Bases de Datos Avanzadas

Héctor Jiménez Salazar
hgimenezs@gmail.com

enero de 2007

No es necesario argumentar la importancia en la actualidad de las Bases de Datos (BD). Ya a principios de 1990, cada 20 meses se duplicaba el volumen de información, había más de cinco millones de bases de datos en el mundo y cada día crecía en 10^{15} bytes la información registrada en los sistemas de cómputo (contando imágenes satelitales). Tal vez esta sea el área de la computación con mayor impacto en las aplicaciones por su volumen. Ahora, además disponemos de internet, que no se divorcia de las otras aplicaciones más bien hace uso de ellas, una red con crecimiento exponencial cuya información es tanto estructurada (BD) como no estructurada (texto, audio, imágenes, etc.).
La forma tradicional de emplear las BD ha empezado a ser substituida por otros enfoques que ganan atracción; por ejemplo las BD orientadas a objetos, las deductivas o el llamado *data warehousing*. La tarea, no solamente de diseñar y analizar BD, sino también de elegir el tipo más adecuado, adaptar uno a cierta aplicación, o finalmente, evolucionar a una nueva es una tarea eminentemente de las Ciencias de la Computación.
Estas notas presentan un brevísimo ejemplo, a la vez clásico, sobre cómo puede emplearse la metodología seguida en el desarrollo de los sistemas formales para fundamentar los sistemas manejadores de BD. Se muestra entonces por qué las operaciones que se ofrecen por los sistemas son de esa forma y no de otra, y también, en algunos casos, la conveniencia del uso de ciertas técnicas de implantación de estos sistemas.

1. El modelo relacional

El modelo relacional fue introducido en 1970 por E.F. Codd. Con el paso del tiempo ganó popularidad respecto a los otros dos modelos: jerárquico y de red.
Se considerarán, en lo sucesivo, conjuntos de diverso tipo y operaciones entre ellos. Haremos algunas definiciones para precisar el modelo.

Definición 1 Sean $D_1, \ldots, D_n$ conjuntos, también llamados *dominios*, y $D = D_1 \times \ldots \times D_n$. Una *relación* sobre D es cualquier subconjunto R de D. Para R definida como antes se tiene que su *aridad* es n.

Definición 2 Los elementos de una relación se llaman *tuplas* y se denotan $< d_1, \ldots, d_n >$, donde $d_i \in D_i$, $1 \leq i \leq n$, y $R \subset D$.

Observación 1

a) Aunque se ha dicho, recalcamos que una relación es un conjunto. Asimismo, existe la relación nula o vacía: $\emptyset$.

b) Supondremos que los dominios son finitos y por tanto nuestra relación también lo será; a tal número lo llamaremos *cardinalidad*: $\#R = n$.

c) El orden en que concibamos las tuplas es irrelevante, aunque para ejemplificar se emplean *tablas* en las que los nombres de las columnas se llaman *atributos*.

d) El conjunto ordenado de los atributos de una relación R es conocido como el *esquema* de R, Esq(R). Alternativamente se hjará referencia a los atributos por su nombre o, convenido un orden, por su posición en el esquema (número de columna en la tabla).

Definición 3 Una *llave* para una relación R es un subconjunto K de los atributos de R y cuyos valores son únicos en R, esto es, tales atributos pueden ser usados para identificar de manera única todas y cada una de las tuplas de R. Una llave debe ser *minimal*, así no contendrá atributos que no son necesarios para la identificación única.

Observación 2

a) Dada una realación siempre existirá al menos una llave, pues trivialmente tomando como subconjunto todos los atributos se tendrá que hay identificación única; si no es así habrá forma de eliminar algún atributo.

b) Es posible que una relación tenga múltiples llaves.

c) La propiedad de que un subconjunto de atributos sea llave está en función de la interpretación de los dominios, y deberá cumplirse para todos los posibles valores que tomen las tuplas, no solamente para el caso específico de una relación en D.

Ejemplo 1 Considere la siguiente tabla para identificar algunos de los elementos antes descritos:

cardex		
Estudiante	Curso	Calif
miguel	autómatas	10
carlos	estadística	9
martín	programación	10
carlos	lenguajes	9

Los atributos son Estudiante, Curso y Calif, y sus dominios son, en el primer caso, de una cardinalidad muy grande, tal vez unos cientos de miles; en el segundo habrá de ser menor de 100 y el último de cardinalidad 10. Tomando $\#D_1 = 100,000$, $\#D_2 = 100$ y $\#D_3 = 10$ tenemos que $\#D = 10^8$. Así **cardex** tiene una porción muy pequeña de D.
El esquema de **cardex** es cardex(Estudiante, Curso, Calif). Vemos además que (Estudiante) no es una llave.

Ejercicio 1 Determine todas las llaves de **cardex**. Si se modifica el esquema como cardex(Estudiante, Curso, Año, Calif), ¿cuáles serían llaves de **cardex**?

1.1. Álgebra relacional

Una base de datos (BD) será concebida como varias relaciones con esquemas no ajenos, y en dominios relacionados dentro del contexto de la aplicación. Hay muchos lenguajes para operar con las relaciones y para recuperar datos a partir de consultas. Una *consulta* expresada en un *lenguaje de consulta* representa una parte de los datos contenidos en la BD. Dos son los formalismos para expresar las consultas: el cálculo relacional y el álgebra relacional. En el primer caso se trata de expresar la consulta como el resultado de la intesión que define un conjunto (esto es, las tuplas que satisfacen a una fórmula de la lógica de primer orden), y en el segundo como el resultado de operaciones sobre conjuntos. Veremos con detalle el álgebra relacional.
Una consulta será una expresión algebráica. Las operaciones empleadas en esta álgebra son en su mayoría conocidas en los conjuntos. Su definición es como sigue.

Definición 4 Las operaciones primitivas del álgebra relacional son:

Unión: Siempre que dos relaciones, R y S, tengan los mismos esquemas puede aplicarse la unión: $R \cup S$ se define como el conjunto de tuplas que pertenecen a la unión de R y S, en el sentido de conjuntos, y el esquema será el mismo que el de las relaciones originales.

Diferencia: Al igual que en el caso anterior, las dos relaciones deben cumplir que $\text{Esq}(R) = \text{Esq}(S)$, y también la definición de $R - S$ es en términos de los conjuntos que representan R y S: eliminar de R las tuplas de S.

Producto cartesiano: Si R tiene aridad r y S tiene aridad s entonces $R \times S$ tendrá aridad $r + s$ y $\text{Esq}(R \times S) = \text{Esq}(R) \| \text{Esq}(S)$, i.e. su esquema se forma por la concatenación de los esquemas de R y S. $R \times S$ se interpreta en el sentido de conjuntos: todas las combinaciones de tuplas de R con tuplas de S.

Selección: Sea $\mathcal{F}$ una fbc, que llamaremos predicado de selección. $\mathcal{F}$ se limita a usar predicados simples (comparación entre valores de atributos y valores constantes). El resultado de la selección $\sigma_{\mathcal{F}}$ sobre la relación R, $\sigma_{\mathcal{F}} R$, es un subconjunto de R que contiene las tuplas donde $\mathcal{F}$ se satisface.

Proyección: Sea R de aridad n. Supongamos que $i_1, \ldots, i_k$, $k \leq n$, son nombres de atributos o índices de posición en el esquema de R. Sea D_i el dominio del atributo i. Entonces, el resultado de la proyección $\Pi_{i_1,\ldots,i_k} R$ es una relación de aridad k, teniendo como atributos $i_1, \ldots, i_k$. Dicha relación será formada por las tuplas de R eliminando los valores de los atributos que no pertenezcan a $i_1, \ldots, i_k$, i.e., $\Pi_{i_1,\ldots,i_k} R \subset D_{i_1} \times \ldots D_{i_k}$.

Observación 3

a) El conjunto de primitivas del álgebra relacional es *relacionalmente completo*, esto es, todas las consultas expresadas por el cálculo relacional pueden ser también expresadas con estas primitivas.

b) Notemos que al aplicar la proyección a una relación la cardinalidad puede disminuir en el resultado, debido a que las tuplas repetidas se eliminan por tratarse de un conjunto.

Ejercicio 2 Sean las relaciones dadas por las tablas:

cursos-01	
Profesor	Curso
josé-luis	estadística
juan	lógica
beatríz	bd

cursos-02	
Profesor	Curso
mario	diseño
juan	lógica
hilda	bd
héctor	lenguajes

y la relación **cardex** dada en el ejemplo 1 con dos nuevas tuplas añadidas: <maría lógica 8> y <ángela bd 10>. ¿Cuáles son las relaciones que resultan de las operaciones $\Pi_{\text{Estudiante,Calif}}$**cardex**, $\sigma_{\text{Calif}=10}$**cardex**, **cursos-01**∪**cursos-02**, **cursos-01** - **cursos-02** y **cursos-01**×**cardex**.

Observación 4 A partir de las primitivas algunas composiciones resultan muy útiles. Tales composiciones de operaciones son:

Junta: Sea $\mathcal{F}$ una fbc simple, la junta de $\mathcal{F}$ aplicada a dos relaciones R y S está definida como:

$$R \bowtie_{\mathcal{F}} S = \sigma_{\mathcal{F}}(R \times S),$$

esto es, la selección de algunas tuplas (las que satisfacen a $\mathcal{F}$) del producto.

Junta natural: Corresponde al caso de una junta en la que la fbc es una conjunción de igualdades de los valores de los atributos comunes en las relaciones. Por tal motivo la fbc siempre queda definida de manera implícita y se evita: $R \bowtie S$.

Semijunta: Para una fbc simple y dos relaciones, R y S, la semijunta es:

$$R \ltimes_{\mathcal{F}} S = \Pi_{\text{Esq}(R)}(R \bowtie_{\mathcal{F}} S).$$

Así, la relación resultante de la semijunta es un sunconjunto del primer argumento. En forma semejante se define la semijunta natural.

Observación 5 El propósito de almacenar las BD es poder hacer consultas. Por tanto, el objetivo del álgebra relacional es representar consultas. Para lograr lo anterior el usuario de un sistema manejador de BD debe estar familiarizado con el contexto de la aplicación. En el siguiente ejemplo se concreta esta observación.

Ejemplo 2 Considérense las relaciones definidas en el ejercicio 2. Como podemos ver, **cursos-01** y **cursos-02** representan cursos impartidos por profesor, en dos periodos, y **cardex** es el registro de calificaciones de los estudiantes para cada materia aprobada. Entonces, la operación $\Pi_{1,2}$**cardex** representa una muestra de calificaciones por estudiante. Notemos que, en el contexto que se han definido las relaciones, el resultado de tal operación es poco útil; es solamente como se dijo una muestra de calificaciones. En tanto que $\sigma_{\text{Calif}=10}$**cardex** responde a la pregunta "cuáles estudiantes y en qué materia obtuvieron calificación 10". Asimismo, **cursos-01**∪**cursos-02** representa la relación de todos los cursos impartidos. Un tanto semejante a $\Pi_{1,2}$**cardex** es es resultado de **cursos-01** - **cursos-02** y **cursos-01**×**cardex**; es decir no representan resultados a preguntas comunes en el dominio que se está trabajando.

Ejercicio 3 Con qué operaciones del álgebra relacional respondemos a la pregunta:

a) ¿cuáles fueron los alumnos que aprobaron al menos una materia en el segundo periodo de cursos?

b) ¿cuáles son las calificaciones de los estudiantes de **curso-01**?

Ejercicio 4 Diga por qué se cumple la igualdad:

$$\Pi_i \sigma_{j=c}(R \bowtie S) = \Pi_i(R \bowtie \sigma_{j=c} S).$$

¿Cuál miembro de la igualdad es más eficiente para hacer el cálculo?

Observación 6 En los lenguajes de consulta a BD, no es apropiado el empleo del álgebra relacional. Más bien se ha definido un lenguaje genérico llamado "lenguaje de consulta estructurada" o *SQL* (siglas de "Structured Query Language"). En dicho lenguaje se identifican tres partes principales de toda consulta: qué forma debe tener nuestro resultado, por ejemplo una lista de estudiantes con su calificación; a partir de qué relaciones se extrae dicha información; y cuál es la cualidad que tienen las tuplas que deseamos. Estas tres partes forman el bloque de una consulta general en SQL:

```
SELECT   < lista − de − atributos >
FROM     < relaciones >
WHERE    < predicado >
```

Por ejemplo, la pregunta cuáles estudiantes obtuvieron calificación diez queda expresada en SQL de la siguiente forma:

```
SELECT   Estudiante
FROM     cardex
WHERE    Calif=10
```

Como se ha definido, el bloque SQL equivale a una proyección sobre una selección: la parte `SELECT` corresponde a la proyección, la parte `FROM` es el argumento de la primera operación relacional lo cual puede considerar un producto cartesiano de relaciones, y la parte `WHERE` representa el predicado de una selección sobre el segundo argumento.
La potencialidad de SQL radica en que el bloque de consulta es recursivo:

```
SELECT   < lista − de − atributos >
FROM     < relaciones >
WHERE    < atributo > IN
         SELECT < lista − de − atributos >
         FROM < relaciones >
         WHERE < predicado >
```

Ejercicio 5 Traduzca a SQL las consultas expresadas en álgebra relacional obtenidas en el ejercicio 3.

Definición 5 Una *vista* es una realción que no se almacena explícitamente en la BD pero que se calcula en el proceso de atención a una consulta.

Observación 7 En el álgebra relacional las vistas son simplemente subexpresiones o variables que almacenan resultados parciales. Por ejemplo, en

$$\Pi_1((\textbf{cursos-01} \cup \textbf{cursos-02}) \bowtie \textbf{cardex}),$$

o

$$v = \textbf{cursos-01} \cup \textbf{cursos-02},$$

y

$$\Pi_1(v \bowtie \textbf{cardex}),$$

donde **cursos-01** $\cup$ **cursos-02** o v es una vista.

1.2. Prolog

Surgido en la década de los 70, *Prolog* (de *Programming in Logic*) es un lenguaje declarativo, como todos los lenguajes de programación lógica. En él se expresa *qué* se cumple, y no *cómo* se calcula. En general, para describir programas Prolog hablaremos de cláusulas, como se define enseguida.

Definición 6 Consideremos un alfabeto de dígitos, letras mayúsculas y minúsculas, y símbolos especiales. Definimos las siguientes entidades:

átomos, cadenas alfanuméricas iniciadas por una letra minúscula;

variables, cadenas alfanuméricas iniciadas por una letra mayúscula, incluyendo la variable "anónima" _;

functores, compuestos de su parte nombre y aridad, por ejemplo todos los predicados son functores;

término estructurado, es una variable o un átomo y, recursivamente, cualquier functor aplicado a términos;

literal, un término en el que su functor más externo es un predicado, son literales negativas aquellas que las precede el símbolo de negación "$\sim$";

cláusula, una colección de literales, las hay de tres tipos:

1. *regla:* hay una literal positiva llamada *cabeza* y el resto forman el *cuerpo*, es denotada por:

   $$cabeza(t_1, \ldots) : -c_1(r_1, \ldots), \ldots c_k(s_1, \ldots),$$

   cuya lectura es "si $c_1, \ldots, c_k$ entonces *cabeza*";
2. *hecho:* contiene solamente una literal (no tiene cuerpo), es una aserción; y
3. *meta:* es una cláusula sin cabeza y constituye una consulta;

programa, es una colección de cláusulas.

La *máquina de inferencia*, implícita en el intérprete de Prolog, realiza los siguientes pasos frente a una consulta $? - q.$:

1. Para cada literal q_i de q: unifica, con la cabeza h_j de una cláusula del programa, instanciando los términos que ocurran en ambas, q_i y h_j.
2. Para cada literal que forma el cuerpo de h_j, $c_1, \ldots, c_k$, con sus variables instanciadas por el paso anterior, la trata como literal de una consulta, recursivamente.
3. Resueltas las literales de una cláusula, continúa con la siguiente literal de la consulta, q_{i+1}, hasta agotarlas, usando las variables instanciadas hasta ese momento.

Observación 8 Debe aclararse que el proceso de solución de una consulta es complementado considerando lo siguiente:

a) Cuando la literal de una consulta se unifica con un hecho termina la recursión aludida en el punto 2 anterior.

b) Es posible que existan varias posibilidades de unificación para una literal. El criterio empleado normalmente es elegir la que aparezca en primer lugar.

c) Si la literal, en curso de solución, no se unifica el programa falla; lo cual significa que no puede dar respuesta a la consulta por la vía que tomó.

d) El evento *falla* es alternado con *retroceso* (*backtracking*) que significa deshacer todas las instanciaciones de la última elección y tratar de unificar de otra forma, sucesivamente en retroceso.

e) Cuando ninguna de las posibles formas permite unificar, el programa falla definitivamente y termina anunciando: `No`.

f) En caso de éxito, el resultado se forma por los valores de las variables que se instanciaron en la consulta.

En Prolog la principal estructura de datos es la *lista*. Ésta se forma por una secuencia de términos, encerrada por "[" y "]". Así, [] representa la lista vacía. Una lista puede verse como un operador, i.e. como un término. Se emplea el constructor "|"para separar los primeros elementos. Por ejemplo, en [`Primero`|`Demas`] el primer elemento lo contiene la variable `Primero`, y el resto lo contiene la variable `Demas`. Para precisar lo anterior veamos un ejemplo, en primer lugar, del proceso de instanciación.

Ejemplo 3 Considérense los siguientes pares de términos en el proceso de unificación y el resultado de cada instanciación:

Término 1	**Término 2**	**Instanciación**
[a]	[V]	V/a
a	[V]	*falla*
[a]	X	X/[a]
[a, b]	[Y]	*falla*
[a, b]	[Z|W]	Z/a,W/[b]
[a,[b]]	[Z|W]	Z/a,W/[[b]]
[a, b]	[Q, R]	Q/a,R/b
A	[a,b]	A/[a,b]
[_|D]	[a,b]	D/[b]
[a]	[Uno|Dos]	Uno/a,Dos/[]
V1	Gato	V1/Gato
p(a,b)	p(a,c)	*falla*
p(a,b)	p(a,b)	
p(a,b)	p(a,C)	C/b
qu([a,b],[C|D])	qu(D,R)	D/[a,b],R/[C,a,b]

El conjunto de parejas de la tercera columna se llama *substitución*. Las substituciones tienen la propiedad de que aplicadas a ambos términos conducen a la identidad de ellos. Aparece, además, el diagnóstico *falla*, cuando no es posible encontrar una substitución que permita el empatamiento de los términos. En particular, existe la substitución vacía (ver uno de los casos de la tabla) que indica que los términos originales son iguales.

Observación 9 Analicemos el siguiente programa que implementa la membresía a un conjunto, donde el primer argumento se toma como el elemento del que se desea saber su pertenencia al segundo (conjunto manejado como lista):

```
miembro(X,[X|_]).
miembro(X,[_|Y]) : −miembro(X,Y).
```

El programa se lee "si el elemento ocurre en la lista, termina con éxito, en caso contrario, delega al cuerpo de la segunda cláusula la pregunta: el elemento buscado pertenence al resto de la lista".

Como podemos ver, se ensaya con la primera definición del predicado `miembro/2`, y si el elemento está al inicio de la lista se termina con valor "verdadero" (V), en caso contrario, hace retroceso para ensayar con la segunda definición del predicado, en este caso unifica el resto de la lista con `Y` y en esa parte repite la operación. Una definición funcional ayuda a aclarar lo anterior:

$$miembro(A, L) = \begin{cases} \text{V} & \text{si } L = [B|C] \text{ y } A = B, \\ miembro(A, C) & \text{si } L = [B|C] \text{ y } A \neq B. \end{cases}$$

Eventualmente, si el elemento no aparece en la lista, ésta quedará vacía y la consulta `miembro(X,[])` fallará, pues con ningún caso de `miembro/2` unifica.

Ejemplo 4 Se muestra a continuación tres programas que implementan respectivamente: la eliminación de la primera ocurrencia de un elemento en una lista, la eliminación de todas las ocurrencias de un elemento en una lista, y el factorial de un número. Hagamos, en primer lugar, las definiciones funcionales de los programas:

$$elim1(A, L) = \begin{cases} [] & \text{si } L = [], \\ C & \text{si } L = [B|C] \text{ y } A = B, \\ [B|elim1(A, C)] & \text{si } L = [B|C] \text{ y } A \neq B. \end{cases}$$

$$elimt(A, L) = \begin{cases} [] & \text{si } L = [], \\ elimt(A, C) & \text{si } L = [B|C] \text{ y } A = B, \\ [B|elimt(A, C)] & \text{si } L = [B|C] \text{ y } A \neq B. \end{cases}$$

$$fact(n) = \begin{cases} 1, & \text{si } n = 1, \\ n \times fact(n-1), & \text{si } n > 1. \end{cases}$$

Claramente $elim1(a, [a, b, a]) \rightsquigarrow [b, a]$, ya que se cumple la segunda condición de la definición (que el primer elemento de la lista sea igual al primer argumento) y, por tanto, el resultado es el resto del segundo argumento. En el caso de $elim1(b, [a, c, d]) \rightsquigarrow [a, c, d]$, el cálculo es mayor porque fallan las dos primeras condiciones y se forma una nueva meta, implícita en el cálculo que define la tercera condición: $[a|elim1(b, [c, d])]$. Así, repetidamente se realiza el tercer caso de la definición y termina con el primero: $elim1(a, [b, c, d]) \rightsquigarrow [a|elim1(b, [c, d])] \rightsquigarrow [a|[c|elim1(b, [d])]] \rightsquigarrow [a|[c|[d|elim1(b, [])]]] \rightsquigarrow [a|[c|[d|[]]]]$, que es igual a $[a, c, d]$. Puede comprobarse similarmente el comportamiento de las demás definiciones funcionales. Tenemos, entonces, las siguientes cláusulas:

```
elim1(X,[],[]).
elim1(X,[X|Y],Y).
elim1(X,[Y|Z],[Y|W]):- elim1(X,Z,W).

elimt(X,[],[]).
elimt(X,[X|Y],Z):- elimt(X,Y,Z).
elimt(X,[Y|Z],[Y|W]):- elimt(X,Z,W).

fact(1,1).
fact(X,Y):- X1 = X-1, fact(X1,Y1), Y = X * Y1.
```

Observación 10 Es importante notar que el predicado `fact/2` tendría problemas si se consulta `?- fact(0,X)`, puesto que intentaría resolver, por la segunda cláusula: `fact(-1,X'), fact(-2,X''),` etc. sin terminar. Conviene entonces redefinir las cláusulas como:

```
fact(X,1):- X <= 1.
fact(X,Y):- X >1, X1 = X-1, fact(X1,Y1), Y = X * Y1.
```

Con lo cual se atiende el caso presentado. Aún, podemos agilizar la parte que verifica la condición del argumento:

```
fact(X,1):- X <= 1, !.
fact(X,Y):- X1 = X-1, fact(X1,Y1), Y = X * Y1.
```

Ya que siempre intenta resolver con la primera cláusula, al ser falsa la condición (`X <=1`) ya no repite la verificación (`X >1`) en la segunda cláusula. El operador *cut*, "!", evita buscar otra forma de resolver la literal `fact/2` en ese punto; cuando el primer argumento es menor o igual a uno.

Ejercicio 6 Implemente las cláusulas correspondientes a las siguientes definiciones:

a) Dado un par de elementos y una lista, el resultado es la lista correspondiente a subsutituir cada ocurrencia del primer elemento por el segundo en la lista original: $sust(a, b, [a, b, c, a, d, a]) \rightsquigarrow [b, b, c, b, d, b]$, $sust(a, b, [e, b, c, f, d, g]) \rightsquigarrow [e, b, c, f, d, g]$, etc.

b) Trátense dos conjuntos como listas y obténgase la intersección de ellos: $inter([a, b, c], [d, e, f]) \rightsquigarrow []$, $inter([a, b, c, d], [f, d, a, h]) \rightsquigarrow [a, d]$, etc. (*Indicación:* use el predicado `miembro`.)

c) Dada una lista el resultado es la reversa de ella: $rev([a]) \rightsquigarrow [a]$, $rev([a, b, c]) \rightsquigarrow [c, b, a]$, etc.

1.3. Prolog como lenguaje de consulta

Considérense las siguientes relaciones:

padre	
Padre	Hijo
juan	javier
javier	margot
margot	ana
juan	antonio
antonio	alberto
antonio	guadalupe
maria	javier
clara	alberto
guadalupe	pablo

persona		
Nombre	Edad	Sexo
pablo	7	m
juan	78	m
javier	55	m
margot	32	f
ana	4	f
antonio	58	m
alberto	24	m
guadalupe	27	f
maria	75	f
clara	45	f

Éstas pueden efectivamente representarse con hechos en Prolog:

```
padre(juan, javier).
padre(javier, margot).
padre(margot, ana).
padre(juan, antonio).
...
persona(pablo, 7, m).
persona(juan, 78, m).
persona(javier, 55, m).
persona(margot, 32, f).
...
```

Y hacer consultas como "¿quién es el padre de guadalupe?":

```
?- padre(X, guadalupe).
X = antonio
```

Ejercicio 7 Formule una consulta Prolog para responder lo siguiente:

a) ¿Quién es la madre de alberto?

b) ¿Quiénes son los nietos de juan?

c) ¿Quién es el hijo más pequeño de antonio?

Como podrá comprobarse, todas las operaciones relacionales pueden ser efectuadas mediante consultas Prolog. Lo importante de esta representación es la posibilidad de expresar *bases de datos intensionales* (BDI), es decir especificar relaciones que no están almacenadas explícitamente. Por ejemplo la relación "hermano" y la relación "primo", que se basan en `padre/2`:

```
hermano(X,Y) :- padre(Z,X), padre(Z,Y), not(X==Y).
primo(X,Y) :- padre(X1,X), padre(Y1,Y), hermano(X1,Y1).
primo(X,Y) :- padre(X1,X), padre(Y1,Y), primo(X1,Y1).
```

Efectivamente estas relaciones, en el lenguaje relacional, son las vistas. Así, además de BDI, hablamos también de *bases de datos*

extensionales, (BDE), aquellas definidas explícitamente, almacenadas físicamente (en nuestro ejemplo: `padre` y `persona`), y *bases de datos virtuales* (BDV), las que se deducen de las BDI. En nuestro ejemplo, hemos definido dos BDV, de las cuales mostraremos la segunda:

primo	
Pariente1	Pariente2
margot	alberto
margot	guadalupe
ana	pablo

Este resultado se obtiene con la siguiente consulta, previamente definidas las BDE y las BDI:

```
?- qprimo.
```

donde `qprimo` se define como:

```
qprimo:- primo(X,Y), writeq(X), write(' '), writeq(Y), nl,
fail.
```

Al ser `fail` un predicado constante que representa falla o el valor de certeza "falso" (F), instancia por primera vez `X` y `Y` y, cuando obtiene falla, hace retroceso para encontrar nuevos valores de `X` y `Y`, sucesivamente, hasta agotar los casos que satisfacen `primo/2`.

Ejercicio 8 Introduzca y pruebe algunas metas en Prolog[1], como la obtención de todos los primos.

Es importante remarcar que las tuplas de las BDV no se calculan, de antemano, más bien sólo al momento de requerirlas.

Recordemos que un manejador de BD considera la validación de las condiciones que deben satisfacer las tuplas almacenadas. Estas condiciones, *restricciones de integridad*, se verifican en el momento de actualizar una relación. En Prolog podemos expresar las restricciones como cláusulas. En el ejemplo que estamos tratando podemos declarar, entre otras, las siguientes restricciones:

```
bd-incorrecta(padre,X) :- padre(X,X).
bd-incorrecta(madre,Z) :- madre(X,Z), madre(Y,Z), not(X==Y).
bd-incorrecta(per-padre,X) :- padre(X,Y), not(persona(X,_,_)).
```

En el primer caso, se enuncia que ninguna persona puede ser padre de sí misma, tampoco alguien puede tener dos madres y, por último en nuestro ejemplo, todo padre debe ser una persona. Al ejecutar el predicado:

```
?- bd-incorrecta(X,Y), writeq(X), write(' '), writeq(Y).
```

éste devolverá el caso incorrecto, si es que existe, y el nombre de una persona involucrado en la tupla de la relación aludida. En caso de no haber violación de integridad en la BD, retornará con falla.

Veamos ahora una base de datos intensional que no es posible definir en los manejadores tradicionales de bases de datos:

```
ancestro(X,Y) :- padre(X,Y).
ancestro(X,Y) :- padre(X,Z), ancestro(Z,Y).
```

Esta relación corresponde a todos los ancestros de cada una de las personas de la base de datos. En la tabla se muestran algunas tuplas de esta relación:

ancestro	
Ancestro	Descendiente
juan	javier
javier	margot
...	...
guadalupe	pablo
juan	margot
maria	margot
javier	ana
...	...
maria	ana
juan	pablo

En suma, el uso de Prolog para representar relaciones y hacer consultas enriquece a los lenguajes de consulta a bases de datos, debido a la recursión, y el control que se gana en la declaración de los predicados. Por un lado, Prolog es un lenguaje que permite el diseño centrado en la lógica, evita detalles sobre los procedimiento, ofrece claridad, y un nivel mayor de expresividad. Además, de proveer una forma más flexible para hacer consultas a BD, la definición de bases de datos intensionales, ahorra espacio en memoria y nos ofrece la posibilidad de realizar deducciones. Es posible, también, establecer restricciones de integridad con más facilidad.

Concluimos la presente sección resumiendo los conceptos semejantes en BD y Prolog:

Bases de Datos	**Programación Lógica**
Relación	Predicado
Atributo	Argumento de predicado
Tupla	Hecho
Vista	Regla
Consulta	Meta
Restricción	Meta (con valor V)

Ejercicio 9

a) Traduzca las consultas del ejercicio 3 a Prolog, considerando las BDE correspondientes.

b) Enuncie cinco cláusulas para ser usadas como restricciones de integridad en la BD que refiere el punto anterior.

2. Sintaxis y semántica de Datalog

Como fue comentado al inicio, tenemos como fin establecer un vínculo entre la definición formal de un sistema manejador de Bases de Datos y las operaciones que realizamos sobre datos concretos. De esta forma aseguraremos que las operaciones, tal como se han definido en el sistema, realizan lo que esperamos. Esto es, emplearemos un sistema de la lógica clásica para describir las operaciones de un manejador de Bases de Datos y exploraremos sus propiedades, entre las cuales nos interesa la solidez y la adecuación.

Ya fue visto que Prolog puede ser utilizado como lenguaje de consulta, y las ventajas que se tienen al fusionar los enfoques de Bases de Datos Relacionales con la Programación Lógica. Estos hechos han motivado la creación de lenguajes de programación lógica orientados a manejar Bases de Datos. Presentaremos *Datalog* que es un ejemplo de este tipo de lenguajes.

La Programación Lógica puede verse como un sistema formal en el que se trata de seguir lo más fielmente las definiciones de la lógica clásica. En esta sección iremos precisando las particularidades de la Programación Lógica (donde Prolog es representante) en general y de Datalog en particular y, en la siguiente sección, atenderemos los propósitos planteados inicialmente.

Definición 7 Datalog considera los siguientes conjuntos de símbolos:

[1]Puede usarse SWI-Prolog: `http://www.swi-prolog.org/`.

Variables, *Var.* Son cadenas finitas de caracteres alfanuméricos empezando con una letra mayúscula.

Constantes, *Const.* Se trata de cadenas finitas de caracteres alfanuméricos, las cuales son únicamente formadas por dígitos decimales o inician con una letra minúscula seguida de dígitos o letras minúsculas.

Predicados, *Pred.* Es el conjunto finito de todas las cadenas de caracteres alfanuméricos iniciando con una letra minúscula. La distinción entre una cadena de Const o Pred se hará clara por el contexto en que aparezca.

Observación 11 Algunas diferencias de Datalog con Prolog son:

a) Prolog herada de la lógica clásica un conjunto de símbolos funcionales, sin embargo en Datalog no son necesarios.

b) De igual forma, en lógica clásica como en Prolog es posible que un predicado tenga aridad cero. En Datalog no existen predicados con aridad cero.

c) Al no existir funciones en Datalog, no pueden construirse términos complejos.

Las limitaciones de Datalog no restringen la formalización que deseamos hacer de los sistemas de Bases de Datos; más bien son convenientes para facilitar la comprensión.

Definición 8 Un *término* es una constante o una variable, i.e.: Term = Const ∪ Var. Un término t es *básico* sii $t \in$ Const. El conjunto de todos los términos básicos se llama el *Universo Herbrand*.

La noción de cláusula, al igual que en Prolog, es una forma Horn, sujetándose a lo antes definido. Precisaremos este concepto.

Definición 9 La definición de cláusula considera lo siguiente:

a) Un *átomo*, $p(t_1, \ldots, t_n)$, consiste de un símbolo predicado p y una lista de argumentos $(t_1, \ldots, t_n)$, donde cada t_i $(1 \leq i \leq n)$ es un término.

b) Una *literal positiva* es un átomo $p(t_1, \ldots, t_n)$, una *literal negativa* es o un átomo negado $\sim p(t_1, \ldots, t_n)$, y *literal básica* es un átomo con términos básicos o un átomo negado con términos básicos. Literales son las literales negativas, las positivas, y las básicas.

c) Una *cáusula* es un conjunto finito de literales, una *cláusula básica* es un conjunto finito de literales básicas, una *cláusula unitaria* aquella que sólo tiene una literal, y *cláusula negativa* (*positiva*) cuando se compone de literales negativas (positivas).

Observación 12

a) Las cláusulas son conjuntos de literales. Sin embargo, en la manipulación algorítmica es útil referir a un orden de las literales que contiene la cláusula. Asumiremos, entonces, el orden en la que aparecen las literales al ser listadas dentro del conjunto que define a la cláusula.

b) La notación de cláusula corresponde a una normalización de las fbc del cálculo de predicados. Por ejemplo, la cláusula $\{\sim p(a, X), q(Y, b)\}$ corresponde a la fórmula cerrada $(\forall X)(\forall Y)(\sim p(a, X) \vee q(Y, b))$.

c) El tipo de una literal puede ser compuesto; por ejemplo, $\{\sim r(d, e)\}$ es una clásusula unitaria básica negativa.

Definición 10 Una *cláusula de Horn* es una cláusula con a lo más una literal positiva.

Observación 13

a) Como se ha visto en Prolog, existen tres tipos de cláusulas de Horn:

Hechos. No tienen literales negativas y sí una positiva:

$$\{p(t_1, \ldots, t_n)\}.$$

Reglas. Tienen literales negativas y una positiva:

$$\{p(t_1, \ldots, t_n), \sim q(r_1, \ldots, r_k), \ldots, \sim r(s_1, \ldots, s_m)\}.$$

Metas. No tienen literales positivas:

$$\{\sim p(r_1, \ldots, r_k), \ldots, \sim q(s_1, \ldots, s_m)\}.$$

b) Por lo antes dicho acerca de la representación de las cláusulas, tenemos que una regla

$$C : \{p(t_1, \ldots, t_n), \sim q(r_1, \ldots, r_k), \ldots, \sim r(s_1, \ldots, s_m)\},$$

es equivalente a una fbc cerrada del cálculo de predicados $\mathcal{L}$:

$$(\forall X_1) \ldots (\forall X_j)(p(t_1, \ldots, t_n) \vee \sim q(r_1, \ldots, r_k) \ldots \vee \sim r(s_1, \ldots, s_m)),$$

donde $X_1, \ldots, X_j$ son todas las variables que ocurren en la cláusula. Además, debido a la equivalencia de conectivos, la regla anterior equivale a:

$$\underline{C} : (\forall X_1) \ldots (\forall X_j)(q(r_1, \ldots, r_k) \wedge \ldots \wedge r(s_1, \ldots, s_m) \rightarrow p(t_1, \ldots, t_n)).$$

Finalmente, lo anterior se escribe en el lenguaje de la programación lógica como:

$$p(t_1, \ldots, t_n) : -q(r_1, \ldots, r_k), \ldots, r(s_1, \ldots, s_m).$$

Ejemplo 5 Consideremos la cláusula de Prolog vista en la subsección 1.3, `hermano/2`. La escritura de `hermano/2` como conjunto de literales sería:

$$\{hermano(X, Y), \sim padre(Z, X), \sim padre(Z, Y), EQ(X, Y)\},$$

donde podemos ver que hay dos literales positivas. Este es un motivo que impide manejar la negación en Datalog (a pesar de ello, como en Prolog, es posible ajustar las definiciones para abarcar la negación en Datalog). Este mismo ejemplo, en $\mathcal{L}$ se podría escribir:

$$(\forall X)(\forall Y)(\forall Z)$$

$$(hermano(X, Y) \vee \sim padre(Z, Y) \vee \sim padre(Z, Y) \vee EQ(X, Y)).$$

Ejercicio 10 Explique por qué la cláusula de Horn del segundo inciso de la observación 13 es equivalente a la implicación de $\mathcal{L}$ que se muestra al final de la observación.

Definición 11 Una *variante* de una clásusla C es el resultado de reemplazar las variables que ocurran en C, por otras que no ocurran en C.

Definición 12 Dos cláusulas, C_1 y C_2, son *equivalentes* si $\vdash C_1 \leftrightarrow C_2$.

Observación 14 Debido a que se cumple:

> Si x_i ocurre libre en $\mathcal{A}(x_i)$ y x_j es una variable que no ocurre en $\mathcal{A}(x_i)$, entonces: $\underset{\mathcal{L}}{\vdash} (\forall x_i)\mathcal{A}(x_i) \leftrightarrow (\forall x_j)\mathcal{A}(x_j)$.

se tiene que las variantes de una cláusula son equivalentes a ella. Sin embargo, una que sea equivalente no es necesariamente una variante.

Ejercicio 11

a) Explique la afirmación que se hace al final de la observación anterior.

b) Dé un ejemplo de una cláusula donde una equivalente no sea una variante.

Observación 15 Se ha visto qué representan las cláusulas de Horn para el caso de reglas. En una meta se atiende la manera en cómo Datalog, y también Prolog, resuelven la consulta. Una meta se ve como la negación de una fórmula, y la solución de ésta se obtiene por medio de una demostración por refutación. En una sección más adelante se veremos en detalle la evaluacón de metas. Por ahora, retomemos el ejemplo de inicio de la sección 1.3, "¿quién es padre de guadalupe?". Sabemos que esta meta se escribe como

```
?- padre(X, guadalupe).
```

lo cual representa la cláusula $\{\sim padre(X, guadalupe)\}$. Y, puesto que en $\mathcal{L}$ debemos cerrar la fórmula, tendríamos: $(\forall X) \sim padre(X, guadalupe)$ que es equivalente a $\sim (\exists X) padre(X, guadalupe)$. Así, al tratar de demostrar que "no existe X que sea padre de guadalupe" debemos encontrar una contradicción. Justamente, el procedimiento que realiza la máquina de inferencia termina al agotarse la lista de literales que forman la meta: una cláusula vacía que representa F.
En Prolog las metas son cláusulas con literales negativas, mientras que una *meta Datalog* es una cláusula unitaria negativa, para la cual el predicado ocurre normalmente como cabeza en una regla. Esto no afecta la generalidad, puesto que cualquier meta de Prolog, compuesta de muchas literales, puede convertirse en una meta Datalog.

Ejercicio 12 Represente la consulta "¿quién es la madre de guadalupe?" como: conjunto de literales, una fbc del cálculo de predicados, una meta de Prolog, y una meta Datalog.

Definición 13 Un conjunto de cláusulas de Horn, $C : \{C_1, \ldots, C_k\}$, en Datalog representan la fbc de $\mathcal{L}$, $\underline{C} : \underline{C_1} \wedge \ldots \wedge \underline{C_k}$.

Ejercicio 13 Escriba la fbc correspondiente a las cláusulas que definen `primo/2`, dadas en la sección 1.3.

El alfabeto de símbolos predicados, Pred, se considera dividido en dos: Pred = IPred∪EPred, tal que IPred∩EPred = $\emptyset$, donde EPred es el conjunto de predicados definidos extensionalmente, e IPred es el conjunto de predicados definidos intensionalmente.

Observación 16 Refiriendo al primer ejemplo de la sección 1.3, EPred está constituido por *padre* y *persona*, en tanto que IPred por *hermano* y *primo*. Dado un conjunto de cláusulas unitarias, S, haremos referencia a $I(S)$ ($E(S)$) como el conjunto de cláusulas unitarias cuyos predicados están definidos en IPred (EPred); notemos que $S = I(S) \cup E(S)$.

Definición 14 La *base Herbrand*, BH, es el conjunto de cláusulas básicas unitarias positivas que pueden formarse con símbolos de Pred y constantes en Const. La parte extensional e intensional de BH es $EBH = E(BH)$ e $IBH = I(BH)$, respectivamente. Una *base de datos extensional*, BDE, es un subconjunto finito de EBH; un conjunto de cláusulas unitarias positivas básicas.

Ejemplo 6 Considerando la cláusula `anc/2`, tenemos que $IBH = \{anc(juan, javier),\ anc(javier, margot),\ anc(guadalupe, pablo),\ anc(maria, ana), \ldots\}$, y $EBH = \{padre(juan, javier), padre(javier, margot), padre(margot, ana), \ldots, persona(pablo, 7, m), persona(juan, 78, m), \ldots\}$.

Definición 15 Un *programa Datalog* es un conjunto finito de cláusulas, P, tal que para toda $C \in P$ se tiene $C \in BDE$ o C es una regla que satisface:

1. El predicado que aparece en la cabeza de C pertenece a IPred.
2. Todas las variables en la cabeza de C también ocurren en el cuerpo de C.

Las anteriores condiciones definen lo que es una *regla Datalog*.

Ejercicio 14 Represente el programa `anc/2` como una fbc.

2.1. Unificación

Definición 16 Definimos los siguientes conceptos relativos a una *substitución* θ:

1. θ es un conjunto finito de la forma $\{X_1/t_1, \ldots, X_n/t_n\}$, donde cada X_i es una variable distinta y t_i es un término tal que $X_i \neq t_i$ $(1 \leq i \leq n)$.
2. Cada elemento X_i/t_i es llamado *enlace*.
3. El conjunto de variables $\{X_1, \ldots, X_n\}$ es llamado el *dominio* de θ, mientras que el conjunto de términos $\{t_1, \ldots, t_n\}$ es llamado el *codominio* de θ.
4. Cuando todos los elementos del codominio son constantes entonces θ es una *substitución básica*.
5. Si θ es una substitución y t un término, la *aplicación* de θ a t, $t\theta$ denota el término definido como:
$$t\theta = \begin{cases} t_i, & \text{si } t/t_i \in \theta; \\ t, & \text{en caso contrario.} \end{cases}$$
6. Si L es una literal, entonces $L\theta$ denota la literal que se obtiene aplicando θ a cada término que ocurra en L
7. Si $C = \{L_1, \ldots, L_n\}$ es una cláusula, entonces $C\theta = \{L_1\theta, \ldots, L_n\theta\}$.
8. Si C y D son dos cláusulas y si existe una substitución θ tal que $C\theta = D$, entonces D es llamada una *instancia* de C.
9. Dadas dos substituciones $\theta = \{X_1/t_1, \ldots, X_n/t_n\}$ y $\phi = \{Y_1/u_1, \ldots, Y_m/u_m\}$, la *composición* de θ y ϕ, $\theta\phi$, es la substitución obtenida a partir del conjunto
$$\{X_1/t_1\phi, \ldots, X_n/t_n\phi, Y_1/u_1, \ldots, Y_m/u_m\},$$
eliminando los enlaces de la forma α/α, y aquellos Y_i/u_i para los cuales $Y_i = X_j$ para alguna $j, 1 \leq j \leq n$.

Observación 17 La composición de dos substituciones es una operación cerrada; i.e. obtenemos una substitución. Sin embargo puede dudarse si es equivalente aplicar la primera substitución seguida de la otra, y componerlas para luego aplicarla. Sean dos substituciones $\theta = \{X_1/t_1, \ldots, X_n/t_n\}$ y $\phi = \{Y_1/u_1, \ldots, Y_m/u_m\}$. Veamos que $t(\theta\phi) = (t\theta)\phi$. El término $t(\theta\phi)$ es, por definición (punto 9):
$$t(\theta\phi) = \begin{cases} t_i\phi, & \text{si } t \in Dom(\theta); \\ u_i, & \text{si } t \notin Dom(\theta), t \in Dom(\phi); \\ t & \text{demás casos.} \end{cases}$$
De acuerdo con la definición, en el punto 5, al aplicar ϕ a $t\theta$ se obtiene:
$$(t\theta)\phi = \begin{cases} t_i\phi, & \text{si } t \in Dom(\theta); \\ t\phi, & \text{demás casos.} \end{cases}$$
donde $t\phi$ es:
$$t\phi = \begin{cases} u_i, & \text{si } t \in Dom(\phi); \\ t, & \text{demás casos.} \end{cases}$$

Ejemplo 7 Dada la cláusula unitaria $C : \{\sim p(a, X, Y, b)\}$ una instancia de ella es $D : \{\sim p(a, c, X, b)\}$ ya que con $\theta = \{X/c, Y/X\}$ se tiene: $C\theta = D$.

Ejercicio 15

a) Sea la cláusula
C : `anc(X,Y):- padre(X,Z),anc(Z,Y).`
y las substituciones $\theta = \{X/Y, Y/U, Z/V\}$, $\phi = \{Y/guadalupe, U/T, V/T, X/antonio\}$. Obtenga $\theta\phi$, $C\theta$, $(C\theta)\phi$, y $C(\phi\theta)$.

b) Con respecto a $C\theta$, $(C\theta)\phi$, y $C(\phi\theta)$, ¿cuáles son variantes de C?

c) Diga por qué la composición de substituciones no es conmutativa.

Definición 17 Una cláusula C *subsume* a una cláusula D, $C \rhd D$, sii existe θ tal que $C\theta \subseteq D$.

Ejemplo 8 Para las cláusulas $C : \{q(X,Z), \sim q(X,Y), \sim q(Y,Z)\}$, $D : \{q(a,a), \sim q(a,a), \sim q(b,b)\}$ y la substitución $\theta = \{X/a, Y/a, Z/a\}$ se cumple que $C\theta \subseteq D$, por tanto $C \triangleright D$.

Observación 18 Los conceptos subsunción, instanciación y variación van de lo más general a lo más específico. Por ejemplo, una variante es una instancia, no al revés.

Proposición 1 Sean C y D dos cláusulas, y $X_1, \ldots, X_n$, $Y_1, \ldots, Y_m$, las variables que ocurren en C y D, respectivamente. Si para dos substituciones θ y ϕ se cumple que $C\theta = D$ y $D\phi = C$, entonces los enlaces $X_i/t_i \in \theta$, y $Y_j/u_j \in \phi$ cumplen que t_i, u_j son variables, para toda $1 \leq i \leq n$, $1 \leq j \leq m$.

Proposición 2 La cláusula C es una variante de D si existen θ y ϕ tales que $C\theta = D$ y $D\phi = C$.

Definición 18 Dadas dos literales L_1 y L_2, si existe una substitución θ tal que $L_1\theta = L_2\theta$, entonces se dice que L_1 y L_2 son *unificables*, y θ es llamado *unificador* de L_1 y L_2.

Observación 19

a) Considérense las literales $L_1 : p(X,a,Z)$, y $L_2 : p(V,W,b)$. Algunos unificadores de L_1 y L_2 son:

   a) $\theta_1 = \{X/V, W/a, Z/b\}$
   b) $\theta_2 = \{X/a, V/a, W/a, Z/b\}$
   c) $\theta_3 = \{X/b, V/b, W/a, Z/b\}$
   d) $\theta_4 = \{X/c, V/c, W/a, Z/b\}$
   e) $\theta_5 = \{X/U, V/U, W/a, Z/b\}$
   f) $\theta_6 = \{X/U, Z/b, V/U, W/a, T/g\}$

b) Para $L_3 : q(X,a,Z)$, $L_4 :\sim q(X,a,Z)$, $L_5 : q(Y,Z,b)$ y $L_6 : q(X,a,Z,Z)$, las parejas L_1, L_3; L_3, L_4; L_3, L_5; y L_3, L_6, no son unificables.

Ejercicio 16 Demuestre que $\theta_\alpha = \{X/\alpha, Z/b, V/\alpha, W/\alpha\}$, con $\alpha \in \text{Const} \cup \text{Var}$ es unificador de L_1 y L_2.

Definición 19

a) Sean θ y ϕ dos substituciones. Se dice que la substitución θ es *más general* que ϕ sii existe γ tal que $\theta\gamma = \phi$.

b) Dadas dos literales unificables, L_1 y L_2, un *unificador más general* (**umg**) de L_1 y L_2 es un unificador de L_1 y L_2 más general que cualquier otro.

Observación 20

a) Ya que si $C : p(X,Z) : -p(X,Y), p(Y,Z)$ y $D : p(a,a) : -p(a,a)$, existe $\theta = \{X/a, Y/a, Z/a\}$ tal que $C\theta = D$, tenemos que D es una instancia de D.

b) Para literales $L_1 : p(X,a,Z)$ y $L_2 : p(Y,a,b)$, se cumple que L_2 es una instancia de L_1. Además, son unificables pero no es variante uno del otro.

c) Un **umg** no es único. Dadas dos literales unificables, pueden existir dos **umg** diferentes; esto es, la relación de generalidad entre unificadores no es antisimétrica (si uno es más general que otro y viceversa, entonces no tienen por que ser iguales).

Ejemplo 9 Considérense las literales L_1 y L_2 de la observación 19. θ_5 es más general que θ_i, $2 \leq i \leq 4$, y θ_1 es más general que θ_5. Además θ_1 es **umg**.

Ejercicio 17

a) En referencia al ejemplo 9 muestre que θ_5 es más general que θ_i, $2 \leq i \leq 4$.

b) Demuestre que el unificador del ejercicio 16 es un **umg**.

Para obtener un **umg** de dos literales se recorre de inicio a fin ambas literales, observando si hay empatamiento. En caso de que no lo haya, y si podemos proponer un enlace que garantice el empatamiento tomando en cuenta los antes definidos, se añade a la substitución candidata. Si no es posible empatar las literales se termina el proceso anunciando que las literales no son unificables. La siguiente función construye un **umg** de un par de literales.

```
function umg(L1, L2)
//entrada: dos literales L1 : ℓ1(t1,...,tn) y L2 : ℓ2(u1,...,um).
//salida: un umg θ, si existe; ∇, en caso contrario.
    if ℓ1 ≠ ℓ2 or n ≠ m then return ∇
    else
        θ = {}; i = 1; unifico =true
        repeat
            if t_i θ ≠ u_i θ
            then if u_i θ ∈ Var
                then θ = θ{u_i θ/t_i θ}
                else if t_i θ ∈ Var
                    then θ = θ{t_i θ/u_i θ}
                    else unifico =false
            i++
        until i > n or not unifico
        if unifico then return θ
        else return ∇
end umg;
```

Ejemplo 10 Considere las literales L_1 y L_2 de la observación 20. La traza de θ en el llamado $\texttt{umg}(L_1, L_2)$ es:

$$\theta = \{\} \quad (1)$$
$$\theta = \{\}\{V/X\} = \{V/X\} \quad (2)$$
$$\theta = \{V/X\}\{W/a\} = \{V/X, W/a\} \quad (3)$$
$$\theta = \{V/X, W/a\}\{Z/b\} = \{V/X, W/a, Z/b\} \quad (4)$$

Ejercicio 18 Obtenga la traza de θ para el llamado $\texttt{umg}(L_1, L_2)$, con $L_1 : p(X,Z,a,U)$ y $L_2 : p(Y,Y,V,W)$.

2.2. Teoría de modelos

Hemos concebido Datalog como un sistema que a partir de una consulta obtiene un resultado en términos de las relaciones que se han definido; esto es, la base de datos extensional y la intensional. La fundamentación trata de asegurar que lo que se espera que haga un sistema sea, efectivamente, lo que se obtiene. Para lograr este propósito antes debemos definir con exactitud qué calcula Datalog. Ya hemos avanzado en este sentido con las definiciones de la sección anterior. Ahora, apoyándonos nuevamente en la lógica clásica, definiremos la semántica de Datalog.
La primera aproximación a la semántica es ver el resultado de una consulta como una consecuencia lógica del programa Datalog (BDI y BDE). Subrrayemos que este resultado, como hemos visto, es un conjunto de cláusulas unitarias positivas básicas; subconjunto, a la vez, de la base Herbrand (BH). Así, debemos especificar qué significa con precisión *consecuencia lógica* en este contexto. Para ello, a manera de recordatorio adaptado a Datalog (sin funciones), se presentan algunos conceptos y resultados del cálculo informal de predicados $\mathcal{L}$.

Definición 20 Una *interpretación* I de $\mathcal{L}$ es un conjunto no vacío D_I, el *dominio* de I, junto con una colección de elementos *distinguidos*, $\bar{a_1}, \bar{a_2}, \ldots \in D_I$, y una colección de relaciones sobre D_I, $\bar{A_i^n} \subset D_I^n, (i > 0, n > 0)$.

Definición 21 Una *valuación* en I es una función v del conjunto de términos de $\mathcal{L}$ al conjunto D_I que cumple $v(a_i) = \bar{a}_i$, para cada constante $a_i \in \mathcal{L}$.

Observación 21

1. Una valuación asigna objetos del dominio a los elementos de $\mathcal{L}$ para poder realizar una interpretación en $\mathcal{L}$.
2. Dada una interpretación I, pueden definirse diversas valuaciones.
3. Ya que D_I es el dominio sobre el cual toman valores las variables de $\mathcal{L}$, una valuación v quedará completamente especificada si se define $v(x_1), v(x_2) \ldots$.

Definición 22 Dos valuaciones v y v' son *equivalentes-i* si $v(x_j) = v'(x_j)$, para toda $j \neq i$.

Definición 23 Sea $\mathcal{A}$ una fbc de $\mathcal{L}$ e I una interpretación de $\mathcal{L}$. Se dice que una valuación v en I *satisface* $\mathcal{A}$ con base en el cumplimiento de:

1. v satisface la fórmula atómica $A_j^n(t_1, \ldots, t_n)$ sii $(\bar{t}_1, \ldots, \bar{t}_n) \in \bar{A}_j^n \subset D_I^n$.
2. v satisface $\sim \mathcal{B}$ sii v no satisface $\mathcal{B}$.
3. v satiface $\mathcal{B} \rightarrow \mathcal{C}$ sii v satisface $\sim \mathcal{B}$, o bien v satisface $\mathcal{C}$.
4. v satisface $(\forall x_i)\mathcal{B}$ sii para cada valuación equivalente-i v' satisface a $\mathcal{B}$.

Ejemplo 11 Para toda valuación v y fbc $\mathcal{A}$, v satisface $\mathcal{A}$ o bien v satisface $\sim \mathcal{A}$, ya que solamente hay dos posibilidades: v satisface $\mathcal{A}$ o no satisface $\mathcal{A}$. Lo último, según la definición 23, significa v satisface $\sim \mathcal{A}$.

Definición 24 Una fbc $\mathcal{A}$ es *verdadera* en una interpretación I si para toda valuación $\mathcal{A}$ se satisface ($I \vDash \mathcal{A}$). $\mathcal{A}$ es *falsa* si no hay valuación en I que satisfaga $\mathcal{A}$.

Definición 25 Una fbc $\mathcal{A}$ de $\mathcal{L}$ es *lógicamente válida* si $I \vDash \mathcal{A}$ en cada I de $\mathcal{L}$. $\mathcal{A}$ es *contradictoria* si es falsa en toda interpretación.

Proposición 3 Sean I una interpretación de $\mathcal{L}$ y $\mathcal{A}$ una fbc de $\mathcal{L}$. Si v y w son valuaciones tales que $v(x_i) = w(x_i)$ para cada variable libre x_i de $\mathcal{A}$, entonces v satisface $\mathcal{A}$ sii w satisface $\mathcal{A}$.

Definición 26 Una fbc $\mathcal{A}$ de $\mathcal{L}$ se dice *cerrada* si no ocurren variables libres en $\mathcal{A}$.

Observación 22 Conviene tener presente que las cláusulas de los programas lógicos son fórmulas cerradas. Considerermos el ejemplo 11. De acuerdo con la proposición 3, si una valuación satisface una fórmula cerrada en una interpretación, entonces todas la satisfarán; y si no la satisface, entonces ninguna la satisfará, por el hecho de que no tiene variables libres. Este resultado se enuncia en la siguiente proposición.

Proposición 4 Si $\mathcal{A}$ es una fbc cerrada de $\mathcal{L}$ e I es una interpretación de $\mathcal{L}$, entonces $I \vDash \mathcal{A}$ o bien $I \vDash \sim \mathcal{A}$.

Observación 23 Un ejemplo muy simple de validez lo constituye la cláusula de Datalog $p(a) : -p(a)$, debido a que sabemos que $\vdash \mathcal{A} \rightarrow \mathcal{A}$, y que hay solidez en $\mathcal{L}$ (todo teorema es válido). Y, un ejemplo de invalidez sería {p(a)}, puesto que podemos con facilidad proponer una interpretación en la que no se satisfaga.

Ejercicio 19 ¿Por qué es válida la cláusula $q(X) : -p(b), q(X)$?

Observación 24 Los conceptos: satisfacción, veracidad, y validez, van en orden de exigencia para el predicado que califican, como se puede apreciar en las definiciones 23, 24, y 25, respectivamente. Para la cláusula $C : p(X, Y) : -p(X, Z), p(Z, Y)$ podemos dar, en un dominio de personas D_p, la interpretación de "amistad" al predicado p. En esta interpretación no siempre se satisface C. Sin embargo, en el dominio de los números naturales, y tomando a p como la relación $\leq$, C es verdadera.

Ejercicio 20 Diga claramente por qué la clásusula de la observación 24 no es válida.

Observación 25 La noción de validez será empleada con asiduidad. Por ello, convendremos que un conjunto de cláusulas $S : \{C_1, \ldots, C_n\}$ es válido sii cada C_i es válida, $1 \leq i \leq n$. De igual forma para satisfacción, y veracidad. En la siguiente definición cambiamos el punto de vista sobre la veracidad, centrándonos en la interpretación y no en las cláusulas.

Definición 27

a) Diremos que una interpretación I es un *modelo* de $S : \{C_1, \ldots, C_n\}$ si S es verdadero en I.

b) Un conjunto de cláusulas S es válido sii toda interpretación es modelo de S.

Observación 26 Debido a que nos interesa formalizar el funcionamiento de Datalog, que, como observamos al inicio de esta sección, se apoya en la BDI y en la BDE para obtener la respuesta a una consulta, es más adecuado considerar deducciones ($\Gamma \vdash \mathcal{A}$) que teoremas ($\vdash \mathcal{A}$); es decir, partimos de un conjunto de supuestos (DBE, BDE: Γ) para poder afirmar que se cumplen ciertos hechos (consulta: $\mathcal{A}$). Debe recordarse que, por solidez y adecuación, puede transitarse de lo formal a lo informal, y viceversa: $\vdash \mathcal{A}$ sii $\models \mathcal{A}$.

Definición 28 Un hecho básico H es una *consecuencia* de un conjunto de claáusulas S, denotado por $S \models H$, sii todo modelo de S es también modelo de H.

Ejemplo 12 Sean $C_1 : \{p(a, b)\}$, $C_2 : p(X, Y) : -p(Y, X)$, y $H : \{p(b, a)\}$. Claramente para cualquier interpretación (dominio, predicado y constantes), H será verdadera si C_1 Y C_2 lo son. Esto es, todo modelo de C_1 y C_2 también lo es de H, por tanto $C_1, C_2 \models H$.

Ejemplo 13 ¿Cómo podría decidirse si un hecho es consecuencia de la BDI y BDE?
Tómese $S = \text{BDE} \cup \text{BDI}$. Hay, en esencia, dos formas:

a) La primera es demostrando $S \vdash \mathcal{A}$, y usar la solidez del cálculo de predicados para concluir: $S \models \mathcal{A}$.

b) La segunda, consiste en revisar todas las interpretaciones para determinar si son modelos de $\mathcal{A}$, en caso de que lo sean de S.

Como vemos, el primer caso plantea un problema difícil. El segundo parece inviable porque tendría que verificarse un número muy grande de interpretaciones sobre S (BDE y BDI), lo cual es complicado. Veremos que la opción es replantear el segundo camino apoyándonos en las propuestas que hicieron J. Herbrand y T. Skolem.

Observación 27 Notemos que, en particular, las interpretaciones de la observación 24, I_P (personas) e I_N (números naturales), son *legibles*. Esto es, al asignar valores del dominio, D_P o D_N, a símbolos variables y constantes, digamos "juan", "ana", y "2", "3" , podemos leer el predicado y entender, en nuestro dominio, lo que expresa: $p(juan, ana)$ o $p(2, 3)$; "juan es amigo de ana" o "$2 \leq 3$", etc. Pero, no hay impedimento para interpretar *literalmente*, *en abstracto*, los símbolos constantes y predicados; por ejemplo: $p(a, b)$ puede leerse "a está p-relacionado con b", aunque **no** se legible. Inclusive, la interpretación abstracta puede realizarse cuando los símbolos predicado tienen nombres conocidos pero no adecuados: "a según b", $2 \odot 3$, etc. Un tipo de interpretaciones, llamadas Herbrand, son abstractas: toman como dominio el conjunto de constantes de $\mathcal{L}$ (Const), y la interpretación de los símbolos predicado son ellos mismos: $\bar{A}_i^n$ es A_i^n. Habrá, entonces, tantas interpretaciones Herbrand como combinaciones de literales básicas positivas; a las que les corresponderá el valor de certeza verdadero. En la definición siguiente se sintetiza lo anterior.

Definición 29 Una *interpretación Herbrand*, I, es un subconjunto de la base Herbrand, $I \subseteq \text{BH}$.

Observación 28 Tomar un subconjunto de BH para definir una interpretación, I, significa considerar como dominio $D_I = \text{Const}$, y con los predicados $\text{Pred} = \{\mathcal{A}_i^j\}_{i,j}$ se constituyen las relaciones subconjuntos de Const^j, $j \geq 1$. Aquellas tuplas que no estén en I se considera que no son satisfechas.

Ejemplo 14 Determine todas las interpretaciones Herbrand para $\text{Const} = \{a, b\}$, y $\text{Pred} = \{p\}$. La BH es el conjunto de todas las cláusulas positivas básicas unitarias: $\{p(a,a), p(b,b), p(a,b), p(b,a)\}$. A partir de ésta debemos calcular su potencia, 2^{BH}:

$$
\begin{aligned}
I_1 &= \emptyset \\
I_2 &= \{p(a,a)\} \\
I_3 &= \{p(b,b)\} \\
I_4 &= \{p(a,b)\} \\
I_5 &= \{p(b,a)\} \\
I_6 &= \{p(a,a), p(b,b)\} \\
I_7 &= \{p(a,a), p(a,b)\} \\
I_8 &= \{p(a,a), p(b,a)\} \\
I_9 &= \{p(b,b), p(a,b)\} \\
I_{10} &= \{p(b,b), p(b,a)\} \\
I_{11} &= \{p(a,b), p(b,a)\} \\
I_{12} &= \{p(a,a), p(b,b), p(a,b)\} \\
I_{13} &= \{p(a,a), p(b,b), p(b,a)\} \\
I_{14} &= \{p(a,a), p(a,b), p(b,a)\} \\
I_{15} &= \{p(b,b), p(a,b), p(b,a)\} \\
I_{16} &= \{p(a,a), p(b,b), p(a,b), p(b,a)\}
\end{aligned}
$$

Notemos que cualquier otra interpretación tendrá la "forma" de una de las interpretaciones Herbrand. Por tanto, basta tratar con interpretaciones Herbrand.

Definición 30 Dada una interpretación Herbrand, I, se define:

a) Un hecho básico, H es *verdadero bajo* I sii $H \in I$; en caso contrario, H es falso.

b) Una regla $R : L_0 : -L_1, \ldots, L_n$ es *verdadera bajo* I sii para cada substitución básica θ, $L_1\theta \in I, \ldots, L_n\theta \in I$ implica $L_0\theta \in I$. En caso contrario R es falsa.

Observación 29 Las interpretaciones Herbrand son un caso particular de las interpretaciones, sin embargo, estamos usando I como conjunto de tuplas de relaciones diversas, y esto merece algunas adaptaciones. Sea $I \subseteq \text{BH}$. Por la definición 30, una cláusula es verdadera si es verdadera bajo I. Si todas las cláusulas de un conjunto S son verdaderas bajo I, entonces S es verdadero bajo I. Si un conjunto de cláusulas S es verdadero bajo I, entonces I es un modelo Herbrand de S. Por último, se reformula la noción de consecuencia. Sea H un hecho básico y S un conjunto de cláusulas Datalog. H es una consecuencia de S, $S \models H$, sii cada modelo Herbrand de S es un modelo Herbrand de H.

Ejemplo 15 Supongamos que se tiene la siguiente regla Datalog $R : amigo(X,Y) : -amigo(Y,X)$. Y considérense las siguientes interpretaciones Herbrand:
$I_1 = \{amigo(juan, pablo), amigo(pablo, juan)\}$,
$I_2 = \{amigo(juan, pablo), amigo(pablo, juan), amigo(ana, juan)\}$
Bajo I_1, R es verdadera. No así bajo I_2, ya que dada la substitución $\theta = \{X/juan, Y/ana\}$, tenemos que $L_1\theta$ es verdadero pero $L_0\theta$ es falso: $L_1\theta \in I_2$ pero $L_0\theta \notin I_2$. Por tanto, I_2 no es un modelo Herbrand de $S = \{R\}$.

Observación 30 Con lo anterior estamos en posición de precisar, lo suficiente para nuestros fines, la semántica de Datalog. La semántica de un lenguaje de programación considera normalmente los programas, los datos, y los resultados. Datalog toma un programa y, dada una meta, devuelve un conjunto de tuplas que son subsumidas por la meta. Esta idea la debemos especificar de acuerdo con nuestra definición de consecuencia; esto es, "cuáles" tuplas obtiene. Supongamos que se tiene una meta:

```
?:-M.
```

y un programa P que utiliza una BDE. Debe entonces verificarse que las tuplas sean consecuencia del programa y la meta:

$$\forall E \subseteq \text{BHE} : \mathcal{W}_{P,M}(E) = \{T | T \in \text{BH} \wedge P \cup E \models T \wedge M \triangleright T\}. \quad (5)$$

Las tuplas, T, que entrega Datalog son aquellas que pertenencen a la base Herbrand (unitarias positivas básicas), que son consecuencia del programa (P: la base de datos intensional) y de la base de datos extensional, E, y además son subsumidas por la meta.
Observemos, además, que las operaciones que han de realizarse para encontrar la respuesta a una consulta Datalog son, aún, costosas puesto que, por un lado, puede haber numerosos modelos Herbrand, por otro, algunos de ellos tendrán muchísimos elementos. Finalmente, debemos considerar las substituciones básicas que también pueden ser numerosas.

2.3. El modelo mínimo

Se tiene determinado cuáles son las tuplas que calcula Datalog, pero no se sabe aún cómo las obtiene. Ciertamente, podemos seguir la definición, ec. 5, pero, como vimos en el ejemplo 14, es intratable para BDE reales; aunque, afortunadamente, son finitas. Para atender este problema se define un modelo Herbrand que reduce los cálculos en esta tarea, el *modelo mínimo de Herbrand*. Para definir este modelo, antes definamos el concepto de consecuencia de un conjunto de cláusulas.

Definición 31 Sea S un conjunto de cláusulas Datalog. El conjunto de todos los *hechos básicos que son consecuencia de* S es:

$$cons(S) = \{H | H \in BH \wedge S \models H\}. \quad (6)$$

Ejercicio 21 Considere $S = \{padre(a,b), padre(a,c), padre(b,d)\}$, y $\text{Const} = \{a, b, c, d\}$.

1. Explique por qué $I = \{padre(a,b)\}$ no es un modelo de S.
2. ¿Cuáles son modelos de S.
3. ¿Cuáles hechos son consecuencia de S?

Ejemplo 16 Considere $\text{Pred} = \{amigo\}$, $\text{Const} = \{juan, pablo, ana\}$ y $S = \{R, amigo(juan, ana), amigo(pablo, juan)\}$, donde $R : amigo(X,Y) : -amigo(X,Z), amigo(Z,Y)$. Determine $cons(S)$.
Debemos proponer elementos de BH que tengan como modelo I, para todo modelo I de S. Elegimos, inicialmente, I como la BDE; debido al ejercicio 21. Por tanto, la BDE es verdadera bajo I. No así R; por ejemplo, con $\theta = \{X/pablo, Y/ana, Z/juan\}$, $R\theta$ tiene en el consecuente a $amigo(pablo, ana)$, pero $amigo(pablo, ana) \notin I$. De esta forma se aumenta I con la tupla necesaria; $amigo(pablo, ana)$. Tenemos, ahora, que $I_1 = \{amigo(juan, ana), amigo(pablo, juan), amigo(pablo, ana)\}$, es un modelo de S. Debemos preguntarnos si existe otra interpretación que sea modelo de P∪I. Claramente, estas interpretaciones son todas aquellas que contengan a I_1: I_j. Enseguida, deben buscarse los hechos, H, que son consecuencia de S; los modelos de S deben también serlo de cada H. El ejercicio 21 permite concluir que tales hechos cumplen $H \in I_j$. Entonces, I_j, obtenido como se ha explicado, es modelo de $R \cup I_j$. Pero, como los hechos que son consecuencia de S deben, para todo modelo de S, también pertenecer al modelo, concluimos que $cons(S) = I_1$.

Ejercicio 22 Dé tres interpretaciones Herbrand que no sean modelos de R en el ejemplo anterior.

Observación 31 Con la ecuación 6 podemos determinar, dado un hecho, si éste es consecuencia de un conjunto de cláusulas; lo cual requiere determinar si el hecho pertenece a la base Herbrand. Como se ha visto, para decidir si una cláusula C es verdadera bajo una interpretación Herbrand I debe procederse según la definición 30 (I sea modelo de C). En el ejemplo anterior se ha mostrado que no es necesario determinar todos los modelos de un conjunto S ya que, procediendo con base en S, podemos ir completando el modelo y todos los demás contendrán a éste. Es por ello que hablamos de un *modelo mínimo*. Por ahora se tiene un procedimiento intuitivo para determinar las consecuencias de un conjunto. En la siguiente sección se presentarán propiedades de los procedimientos implícitos en esta determinación. Terminaremos la presente sección enunciando dos resultados que generalizan la observación sobre el modelo mínimo.

Proposición 5 Para un conjunto de cláusulas S, $cons(S)$ es la intersección de todos los modelos Herbrand de S:

$$cons(S) = \bigcap\{\mathcal{J} | \mathcal{J} \text{ es un modelo Herbrand de } S\}.$$

Proposición 6 $cons(S)$ es un modelo Herbrand de S.

Observación 32 La proposición anterior nos garantiza que $cons(S)$ es un modelo Herbrand de s y, como vimos en la observación 31, se obtiene con más facilidad; es el primero que se encuentra siguiendo las definiciones de modelo y consecuencia. La proposición 5 caracteriza al modelo mínimo de manera formal, pero no pretende indicar la forma en que se determina algoritmicamente.

Ejercicio 23

A) Sean las substituciones $\theta = \{X/Z, Y/a\}$, $\phi = \{X/Y, Y/Z\}$, y $\gamma = \{Z/a, X/Z\}$. Obtenga $\theta\phi$, $\theta\gamma$, $\phi\gamma$, $\gamma\phi\theta$, y $\gamma\theta\phi$.

B) Para el conjunto de todas las substituciones diga cuál es la substitución *identidad*; i.e. ϵ tal que $\theta\epsilon = \epsilon\theta = \theta$, para toda θ.

C) Proponga dos cláusulas tales que $C \neq D$ y $C \rhd D$ y $D \rhd C$.

D) Proponga dos reglas, $L : L_0 : -L_1, \ldots, L_n$, $M : M_0 : -M_1, \ldots, M_m$, tales que $L_0 \rhd M_0$ y $L_1, \ldots, L_n \rhd M_1, \ldots, M_m$, pero $L \not\rhd M$.

E) Sean los siguientes tres pares de literales: $p(Y, W, Z), p(U, U, V)$, $q(Y, a, b), q(Q, Q, Y)$, y $r(a, b, c), r(R, Z, R)$. Diga cuáles son unificables y por qué.

F) Mostrar que $\theta = \{X/V, U/a\}$ más general que $\phi\{V/X, U/a\}$ y viceversa.

G) Trace θ para el llamado $\texttt{umg}(L_1, L_2)$, donde $L_1 : p(Y, Q, a, R, b)$ y $L_2 : p(a, Y, V, V, Q)$.

H) Muestre que si L es una literal básica y M cualquier otra literal, entonces a lo más existe un **umg** de L y M.

I) Sea $S = \text{BDE} \cup C$ donde $C : L_0 : -L_1, \ldots, L_n$ es una cláusula Datalog. Suponga que I_1 e I_2 son modelos Herbrand de S. Muestre que $I_1 \cap I_2$ es un modelo Herbrand de S.

J) En referencia al ejercicio 21, considere la siguiente lista de cláusulas unitarias básicas positivas:

   a) $\{padre(a, b), padre(b, c), padre(b, d), padre(a, c)\}$
   b) $\{padre(b, a), padre(b, c), padre(b, d), padre(a, c)\}$
   c) $\{padre(a, b), padre(b, c), padre(b, a), padre(a, c), padre(a, a)\}$
   d) $\{padre(a, c), padre(b, c), padre(b, d), padre(a, d)\}$

   - Cuáles y por qué son modelos Herbrand de S.
   - Cuáles y por qué son interpretaciones Herbrand de S.

K) Obtenga $cons(S)$ para $S = \{amigo(ana, juan), amigo(pablo, ana), amigo(juan, pablo), R\}$, donde $R : amigo(X, Y) : -amigo(X, Z), amigo(Z, Y)$.

L) ¿Cuál es el modelo mínimo de S en el ejercicio J y en el ejercicio K?

3. Paradigmas de evaluación

En la sección anterior vimos la sintaxis del lenguaje Datalog y el significado de los programas Datalog, hasta determinar cuáles interpretaciones hacen verdaderas a las cláusulas de un conjunto y, por tanto, permiten calcular la respuesta a una consulta. La presente sección está inspirada en un tratamiento formal, concibiendo Datalog como un sistema con una regla de inferencia que, como veremos, es justamente *modus ponens*, aunque la llamaremos *producción elemental*. De esta manera habilitamos en Datalog la construcción de deducciones partiendo de un conjunto de premisas; i.e. nuestro programa Datalog. A la vez, mucho de lo que hemos visto, en la parte de semántica, sirve como base para las definiciones y para los resultados centrales: solidez y adecuación del sistema. Por último, también se presentará una forma alternativa para determinar los hechos inferidos a partir de un programa y su equivalencia con lo antes planteado.

3.1. Teoría de la demostración

Definición 32 Dada una regla Datalog $L_0 : -L_1, \ldots, L_n$ y una lista de hechos básicos, $H_1, \ldots, H_n$, si existe una substitución θ tal que $H_i = L_i\theta$ $(1 \leq i \leq n)$, entonces $L_0\theta$ es un *hecho inferido en un paso*. Esta forma de obtener un hecho básico se llama *producción elemental* (*pe*).

Observación 33 El hecho inferido por pe puede ser nuevo en un programa Datalog o ya conocido.

Ejercicio 24 Considérese la regla Datalog $p(X,Y) : -p(Y,X)$ y el hecho básico $p(a,b)$. $p(b,a)$ es un hecho inferido en un paso usando pe, pues existe $\theta = \{Y/a, X/b\}$ tal que $L_1\theta = p(a,b)$. Asimismo, partiendo del hecho básico $p(a,a)$ obtenenos en un paso de inferencia el hecho mismo: $p(a,a)$.

La inferencia por pe puede ser aplicada de manera sistemática a parejas de reglas y listas de hechos básicos. Un algoritmo que obtiene inferencias en un paso es el que se muestra enseguida.

```
function pe(R, H_1, ..., H_n)
//entrada: una regla de la forma R : L_0 : -L_1, ..., L_n y una lista
de hechos básicos H_1, ..., H_n).//
//salida: un hecho inferido en un paso, si es aplicable PE; ∇, en
caso contrario.//
    for i = 0 to n do K_i = L_i
    for i = 1 to n do
        λ =umg(K_i, H_i)
        if λ == ∇ then return ∇
        else for j = 0 to n do K_j = K_jλ
    return K_0
end pe;
```

Observación 34 El algoritmo `pe` es correcto; esto es, obtiene lo que indica la definición 32. Para clarificar lo anterior, en primer lugar observemos que `pe` siempre termina. En segundo lugar, debe mostrarse que obtiene una substitución, θ, que cumple $H_i = L_i\theta$; específicamente $H_i = K_i\theta$, según la copia que hace `pe`. En el paso i, $\texttt{umg}(H_i, K_i)$ determina una substitución, λ_i, tal que $H_i = K_i\lambda_i$, puesto que H_i es una literal básica. Al finalizar `pe`, si $\texttt{umg}(H_i, K_i) \neq \nabla$ para toda $1 \leq i \leq n$, tendremos que con $\theta = \lambda_1 \ldots \lambda_n$ se cumple $H_i = L_i\theta$.

Ejercicio 25 Apliquemos `pe` a la regla $p(X,Z) : -p(X,Y), p(Y,Z)$ con los hechos $p(a,b), p(b,c)$. Tomando $H_1 = (a,b)$ y $H_2 = p(b,c)$, en el llamado a `pe`, después de hacer la copia a las variables K_i, tenemos que $\lambda = \{X/a, Y/b\}$ y la aplicación de esta substitución a los K_j conduce a $p(a,Z) : -p(a,b), p(b,Z)$. En la segunda iteración, $\lambda = \{Z/c\}$, lo cual produce la nueva regla $p(a,c) : -p(a,b), p(b,c)$. Así, $K_0 = p(a,c)$.

Observación 35 Si tomamos $H_1 = (b,c)$ y $H_2 = p(a,b)$ en el ejemplo anterior, obtenemos ∇. Esto significa que el algoritmo propuesto para `pe` tiene debilidades que tendrán que superarse, pues podríamos esperar que fuera insensible al orden de los hechos. Además, nos interesaría obtener todos los hechos que se infieren. El siguiente algoritmo calcula, para cada regla de un conjunto todos los hechos que se infieren en un paso, utilizando `pe`. El algoritmo delega a un mecanismo no determinístico el orden de los hechos que se proveen a `pe`, lo cual atiende, de una forma, el señalamiento realizado al inicio de esta observación.

```
function infer1(S)
//entrada: un conjunto de cláusulas, S.//
//salida: el conjunto de todos los hechos que pueden ser inferidos
en un paso a partir de S.//
    resultado = ∅
    for each R : L_0 ... L_n ∈ S do
        for each < H_1, ..., H_n >, H_i ∈ S do
            hi =pe(R, H_1, ..., H_n)
            if hi ≠ ∇ then resultado = resultado ∪ {hi}
    return resultado
end infer1;
```

Proposición 7 `infer1` es correcto.

Ejercicio 26 Sea $S = \{R_1, R_2, p(a,b), p(b,c), p(c,d), p(d,e)\}$, donde $R_1 : p(X,Z) : -p(X,Y), p(Y,Z)$ y $R_2 : p(X,Y) : -p(Y,X)$. Calcule `infer1`(S).

Observación 36 Puesto que tratamos con el mismo sistema (Datalog) que en la semántica, el camino que estamos siguiendo es muy parecido, aunque ahora tratamos de averiguar si una cláusula (literal) es derivable, y no si es verdadera; no nos interesa, por tanto, cómo se interpretan las cláusulas, e, independientemente de su valor de certeza, hemos establecido mecanismos para producir cláusulas, presumiblemente "aceptadas". Es por ello que debemos precisar cómo definimos inferencia, o deducción en general, dentro del sistema.

Definición 33 Un hecho básico M es *inferido* a partir de un conjunto de cláusulas S, $S \vdash M$, si y sólo si:

a) $M \in S$, o bien

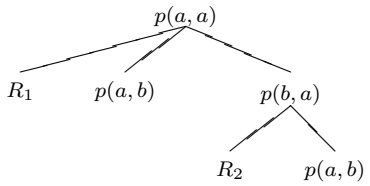
b) existe una regla $R \in S$ y n hechos básicos, $H_1, \ldots, H_n$ tal que $S \vdash H_i$ para toda $i, 1 \leq i \leq n$, y $M =\texttt{pe}(R, H_1, \ldots, H_n)$.

Ejemplo 17 Considere el conjunto S del ejercicio 26. Según la definición de inferencia, puede verse que $S \vdash p(b,a)$, ya que $p(a,b) \in S$ y $p(b,a) =\texttt{pe}(R_2, p(a,b))$. También $S \vdash p(a,a)$ porque $p(a,b), p(b,a) \in S$ y $p(a,a) =\texttt{pe}(R_1, p(a,b), p(b,a))$.

Observación 37 Llamaremos demostración o deducción, igual que acostumbramos hacerlo en lógica clásica, a la secuencia de fórmulas tomadas de las premisas o axiomas, o bien resultado de aplicar una regla de inferencia. Cada paso de esta secuencia es la aplicación de `pe` o la elección de un hecho básico, sea inferido previamente o perteneciente a S; según la definición. Es conveniente, por lo anterior, hablar de un árbol de demostración.

Definición 34 Dado un conjunto de cláusulas, S, un *árbol de demostración* correspondiente a un hecho básico M está compuesto de nodos etiquetados con elementos de S o bien por hechos H tales que $S \vdash H$ en un paso. Los nodos más inferiores, u hojas, son elementos de S. Un nodo de nivel superior, N se obtiene por la aplicación de `pe` a los de niveles inferiores, R, $H_1, \ldots, H_n$, tal que $N =\texttt{pe}(R, H_1, \ldots, H_n)$. El nodo raíz, con ninguno superior a él, es M.

Ejemplo 18 Tomemos el ejemplo 17. El árbol de demostración es:



Ejercicio 27 Construya el árbol de demostración del hecho básico $p(d, a)$, considerando S como en el ejercicio 26.

Observación 38 Identificaremos, en un árbol de demostración, la profundidad de cada nodo como el camino más largo desde el nodo hacia un nodo terminal. En el ejemplo 18, $p(a, a)$ tiene profundidad 2, $p(b, a)$ tiene profundidad 1, y todos los demás nodos tienen profundidad 0.

Ejercicio 28

A) Dado $S = \{R_1,\ p(a,b),\ p(a,c),\ p(b,d),\ p(b,e),\ p(b,f),\ p(c,g),\ p(c,h)\}$, con $R_1 : h(X,Y) : -p(Z,X), p(Z,Y)$, diga cuáles de los siguientes hechos son inferidos en un paso, y cuál es la substitución a la que refiere la definición correspondiente.

   a) $h(d,f)$

   b) $h(d,g)$

   c) $h(e,h)$

   d) $h(h,g)$

B) Sea $R : h(X,Y) : -p(Z,X), p(Z,Y)$. Trace el valor de λ para cada iteración del algoritmo `pe` con los pares: $H_1 : p(b,e)$, $H_2 : p(c,g)$, $H_1 : p(b,f)$, $H_2 : p(b,e)$,y $H_1 : p(b,d)$, $H_2 : p(b,f)$.

C) Calcule `infer1`(S), con S como lo presenta el problema A.

D) Una forma determinística de elegir tuplas como argumentos de `pe` dentro del algoritmo `infer1` es realizando todas las combinaciones posibles. Por ejemplo, si la tupla es de dimensión k, entonces el conjunto de combinaciones sería BH^k; esto implica generar la base Herbrand y, a partir de ella, recorrerla con k ciclos `for` anidados. Proponga un mecanismo más eficiente para proporcionar los argumentos de `pe` cuando es llamado dentro de `infer1`.

E) Sea $S = \{R_1,\ R_2,\ p(a,b),\ p(a,c),\ p(a,d),\ p(b,e),\ p(b,f),\ p(c,g),\ p(c,h),\ p(d,i),\ p(d,j)\}$, con $R_1 : h(X,Y) : -p(Z,X), p(Z,Y)$ y $R_2 : pr(X,Y) : -h(Z,W), p(Z,X), p(W,Y)$. Diga cuáles de las siguientes afirmaciones son ciertas y por qué.

   a) $S \vdash pr(e,g)$

   b) $S \vdash pr(g,f)$

   c) $S \vdash pr(h,j)$

   d) $S \vdash pr(j,i)$

   e) $S \vdash pr(f,i)$

   f) $S \vdash pr(c,d)$

   g) $S \vdash pr(f,a)$

   h) $S \vdash pr(h,b)$

F) Sea S como en el problema E. Construya el árbol de demostración para las inferencias:

   a) $S \vdash pr(e,h)$

   b) $S \vdash pr(i,f)$

   c) $S \vdash pr(j,e)$

3.1.1. Solidez y adecuación

Proposición 8 (Solidez) Si S es un conjunto de cláusulas Datalog y M es un hecho básico, entonces $S \vdash M$ implica $S \models M$.

Proposición 9 (Adecuación) Si S es un conjunto de cláusulas Datalog y M es un hecho básico, entonces $S \models M$ implica $S \vdash M$.

Corolario 1 Si S es un conjunto de cláusulas Datalog y M es un hecho básico, entonces $S \models M$ sii $S \vdash M$.

Corolario 2 Si S es un conjunto de cláusulas Datalog y M es un hecho básico, entonces $cons(S) = \{M | M$ es un hecho básico y $S \vdash M\}$.

Observación 39 $cons(S)$ puede calcularse con el siguiente algoritmo: `function infer`(S)

```
//entrada: un conjunto finito de cláusulas, S.//
//salida: cons(S).//
    iniciales = ∅
    nuevas = S
    while iniciales ≠ nuevas do
        iniciales = nuevas
        nuevas = nuevas∪infer1(nuevas)
    resultado = hechosBasicos(nuevas)
    return resultado
end infer;
```

Proposición 10 `infer` es correcto.

Observación 40 Si S es finito entonces $cons(S)$ también lo es.

Ejercicio 29 Trace el valor de la variable *nuevas* del algoritmo `infer` para

1. El conjunto S del ejercicio 26.
2. El conjunto de cláusulas del ejemplo 16 añadiéndole la regla $Q : amigo(X,Y) : -amigo(Y,X)$.

Observación 41 Con el algoritmo `infer` tenemos un ejemplo de cómo obtener la respuesta a una consulta planteada a Datalog. Y, como fue advertido inicialmente, este es un método puramente formal pero inconveniente como motor de inferencias en Datalog. Ciertamente, `infer` establece una forma de abordar la solución a una consulta llamada *ascendente*; pues, como se ha visto en los árboles de demostración, se parte de nodos de profundidad cero para construir hechos de profundidad superior. Esta forma es heredada de `infer1` (y `pe`) en el cual se sigue la dirección del signo de implicación: *encadenamiento hacia adelante*. Frente a estas formas de razonamiento se han desarrollado las alternas, *descendente* (que parte de la meta) y *encadenamiento hacia atrás* (yendo en dirección contraria al símbolo de implicación en las reglas: partiendo del consecuente para obtener los antecedentes). Estos son solamente los primeros y más básicos paradigmas de evaluación que hay, en general, en programación lógica y, en particular, en Datalog.