DATALOG

CATEDRA DE BASE DE DATOS

Departamento de Computación FCEyN UBA
Gerardo Rossel

Resumen *En este apunte, basado en el libro Foundation of Databases de Abiteboul[1], describiremos una extensión de los lenguajes de consulta ya vistos, agregando recursividad y posteriormente negación. Para ello usaremos datalog. Si bien posible definir un lenguaje datalog sin recursión y con negación dicho lenguaje sería equivalente al algebra relacional. La parte interesante de datalog es la utilización de la recursividad ya que existen muchas consultas que no pueden ser resueltas sin recursividad*

Datalog es un lenguaje inspirado en el paradigma de programación lógica y el lenguaje Prolog, y es ampliamente estudiado en el campo de las bases de datos deductivas . Supongamos una base de datos que describe el subte. La misma es básicamente la representación de un grafo. Podríamos tener en esa base una relación Vinculo(Linea,Desde,Hasta) con instancias como (B, Florida, Leando N. Alem) que describe que la línea B tiende un vinculo entre las estaciones Florida y Leandro.N. Alem. Consultas tales como: ¿Cuáles son las estaciones a las que se puede llegar desde la estación Florida? ¿Se puede ir desde la estación Primera Junta a la estación Lavalle?, no pueden ser resueltas con algebra relacional. Es en realidad el problema de lograr la clausura transitiva de un grafo; para una cantidad de juntas de algebra relacional se puede siempre encontrar un grafo que tenga una distancia mayor entre nodos de modo que no pueda ser resuelta. Para ello es necesario la recursividad.

1. PROGRAMA DALALOG

Un programa datalog es básicamente un conjunto finito de reglas , cada regla tiene la forma:

$$Cabeza \leftarrow Cuerpo$$

O más formalmente:

$$R_1(u_1) \leftarrow R_2(u_1), ..., R_n(u_n), \text{ con } n \geq 1$$

Donde $R_1, ..., R_n$ son nombres de relación y $u_1, ..., u_n$ son tuplas libres de aridad apropiada

Para entenderlo mejor tomaremos un ejemplo. Intentemos responder la siguiente pregunta: ¿Cómo se calcularía la clausura transitiva de un grafo? Si usamos $G(x, y)$ para representar los arcos entre x e y (es decir que la relación G modela el grafo): la clausura transitiva sería una relación $T(x, y)$ tal que exista un camino en G entre x e y. Como primera regla podemos establecer que todas las tuplas de $G(x, y)$ pertenecen a la clausura transitiva. Eso lo notamos con la siguiente regla:

$$T(x, y) \leftarrow G(x, y)$$

Ahora necesitamos todos los pares x, y tales que haya un camino entre ellos. Hay un camino entre x e y si hay un camino entre x y z y a su vez hay un camino entre z e y. Para ello agregamos otra regla que establezca eso, quedando entonces nuestro programa datalog de la siguiente manera:

$$T(x, y) \leftarrow G(x, y)$$
$$T(x, y) \leftarrow G(x, z), T(z, y)$$

Los programas Datalog, como el del ejemplo, tienen dos tipos de relaciones:

- Relaciones extensionales: son las relaciones que aparecen solamente en el cuerpo de las reglas.

- Relaciones intencionales: son aquellas relaciones que aparecen en la cabeza de alguna regla.

Para un programa datalog P un esquema, que se nota cómo $sch(P)$, es la unión del esquema extensional y el esquema intencional ($edb(P)$ e $idb(P)$). Es decir que el esquema de un programa P datalog es la unión de todas las relaciones extensionales, $edb(P)$, con las relaciones intencionales, $idb(P)$. En algunos contextos se puede llamar datos de entrada a la base extensional y programa a la base intencional. La semántica de un programa datalog se define básicamente como un mapeo entre instancias de bases de datos sobre $edb(P)$ e instancias de bases de datos sobre $idb(P)$, es decir un mapeo entre instancias de relaciones que ocurren sólo en los cuerpos de las reglas a relaciones que ocurren en las cabezas.

1.1. Ejemplo

Consideremos nuevamente la base de datos que describe un subte con la relación $Vinculo(Lnea, Desde, Hasta)$. Vamos a realizar un programa $Psubte$ que responda a las siguientes consultas:

- ¿Cuáles son las estaciones a las que se puede llegar desde la estación Florida?
- ¿A cuales líneas puede llegarse desde Florida?
- ¿Se puede ir desde la estación Florida a la estación Lavalle?

El programa es el siguiente:

$$
\begin{aligned}
&EstacionAlcanzable(x,x) \leftarrow \\
&EstacionAlcanzable(x,y) \leftarrow EstacionAlcanzable(x,z), Vinculo(u,z,y) \\
&LineaAlcanzable(x,u) \leftarrow EstacionAlcanzable(x,z), Vinculo(u,z,y)
\end{aligned}
$$

$$
\begin{aligned}
&Ans_1(y) \leftarrow EstacionAlcanzable(Florida, y) \\
&Ans_2(u) \leftarrow LineaAlcanzable(Florida, u) \\
&Ans_3() \leftarrow EstacionAlcanzable(Florida, Lavalle)
\end{aligned}
$$

¿Cuáles serían $edb(Psubte)$ e $idb(Psubte)$?

$$
\begin{aligned}
edb(Psubte) &= \{Vinculo\} \\
idb(Psubte) &= \{EstacionAlcanzable, Ans_1, Ans_2, Ans_3\}
\end{aligned}
$$

2. Progración Lógica y Datalog

Hay algunas diferencias entre la programación lógica y Datalog. La diferencia más notoria es que la programación lógica permite la utilización de símbolos de función o *functores* dentro de las reglas. Dada la semántica de los símbolos de función en programación lógica es posible utilizarlos para definir estructuras complejas (listas, árboles). En las bases de datos deductivas se espera gran cantidad de hechos definidos en relaciones y una cantidad de reglas relativamente pequeña. En programación lógica por otra parte los hechos se incorporan en general en el propio programa mediante reglas.

3. Semántica de los programas Datalog

Se pueden dar tres diferentes y equivalentes enfoques para la semántica de programas datalog: model-theoric, proof-theoric y semántica del punto fijo. Comenzaremos con la primera.

3.1. Enfoque Model-Theoric

Básicamente este enfoque ve un programa datalog cómo un conjunto de sentencias de primer orden (cada regla tiene una sentencia asociada). En realidad como un conjunto especial de sentencias de primer orden llamadas clausulas de Horn. Una clausula es un disyunción finita de literales que puede consistir de un solo literal en cuyo caso se llama cláusula unitaria. Un tipo especial de cláusulas son las denominadas cláusulas de Horn. La importancia de este tipo de cláusulas residen en su eficiencia computacional para inferir sobre ellas por sobre las cláusulas comunes. La aplicación de refutación por resolución en cláusulas de Horn es un mecanismo ampliamente utilizado. *Una Cláusula de Horn es una cláusula con a lo sumo un literal positivo.*

Entonces un programa datalog es un conjunto de clausulas de Horn que describen la respuesta esperada. De esta forma una instancia de la base de datos constituye el resultado que satisface las sentencias. Dicha instancia se dice que es un modelo de las sentencias.

Cómo puede haber muchas instancias que satisfagan las sentencias, las mismas no pueden por sí solas identificar la respuesta esperada, es necesario especificar el modelo que sea la respuesta esperada. Pero, ¿qué es un modelo?

Sea un programa datalog P y una instancia $\mathbf{I}$ sobre $edb(P)$. Un modelo de P es una instancia sobre $sch(P)$ que satisface el conjunto de clausulas asociadas a las reglas de P (que se denota como $\sum_P$) . Se da que la semántica de P sobre I, que denotamos $P(\mathbf{I})$ es un modelo minimal de P conteniendo $\mathbf{I}$, si es que este existe.

Tomar el modelo *minimal* está relacionado con la suposición del *mundo cerrado*. Si bien se puede suponer que todo hecho en la base de datos es verdadero no hay forma de saber algo acerca de lo que no se encuentra en ella. La suposición del mundo cerrado justamente asume que todos los hechos que no están en la base de datos son falsos, es decir se asume completa. Si se la asume completa entonces son verdaderos sólo los hechos que son verdaderos en todos los mundos modelados por la base de datos. Por eso se escoge el modelo minimal, el cual consiste de todos los hechos que deben ser verdaderos en todos los mundos que satisfacen las sentencias.

3.2. Semántica del Punto Fijo

La semántica del punto fijo nos da una semántica operacional de los programas datalog. El punto fijo de una función f es un valor v tal que la función aplicada al valor devuelve el mismo resultado. O sea $f(v) = v$.

Para la ecuación de punto fijo que necesitamos usamos un operador llamado: *operador de consecuencia inmediata*. Este operador permite obtener nuevos hechos a partir de hechos existentes y para ello utiliza las reglas del programa datalog. La semántica de *model-theoric*, $P(\mathbf{I})$, definida anteriormente también pude verse como la solución más pequeña de la ecuación de punto fijo con el operador de consecuencia inmediata.

Definamos mas formalmente el operador de consecuencia inmediata sobre un programa P al que notamos T_P. Para un instancia $\mathbf{K}$ sobre $sch(P)$ se dice que un hecho A es consecuencia inmediata de $\mathbf{K}$ y P si:

1. $A \in K(R)$ para alguna relación R perteneciente a $edb(P)$, o
2. $A \leftarrow A_1, ..., A_n$ es una instanciación de una regla en P y cada $A_i \in K$

El operador de consecuencia inmediata se define de la siguiente manera: para una instancia $\mathbf{K}$, $T_P(\mathbf{K})$ consiste de todos los hechos A que son consecuencias inmediatas para $\mathbf{K}$ y P . Notar que el operador se define sobre conjuntos de instancias. El operador cuenta dos propiedades interesantes:

1. Es monótono, es decir si para $\mathbf{I}, \mathbf{J}, \mathbf{I} \subseteq \mathbf{J}$ entonces $T(\mathbf{I}) \subseteq T(\mathbf{J})$.
2. $\mathbf{K}$ es un *punto fijo* de T si $T(\mathbf{K}) = \mathbf{K}$

Se puede demostrar que para cada programa P y cada interpretación $\mathbf{I}$, T_P tiene un punto fijo minimal conteniendo $\mathbf{I}$ el cual es igual a $P(\mathbf{I})$.

3.3. Semántica Proof-Theoretic

El tercer enfoque a la semántica de un programa datalog está basado en pruebas. La respuesta a un programa P sobre una interpretación **I** consiste del conjunto de hechos que pueden ser probados usando$P$ e **I**. El resultado es coincidente con P(**I**).

Para demostrar o probar un hecho A se utiliza el árbol de demostración. Un árbol de demostración de A desde **I** y **P** es un árbol etiquetado donde:

1. Cada vértice del árbol es etiquetado con un hecho
2. Cada hoja es etiquetada con un hecho en **I**
3. La raíz es etiquetada cómo A
4. Para cada vértice interno existe una instanciación de una regla $A_1 \leftarrow A_2, ..., A_n$ en P tal que el vértice es etiquetado cómo A_1 y sus hijos son etiquetados respectivamente como $A_2, ..., A_n$.

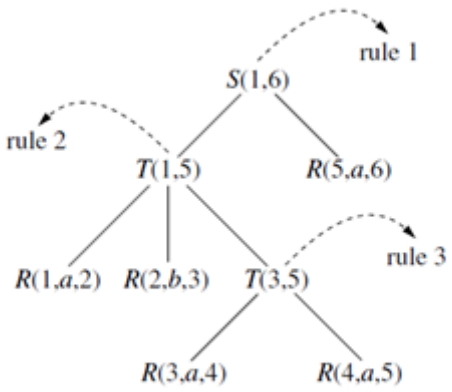
Se dice que el árbol que acabamos de definir provee una demostración para A.
Dado el siguiente programa:

$$S(x_1, x_3) \leftarrow T(x_1, x_2), R(x_2, a, x_3)$$
$$T(x1, x4) \leftarrow R(x_1, a, x_2), R(x_2, b, x_3), T(x_3, x_4)$$
$$T(x1, x3) \leftarrow R(x_1, a, x_2), R(x_2, a, x_3)$$

Y la instancia:

$$\{R(1, a, 2), R(2, b, 3), R(3, a, 4), R(4, a, 5), R(5, a, 6)\}$$

El árbol de demostración de $S(1, 6)$ es:



4. Datalog y negación

Existen consultas que no pueden ser resueltas sin la incorporación de la negación. Por ejemplo consultas cómo ¿Cuáles son los pares de estaciones de subte que **NO** están conectadas por una línea?, son programas que necesitan de la negación. Es necesario para el agregado de la negación definir la semántica de la misma. La elección de la semántica depende del punto de vista con el cual se ve al programa datalog. Sintácticamente es sencillo agregar la negación, por ejemplo en el caso de la clausura transitiva de un grafo teníamos el programa

$$T(x, y) \leftarrow G(x, y)$$
$$T(x, y) \leftarrow G(x, z), T(z, y)$$

Si quisiéramos obtener el complemento, es decir los pares (x,y) que no estén conectados podríamos definir el siguiente programa.

$$CT(x,y) \leftarrow \neg T(x,y).$$

El símbolo $\neg$ es la negación, diríamos que son los pares (x,y) que no están en $T(x,y)$.Tenemos entonces el lenguaje datalog con negación, *datalog*$\neg$, que se define por permitir literales de la forma $\neg R_i(u_i)$ en los cuerpos de las reglas, de tal manera que R_i es el nombre de una relación y u_i es una tupla libre. La negación siempre se ubica en los cuerpos de las reglas y nunca en la cabeza.

La introducción de la negación permite a Datalog contar con un conjunto de predicados *built-in* que pueden ser usados en la construcción de reglas. Hay predicados binarios de comparación sobre dominios ordenados: $<$ (less), $\leq$ (less_ or_equal),$>$ (greater), y $\geq$ (greater_or_equal). También predicados de comparación sobre dominios no ordenados:$=$ (equal) y $\neq$ (not_equal). Pueden usarse en notación infija o no, por ejemplo $less(3,4)$ y $3 <= 4$ son equivalentes.

Desafortunadamente no es posible extender las semánticas vistas (model-theoretic, punto fijo y proof-theoretic) tan fácilmente como la sintaxis. Por lo tanto veremos brevemente en qué consisten dos semánticas para programas datalog$\neg$. La semántica estratificada (*stratified*) y la semántica bienfundada (*well founded*).

4.1. Semántica estratificada para *datalog*$\neg$

En primer lugar consideraremos un *datalog*$\neg$ semipositivo. Este datalog es aquel en el cual la negación sólo puede ser aplicada a relaciones de *edb*. En este caso la diferencia entre dos relaciones R' y R puede definirse de la siguiente manera:

$$Diff(x) \leftarrow R(x), \neg R'(x)$$

La semántica que le damos a $\neg R'(x)$ utiliza suposición del mundo cerrado donde $\neg R'(x)$ es verdad si y sólo si x está en el dominio activo y $x \notin R$. Al ser R una relación *edb* su contenido depende de la base de datos. Por ejemplo para el caso del complemento de la clausura del grafo no podríamos usar:

$$CT(x,y) \leftarrow \neg T(x,y)$$

No se puede usar porque en dicho caso $T(x,y)$ no es miembro de *edb*. Para solucionarlo podemos escribir el siguiente programa:

$$CT(x,y) \leftarrow \neg G(x,y)$$
$$CT(x,y) \leftarrow \neg G(x,z), CT(z,y)$$

$G(x,y)$ es miembro de *edb* por lo el programa cumple con ser *semipositivo*. El programa dado calcula la clausura transitiva para el complemento de G. Notar que un programa semipositivo podría transformarse en positivo si se escribe otra relación que contenga el complemento, en el dominio activo, de la relación negada.

Los programas semipositivos se basan en restringir la negación a las relaciones de *edb*. Consideremos ahora que como hemos visto un programa puede definir una relación mediante reglas. Una vez que esa relación ha sido definida puede ser usada por otro programa cómo si fuera una relación de *edb*. En dicho caso sería posible aplicar la negación a esta relación definida. Esta es la idea subyacente de una extensión a los programas semipositivos, los *programas estratificados*.

Para un programa *datalog*$\neg$ P cada relación de *idb* es definida por una o más reglas de P. Si el programa puede ser leído de tal manera que para cada relación R negada de *idb* la parte del programa que la define está *antes sintácticamente* de la negación entonces es posible calcular R previamente a que sea negada. El programa que calcula el complemento de la clausura transitiva de un grafo puede ser escrito de la siguiente manera:

$$T(x,y) \leftarrow G(x,y)$$
$$T(x,y) \leftarrow G(x,z), T(z,y).$$
$$CT(x,y) \leftarrow \neg T(x,y).$$

En este caso T es definido antes de que sea negado en la regla que define CT. Esta manera de leer P es llamada ***estratificación*** de P. Es decir una estratificación es un re-ordenamiento de las reglas de tal manera que todas las relaciones de *idb* sean definidas antes de su uso negado.

Formalizamos un poco la idea:

Una estratificación de un programa *datalog*¬ P es una secuencia de programas *datalog*¬: $P^1, ..., P^n$ tal que para algún mapeo σ desde $idb(P)$ a $[1..n]$ ocurre:

i. $P^1, ..., P^n$ es una partición de P

ii. Para cada predicado R todas las reglas en P que definen a R están en $P^{\sigma(R)}$

iii. Si $R(u) \leftarrow ...R'(v)....$ es una regla en P y R' es una relación en *idb* entonces $\sigma(R') <= \sigma(R)$

iv. Si $R(u) \leftarrow ...\neg R'(v)....$ es una regla en P y R' es una relación en *idb* entonces $\sigma(R') <= \sigma(R)$

Para una estratificación $P^1, ..., P^n$ de P cada P^i se denomina estrato de la estratificación y σ se llama mapeo de la estratificación. Lo que se consigue es una forma de parsear una programa *datalog*¬ como una secuencia de subprogramas. Es importante notar que no todo programa *datalog*¬ puede ser estratificado. Un programa que tenga la siguiente forma no es posible que sea estratificado: $p \leftarrow \neg q, q \leftarrow \neg p$. Claramente no hay manera de escribirlo de tal forma que se pueda definir q antes que p y viceversa.

4.1.1. Ejemplo

Encontrar las estratificaciones del siguiente programa:

$$r1 : S(x) \leftarrow R'_1(x), \neg R(x)$$
$$r2 : T(x) \leftarrow R'_2(x), \neg R(x)$$
$$r3 : U(x) \leftarrow R'_3(x), \neg T(x)$$
$$r4 : V(x) \leftarrow R'_4(x), \neg S(x), \neg U(x)$$

Tenemos como resultado las siguientes estratificaciones:

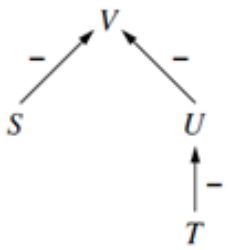
$\{r1\}, \{r2\}, \{r3\}, \{r4\}$
$\{r2\}, \{r1\}, \{r3\}, \{r4\}$
$\{r2\}, \{r3\}, \{r1\}, \{r4\}$
$\{r1, r2\}, \{r3\}, \{r4\}$
$\{r2\}, \{r1, r3\}, \{r4\}$

4.1.2. Verificar si un programa es estratificable

Cómo mencionamos un programa de la forma $p \leftarrow \neg q, q \leftarrow \neg p$ no es estratificable. En este caso se ve un ciclo entre definición y el uso de la negación. La misma idea puede ser aplicable a cualquier programa evitando ciclos de ese tipo. Definimos entonces el grafo de precedencia GP de un programa P. Un grafo de precedencia de un programa P es un grafo etiquetado tal que sus nodos son relaciones en idb y sus ejes cumplen las siguientes condiciones.

1. Si $R(u) \leftarrow ...R'(v)....$ es una regla en P entonces $< R', R >$ es un eje en GP con etiqueta $+$
2. Si $R(u) \leftarrow ...\neg R'(v)....$ es una regla en P entonces $< R', R >$ es un eje en GP con etiqueta $+$. Se denomina a ese eje: *eje negativo*

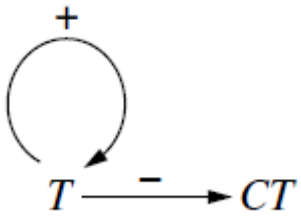
El grafo del programa del ejemplo 4.1.1 es



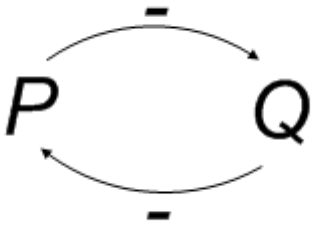
Veamos el grafo del programa del complemento de la clausura de un grafo cuyo programa es:

$$T(x,y) \leftarrow G(x,y)$$
$$T(x,y) \leftarrow G(x,z), T(z,y)$$
$$CT(x,y) \leftarrow \neg T(x,y).$$

El grafo sería:



Un grafo de un programa que tenga la forma $p \leftarrow \neg q, q \leftarrow \neg p$ sería:



Un lema importante es el siguiente: Sea un programa P con estratificación σ. Si hay un camino desde R' a R en GP entonces $(R') \leq \sigma(R)$ y si hay un camino desde R' a R en GP conteniendo algún eje negativo entonces $(R') < \sigma(R)$.

Este lema nos permite usar el grafo de precedencia para verificar si un programa datalog es estratificable o no.

> Un programa *datalog*¬ es estratificable **si y sólo si** su grafo de precedencia GP no tiene ciclos conteniendo un eje negativo.

4.1.3. Semántica de programas estratificados

La forma de dar semántica a un programa estratificable es utilizar programas semipositivos. Podemos considerar que dada una estratificación de $P : \sigma = P^1, ..., P^n$ puede considerarse que la $edb(P^i)$ puede contener alguna relación $idb(P)$ definida en un estrato menor. Si usa la negación de una relación R entonces dicha relación pertenece a $edb(P^i)$ ya que o bien es parte de $edb(P)$ o bien fue definida antes (en algún P^j que precede a P^i) por lo cual puede considerarse miembro de $edb(P^i)$. Entonces puede considerarse que cada programa P^i es semipositivo con respecto a las relaciones definidas previamente. La semántica de P es obtenida por aplicar, en orden, los programas P^i.

Sea $\mathbf{I}$ una instancia sobre $edb(P)$, se puede definir una secuencia de instancias de la siguiente manera:

- $\mathbf{I}0 = \mathbf{I}$.
- $\mathbf{I}i = \mathbf{I}i-1 \cup P^i(\mathbf{I}i-1 \mid edb(P^i)), 0 < i \leq n$.

En dicha secuencia cada $\mathbf{I}i$ extiende a $\mathbf{I}i-1$ proveyendo valores a las relaciones definidas en P^i. $P^i(\mathbf{I}i-1 \mid edb(P^i))$ es la semántica del programa semipositivo P^i aplicado a los valores de sus relaciones

edb provistas por $\mathbf{I}i-1$. La instancia final $\mathbf{I}n$ la denotamos: $\sigma?(\mathbf{I})$. Esto nos provee una semántica de un programa *datalog*¬ bajo la estratificación σ.

Un aspecto importante, que no demostraremos aquí, es que si bien un programa puede tener diferentes estratificaciones dichas estratificaciones producen la misma semántica. Por lo tanto podemos hablar de la semántica de un programa *datalog*¬ estratificable para una entrada **I**. A dicha semántica la notamos $P^{strat}(\mathbf{I})$ y es la semántica de cualquier estratificación σ de P. $P^{strat}(\mathbf{I})$ puede ser computada en tiempo polinomial respecto a **I**.

Dos propiedades nos permiten vincular está semántica con lo visto anteriormente:

1. $P^{strat}(\mathbf{I})$ es un modelo minimal de of $\sum_P$ (el conjunto de clausulas asociadas) cuya restricción a $edb(P)$ es igual a **I**.

2. $P^{strat}(\mathbf{I})$ es un punto fijo minimal de TP cuya restricción a $edb(P)$ es igual a **I**.

4.2. Semántica bien fundada

La semántica *well-founded* se basa en que las respuestas que hasta aquí eran verdadera o falsa sobre cualquier hecho pueden tener otro valor: desconocida (*unknown*). Este cambio de expectativa en cuanto a la respuesta es el precio que pagamos para tener una semántica natural para todo programa *datalog*¬.

Este cambio pone en igualdad de condiciones a los hechos negativos y positivos. Ahora no se puede decir que $\neg R(u)$ es verdadero simplemente porque $R(u)$ no es una respuesta. Tanto los hechos positivos como los negativos deben ser inferidos.

La forma de formalizar lo que venimos diciendo es proveyendo de tres valores de verdad (3-valued) a los hechos: verdadero, falso y desconocido (true, false, unknown).

4.2.1. Semántica declarativa

Dar una semántica a un programa *datalog*¬ P,es encontrar un modelo apropiado **I**, *3-valuado*, del conjunto de clausulas asociadas a las reglas de P ($\sum_p$). Que sea apropiado implica que haya consistencia en el proceso de razonamiento, específicamente que no se pueda usar un hecho y luego inferir su negación. Un modelo 3-valuado es llamado **3-estable** si satisface la siguientes puntos (que indican que es apropiado):

- Los hechos positivos de **I** deben ser inferidos de P asumiendo los hechos negativos en **I**
- Todos los hechos negativos que pueden ser inferidos de **I** ya deben estar en **I**.

Podemos denotar los tres valores posibles de la siguiente manera:

$true = 1$, $false = 0$ y $unknown = 1/2$.

Llamamos a $P_{\mathbf{I}}$ al programa resultante de agregar a P las clausulas unitarias que establecen que los hechos en **I** son verdaderos, a su vez $\mathbf{B}(P)$ denota todos los hechos de la forma $R(a_1,...,a_k)$, donde R es una relación y $a_1,...,a_k$ son constantes que ocurren en P. En particular $\mathbf{B}(P_{\mathbf{I}}) = \mathbf{B}(P,\mathbf{I})$.

Una instancia *3-valuada* **I** sobre $sch(P)$ es un mapeo total desde $\mathbf{B}(P)$ a $\{0, 1/2, 1\}$ podemos denotar como $\mathbf{I}^1, \mathbf{I}^{1/2}, \mathbf{I}^0$ al conjunto de átomos cuyo valor de verdad es $1, 1/2$ y 0 respectivamente. Una instancia es total o *2-valuada*, si $\mathbf{I}^{1/2} = \emptyset$.

Tenemos un orden natural $\prec$, entre instancias *3-valuadas* sobre $sch(P)$ que se define como:

- $\mathbf{I} \prec \mathbf{J}$ si y sólo si cada $A \in \mathbf{B}(P), \mathbf{I}(A) \leq \mathbf{J}(A)$.

Debemos definir los valores de verdad para los conectivos lógicos ahora con una lógica *3-valuada*. El valor de verdad de una combinación α de hechos que notamos $\hat{\mathbf{I}}(\alpha)$se define de la siguiente manera

1. $\hat{\mathbf{I}}(\beta \wedge \gamma) = min\ \{\hat{\mathbf{I}}(\beta), \hat{\mathbf{I}}(\gamma)\}$
2. $\hat{\mathbf{I}}(\beta \vee \gamma) = max\ \{\hat{\mathbf{I}}(\beta), \hat{\mathbf{I}}(\gamma)\}$
3. $\hat{\mathbf{I}}(\neg\beta) = 1 - \hat{\mathbf{I}}(\beta)$

4. $\hat{\mathbf{I}}(\beta \leftarrow \gamma) = 1$ si $\hat{\mathbf{I}}(\gamma) \leq \hat{\mathbf{I}}(\beta)$, en otro caso: 0.

Entonces se dice que una instancia $\mathbf{I}$ sobre $sch(P)$ satisface una combinación α de átomos en $\mathbf{B}(P)$ si y sólo si:

- $\hat{\mathbf{I}}(\alpha) = 1$.

Dado un programa *datalog*¬ P un modelo *3-valuado* de $\sum_P$ es un instancia 3-valuda sobre $sch(P)$ que satisface el conjunto de implicaciones correspondientes las reglas $ground(P)$ (recordar que un clausula ground es aquella en la que no hay ocurrencia de variables).

Para un programa datalog (sin negación) P se puede extender la definición de semántica para instancias 3-valuadas. De tal forma que dicha semántica sea el mínimo modelo 3-valuado de P.

Para comprender que significa modelo mínimo definimos un operador de consecuencia inmediata como para los programas 2-valuados que llamamos operador de consecuencia inmediata 3-valuado: 3-T_P. Dado una instancia $\mathbf{I}$ 3-valuada y $A \in B(P)$ entonces 3-$T_P(A)$ se define de la siguiente manera:

- 1 si hay una regla $A \leftarrow cuerpo$, en $ground$(P) tal que $\hat{\mathbf{I}}(cuerpo) = 1$
- 0 si para cada regla $A \leftarrow cuerpo$ en $ground$(P) $\hat{\mathbf{I}}(cuerpo) = 0$ y en particular si no hay ninguna regla con A en la cabeza.
- $1/2$ de otra manera.

Para obtener el modelo minimal utilizamos el orden $\prec$ definido anteriormente. Es sencillo notar que la mínima instancia 3-valuada con respecto a $\prec$ es aquella donde todos los átomos son falsos. Dicha instancia la denotamos con $\perp$. Podemos entonces definir una secuencia a partir de $\perp$ aplicando 3-T_P. Veamos el siguiente ejemplo para que sea más claro:

P es el siguiente programa $\{p \leftarrow 1/2; p \leftarrow q, 1/2; q \leftarrow p, r; q \leftarrow p, s; s \leftarrow q; r \leftarrow 1\}$. 3-$T_P(\perp)$ es 3-$T_P(\{\neg p, \neg q, \neg r, \neg s\})$ entonces nos queda:

1. 3-$T_P(\{\neg p, \neg q, \neg r, \neg s\}) = \{\neg q, r, \neg s\}$
2. 3-$T_P(\{\neg q, r, \neg s\}) = \{r, \neg s\}$
3. 3-$T_P(\{r, \neg s\}) = \{r\}$
4. 3-$T_P(\{r\}) = \{r\}$

Podemos enunciar el siguiente lema:

- 3-T_P es monótono y la secuencia 3-$T_P{}^i(\perp)_{i>0}$ es incremental y converge al punto fijo minimal de 3-T_P
- P tiene un único modelo 3-valuado que es igual al punto fijo minimal de 3-T_P

Dicho modelo minimal es la semántica de un programa *datalog* extendido 3-valuado. Es análoga al datalog convencional 2-valuado y la denotamos como $P(\perp)$.

REFERENCIAS

[1] Abiteboul, Richard Hull, Victor Vianu (1995). Foundation of Databases Foundations of Databases: The Logical Level.