PARTE II
BASES DE DATOS DEDUCTIVAS

0. INTRODUCCIÓN

Los *Sistemas de Bases de Datos Deductivas*, llamados también *de Sistemas de Bases de Conocimiento*, generalizan los sistemas tradicionales de bases de datos, particularmente los sistemas relacionales. Una base de datos deductiva se compone de dos tipos de información: hechos y reglas. Los hechos representan la información almacenada explícitamente y constituyen la parte extensional de la base de datos; las reglas deductivas permiten derivar información a partir del conjunto de hechos y constituyen la parte intensional de la base de datos. La incorporación de un lenguaje de reglas que permite definir información implícita, aumenta la capacidad expresiva de estos sistemas, pero al mismo tiempo agrava los problemas de manipulación, ya existentes en los sistemas clásicos: actualización de la base de datos, comprobación de la integridad, evaluación de consultas, etc. En la solución de estos problemas, la lógica, y más concretamente la programación lógica, ha ocupado un lugar importante, proporcionando una base para la representación del conocimiento (semántica declarativa) y una base para su procesamiento (semántica procedural) [Min88], [GM92], [Llo90].

Los primeros trabajos en el campo de las bases de datos deductivas, fueron presentados en el *1er Workshop on Logic and Databases* que tuvo lugar en Toulouse en 1977 [GM78]. Algunos años más tarde, Reiter hizo un primer intento de elaborar una fundamentación teórica de las bases de datos deductivas [Rei84]; y Gallaire, Minker y Nicolas [GMN84a] presentaron el primer resumen que aparece en el área. Desde un principio se vio clara la estrecha relación existente entre las bases de datos deductivas y la programación lógica; los primeros trabajos donde se establece esta relación fueron presentados por Lloyd y Topor, [LT85], [LT86]. En los años siguientes, el área se desarrolló rápidamente y aparecieron las primeras implementaciones de sistemas de gestión de bases de datos deductivas [GMS85], [ChW86], [CFK86], [SR86], [BDN86], [MUvG86], [Zan88]. Una historia detallada de los desarrollos en este campo aparece en "Perspectives in Deductive Databases" de Minker [Min88b]. En la actualidad existen ya textos donde se puede encontrar los fundamentos y resultados más importantes en este área, estos son [D92] y [CGT90].

1. Bases de datos Deductivas

1.1 Introducción

Las bases de datos deductivas extienden la capacidad expresiva de las bases de datos relacionales, incorporando reglas que permiten derivar información a partir de la información almacenada explícitamente.

En el esquema de una base de datos deductiva se distinguen dos tipos de relaciones: **relaciones básicas** cuyas tuplas se almacenan explícitamente y **relaciones derivadas** definidas por reglas a partir de relaciones básicas y de otras relaciones derivadas. De acuerdo con este esquema los estados de la base de datos están formados por **hechos**, tuplas de las relaciones básicas, que constituyen la parte extensional de la base de datos y **reglas deductivas** que definen las relaciones derivadas y constituyen la parte intensional de la base de datos.

La incorporación de la capacidad deductiva introduce nuevos problemas en la construcción de sistemas de gestión de bases de datos:

a) el sistema debe proporcionar un lenguaje de definición de reglas. Este lenguaje puede ser una extensión del lenguaje de definición de vistas de los sistemas relacionales que incorpore la posibilidad de definir relaciones recursivas.

b) debe asociarse una semántica a la base de datos, es decir debe definirse cual es la información "derivable" a través de las reglas deductivas. En presencia de negación y de relaciones recursivas la extensión asociada a las relaciones derivadas puede no ser única.

c) debe elegirse una representación para el almacenamiento de las reglas deductivas.

d) el mecanismo de evaluación de consultas debe ser capaz de manipular relaciones recursivas, e información incompleta (negación). Cuando el número de reglas y hechos es elevado, la eficiencia del mecanismo de evaluación es uno de los problemas más importantes.

La solución de todos estos problemas exige una definición formal de los mismos apoyada en una sólida fundamentación teórica de las bases de datos deductivas. En esta línea todos los trabajos teóricos se han basado en la lógica [ChL73], [Gal86] [Ham81].

La lógica de primer orden ya había sido utilizada en el desarrollo del modelo relacional de datos [Dat93], [Ull80] desde su aparición en 1970 [Cod70]. Los problemas más relevantes

que la lógica ha ayudado a estudiar y algunos de los primeros trabajos en este área se exponen a continuación:

- formalización [NG78], [Kow78], [Kow81], [GMN84a], [Rei84]: un estado de una base de datos relacional puede formalizarse en lógica de primer orden como una interpretación de un lenguaje de primer orden o como una teoría de primer orden.

- definición de lenguajes de interrogación [Cod72]: cálculo relacional de tuplas y cálculo relacional de dominios.

- definición del concepto de independencia del dominio y caracterización de clases de fórmulas que cumplan dicha propiedad: fórmulas de rango separable [Cod72], fórmulas de rango restringido [Nic82], fórmulas seguras [Ull80], ...

- formulación y evaluación simplificada de restricciones de integridad: [Nic79], [Nic82], [NY78].

- optimización de consultas: uso de criterios de simplificación sintácticos [ChM76], uso de información estadística [Dem80], uso de criterios de simplificación semánticos [Kin81].

- diseño de bases de datos: especificación formal [VCF81], [BW81], definición y análisis de dependencias funcionales [Fag82].

La idea básica que subyace en el uso de la lógica para el estudio de los sistemas de bases de datos es una idea común a todos los campos de la computación lógica: "la semántica por **teoría de modelos** de la lógica puede proporcionar una base para la representación del conocimiento, y la semántica por **teoría de la demostración** puede proporcionar una base para la computación" [Llo90].

La lógica es utilizada en el desarrollo de los sistemas de gestión de bases de datos deductivas en dos sentidos:

a) Desde un punto de vista declarativo:

- se va a utilizar un único lenguaje (lenguaje de primer orden) para la definición de datos, consultas y restricciones de integridad, en este sentido ya no va a ser necesario el uso de *lenguajes huésped* para el desarrollo de aplicaciones de bases de datos.

- se va a asociar una semántica declarativa a la base de datos que defina la información implícitamente almacenada. Esta semántica va a permitir determinar las respuestas correctas a una consulta.

b) Desde un punto de vista operacional:

- los desarrollos en el campo de la demostración automática van a ser utilizados para la implementación de sistemas de gestión de bases de datos deductivas.

1.2 Formalización lógica de una base de datos relacional

1.2.1 Modelo Relacional de Datos [Cod70]

Estructuras del Modelo:

Tupla:

***** Esquema de tupla: conjunto de pares atributo-dominio:

$$\tau = \{(A_1:D_1), (A_2:D_2),...(A_n:D_n)\}$$

***** Tupla t de esquema $\tau = \{(A_1:D_1), (A_2:D_2),...(A_n:D_n)\}$: conjunto de pares atributo-valor:

$$t = \{(A_1:D_1), (A_2:v_2),...(A_n:v_n)\} \text{ tal que } v_i \in D_i.$$

Relación:

* Esquema de relación: conjunto de pares atributo-dominio:

$$\rho = \{(A_1:D_1), (A_2:D_2),...(A_n:D_n)\}$$

***** Relación R de esquema $\rho = \{(A_1:D_1), (A_2:D_2),...(A_n:D_n)\}$: conjunto de tuplas de esquema ρ

Mecanismos para expresar restricciones de integridad:

- definición de dominios
- definición de claves candidatas (clave primaria, claves alternativas)
- definición de claves ajenas

Lenguajes del modelo:

- Álgebra relacional
- Cálculo Relacional de Tuplas
- Cálculo Relacional de Dominios

De la definición de tupla que se ha dado, se deduce que en una tupla t no hay un orden definido entre sus componentes, referenciándose cada uno de ellos por el nombre del

correspondiente atributo ($t(A_i)$). Sin embargo, como es sabido, en un lenguaje de primer orden todo predicado lleva asociada una aridad, y sus argumentos se referencian por su posición relativa en la correspondiente fórmula atómica; por este motivo y por comodidad al hacer la formalización lógica, vamos a introducir un orden en el conjunto de atributos del esquema de una relación (biyección entre los atributos y el subconjunto de los naturales {1..n}), de forma que las tuplas pasen a ser conjuntos ordenados de valores cuyos componentes se referencien por su posición relativa. De esta forma la estructura relación del modelo relacional coincide con el concepto matemático de relación, y los conceptos de esquema de la base de datos y ocurrencia del esquema (o base de datos) pueden definirse de la forma:

• **esquema de base de datos**: consta por una parte, de un conjunto de esquemas de relación de la forma R(A_1: D_1, A_2: D_2,..., A_n: D_n) donde **R** es el nombre de la relación, $\mathbf{A_j}$ ($1 \leq j \leq n$) es el nombre del atributo jotaésimo de la relación R y $\mathbf{D_j}$ ($1 \leq j \leq n$) es el dominio asociado a ese atributo; y por otra parte, de un conjunto de restricciones de integridad que son reglas que permiten expresar propiedades semánticas de los datos. Los subíndices ($1 \leq j \leq n$) indican el orden definido sobre los atributos.

• **ocurrencia de un esquema** (extensión del esquema o simplemente base de datos): es un conjunto de relaciones R, donde una relación R es un subconjunto del producto cartesiano de los dominios del esquema de R, es decir $R \subset D_1 \times D_2 \times ... \times D_n$. Para que una ocurrencia de un esquema sea válida debe satisfacer el conjunto de restricciones de integridad del esquema.

Ejemplo 1.1

Esquema:

Dominios:

* Dom_P= { A, B, C }
* Dom_A ={ a, b, c, d }
* Dom_C = { CS100, CS200, P100, P200 }

Relaciones:

* Prof (cod_prof : Dom_P)
  CP: cod_prof

* Alumno (cod_alum : Dom_A)
  CP: cod_alum

* Curso (cod_curso : Dom_C)
  CP: cod_curso

* Enseñar (cod_prof: Dom_P, cod_curso: Dom_C)
  CP: (cod_prof, cod_curso)
  CAj: cod_prof → Prof
  cod_curso→ Curso

* Matriculado:(cod_alum: Dom_A, cod_curso: Dom_C)
  CP: (cod_alum, cod_curso)
  CAj: cod_alum → Alumno
  cod_curso → Curso

Restricciones:

* "Todo profesor imparte algún curso"
* "Todo curso es impartido por algún profesor"

Base de Datos

Prof

A
B
C

Alumno

a
b
c
d

Curso

CS100
CS200
P100
P200

Enseñar

A	CS100
A	CS200
B	P100
C	P200

Matriculado

a	CS100
a	CS200
b	CS100
c	P100
d	CS100
d	P200

1.2.2 Formalización lógica de una base de datos relacional como una interpretación de un lenguaje de primer orden

El primer paso para formalizar una base de datos relacional como una interpretación de un lenguaje de primer orden, es la definición de dicho lenguaje a partir del esquema de la base de datos para, posteriormente, definir la interpretación a partir de su extensión:

Esquema = (L, RI) donde :

- L es un lenguaje de primer orden
- RI es el conjunto de restricciones de integridad, fórmulas cerradas de L

Base de Datos = Interpretación de L que es modelo de RI

Definición del lenguaje a partir del esquema de la base de datos:

Sea $\mathbb{D}$ la unión de todos los dominios que aparecen en el esquema de la base de datos entonces, los símbolos de constantes y de predicados del lenguaje son los siguientes:

• *Constantes*: por cada elemento de $\mathbb{D}$, se introduce un símbolo de constante y nada más que uno. Para simplificar, se escogen como símbolos de constantes los símbolos que denotan a los elementos de $\mathbb{D}$.

• *Predicados*: por cada esquema de relación n-aria, R_i en el esquema de la base de datos, se introduce un símbolo de predicado n-ario. Para simplificar, se escogerán como símbolos de predicado los nombres de las relaciones. Por razones que se justificarán posteriormente, se incluye el predicado =, que se interpretará como la igualdad.

En este alfabeto, por simplicidad, no se incluyen los símbolos de función. Los símbolos de variables, las conectivas lógicas, los símbolos especiales y los cuantificadores son los propios de un lenguaje de primer orden. Las fbfs de este lenguaje se construyen de la forma usual.

Definición de la Interpretación a partir de la extensión de la base de datos:

• El dominio de la interpretación es $\mathbb{D}$.

• Asignación a los símbolos de constantes: a cada símbolo de constante se le asigna el elemento de $\mathbb{D}$ que viene denotado por dicho símbolo.

• Asignación a los símbolos de predicados: a cada símbolo de predicado se le asigna la extensión de la relación de la base de datos cuyo nombre coincide con ese símbolo.

En el ejemplo 1.1:

Esquema = (L, RI) donde

L es el lenguaje de primer orden:

Constantes = {A, B, C , a, b, c, d, CS100, CS200, P100, P200}
Predicados ={Prof, Alumno, Curso, Enseñar, Matriculado}

RI es el conjunto de restricciones de integridad:

$\forall x$ (Prof(x) $\rightarrow \exists y$ (Enseñar(x, y))
$\forall x$ (Curso(x) $\rightarrow \exists y$ (Enseñar(y, x))

Interpretación:

$\mathbb{D}$ = {A, B, C , a, b, c, d, CS100, CS200, P100, P200}

• Asignación a las constantes:
K: C $\rightarrow \mathbb{D}$ / **K**= {(c, d): c $\in$ C y d $\in \mathbb{D}$ y c=d}

• Asignación a los predicados:
E(=) = {(x,x): x $\in \mathbb{D}$}

E(Prof):

A
B
C

E(Alumno):

a
b
c
d

E(Curso):

CS100
CS200
P100
P200

E(Enseñar)

A	CS100
A	CS200
B	P100
C	P200

E(Matriculado)

a	CS100
a	CS200
b	CS100
c	P100
d	CS100
d	P200

Una base de datos así formalizada, puede interrogarse mediante fórmulas bien formadas abiertas del lenguaje L de manera que, el resultado de evaluar esas fórmulas en la interpretación asociada a la base de datos proporciona la respuesta a la pregunta.

1.2.3 Formalización de una base de datos relacional como una teoría de primer orden

Una base de datos relacional se ha definido desde la perspectiva de la Teoría de Modelos como una interpretación I de un lenguaje de primer orden L. Vamos ahora a definir la misma base de datos desde la perspectiva de la Teoría de la Demostración como una teoría T de primer orden sobre el mismo lenguaje L. Esta definición se debe hacer de forma que las dos formalizaciones de la base de datos sean ***"equivalentes"***. Es decir para toda fbf F de L:

$$\models_I F \quad \text{sii} \quad T \models F$$

Definición de la teoría:

Axiomas de T :

1) Información de información básica

Por cada predicado n-ario p de L y por cada tupla $<d_1, ..., d_n>$ perteneciente a la relación p en la base de datos, se incluye en T el átomo $\mathbf{p(d_1, ..., d_n)}$.

En el ejemplo 1.1 la teoría T tendría los siguientes axiomas:

$$T= \{ Prof(A), Prof(B), ..., Matriculado(d, P200)\} \cup \{\forall x =(x, x)\}$$

En esta teoría ya se pueden deducir fórmulas que eran ciertas en I, como son:

$T \models Matriculado(d, P200)$
$T \models Matriculado(a, P100) \wedge Enseña(B, P200)$
$T \models \exists x\ (Enseña(A, x) \wedge Matriculado(a, x))$

2) Axioma de cierre de dominio

Existen fórmulas F que no son consecuencia lógica de la teoría T y sin embargo son ciertas en la interpretación I, es decir $T \not\models F$ y $\models_I F$, por ejemplo la fórmula *dependiente del dominio*, $F = \forall x\ (Prof(x) \vee Curso(x) \vee Alumno(x))$.

Por ello habrá que añadir a T un axioma que defina explícitamente el domino:

$$\mathbf{\forall x\ (=(x, d_1) \vee ... \vee =(x, d_n))}$$

tal que $\{d_1, d_2,...d_n\}$ son las constantes del lenguaje.

Con este axioma la teoría T queda de la siguiente forma

$$T= \{ Prof(A), Prof(B), ..., Matriculado(d, P200), \forall x =(x, x), \forall x (=(x, A) \vee =(x, B) \vee ... \vee =(x, P200)) \}$$

3) Axiomas de compleción (información negativa)

La fórmula $\neg$ Prof(P100) es cierta en I, pero no es consecuencia lógica de T. Para poder deducir esta fórmula, T necesita "saber" que los únicos individuos que son profesores (que están en la extensión de Prof) son A, B y C. Este conocimiento lo expresan los *axiomas de compleción*. Existe un axioma de compleción por cada predicado de la base de datos (excepto el predicado =). Veamos dos ejemplos:

$$\forall x (Prof(x) \rightarrow (=(x, A) \vee =(x, B) \vee =(x,C)))$$

$$\forall x,y(Enseña(x, y) \rightarrow((=(x, A) \wedge =(y, CS100)) \vee (=(x, A) \wedge =(y, CS200)) \vee (=(x, B) \wedge =(y, P100)) \vee (=(x, C) \wedge =(y, P200))))$$

Es posible escribir de una forma más compacta los axiomas básicos de un predicado junto con su axioma de compleción, como se ve en el siguiente ejemplo:

Atomos básicos de Prof:

Prof(A)
Prof(B) $\equiv$ $\forall x(Prof(x) \leftarrow (=(x,A) \vee =(x, B) \vee =(x, C)))$
Prof(C)

Axioma de compleción de Prof:

$$\forall x(Prof(x) \rightarrow(=(x,A) \vee =(x, B) \vee =(x, C)))$$

por tanto se puede obtener la siguiente fórmula equivalente:

$$\forall x(Prof(x) \leftrightarrow (=(x,A) \vee =(x, B) \vee =(x, C)))$$

¿Qué ocurre si la extensión del predicado Prof estuviera vacía? El axioma de compleción sería:

$$\forall x (\neg Prof(x))$$

En general, para cada predicado n-ario p de L, distinto de =, T contendrá el siguiente axioma de compleción:

1) si la relación p no está vacía:

$$\forall x_1, ..., x_n\ (p(x_1, ..., x_n) \rightarrow ((=(x_1, d_{11}) \wedge ... \wedge =(x_n, d_{1n})) \vee$$
$$\vdots$$
$$(=(x_1, d_{m1}) \wedge ... \wedge =(x_n, d_{mn})))$$

tal que $<d_{j1}, ..., d_{jn}>$ $(1 \leq j \leq m)$ es una tupla de p en la base de datos.

2) si la relación p está vacía:

$$\forall x_1, ..., x_n\ (\neg\ p(x_1, ..., x_n))$$

Finalmente la teoría T con estos axiomas queda de la forma siguiente:

T= { Prof(A), Prof(B), ..., Matriculado(d, P200), $\forall$ x =(x, x)
$\forall$x (=(x, A) $\vee$ =(x, B) $\vee$ ... $\vee$ =(x, P200)),
$\forall$x (Prof(x) $\rightarrow$ (=(x, A) $\vee$ =(x, B) $\vee$ =(x, C)))
. . .
$\forall$x,y(Matriculado(x,y) $\rightarrow$ .... }

En la formalización de una base de datos relacional como una teoría de primer orden, los axiomas de compleción formalizan la **Hipótesis del Mundo Cerrado** [Rei78b]. Esta hipótesis para derivar información negativa se enuncia de la siguiente forma:

HMC: $D \not\models A$

$$\frac{}{\neg A}$$

esta hipótesis expresa la idea de que la información no explícita en la base de datos debe considerarse falsa.

4) Axiomas de nombre único

Las fórmulas siguientes son ciertas en I, pero no son consecuencias lógicas de T:

$\neg$ =(A, B), $\neg$ =(A, C), ..., $\neg$ =(P100, P200)

El problema reside en que T sabe cuando dos constantes son iguales, pero no cuando son distintas. Es necesario, por tanto, incluir por cada par de constantes distintas c, c' pertenecientes al conjunto de constantes del lenguaje L, el siguiente axioma en T:

$$\neg =(c, c')$$

Con estos axiomas la teoría T queda:

T= { Prof(A), Prof(B), ..., Matriculado(d, P200), $\forall$ x =(x, x)
$\forall$x (=(x, A) $\vee$ =(x, B) $\vee$ ... $\vee$ =(x, P200)),
$\neg$ =(A, B), $\neg$ =(A, C), ..., $\neg$ =(P100, P200),
$\forall$x (Prof(x) $\rightarrow$ (=(x, A) $\vee$ =(x, B) $\vee$ =(x, C)))
. . .
$\forall$x,y(Matriculado(x,y) $\rightarrow$ .... }

TEORIA T:

1) Información básica

Por cada predicado n-ario p de L se incluye un conjunto de átomos base, de la siguiente forma:

AB(p)= { $\mathbf{p(c_1, ..., c_n)}$: $<c_1, ..., c_n>$ es una tupla de p en la base de datos}
$\cup$
{$\forall$ x =(x, x)}

A AB(p) se le denomina *extensión de p en T*

2) Axioma de cierre de dominio

Si $c_1, ..., c_n$ son las constantes del lenguaje L, T contiene el siguiente axioma:

$$\forall x (=(x, c_1) \vee ... \vee =(x, c_n))$$

3) Axiomas de compleción

Sea un predicado n-ario p de L (distinto de =), si

a) AB(p) $\neq \emptyset$, entonces T contiene el axioma

$$\forall x_1, ..., x_n(p(x_1, ..., x_n) \rightarrow ((=(x_1, d_{11}) \wedge ... \wedge =(x_n, d_{1n})) \vee$$
$$...$$
$$(=(x_1, d_{m1}) \wedge ... \wedge =(x_n, d_{mn})))$$

tal que $<d_{j1}, ..., d_{jn}>$ $(1 \leq j \leq m)$ es una tupla de p en la base de datos

b) si AB(p) = $\emptyset$, entonces T contiene el axioma

$\forall x_1, ..., x_n (\neg p(x_1, ..., x_n))$

4) Axiomas de nombre único

Por cada par de constantes distintas c_i, c_j del lenguaje L, T contiene el siguiente axioma:

$\neg =(c_i, c_j)$

De la formalización que se ha presentado de una base de datos relacional como una teoría T de primer orden se deduce que T es la compleción de D (átomos que representan las tuplas de la relación) más el axioma de cierre de dominio:

$D= \{ \{ p(\mathbf{c_1, ..., c_n}) : <c_1, ..., c_n> \in E(p) \text{ en } I\}$
$\cup$
$\ldots \quad \}$

$T= comp(D) \cup \{\text{axioma de cierre de dominio}\}$

1.3 Formalización lógica de una base de datos deductiva

Un esquema de base de datos deductiva es un par (L, RI) donde:

- L es un lenguaje de primer orden.
- RI es un conjunto de restricciones de integridad, fórmulas cerradas de L.

Una base de datos deductivas consiste en un conjunto de hechos (tuplas de la relaciones básicas) y un conjunto de reglas deductivas que definen las relaciones derivadas.

Si llamamos D al conjunto de fórmulas de L que representan los hechos y las reglas deductivas (sentencias de base de datos):

$D= \{A \leftarrow W:$ A es un átomo y W es una fórmula bien formada$\}$

la teoría que formaliza la base de datos deductiva, extendiendo la formalización hecha para el caso relacional es:

$T= comp(D) \cup \{\text{axioma de cierre de dominio}\}$

A lo largo del texto se utilizará la notación propia de la programación lógica en la cual todas las variables libres en $A \leftarrow W$ se consideran cuantificadas universalmente, la sentencia sin cuerpo $A \leftarrow$ representa la fórmula $\overline{\forall} A$ y la sentencia sin cabeza $\leftarrow W$ representa la fórmula

(¬W). Si W está ausente la sentencia A← representa un hecho, en caso contrario la sentencia A ← W representa una regla deductiva.

Una base de datos normal es una base de datos cuyas sentencias son cláusulas, es decir son de la forma $A \leftarrow L_1 \wedge L_2 \wedge ... \wedge L_n$ donde L_i es un literal y $n \geq 0$.

Una base de datos definida es una base de datos cuyas sentencias son cláusulas de Horn, es decir son de la forma $A \leftarrow B_1 \wedge B_2 \wedge ... \wedge B_n$ donde B_i es un átomo y $n \geq 0$.

De la formalización anterior se deduce que una base de datos deductiva es lo mismo que un programa lógico [LT85], [LT86], [Llo87] estando ambos formados por un conjunto de hechos y reglas. La única diferencia entre ambos conceptos es cuantitativa, en un programa lógico el número de reglas es comparable al número de hechos, mientras que una base de datos deductiva está formada por un número elevado de hechos y pocas reglas. Esta diferencia será relevante en la construcción de sistemas de gestión de bases de datos deductivas, pero desde un punto de vista teórico el tratamiento es el mismo.

Como han demostrado Lloyd y Topor, para toda base de datos se puede obtener una base de datos normal equivalente utilizando el algoritmo de transformación propuesto en [LT84]. La forma normal obtenida es equivalente a la anterior en el siguiente sentido: "Sea D una base de datos cualquiera y D' una forma normal de D. Si W es una fórmula cerrada que es consecuencia lógica de comp(D') (la compleción de D') y sólo contiene símbolos de predicado que aparecen en D, entonces W es consecuencia lógica de comp(D)" [LT84].

En la formalización lógica de una base de datos deductiva presentada y debido al uso de una lógica homogénea, desaparece un concepto importante en el modelo relacional de datos, el concepto de dominio de un atributo. Esta limitación de la representación podría superarse usando una lógica heterogénea que permitiera asociar un tipo a cada atributo de una relación. En los trabajos de Lloyd y Topor [LT86], [Llo87] se presenta una formalización de una base de datos deductiva utilizando una lógica heterogénea (con tipos); asimismo se propone una transformación que permite pasar de la teoría con tipos que representa un estado de la base de datos en la semántica de la compleción, a una teoría sin tipos. Esta transformación se realiza introduciendo en el lenguaje predicados de tipo e incluyendo en la teoría los correspondientes axiomas que definen su extensión. Debido a la existencia de esta transformación y por claridad en la exposición, en el texto hemos propuesto una formalización a partir de una lógica homogénea, sin que esta elección quite generalidad al trabajo.

Existen otras posibles formalizaciones de una base de datos deductiva además de la formalización por la teoría de la compleción que se ha presentado, a continuación se presentan

un resumen de ellas. [Man89], [Man91], [She84], [She85], [She88], [Bid91], [CD91], [PP90], [Llo87], [BM89]

1.4 Semántica declarativa de una base de datos deductiva

De la definición de base de datos dada anteriormente (1.2) se deduce que no es posible derivar información negativa de una base de datos. Dado un estado D y un átomo base A (átomo de la base de Herbrand de L (B_L)), $D \not\models \neg A$ ya que $D \cup \{A\}$ siempre es satisfactible.

En una base de datos relacional, formada sólo por hechos, la información negativa se deriva utilizando la regla de inferencia conocida como **regla del mundo cerrado** (RMC) [Rei78b]:

$$\text{RMC:}\quad \frac{D \not\models A}{\neg A}$$

esta regla expresa la idea de que la información no explícita en la base de datos debe considerarse falsa.

En bases de datos deductivas sin embargo la información contenida en la base de datos no se obtiene por simple observación, en éstas la aplicación de la RMC para derivar ¬A de D, exige demostrar que $D \not\models A$ y debido a que éste es un problema indecidible en lógica de primer orden, en la práctica esta regla se restringe al conjunto F_D de átomos base A para los cuales el intento de demostrar A a partir de D falla en un tiempo finito. Esta formulación restringida de la regla del mundo cerrado se conoce como **regla de la negación como fallo** (RNF) [CLa78]:

$$\text{RNF:}\quad \frac{A \in F_D}{\neg A}$$

Las reglas de inferencia anteriores nos permiten derivar información negativa de una base de datos, pero ¿cual es la semántica declarativa respecto a la cual son correctas estas reglas?. Dar una semántica declarativa significa definir la información tanto positiva como

negativa implícita en la base de datos. Las propuestas existentes se pueden clasificar en dos aproximaciones:

Aproximación sintáctica: consiste en representar la base de datos D por una teoría Tr e interpretar la negación en Tr de la forma clásica, es decir ¬A es *cierto* en D si y sólo si Tr$\models$ ¬A. La semántica en esta aproximación viene definida por el conjunto de literales base (sin variables) que son consecuencia lógica de Tr, Sem_Tr={L: L es un literal base, Tr$\models$ L}. A esta aproximación pertenece la semántica de la compleción [Cla78].

Aproximación por modelo minimal: consiste en seleccionar un modelo M entre los modelos minimales de Herbrand de D e interpretar la negación en dicho modelo, es decir ¬A es *cierto* en D si y sólo si $\models_M$ ¬A. La semántica en esta aproximación viene definida por el conjunto de literales base ciertos en M, Sem_M={L: L es un literal base, $\models_M$ L}. A esta aproximación pertenecen la semántica del punto fijo iterado [ABW88], la semántica del modelo perfecto [Prz88], la semántica del modelo estable [GL88] [SZ90] y la semántica bien fundada [vGRS88].

1.4.1 Semántica de la compleción

Esta semántica se basa en la idea de interpretar el conocimiento expresado en la base de datos como *completo*, es decir el conjunto de sentencias que tienen el mismo símbolo de predicado en la cabeza se interpreta como la definición *completa* de dicho predicado en D.

La compleción de una base de datos se obtiene añadiendo a D los axiomas de compleción para cada predicado de L junto con los axiomas de la teoría de la igualdad (estos últimos son necesarios al aparecer el predicado igualdad en los axiomas anteriores).

Los axiomas de compleción para un predicado p se definen de la forma siguiente:

a) si en D existen sentencias con el predicado p en la cabeza de la forma $p(t_1,..,t_n)\leftarrow L_1\wedge L_2\wedge...\wedge L_m$, cada una de ellas se transforma en la forma:

$$p(x_1,...,x_n)\leftarrow \exists y_1...\exists y_d\,(x_1=t_1\wedge...\wedge x_n=t_n\wedge L_1\wedge L_2\wedge...\wedge L_m)$$

donde $y_1,...,y_d$ son las variables en la sentencia original. Si existen k ≥1 sentencias con el predicado p en la cabeza tenemos:

$$p(x_1,...,x_n)\leftarrow E_1$$
$$\vdots$$
$$p(x_1,...,x_n)\leftarrow E_k$$

donde E_i tiene la forma general

$E_i = \exists y_1 ... \exists y_d \, (x_1 = t_1 \wedge ... \wedge x_n = t_n \wedge L_1 \wedge L_2 \wedge ... \wedge L_m)$

Con estas transformaciones el axioma de compleción para p se define:

$\forall x_1 ... \forall x_n (p(x_1,...,x_n) \rightarrow (E_1 \vee ... \vee E_k))$

b) si en D no existe ninguna sentencia con el predicado p en la cabeza el axioma de compleción para p es:

$\forall x_1 ... \forall x_n \, (\neg p(x_1,...,x_n))$

Los axiomas de la teoría de la igualdad son:

1. $c \neq d$ para todo par de símbolos de constante c y d distintos.
2. $\overline{\forall} \, (f(x_1,...,x_n) \neq g(y_1,...,y_n))$ para todo par de símbolos de función f y g distintos.
3. $\overline{\forall} \, (f(x_1,...,x_n) \neq c)$ para toda constante c y toda función f.
4. $\overline{\forall} \, (t[x] \neq x)$ para cada término t[x] que contiene a x y es distinto de x.
5. $\overline{\forall} \, ((x_1 \neq y_1) \vee ... \vee (x_n \neq y_n) \rightarrow f(x_1,...,x_n) \neq f(y_1,...,y_n))$ para cada símbolo de función f.
6. $\forall x (x = x)$.
7. $\overline{\forall} \, ((x_1 = y_1) \wedge ... \wedge (x_n = y_n) \rightarrow f(x_1,...,x_n) = f(y_1,...,y_n))$ para cada símbolo de función f.
8. $\overline{\forall} \, ((x_1 = y_1) \wedge ... \wedge (x_n = y_n) \rightarrow p(x_1,...,x_n) \rightarrow p(y_1,...,y_n))$ para cada símbolo de predicado p

La teoría Tr que representa la base de datos D en la semántica de la compleción es:

$Tr = comp(D) =$ D
$\cup$
{axiomas de compleción para
cada predicado de L (distinto de =) }
$\cup$
{axiomas de la igualdad}

La semántica de la compleción viene definida por el conjunto de literales base que son consecuencia lógica de Tr, $COMP(D) = \{L: L \text{ es un literal base}, Tr \models L\}$

La compleción de una base de datos tiene una serie de propiedades importantes:

a) Si D es una base de datos definida y A es un átomo base:

- comp(D) es consistente
- $comp(D) \models A$ sii $D \models A$

- comp(D)$\models \neg A$ sii $A \in F_D$, donde F_D es el conjunto de fallo finito de D ($F_D = B_L \backslash T_D \downarrow w$) [Llo87]

b) Si D es una base de datos normal:

- comp(D) puede no ser consistente
- comp(D)$\models$ D

La semántica de la compleción fue introducida por Clark [Cla78] para dar una semántica declarativa a la regla de la negación como fallo utilizada en el lenguaje Prolog y que se enuncia de la forma siguiente: si existe un árbol SLDNF fallado finitamente para $D \cup \{G\}$ entonces comp(D)$\models$ G.

1.4.2 Semántica del modelo minimal

La semántica por modelo minimal se basa en la idea de seleccionar entre todos los modelos minimales de Herbrand de D aquél que capture la *semántica supuesta* de D.

Bases de Datos Definidas

Si D es una base de datos definida existe un modelo mínimo de Herbrand de D, M_D, tal que $M_D = \{A: A \in B_L, D \models A\}$. Este modelo M_D puede caracterizarse por el concepto de punto fijo, concretamente como el menor punto fijo (mpf) del operador consecuencia inmediata T_D definido de la forma $T_D : 2 \rightarrow 2$y $T_D(I) = \{A: A \in B_L, A \leftarrow A_1 \wedge ... \wedge A_n$ es una instancia base (instancia en el universo de Herbrand de L) de una sentencia de D y $\{A_1,...,A_n\} \subseteq I\}$. Si D es una base de datos definida y M_D es su modelo mínimo de Herbrand se cumple $M_D = mpf(T_D)$.

La existencia de este modelo mínimo conduce a definirlo como semántica estándar de una base de datos definida D. Vamos a representar esta semántica por el conjunto de literales base ciertos en M_D: MIN(D) = $\{L : L$ es base, $\models_{M_D} L\}$.

La relación entre la semántica del modelo mínimo y la semántica de la compleción para bases de datos definidas es:

$$COMP(D) \subseteq MIN(D)$$

Bases de Datos Normales

En el caso de bases de datos normales la propiedad anterior desaparece, ya que una base de datos normal puede tener más de un modelo minimal de Herbrand.

Ejemplo 1.2

Sea D= {p(x)←¬q(x)∧ r(x), r(a)←}. D tiene dos modelos minimales: M_1={r(a), q(a)} y M_2={r(a), p(a)}.

La semántica del modelo minimal viene definida por el conjunto de literales base ciertos en todo modelo minimal de D:

MIN(D)= { L: L es un literal base, $\models_M$ L para todo modelo minimal M de D}

Esta semántica es consistente con la regla de inferencia denominada **regla del mundo cerrado generalizada** (RMCG) [Min82] que consiste en:

RMCG: $\not\models_M$ A para todo modelo minimal M de D

¬ A

La semántica del modelo minimal definida de esta forma no es satisfactoria. En el ejemplo 1.2 MIN(D)={r(a)} sin embargo la *semántica supuesta* para esta base de datos viene dada por el modelo M_2={r(a), p(a)}, es decir puesto que no hay motivo evidente para suponer que q(a) es cierto en D lo suponemos falso y derivamos p(a). Para expresar esta idea intuitiva de la *semántica supuesta* de una base de datos se introduce el concepto de **modelo soportado de D**.

Dada una base de datos y una interpretación de Herbrand I, diremos que I es soportada si y sólo si para todo átomo A de I existe una instancia base de una sentencia de D de la forma A ←L_1∧L_2∧...∧L_n tal que L_1∧L_2∧...∧L_n es cierto en I.

En el ejemplo 1.2 M_2 es un modelo soportado de D mientras que M_1 no lo es. La idea consiste ahora en elegir entre todos los modelos minimales de D uno que sea soportado. Vamos a estudiar distintas clases de bases de datos para las cuales existe un modelo minimal soportado.

<u>Semántica del punto fijo iterado [ABW88]</u>

Esta semántica está definida para bases de datos estratificadas. La división en niveles D_1,D_2,...,D_k de una base de datos estratificada sugiere un modo de seleccionar un modelo minimal de D. Este modelo minimal se construye de la forma siguiente:

D_1 M_1 (modelo mínimo de D_1)

$D_2 \cup D_1$ M_2 (minimal y $M_1 \subseteq M_2$)

....

$D_k \cup D_{K-1} \cup ... \cup D_1$ M_k (minimal y $M_{k-1} \subseteq M_k$)

Sea $\mu_D = M_K$. La semántica del punto fijo iterado se define por el conjunto de literales base ciertos en μ_D

$$ESTRA(D) = \{ L: L \text{ es un literal base}, \models_{\mu_D} L\}$$

este modelo es independiente de la estratificación de D.

El modelo μ_D (modelo estándar) puede caracterizarse por el concepto de punto fijo. Sea T_D el operador consecuencia inmediata definido para bases de datos normales como T_D: $2 \rightarrow 2^y$ $T_D(I)=\{A: A \in B_L , A \leftarrow L_1 \wedge ... \wedge L_n$ es una instancia base de una sentencia de D y $L_1 \wedge ... \wedge L_n$ es cierto en I} (este operador en general no es monótono). Definamos ahora el operador τ_D tal que $\tau_D: 2 \rightarrow 2^y$ $\tau_D(I)= T_D(I) \cup I$.

Si D es una base de datos estratificada, $D_1, D_2, ..., D_k$ su división en niveles y $M_0, ..., M_k$ la secuencia definida como:

$M_0 = \varnothing$

$M_j = \tau_{D_j} \uparrow w\ (M_{j-1})$ para j:1,...,K

se cumple:

- M_K es un modelo minimal y soportado de D
- M_K es independiente de la estratificación elegida para D
- $M_K = \mu_D$

La relación entre la semántica del punto fijo iterado y la semántica de la compleción es:

$$COMP(D) \subseteq ESTRA(D)$$

Semántica del modelo perfecto [Prz88]

La semántica del punto fijo iterado puede extenderse a bases de datos que aunque no son estratificadas su instancia base lo es. Esta clase de bases de datos recibe el nombre de localmente estratificadas. La instancia base de D es el conjunto de todas las instancias base de las sentencias de D. Es importante resaltar que M es un modelo de la instancia base de D si y sólo si M es un modelo de Herbrand de D.

Ejemplo 1.3

Sea D={par(0)←, par(s(x))←¬par(x)}; para esta base de datos no está definido ESTRA(D) ya que no es estratificada, sin embargo la *semántica supuesta* para D es el modelo: M={par(0), par(s(s(0)), par(s(s(s(s(0))))),...}.

Se puede comprobar que este modelo coincide con el menor punto fijo del operador τ_D para la instancia base de D.

Para toda base de datos localmente estratificada existe un modelo minimal soportado π_D, el modelo perfecto de D, que coincide con el menor punto fijo del operador τ_D para la instancia base de D. La semántica del modelo perfecto viene definida por el conjunto:

$$ESTRA\_L=\{L: L \text{ es un literal base y } \models_{\pi_D} L\}$$

La relación entre la semántica del modelo perfecto y la semántica de la compleción es:

$$COMP(D) \subseteq ESTRA\_L(D)$$

Semántica del modelo estable [SZ90] [GL88]

Existen bases de datos que no siendo localmente estratificados tienen una *semántica supuesta* clara.

Ejemplo 1.4

Sea D={p(x)←q(x,y)∧¬p(y), q(a,b)←}; la instancia base de D contiene la sentencia p(a) ←q(a,a) ∧ ¬p(a), por lo tanto no es localmente estratificada y para ella no está definido ESTRA_L(D). Sin embargo, esta base de datos tiene una *semántica supuesta* clara que coincide con el modelo minimal {q(a,b), p(a)}.

La idea que subyace en la semántica del modelo estable es una definición más restrictiva del concepto de interpretación soportada: "dada una interpretación I los átomos ciertos en I deben ser derivables de D y de los átomos falsos en I".

Históricamente, la semántica del modelo estable fue propuesta por Gelfond y Lifschitz [GL88] para una lógica bivaluada (modelo estable total) y extendida posteriormente para una lógica trivaluada en los trabajos de Sacca y Zaniolo [SZ90] (modelo estable parcial). Vamos a presentar el modelo estable parcial ya que el modelo estable total es un caso particular de éste.

Sea B la instancia base de una base de datos D y sea F un conjunto de átomos base. Definimos $\Gamma(B, F)$ como la base de datos obtenida a partir de B aplicando las siguientes reglas:

i) eliminar toda sentencia que contenga un átomo positivo A tal que $A \in F$.
ii) eliminar de las sentencias restantes todos los literales negativos $\neg A$ tales que $A \in F$.
iii) eliminar las restantes sentencias que contengan un literal negativo.

$\Gamma(B, F)$ es una base de datos definida que representa la información positiva derivable de D, asumiendo como falsos los átomos del conjunto F.

Sea Con(B,F) el modelo mínimo de $\Gamma(B, F)$ e $I=<T,F>$ una interpretación parcial de B, diremos que I es admisible si:

i) T=Con(B,F)
ii) Para todo $A \in F$ y para toda sentencia en B del tipo $A \leftarrow L_1 \wedge .. \wedge L_n$, $L_1 \wedge .. \wedge L_n$ es falso en I.

El conjunto de interpretaciones admisibles de B es un conjunto parcialmente ordenado respecto a la relación de orden: $<T_1, F_1> << <T_2, F_2>$ sii $F_1 \subseteq F_2$.

El concepto de interpretación admisible es una versión más restrictiva del concepto de interpretación soportada.

Ejemplo 1.5

Sea $D=\{p \leftarrow p\}$; D tiene tres modelos parciales $M_1=<\varnothing,\{p\}>$, $M_2=<\{p\},\varnothing>$ y $M_3=<\varnothing,\varnothing>$; M_1 y M_3 son admisibles y M_2 es soportado. Como puede observarse el modelo que captura mejor la semántica de D es M_3.

Una interpretación parcial M es un modelo estable (parcial) de B si es admisible y maximal. Un modelo estable de una base de datos es un modelo estable de su instancia base. Para toda base de datos existe al menos un modelo estable (parcial).

Sea D una base de datos, la semántica del modelo estable viene definida por el conjunto de literales base que son ciertos en todo modelo estable (parcial) de D:

$$ESTAB(D)=\{L: L \text{ es un literal base y } \models_M L, \text{ para todo modelo estable M}\}$$

El modelo estable de D en el ejemplo 1.4 es $M=<\{q(a,b), p(a)\}, \{p(b), q(a,a), q(b,b), q(b,a)\}>$, este modelo estable representa la *semántica supuesta* de D.

Un modelo estable será total si $T \cup F$ cubre la base de Herbrand de L. En el ejemplo anterior el modelo estable es un modelo total. Existen bases de datos para las que no está definido un modelo estable total. Para bases de datos (localmente) estratificadas existe un único modelo estable total que coincide con el modelo μ_D.

Ejemplo 1.6

Sea D={p←¬p}; D sólo tiene un modelo estable parcial M=<∅,∅>, que captura su *semántica supuesta*.

Semántica del modelo bien fundado [vGRS88]

La semántica del modelo estable aunque está definida para toda base de datos no permite aislar un único modelo.

La propuesta de Van Gelder, Ross y Schlipf [vGRS88] intenta seleccionar un modelo admisible de D tal que el conjunto de átomos F cumpla ciertas propiedades "razonables". Estas propiedades se basan en el concepto de **conjunto infundado**.

Sea B la instancia base de una base de datos D, I=<T,F> una interpretación parcial de B y X un conjunto de átomos base. Decimos que X es infundado respecto a I si para toda sentencia de B de la forma $A \leftarrow L_1 \wedge ... \wedge L_n$ con $A \in X$ se cumple una de las siguientes condiciones:

i) $L_1 \wedge ... \wedge L_n$ es falso en I
ii) $L_1 \wedge ... \wedge L_n$ contiene un átomo positivo de X.

Estas condiciones tienen el siguiente significado: las reglas de B que satisfacen la primera condición no son aplicables para derivar información ya que su cuerpo es falso en I, la segunda condición establece que, las restantes reglas de B con un átomo de X en la cabeza sólo son aplicables si se asumen como ciertos los átomos del conjunto X.

Ejemplo 1.7

Sea D={p←p}; el conjunto {p} es infundado respecto a <∅,∅>. La derivabilidad de p exige asumir p como cierto.

Sea D={p←q, q←p}; El conjunto {p,q} es infundado respecto a <∅,∅>. La derivabilidad de p requiere asumir q como cierto y viceversa.

Dada una interpretación I los átomos que están en un conjunto X infundado respecto a I no pueden ser considerados ciertos sin violar la condición (i) de la definición de interpretación admisible por ello podemos extender I considerando todos los átomos de X falsos. La semántica bien fundada consiste en iterar este proceso partiendo de la interpretación parcial en la cual todo átomo es indefinido (I=<∅,∅>).

Dada una interpretación I de B denominamos:

- $U_D(I)$ la unión de todos los conjuntos infundados respecto a I
- $T_D(I)$ el conjunto de los átomos A tales que existe una sentencia $A \leftarrow L_1 \wedge ... \wedge L_n$ en B con $L_1 \wedge ... \wedge L_n$ cierto en I
- $W_D(I)$ la interpretación $<T_D(I),U_D(I)>$

El operador W_D es monótono y su menor punto fijo es el modelo bien fundado ω_D de B. El modelo bien fundado de D es el modelo bien fundado de B. La semántica bien fundada viene definida por el conjunto de literales ciertos en ω_D:

$$BF(D)=\{L: L \text{ es un literal base}, \models_{\omega_D} L\}.$$

Ejemplo 1.8

Sea $D=\{p \leftarrow \neg q,\ q \leftarrow \neg p\}$; D tiene tres modelos admisibles $M_1=<\{p\},\{q\}>$, $M_2=<\{q\},\{p\}>$ y $M_3=<\emptyset,\emptyset>$. El modelo bien fundado de D es M_3, este modelo captura la *semántica supuesta* de D, es decir, ya que no hay información en D que permita derivar p o q, ambos átomos se suponen desconocidos.

El modelo bien fundado y el modelo estable se corresponden respectivamente con las visiones minimalista y maximalista de la propiedad de admisibilidad. El modelo bien fundado es admisible mientras que un modelo estable es admisible y maximal. La relación entre la semántica bien fundada y la del modelo estable es:

$$BF(D) \subseteq ESTAB(D)$$

En el caso en que el modelo bien fundado es total, éste coincide con el único modelo estable. Para bases de datos (localmente) estratificadas, la semántica ESTRA_L, ESTAB y BF coinciden.

2. Actualizaciones en bases de datos deductivas

2.1 Introducción

La actualización de Bases de Datos Deductivas es un caso particular de un problema más general, la *asimilación del conocimiento* en sistemas deductivos. Este problema ha sido tratado desde distintos campos y aproximaciones: Inteligencia Artificial ("belief revision", "incremental concept-learning", "truth maintenance"), Bases de Datos Deductivas ("intensional database updating", "knowledge base updating") y Programación Lógica ("non-monotonic logic", "default reasoning", "abductive reasoning", "hypothetical reasoning").

La *asimilación del conocimiento* en Bases de Datos Deductivas es el **proceso por el cual un nuevo conocimiento es incorporado a la base de datos** [Dec92]. Durante la *adquisición del conocimiento* un especialista del sistema analiza y sintetiza (traduce al formalismo adecuado) el conocimiento que distintos expertos tienen sobre el área o universo de discurso. El nuevo conocimiento a incorporar a la base de datos, traducido por el especialista en términos de *requerimientos de actualización,* es analizado por el *subsistema asimilador* que, en un proceso interactivo, traduce el requerimiento de actualización en una (o varias) *transacciones* (operaciones de inserción y/o borrado de elementos de información) a ejecutar sobre la base de datos.

El problema de la asimilación de conocimiento en Bases de Datos Deductivas es una generalización del problema de la actualización de vistas en los sistemas relacionales, que como es sabido consiste en traducir una operación de actualización sobre una vista en una transacción formada por operaciones de actualización sobre las relaciones básicas sobre las que está definida la vista [CP84], [BS81], [DB82]. En el caso de las Bases de Datos Deductivas este problema puede ser más complejo debido a que el lenguaje de definición de reglas (información implícita) es más potente y los requerimientos de actualización a su vez pueden ser más complejos [Abi88]. El problema puede enunciarse de la forma siguiente: Dados una base de datos deductiva D, que satisface un conjunto de restricciones de integridad RI, y un requerimiento de actualización insertar(ϕ) (resp. borrar(ϕ)) donde ϕ es una fórmula cerrada, encontrar una transacción T tal que T(D), la base de datos resultante de aplicar T a D, satisfaga RI y el requerimiento de actualización, es decir ϕ sea "cierto" (resp. "falso") en T(D).

En el problema de la asimilación del conocimiento se pueden identificar varios subproblemas que han sido estudiados en muchas ocasiones de forma aislada:

- **Formalización de la evolución de la base de datos**: definición de lenguajes de actualizaciones [Rei92], [MW88], [Bry90]. Tradicionalmente las actualizaciones y la

evolución de la base de datos se han expresado con metapredicados sin una semántica declarativa definida.

- **Actualización del conocimiento**: definición de estrategias para determinar el conjunto de transacciones que satisfacen un requerimiento de actualización. La forma más sencilla de satisfacer un requerimiento de actualización es incorporar explícitamente el nuevo conocimiento a la base de datos, sin embargo esto no es siempre lo mas adecuado: en muchas ocasiones la forma en que el conocimiento es adquirido no tiene en cuenta la representación de dicho conocimiento en la base de datos, además la incorporación explícita del nuevo conocimiento podría ocasionar la pérdida de la información potencialmente derivable a partir de él [Kow79]. Para optimizar el proceso de obtención de las transacciones que satisfacen un requerimiento de actualización, las propuestas existentes concentran la atención en la parte de la base de datos de la cual depende el nuevo conocimiento, esto se consigue analizando el árbol de derivación (en la semántica procedural asociada) que tiene el requerimiento de actualización como raíz. El método propuesto en [KM90] se basa en el procedimiento de resolución de Sadri&Kowalski [SK88] y en el procedimiento de razonamiento abductivo de Eshghi&Kowalski [EK89]. Los métodos propuestos en [GL90], [GL91], [Dec90] y [TO92] se basan en el procedimiento de resolución SLDNF. Algunos problemas asociados a la actualización del conocimiento son: la eficiencia del método, la definición de criterios para seleccionar la transacción más adecuada entre el conjunto de las obtenidas, el problema de la completitud del conjunto de transacciones y la generación de explicaciones sobre las soluciones obtenidas.

- **Comprobación de la Integridad:** Como se ha dicho anteriormente, el estado posterior a la transacción debe satisfacer el conjunto de restricciones de integridad. Esto obliga a incorporar al proceso de asimilación del conocimiento técnicas de comprobación de la integridad. La comprobación de la integridad es un problema clásico en bases de datos; todos los métodos propuestos en la literatura se basan en el hecho de que la base de datos era íntegra antes de la transacción y simplifican el proceso de comprobación evaluando sólo instancias de las restricciones de integridad obtenidas a partir de las actualizaciones inducidas por la transacción.

- **Restauración de la consistencia**: cuando la transacción que satisface un requerimiento de actualización viola la integridad de la base de datos, el sistema debe proponer un conjunto adicional de cambios (transacciones) que devuelvan la base de datos a un estado íntegro. El problema de la restauración (reparación) de la base de datos ha adquirido relevancia en los últimos años (bases de datos activas), aunque todavía son muy pocas las propuestas existentes; así en el campo de las bases de datos relacionales son destacables los trabajos de Ceri&Widom [CW90]; en el campo de las bases de datos deductivas son

importantes los trabajos de Moerkotte&Lockemann [ML91]. De nuevo la eficiencia del método de restauración es uno de los objetivos más importantes en este campo.

2.2 Método de Kakas&Mancarella [KM90]

2.2.1 Conceptos Previos

Enunciado del problema:

"Dados una base de datos localmente estratificada D= EDB ∪ IDB y un requerimiento de actualización insertar(ϕ) (resp. borrar(ϕ)) tal que ϕ es un átomo base sobre un predicado derivado, encontrar una transacción T sobre EDB tal que T(D), la base de datos resultante de aplicar T a D, satisfaga el requerimiento de actualización, es decir ϕ sea "cierto" (resp. "falso") en el modelo estable de T(D) [GL88]".

Fundamentos del procedimiento de actualización:

- procedimiento de abducción de Eshghi&Kowalski [EK89]

Abducción: Dados una base de datos D, un conjunto de restricciones de integridad RI y una fórmula F, una explicación abductiva de F es un conjunto de fórmulas H (hipótesis), tal que:

- D ∪ H es consistente
- D ∪ H implica F (en la semántica asumida)
- D ∪ H satisface RI.

Ejemplo 2.1: (Base de datos definida)

D=IDB ∪ EDB

IDB: $p(x) \leftarrow B2(x) \wedge q(x)$ EDB: B1(c)

$q(x) \leftarrow B1(x)$ B1(d)

U: insertar(p(c))

Predicados abducibles: B1, B2 (predicados básicos)

Paso A: Obtener una explicación abductiva de p(c) en IDB (independiente de EDB):

insertar(p(c)) en D — abducción → { insertar (B2(c)), insertar(B1(c)) en EDB }

Paso B: ejecutar en EDB la actualización obtenida en el paso A.

Marco abductivo

Dada una base de datos D=IDB∪EDB, el marco abductivo asociado con D es <IDB*,Ab,RI*> donde:

• IDB* es el programa lógico definido obtenido a partir de IDB, reemplazando cada literal negativo ¬q(t) por un nuevo literal positivo q*(t)

• Ab es el conjunto de símbolos de predicados básicos junto con un nuevo predicado α^* por cada predicado básico o derivado α.

• RI* es un conjunto de restricciones de integridad tal que para todo conjunto Δ de hipótesis abducidas sobre predicados de Ab, IDB*∪Δ debe satisfacer:

$RI^*=\{\leftarrow\alpha(t) \wedge \alpha^*(t)$: α es un predicado de la base de datos$\}$

$\cup$

$\{Demo(IDB^*\cup\Delta, \alpha(t)) \vee Demo(IDB^*\cup\Delta, \alpha^*(t))$:

α es un predicado de la base de datos y Δ un conjunto de hipótesis$\}$

(estas restricciones de integridad pretenden definir la equivalencia entre $\alpha^*(t)$ y $\neg\ \alpha(t)$)

Ejemplo 2.2: (Base de datos normal)

D=IDB ∪ EDB

IDB: p(x)←¬q(x) EDB: B(a)

q(x)←B(x)

U : insertar(p(a))

Marco Abductivo:

IDB*: p(x)←q*(x)

q(x)←B(x)

Ab = {B, B*, q*, p*}

RI*: {←q(x) ∧ q*(x), q(x) ∨ q*(x)

←p(x) ∧ p*(x), p(x) ∨ p*(x)

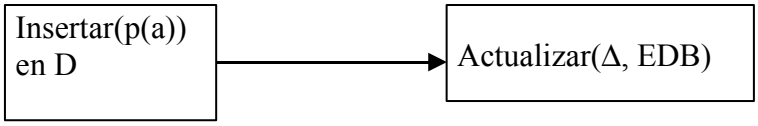
←B(x) ∧ B*(x), B(x) ∨ B*(x)}

Estrategia para actualizaciones:

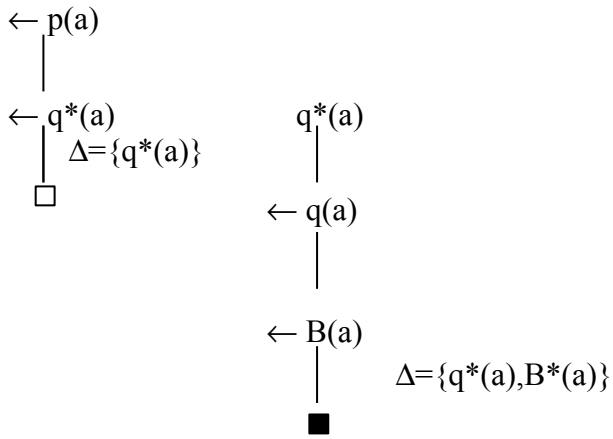
Dado un requerimiento de actualización insertar(p(a)) (resp. borrar(p(a)))

Paso A: Resolver abductivamente ←p(a) (resp. ←p*(a)) en <IDB*,Ab,RI*> generando un conjunto de hipótesis Δ tal que :

- IDB* ∪ Δ implica p(a) (resp. (p*(a))
- IDB* ∪ Δ es consistente y no viola RI*



Paso B: Encontrar una transacción T en EDB tal que T(D) implique cualquier hipótesis en Δ



2.2.2 Procedimiento de actualización

Dado un átomo L=p(t_1,...t_k) (resp. p*(t_1,...,t_k)) representaremos por L* el átomo p*(t_1,...,t_k) (resp. p(t_1,...,t_k)), además un átomo abducible es un átomo abducible básico si es de la forma B (t_1,...t_k) o B*(t_1,...t_k) donde B es un predicado básico.

Regla de selección segura:

Una regla de selección segura es una función (parcial) que dado un objetivo ←$L_1 \wedge ... \wedge L_k$ (K ≥ 1) devuelve un átomo L_i, i=1,...K tal que:

L_i no es abducible, o
L_i es abducible y base

Procedimiento de demostración abductivo:

• Una **derivación abductiva** de $(G_1\ \Delta_1)$ a $(G_n\ \Delta_n)$ vía una regla segura R es una secuencia

$$(G_1\ \Delta_1), (G_2\ \Delta_2),..., (G_n\ \Delta_n)$$

tal que para cada $i>1$ G_i tiene la forma $\leftarrow L_1 \wedge ... \wedge L_k$, $R(G_i)=L_j$ y $(G_{i+1}\ \Delta_{i+1})$ se obtiene de acuerdo a una de las siguientes reglas:

1) Si L_j no es abducible entonces $G_{i+1}= C$ y $\Delta_{i+1}=\Delta_i$ donde C es el resolvente de alguna cláusula en IDB* con G_i sobre el literal L_j

2) Si L_j es abducible y $L_j \in \Delta_i$ entonces $G_{i+1}=\leftarrow L_1 \wedge ... L_{j-1} \wedge L_{j+1} \wedge ... L_k$ y $\Delta_{i+1}=\Delta_i$

3) Si L_j es abducible básico, $L_j \notin \Delta_i$ y $L_j^* \notin \Delta_i$ entonces $G_{i+1}=\leftarrow L_1 \wedge ... L_{j-1} \wedge L_{j+1} \wedge ... L_k$ y $\Delta_{i+1}=\Delta_i \cup \{L_j\}$

4) Si L_j es abducible no-básico y $L_j \notin \Delta_i$ y existe una derivación de consistencia de $(\{\leftarrow L_j^*\}\ \Delta_i \cup \{L_j\})$ a $(\{\}\ \Delta')$ entonces $G_{i+1}=\leftarrow L_1 \wedge ... L_{j-1} \wedge L_{j+1} \wedge ... L_k$ y $\Delta_{i+1}=\Delta'$

Los pasos 1 y 2 son pasos de resolución SLD utilizando las reglas de IDB* y las hipótesis abductivas respectivamente. En los pasos 3 y 4 una nueva hipótesis es generada y añadida a Δ después de comprobar su consistencia (en el paso 3 esta prueba de consistencia es trivial).

• Una **derivación de consistencia** de $(F_1\ \Delta_1)$ a $(F_n\ \Delta_n)$ vía una regla segura R es una secuencia

$$(F_1\ \Delta_1), (F_2\ \Delta_2),..., (F_n\ \Delta_n)$$

tal que para cada $i>1$ F_i tiene la forma $\{\leftarrow L_1 \wedge ... \wedge L_k\} \cup F_i'$, $R(\leftarrow L_1 \wedge ... \wedge L_k)=L_j$ y (F_{i+1}, Δ_{1+1}) se obtiene de acuerdo a una de las siguientes reglas:

1) Si L_j no es abducible entonces $F_{i+1}= C' \cup F'$ donde C' es el conjunto de todos los resolventes de cláusulas de IDB* con $\leftarrow L_1 \wedge ... \wedge L_k$ sobre el literal L_j y $\notin C'$ y $\Delta_{i+1}=\Delta_i$

2) Si L_j es abducible básico, $L_j \in \Delta_i$ y $k>1$ entonces $F_{i+1}= \{\leftarrow L_1 \wedge ... \wedge L_{j-1} \wedge L_{j+1} \wedge ... \wedge L_k\} \cup F_i'$ y $\Delta_{i+1}=\Delta_i$

3) Si L_j es abducible básico y $L_j^* \in \Delta_i$ entonces $F_{i+1}=F_i'$ y $\Delta_{i+1}=\Delta_i$

4) si L_j es abducible básico, $L_j \notin \Delta_i$ y $L_j^* \notin \Delta_i$ entonces $F_{i+1}=F_i'$ y $\Delta_{i+1}=\Delta_i \cup \{L_j^*\}$

5) Si L_j es abducible no-básico, y existe una derivación abductiva de ($\leftarrow L_j^*\ \Delta_i$) a (Δ') entonces $F_{i+1}= F_i$ y $\Delta_{i+1}= \Delta$'

En el caso 1, la rama actual se divide en tantas ramas como resolventes de $\leftarrow L_1 \wedge ... \wedge L_k$ con cláusulas de IDB* sobre L_j existen; si la cláusula vacía es uno de estos resolventes la comprobación de la consistencia falla. El caso 2 es similar al anterior pero se resuelve con una hipótesis. En el caso 3 la rama es ya consistente y por lo tanto es eliminada. Los casos 4 y 5 corresponden a casos de abducción de átomos básicos y no-básicos respectivamente.

Transacciones asociadas a Δ

Sea Δ un conjunto de hipótesis generadas por el procedimiento abductivo, T_Δ es una transacción asociada a Δ si dado un estado E de EDB, para cada átomo abducible básico $L \in \Delta$ ($L^* \in \Delta$), $T_\Delta (E) \models L$ ($\neg L$).

Corrección y Completitud del procedimiento de actualización

• Sea IDB una base de datos localmente estratificada, U un requerimiento insertar(L) (resp. borrar(L)). Si ($\leftarrow$L {}) (resp. ($\leftarrow L^*$ {})) tiene una derivación abductiva a (Δ) entonces para cualquier EDB, T_Δ(EDB) satisface U, es decir L (resp. ¬L) es cierto en el modelo estable de IDB$\cup T_\Delta$(EDB)

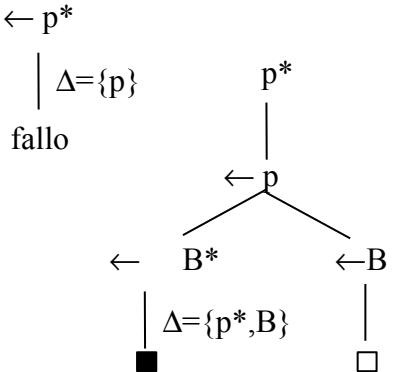
• Sea IDB una base de datos acíclica y permitida, U un requerimiento insertar(L) (resp. borrar(L))**.** Si existe una base de datos extensional EDB tal que L (resp. ¬L) es cierto en el modelo estable de IDB $\cup$ EDB entonces ($\leftarrow$L {}) (resp. ($\leftarrow L^*$ {})) tiene una derivación abductiva a (Δ) tal que EDB $\models$L' (¬L') para cada átomo abducible básico $L' \in \Delta$ ($L'^* \in \Delta$).

Ejemplo 2.3

IDB: p$\leftarrow$ ¬B EDB: B
p$\leftarrow$ B

IDB*:p$\leftarrow$ B* EDB: B
p$\leftarrow$ B
RI*: $\leftarrow p \wedge p^*, p \vee p^*$
$\leftarrow B \wedge B^*, B \vee B^*$

U: borrar(p)



El requerimiento de actualización no puede satisfacerse con una transacción sobre EDB.

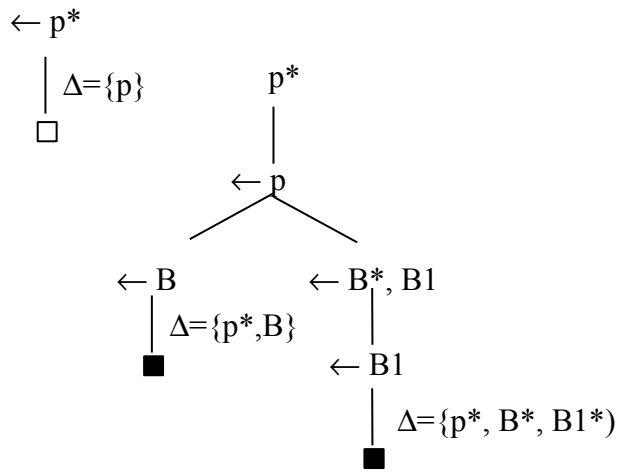
Ejemplo 2.4

IDB: $p \leftarrow \neg B, B_1$ EDB: B
$p \leftarrow B$ B1

IDB*: $p \leftarrow B^*, B_1$ EDB: B
$p \leftarrow B$ B1

RI*: $\leftarrow B \wedge B^*,\ B \vee B^*$
$\leftarrow B1 \wedge B1^*,\ B1 \vee B1^*$
$\leftarrow p \wedge p^*,\ p \vee p^*$

U: borrar(p)



T_Δ={borrar(B), borrar(B1)}

2.3 Método de Guessoum&Lloyd [GLl90]

2.3.1 Conceptos previos

Enunciado del problema: Dados una base de datos D, un requerimiento de actualización borrar(W) (resp. insertar(W)) (W es una fórmula), y un conjunto de restricciones de integridad RI tales que comp(D) es consistente, D satisface RI, y ∃ (W) es (resp. no es) consecuencia lógica de comp(D); encontrar una transacción T sobre D tal que comp(T(D)) es consistente, T(D) satisface RI, y ∃ (W) no es (resp. es) consecuencia lógica de comp(D).

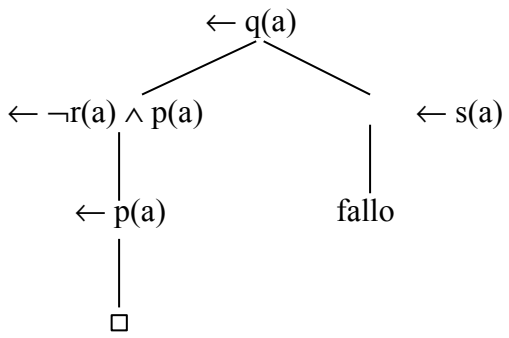
Fundamentos del procedimiento de actualización

- Procedimiento de Resolución SLDNF

Ejemplo 2.5

D: $q(x) \leftarrow \neg r(x) \wedge p(x)$ $q(a)$
$q(x) \leftarrow s(x)$ p(a

U: borrar(q(a))



T_1={borrar(q(a)), insertar(r(a))}
T_2={borrar(q(a)), borrar(p(a))}
T_3={borrar(q(a)), borrar(q(x)←¬r(x)∧p(x))}

2.3.2 Procedimiento de borrado de un átomo

Entrada: una base de datos consistente en llamada D, un átomo A y un conjunto de restricciones de integridad RI tal que ∃(A) es una consecuencia lógica de comp(D) y D satisface RI.

Salida: τ ={T: T es una transacción , T(D) satisface RI, y $\exists$(A) no es consecuencia lógica de comp(T(D)}.

Inicio t :=un árbol SLDNF finito para D$\cup${$\leftarrow$A}

τ_0:= { [acción(C_1),..., acción(C_n)]:

acción(C_i)= borrar(C_i) donde C_i es una sentencia de D utilizada como cláusula de entrada en una rama de t no fallada.

acción(C_i)= insertar(C_i) donde C_i es un hecho B tal que $\neg$B tiene éxito en una rama no fallada de t}

τ:= {T:T$\in$ τ_0, T(D)$\cup${$\leftarrow$A} tiene un árbol fallado finitamente y T(D) satisface RI}

fin

Observar que el procedimiento presentado puede no terminar en un tiempo finito. Por ejemplo al comprobar que T(D) satisface RI o al buscar un árbol fallado finitamente para T(D)$\cup${$\leftarrow$A}:

Ejemplo 2.6

D: p$\leftarrow$q
p$\leftarrow\neg$r
q
r$\leftarrow$q
r$\leftarrow$r

U:borrar(p)

Una posible transacción que satisface el requerimiento de actualización es T=[borrar(q)] pero el procedimiento anterior no terminaría al buscar un árbol fallado finitamente para T(D)$\cup$ {$\leftarrow$p}.

Corrección y completitud del procedimiento de borrado

• Sea D una base de datos (consistente en llamada) y A un átomo. Sea T una transacción, T$\in$ $\tau_{D,A}$ entonces $\exists$(A) no es consecuencia lógica de comp(T(D)).

• Sea D una base de datos (consistente en llamada), A un átomo y t un árbol SLDNF (no trivial) para D$\cup${$\leftarrow$A}. Sea D' la base de datos obtenida a partir de D eliminando cada

cláusula utilizada como primera cláusula de entrada en una rama no fallada de t. Entonces ∃ (A) no es una consecuencia lógica de comp(D')

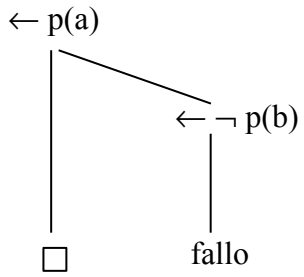
La condición de ser consistente en llamada no puede eliminarse.

Ejemplo 2.7

D: p(x)
p(a)←¬p(b)

U:borrar(p(a))

D': p(a)←¬p(b)



p(a) es consecuencia lógica de comp(D').

2.3.3 Procedimiento de inserción de un átomo

Entrada: una base de datos (consistente en llamada) D, un átomo A, y un conjunto de restricciones de integridad RI tal que ∃(A) no es consecuencia lógica de comp(D) y D satisface RI

Salida: τ ={T:T es una transacción, T(D) satisface RI, y ∃(A) es una consecuencia lógica de comp(T(D))

Inicio:

t:= un árbol SLDNF finito para D∪{←A}

τ_0={[insertar(A_1←),...,insertar(A_n←)]: ←A_1∧...∧ A_n es un objetivo definido en t}

τ={T:T ∈ τ_0,existe una refutación SLDNF para T(D)∪{←A}, y T(D) satisface RI}

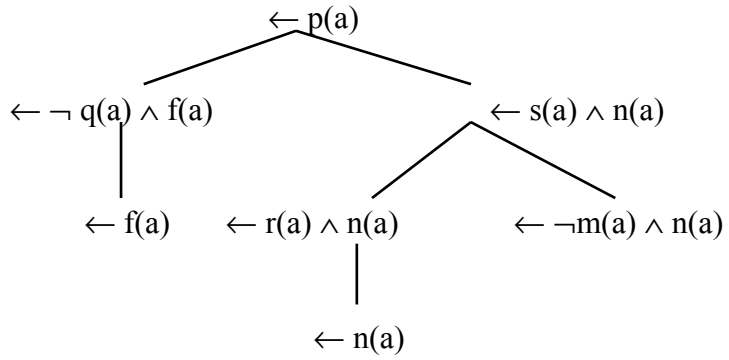
fin

Ejemplo 2.8

D: $p(x) \leftarrow \neg q(x) \wedge f(x)$
$p(x) \leftarrow s(x) \wedge n(x)$
$s(x) \leftarrow r(x)$

$s(x) \leftarrow \neg m(x)$
$r(a)$

U: insertar(p(a))



T_1=[insertar(p(a))]
T_2=[insertar(f(a))]
T_3=[insertar(s(a)), insertar(n(a))]
$T_{4=}$[insertar(r(a)), inserta(n(a))]
T_5=[insertar(n(a))]

Corrección y completitud del procedimiento de inserción

• Sea D una base de datos (consistente en llamada) y A un átomo. Sea T una transacción de $\tau_{D,A}$ entonces $\exists$(A) es consecuencia lógica de comp(T(D)).

• Sea D una base de datos (estricta), A un átomo y t un árbol SLDNF para $D \cup \{\leftarrow A\}$. Sea T la transacción que inserta los hechos A_1, A_2,..., A_n, tales que $\leftarrow A_1 \wedge ... \wedge A_n$ es un objetivo definido en t. Entonces $\exists$(A) es consecuencia lógica de comp(T(D)).

3. Comprobación de la integridad en bases de datos deductivas

3.1 Definición del problema

Una restricción de integridad es una propiedad que una base de datos debe *satisfacer* en cualquier instante para ser consistente con cierto modelo del mundo real.

La evolución en el tiempo de una base de datos se puede representar por una secuencia de estados donde, dado un estado D su sucesor D' se obtiene aplicando a D una transacción T. Dependiendo del número de estados implicados en la propiedad existen dos tipos de restricciones de integridad:

- **restricciones estáticas:** hacen referencia a un único estado de la base de datos. Estas restricciones restringen los estados válidos con independencia de la secuencia de los mismos.

- **restricciones dinámicas:** hacen referencia a dos o más estados de la base de datos. Estas restricciones restringen las secuencias de estados válidas. Un caso particular de restricciones dinámicas son las **restricciones de transición** que restringen dos estados consecutivos válidos.

La comprobación de la integridad en bases de datos consiste en *comprobar* si el par de estados (D,D') implicados en una transacción T *satisface* las restricciones de transición y si el estado final D' *satisface* las restricciones estáticas.

Si formalizamos un estado de la base de datos como una teoría de primer orden, [Rei84], [Llo87], [GMN84a], las restricciones estáticas se pueden representar (en la mayoría de los casos) por fórmulas bien formadas cerradas del lenguaje subyacente a la teoría. Sin embargo, debido a que las restricciones dinámicas hacen referencia a dos o más estados, éstas no se pueden expresar como fórmulas bien formadas de dicho lenguaje necesitándose para ello otros formalismos.

Ejemplos de restricciones estáticas son:

"Todo profesor está adscrito a un único departamento" (restricción de dependencia funcional)

$$\forall x \forall y \forall z(\text{adscrito}(x,y) \wedge \text{adscrito}(x,z) \rightarrow y = z)$$

"Todo empleado tiene un superior" (restricción de existencia)

$$\forall x(\text{empleado}(x) \rightarrow \exists y\ \text{superior}(y,x))$$

"El número de empleados en el departamento debe ser menor que 100" (restricción de agregado de valores)

no es expresable en lógica de primer orden

Ejemplos de restricciones dinámicas son:

"La edad de una persona no puede decrecer" (restricción de transición).

no es expresable en la teoría de primer orden que representa el estado

El presente capítulo se dedica al estudio de los métodos de comprobación de la integridad para restricciones estáticas en bases de datos deductivas. A partir de ahora, restricción de integridad hará referencia a restricción de integridad estática.

El problema de la comprobación de la integridad en bases de datos deductivas se puede enunciar de la forma siguiente:

Dados:

- el esquema (L,RI) de una base de datos deductiva, donde:
  - L es un lenguaje de primer orden, y
  - RI es el conjunto de restricciones de integridad, fórmulas cerradas de L.

- un estado D de la base de datos:
  $D = \{A \leftarrow B:$ A es un átomo, B es una fbf $\}$
  tal que D *satisface* W (para toda $W \in RI$).

- una transacción T formada por dos conjuntos de sentencias de base de datos:
  - T_{ins}: hechos y reglas que van a ser insertados por la transacción y
  - T_{del}: hechos y reglas que van a ser borrados por la transacción.

  Supondremos que los conjuntos T_{ins} y T_{del} son tales que $T_{ins} \cap T_{del} = \varnothing$, $T_{del} \subseteq D$ y $T_{ins} \cap D = \varnothing$.

- el estado D' resultante de aplicar a D la transacción T:
  $$D' = (D \cup T_{ins}) \setminus T_{del}.$$

Comprobar: D' *satisface* W (para toda $W \in RI$).

Un **método de comprobación de la integridad** es un procedimiento de decisión tal que, dado un estado D y una restricción de integridad W, decide con una respuesta binaria si/no si el estado D satisface/viola la restricción W.

La forma más sencilla de comprobar las restricciones estáticas es evaluar cada una de ellas después de la transacción; sin embargo esta aproximación puede ser muy costosa en bases de datos voluminosas ya que, usualmente, las restricciones representan propiedades generales sobre la base de datos y suelen estar representadas por fórmulas con variables cuantificadas universalmente cuyos dominios pueden ser muy extensos. La comprobación de

la integridad podría simplificarse si consideráramos sólo los "cambios" que la transacción ha producido en la base de datos.

Todos los métodos propuestos para simplificar la comprobación de la integridad suponen que la base de datos era íntegra (es decir, satisfacía todas las restricciones de integridad) antes de la transacción. Apoyándose en esta hipótesis, los métodos comprueban sólo instancias de las restricciones generadas a partir de las *actualizaciones* (inserciones y borrados) inducidas por la transacción, evitando comprobar instancias que ya se satisfacían antes de la transacción y que además no se ven afectadas por ésta.

La comprobación simplificada de la integridad es un problema clásico en bases de datos; los primeros métodos fueron propuestos para la comprobación de restricciones estáticas en bases de datos relacionales (apartado 3.2), extendiéndose posteriormente a las bases de datos deductivas (apartado 3.3).

3.2 Comprobación de la integridad en bases de datos relacionales

En este apartado se presentan dos métodos para simplificar la comprobación de la integridad en bases de datos relacionales.

3.2.1 Restricciones estáticas: Método de Nicolas [Nic82]

Concepto de satisfacción

En el trabajo original de Nicolas, [Nic79], [Nic82], una base de datos relacional se formaliza como una interpretación I del conjunto de fórmulas de primer orden que representan las restricciones de integridad; dicha interpretación se construye a partir de la extensión de la base de datos [NG78]. En esta formalización el concepto de satisfacción utilizado es el siguiente: D satisface W sii $\models_I W$.

Desde la teoría de la demostración se puede construir una formalización lógicamente equivalente a la anterior donde la base de datos se representa por una teoría Tr tal que $\models_I W$ sii $Tr \models W$.

Esta formalización consiste en:

- (L,RI) es el esquema de la base de datos, donde:
  - L es un lenguaje de primer orden, con un conjunto finito de símbolos de constante (construido a partir de la extensión de la base de datos); un conjunto

finito de símbolos de predicado (uno por cada identificador de relación) y sin símbolos de función

- RI es el conjunto de restricciones de integridad, fórmulas cerradas de L.

• D = { $p(c_1,...,c_n)\leftarrow$: la tupla $(c_1,...,c_n)$ aparece en la extensión de p} es un estado de la base de datos.

• La semántica asumida es la de la compleción más el axioma de cierre de dominio (ACD). La teoría que representa el estado D es Tr=comp(D)$\cup${ACD} y la semántica declarativa asociada viene definida por el conjunto COMP(D)={L: L es un literal base, Tr$\models$ L}. El axioma de cierre de dominio [Rei78a] define explícitamente las constantes del lenguaje y tiene la forma $\forall x((x=a_1)\vee...\vee(x=a_k))$ donde $a_1,...,a_k$ son las únicas constantes del lenguaje L.

• En esta formalización el concepto de satisfacción es: D satisface W sii Tr $\models$ W.

Conceptos previos

Antes de enunciar el teorema de simplificación de Nicolas vamos a ilustrar el método con un ejemplo.

Ejemplo 3.1

Sea:

• D un estado de la base de datos:

$D = \{p(1,1)\leftarrow,\ p(2,2)\leftarrow$
$q(1,1,1)\leftarrow,\ q(1,2,2)\leftarrow,\ q(2,1,1)\leftarrow,\ q(1,3,3)\leftarrow\}$.

• RI el conjunto de restricciones de integridad:

$RI = \{W_1 = \forall x\forall y(p(x,y)\rightarrow\exists z q(z,x,y))$
$W_2 = \exists z\forall x\forall y(p(x,y)\rightarrow q(z,x,y))\}$.

D es íntegro.

• T la siguiente transacción:

$T_{ins}=\{p(3,3)\leftarrow\}$, $T_{del}=\{q(2,1,1)\leftarrow\}$.

El método de Nicolas consiste en: dado un estado íntegro y una transacción, obtener *instancias simplificadas de las restricciones de integridad relevantes para la transacción*, que será suficiente comprobar en D' para asegurar su integridad.

Restricciones de integridad relevantes para T:

Si W es una restricción de integridad de rango restringido [Nic82] (independiente del dominio), podemos afirmar que:

"W es relevante respecto a la inserción (resp. borrado) de la tupla $R(e_1,e_2,...,e_n)\leftarrow$ si y sólo si $R(e_1,e_2,...,e_n)$ es unificable con un átomo que ocurre negativamente (resp. positivamente) en W".

Un átomo A ocurre negativamente (resp. positivamente) en una fórmula W sii $\neg$A (resp. A) aparece en la forma prenexa normal de W.

En el ejemplo 3.1, W_1 y W_2 son relevantes respecto a las dos operaciones de la transacción.

Instancias de las restricciones de integridad:

Sea:

• W una restricción de integridad relevante respecto a la inserción (resp. borrado) de $R(e_1,e_2,...,e_n)\leftarrow$.

• θ el unificador más general que unifica $R(e_1,e_2,...,e_n)$ con un átomo que ocurre negativamente (resp. positivamente) en W.

• ϕ la restricción de θ a aquellas variables cuantificadas universalmente no precedidas de un cuantificador existencial.

Entonces, definimos una instancia de W generada por la inserción (resp. borrado) de $R(e_1,e_2,...e_n)\leftarrow$ como la fórmula $W\phi$.

La necesidad de restringir la sustitución θ se puede comprobar en el ejemplo 3.1:

a) La sustitución θ no debe aplicarse a las variables cuantificadas existencialmente.

La operación borrar($q(2,1,1)\leftarrow$) genera una instancia de W_1:

$\theta=\{z/2, x/1, y/1\}$ $W_1\theta=p(1,1)\rightarrow q(2,1,1)$.

D' no satisface $W_1\theta$. y sin embargo D' satisface W_1.

b) La sustitución θ no debe aplicarse a las variables cuantificadas universalmente precedidas de un cuantificador existencial.

La operación borrar(q(2,1,1)←) genera una instancia de W_2:

$\theta=\{z/2, x/1, y/1\}$ $W_2\theta=p(1,1)\rightarrow q(2,1,1)$.

D' no satisface $W_2\theta$ y sin embargo D' satisface W_2.

En nuestro ejemplo las instancias de las restricciones generadas por la transacción son:

Para W_1:

insertar(p(3,3)←): $\phi_1=\{x/3, y/3\}$, $W_1\phi_1= p(3,3)\rightarrow \exists zq(z,3,3)$

borrar(q(2,1,1)←): $\phi_2=\{x/1, y/1\}$, $W_1\phi_2= p(1,1)\rightarrow \exists zq(z,1,1)$.

Para W_2:

W_2 es relevante para la transacción pero no se generan instancias de ella.

Simplificación de las instancias:

Las instancias de W pueden simplificarse reemplazando las ocurrencias de $R(e_1,e_2,...,e_n)$ por el valor cierto (resp. falso) si la transacción ha insertado (resp. borrado) la tupla $R(e_1,e_2,...,e_n)\leftarrow$ y aplicando reglas de absorción.

En nuestro ejemplo:

$(W_1\phi_1)_s = \exists zq(z,3,3)$

$(W_1\phi_2)_s = p(1,1)\rightarrow\exists zq(z,1,1)$.

Comprobación de la integridad (método de Nicolas):

RI = $\{W_1 = \forall x\forall y(p(x,y)\rightarrow\exists zq(z,x,y))$

$W_2 = \exists z\forall x\forall y(p(x,y)\rightarrow q(z,x,y))\}$.

$W_{1S}=(W_1\phi_1)_s \wedge(W_1\phi_2)_s = \exists zq(z,3,3)\wedge(p(1,1)\rightarrow\exists zq(z,1,1))$.

D' satisface W_{1S}, entonces D' satisface W_1.

D' satisface W_2.

Teorema de simplificación:

Sea:

• (L,RI) el esquema de una base de datos relacional donde:

- L es un lenguaje de primer orden, con conjuntos finitos de símbolos de constante y de predicado y sin símbolos de función

- RI = {W: $W=\forall x_1 \forall x_2 ... \forall x_n W'$} es el conjunto de restricciones de integridad, fórmulas cerradas de L, de rango restringido y en forma prenexa normal ($x_1,x_2,...,x_n$ son las variables de W universalmente cuantificadas y que no están precedidas por un cuantificador existencial).

• D = {A←: A es un átomo base} un estado de la base de datos.

• La semántica asumida es la de la compleción más el axioma de cierre de dominio. La teoría que representa el estado D es $Tr = comp(D) \cup \{ACD\}$.

• T una transacción formada por un conjunto T_{ins} de inserciones de tuplas y un conjunto T_{del} de borrados de tuplas, representadas como cláusulas de la forma A←, donde A es un átomo base. La transacción puede cambiar el lenguaje.

• D' es el estado resultante de aplicar a D la transacción T: $D' = (D \cup T_{ins}) \setminus T_{del}$.

• W una restricción de integridad tal que D satisface W ($Tr \models W$).

• $\Theta = \{\theta$: θ es la restricción a $x_1,x_2,...,x_n$ del unificador más general entre un átomo que ocurre positivamente (resp. negativamente) en W y un átomo $R(e_1,e_2,...,e_n)$ tal que la transacción ha borrado (resp. insertado) la tupla $R(e_1,e_2,...,e_n)\leftarrow\}$.

• $\Theta_s = \{\theta_i: \theta_i \in \Theta$ y $\exists / \theta_j \in \Theta$ $(i \neq j)$ tal que θ_j subsume a $\theta_i\}$

• W_s es la fórmula resultante de aplicar a $W\theta_1 \wedge W\theta_2 \wedge ... \wedge W\theta_n$ (para todo $\theta_i \in \Theta_s$) las siguientes reglas de simplificación:

- sustituir cada ocurrencia de $R(e_1,e_2,...e_n)$ por el valor cierto (resp. falso) si la transacción ha insertado (resp. borrado) la tupla $R(e_1,e_2,...,e_n)\leftarrow$.

- aplicar reglas de absorción.

Se cumple: D' satisface W sii D' satisface W_s.

Si Θ_s es el conjunto vacío, la restricción W no se ve afectada por la transacción es decir, ésta sigue satisfaciéndose en D'.

Si $\varepsilon \in \Theta_s$, la restricción W no se puede simplificar y debe ser evaluada en D' en su forma original (ε representa la sustitución identidad).

3.2.2 Restricciones dinámicas: Método de Nicolas y Yazdanian [NY78]

Para poder expresar las restricciones dinámicas en el mismo marco formal (lógica de primer orden), Nicolas y Yazdanian propusieron extender el lenguaje de la base de datos con unos predicados de actualización.

Por cada predicado R del esquema relacional se incorporan tres nuevos predicados: UPD-R, ENT-R, DEL-R, de aridad respectivamente 2xn, n, n, donde n es la aridad de R.

La extensión de estos predicados es vacía hasta que se realiza una operación de modificación, inserción o borrado respectivamente sobre R, en este caso, por cada una de las operaciones, aparece en la extensión de dichos predicados una tupla que representa la actualización realizada:

UPD-R($a_1,a_2,...,a_n,a'_1,a'_2,...,a'_n$) representa que la tupla $<a_1,a_2,...,a_n>$ ha sido modificada, siendo el nuevo valor $< a'_1,a'_2,...,a'_n>$.

ENT-R($a_1,a_2,...,a_n$) y DEL-R($a_1,a_2,...,a_n$) representan respectivamente, la inserción y el borrado de la tupla $<a_1,a_2,...,a_n>$ en R.

Con esta extensión del lenguaje las restricciones de transición pueden expresarse como fórmulas bien formadas cerradas, que deben ser comprobadas en la teoría que representa el nuevo estado de la base de datos.

3.3 Comprobación de la integridad en bases de datos deductivas

Siguiendo las ideas propuestas por Nicolas [Nic82] para bases de datos relacionales, los métodos de comprobación de la integridad para bases de datos deductivas se basan en la idea común de evaluar instancias simplificadas de las restricciones de integridad, obtenidas a partir de las *actualizaciones* generadas por la transacción. La existencia de reglas deductivas introduce un nuevo problema respecto al caso relacional, ya que las *actualizaciones* generadas por la transacción no son sólo las explícitamente requeridas por ésta, sino también las inducidas por la presencia de reglas deductivas en la base de datos. Los métodos se diferencian entre sí en la estrategia seguida para el cálculo de dichas *actualizaciones* y la instanciación a partir de ellas de las restricciones de integridad.

La estructura del apartado 3.3 es la siguiente: en 3.3.1 se propone una definición de los conceptos de satisfacción existentes en bases de datos deductivas independiente de la semántica asumida; en 3.3.2 se propone un algoritmo para la comprobación simplificada de la integridad independiente de la estrategia seguida por cada método particular; en 3.3.3 se proponen definiciones de los conceptos de corrección y completitud de un método; en 3.3.4 se incluye una revisión de los principales métodos para la comprobación de la integridad propuestos en la literatura y en 3.3.5 se hace un análisis de los mismos.

Con el fin de presentar los fundamentos teóricos de la comprobación de la integridad en un marco uniforme e independiente de la semántica asumida por cada método, vamos a representar un estado D de la base de datos por una teoría de primer orden. Esta teoría Tr es la siguiente:

a) Si asumimos la semántica de la compleción:

$$Tr = comp(D) = D \cup \{\text{ axiomas de compleción para cada predicado de L }\} \cup \{\text{ axiomas de igualdad }\}$$

La semántica declarativa asociada viene definida por el conjunto COMP(D)={L: L es un literal base, Tr$\models$L}

b) Si asumimos la semántica de un modelo minimal M de D, la semántica declarativa asociada viene definida por el conjunto MIN(D)={L: L es un literal base, $\models_M$ L}. En este caso el estado D puede representarse por la teoría:

$$Tr = comp(Tr') \cup \{ACD\}$$

donde Tr' = {A: A es un átomo base y A$\in$ M} y ACD es el axioma de cierre de dominio. El axioma de cierre de dominio que fue definido por Reiter para un lenguaje sin símbolos de función en [Rei78a], se define en el caso general de la forma siguiente [Llo87]:

$$\forall x((x=a_1) \vee ... \vee (x=a_k) \vee \exists x_1 ... \exists x_m (x=f_1(x_1,...,x_m)) \vee ... \vee \exists y_1 ... \exists y_n (x=f_r(y_1,...,y_n)))$$

donde $a_1,...,a_k$ son las constantes de L y $f_1,...,f_r$ son los símbolos de función de L. Informalmente hablando, dicha teoría es la compleción del modelo M de D. Es importante resaltar que para todo literal base L se cumple Tr $\models$ L sii $\models_M$ L.

3.3.1 Concepto de satisfacción

Sea:

- (L, RI) un esquema de base de datos deductiva donde:
  - L es un lenguaje de primer orden y
  - RI es el conjunto de restricciones de integridad, fórmulas cerradas de L.
- D un estado de la base de datos:
  D = {A← B: A es un átomo, B es una fbf}.

Para bases de datos deductivas existen dos definiciones del concepto de satisfacción [SK87]. Sea W (W∈ RI) una restricción de integridad y Tr la teoría que representa el estado D en la semántica asumida; supongamos que Tr es una teoría consistente (Tr tiene un modelo):

a) punto de vista de la **demostración:**

D satisface W sii Tr $\models$ W.

b) punto de vista de la **consistencia**:

D satisface W sii Tr ∪ {W} es consistente.

El concepto de violación se define en términos del concepto de satisfacción:

D viola W sii no(D satisface W).

Diremos que un estado D es **íntegro** si, para toda restricción W perteneciente a RI, D satisface W.

Para bases de datos para las que Tr es consistente, satisfacción desde el punto de vista de la demostración implica satisfacción desde el punto de vista de la consistencia. El punto de vista de la demostración y el punto de vista de la consistencia coinciden cuando Tr es categórica. Una teoría es categórica cuando para toda fórmula cerrada W se cumple Tr |=W o Tr |= ¬W. Es importante destacar que, en la semántica del modelo minimal, la teoría que representa el estado D es categórica y por lo tanto en dicha semántica ambos conceptos de satisfacción coinciden.

En [Rei90] se presenta un nuevo concepto de satisfacción basado en lógica modal [Tha89].

Ejemplo 3.2

$D = \{p(a) \leftarrow q(x) \wedge \neg r(x),$
$q(a) \leftarrow,$
$r(x) \leftarrow r(x)\}.$

Si asumimos la semántica de la compleción:

$Tr = \{\forall y(p(y) \leftrightarrow y=a \wedge \exists x(q(x) \wedge \neg r(x))),$
$\forall x(q(x) \leftrightarrow x=a),$
$\forall x(r(x) \leftrightarrow r(x))\}.$

Desde el punto de vista de la consistencia D satisface: $W_1=q(a)$, $W_2=r(a)$, $W_3=p(a)$, $W_4=\neg r(a)$. Desde el punto de vista de la demostración D satisface: $W_1=q(a)$.

Si asumimos la semántica del punto fijo iterado:

$Tr = \{\forall x(p(x) \leftrightarrow x=a),$
$\forall x(q(x) \leftrightarrow x=a),$
$\forall x(\neg r(x)),$
$\forall x(x=a)\}.$

D satisface $W_1=q(a)$ y $W_2=p(a)$ en los dos conceptos de satisfacción.

3.3.2 Fases en la comprobación de la integridad

Sea:

- (L,RI) el esquema de una base de datos deductiva donde:
  - L es un lenguaje de primer orden y
  - RI = { W: $W=\forall x_1 \forall x_2 ... \forall x_n W'$} es el conjunto de restricciones de integridad, fórmulas cerradas de L en forma prenexa normal.

- D un estado de la base de datos:
  $D = \{A \leftarrow B:$ A es un átomo, B es una fbf $\}$

  tal que D satisface W (para toda $W \in$ RI) en cualquier concepto de satisfacción.

- T una transacción formada por dos conjuntos de cláusulas:
  - T_{ins}: hechos y reglas que van a ser insertados por la transacción

- T_{del}: hechos y reglas que van a ser borrados por la transacción
  ($T_{ins} \cap T_{del} = \varnothing$, $T_{del} \subseteq D$ y $T_{ins} \cap D = \varnothing$) .

• D' el estado resultante de aplicar a D la transacción T:

$$D' = (D \cup T_{ins}) \setminus T_{del}$$

Independientemente de la estrategia seguida por cada método, todos ellos simplifican la comprobación de la integridad según el siguiente esquema:

Hipótesis: D es íntegro.

Comprobación de la integridad:

FASE I: Fase de Generación

Paso1: Cálculo de conjuntos de literales que "capturen" la diferencia entre los estados consecutivos D y D'.
Paso 2: Identificación de las restricciones relevantes.
Paso 3: Instanciación de las restricciones relevantes.
Paso 4: Simplificación de las instancias de las restricciones relevantes.

FASE II: Fase de Evaluación

Paso 5: Comprobación en D' de las instancias simplificadas de las restricciones relevantes.

El **Paso 1** consiste en calcular conjuntos de literales que "capturan" la diferencia entre los estados consecutivos D y D'. Esta diferencia viene definida por los conjuntos :

$INS_{D,D'} = \{ L: L \text{ es un literal base, } Tr' \models L, Tr \not\models L\}$
$DEL_{D,D'} = \{ L: L \text{ es un literal base, } Tr \models L, Tr' \not\models L\}$

donde Tr (resp. Tr') es la teoría que representa el estado D (resp. D') en la semántica asumida.

Los conjuntos $INS_{D,D'}$ y $DEL_{D,D'}$ representan el cambio producido por la transacción. Para analizar el significado de los elementos de dichos conjuntos, definimos los subconjuntos siguientes:

$INS_{D,D'}^{+}$ (resp. $DEL_{D,D'}^{+}$) = {A: A es un átomo base, $A \in INS_{D,D'}$ (resp. $DEL_{D,D'}$)}

$INS_{D,D'}^{-}$ (resp. $DEL_{D,D'}^{-}$) = {A: A es un átomo base, $\neg A \in INS_{D,D'}$ (resp. $DEL_{D,D'}$)}.

Es inmediato comprobar que entre los conjuntos anteriores existen las siguientes relaciones, donde "falso" significa que ¬A es consecuencia lógica de la correspondiente teoría, "cierto" significa que A es consecuencia lógica de la correspondiente teoría, e "indefinido" significa que ni A ni ¬A son consecuencia lógica de la correspondiente teoría.

$INS_{D,D'}^{+} \cap DEL_{D,D'}^{-} = \{A: Tr \models \neg A, Tr' \models A\}$
(falso ⟶ cierto)

$INS_{D,D'}^{+} \setminus DEL_{D,D'}^{-} = \{A: Tr \not\models A, Tr \not\models \neg A, Tr' \models A\}$
(indefinido ⟶ cierto)

$DEL_{D,D'}^{-} \setminus INS_{D,D'}^{+} = \{A: Tr \models \neg A, Tr' \not\models \neg A, Tr' \not\models A\}$
(falso ⟶ indefinido)

$INS_{D,D'}^{-} \cap DEL_{D,D'}^{+} = \{A: Tr \models A, Tr' \models \neg A\}$
(cierto ⟶ falso)

$INS_{D,D'}^{-} \setminus DEL_{D,D'}^{+} = \{A: Tr \not\models A, Tr \not\models \neg A, Tr' \models \neg A\}$
(indefinido ⟶ falso)

$DEL_{D,D'}^{+} \setminus INS_{D,D'}^{-} = \{A: Tr \models A, Tr' \not\models \neg A, Tr' \not\models A\}$
(cierto ⟶ indefinido).

Si Tr es categórica, como sucede cuando se utiliza una semántica de modelo minimal, se cumple:

$INS_{D,D'}^{-} = DEL_{D,D'}^{+}$ y
$DEL_{D,D'}^{-} = INS_{D,D'}^{+}$

y entonces, los conjuntos que definen la diferencia pueden definirse, sin pérdida de información, de la forma:

$INS_{D,D'} = \{ A: A$ es un átomo base, $Tr' \models A, Tr \not\models A \}$ (inserciones)
$DEL_{D,D'} = \{ A: A$ es un átomo base, $Tr \models A, Tr' \not\models A \}$ (borrados).

Debido a que el cálculo de $INS_{D,D'}$ y $DEL_{D,D'}$ puede ser muy costoso, algunos métodos trabajan con conjuntos de literales (no necesariamente base) que "capturan" la diferencia entre D y D', es decir trabajan con supraconjuntos de los conjuntos que definen la diferencia real.

En algunas ocasiones, denominaremos **actualizaciones reales** a los elementos de los conjuntos $INS_{D,D'}$ y $DEL_{D,D'}$; y **actualizaciones potenciales** a los elementos de los conjuntos utilizados para "capturar" la diferencia entre D y D' cuando estos han sido obtenidos sin acceder a la base de datos explícita (hechos) .

El **Paso 2** consiste en identificar por unificación las restricciones de integridad relevantes respecto a los elementos de los conjuntos obtenidos en el Paso 1. Una restricción W es relevante respecto a la transacción T si y sólo si existe un elemento de $INS_{D,D'}^{+} \cup DEL_{D,D'}^{-}$ (resp. $INS_{D,D'}^{-} \cup DEL_{D,D'}^{+}$) unificable con un átomo que ocurre negativamente (resp. positivamente) en W.

Es interesante observar que el conjunto $INS_{D,D'}^{+} \cup DEL_{D,D'}^{-}$ representa la información añadida por la transacción, y que el conjunto $INS_{D,D'}^{-} \cup DEL_{D,D'}^{+}$ representa la información borrada por la transacción.

El **Paso 3** consiste en instanciar las restricciones de integridad relevantes, utilizando para ello las sustituciones obtenidas en las unificaciones del paso anterior. Estas sustituciones pertenecen a uno de los siguientes conjuntos:

$\Theta = \{\theta$: θ es la restricción a $x_1,x_2,...,x_n$ de un unificador más general entre un átomo que ocurre positivamente en W y un átomo de $INS_{D,D'}^{-} \cup DEL_{D,D'}^{+}\}$

$\Psi = \{\varphi$: φ es la restricción a $x_1,x_2,...,x_n$ de un unificador más general entre un átomo que ocurre negativamente en W y un átomo de $INS_{D,D'}^{+} \cup DEL_{D,D'}^{-}\}$.

Entonces, para cada sustitución $\phi \in \Psi \cup \Theta$ existe una instancia de W definida de la forma $W\phi$.

El **Paso 4** consiste en simplificar las instancias de las restricciones relevantes.

El **Paso 5** consiste en comprobar en el nuevo estado las instancias simplificadas obtenidas en el Paso 4, según el concepto de satisfacción asumido por el método.

3.3.3 Corrección y completitud

Como hemos dicho en 2.1, un método M para la comprobación de la integridad es un procedimiento de decisión tal que, dado un estado D y una restricción de integridad W, decide con una respuesta binaria si/no si el estado D satisface/viola la restricción W. Para que un método M sea "válido" debe demostrarse que es correcto y completo en el sentido que se expone a continuación.

Sea:

• (L, RI) un esquema de base de datos deductiva donde:

- L es un lenguaje de primer orden y
- RI es el conjunto de restricciones de integridad, fórmulas cerradas de L.

• D un estado de la base de datos:

D = {A← B: A es un átomo, B es una fbf}.

• M un método de comprobación de la integridad. D satisface$_M$ (resp. viola$_M$) W significa que el método M decide que el estado D satisface (resp. viola) la restricción W (W∈ RI).

• CS el concepto de satisfacción asumido por el método M. D satisface$_{CS}$ (resp. viola$_{CS}$) W significa que el estado D satisface (resp. viola) la restricción W (W∈ RI) en el concepto de satisfacción CS.

Definimos los conceptos de corrección y completitud de un método M de la forma siguiente:

- **un método M es correcto** cuando se cumple:

si D satisface$_M$ W entonces D satisface$_{CS}$ W (correcto para satisfacción)
si D viola$_M$ W entonces D viola$_{CS}$ W (correcto para violación).

- **un método M es completo** cuando se cumple:

si D satisface$_{CS}$ W entonces D satisface$_M$ W (completo para satisfacción)
si D viola$_{CS}$ W entonces D viola$_M$ W (completo para violación).

Según las definiciones de satisfacción y de violación dadas anteriormente (2.3.1), satisface$_{CS}$ y viola$_{CS}$ son dos conceptos opuestos, es decir D viola$_{CS}$ W sii no(D satisface$_{CS}$ W). Sin embargo, en algunos métodos satisface$_M$ y viola$_M$ no son opuestos; por ejemplo, un método basado en el procedimiento SLDNF puede tener definidos estos dos conceptos en base a la existencia de refutaciones o de árboles fallados finitamente, que como es sabido, no son problemas opuestos; excepto para algunas clases de bases de datos, la inexistencia de una refutación SLDNF para D∪{R} no implica la existencia de un árbol fallado finitamente para D∪{R}. Por este motivo, definimos los conceptos de corrección y completitud de un método en términos de la corrección y la completitud para satisfacción, y la corrección y la completitud para violación. Cuando un método es correcto y completo significa que satisface$_M$ y viola$_M$ son conceptos opuestos, es decir: D viola$_M$ W sii no(D satisface$_M$ W).

Si el método M tiene una clara separación entre la Fase de Generación y la Fase de Evaluación, las propiedades de corrección y completitud del método pueden enunciarse en términos de corrección y completitud de cada una de las dos fases.

La **Fase de Generación** de un método consiste en la obtención de un conjunto de instancias simplificadas de las restricciones de integridad relevantes respecto a la transacción, que será suficiente evaluar en el nuevo estado para comprobar la integridad. Estas instancias se obtienen a partir de las actualizaciones inducidas por la transacción. Sea **Ω** el conjunto de instancias simplificadas de la restricción W obtenidas en la Fase de Generación del método M, $\Omega=\{W_s\}$. Definimos los conceptos de corrección y completitud de la Fase de Generación de M de la forma siguiente:

• la Fase de Generación de M es **correcta** cuando se cumple:

si D satisface$_{CS}$ W_s (para toda W_s en Ω) entonces D satisface$_{CS}$ W

• la Fase de Generación de M es **completa** cuando se cumple:

si D satisface$_{CS}$ W entonces D satisface$_{CS}$ W_s (para toda W_s en Ω)

La **Fase de Evaluación** de un método consiste en comprobar con algún procedimiento de evaluación el conjunto de instancias simplificadas obtenidas en la Fase de Generación. Sea W una fórmula bien formada cerrada del lenguaje L y sea PE el procedimiento de evaluación utilizado en la Fase de Evaluación del método M. D satisface$_{PE}$ (resp. viola$_{PE}$) W significa que el procedimiento de evaluación PE decide que D satisface (resp. viola) W. Definimos los conceptos de corrección y completitud de la Fase de Evaluación de M de la forma siguiente:

• la Fase de Evaluación de M es **correcta** cuando se cumple:

si D satisface$_{PE}$ W entonces D satisface$_{CS}$ W (correcta para satisfacción)
si D viola$_{PE}$ W entonces D viola$_{CS}$ W (correcta para violación).

• la Fase de Evaluación de M es **completa** cuando se cumple:

si D satisface$_{CS}$ W entonces D satisface$_{PE}$ W (completa para satisfacción)
si D viola$_{CS}$ W entonces D viola$_{PE}$ W (completa para violación).

Un método M es correcto y completo cuando ambas fases son correctas y completas.

3.3.4 Métodos para la comprobación de la integridad

A continuación se presenta una selección de los principales métodos para la comprobación de la integridad propuestos en la literatura; algunos métodos existentes ([AIM88], [BD88]), no han sido incluidos en la relación porque consisten en variantes de los presentados. La exposición se hace por orden cronológico y posteriormente se hará un análisis de los mismos. De cada método se presentan los siguientes aspectos:

- concepto de satisfacción,
- representación y propiedades de las restricciones,
- tipo y propiedades de la base de datos,
- estrategia del método y
- teorema o algoritmo de simplificación.

Resúmenes de los métodos existentes se pueden encontrar en [DW89b], [CM91] y [BMM90].

3.3.4.1 Método de Decker [Dec86]

Concepto de Satisfacción

Este método utiliza la siguiente definición de satisfacción:

D satisface W sii W es "derivable" de D (D |-- W) .

Conceptos Previos

Actualizaciones Reales

El método trabaja con conjuntos de actualizaciones reales asociados a una cláusula C, C= A←B, definidos de la forma:

D^C= { Aθ: Aθ es base, Bθ es derivable de D' y Aθ no es derivable de D }

D_C= { Aθ: Aθ es base, Bθ es derivable de D y Aθ no es derivable de D' }.

Algoritmo de Simplificación

Sea:

• (L,RI) el esquema de una base de datos deductiva, donde:

- L es un lenguaje de primer orden y
- RI es el conjunto de restricciones de integridad, fórmulas cerradas de L de rango restringido.

• D un estado de la base de datos:
$D=\{ A \leftarrow L_1 \wedge L_2 \wedge ... \wedge L_n$: A es un átomo, L_i es un literal, $n \geq 0\}$.

• T una transacción formada por operaciones de la forma:

- insertar(C)
- borrar(C)

donde C es una cláusula.

• D' el estado resultante de aplicar a D la transacción T:
$D' = D \cup \{C: \text{insertar}(C) \in T\} \setminus \{C: \text{borrar}(C) \in T\}$.

• W una restricción de integridad tal que D satisface W.

• RA={insertar $\underline{L}$ sólo si $W^{\underline{L}}$: L es un átomo que ocurre negativamente en W}
$\cup$
{borrar $\underline{L}$ sólo si $W_{\underline{L}}$: L es un átomo que ocurre positivamente en W}.

el conjunto de restricciones auxiliares asociado a la restricción de integridad W, donde:

- $\underline{L}$ se obtiene a partir de L renombrando las variables de L cuantificadas existencialmente o cuantificadas universalmente precedidas de un cuantificador existencial y

- las formas simplificadas $W^{\underline{L}}$ (resp. $W_{\underline{L}}$) se obtienen reemplazando cada ocurrencia positiva de $\underline{L}$ en W por el valor cierto (resp. falso), cada ocurrencia negativa de $\underline{L}$ en W por el valor falso (resp. cierto) y aplicando las correspondientes reglas de absorción.

• DP = {ocurre_positivo(L, R): L es un átomo, R es una regla de D, L aparece en el cuerpo de R}.

• DN = {ocurre_negativo (L, R): L es un átomo, R es una regla de D, ¬L aparece en el cuerpo de R}.

La comprobación simplificada de la integridad se realiza aplicando el siguiente algoritmo α con argumento insertar(C) (resp. borrar(C)) para cada operación de la transacción.

ALGORITMO α

PASO 1: para cada átomo L* de D^C (resp. D_C) y para cada restricción auxiliar **insertar L sólo si F** (resp. **borrar L sólo si F**) tal que L unifica con L* con unificador más general θ, evaluar Fθ en D'. Si Fθ se evalúa a falso, parar (la transacción viola la integridad).

PASO 2: para cada átomo L* de D^C (resp. D_C) y para cada cláusula ocurre_positivo(L,R) tal que L unifica con L* con unificador más general μ, llamar al algoritmo con argumento insertar(Rμ) (resp. borrar(Rμ)).

PASO 3: para cada átomo L* de D^C (resp. D_C) y para cada cláusula ocurre_negativo(L,R) tal que L unifica con L* con unificador más general φ, llamar al algoritmo con argumento borrar(Rφ) (resp. insertar(Rφ)).

Cuando todas las operaciones de la transacción han sido procesadas por el algoritmo sin que se detecte una violación de la integridad, se puede afirmar que D' es íntegro.

3.3.4.2 Método de Lloyd, Sonnenberg y Topor [LST87]

Concepto de satisfacción

Este método utiliza el punto de vista de la demostración con Tr = comp(D) $\cup$ {ACD}:

D satisface W sii Tr $\models$ W.

Conceptos previos

El método presentado en [LST87] es una versión revisada del método presentado en [LT85]. Este método trabaja con dos conjuntos de actualizaciones potenciales (inserciones y borrados potenciales) que pueden ser calculados sin acceder a la base de datos extensional.

Actualizaciones potenciales

Sea T una transacción y D y D' los estados consecutivos relacionados con T tales que D $\subseteq$D'; entonces, se definen $pos_{D,D'}$ (conjunto de inserciones potenciales) y $neg_{D,D'}$ (conjunto de borrados potenciales) inductivamente de la forma siguiente:

$pos_{D,D'}^{0}$ = {A: A$\leftarrow$W $\in$ D' \D} (inserciones potenciales explícitas)

$pos_{D,D'}^{n+1}$ = {Aθ: A$\leftarrow$W $\in$ D, B ocurre positivamente en W, C $\in$ $pos_{D,D'}^{n}$ y θ = mgu(B,C)}

$\cup$

{Aθ: A$\leftarrow$W $\in$ D, B ocurre negativamente en W, C $\in$ $neg_{D,D'}^{n}$ y θ = mgu(B,C)}

(inserciones potenciales inducidas)

$neg_{D,D'}^{0}$ = $\varnothing$ (borrados potenciales explícitos)

$neg_{D,D'}^{n+1}$ = {Aθ: A$\leftarrow$W $\in$ D, B ocurre positivamente en W, C $\in$ $neg_{D,D'}^{n}$ y θ = mgu(B,C)}

$\cup$

{Aθ: A$\leftarrow$W $\in$ D, B ocurre negativamente en W, C $\in$ $pos_{D,D'}^{n}$ y θ = mgu(B,C)}

(borrados potenciales inducidos)

$pos_{D,D'} = \cup pos_{D,D'}^{n}$

$neg_{D,D'} = \cup neg_{D,D'}^{n}$

Según la anterior definición, la obtención de los conjuntos $pos_{D,D'}$y $neg_{D,D'}$podría significar el cálculo de infinitos conjuntos $pos_{D,D'}^{n}$ y $neg_{D,D'}^{n}$. En la práctica, el cálculo puede realizarse en un número finito de pasos si se utiliza alguna regla de parada del tipo siguiente: en lugar de calcular los conjuntos $pos_{D,D'}^{n}$ y $neg_{D,D'}^{n}$ calcular los conjuntos P^n y N^n definidos de la siguiente forma:

P^n (resp. N^n) = {A: A $\in$ $pos_{D,D'}^{n}$ (resp. $neg_{D,D'}^{n}$) y $\exists$/ A' $\in$ $pos_{D,D'}^{k}$ (resp. $neg_{D,D'}^{k}$) (0 $\leq$ k $\leq$ n) tal que A es una instancia de A'}.

La computación finaliza cuando, para un valor de n, los conjuntos P^n y N^n son vacíos.

El conjunto $pos_{D,D'}$se caracteriza porque cualquier inserción (resp. borrado) real es una instancia de alguno de sus elementos.

Teorema de Simplificación

Sea:

• (L,RI) el esquema de una base de datos deductiva, donde:

- L es un lenguaje de primer orden heterogéneo. Los conjuntos de símbolos de constante, función y predicado de L son finitos.
- RI = {W: $W=\forall x_1 \forall x_2 ... \forall x_n W'$} es el conjunto de restricciones de integridad, fórmulas cerradas de L en forma prenexa normal.

• D un estado de la base de datos: D = {A← B: A es un átomo, B es una fbf }.

• La semántica asumida es la de la compleción más el axioma de cierre de dominio. La teoría que representa el estado D es Tr = comp(D) ∪ {ACD}. Es importante destacar que comp(D) y ACD son las versiones con tipos de la compleción de D y del axioma de cierre de dominio [Llo87].

• T una transacción formada por dos conjuntos de cláusulas:

T_{del}: borrados de T

T_{ins}: inserciones de T.

La transacción no es contradictoria (no inserta y borra la misma cláusula) y además no modifica el lenguaje L.

• D'' y D' estados tales que:

$D'' = D \setminus T_{del}$

$D' = D'' \cup T_{ins}$.

• D y D' son estratificadas.

• W una restricción de integridad tal que D satisface W.

• Θ = {θ: θ es la restricción a $x_1,x_2,...,x_n$ del unificador más general entre un átomo que ocurre negativamente en W y un átomo de $pos_{D'',D'}$ o de un átomo que ocurre positivamente en W y un átomo de $neg_{D'',D'}$}.

• Ψ = {φ: φ es la restricción a $x_1,x_2,...,x_n$ del unificador más general entre un átomo que ocurre positivamente en W y un átomo de $pos_{D'',D}$ o de un átomo que ocurre negativamente en W y un átomo de $neg_{D'',D}$ }.

Se cumple:

a) D' satisface W sii D' satisface $\forall^{-}$ (W'ϕ) para todo $\phi \in \Psi \cup \Theta$.

b) Si D' $\cup$ $\{\leftarrow \overline{\forall}(W'\phi)\}$ tiene una refutación SLDNF para todo $\phi \in \Psi \cup \Theta$, entonces D' satisface W.

c) Si D' $\cup$ $\{\leftarrow \overline{\forall}$ (W'ϕ)$\}$ tiene un árbol SLDNF fallado finitamente para algún $\phi \in \Psi \cup \Theta$, entonces D' viola W.

Si $\Psi \cup \Theta$ es el conjunto vacío, W no se ve afectada por la transacción.

Si $\varepsilon \in \Psi \cup \Theta$ W no admite simplificación.

3.3.4.3 Método de Sadri y Kowalski [SK87]

Concepto de satisfacción

Este método utiliza el punto de vista de la consistencia, con Tr=comp(D) (semántica de la compleción):

D satisface W sii Tr $\cup$ {W} es consistente.

Conceptos previos

El método presentado en [SK87] aparece también en [KSS87]. El método se caracteriza por utilizar una extensión del procedimiento SLDNF que permite aplicar resolución a partir de las actualizaciones de la transacción.

Actualizaciones de la transacción

Sea T una transacción formada por dos conjuntos de cláusulas, un conjunto T_{ins} de inserciones y un conjunto T_{del} de borrados. El conjunto de actualizaciones de la transacción se define como sigue:

ACT= { A: $A \leftarrow L_1 \wedge L_2 \wedge ... \wedge L_n \in T_{ins}$}

$\cup$

{$\neg$A: $A \leftarrow \in T_{del}$ y existe un árbol SLDNF fallado finitamente para $D' \cup \{\leftarrow A\}$}

$\cup$

{$\neg A\theta$: $A \leftarrow L_1 \wedge L_2 \wedge ... \wedge L_n \in T_{del}$, θ es una respuesta computada para $D \cup \{\leftarrow L_1 \wedge L_2 \wedge ... \wedge L_n\}$ y existe un árbol SLDNF fallado finitamente para $D' \cup \{\leftarrow A\theta\}$}.

Procedimiento SLDNF* (SLDNF extendido)

Sea:

- S = D $\cup$ RI donde:
  - D es un conjunto de cláusulas de la forma:

    $A \leftarrow L_1 \wedge L_2 \wedge ... \wedge L_n$ $(n \geq 0)$
  - RI un conjunto de cláusulas de la forma:

    $\leftarrow L_1 \wedge L_2 \wedge .. \wedge L_n$ $(n>0)$

Li es un literal; si L_i es positivo (resp. negativo) le denominaremos condición positiva (resp. condición negativa).

• C_0 una cláusula de S o un átomo negado (¬A) tal que existe un árbol SLDNF* fallado finitamente para $S \cup \{\leftarrow A\}$.

• R una regla de computación segura.

Una **derivación** vía R para $S \cup \{C_0\}$ es una secuencia posiblemente infinita $C_0,C_1,C_2...$ tal que, C_i (i>0) es una cláusula, y para todo i≥0, C_{i+1} se obtiene a partir de C_i de la siguiente forma:

a) Si R selecciona de C_i un literal L que no es una condición negativa de C_i, entonces C_{i+1} es el resolvente sobre L de C_i y alguna cláusula de S.

b) Si R selecciona una condición negativa ¬A de C_i, entonces C_{i+1} es C_i eliminando el literal seleccionado, ¬A, si existe un árbol SLDNF* fallado finitamente para $S \cup \{\leftarrow A\}$.

c) Si C_i es ¬A y en S hay una cláusula B←¬A', C de forma que A y A' unifican a través del unificador más general θ, entonces C_{i+1} es $(B\leftarrow C)\theta$.

El SLDNF* es **correcto** para consistencia:

"Si existe una refutación SLDNF* para $S\cup\{C_0\}$ entonces comp(D)∪RI es inconsistente"

Teorema de simplificación

Sea:

• (L,RI) el esquema de una base de datos deductiva, donde:

- L es un lenguaje de primer orden y
- RI es el conjunto de restricciones de integridad, fórmulas cerradas de L.

• $\{\leftarrow inc_W: inc_W\leftarrow\neg W, W\in RI\}$ el conjunto de restricciones de integridad en forma negada (las cláusulas resultantes de aplicar el algoritmo Lloyd y Topor [LT84] a $\{inc_W\leftarrow\neg W: W\in RI\}$ deben ser de rango restringido y formar parte de cualquier estado de la base de datos).

• D un estado de la base de datos:

$D= \{A\leftarrow L_1\wedge L_2\wedge...\wedge L_n$: A es un átomo, L_i es un literal y $n \geq 0\}$.

• La semántica asumida es la de la compleción.

• T una transacción formada por dos conjuntos de cláusulas, el conjunto T_{ins} de inserciones y un conjunto T_{del} de borrados.

• D' el estado resultante de aplicar a D la transacción T:

$D'=(D\cup T_{ins})\setminus T_{del}$.

• D y D' son estratificados y de rango restringido.

• W una restricción de integridad tal que D satisface W.

Se cumple:

"Si existe una refutación SLDNF* para D' $\cup$ RI $\cup$ $\{C_0\}$ para alguna actualización C_0 de la transacción entonces D' viola RI".

La comprobación de la integridad utilizando el procedimiento SLDNF* tal como ha sido presentado anteriormente no tiene en cuenta los borrados inducidos por inserciones o borrados. Para resolver este problema se proponen dos soluciones [SK87]:

a) uso de metarreglas que, intercaladas entre los pasos de derivación, calculan los borrados inducidos en los casos en que estos pueden aparecer. Estas metarreglas son:

R1: borrado D',D,F) $\leftarrow$ $\neg$ P $\wedge$
en(F$\Leftarrow$B,D) $\wedge$
depende(B,P) $\wedge$
demo(D,F) $\wedge$
$\neg$ demo(D',F)

para el cálculo de los borrados inducidos por borrados.

R2: borrado(D',D,F) $\leftarrow$ P $\wedge$
en(F$\Leftarrow$B,D) $\wedge$
depende(B,) $\wedge$
demo(D,F) $\wedge$
$\neg$ demo(D',F)

para el cálculo de los borrados inducidos por inserciones.

En estas metarreglas, *en(C, D)* es cierto si C es una cláusula de D, *depende(B,P)* es cierto si en la conjunción de literales B aparece P y *demo(D,G)* es cierto si la conjunción de literales G es derivable de D utilizando el SLDNF*.

b) Uso de un metaprograma que integre el procedimiento SLDNF* y las metarreglas para el cálculo de los borrados inducidos:

R1:
inconsistente(S,[])$\leftarrow$

R2:

inconsistente(S,C)← seleccionar_literal(L,C,A) ∧
en(H,S) ∧
resolver(H,C,L,A,R) ∧
inconsistente(S,R)

R3:

inconsistente(S,C)← seleccionar_literal(noP,C,Condicion) ∧
¬ inconsistente(S,⇐P) ∧
eliminar_literal(C,noP,Condicion,C') ∧
inconsistente(S,C')

R4:

inconsistente(D'∪RI, no P⇐ C)←
seleccionar_literal(noP,no(P⇐C),Concl) ∧
en(F⇐B,D') ∧
depende(P,B) ∧
demo(D,F) ∧
¬ demo(D',F) ∧
inconsistente(D'∪RI, no(F⇐C))

R5:

inconsistente(D'∪RI, P⇐C)←
seleccionar_literal(P,P⇐C,Concl) ∧
en(F⇐B, D') ∧
depende(noP,B) ∧
demo(D, F) ∧
¬ demo(D',F) ∧
inconsistente(D'∪RI,no(F⇐C))

En estas metarreglas *seleccionar_literal(L,C,A)* es cierto si se selecciona el literal L en A (condición o conclusión) de la cláusula C, *resolver(H,C,L,A,R)* es cierto si R es el resolvente de C y H sobre el literal L en A (condición o conclusión) de C, *eliminar_literal(C,L,A,C')* es cierto si C' es la cláusula resultante de eliminar L en A (condición o conclusión) de C.

Las reglas R1, R2 y R3 junto con las definiciones subsidiarias necesarias, formalizan el procedimiento SLDNF*. Las reglas R5 y R6 formalizan las reglas para el cálculo de los borrados inducidos.

3.3.4.4 Método de Bry, Decker y Manthey [BDM88]

Concepto de satisfacción

Este método utiliza el punto de vista de la consistencia con Tr=comp(μ_D)$\cup${ACD} (semántica del punto fijo iterado [ABW88]):

D satisface W sii Tr $\cup$ {W} es consistente.

Conceptos previos

Este método sólo considera actualizaciones simples de hechos: inserciones de hechos no explícitos en D y borrados de hechos explícitos en D. En [BD88] esta propuesta se extiende considerando actualizaciones más generales: actualizaciones de reglas, actualizaciones múltiples o condicionales (actualizaciones de varios hechos) y transacciones.

Sea L un literal y U un literal base que representa una actualización (un literal positivo representa la inserción de un hecho y un literal negativo el borrado de un hecho):

D' = D $\cup$ {U: U es positivo} o bien

D' = D \ {¬U: U es negativo}.

El método define los siguientes conceptos:

Actualizaciones potenciales

Un átomo A (resp. ¬), no necesariamente base, **depende directamente de un literal L** sii:

- existe una regla deductiva, A'←B tal que B contiene un literal L' unificable con L (resp. con el complemento de L) y
- A = A'θ, donde θ = mgu(L,L') (resp. mgu(complemento de L,L')).

A **depende** de L sii A depende directamente de L o de un literal que depende de L. Todo literal que depende de U es una actualización potencial inducida por U.

El concepto de actualización potencial definido de esta forma coincide con el de [LST87], aunque en [LST87] se diferencian dos conjuntos de átomos *pos* (inserciones potenciales) y *neg* (borrados potenciales) en lugar del conjunto único de literales (actualizaciones potenciales) de este método.

Actualizaciones Reales

Un átomo base A (resp. ¬A) **es inducido directamente por L** sobre D' sii:

- existe una regla deductiva, A'←B tal que B contiene un literal L' unificable con L (resp. con el complemento de L) con unificador más general θ,
- A=(A'θ)φ, donde φ es una respuesta obtenida al evaluar (B\L')θ en D', (B\L' representa B sin L' o cierto si B=L') (semántica del modelo μ_D), y
-A (resp. ¬A) se evalúa a falso (resp. cierto) en D (resp. en D').

Un literal es **inducido** por L sobre D' sii es directamente inducido por L sobre D' o por un literal inducido por L sobre D'. Todo literal inducido por U sobre D' es una actualización real inducida por U.

Al igual que en el método de Lloyd, Sonnenberg y Topor, toda actualización real es una instancia de una actualización potencial.

Teorema de simplificación

Sea:

• (L,RI) el esquema de una base de datos deductiva, donde:
- L es un lenguaje de primer orden sin símbolos de función y
- RI es el conjunto de restricciones de integridad, fórmulas cerradas de L de cuantificadores restringidos, en forma normalizada.

En una fórmula de cuantificadores restringidos, las subfórmulas cuantificadas tienen la forma: $\exists x_1\exists x_2...\exists x_n(A_1\wedge...\wedge A_m\wedge Q)$ y $\forall x_1\forall x_2...\forall x_n(\neg A_1\vee...\vee\neg A_m\vee Q)$, donde cada x_i $(1\leq i\leq n)$ aparece en algún A_j $(1\leq i\leq m)$ y Q es una fórmula bien formada de cuantificadores restringidos.

La forma normalizada utilizada en el método se obtiene:

- reduciendo al máximo el ámbito de los cuantificadores,
- expresando las implicaciones y equivalencias con las conectivas lógicas ∧,∨,¬,
- llevando las negaciones sobre los átomos y
- distribuyendo las disyunciones respecto a las conjunciones.

• D un estado de la base de datos:

$D = \{A: A \text{ es un átomo base}\} \cup \{A\leftarrow L_1\wedge L_2\wedge...\wedge L_n: L_i \text{ es un literal, } n>o\}.$

• La semántica asumida es la del punto fijo iterado.

• U un literal base que representa la actualización (inserción o borrado de un hecho):

• D' el estado resultante de aplicar a D la actualización U:

D' = D ∪ { U: U es positivo} o bien

D' = D \ { ¬U: U es negativo}.

• D y D' son estratificados y de rango restringido.

• W una restricción de integridad tal que D satisface W.

• *delta* un metapredicado tal que para todo literal totalmente instanciado L, delta(U,L) es cierto sii L es cierto en D' y falso en D (semántica del modelo estándar μ_D).

• *new* un meta-predicado tal que new(U,F) es cierto sii F es cierto en D'.

• RA={$\forall^{-}$ (delta(U,Lθ)→new(U,W_S))} el conjunto de restricciones auxiliares asociadas a W que se obtiene aplicando a la restricción el método de simplificación de Nicolas a partir del conjunto de actualizaciones potenciales inducidas por U donde:

- L es una actualización potencial inducida por U, o bien U.
- θ es la restricción del unificador más general entre un literal L' de W y el complementario de L, a las variables de L' cuantificadas universalmente no precedidas de un cuantificador existencial y a las variables de L.
- W_S es la instancia simplificada de W obtenida a partir de Wθ aplicando las siguientes reglas de simplificación:
  i) eliminando los cuantificadores sobre las variables instanciadas por θ, y
  ii) reemplazando cada literal de Wθ unificable con el complementario de Lθ por falso y aplicando las reglas de absorción.

Se cumple :

D' satisface W sii las restricciones auxiliares asociadas a W se satisfacen en D'.

3.3.4.5 Método de Das and Williams [DW89a]

Concepto de satisfacción

Este método utiliza el punto de vista de la demostración con Tr = comp(D):

D satisface W sii Tr $\models$ W.

Conceptos previos

La comprobación de la integridad en D' consiste en buscar un *camino* desde una *actualización* de la transacción hasta un *átomo de inconsistencia*.

Actualizaciones de la transacción

Si A es un átomo base y H←B es una regla deductiva, el conjunto de actualizaciones de la transacción T se define:

$$ACT = \{H\theta: H \leftarrow B \in T_{ins} \text{ y } \theta \text{ es una respuesta computada SLDNF para } D' \cup \{\leftarrow B\}\} \cup \{\neg H: H \leftarrow B \in T_{del}\}$$

.

Camino

Si D es un estado de la base de datos, un camino se define como una secuencia de literales:

$$L_0 \text{--} R_1 \text{-->} L_1 \text{--} R_2 \text{-->} \ldots \text{--} R_n \text{-->} L_n$$

donde L_0 es el origen del camino, L_n el destino, n su longitud y $R_1, R_2, \ldots, R_n$ son cláusulas de D utilizadas para construir el camino de L_0 a L_n. Si L_0 es positivo, entonces es base y debe existir una refutación SLDNF para $D \cup \{\leftarrow L_0\}$.

L_{i+1} se define a partir de L_i de la forma:

1.- Si:

- L_i es positivo,
- L_i unifica con un literal positivo del cuerpo de la cláusula R_i:H←B con unificador más general α,
- G' es el resolvente de ←B y L_i y
- θ es una respuesta computada SLDNF para $D \cup \{\leftarrow G'\}$.

entonces, L_{i+1} es $H\alpha\theta$.

2.- Si:

- L_i es positivo,

- L_i unifica con el complementario de un literal negativo del cuerpo de la cláusula R_i:H←B con unificador más general α y
- $\neg A\alpha$ no es una instancia de algún L_i ($0 \leq j \leq i$).

entonces, L_{i+1} es $\neg H\alpha$.

3.-Si:

- L_i es negativo,
- L_i unifica con un literal negativo del cuerpo de la cláusula R_i:H←B con unificador más general α y
- θ es una respuesta computada SLDNF para $D \cup \{G\}$ donde $G=\leftarrow B\alpha$

entonces, L_{i+1} es $H\alpha\theta$.

4.-Si:

- L_i es negativo,
- L_i unifica con el complementario de un literal L que ocurre en el cuerpo de una cláusula R_i:H←B con unificador más general α y
- $\neg H\alpha$ no es una instancia de algún L_j ($0 \leq j \leq i$)

entonces, Li+1 es $\neg H\alpha$.

Un camino que finaliza en algún átomo de inconsistencia (para cada restricción W existe un átomo de inconsistencia inc_w definido por la regla $inc_W \leftarrow \neg W$) se denomina camino de éxito, en caso contrario se denomina camino de fallo.

Teorema de simplificación

Sea:

• (L,RI) el esquema de una base de datos deductiva, donde:

-L es un lenguaje de primer orden y

-RI es el conjunto de restricciones de integridad, fórmulas cerradas de L.

• $\{\leftarrow inc_W: inc_W \leftarrow \neg W, W \in RI\}$ el conjunto de restricciones de integridad en forma negada (las cláusulas resultantes de aplicar el algoritmo de Lloyd y Topor [LT84] a $\{inc_W \leftarrow \neg W: W \in RI\}$ deben ser permitidas y formar parte de cualquier estado de la base de datos).

• D un estado de la base de datos:

$D= \{A \leftarrow L_1 \wedge L_2 \wedge ... \wedge L_n$: A es un átomo, L_i es un literal y $n \geq 0\}$.

• La semántica asumida es la de la compleción.

• T una transacción formada por dos conjuntos de cláusulas, un conjunto T_{ins} de inserciones y un conjunto T_{del} de borrados. Una transacción es válida si no es contradictoria (no exige la inserción y el borrado de la misma cláusula) y además el conjunto de actualizaciones ACT puede ser computado.

• D' es el estado resultante de aplicar a D la transacción T: $D'=(D \cup T_{ins}) \setminus T_{del}$.
• D y D' son estratificados y permitidos.

• W una restricción de integridad tal que D satisface W.

Se cumple:

a) si existe un camino de éxito en D' con origen en alguna de las actualizaciones de T, entonces D' viola RI.

b) si no existe un camino de éxito en D' para ninguna de las actualizaciones de T como origen, entonces D' satisface RI.

3.3.4.6 Método de Olivé [Oli91]

Concepto de satisfacción

Este método utiliza el punto de vista de la demostración con Tr = comp(D):

D satisface W sii Tr $\models$ W.

Conceptos previos

El método se caracteriza por :

- extender el lenguaje de la base de datos con dos tipos de predicados :

  i) predicados de *transición*: permiten simular la base de datos actualizada y
  ii) predicados de *eventos internos* (inserción y borrado): representan las actualizaciones generadas por la transacción.

- ampliar la base de datos con las reglas deductivas que definen dichos predicados.

- la comprobación directa de las restricciones de integridad utilizando dichas reglas.

- el tratamiento uniforme de restricciones de integridad estáticas y de transición.

Predicados de eventos internos: actualizaciones generadas por T

Para cada predicado P, introducimos un predicado de transición P', que representa el predicado P en la base de datos actualizada D' y dos predicados de eventos internos:

ιP: predicado de evento interno de inserción
δP: predicado de evento interno de borrado.

Estos predicados representan las actualizaciones (inserciones y borrados) explícitas o inducidas generadas por una transacción T sobre el predicado P. De la definición de actualización inducida por una transacción se puede afirmar:

$\forall x_1 \forall x_2 ... \forall x_n \, (\iota P(x_1,x_2,...,x_n) \leftrightarrow P'(x_1,x_2,...,x_n) \wedge \neg P(x_1,x_2,...,x_n))$
$\forall x_1 \forall x_2 ... \forall x_n \, (\delta P(x_1,x_2,...,x_n) \leftrightarrow P(x_1,x_2,...,x_n) \wedge \neg P'(x_1,x_2,...,x_n))$

Si P es un predicado base, ιP y δP serán predicados base cuyos hechos serán determinados directamente por las actualizaciones explícitas de la transacción T sobre P.

Si P es un predicado derivado, ιP y δP serán predicados derivados definidos por reglas deductivas que nos permitirán obtener las actualizaciones inducidas por la transacción T sobre P.

Reglas de eventos internos

Para poder determinar las reglas deductivas que definen los predicados ιP y δP para predicados derivados de la base de datos, vamos a determinar previamente las reglas que definen el predicado P'.

Como P' representa al predicado P en la base de datos actualizada, se cumplirá:

$\forall x_1 \forall x_2 ... \forall x_n (P'(x_1,x_2,...,x_n) \leftrightarrow (P(x_1,x_2,...,x_n) \wedge \neg \delta P(x_1,x_2,...,x_n)) \vee \iota P(x_1,x_2,...,x_n))$
$\forall x_1 \forall x_2 ... \forall x_n (\neg P'(x_1,x_2,...,x_n) \leftrightarrow (\neg P(x_1,x_2,...,x_n) \wedge \neg \iota P(x_1,x_2,...,x_n)) \vee \delta P(x_1,x_2,...,x_n))$

Si el predicado derivado P viene definido por m reglas deductivas en D:

$P \leftarrow P_i \quad 1 \le i \le m \quad$ siendo $\quad P_i \leftrightarrow L_1 \wedge L_2 \wedge ... \wedge L_n$

entonces P' estará definido por m reglas deductivas de la forma:

$P' \leftarrow P'_i \quad 1 \le i \le m \quad$ siendo $\quad P'_i \leftrightarrow L'_1 \wedge L'_2 \wedge ... \wedge L'_n$

reemplazando cada L'_j de la siguiente forma:

si $L_j=Q_j(x_1,x_2,...,x_n)$ (literal positivo), entonces:
$L'_j=(Q_j(x_1,x_2,...,x_n)\wedge\neg\delta Q_j(x_1,x_2,...,x_n))\vee\iota Q_j(x_1,x_2,...,x_n)$

si $L_j=\neg Q_j(x_1,x_2,...,x_n)$ (literal negativo), entonces:
$L'_j=(\neg Q_j(x_1,x_2,...,x_n)\wedge\neg\iota Q_j(x_1,x_2,...,x_n))\vee\delta Q_j(x_1,x_2,...,x_n)$

con lo que se obtienen m reglas que definen P' en términos (conjunción de disyunciones) de predicados de base de datos y predicados de eventos internos. Para obtener un conjunto equivalente de cláusulas basta distribuir las conjunciones sobre las disyunciones.

Una vez obtenidas las reglas que definen P', las reglas que definen ιP y δP serán:

$\iota P(x_1,x_2,...,x_n)\leftarrow P'(x_1,x_2,...,x_n)\wedge\neg P(x_1,x_2,...,x_n)$
$\delta P(x_1,x_2,...,x_n)\leftarrow P(x_1,x_2,...,x_n)\wedge\neg P'(x_1,x_2,...,x_n)$

Las reglas de eventos internos pueden simplificarse después de aplicarles algunas transformaciones como se indica en [Oli91].

Teorema de simplificación

Sea:

- (L,RI) el esquema de una base de datos deductiva donde:
  - -L es un lenguaje de primer orden, con conjuntos disjuntos de símbolos de predicados base y predicados derivados, y
  - -RI es el conjunto de restricciones de integridad, fórmulas cerradas de L.

- $\{\leftarrow inc_W(x_1,x_2,...,x_n)$: $inc_W(x_1,x_2,...,x_n)\leftarrow\neg W'$, $\forall x_1\forall x_2...\forall x_n W'\in RI\}$ el conjunto de restricciones de integridad en forma negada (las cláusulas resultantes de aplicar el algoritmo de Lloyd y Topor [LT84] a $\{inc_W(x_1,x_2,...,x_n)\leftarrow\neg W'$: $\forall x_1\forall x_2...\forall x_n W'\in RI\}$ deben ser permitidas y formar parte de cualquier estado de la base de datos).

- D un estado de la base de datos:
  $D=\{A\leftarrow L_1\wedge L_2\wedge...\wedge L_n$: A es un átomo, L_i es un literal y $n\geq 0\}$.

- La semántica asumida es la de la compleción.

- T una transacción formada por un conjunto T_{ins} de inserciones y un conjunto T_{del} de borrados de hechos y reglas.

- D' es el estado resultante de aplicar a D la transacción T: $D'=(D\cup T_{ins})\backslash T_{del.}$

- D y D' son permitidos.

• W una restricción de integridad tal que D satisface W.

Si A(D) es la base de datos resultante de añadir a D las reglas de transición y de eventos internos para cada predicado derivado y para cada predicado de inconsistencia de D, entonces diremos que:

a) D' viola la restricción W si existe una refutación SLDNF para $A(D) \cup T \cup \{\leftarrow \iota inc_W(x_1,x_2,...,x_n)\}$.

b) D' satisface la restricción W si existe un árbol SLDNF fallado finitamente para $A(D) \cup T \cup \{\leftarrow \iota inc_W(x_1,x_2,...,x_n)\}$.

La comprobación de la integridad se puede optimizar (reduciendo el árbol de búsqueda) si la regla de computación del SLDNF selecciona en primer lugar los literales de eventos internos.

3.3.5 Análisis de los métodos

En este apartado se pretende hacer un análisis de los métodos presentados referente a: concepto de satisfacción, requisitos sintácticos de la base de datos, representación y requisitos sintácticos de las restricciones, estrategia seguida por el método para simplificar la comprobación de la integridad y resultados de corrección y completitud.

En la tabla 2.1 se presenta un resumen de las características de cada método.

3.3.5.1 Concepto de satisfacción

La primera definición del concepto de satisfacción que aparece en la literatura es la del punto de vista de la demostración, habiendo sido utilizada ésta en la mayoría de los métodos. En [SK87] se introduce por primera vez una definición alternativa de satisfacción, la correspondiente al punto de vista de la consistencia.

Los dos conceptos de satisfacción pueden enunciarse de la forma siguiente:

- punto de vista de la demostración: D satisface W sii $Tr \models W$
- punto de vista de la consistencia: D satisface W sii $Tr \not\models \neg W$

siendo Tr la teoría que representa el estado D en la semántica asumida.

Ambos puntos de vista pueden presentar problemas, ya que en general determinar que una fórmula cerrada no es consecuencia lógica de una teoría Tr es un problema indecidible en lógica de primer orden. Así:

- en el punto de vista de la demostración, aquellas restricciones de integridad que son violadas de forma que Tr $|\!=\!/$ W pero Tr $|\!=\!/$ ¬W no pueden ser detectadas.

- en el punto de vista de la consistencia, aquellas restricciones de integridad que se satisfacen de forma que Tr $|\!=\!/$ ¬W pero Tr $|\!=\!/$ W no pueden ser detectadas.

Esto significa que no existe un método para comprobar la integridad que sea correcto y completo (2.3.3), es decir, tal que para todo estado D y para toda restricción W sea capaz de decidir si D satisface W o si D viola W. Este problema de incompletitud se puede resolver imponiendo a la base de datos y a las restricciones de integridad requisitos sintácticos que aseguren que, para todo estado D y para toda restricción W, se cumpla Tr $\models$ W o Tr $\models$ ¬W.

A pesar de las limitaciones de ambas definiciones, el punto de vista de la consistencia parece, en general, más adecuado. Desde un punto de vista filosófico, esta definición del concepto de satisfacción proporciona más flexibilidad para el cambio, ya que la condición de satisfacción es menos restrictiva. Desde un punto de vista práctico, esta definición asegura que cualquier violación de la integridad es detectada, ya que el conjunto de restricciones W de las cuales no se puede decir computacionalmente nada porque Tr$|\!=\!/$ W y Tr$|\!=\!/$ ¬W, son restricciones que se satisfacen en D.

Los método presentados se clasifican de acuerdo al concepto de satisfacción que utilizan de la forma siguiente:

Métodos en el punto de vista de la demostración: [LST87], [DW89a] y [Oli91].

Métodos en el punto de vista de la consistencia: [SK87] y [BDM88] .

3.3.5.2 Tipo de base de datos

El método de Lloyd, Sonnenberg y Topor [LST87] trabaja con bases de datos generales, es decir, con reglas de la forma A←W, donde A es un átomo y W una fórmula bien formada cualquiera.

Los restantes métodos se definen para bases de datos normales con reglas de la forma: $A \leftarrow L_1 \wedge L_2 \wedge ... \wedge L_n$, donde A es un átomo y L_i un literal. Este requisito no quita generalidad a un método ya que cualquier base de datos general puede transformarse en una base de datos normal siguiendo el algoritmo de Lloyd y Topor [LT84] [Llo87]; sin embargo, en este caso

hay que tener en cuenta que al aplicar el algoritmo se pueden perder ciertas propiedades sintácticas de la base de datos.

Ejemplo 3.3

Sea $W=\exists y \forall x(p(x) \rightarrow q(x,y))$ una restricción de integridad de rango restringido cuya forma negada es $\leftarrow inc_w$ donde inc_w se define por la regla $inc_w \leftarrow \neg W$. Para incluir esta regla en la base de datos hay que transformarla en un conjunto de cláusulas normales aplicando el algoritmo de Lloyd y Topor [LT84]. Estas transformaciones son:

$inc_w \leftarrow \forall y \exists x(p(x) \wedge \neg q(x,y))$
$inc_w \leftarrow \neg \exists y(\neg \exists x(p(x) \wedge \neg q(x,y)))$
$inc_w \leftarrow \neg aux_1$
$aux_1 \leftarrow \neg \exists x(p(x) \wedge \neg q(x,y))$
$aux_1 \leftarrow \neg aux_2(y)$
$aux_2(y) \leftarrow p(x) \wedge \neg q(x,y)$

siendo el conjunto de cláusulas resultante:

$inc_w \leftarrow \neg aux_1$
$aux_1 \leftarrow \neg aux_2(y)$
$aux_2(y) \leftarrow p(x) \wedge \neg q(x,y)$

Como puede observarse la propiedad de rango restringido de la regla $inc_w \leftarrow \forall y \exists x(p(x) \wedge \neg q(x,y))$ se ha perdido en el conjunto de cláusulas resultantes de la transformación.

3.3.5.3 Requisitos sintácticos

Los métodos de comprobación simplificada de la integridad exigen a la base de datos y a la restricción propiedades sintácticas que aseguren:

- la consistencia de Tr (la teoría que representa el estado D en la semántica asumida) y
- la corrección y la completitud del método.

Consistencia de la base de datos

Los métodos exigen que la base de datos D sea estratificada, asegurando de esta forma que:

- si se asume la semántica de la compleción, comp(D) es consistente

- si se asume la semántica del modelo minimal, D tiene un único modelo minimal soportado, el modelo μ_D [ABW88].

Como se ha visto en el capítulo 1, existen requisitos más débiles que aseguran también la consistencia de la base de datos.

Corrección y Completitud del método.

Como se vió en 2.3.3 las propiedades de corrección y completitud de un método con una clara separación en dos fases, pueden definirse en términos de la corrección y la completitud de la Fase de Generación y de la Fase de Evaluación, a continuación vamos a estudiar los requisitos sintácticos necesarios (no suficientes) para asegurar estas propiedades en cada una de las fases.

• Fase de Generación

Como se vió en 2.3.2, la Fase de Generación de un método consiste en la obtención de un conjunto de instancias (simplificadas) de las restricciones de integridad relevantes respecto a la transacción. Estas instancias se obtienen a partir de las actualizaciones inducidas por la transacción (conjuntos $INS_{D,D'}$ y $DEL_{D,D'}$) o a partir de conjuntos de literales que "capturen" la diferencia entre los estados D y D' [LST87], [BDM88].

La corrección y la completitud de la Fase de Generación dependen del concepto de satisfacción y de la semántica asumidos por el método. Vamos a estudiar distintos casos en base a unos ejemplos:

a) Si el concepto de satisfacción utilizado es el correspondiente al punto de vista de la consistencia, la Fase de Generación no es correcta ni completa en el caso general (ejemplo 3.4); es decir no es posible simplificar la comprobación de la integridad sin exigir requisitos a la base de datos y a la restricción.

Ejemplo 3.4

Asumimos la semántica de la compleción.

$D = \{p(a)\leftarrow q(x)\wedge r(x),\ r(x)\leftarrow r(x),\ q(a)\leftarrow\}$

$Tr = \{\forall x(p(x)\leftrightarrow x=a\wedge\exists y(q(y)\wedge r(y))),\ \forall x(r(x)\leftrightarrow r(x)),\ \forall x(q(x)\leftrightarrow x=a)\}$

$W=\forall x\neg p(x)$ (D satisface W).

$T_{ins}=\{p(b)\leftarrow q(x)\wedge\neg r(x)\}$.

$D' = \{p(a) \leftarrow q(x) \wedge r(x),$
$p(b) \leftarrow q(x) \wedge \neg r(x),$
$r(x) \leftarrow r(x),$
$q(a) \leftarrow \}$

$Tr' = \{\forall x(p(x) \leftrightarrow (x=a \wedge \exists y(q(y) \wedge r(y)))$
$\vee (x=b \wedge \exists y(q(y) \wedge \neg r(y)))),$
$\forall x(r(x) \leftrightarrow r(x)),$
$\forall x(q(x) \leftrightarrow x=a)\}$

Supongamos un método que utiliza en la Fase de Generación los conjuntos de actualizaciones potenciales $pos_{D,D'}=\{p(b)\}$ y $neg_{D,D'}=\varnothing$ que "capturan" respectivamente los conjuntos $INS_{D,D'}$ y $DEL_{D,D'}$. Utilizando estos conjuntos se obtiene la instancia de W, $W_s=\neg p(b)$. Es inmediato comprobar que D' satisface W_s, siendo $M=\{q(a), r(a), p(a)\}$ un modelo para $Tr' \cup \{W_s\}$ y sin embargo D' viola W no existiendo ningún modelo para $Tr' \cup \{W\}$, es decir la simplificación realizada en la Fase de Generación no es correcta.

Para estudiar el problema de una forma general, vamos a considerar que las restricciones se representan en forma negada. Dada una restricción $W=\forall x_1 \forall x_2 ... \forall x_n W'$ en forma prenexa normal, su forma negada es $\leftarrow inc_w(x_1,x_2,...,x_n)$ donde el predicado inc_w se define por la regla $inc_w(x_1,x_2,...,x_n) \leftarrow \neg W'$ que forma parte de cualquier estado de la base de datos. Con esta representación el concepto de violación en el punto de vista de la consistencia puede formularse de la forma:

D' viola W sii $Tr' \models \exists x_1 \exists x_2 ... \exists x_n inc_w(x_1,x_2,...,x_n)$

y asumiendo que D satisface W, es decir $Tr \not\models \exists x_1 \exists x_2 ... \exists x_n inc_w(x_1,x_2,...,x_n)$, la Fase de Generación puede enunciarse como:

D' viola W sii $Tr' \models \exists^{-} inc_w(t_1,t_2,...,t_n)$ (para algún $inc_w(t_1,t_2,...,t_n) \in pos_{D'D'}$)

donde $pos_{D,D'}$ es un conjunto de átomos que "captura" las inserciones inducidas por T.

AQUÍ

Es evidente que la Fase de Generación es correcta (para violación), es decir si $Tr' \models \exists^{-} inc_w(t_1,t_2,...,t_n)$ $(inc_w(t_1,t_2,...,t_n) \in pos_{D,D'})$ entonces D' viola W, pero no es completa (para violación), es decir D' puede violar W ($Tr' \models \exists x_1 \exists x_2 ... \exists x_n inc_w(x_1,x_2,...,x_n)$) y sin embargo $Tr' \not\models \exists^{-} inc_w(t_1,t_2,...,t_n)$ para todo $inc_w(t_1,t_2,...,t_n)$ perteneciente a $pos_{D,D'}$. En el ejemplo 3.4, la forma negada de W es $\leftarrow inc_w(x)$ donde $inc_w(x) \leftarrow p(x)$, $pos_{D,D'}=\{p(b)\}$ y $Tr' \models \exists x p(x)$ sin embargo $Tr' \not\models p(b)$.

b) Si el concepto de satisfacción utilizado es el correspondiente al punto de vista de la demostración, existen ciertos requisitos sintácticos necesarios (no suficientes) para asegurar

que la Fase de Generación es completa, estos requisitos dependen de la semántica asumida por el método. Además de éstos pueden aparecer otros requisitos que dependan de la estrategia de simplificación propia del método. Vamos a diferenciar dos casos en función de la semántica asumida por el método:

b1) La teoría Tr que representa el estado D en la semántica asumida incluye el axioma de cierre de dominio [LST87], [BDM88]. En los ejemplos siguientes (ejemplos 3.5, 3.6, 3.7 y 3.8) suponemos un método de comprobación de la integridad hipotético, tal como ha sido descrito en el apartado 2.3.2, que utiliza en la Fase de Generación los conjuntos $INS_{D,D'}$ y $DEL_{D,D'}$.

Ejemplo 3.5

Asumimos la semántica de la compleción.

D = { p(x)←q(x),
q(a)←,
r(a)← }

Tr = { ∀x(p(x)↔q(x)),
∀x(q(x)↔x=a),
∀x(r(x)↔x=a),
∀x(x=a)}

RI={W_1=∀xp(x), W_2=¬r(b)}.
D satisface W_1 y W_2.
T_{ins}={r(b)}.

D' = { p(x)←q(x),
q(a)←,
r(a)←,
r(b)←}

Tr' = {∀x(p(x)↔q(x)),
∀x(q(x)↔x=a),
∀x(r(x)↔x=a∨x=b) ,
∀x(x=a∨x=b}

La Fase de Generación nos dice que:

- W_1 no es relevante respecto a T, y que por lo tanto D' satisface W_1, sin embargo D' viola W_1 (Tr'|=/ W_1).

- W_2 es relevante respecto a T, siendo la instancia simplificada W_{2s}=falso, D' viola W_{2s} y por lo tanto D viola W_2.

Es interesante observar que la Fase de Generación no es correcta para W_1 (fórmula dependiente del dominio) y en cambio es correcta para W_2 (fórmula independiente del dominio).

Ejemplo 3.6

Asumimos la semántica de la compleción.

D = {t(a)←¬p(a),
p(a)←¬q(x)∧ r(x),
q(a)←,
r(x)←r(x)}

Tr = {∀x(t(x)↔x=a∧¬p(a)),
∀x(p(x)↔x=a∧∃y(¬q(y)∧r(y))),
∀x(q(x)↔x=a),
∀x(r(x)↔r(x)),
∀x(¬s(x)),
∀x(x=a)}

W=∃xt(x).
D satisface W.

T_{ins}={s(b)}.

D' = {t(a)←¬p(a),
p(a)←¬q(x)∧r(x),
q(a)←,
r(x)←r(x),
s(b)←}

Tr' = {∀x(t(x)↔x=a∧¬p(a)),
∀x(p(x)↔x=a∧∃y(¬q(y)∧r(y))),
∀x(q(x)↔x=a),
∀x(r(x)↔r(x)),
∀x(s(x)↔x=b),
∀x(x=a∨x=b)}

La Fase de Generación nos dice que W no es relevante respecto a T, y que por lo tanto D' satisface W, sin embargo D' viola W (Tr' |=/ W). Esto significa que la Fase de Generación no es correcta.

Ejemplo 3.7

Asumimos la semántica del punto fijo iterado:

D = {p(x)←¬q(x),
q(a)← }

Tr = {∀x(q(x)↔x=a),
∀x(¬p(x)),
∀x(¬s(x)),
∀x(x=a)}

W=∀x¬p(x).
D satisface W.
T_{ins}={s(b)}.

D' = {p(x)←¬q(x),
q(a)←,
s(b)← }

Tr' = {∀x(q(x)↔x=a),
∀x(p(x)↔x=b),
∀x(s(x)↔x=b),
∀x(x=a∨x=b)}

La Fase de Generación nos dice que W no es relevante respecto a T, y que por lo tanto D' satisface W, sin embargo D' viola W (Tr' |=/ W). Esto significa que la Fase de Generación no es correcta.

Ejemplo 3.8

Asumimos la semántica del punto fijo iterado:

D = {p(x)←¬q(x)∧r(x),
q(a)←,
r(x)←r(x)}

Tr = {∀x(q(x)↔x=a),
∀x(¬p(x)),
∀x(¬r(x)),

$\forall x\ (\neg s(x)),$
$\forall x(x=a)\}$

$W=\forall x \neg p(x).$
D satisface W.
$T_{ins}=\{s(b)\}.$

$D' = \{p(x) \leftarrow \neg q(x) \wedge r(x),$
$q(a) \leftarrow,$
$r(x) \leftarrow r(x),$
$s(b) \leftarrow \}$

$Tr' = \{\forall x(q(x) \leftrightarrow x=a),$
$\forall x(s(x) \leftrightarrow x=b),$
$\forall x(\neg p(x)),$
$\forall x(\neg r(x)),$
$\forall x(x=a \vee x=b)\}$

D' satisface W.

En este ejemplo la Fase de Generación nos dice que W no es relevante respecto a T, por lo tanto D' satisface W. Esto significa que la Fase de Generación es correcta.

Del análisis de los ejemplos anteriores podemos deducir que las soluciones a este problema pueden ser:

- exigir a la base de datos y a la restricción propiedades sintácticas. Si se asume la semántica del punto fijo iterado la condición de rango restringido para D, D' y W es suficiente [BDM88]; en el ejemplo 3.7 D y D' no son independientes del dominio y la Fase de Generación no es correcta; en el ejemplo 3.8 D y D' son de rango restringido (independientes del dominio) y la Fase de Generación es correcta. Si se asume la semántica de la compleción estos requisitos pueden ser suficientes en algunos casos, por ejemplo si D y D' son relacionales o jerárquicas (ejemplo 3.5) pero no en el caso general (ejemplo 3.6). El problema de la independencia del dominio para bases de datos y requerimientos se estudia en [Top87], [TS88], [Dec88] y [Kif88].

- no permitir que la transacción cambie el lenguaje, es decir el axioma de cierre de dominio es fijo [LST87].

b2) La teoría Tr que representa el estado D en la semántica asumida no incluye el axioma de cierre de dominio [SK87], [DW89a] [Oli91], en este caso conjeturamos que no son necesarias restricciones sintácticas especiales para asegurar la corrección y completitud de la Fase de Generación.

• Fase de Evaluación

Los métodos que utilizan el SLDNF como procedimiento en la Fase de Evaluación, exigen a la base de datos y a la restricción propiedades sintácticas que aseguren que, en las computaciones SLDNF realizadas no aparece el problema del "tropiezo" (en algún punto de la derivación se obtiene un objetivo que sólo contiene literales negativos que no son base) [Llo87], [Dec89]. En el método de Lloyd, Sonnenberg y Topor [LST87], el uso de un lenguaje heterogéneo asegura que la "forma normal sin tipos" de una base de datos y un requerimiento es débilmente permitida [Llo87]. En los restantes métodos se exige a cada cláusula normal de la base de datos y a la restricción la propiedad de "rango restringido" (o alguna propiedad sintáctica que implique ésta).

Para que la Fase de Evaluación sea correcta y completa utilizando como procedimiento de evaluación el procedimiento SLDNF es necesario exigir a la base de datos y a las restricciones de integridad propiedades sintácticas que aseguren la completitud y la terminación del SLDNF (capítulo 1).

En primer lugar vamos a definir los conceptos de satisface$_{PE}$ y viola$_{PE}$ cuando PE es el procedimiento SLDNF. Sea D un estado de la base de datos y W una restricción de integridad cuya forma negada es $\leftarrow inc_w(x_1,...,x_n)$, donde el predicado inc_w se define por la regla $inc_w(x_1,x_2,...,x_n)\leftarrow\neg W'$ que forma parte de cualquier estado de la base de datos. Definimos los conceptos satisface$_{SLDNF}$ y viola$_{SLDNF}$ de la forma siguiente:

•D satisface$_{SLDNF}$ W sii existe un árbol SLDNF fallado finitamente para $D\cup\{\leftarrow inc_w(x_1,...,x_n)\}$

•D viola$_{SLDNF}$ W sii existe un refutación SLDNF para $D\cup\{\leftarrow inc_w(x_1,...,x_n)\}$.

Con las definiciones anteriores, los conceptos de corrección y completitud se definen:

a) en el punto de vista de la demostración:

• si existe un árbol fallado finitamente para $D\cup\{\leftarrow inc_w(x_1,...,x_n)\}$ entonces $comp(D)\models(\leftarrow inc_w(x_1,...,x_n))$ (correcto para satisfacción) (1).

• si existe una refutación SLDNF para $D\cup\{\leftarrow inc_w(x_1,...,x_n)\}$ entonces $comp(D)\not\models(\leftarrow inc_w(x_1,...,x_n))$ (correcto para violación) (2).

• si $comp(D)\models(\leftarrow inc_w(x_1,...,x_n))$ entonces existe un árbol fallado finitamente para $D\cup\{\leftarrow inc_w(x_1,...,x_n)\}$ (completo para satisfacción) (3).

• si $comp(D)\not\models(\leftarrow inc_w(x_1,...,x_n))$ entonces

existe una refutación SLDNF para $D \cup \{\leftarrow inc_w(x_1,...,x_n)\}$
(completo para violación) (4).

b) en el punto de vista de la consistencia:

• si existe un árbol fallado finitamente para $D \cup \{\leftarrow inc_w(x_1,...,x_n)\}$ entonces
$comp(D) \cup \{\leftarrow inc_w(x_1,...,x_n)\}$ es consistente
(correcto para satisfacción) (5).

• si existe una refutación SLDNF para $D \cup \{\leftarrow inc_w(x_1,...,x_n)\}$ entonces
$comp(D) \cup \{\leftarrow inc_w(x_1,...,x_n)\}$ es inconsistente
(correcto para violación) (6).

• si $comp(D) \cup \{\leftarrow inc_w(x_1,...,x_n)\}$ es consistente entonces
existe un árbol fallado finitamente para $D \cup \{\leftarrow inc_w(x_1,...,x_n)\}$
(completo para satisfacción) (7).

• si $comp(D) \cup \{\leftarrow inc_w(x_1,...,x_n)\}$ es inconsistente entonces
existe una refutación SLDNF para $D \cup \{\leftarrow inc_w(x_1,...,x_n)\}$
(completo para violación) (8).

Debido a que el procedimiento SLDNF es correcto respecto a la semántica de la compleción, la Fase de Evaluación es correcta para satisfacción y para violación en los dos conceptos de satisfacción (casos (1), (2), (5) y (6)).

Por otro lado la Fase de Evaluación en los casos (3) y (8) será completa si D y W cumplen propiedades sintácticas que aseguran la completitud del SLDNF. En los casos (4) y (7) será necesario exigir a D y W propiedades sintácticas que aseguren la terminación de las computaciones SLDNF (capítulo 1).

3.3.5.4 Estrategia del método

Analizando la estrategia seguida por los métodos, podemos clasificarlos por distintos criterios:

a) Existencia de una Fase de Generación potencial, es decir sin acceso a los hechos almacenados explícitamente

- métodos con fase de generación potencial: [LST87], [BDM88]

- métodos sin fase de generación potencial: [SK87], [DW89a], [Dec86], [Oli91]

Algunos métodos sin Fase de Generación potencial pueden simular esta fase utilizando algunos aspectos flexibles del método: en [SK87] utilizando una regla de computación que seleccione en primer lugar el literal de la cabeza, si éste existe; en [Oli91] utilizando una regla de computación que seleccione en primer lugar los literales sobre predicados de inserción o borrado. En ambos casos habrá que asegurar que la regla de computación utilizada es segura.

b) Intercalación de la Fase de Generación y la Fase de Evaluación: en todos los métodos es posible realizar esta intercalación.

c) Existencia de una etapa compilada independiente de la transacción:

- métodos con etapa de compilación: [Dec86], [Oli91]

- métodos sin etapa de compilación: [LST87], [BDM88], [SK87], [DW89a]

El análisis de las características de los métodos nos permite realizar las siguientes consideraciones:

i) la existencia de una Fase de Generación potencial es interesante porque permite, sin acceder a los hechos, eliminar del proceso de comprobación:
- restricciones no relevantes para la transacción
- actualizaciones no relevantes para la integridad.

ii) cuando la Fase de Generación es potencial, la instanciación de las restricciones a partir de actualizaciones potenciales, genera un conjunto de restricciones menos instanciadas que en el caso de trabajar con actualizaciones reales, y por lo tanto, en general más costosas de comprobar.

iii) aunque en la Fase de Evaluación, la comprobación de las restricciones simplificadas obtenidas en la Fase de Generación potencial se realice sólo para aquellas instancias correspondientes a instancias de las actualizaciones potenciales que coinciden con una actualización real, el proceso puede ser muy costoso ya que los métodos no hacen uso de la información disponible en la fase de generación (referente a caminos de derivación) para la obtención de estas actualizaciones reales.

iv) los métodos con Fase de Generación potencial pueden optimizarse intercalando Fase de Generación y Fase de Evaluación.

i) en los métodos sin Fase de Generación potencial, la selección como origen de la derivación de actualizaciones que inducen a su vez actualizaciones no relevantes para la integridad, puede elevar el coste de la comprobación.