LENGUAJES DE PROGRAMACIÓN

(Sesión 7)

4. PROGRAMACIÓN DECLARATIVA

4.1. Lógica computacional

4.2. Algoritmos

Objetivo: Comprender y aplicar algunos de los principales principios de la lógica computacional.

Recopilación extraída de https://www.yumpu.com/es/document/view/14367610/materiales-de-lectura-y-estudio-cursos-abiertos

Una *representación declarativa* del conocimiento es aquella en la que dicho conocimiento está especificado, pero sin embargo, no viene dada la manera en que debe ser usado tal conocimiento. Por tanto, para utilizar el conocimiento de una representación declarativa debe disponerse de un programa que especifique que debe hacerse con el conocimiento y de qué modo debe hacerse. Por contra, una *representación procedimental* es aquella en la que la información del control necesaria para utilizar el conocimiento se encuentra embebida en el propio conocimiento.

Con la aparición del lenguaje LISP en los años sesenta surge un nuevo estilo de programación, los lenguajes de *programación declarativa*, basados en un formalismo abstracto (la teoría matemática del lambda calculus de Church en el caso de LISP y el cálculo de predicados en el caso de la programación lógica) que nos indica cómo usar el conocimiento dado en el programa. Un programa declarativo es aquel cuyas sentencias tienen una interpretación declarativa, y por tanto pueden ser leídos como una descripción formal del problema sin tener que recurrir al comportamiento de ninguna máquina. Este modelo de lenguajes está claramente influido por un entendimiento matemático de las *descripciones*. El acercamiento hacia un concepto declarativo de la programación intenta aproximar las nociones de programa y de especificación. Las principales características que presentan este tipo de lenguajes son :

- Expresivos : las descripciones de los problemas no son difíciles de escribir.
- Fiables : protegen al usuario, en la medida de lo posible, de cometer errores.
- Matemáticamente elegantes : tienen un soporte matemático en la mayoría de actividades de la programación.

Aunque los orígenes de la programación declarativa datan de los primeros tiempos de la computación, sólo en los últimos años han alcanzado cierta aceptación. Ello puede ser atribuido por una parte al alcance de las inversiones en los lenguajes existentes, y por otra al "hueco semántico" entre los lenguajes declarativos y los ordenadores convencionales. Los avances tecnológicos recientes han estrechado este hueco y, por ello, han aumentado las ventajas de la programación declarativa. Otro importante aspecto en favor de la programación declarativa es el desarrollo de las nuevas arquitecturas paralelas de los ordenadores.

Hay dos clases de lenguajes declarativos : los lenguajes funcionales y los lógicos. Nosotros estudiaremos los lenguajes lógicos. La programación lógica trata con relaciones en lugar de con funciones, lo que nos proporciona mayor flexibilidad, ya que las relaciones no tienen sentido de la dirección y tratan uniformemente argumentos y resultados (no hay distinción entre parámetros de entrada y de salida). Consideraremoslas sentencias lógicas (fórmulas bien formadas) como representaciones del conocimiento, y por ello, como líneas de código de nuestros programas.

La aparición de este nuevo paradigma de programación, la PROGRAMACIÓN LÓGICA, viene precedida de distintos estudios y avances en el campo de la informática. Vamos a comentar brevemente algunos de ellos:

- El cambio cualitativo producido a primeros de los años setenta en el concepto de la programación introduciendo la metodología de diseño estructurado y la verificación de programas fue un primer paso de replanteamiento del diseño de los procesos: en lugar de basados en *cómo computarlos*, basados en *cómo entenderlos*.

- Los desarrollos en el área de la deducción automática dieron lugar al inicio de una línea en la que se plantea la simple especificación lógica de los procesos prescindiendo de la especificación de procedimientos.

- La base de estos métodos es la regla de resolución de Robinson (1965) y las técnicas de extracción de respuestas de los procesos de resolución iniciados por Green (1969) y seguidos por Colmerauer (1972) y Kowalski (1974).

Así, la **Programación Lógica** se ha convertido en nueva línea de desarrollo de software, válida tanto para la especificación de algoritmos como para modelos de Inteligencia Artificial, ambivalencia que ha dado lugar a su utilización como base del proyecto de ordenadores de quinta generación (máquinas especializadas en el procesamiento de conocimientos).

Veamos una rápida descripción de lo que es la programación lógica ; descripción que detallaremos en los siguientes apartados. La *programación lógica* usa como base sentencias de la lógica de primer orden, en concreto las cláusulas de Horn (restricción del Cálculo de Predicados de Primer Orden) y su forma de ejecución es el principio de Resolución de Robinson. Trata de representar conocimiento mediante *relaciones* (predicados) entre *objetos* (datos). Un programa lógico consiste en un conjunto de relaciones, y su ejecución vendrá a demostrar que una nueva relación se sigue de las que constituían el programa. Las relaciones serán especificadas con *reglas* y *hechos*. La ejecución de programas lógicos consiste en la demostración de hechos sobre las relaciones por medio de *preguntas*. Así, el hecho de programar consistirá en proporcionar al ordenador un universo finito en forma de hechos y reglas (*Base de Conocimientos*), dotándolo de los medios para realizar inferencias de un hecho a otro. Si posteriormente se le hacen las preguntas adecuadas, el sistema buscará las respuestas en dicho universo y nos las mostrará en pantalla.

Para terminar esta introducción comentar que la idea central de todo esto la podemos expresar utilizando la conocida ecuación de Kowalski:

algoritmo = lógica +control

de manera que el *control* (estrategia para encontrar la solución) la dejamos en manos de la maquina, y sólo debemos preocuparnos de la *lógica* (información acerca del problema que queremos resolver).

SINTAXIS Y SEMÁNTICA DE LOS LENGUAJES DE PROGRAMACIÓN

Un lenguaje de programación queda totalmente definido por su *sintaxis* y su *semántica*. La sintaxis nos dirá "cómo" escribir nuestros programas, y la semántica "qué" significan dichos programas.

La **sintaxis** es lo primero que tenemos que conocer de un lenguaje de programación para poder empezar a usarlo, y nos dice cuál es la forma correcta de escribir los programas en dicho lenguaje. El aspecto sintáctico de los programas lógicos lo veremos, de manera general, en el siguiente apartado (3.2) "Representación de Programas Lógicos", y cuando se estudie Prolog se dará la sintaxis concreta de este lenguaje.

La **semántica** del lenguaje especifica el significado de los programas que podemos construir con este lenguaje. El conocimiento de la semántica será necesario para poder escribir programas correctos y a la vez ser capaces de predecir los efectos de la ejecución de cualquier instrucción. Para hacer la definición formal de la semántica de un lenguaje existen distintos métodos. Vamos a comentar brevemente algunos de ellos.

La *semántica declarativa* es un tipo de semántica que especifica el significado de los objetos sintácticos por medio de su traducción en elementos y estructuras de un dominio matemático conocido :

La semántica denotacional trata al programa como si fuera un conjunto de funciones matemáticas, la composición de las cuales nos dará su significado. Las funciones se definen en términos de

cambios de estado ; dado un constructor y un estado, obtenemos un nuevo estado.

La semántica por teoría de modelos expresa el significado de un programa por el conjunto de sus consecuencias lógicas. Se basa en los conceptos lógicos de interpretación y modelo.

La *semántica operacional* es la más cercana a la intuición del programador. En un primer momento define una máquina abstracta que soporte un conjunto de operaciones y estructuras de datos simples ; posteriormente se definen los constructores del lenguaje en términos de la máquina definida. La semántica operacional sería como un interprete de nuestro lenguaje en la máquina abstracta, así el significado de nuestro programa se define en términos de las acciones que serían ejecutadas por dicho modelo abstracto de máquina.

REPRESENTACIÓN DE PROGRAMAS LÓGICOS

FORMA CLAUSAL

Cualquier fórmula del Cálculo de Predicados puede ser normalizada transformándola a **Forma Clausal**, que consiste en una colección de cláusulas, donde cada cláusula es a su vez una disyunción de literales { L_1, ..., L_k}. Estos literales son fórmulas atómicas negadas o no. Así, en la representación en forma clausal, tendremos un subconjunto {A_1, ..., A_n} de literales afirmados y otro {$\neg N_1$, ..., $\neg N_m$} de negados.

$$C = \{L_1 , ..., L_k\} = \{A_1 , ..., A_n\} \quad \{\neg N_1 , ..., \neg N_m\}$$

La notación en Forma Clausal tiene la ventaja de que reduce a una forma única lo que se puede escribir de diversas formas. Esto resulta imprescindible si queremos llevar a cabo manipulaciones formales sobre fórmulas del Cálculo de Predicados y desarrollar procesos de automatización. De ahí la importancia de la forma clausal cuando tratamos de aplicar la lógica a la informática y en concreto a la programación lógica.

Para transformar a forma clausal partiremos de una fórmula bien formada de la Lógica de Primer Orden que no tiene variables libres y seguiremos los pasos enumerados a continuación:

1º) REDUCIR CONSTANTES LÓGICAS : consiste en la eliminación de los implicadores y coimplicadores que puedan aparecer en la fórmula original. Para ello podemos utilizar la definición del coimplicador en términos de implicación y conjunción para eliminar los coimplicadores; y la definición del implicador en función del negador y la disyunción para eliminar los implicadores:

2º) NORMALIZAR NEGADORES: interiorización de los negadores de manera que cada negador quede directamente adosado a una fórmula atómica. Para ello podemos utilizar las leyes de De Morgan, la Doble Negación, la Negación del Universal y la Negación del Existencial:

3º) NORMALIZAR VARIABLES : si es necesario, renombrar variables de forma que cada cuantificador tenga como índice una variable con nombre distinto.

4º) ELIMINAR CUANTIFICADORES EXISTENCIALES : para lo cual utilizaremos las constantes y las funciones de skolem.

a) Si el *cuantificador existencial <u>está</u> en el ámbito de un cuantificador universal*, entonces admitimos la posibilidad de que la variable del cuantificador existencial dependa del valor de la variable afectada por el cuantificador universal; esta dependencia la representamos por el nombre de una función. Por tanto, reemplazaremos todas las ocurrencias de esta variable cuantificada existencialmente por una ***función de Skolem*** cuyos argumentos serán aquellas variables cuantificadas universalmente en cuyo ámbito esté el cuantificador existencial que se elimina.

b) Si el *cuantificador existencial a eliminar* <u>*no está*</u> *dentro del ámbito de ningún cuantificador universal*, sustituimos la variable cuantificada existencialmente por una función sin argumentos: ***constante de Skolem.***

5º) FORMA PRENEXA : adelantar los cuantificadores universales a la cabeza de la fórmula. En este punto tenemos que toda fórmula parcial está conectada a una cuantificada por disyunción ó conjunción, no hay dos cuantificadores con el mismo
índice (variable) y no existen cuantificadores existenciales. Por tanto aplicamos :

6º) ELIMINAR CUANTIFICADORES UNIVERSALES: llegados a este punto, sabemos que todas las variables que aparecen están cuantificadas universalmente, por tanto para simplificar la notación no escribiremos los prefijos cuantificacionales.

7º) EXTERIORIZAR CONJUNTORES: obtener la Forma Normal Conjuntiva de la matriz de la fórmula. Utilizaremos las leyes Distributivas de la Disyunción respecto a la Conjunción.

8º) EXTRAER LAS CLÁUSULAS: eliminación de conjunciones (separación en cláusulas) y obtención de las distintas cláusulas.

9º) NORMALIZAR VARIABLES: si es necesario, volvemos a renombrar las variables.

Ejemplos:

1.- Obtener la FC de la siguiente fbf:

$$\forall x \{ [P(x) \bullet \neg \forall y (Q(x, y) \bullet \forall z P(z))] \bullet \forall y (Q(x, y) \bullet R(y)) \}$$

Solución:

1º. Eliminar implicadores y coimplicadores

$$\forall x \{ [\neg P(x) \lor \neg \forall y (\neg Q(x, y) \lor \forall z P(z))] \land \forall y (\neg Q(x, y) \lor R(y)) \}$$

2º. Normalizar negador

$$\forall x \{ [\neg P(x) \lor \forall y \neg(\neg Q(x, y) \lor \forall z P(z))] \land \forall y (\neg Q(x, y) \lor R(y)) \} \quad \text{I, NU}$$

$$\forall x \{ [\neg P(x) \lor \forall y (\neg\neg Q(x, y) \land \neg \forall z P(z))] \land \forall y (\neg Q(x, y) \lor R(y)) \} \quad \text{I, DM}$$

$$\forall x \{ [\neg P(x) \lor \forall y (\neg\neg Q(x, y) \land \exists z \neg P(z))] \land \forall y (\neg Q(x, y) \lor R(y)) \} \quad \text{I, NE}$$

T/x { [•P(x) v 3y (Q(x, y)'' z •P(z))] ''T/y (•Q(x, y) v R(y))} I, E

3°. Normalizar variables

T/x { [•P(x) v 3y (Q(x, y) ''T/z -.p(z))] ''T/w (•Q(x, w) v R(w))}

4°. Eliminar cuantificadores existenciales: forma de Skolem

T/x { [•P(x) v (Q(x, f(x)) ''T/z -.p(z))] ''T/w (•Q(x, 1) v R(v.r))}

5°. Sacar cuantificadores universales fuera de la formula: forma Prenexa

T/x { [•P(x) v T/z (Q(x, f(x)) '' -.p(z))] '' T/w (•Q{x, 1) v R(w)) }

T/x T/z T/w { [•P(x) v (Q(x, f(x)) '' •P(z))] '' (•Q(x, w) v R(w)) }

6°. Eliminar cuantificadores universales

[•P(x) v (Q(x, f(x)) '' •P(z))) '' (•Q(x, w) v R('))

7°. FNC de la matriz de la formula: aplicamos la propiedad distributiva

[•P(x) v Q(x, f(x))] '' [•P(x) v •P(z)] '' [•Q(x, w) v R(w)]

8°. Separamos las Clausulas:

C1: •P(x) v Q{x, f(x))

C2: •P(x) v -.p(z)

C3: •Q(x, w) v R(w)

9°. Normalizamos variables

Ct: -.p(Xt) v Q(Xt, f(Xt))

C2: ,P(x2) v -.p(z)

C3: -.Q(x3, w) v R(w)

\Ix 3y{ -A(x,y) (B(x) A C(y)).JA x [B(x) -4. vx 'Iiy D{x,y)J A -.3x E(x) A l \Ix \Iy D(x,y) 3x E(x)J

S.o uci 6n:

1°. Eliminar implicadores y compiladores:

Ttx 3y [•A(x,y) v (IB(x) A O(y))] A Ttx [-.B(x} v -.lrfx Tfy D(x,y)] A •3x E(x) A l Tfx '>r/lD(xy) v 3x E(x)]

2°. om1allizar negadores:

Vx 3y [-.A(x:,y) v (B(x)0 A qy))] A Vx [•'B(x) v 3x 3y •D(xy)0] A Vx •'E(x A [Vx \rf1J D(x,y) v 3x IE(x)]

3°. Normalirzar variables:

Vx 3y [-.A(x,y) v (B(x)0 A C(y))] A Vz [•8 Z) v 3v 3w -.D(v, IV}] A Vu -.E(UJ) A [Vs VlD(s.,t) v 3r E(r)

4°. Eliminar cuantificadores existenciales:

VX [•A(x,f(x)) v (B(x)0 A C(f(x)) 1 A Vz 1 •'B(z) v -.D(g(z),h(z)}] A \tiu•oE{u) A [Vs \It D(s, t) v E[a} 1

5o. Forma prenexa:

Ttx Vz Vu Vs Vt { [•A(x,f{x)) v {B(x) A C(f(x) 1 A 1 •B(z) v •D(g(z),h(z)) A -.E(u) A [D(s,) v IE(a)

6o. Eliminar cuantifica.dlores universales:

[•A{x, f(x)) v (B(x) A C(f(x))] A [-.B{z) v -.D{g(z), h(z)} 1A -.E(u) A [O(s, t) v E(a)]

7°. FNG de la matriz de la formula:

l -.A(x,f(x)) v B x)] A [-.A(x, f{x)) v C(f(x:}}] A [-.B{z) v •D(g(z),lh(z)} 1A -.E(u) A l[ID(s,I) v E(a)]

8°. Extraccion de clausulas: elim

inacibn de 0 ::0nj nci ones

G_1: •A{x, f{x)) v B(x ,

G_2: •A{x, f{x)) v C(f(x))

NOTACIÓN PARA LA PROGRAMACIÓN LÓGICA

Para poder acercarnos en lo posible a los lenguajes de programación, a partir de ahora, para las fórmulas lógicas en forma clausal utilizaremos la siguiente notación específica de la programación lógica (es una notación más intuitiva y cercana a una interpretación procedimental):

1. Como es una colección de cláusulas, las escribiremos en secuencia. Recordemos que el orden es irrelevante ya que se trata de conjunciones (propiedades asociativa y conmutativa de la conjunción).

2. Cada cláusula es una colección (disyunción) de literales, que son fórmulas atómicas afirmadas o negadas:

- Escribiremos primero las fórmulas atómicas no negadas llamadas *cabeza de la cláusula*, seguidas de las negadas, *cuerpo de la cláusula*, separados ambos grupos por el símbolo especial " ← " (si).

- Las fórmulas atómicas no negadas (cabeza de la cláusula) irán separadas por el símbolo ";" (o), y las negadas (cuerpo de la cláusula) serán escritas sin el negador (¬) y separadas por el símbolo "," (y).

Ejemplos :

1.- Supongamos que tenemos la siguiente cláusula:

$$p \vee \neg q \vee \neg r \vee s \vee \neg t \vee u$$

si agrupamos y realizamos transformaciones:

$$\neg q \vee \neg r \vee \neg t \vee p \vee s \vee u$$

$$\neg(q \wedge r \wedge t) \vee (p \vee s \vee u)$$

$$(q \wedge r \wedge t) \leftarrow (p \vee s \vee u)$$

que significa que:
o lo que es lo mismo:

“Si ocurren q, r y t entonces deben ocurrir p, s o u”

“Para resolver el problema p, s o u, antes deben resolverse q, r y t”

es decir, q, r y t son suficientes para que se de p, s o u.

La cláusula anterior, escrita en la notación que acabamos de comentar quedaría:

p ; s ; u ← q , r , t

2.- Tenemos la fórmula del Cálculo de Predicados:

∀y ∀x (mujer(x) ∧ mujer(y) ∧ ∀p ∀m (padresDe(x, p, m) ∧ padresDe(y, p, m)) → hermanas(x, y))

1º Obtenemos la Forma Clausal correspondiente:

∀y ∀x (¬(mujer(x) ∧ mujer(y) ∧ ∀p ∀m (padresDe(x, p, m) ∧ padresDe(y, p, m)))∨hermanas(x,y))

∀y ∀x (¬mujer(x)∨¬mujer(y)∨¬∀p∀m(padresDe(x, p, m)∧padreDe(y, p, m))∨hermanas(x, y))

∀y ∀x (¬mujer(x)∨¬mujer(y)∨∀p∀m¬(padresDe(x, p, m)∧padreDe(y, p, m))∨hermanas(x, y))

∀y ∀x ∀p ∀m (¬mujer(x)∨¬mujer(y)∨¬padresDe(x, p, m)∨¬padreDe(y, p, m)∨hermanas(x, y))

∧mujer(x) ∨ ∧mujer(y) ∨ ∧padresDe(x, p, m) ∨ ∧padresDe(y, p, m) ∨ hermanas(x, y)

2º La cláusula la escribimos en la notación de la programación lógica vista antes:

hermanas(x, y) ← mujer(x), mujer(y), padresDe(x, p, m), padresDe(y, p, m)

que se puede leer como:

"Dos personas son hermanas si ambas son mujeres y tienen los mismos padres"

CLAUSULAS DE HORN DEFINIDAS

Vamos a restringir el conjunto de fórmulas que utilizaremos a las llamadas *fórmulas de Horn*[25], lo cuál nos permitirá una mejor y más eficiente implementación de la comprobación de teoremas. De esta manera, a partir de ahora trabajaremos con una "sublógica" (subconjunto) de la Lógica de Predicados, que llamaremos **programas lógicos**.

Ejemplo :

Retomando el ejemplo1 del apartado anterior, teníamos que la cláusula:

p ; s ; u ← q , r , t

significaba que:

"si ocurren q, r y t entonces deben ocurrir p, s o u"

pero no sabremos cual (o cuales) de las 3 alternativas (p, s o u) es la que se cumplirá. Para poder trabajar sin estás opciones, obligaremos a que nuestras cláusulas tengan una única posibilidad en la cabeza.

Cláusula de Horn Definida : es una cláusula con, como máximo, un literal no negado.

Siguiendo con la representación utilizada antes para las cláusulas:

$$C = \{L_1, ..., L_k\} = \{A_1, ..., A_n\} + \{\neg N_1, ..., \neg N_m\}$$

diremos que es una cláusula de Horn definida si n 1. Así podemos diferenciar los siguientes tipos de cláusulas de Horn:

[25] Reciben este nombre porque este tipo de fórmulas fueron investigadas por primera vez por el lógico Alfred Horn (1951).

1.- Hipótesis o cláusulas con un literal no negado (n=1), llamadas *cláusulas con cabeza*:

$$C = \{A\} + \{\neg N_1 , ..., \neg N_m\}$$

que pueden ser a su vez:

- **Hechos**: sin literales negados (m=0) $\{A\}$

Ejemplo: mujer(lirios) ←

- **Reglas**: con literales negados (m>0) $\{A,\neg N_1,...,\neg N_m\}$

Ejemplo:

hermanas(x, y) · mujer(x), mujer(y),
padresDe(x, p, m),
padresDe(y, p, m)

2.- Preguntas o cláusulas con ningún literal no negado (n=0) o teoremas a demostrar, llamadas *cláusulas sin cabeza* o decapitadas

- **Metas** u **objetivos**: (m>0) $\{\neg N_1 , ..., \neg N_m\}$

Ejemplo: · hermanas(lirios, rosana)

Hay trabajos demostrando que todo problema resoluble con cláusulas generales, tiene un modelo equivalente resoluble en Cláusulas de Horn, por lo que las restricciones de notación no conllevan una pérdida de capacidad de representación, y en cambio, sí que facilitan la formulación.

INTERPRETACIÓN DE LAS VARIABLES EN UN PROGRAMA LÓGICO

Explícitamente la Programación Lógica no utiliza los símbolos de cuantificación para las variables, pero implícitamente sí que están cuantificadas las variables de

trérminos. En general, todas las variables que aparecen están cuantificadas universalmente, ya que proceden de la notación en forma clausal, y como hemos visto antes (apartado 3.2.1), todas las variables están cuantificadas universalmente aunque ya no escribamos explícitamente el cuantificador (paso 6ª de la transformación a forma clausal : eliminación de cuantificadores universales). Pero veamos que significado tienen dependiendo de su ubicación. Si las fórmulas atómicas de un programa lógico contienen variables, el significado de estas es :

Las variables que aparecen en los hechos están cuantificadas universalmente ya que en una cláusula todas las variables que aparecen están cuantificadas universalmente de modo implícito.

gusta(jorge,x) · equivale a la fórmula ∀x gusta(jorge,x)

y significa que a jorge le gusta cualquier cosa

Las variables que aparecen en la cabeza de las reglas (átomos afirmados) están cuantificadas universalmente. Las variables que aparecen en el cuerpo de la regla (átomos negados), pero no en la cabeza, están cuantificadas existencialmente.

abuelo(x,y) · padre(x,z), padre(z,y) equivale a la fórmula

$\forall x \forall y \forall z$ [abuelo(x,y) ∨ ¬padre(x,z) ∨ ¬padre(z,y)]

$\forall x \forall y$ [abuelo(x,y) ∨ $\forall z$ ¬[padre(x,z) ∧ padre(z,y)]]

$\forall x \forall y$ [abuelo(x,y) ∨ ¬$\forall z$ [padre(x,z) ∧ padre(z,y)]]

$\forall x \forall y$ [$\forall z$ [padre(x,z) ∧ padre(z,y)] ⊃ abuelo(x,y)]

que significa que para toda pareja de personas, una será el abuelo de otra si existe alguna persona de la cuál el primero es padre y a su vez es padre del segundo

Las variables que aparecen en las preguntas están cuantificadas existencialmente.

· gusta(jorge,x) equivale a la fórmula $\forall x$ ¬gusta(jorge,x) ≡ ¬$\forall x$ gusta(jorge,x)

y que pregunta si existe algo que le guste a jorge, ya que como veremos utilizamos refutación y por tanto negamos lo que queremos demostrar.

SEMÁNTICA OPERACIONAL

PRINCIPIO DE RESOLUCIÓN

Un hecho fundamental en el desarrollo de la *demostración automática de teoremas* fue el descubrimiento del **principio de resolución** por J. Alan Robinson[26], que nos dice cómo puede derivarse una proposición a partir de otras. Así utilizando el principio de resolución podemos comprobar teoremas de una forma puramente mecánica como veremos más adelante.

Cláusulas Padre

{ L, A_1, A_2, ..., A_n } { M, B_1, B_2, ..., B_m }

= UMG(L, ¬M)

{ A_1, ..., A_n, B_1, ..., B_m }④

Cláusula Resolvente

Regla de Resolución

[26] Robinson, J.A. "A Machine-oriented Logic Based on the Resolution Principle". Journal of the ACM (Association for Computing Machinery), vol. 12, nº 1, 1.965, pp. 23-41

Ejemplo:

1.- Sean las siguientes premisas:

Todos los informáticos saben lógica

Juan es informático

¿Qué puedo deducir?

Si obtenemos la forma clausal de las premisas, tendremos las siguientes cláusulas padre

$$\{ \neg I(x) \vee L(x) \} \text{ y } \{ I(juan) \}$$

Por resolución obtengo la resolvente L(juan) con la sustitución $\theta = \{ juan/x \}$

2.- Sean las siguientes cláusulas padre

$$\{ P(x, f(a)) \vee P(x, f(y)) \vee Q(y) \} \text{ y } \{ \neg P(b, f(a)) \vee \neg Q(b) \}$$

Con { a/y } unifico P(x, f(a)) y P(x, f(y)) en P(x, f(a))

Y con { b/x } unifico P(x, f(a)) y P(b, f(a))

Por resolución obtengo la resolvente Q(a) $\quad Q(a) \vee \neg Q(b)$ con el UMG $\theta = \{ b/x, a/y \}$

Un resultado muy interesante de la regla de resolución es:

Si aplicando la regla de resolución a un conjunto de cláusulas se obtiene la cláusula vacía (NADA), el conjunto de cláusulas es ***insatisfactible***

Esta característica de la regla de resolución de encontrar las contradicciones nos servirá, como veremos más adelante, para hacer demostraciones aplicando la técnica de reducción al absurdo o refutación.

El problema es que la resolución no nos dice cómo decidir qué cláusulas mirar ni qué literales unificar. Para ello están las *estrategias de resolución*, que caen dentro del campo de la Inteligencia Artificial y que pasaremos a comentar brevemente.

Bibliografía Recomendada:

"Razonando con Haskell. Un curso sobre programación funcional" B.C. Ruiz, F. Gutiérrez, P. Guerrero y J.E. Gallard Ed. Thomson , 2004.

"Introducción a la programación funcional con Haskell" (2ed) R. Bird Ed. Prentice-Hall, 1999.

"The Craft of Functional Programming" cS. Thompson

Richard Bird. Introducción a la Programación Funcional con Haskell. Prentice-Hall, 2000.

Seif Haridi and Peter van Roy. Concepts, Techniques, and Models of Computer Programming. The MIT Press, 2004.

Paul Hudak. The Haskell School of Expression. Learning functional programming through multimedia. Cambridge University Press, 2000.

Ulf Nilsson and Jan Maluszynski Logic, Programming and Prolog (2ed) John Wiley & Sons Ltd, 2000.

Rinus Plasmeijer and Marko van Eekelen. Functional Programming and Parallel Graph Rewriting. Addison-Wesley, 1993.

Ed. Addison-Wesley, 1999. Adicional

K. R.. Apt. From Logic Programming to Prolog. Prentice Hall. 1997.

C. S. Clocksin, C. W. Mellish. Programacion en Prolog. Gustavo Gili. 1988.

J. Lloyd. Foundations of Logic Programming. Springer-Verlag. 1987.

L. Sterling, E. Shapiro. The Art of Prolog. MIT Press. 1986.

Graham Hutton; Programming in Haskell; Cambridge University Press, 2007;
* Alejandro Serrano Mena; Beginning Haskell: A Project-Based Approach; Apress, 2014;
* R. Bird; Introducción a la Programación Funcional con Haskell; Segunda edición, Prentice Hall, 2000;
* B.C. Ruiz, F. Gutiérrez, P. Guerrero, J.E. Gallardo; Razonando con Haskell: un curso sobre programación funcional; Thomson, 2004;
Libros de programación lógica
* L.Sterling, E.Shapiro; The Art of Prolog. Advanced Programming Techniques; The MIT Press, 2ª Edición, 1994;
* P. Julián, M. Alpuente; Programación Lógica, Teoría y Práctica; Pearson, 2007;
* W.F. Clocksin, C.S. Mellish; Programming in Prolog Using the ISO Standard; Springer Verlag, 5ª edición, 2003;

Sitios consultados
http://www.ida.liu.se/~ulfni/lpp/bok/bok.pdf

http://babel.ls.fi.upm.es/teaching/
http://djaramillo2dani.blogspot.mx/2011/04/guia-practica-uno-1-estructura-228106.html
http://marcelo-trabajo.blogspot.mx/
http://cursos.aiu.edu/Lenguages%20de%20Programacion/PDF/Tema%201.pdf
http://algoritmosylenguajes.blogspot.mx/2008/05/unidad-iii_31.html