LÓGICA COMPUTACIONAL
PROGRAMACIÓN LÓGICA

Francisco Hernández Quiroz

Departamento de Matemáticas
Facultad de Ciencias, UNAM
E-mail: fhq@ciencias.unam.mx
Página Web: www.matematicas.unam.mx/fhq

Facultad de Ciencias

Introducción

- La programación lógica es uno de los paradigmas de programación más alejados del imperativo.
- Un sistema de demostración de teoremas funciona como un mecanismo computacional.
- La programación lógica es declarativa: un programa es una *declaración* de qué se desea, y no una receta de cómo obtenerlo.
- En la práctica, no siempre se puede alcanzar este ideal, por cuestiones de eficiencia.
- La fundamentación semántica de la programación lógica son los modelos de Herbrand.
- Un problema difícil en programación lógica es el de la *negación*. Aquí se presentarán dos enfoques distintos del tema.
- El lenguaje Prolog es un compromiso entre los principios de la programación lógica y la eficiencia de implementación.

Unificación

La unificación es un caso especial de sustitución.

Definición

Sean E y F dos expresiones (ya sean dos fórmulas atómicas o dos términos funcionales). Un unificador *de E y F es una sustitución σ tal que*

$$E_\sigma = F_\sigma.$$

Ejemplos

Los términos

$$f_1^2(x, f_1^2(c, z)) \qquad \text{y} \qquad f_1^2(c, f_1^2(y, x))$$

son unificados por la sustitución

$$[x, y, z := c, c, c],$$

pues

$$f_1^2(x, f_1^2(c, z))_{[x,y,z:=c,c,c]} = f_1^2(c, f_1^2(c, c)) = f_1^2(c, f_1^2(y, x))_{[x,y,z:=c,c,c]}.$$

A las fórmulas atómicas

$$P_1^2(f_1^2(x, y), c) \qquad \text{y} \qquad P_1^2(z, y)$$

las unifica la sustitución

$$[z, y := f_1^2(x, c), c]$$

$$P_1^2(f_1^2(x, y), c)_{[z,y:=f_1^2(x,c),c]} = P_1^2(f_1^2(x, c), c) = P_1^2(z, y)_{[z,y:=f_1^2(x,c),c]}.$$

No siempre es posible unificar dos expresiones:

$$f_1^2(x, c) \qquad \text{y} \qquad f_1^2(c_1, x)$$

no tienen un unificador.
En cambio, en otros casos tenemos más de uno, como con las fórmulas

$$P_1^2(x, y) \qquad \text{y} \qquad P_1^2(y, x)$$

con los unificadores

$$[x := y], \qquad [y := x] \qquad \text{y} \qquad [x, y := c, c] :$$

$$\begin{aligned} P_1^2(x, y)_{[x:=y]} &= P_1^2(y, y) = P_1^2(y, x)_{[x:=y]} \\ P_1^2(x, y)_{[y:=x]} &= P_1^2(x, x) = P_1^2(y, x)_{[y:=x]} \\ P_1^2(x, y)_{[x,y:=c,c]} &= P_1^2(c, c) = P_1^2(c, c)_{[x,y:=c,c]} \end{aligned}$$

Composición de sustituciones

La composición de dos sustituciones se define así:

$$\begin{aligned} \sigma &= [x_1, \ldots, x_n := t_1, \ldots, t_n] \\ \eta &= [y_1, \ldots, y_m := s_1, \ldots, s_m] \\ \eta \circ \sigma = \sigma\eta &= ([x_1, \ldots, x_n := t_{1\eta}, \ldots t_{n\eta}] - [x_i := t_{i\eta} \mid x_i = t_{i\eta}]) \\ &\quad \cup ([y_1, \ldots, y_m := s_1, \ldots, s_m] - [y_i := s_i \mid y_i \in dom(\sigma)]) \end{aligned}$$

Ejemplo

Sean

$$\begin{aligned} \sigma &\equiv_{def} [x, y, w := z, f_1^1(x), v] \\ \eta &\equiv_{def} [z, x, v := y, c, w] \end{aligned}$$

dos sustituciones. Entonces

$$\begin{aligned} \sigma\eta &= [x, y, w := z_\eta, f_1^1(x)_\eta, v_\eta] \cup ([z, x, v := y, c, w] - [x := c]) \\ &= ([x, y, w := y, f_1^1(c), w] - [w := w]) \cup [z, v := y, w] \\ &= [x, y, z, v := y, f_1^1(c), y, w] \end{aligned}$$

Unificadores de máxima generalidad

Definición

Sean E y F dos expresiones y sea Σ el conjunto de sus unificadores. Un unificador de máxima generalidad *(umg) es una sustitución $\mu \in \Sigma$ tal que para toda $\sigma \in \Sigma$ se tiene que $\mu\sigma = \sigma$.*

Ejemplos

En el ejemplo de las expresiones $P_1^2(x, y)$ y $P_1^2(y, x)$, los dos primeros unificadores propuestos son de máxima generalidad, pues

$$\begin{aligned}
[x := y][x, y := c, c] &= [x, y := c, c] \\
[x := y][y := x] &= [y := x] \\
[y := x][x, y := c, c] &= [x, y := c, c] \\
[y := x][x := y] &= [x := y]
\end{aligned}$$

Diferencias

Definición

(a) Sean E y F dos expresiones. El conjunto diferencia de E y F, denotado por $d(E, F)$, se define así:

$$\begin{aligned}
d(E, F) &= \emptyset \qquad \text{si } E = F \\
d(f_m^n(t_1, \ldots, t_n), f_m^n(t'_1, \ldots, t'_n)) &= d(t_1, t'_1) \cup \cdots \cup d(t_n, t'_n)) \\
d(P_m^n(t_1, \ldots, t_n), P_m^n(t'_1, \ldots, t'_n)) &= d(t_1, t'_1) \cup \cdots \cup d(t_n, t'_n)) \\
d(E, F) &= \{E : F\} \qquad \text{en cualquier otro caso}
\end{aligned}$$

(b) Una sustitución reductora *de $d(E, F)$ es una sustitución $x := t$ derivada de algún par $E_i : F_i \in d(E, F)$ tal que $x = E_i$ y $t = F_i$ (o viceversa) y x no aparece en t.*
(c) $d(E, F)$ es reductible *sii se puede derivar una sustitución reductora de todo par $t : t' \in d(E, F)$.*

Un algoritmo para umg

El algoritmo es muy simple: cónstruyase una sucesión de pares de expresiones y otra sucesión de sustituciones:

$$\begin{aligned}
(E_0, F_0) &= (E, F) \\
(E_{n+1}, F_{n+1}) &= (E_{n\sigma_n}, F_{n\sigma_n}) \\
\mu_0 &= \emptyset \\
\mu_{n+1} &= \mu_n \sigma_n
\end{aligned}$$

donde σ_i es una sustitución reductora de E_i y F_i. La sucesión $\langle (E_0, F_0), \ldots \rangle$ es finita y al final se llega a uno de dos casos: (a) $d(E_n, F_n) = \emptyset$, o (b) el conjunto $d(E_n, F_n)$ es irreductible.

Ejemplo

Unificaremos las expresiones $f_1^2(x, f_2^2(a, y))$ y $f_1^2(a, f_2^2(x, z))$:

$$\begin{aligned}
d(f_1^2(x, f_2^2(a, y)), f_1^2(a, f_2^2(x, z))) &= \{x : a, a : x, y : z\} \\
\sigma_0 &= [x := a] \\
\mu_1 &= \sigma_0 \\
(f_1^2(x, f_2^2(a, y))_{\sigma_0}, f_1^2(a, f_2^2(x, z))_{\sigma_0}) &= (f_1^2(a, f_2^2(a, y)), f_1^2(a, f_2^2(a, z)) \\
d(f_1^2(a, f_2^2(a, y)), f_1^2(a, f_2^2(a, z)) &= \{y : z\} \\
\sigma_1 &= [y := z] \\
\mu_2 &= \mu_1\sigma_1 = [x := a, y := z] \\
(f_1^2(a, f_2^2(a, y))_{\sigma_1}, f_1^2(a, f_2^2(a, z)_{\sigma_1}) &= (f_1^2(a, f_2^2(a, z), f_1^2(a, f_2^2(a, z))
\end{aligned}$$

y como los dos últimos términos son iguales, el umg buscado es μ_2.

Cláusulas definidas

- En principio se podría tener un lenguaje de programación lógica que abarcara todo el cálculo de predicados de primer orden.
- Por cuestiones de eficiencia computacional nos ocuparemos sólo de las fórmulas que se ajustan a una sintaxis particular y que se conocen como *cláusulas definidas*.

Definición

Una cláusula definida *es un enunciado de la forma*

$$\forall x_1, \ldots, x_m \,.\, \exists y_1, \ldots, y_k \,.\, \beta_1 \wedge \cdots \wedge \beta_n \Rightarrow \alpha,$$

donde $\beta_1, \ldots, \beta_n$ *y* α *son fórmulas atómicas,* $x_1, \ldots, x_m$ *son las variables que aparecen en* α *y* $y_1, \ldots, y_k$ *son las variables que aparecen en las* β_i *pero no en* α*. Siguiendo la tradición, la cláusula del ejemplo se escribirá*

$$\alpha \leftarrow \beta_1, \ldots, \beta_n.$$

La fórmula α *se conoce como el* encabezado*, mientras que* $\beta_1, \ldots, \beta_n$ *son el* cuerpo*.*
Una fórmula puede caracer de cuerpo o de encabezado, y en este último caso se llamará una meta*.*
Un programa lógico *es un conjunto de cláusulas definidas, ninguna de las cuales es una meta.*

Más sobre las metas

Por otra parte, conviene notar que de acuerdo con nuestra notación una meta equivale a la negación de un enunciado existencial:

$$\leftarrow \beta \equiv \neg \exists y_1, \ldots, y_k \,.\, \beta.$$

donde $y_1, \ldots, y_k$ aparecen libres en β.

Ejemplo

1. $P_1^2(c, f_1^1(c)) \leftarrow$
2. $P_1^2(f_1^1(c), f_1^1(c)) \leftarrow$
3. $P_1^2(f_1^1(f_1^1(x)), f_1^2(y, z)) \leftarrow P_1^2(x, y), P_1^2(f_1^1(x), z)$.

Si se interpretan las funciones f_1^1 y f_1^2 como sucesor y +, respectivamente, y la constante c como 0, P_1^2 corresponde a una relación que asocia el natural n con el n-ésimo número de Fibonacci.
Para hacer más transparentes los programas lógicos, en adelante nuestro lenguaje lógico incluirá símbolos que hagan obvia su interpretación.
El programa anterior se puede rescribir así:

1. $\text{Fib}(0, s(0)) \leftarrow$
2. $\text{Fib}(s(0), s(0)) \leftarrow$
3. $\text{Fib}(s(s(x)), y + z) \leftarrow \text{Fib}(x, y), \text{Fib}(s(x), z)$.

Resolución

- La resolución es una regla de inferencia que permite demostrar teoremas por medio de *refutación*.
- Supóngase que se tiene un programa lógico Γ y se quiere demostrar que una fórmula atómica α se sigue de Γ.
- Entonces se incluye como premisa la meta $\leftarrow \alpha$ (es decir, $\neg\alpha$).
- Una demostración por resolución nos permitirá derivar fórmulas hasta que, en caso de éxito, lleguemos a una contradicción y entonces se afirmará que α se sigue de Γ.

Resolución proposicional

$$\text{Resolución} \quad \frac{\alpha \leftarrow \beta_1, \ldots, \beta_n \qquad \leftarrow \alpha, \gamma_1, \ldots, \gamma_k}{\leftarrow \beta_1, \ldots, \beta_n, \gamma_1, \ldots, \gamma_k}$$

Las dos premisas son *resolventes* y la conclusión es la *resolución*.
$\vdash_R$ denota los teoremas demostrables por resolución.
Por ejemplo, demostremos que $q \wedge r \Rightarrow p, q, r \vdash_R p$:

1	$p \leftarrow q, r$	Programa
2	$q \leftarrow$	Programa
3	$r \leftarrow$	Programa
4	$\leftarrow p$	Meta
5	$\leftarrow q, r$	Resolución 4, 1
6	$\leftarrow r$	Resolución 5, 2
7	$\leftarrow$	Resolución 6, 3

Versión para el cálculo de predicados

La regla de resolución para el cálculo de predicados es:

$$\frac{\alpha \leftarrow \beta_1, \ldots, \beta_n \qquad \leftarrow \alpha', \gamma_1, \ldots, \gamma_k}{\leftarrow \beta_{1\mu}, \ldots, \beta_{n\mu}, \gamma_{1\mu}, \ldots, \gamma_{k\mu}}$$

donde μ es un umg de α y α' y las variables que aparecen en la primera premisa no aparecen en la segunda.

Una instancia de la regla es:

$$\frac{P(x, f(x)) \leftarrow Q(x, y), R(y) \qquad \leftarrow P(a, z), R(z)}{\leftarrow Q(a, y), R(y), R(f(a))}$$

donde $\mu \equiv_{def} x, z := a, f(a)$.

Resolución y computación

¿Cómo se puede computar un resultado por medio de la resolución?
Supóngase que se tiene un programa lógico P y una meta

$$\leftarrow P_m^n(t_1, \ldots, t_n)$$

es decir,

$$\neg\exists x_1, \ldots, x_q \,.\, P_m^n(t_1, \ldots, t_n).$$

Si al aplicar Res k veces se llega a la cláusula vacía $\leftarrow$, se habrá demostrado que

$$\exists x_1, \ldots, x_q \,.\, P_m^n(t_1, \ldots, t_n)$$

y, al mismo tiempo, se tendrá un umg por cada aplicación de Res:

$$\mu_1, \ldots, \mu_k.$$

La aplicación de las sustituciones:

$$P_m^n(t_1, \ldots, t_n)_{\mu_1 \cdots \mu_k}$$

constituye el cómputo deseado.

Ejemplo I

1	$\text{Fact}(0, s(0)) \leftarrow$	Prog
2	$\text{Fact}(s(n), m \times s(n)) \leftarrow \text{Fact}(n, m)$	Prog
3	$\leftarrow \text{Fact}(s(s(s(0))), x)$	Meta
4	$\leftarrow \text{Fact}(s(s(0)), m)$	Res 2, 3 con $n, x := s(s(0)), m \times s(s(s(0)))$
5	$\leftarrow \text{Fact}(s(s(0)), m')$	$m := m'$ en 4
6	$\leftarrow \text{Fact}(s(0), m)$	Res 2, 5 con $n, m' := s(0), m \times s(s(0))$
7	$\leftarrow \text{Fact}(s(0), m')$	$m := m'$ en 6
8	$\leftarrow \text{Fact}(0, m)$	Res 2, 7 con $n, m' := 0, m \times s(0)$
9	$\leftarrow$	Res 1, 8 con $m := s(0)$

Ejemplo II

5 y 7 cumplen hacen que las variables de los resolventes sean disjuntas (m aparece en 3 y en 4 y 6). La sustitución final es:

$$\begin{aligned}
\mu_1 &= [n, x := s(s(0)), m \times s(s(s(0)))] \\
\mu_2 &= [m := m'] \\
\mu_3 &= [n, m' := s(0), m \times s(s(0))] \\
\mu_4 &= [m := m'] \\
\mu_5 &= [n, m' := 0, m \times s(0)] \\
\mu_6 &= [m := s(0)] \\
\mu_1\mu_2\mu_3\mu_4\mu_5\mu_6 &= \mu \\
\mu &= [x := s(0) \times s(0) \times s(s(0)) \times s(s(s(0)))] \\
\text{Fact}(s(s(s(0))), x)_\mu &= \text{Fact}(s(s(s(0))), s(0) \times s(0) \times s(s(0)) \times s(s(s(0)))) \\
&= \text{Fact}(3, 6)
\end{aligned}$$

La sustitución μ está simplificada para recoger sólo el valor de x.

Modelos de Herbrand

Definición

Sea P un programa lógico y sean $C_P = \{c_1, \ldots, c_n\}$ y $F_P = \{f_m^n\}$ el conjunto de constantes y el conjunto de símbolos de funciones que aparecen en P. La cerradura inductiva de C_P bajo F_P es C_P^+.
Constrúyase un conjunto U_P con un elemento por cada elemento en C_P^+.
Este conjunto es el universo de Herbrand *de P y sus elementos son los* términos básicos.
La base de Herbrand *es el conjunto de fórmulas atómicas con predicados que aparecen en P y cuyos argumentos son términos básicos.*
Una interpretación de Herbrand *es el subconjunto de la base de Herbrand satisfechos en la interpretación.*
Un modelo de Herbrand *es una interpretación que hace verdaderas todas las cláusulas de P.*
El modelo mínimo de Herbrand *es la intersección de todos los modelos de Herbrand. Éste es el modelo buscado para el programa P.*

Ejemplo

En el programa lógico para calcular la función factorial tenemos que

- El universo de Herbrand es

$$\{0, s(0), s(s(0)), \ldots, 0 \times 0, 0 \times s(0), s(0) \times s(0)\}.$$

- La base de Herbrand es

$$\{\text{Fact}(0, 0), \text{Fact}(0, s(0)), \ldots\}.$$

- Un modelo de Herbrand tiene a la relación

$$\{(0, s(0)), (s(0), s(0)), (s(s(0)), s(s(0))), \ldots\}$$

como interpretación de la relación Fact.

Negación

¿Cómo se puede demostrar, dado el programa para la función factorial anterior, que $\neg\text{Fact}(3,9)$? Hay al menos dos problemas:

(a) Dado que la resolución busca refutar una fórmula, para demostrar $\neg\text{Fact}(3,9)$ debemos partir de $\neg\neg\text{Fact}(3,9)$, que equivale a la cláusula $\text{Fact}(3,9) \leftarrow$. Obviamente, no se puede aplicar resolución con esta "meta".

(b) Aunque la intención del programa fue definir *precisamente* la función factorial (y no algo "parecido"), desde el punto de vista lógico $\text{Fact}(3,9)$ no es inconsistente con el programa: si se interpreta el predicado Fact como la relación

$$\{(n,m) \mid m = n!\} \cup \{(3,9)\}$$

se tiene un modelo del programa y de la fórmula $\text{Fact}(3,9)$ *a la vez*. Es decir,

$$\text{Fact}(0,s(0)) \leftarrow, \text{Fact}(s(n), m \times s(n)) \leftarrow \text{Fact}(n,m) \not\models \neg\text{Fact}(3,9).$$

¿Una regla más poderosa que resolución?

Para resolver el problema (a) se puede pensar en una regla de resolución más fuerte, digamos $\vdash_{R'}$, que nos permita demostrar teoremas de la forma

$$\Gamma \vdash_{R'} \neg\alpha.$$

Pero esto nos llevaría a demostrar

$$\text{Fact}(0,s(0)) \leftarrow, \text{Fact}(s(n), m \times s(n)) \leftarrow \text{Fact}(n,m) \vdash_{R'} \neg\text{Fact}(3,9)$$

y ya sabemos que esto no debería probarse, como nos dice (b). De hecho, el problema no se limita al caso de Fact.

Negación y corrección

Teorema

Sea $\vdash_{R'}$ una regla que es más fuerte que $\vdash_R$ en el sentido de que permite demostrar teoremas de la forma $\Gamma \vdash_R \neg\alpha$. Entonces, $\vdash_{R'}$ no es correcta, es decir, existen Γ y α tales que $\Gamma \vdash_R \neg\alpha$, pero se tiene que $\Gamma \not\models \neg\alpha$.

El ejemplo de $\neg\text{Fact}(3,9)$ es una instancia de este teorema.

Negación y consistencia

Un principio de solución es restringir las fórmulas negativas demostrables, es decir, dado un programa Γ y una fórmula $\neg\alpha$, sólo se podría demostrar

$$\Gamma \vdash_{R'} \neg\alpha$$

en caso de que $\Gamma \cup \{\neg\alpha\}$ sea consistente (en este caso se dirá que $\vdash'_R$ es correcta en sentido débil). Pero, ¿cómo calcular las fórmulas negativas consistentes con un programa?

Closed world assumption I

Una perspectiva es la llamada *suposición del mundo cerrado* (o CWA):

Definición

Sea Γ *un programa lógico y sea* α *una fórmula no negativa. La regla de derivación* CWA *es:*

$$\frac{\Gamma \nvdash_R \alpha}{\neg\alpha}.$$

El conjunto de consecuencias negativas de Γ *bajo la hipótesis del mundo cerrado es el conjunto*

$$cwa(\Gamma) = \{\neg\alpha \mid \Gamma \nvdash_R \alpha\}.$$

CWA nos dice que un programa lógico es un mundo "cerrado" sin intereferencia de otras cláusulas que no pertenecen al programa.

Closed world assumption II

Si el "hecho" α no es derivable del programa, y considerando que éste describe el "mundo" por completo, concluiremos que $\neg\alpha$.
El siguiente teorema es el fundamento de la suposición del mundo cerrado:

Teorema

Si Γ *es un programa y* $\neg\alpha \in cwa(\Gamma)$*, entonces* $\Gamma \cup \{\neg\alpha\}$ *es consistente.*

Desventajas

Desafortunadamente, el teorema siguiente plantea un problema (computacionalmente) insuperable:

Teorema

Dado un programa Γ *y una fórmula* α*, la pregunta ¿*$\Gamma \vdash_R \alpha$*? es indecidible, es decir, no existe un algoritmo que la conteste en todos los casos.*

Negation by failure

- Otra perspectiva, más modesta, es la llamada *negación por fracaso* o NF.
- Si para un programa *P* y una fórmula específica α es posible determinar de manera *computable* que $P \nvdash_R \alpha$, entonces se aceptará $\neg\alpha$.
- Desafortunadamente, los detalles técnicos se dejarán para un curso más avanzado.

Bases de datos relacionales

- Las bases de datos relacionales son un formalismo para representar y analizar bases de datos.
- En este enfoque, una base de datos está formada por un modelo con objetos y relaciones entre ellos y un subconjunto del cálculo de predicados que nos permite utilizar este modelo para extraer información.
- Una consulta a una base de datos es afín a la demostración de un teorema existencial.

Atributos y relaciones I

- El conjunto de atributos es $\mathcal{U}$. Los atributos pertenecen a este conjunto y los denotaremos con las letras $A, B, \ldots$
- Los dominios de los atributos se señalarán con $\Delta(A), \Delta(B), \ldots$. La unión de todos los dominios de atributos es Δ.
- El universo de atributos es $U \subseteq_{\text{fin}} \mathcal{U}$.
- Un esquema de relaciones es un conjunto $R \subseteq U$. Un esquema de base de datos D basado en U es un conjunto de esquemas de relaciones tal que
$$\bigcup_{R \in D} R = U.$$
- Sea $X \subseteq U$. Una X-ada v es un una mapeo $X \to \Delta$ tal que $X(A) \in \Delta(A)$.
- Sea v una X-ada y sea $Y \subseteq X$. La proyección de v en Y, denotada por $v[Y]$ es la restricción de v al subdomino Y.

Atributos y relaciones II

- Si R es un esquema de relación, una relación r es un conjunto de R-adas.
- Si D es un esquema de base de datos, una base de datos d es un conjunto de relaciones, una por cada esquema en D.
- $\alpha(r)$ es el esquema de relación de r y $\alpha(d)$ es el esquema de base de datos de d.

Ejemplos

- Sea $U \subseteq \mathcal{U}$ el conjunto
$$\{N = \text{nombre}, B = \text{nacionalidad}, A = \text{afiliación}, E = \text{edad}, F = \text{contribución}\},$$
con dominios definidos de acuerdo con la interpretación intuitiva de los atributos.
- Sean P y Q los siguientes esquemas de relación:
$$P = \{N, B, A\} \qquad Q = \{A, E, F\}$$
- Sea p la relación definida por esta tabla:

Nombre	Nacionalidad	Afiliación
Alan Turing	Británico	Cambridge
C.A.R Hoare	Británico	Oxford
E. Dijkstra	Holandés	U. de Texas en Austin

Lenguaje de búsquedas del álgebra relacional I

Sean $r, r_1, r_2, \ldots$ relaciones. Entonces

- $\Pi_X(r)$ es la *proyección de r en X*, con las siguientes propiedades:
  1. $X \subseteq \alpha(r)$ y $X = \alpha(\Pi_X(r))$;
  2. $\Pi_X(r) = \{v[X] \mid v \in r\}$.
- $r_1 \bowtie r_2$ es la *conjunción natural* definida por las siguientes propiedades:
  1. $\alpha(r_1 \bowtie r_2) = \alpha(r_1) \cup \alpha(r_2)$;
  2. $r_1 \bowtie r_2 = \{v \mid v$ es una $\alpha(r_1) \cup \alpha(r_2) -$ *ada* tal que $v[\alpha(r_1)] \in r_1, v[\alpha(r_2)] \in r_2\}$.
- $r_1 \cup r_2$ es la *unión* con las propiedades:
  1. $\alpha(r_1) = \alpha(r_2)$ y $\alpha(r_1 \cup r_2) = \alpha(r_1)$;
  2. $r_1 \cup r_2 = \{v \mid v \in r_1$ o bien $v \in r_2\}$.
- $r_1 - r_2$ es la *diferencia* con las propiedades:
  1. $\alpha(r_1) = \alpha(r_2)$ y $\alpha(r_1 - r_2) = \alpha(r_1)$;
  2. $r_1 - r_2 = \{v \mid v \in r_1$ y $v \notin r_2\}$.

Lenguaje de búsquedas del álgebra relacional II

- $\varsigma_{A=B}(r)$ es la *selección en r por A = B*:
  1. $A, B \in \alpha(r)$ y $\alpha(\varsigma_{A=B}(r)) = \alpha(r)$;
  2. $\varsigma_{A=B}(r) = \{v \mid v \in r$ y $v[A] = v[B]\}$.
- $\varrho_{B|A}(r)$ es el *cambio de nombre en r de A a B*:
  1. $A \in \alpha(r)$, $B \notin \alpha(r)$ y $\alpha(\varrho_{B|A}(r)) = (\alpha(r) - \{A\}) \cup B$;
  2. $\varrho_{B|A}(r) = \{v \mid \exists v' \in r \,.\, v[B] = v'[A]$ y $\forall C \neq B \,.\, v[C] = v'[C]\}$.

Lenguaje para el álgebra relacional I

- Sea α una fórmula del cálculo de predicados ampliado para incluir las relaciones definidas en las láminas anteriores. Supongamos que α tiene n variables libres. Entonces

$$\{R, x_1, \ldots, x_n : \alpha(x_1, \ldots, x_n)\}$$

  es una expresión del cálculo relacional.
- Sea d una base de datos y sea δ la unión de los dominios de los valores de los atributos que aparecen en d (como d es finita, también lo es δ). El par $\langle d, \delta \rangle$ puede constituir la base de una interpretación de una expresión $\{R, x_1, \ldots, x_n : \alpha(x_1, \ldots, x_n)\}$ de la siguiente manera:
  - Los símbolos de relación que aparecen en α son interpretados por los esquemas de relaciones $\alpha(d)$.
  - δ forma el universo de interpretación para las variables $x_1, \ldots, x_n$

Lenguaje para el álgebra relacional II

- La interpretación de $\{R, x_1, \ldots, x_n : \alpha(x_1, \ldots, x_n)\}$ consiste en las n-adas en d tales que al sustituir las variables $x_1, \ldots, x_n$ por valores en δ se obtienen n-adas tales que existen r-adas en d de las cuales las n-adas sean proyecciones restringidas a α.

Ejemplo

Consideremos la base de datos de la lámina 5. Sea $X \subseteq P$, con $X = \{N; B\}$. Sea $\alpha = \exists x \,.\, R_1(x, y)$, donde $R_1 = \Pi_X(p)$. Entonces

$$\{R\ y : \exists x \,.\, R_1(x, y)\} = \{\text{británico}, \text{holandés}\},\ \text{pues}$$

- P es un esquema de relación con los atributos N, B, A;
- la relación p está basada en este esquema;
- la proyección $\Pi_X(p)$ reduce las p-adas en p a sus proyecciones en los dos únicos atributos de X, a saber N y B;
- la relación interpretada por $\Pi_X(p)$ tiene dos argumentos
- el cuantificador en α captura el primero de estos y deja libre sólo el segundo
- En resumen, nos preguntamos por los valores b de B tales que existe un algún valor a del atributo N tal que el par (b, a) está en $\Pi_X(p)$.