cesar) ∧ asesina (marco, cesar)

(4)
persona (marco) ∧ asesina (marco, cesar)

(8)
persona (marco)

El problema que se nos presenta es que aunque sabemos que Marco era un hombre, no tenemos ninguna manera de concluir que Marco era una persona. Tendríamos que añadir la representación de otro hecho a nuestro sistema:

9.- Todos los hombres son personas
$\forall x$ [hombre (x) → persona(x)]

Ahora si podemos satisfacer la meta obtenida en el último paso y producir una demostración de que Marco no le era leal a Cesar, y por lo tanto, responder a la pregunta que habíamos hecho al sistema.

(9)
hombre (marco)

□

Veamos otro ejemplo:

1.- Marco era un hombre
hombre (marco)

2.- Marco era de Pompeya
pompeya (marco)

3.- Marco nació en el 40 a.D.
nacio (marco, 40)

4.- Todos los hombres son mortales
$\forall x$ [hombre(x) → mortal(x)]

5.- Todos los pompeyanos murieron cuando el volcán entró en erupción en el 79 a.D..
erupcion(volcan, 79) ∧ $\forall x$ [pompeyano(x) → muerto(x, 79)]

6.- Ningún mortal vive más de 150 años
$\forall x \forall t1 \forall t2$ [mortal (x) $\wedge$ nacer(x, t1) $\wedge$ mayor(t2-t1, 150) $\rightarrow$ muerto(x, t2)]

7.- Ahora estamos en 1995
ahora = 1995

Queremos la respuesta a la pregunta: ¿Está Marco vivo?

De la anterior base de conocimiento se desprende:

a.- Podemos demostrar que Marco está muerto porque murió en la erupción del volcán en el 79.

b.- Podemos demostrar que Marco está muerto porque en caso contrario tendría más de 150 años y eso es contrario a las premisas.

Pero cuando empezamos a seguir cualquiera de los dos caminos, nos encontramos con que nos hace falta información adicional. Sin ir más lejos la forma de redactar la pregunta: ¿Está Marco vivo? En nuestra base de conocimiento no tenemos nada que nos diga que estar vivo es lo opuesto a estar muerto.

8.- Vivo significa no muerto
$\forall x \forall t$ [~vivo (x, t) $\leftrightarrow$ muerto(x, t)]

9.- Si alguien muere entonces sigue muerto en todo el tiempo futuro
$\forall x \forall t1 \forall t2$ [muerto (x, t1) $\wedge$ mayor (t1, t2) $\rightarrow$ muerto (x, t2)]

Intentamos ahora responder a la repuesta ¿Está Marco vivo? y para ello vamos a demostrar ~ **vivo (marco, ahora)**

DEMOSTRACION DE QUE MARCO ESTA MUERTO

~ vivo (marco, ahora)

(8)
muerto (marco, ahora)

(9)
muerto (marco, t1) ∧ mayor (ahora, t1)

(5)
pompeyano(marco) ∧ mayor (ahora, 79)

(5)
pompeyano (marco) ∧ mayor (ahora, 79)

(2)
mayor (ahora, 79)

(7)
mayor(1995, 79)

Para obtener las anteriores demostraciones nos hemos encontrado cón:

a.- Incluso las conclusiones más simples requieren muchos pasos para su demostración.

b.- Para obtener la demostración hemos empleado distintos procesos tales como:

- Emparejamiento
- Sustitución
- Encadenamiento hacia atrás → más adelante
- Conocimiento del Universo → espacio

Una conclusión que se desprende de todo lo anterior es que si queremos hacer razonamientos no triviales, tenemos que encontrar métodos de demostración que pueden tomar pasos más grandes. Para ellos usaremos el: Principio de Resolución de Robinson. Antes hay que ver el tema de Unificación.

UNIFICACION

Para poder aplicar una regla de inferencia, por ejemplo, Modus Ponens o Resolución, un sistema lógico debe ser capaz de determinar cuando dos expresiones son iguales o están "emparejadas". En el cálculo de proposiciones, esto es trivial: dos expresiones se emparejan si y sólo si, son sintácticamente idénticas. En el cálculo de predicados, el proceso de emparejar dos sentencias se complica por la existencia de variables en las expresiones. La instanciación universal permite que las variables cuantificadas universalmente se sustituyan por términos del dominio. Esto requiere un proceso de decisión para determinar las sustituciones de variables bajo las cuales dos o más expresiones se pueden considerar idénticas.

UNIFICACION es un algoritmo para determinar las sustituciones que se necesitan para hacer que dos expresiones del cálculo de predicados se emparejen. Por ejemplo,

$\forall x$ [hombre(x) $\rightarrow$ mortal(x)]
hombre(socrates)

podemos sustituir **socrates** por la **x** para poder entonces aplicar Modus Ponens y conseguir **mortal(socrates)**.

Unificación y una regla de inferencias cualquiera nos permite hacer inferencias sobre una base de conocimientos aseverativa. Para hacer esto la base de conocimientos debe expresarse de una forma apropiada. Un aspecto esencial de esta forma es el requerimiento de que todas las variables estén cuantificadas universalmente. Esto nos permite calcular libremente las sustituciones.

Las variables cuantificadas existencialmente se pueden eliminar de la base de conocimientos, sustituyéndolas por medio de constantes que las haga que sean verdadera.

Por ejemplo, $\exists$**x padre(x, tomas)**, se podría sustituir por la expresión: **padre(roberto, tomas)** o **padre(maria, tomas)**, suponiendo que Roberto y María son los padres de Tomás bajo esta interpretación.

El proceso de eliminar las variables cuantificadas existencialmente se complica por el hecho de que el valor de estas sustituciones puede que dependa del valor de otras variables de la expresión. Por ejemplo, supongamos la expresión: $\forall$**x**$\exists$**y madre(y, x)**, el valor de la variable **y**, cuantificada existencialmente, depende del valor de **x**. El proceso llamado **Skolemización** sustituye cada variable cuantificada existencialmente por una función que devuelve la constante apropiada bajo forma de una función de otras variables de la sentencia. En el ejemplo anterior, puesto que

el valor de **y** depende de la **x**, se podría sustituir **y** por una función de **Skolem**, **f(x)**. Esto nos da el predicado ∀**x madre(x, f(x))**.

Una vez retiradas las variables cuantificadas existencialmente de la base de conocimientos, se puede usar **Unificación** para emparejar las sentencias y permitir el uso de reglas de inferencia.

Unificación se complica debido al hecho de que una variable puede ser sustituida por cualquier término, incluyendo otras variables y funciones de complejidad arbitraria. Por ejemplo, **padre(pepe)** se puede sustituir por la **X** en **hombre(X)** para inferir que el padre de Pepe, quienquiera que sea, es mortal.

Ejemplos de sustituciones legales de la expresión:

foo(x, a, goo(y))

serían:

1.- foo(federico, a, goo(z))
2.- foo(w, a, goo(pepe))
3.- foo(z, a, goo(moo(z)))

En el ejemplo, las sustituciones o unificaciones que hacen que la expresión inicial sea idéntica a cada una de las otras tres sería:

1.- {federico/x, z/y}
2.- {w/x, pepe/y}
3.- {z/x, moo(z)/y}

Las sustituciones también se llaman ligaduras. Una variable se dice que está ligada al valor que le sustituye.

Al definir el algoritmo de unificación que calcula las sustituciones requeridas para que dos expresiones se emparejen, hay que tener varias cosas en consideración:

1.- Aunque una constante puede sistemáticamente sustituir una variable, cualquier constante se considera una instancia base (ground instance) y no puede ser sustituida. Ni tampoco dos instancias base diferentes pueden ser sustituidas por una variable.

2.- Una variable no puede ser unificada por un término que contiene esa variable. Por ejemplo, **x** no puede ser sustituida por **P(x)** puesto que esto crearía la expresión infinita P(P(P...(...x)...).

Generalmente, un proceso de resolución de problemas requerirá múltiples inferencias y, por consiguiente múltiples unificaciones sucesivas. Los mecanismos generales de solución lógicos deben mantener la consistencia en las sustituciones de variables. Es importante que cualquier unificación que se haga, se haga en todas las ocurrencias de la variable de ambas expresiones que se están empareja

Una vez que está ligada una variable, unificaciones futuras e inferencias deben tomar este nuevo valor en consideración. Si una variable está ligada a una constante, no se le puede dar un nuevo valor en una futura unificación. Si una variable X_1 es sustituida por otra variable X_2 y más tarde X_1 es sustituida por una constante, entonces X_2 también debe reflejar esta ligadura. El conjunto completo de ligaduras usadas en una secuencia de inferencias es importante porque puede contener la respuesta a un query.

Por ejemplo, Si **P(a, x)** se unifica con la premisa de **P(y, z)** → **Q(y, z)** utilizando la sustitución **{a/y, x/z}**. Modus Pones entonces nos permite deducir **Q(a, x)**. Si ahora emparejamos esto con la expresión **Q(w, b)** → **R(w)** podríamos inferir **R(w)** si hacemos la sustitución **{a/w, b/x}**.

Esto requiere que los resultados de unificaciones sucesivas se **"compongan"** para formar un conjunto único y consistente de sustituciones.

El conjunto **{a/w, b/x}** del anterior ejemplo se derivó de componer **{a/y, x/z}** de la primera unificación con **{y/w, b/z}** de la segunda. Si **S** y **S'** son los dos conjuntos de sustituciones, entonces el compuesto de **S** y **S'** (se escribe **SS**) se obtiene aplicando **S** a los elementos de **S'** y añadiendo los resultados a S. Veamos el siguiente ejemplo de componer la secuencia de sustituciones.

{x/y, w/z}, {v/x}, {a/v, f(b)/w}

Esto es equivalente a la única sustitución:

{a/y, f(b)/z}

Esto se obtuvo al componer {x/y, w/z} con {v/x} para darnos {v/y, w/z} y componiendo esto con {a/v, f(b)/w} para producir {a/y, f(b)/z}.

Se puede demostrar que la composición es asociativa pero no conmutativa. Un requerimiento final del algoritmo de unificación es que el unificador sea tan general como sea posible, es decir, que se encuentre el "unificador más general" de las dos expresiones. Esto es importante como se verá a continuación, debido a que si se pierde la generalidad en el proceso de resolución, disminuye el ambiente de la solución (scope) o incluso la posibilidad de resolver el problema.

Por ejemplo, al unificar P(x) y P(y), cualquier constante nos valdría, {pepe/x, pepe/y}. Sin embargo, pepe no es el unificador más general, cualquier variable funcionaría y produciría una expresión más general {z/x, z/y}. Las soluciones obtenidas de la primera ronda de sustituciones restringen las inferencias resultantes, reduciendo la generalidad del resultado. Es decir, **pepe** sería un unificador pero, reduciría la generalidad del resultado

UNIFICADOR MAS GENERAL

Si **S** es cualquier unificador de las expresiones de **E** y **g** es el unificador más general de ese conjunto de expresiones, entonces para **S** aplicado a **E**, existe otro unificador **S'** tal que $E_S = E_{gS'}$ (donde **gs'** es la composición de unificadores).

El unificador más general de un conjunto de expresiones es único excepto en variantes alfabéticas (es decir, si una variable eventualmente se llama **x** o **y** realmente no importa a la generalidad de las unificaciones resultantes.

RAZONAMIENTO AUTOMATICO - RESOLUCION

Resolución es una técnica de demostración de teoremas del Cálculo de Predicados y procede de las investigaciones llevadas a cabo en los años 60 [Herbrand, Davis-Putnam, etc.]. Resolución es una técnica introducida por Robinson en 1965 y describe la manera de encontrar contradicciones en una base de datos de cláusulas con un mínimo de sustituciones.

La refutación demuestra un teorema negando la conclusión a la que se quiere llegar y añadiendo este objetivo al conjunto de axiomas que se suponen ciertos. Entonces usa la regla de resolución para mostrar que esto nos lleva a una contradicción. Una vez demostrado que el objetivo negado es inconsistente con el conjunto de axiomas, se sigue que el objetivo original debe ser consistente. Esto es la demostración del teorema.

ALGORITMO

1.- Convertir todas las fórmulas a forma clausular.

2.- Negar la conclusión y convertir el resultado a forma clausular y añadirlo al conjunto.

3.- Repetir el siguiente proceso hasta que se encuentre una contradicción o no se haga ningún progreso adicional.

a.- Seleccionar las claúsulas. Se les llama claúsulas "padre".

b.- "Resolverlas" entre si. La claúsula "resolvente" será la disyunción de todos los literales de ambas claúsulas, pero con la siguiente excepción:
Si hay un par cualquiera de literales L y ~ L, tal que una de las claúsulas padre contenga la L y otra la ~ L, entonces eliminar ambas L y ~ L del resolvente.

c.- Si el resolvente es la claúsula vacia, entonces significa que se ha encontrado una contradicción. Si el resolvente no es una cláusula vacía, añadirlo al conjunto de claúsulas disponibles.

FORMA CLAUSULAR

Chang & Lee proponen un algoritmo para reducir cualquier expresión del cálculo de predicados a forma clausular. Esta reducción no produce ninguna pérdida de valor de verdad ni de generalidad.

La forma clausular no será estrictamente equivalente al conjunto original de expresiones del cálculo de predicados en cuanto a que se perderan ciertas interpretaciones. Esto ocurre debido a que la skolemización restringe las posibles sustituciones de las variables cuantificadas existencialmente. Sin embargo, conserva la insatisfacibilidad. Es decir, si había una refutación (contradicción) dentro del conjunto original de expresiones del cálculo de predicados, sigue existiendo una contradicción en la forma clausular. Las transformaciones no sacrifican la completitud en la refutación.

ALGORITMO

1.- Se eliminan las implicaciones.

2.- Se introduce la negación hasta situarla delante de las fórmulas simples.

3.- Se normalizan las variables de tal manera que cada cuntificador esté ligado a una única variable.

4.- Se ponen todos los cuantificadores a la izquierda de la expresión sin cambiar el orden. Forma PRENEX.

5.- Se eliminan todos los cuantificadores existenciales mediante un procedimiento llamado skolemización.

6.- Se eliminan todos los cuantificadores universales.

7.- Se convierte la expresión a una conjunción de disyunciones.

8.- Se separan las conjunciones en cláusulas distintas.

9.- Se normalizan otra vez las variables.

Resolución en el cálculo de predicados, dónde un literal puede contener variables, tiene que tener un procedimiento bajo el cual dos literales con diferentes nombres de variables o uno con una constante se puedan considerar iguales. Este procedimiento se llama **UNIFICACION**, y consiste en determinar las sustituciones más generales que hagan que el predicado sea equivalente (Ver apartado anterior). Por lo tanto, el algoritmo de resolución en el cálculo de predicados tiene que:

1.- Un literal y su negación en las cláusulas padre producen un resolvente sólo si se unifican bajo una sustitución σ. Entonces σ se aplica al resolvente antes de añadirlo al conjunto de cláusulas.

2.- Las sustituciones por unificación usadas para encontrar la contradicción ofrecen ligaduras de variables bajo las cuales la consulta es cierta.

EJEMPLO

Dado el siguiente argumento, demostrar su validez utilizando el método de Robinson.

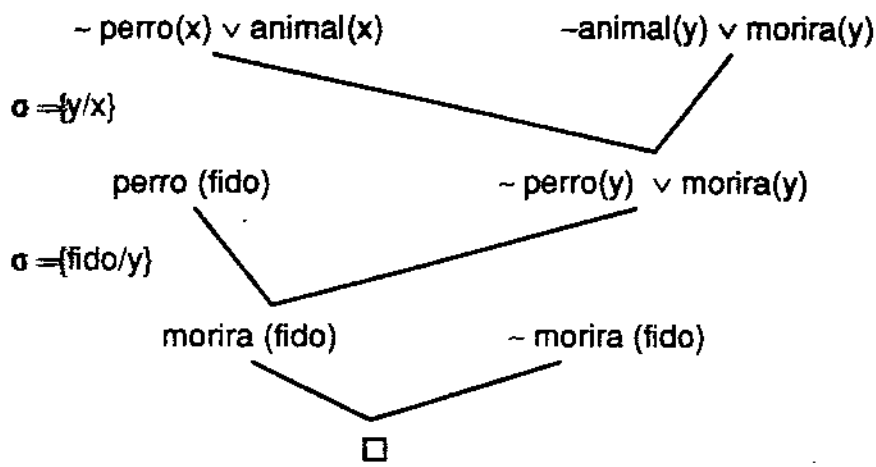
Fido es un perro.
Todos los perros son animales.
Todos los animales moriran.
Luego, Fido morirá.

PREDICADOS:

$\forall x$ [perro (x) $\rightarrow$ animal (x)]
perro (fido)
$\forall y$ [animal (y) $\rightarrow$ morira (y)]
morira (fido)

FORMA CLAUSULAR CON CONCLUSION NEGADA

~ perro(x) $\vee$ animal(x)
perro (fido)
~ animal(y) $\vee$ morira(y)
~ morira (fido)



EJEMPLO

Cualquier persona que apruebe el examen de I.A. y le toque la "Loto" se puede considerar feliz. Pero cualquier persona que estudie o sea afortunada puede aprobar todos sus examenes. Juan no estudió pero el es afortunado. Cualquier persona que sea afortunada gana la "Loto". ¿Es Juan Feliz?

TRADUCCION A CALCULO DE PREDICADOS

$\forall x$ [aprobar(x, ia) $\wedge$ gana(x, loto) $\rightarrow$ feliz(x)]
$\forall x \forall y$ [estudia(x) V afortunado(x) $\rightarrow$ aprobar(x, y)]
~ estudia (juan) $\wedge$ afortunado (juan)
$\forall x$ [afortunado(x) $\rightarrow$ gana(x, loto)

FORMA CLAUSULAR

~aprobar(x, IA) $\vee$ ~gana(x, loto) $\vee$ feliz(x)
~estudia(y) $\vee$ aprobar(y, z)
~afortunado(w) $\vee$ aprobar(w, v)
~estudia (juan)
afortunado (juan)
~afortunado(u) $\vee$ gana(u, loto)

NEGACION CONCLUSION

~feliz (juan)

SOLUCION: