Generación de reglas restauradoras de la consistencia en esquemas relacionales con vistas

Laura Mota Herranz

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Memoria para optar al Grado de Doctora en Informática

Dirigida por la Dra. **Matilde Celma Giménez**

9 de septiembre de 2002

A mi abuela Leonor a quien no conocí.
A mi abuelo Benigno de quien recuerdo una sonrisa.
A mi abuelo Fausto que me cepillaba el pelo.
A mi abuela Elisa de quien no pude despedirme.
A mis padres Ramón y Elisa.
A Vicente.

"Y al conde le plugo mucho este consejo que
Patronio le dio, y fízolo así y fallóse ende bien."

"El Conde Lucanor", Don Juan Manuel.

Gracias Mati.

Resumen

En un sistema de base de datos, el cambio en el mundo real se modela mediante la ejecución de transacciones de usuario que modifican un estado generando otro que representa la nueva situación. Este nuevo estado puede violar las restricciones de integridad del esquema, restricciones que representan las propiedades de ese mundo. La reacción más frecuente de los sistemas de bases de datos ante la violación de la integridad consiste en rechazar la transacción que la ha provocado, devolviendo la base de datos al estado anterior a su ejecución. Esta solución tan simple es, sin embargo, usualmente poco satisfactoria para sistemas reales. Una alternativa a este comportamiento consiste en que el sistema modifique el estado inconsistente de forma que se repare la violación provocada por la transacción de usuario respetando los cambios propuestos por ésta. Se dice entonces que el sistema ha restaurado la consistencia de la base de datos.

En esta tesis se ha propuesto un método para la restauración de la consistencia en bases de datos relacionales con vistas (o bases de datos deductivas) que utiliza el lenguaje de reglas de un sistema activo. Así, a partir del conjunto de restricciones de integridad y del conjunto de reglas deductivas del esquema, el método genera un conjunto de reglas de actividad que restaura la consistencia de la base de datos cuando, como consecuencia de la ejecución de una transacción de usuario, se ha producido la violación de alguna restricción. Estas reglas se han denominado *reglas restauradoras*.

Las características más destacables del nuevo método son:

- La obtención del conjunto de reglas restauradoras se realiza en tiempo de definición de la base de datos descargando al sistema en tiempo de ejecución.

- Cuando es posible, se resuelve el "problema de la falta de valores" sin implicar al usuario.

- Las reglas restauradoras podrían trasladarse a un sistema relacional definido en *SQL3* donde se utilizarían los disparadores (*triggers*) para implementar las reglas restauradoras.

- El método podría adaptarse fácilmente como método para resolver el problema de la actualización de una base de datos. Para ello sería necesario definir el T-árbol (nueva estructura que representa la acción de una regla restauradora) de inserción (resp. borrado) para cada predicado derivado. En ese T-árbol estarán representadas todas las formas de inducir la inserción (resp. el borrado) de instancias de ese predicado.

- El método es correcto ya que el procesamiento del T-árbol de una regla disparada cuya condición es cierta en un estado de base de datos genera un estado en el cual esa condición ya no se cumple.

El método propuesto tiene una limitación ya que las bases de datos a las que se puede aplicar deben ser estrictas. Así, una línea clara de trabajo futuro consistirá en modificarlo de forma que se pueda eliminar esta restricción. Para ello se intuye que será necesario almacenar información en los T-árboles que permita controlar que las

actualizaciones que se están proponiendo no deshacen otras propuestas con anterioridad. Será también necesario modificar el algoritmo de recorrido del T-árbol con la posibilidad de retroceder a nodos anteriores.

Por último, dada la estrecha relación que el problema de la actualización de vistas en una base de datos tiene con el problema de la restauración, otro objetivo cubierto por esta tesis ha sido la revisión de los métodos más relevantes que se han propuesto en la literatura para actualizar una base de datos deductiva.

Resum

En un sistema de base de dades, el canvi en el món real es modela per mitjà de l'execució de transaccions d'usuari que modifiquen un estat generant un altre que representa la nova situació. Aquest nou estat pot violar les restriccions d'integritat de l'esquema, restriccions que representen les propietats d'eixe món. La reacció més freqüent dels sistemes de bases de dades davant de la violació de la integritat consisteix a rebutjar la transacció que l'ha provocat, tornant la base de dades a l'estat anterior a la seua execució. Aquesta solució tan simple és, no obstant, usualment poc satisfactòria per a sistemes reals. Una alternativa a aquest comportament consisteix que el sistema modifique l'estat inconsistent de forma que es repare la violació provocada per la transacció d'usuari respectant els canvis proposats per aquesta. Es diu llavors, que el sistema ha restaurat la consistència de la base de dades.

En aquesta tesi s'ha proposat un mètode per a la restauració de la consistència en bases de dades relacionals amb vistes (o bases de dades deductives) que utilitza el llenguatge de regles d'un sistema actiu. Així, a partir del conjunt de restriccions d'integritat i del conjunt de regles deductives de l'esquema, el mètode genera un conjunt de regles d'activitat que restaura la consistència de la base de dades quan, com a conseqüència de l'execució d'una transacció d'usuari, s'ha produït la violació d'alguna restricció. Aquestes regles s'han denominat *regles restauradores.*

Les característiques més destacables del nou mètode són:

- L'obtenció del conjunt de regles restauradores es realitza en temps de definició de la base de dades descarregant al sistema en temps d'execució.
- Quan és possible, es resol el "problema de la falta de valors" sense implicar l'usuari.
- Les regles restauradores podrien traslladar-se a un sistema relacional definit en *SQL3* on s'utilitzarien els disparadors (*triggers*) per a implementar les regles restauradores.
- El mètode podria adaptar-se fàcilment com a mètode per a resoldre el problema de l'actualització d'una base de dades. Per a això seria necessari definir el T-arbre (nova estructura que representa l'acció d'una regla restauradora) d'inserció (resp. d'esborrat) per a cada predicat derivat. En eixe T-arbre estaran representades totes les formes d'induir la inserció (resp. l'esborrat) d'instàncies d'eixe predicat.
- El mètode és correcte ja que el processament del T-arbre d'una regla disparada la condició de la qual és certa en un estat de base de dades, genera un estat on aquesta condició ja no es compleix.

El mètode proposat té una limitació ja que les bases de dades a què es pot aplicar han de ser estrictes. Així, una línia clara de treball futur consistirà a modificar-lo de forma que es puga eliminar esta restricció. Per a això s'intueix que serà necessari emmagatzemar informació en els T-arbres que permeta controlar que les actualitzacions que s'estan proposant no desfan altres propostes amb anterioritat. Serà també necessari modificar l'algoritme de recorregut del T-arbre amb la possibilitat de retrocedir a nodes anteriors.

Finalment, donada l'estreta relació que el problema de l'actualització de vistes en una base de dades té amb el problema de la restauració, un altre objectiu cobert per aquesta tesi ha sigut la revisió dels mètodes més rellevants que s'han proposat en la literatura per a actualitzar una base de dades deductiva.

Abstract

In a database system, the change of reality is modelled by the execution of users transactions that update a database state generating another one that represents the new situation. The traditional behavior of database systems, when a transaction containing one or more violating updates occurs, is to roll back the transaction in its entirely. This simple rejection is obviously unsatisfactory for most real databases. One possible alternative to this behavior consists of allowing the system to update the inconsistent database state in such a way that the new state satisfies the user's updates and the integrity constraints. It is said, then that the system has repaired the database consistency.

In this thesis, we propose a method for repairing the consistency in a relational database with views (or deductive databases), i.e. a new method for solving the integrity enforcement problem. The new method uses the rule language in an active system. Therefore, from the set of integrity constraints and the set of deductive rules, the method generates a set of active rules in order to repair the consistency of the database when, after a user's transaction, some constraint is violated. This rules has been called *repair rules*.

The most outstanding features of the new method are:

- The set of repair rules is generated at database definition time. This improves the system performance at execution time.
- The "missing values problem" is solve without asking the user when it is possible.
- The repair rules could be incorporated in a relational system defined with $SQL3$. The repair rules would be translated into *triggers*.
- The method could be easily adapted as a method for solving the view-updating problem in a deductive database. It would be necessary to define, for each derived predicate, the insert (resp. delete) T-tree (the new structure which represents the action part in a repair rule). In this T-tree, we represent all the possible ways of inducing the insertion (rep. deletion) of instances of this derivate predicate.
- The method is correct since the process of a T-tree of a triggered rule whose condition is true in a database state, generates a state where this condition is not satisfied.

The proposed method has a limitation because the databases it can manage, must be strict. Therefore, a future work could consist on changing the method in such a way that this restriction could be avoided. With this purpose, it would be necessary to store additional information, in the T-trees. This information will allow of checking if the new updates proposed by the method undo other previous updates. It would be also necessary to change the T-tree processing algorithm in order to be able to come back to previous nodes in the T-tree.

Finally, due to the close relationship between the view update problem and the integrity enforcement problem, other goal covered by this thesis has been the revision of the most relevant methods proposed in the literature to solve the first problem in a deductive database.

Índice General

Capítulo 1

Introducción

En la larga evolución experimentada por la tecnología de bases de datos, la aparición del modelo relacional representó uno de sus hitos más importantes. Se puede afirmar que el estado actual de esta tecnología se caracteriza por el uso extendido de los sistemas relacionales que, cada vez en mayor medida, son utilizados en nuevos campos de aplicación. Es sin embargo este uso generalizado el que ha puesto en evidencia las limitaciones de estos sistemas a la hora de dar respuesta a los requisitos de las nuevas aplicaciones. Este hecho ha obligado a una continua transformación y mejora del modelo original en la línea de integrar en su definición funcionalidades y conceptos heredados de otras disciplinas informáticas. Esta capacidad de adaptación ha quedado reflejada recientemente en la propuesta del estándar $SQL3$ [31][1].

Las direcciones más relevantes en las que ha evolucionado el modelo relacional han sido: la incorporación de actividad, la mejora de la capacidad deductiva y la incorporación de características de la orientación a objetos. Se puede afirmar que el $SQL3$ define un modelo objeto-relacional activo y deductivo.

Como ha quedado dicho, una de las mejoras ha ido dirigida a la incorporación de mayor capacidad deductiva, superando las limitaciones del mecanismo de vistas de las versiones anteriores. Las vistas representan en el modelo relacional el mecanismo para definir relaciones derivadas, es decir, información implícita definida a través de reglas de conocimiento sobre el universo de discurso. Desde una perspectiva de uso, las vistas son tratadas como relaciones básicas y han sido utilizadas tradicionalmente con distintos fines (seguridad, esquemas externos, estructuración de consultas, etc.). Esta capacidad deductiva que sintácticamente ofrece el modelo relacional, ha adolecido tradicionalmente de importantes limitaciones que se evidenciaban cuando se hacía un uso generalizado de ella. Así, la capacidad de definir información implícita en forma de vistas estaba muy limitada ya que no se podían definir vistas recursivas y por otro lado, la actualización de vistas estaba restringida a un número muy reducido de casos. Otras limitaciones aparecían a la hora de intentar definir sobre ellas restricciones de integridad o disparadores.

Al intentar superar estas limitaciones, muchos problemas estudiados en el campo clásico de las bases de datos deductivas [30] han adquirido actualidad y muchos de ellos, todavía no resueltos de forma satisfactoria, vuelven a recabar interés.

[1] Aparecido a final de 1999.

Es este contexto el que justifica el trabajo de tesis presentado en esta memoria.

El trabajo desarrollado se puede enmarcar en los campos de las *bases de datos deductivas* y de las *bases de datos activas*, ya que aborda el problema de la *restauración de la consistencia en una base de datos relacional con vistas* es decir en una *base de datos deductiva*[2], utilizando para ello el mecanismo de actividad del sistema. En la solución propuesta a este problema se utiliza el lenguaje de reglas de los sistemas activos para especificar las reglas restauradoras de la consistencia. Para la generación de estas reglas se recurre, mejorándolos en algunos casos, a los resultados obtenidos en los campos de la *comprobación simplificada de la integridad* y de la *actualización del conocimiento* en bases de deductivas. Así pues, el trabajo realizado es un trabajo de síntesis en el que convergen distintas líneas de investigación.

Los objetivos concretos que se han planteado son los siguientes:

- Resumen de los fundamentos teóricos de las bases de datos deductivas y de las bases de datos activas;
- Revisión de los métodos propuestos para la actualización del conocimiento y para la restauración de la consistencia en el contexto de las bases de datos deductivas;
- Propuesta de un nuevo método para la restauración de la consistencia en una base de datos deductiva.

Estos objetivos se desarrollan en tres capítulos.

En el capítulo 2 se repasan los conceptos fundamentales de las bases de datos deductivas presentando una definición formal y homogénea de los problemas de la "comprobación simplificada de la integridad", de la "actualización del conocimiento" y de la "restauración de la consistencia"; en este capítulo se resumen también las principales características de las bases de datos activas.

En el capítulo 3 se hace una revisión de los métodos más relevantes propuestos en la literatura para la actualización del conocimiento y para la restauración de la consistencia en una base de datos deductiva.

En el capítulo 4 se propone un nuevo método para la restauración de la consistencia en una base de datos deductiva. Este método utiliza las reglas de actividad de un sistema activo para especificar las reglas restauradoras de la consistencia.

Además de estos capítulos, en la memoria se incluyen cuatro apéndices. En el apéndice A se incluye un índice de todas las definiciones, teoremas y algoritmos que se han realizado en la propuesta del nuevo método; en el apéndice B se presentan las abreviaturas y formalismos utilizados a lo largo del texto; en el apéndice C se realiza un resumen de las propiedades sintácticas que se utilizan en algún momento y por último, en el apéndice D se realiza una revisión de las principales semánticas declarativas y operacionales definidas para bases de datos deductivas.

[2] Éste será el término utilizado a partir de este punto para hacer referencia a una "base de datos relacional con vistas".

Capítulo 2

Bases de datos deductivas y bases de datos activas

2.1 Bases de datos deductivas

2.1.1 Introducción

Las bases de datos deductivas incrementan la capacidad expresiva de las bases de datos relacionales, incorporando reglas que permiten definir conocimiento de forma implícita, es decir derivar información a partir de la información almacenada explícitamente.

En el esquema de una base de datos deductiva se pueden distinguir dos tipos de relaciones: relaciones básicas cuyas tuplas se almacenan explícitamente y relaciones derivadas o vistas que se definen por reglas deductivas a partir de relaciones básicas y de otras relaciones derivadas. Un estado de base de datos está formado por un conjunto de hechos, tuplas de las relaciones básicas y por un conjunto de reglas deductivas que definen las relaciones derivadas.

Forman también parte del esquema de una base de datos las restricciones de integridad que restringen los estados válidos o las transiciones de estado permitidas. Una restricción de integridad es una propiedad que una base de datos debe cumplir en cualquier instante para ser consistente con la parcela del mundo real que la base de datos representa.

Todas estas ideas se presentan formalmente en el siguiente apartado.

2.1.2 Formalización 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 con un conjunto finito de símbolos y sin símbolos de función, y RI es un conjunto de restricciones de integridad representadas por fórmulas cerradas de L[1].

Un *estado de base de datos* D es un conjunto de fórmulas de L, que se denominan *sentencias de base de datos,* en las que todas las variables que aparecen libres se

[1] Con este formalismo sólo se pueden expresar restricciones estáticas que son las que restringen estados válidos.

consideran universalmente cuantificadas:

$$D \subseteq \{A[\leftarrow F]^2 : A \text{ es un átomo y } F \text{ es una fórmula bien formada}\}$$

Si F está ausente y A es *base* entonces la sentencia representa un *hecho,* en otro caso, la sentencia $A[\leftarrow F]$ representa una *regla deductiva* cuya *cabeza* es A y cuyo *cuerpo* es F. En una base de datos se distingue la parte extensional, BDE, formada por los hechos y la parte intensional, BDI, formada por las reglas deductivas. A los predicados que aparecen en BDE se les denomina *predicados básicos*; a los que aparecen en la cabeza de una regla deductiva se les denomina *predicados derivados.* Usualmente los conjuntos de predicados básicos y derivados son disjuntos.

De acuerdo con esta formalización y atendiendo a criterios sintácticos, se pueden distinguir varias clases de bases de datos deductivas. Una base de datos deductiva es *definida*[3] si los cuerpos de sus reglas son conjunciones de átomos; es *normal* si los cuerpos de sus reglas son conjunciones de literales, y, por último, es *general* si los cuerpos de sus reglas son fórmulas bien formadas cualesquiera. Algunos de los resultados más interesantes obtenidos en el campo de las bases de datos deductivas se han formulado para bases de datos normales; sin embargo, esto no supone una limitación en el alcance de las soluciones propuestas ya que, como han demostrado Lloyd y Topor, para toda base de datos (general) se puede obtener una base de datos normal equivalente en la semántica de la compleción [37]; debido a este resultado y sin pérdida de generalidad, a partir de ahora siempre se considerarán bases de datos normales.

Otro criterio de clasificación de las bases de datos deductivas, atendiendo a la presencia o no de recursividad en sus reglas, distingue entre bases de datos *jerárquicas,* en las que no se admite recursividad, y bases de datos *estratificadas,* en las que sí se admite siempre que no sea en términos de negación.

Un punto importante a destacar es el de la semántica asociada a la base de datos. Dar semántica a una base de datos deductiva significa definir cuál es la información implícitamente almacenada en ella. Se distinguen dos tipos de semántica, la *semántica declarativa* que define cuál es esta información y la *semántica operacional* que define cómo computarla[4]. La evaluación de una fórmula F en un estado D dependerá de la semántica asumida. Si se representa el estado D por una teoría de primer orden Tr (distinta según la semántica asumida [15]) entonces si F es una fórmula cerrada, F se evalúa a cierto (i.e. es cierta) en D si y sólo si $Tr \vDash F$, en caso contrario F se evalúa a falso (i.e. es falsa). Si F es una fórmula abierta entonces la evaluación de F en D es el conjunto $\{\overrightarrow{t} \mid Tr \vDash F(\overrightarrow{t})$ siendo $\overrightarrow{t}$ un vector de términos$\}$.

La evolución en el tiempo de una base de datos puede representarse por una secuencia de estados de bases de datos donde, dado un estado su sucesor se obtiene aplicando a éste una transacción. Una transacción T que modifica un estado de base

[2] En el apéndice B se pueden encontrar los símbolos, expresiones y abreviaturas que se han utilizado a lo largo del texto.

[3] En el apéndice C se han definido formalmente algunos conceptos utilizados a lo largo de la memoria. También en este apéndice se definen las propiedades que puede tener una base de datos deductiva.

[4] En el apéndice D se presenta un resumen de las semánticas para bases de datos deductivas que se han propuesto en la literatura.

de datos D está formada por dos conjuntos de sentencias de base de datos (T_{ins}, T_{bor})[5] donde T_{ins} es el conjunto de sentencias que van a ser insertadas por la transacción y T_{bor} es el conjunto de sentencias que van a ser borradas por la transacción. El estado de base de datos $D' \equiv T(D)$ resultante de aplicar a D la transacción T es $(D \cup T_{ins}) - T_{bor}$.

Por último, una propiedad deseable para una base de datos deductiva es la de *independencia del dominio.* Esta propiedad asegura que la evaluación de una fórmula en la base de datos depende sólo de las extensiones de los predicados que aparecen en ella. Esta propiedad caracteriza el conjunto de bases de datos consideradas "razonables". Determinar si una base de datos es independiente del dominio o no, no es decidible; sin embargo se pueden encontrar otras propiedades decidibles que la impliquen (p.e. la propiedad de *rango restringido* o la propiedad de ser *permitida*).

A continuación, en los apartados 2.1.3 y 2.1.4 se definen de forma resumida los problemas de la comprobación simplificada de la integridad y de la actualización del conocimiento en bases de datos deductivas. Esta presentación se justifica por el hecho de que, como ya se ha comentado en la introducción, los resultados obtenidos en la resolución de estos problemas van a ser utilizados en este trabajo de tesis. En el apartado 2.1.5 se presenta el problema de la restauración de la integridad.

2.1.3 Definición del problema de la comprobación simplificada de la integridad en bases de datos deductivas

Si se formaliza una base de datos como una teoría de primer orden ([30], [34], [52]), las restricciones de integridad estáticas se pueden representar en la mayoría de los casos por fórmulas bien formadas cerradas del lenguaje subyacente a la teoría.

En este contexto, el problema de la comprobación de la integridad se puede enunciar de la forma siguiente:

$\Longrightarrow$ Dados:

- el esquema (L, RI) de una base de datos deductiva,
- un estado de base de datos D de ese esquema tal que $\forall W \in RI$ se cumple que D *satisface* W,
- una transacción T, y
- el estado de base de datos $D' = T(D)$

$\Longrightarrow$ comprobar que $\forall W \in RI$ se cumple que D' *satisface* W.

Así enunciado el problema, es necesario definir qué significa que una base de datos satisface una restricción de integridad. Si se representa el estado de base de datos D por una teoría de primer orden el *concepto de satisfacción* se define como sigue: "Sea W una restricción de integridad y Tr la teoría que representa el estado de base de datos D en la semántica asumida, entonces D satisface W si y sólo si $Tr \models W$[6]". El

[5] Otra forma de representar una transacción es mediante un conjunto de operaciones de la forma *insertar*(S) o *borrar*(S) donde S es una sentencia de base de datos.

[6] Otra definición del concepto de satisfación menos frecuente es la siguiente: "D satisface W si y sólo si $Tr \cup \{W\}$ es consistente".

concepto de violación se define en términos del concepto de satisfacción: "D viola W si y sólo si no se cumple que D satisface W".

Se dice que un estado de base de datos D es *consistente* o *íntegro* si, para toda restricción W de RI, D satisface W.

Una vez establecido el concepto de satisfacción, se puede definir un *método de comprobación de la integridad* como un procedimiento de decisión tal que, dado un estado de base de datos D y una restricción de integridad W, decide si D satisface o 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 expresadas por fórmulas con variables cuantificadas universalmente cuyos dominios pueden ser muy extensos. La comprobación de la integridad podría simplificarse si se consideran sólo los "cambios" que la transacción ha producido en la base de datos. En esta línea, todos los métodos propuestos para la comprobación de la integridad simplifican dicha comprobación suponiendo que la base de datos era consistente 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.

Los principales métodos propuestos para la comprobación simplificada de la integridad aparecen en [4], [9], [10], [13], [14], [18], [22], [36], [44] y [55]. Resúmenes de dichos métodos pueden encontrarse en [11], [23] y [38] .

2.1.4 Definición del problema de la actualización del conocimiento en bases de datos deductivas

La asimilación del conocimiento en bases de datos deductivas es el proceso por el cual una nueva información es incorporada a la base de datos. Este problema 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.

En el caso de las bases de datos deductivas este problema puede ser más complejo debido a la mayor capacidad expresiva del lenguaje de definición de reglas deductivas y a que el conocimiento a asimilar, traducido en términos de *requisitos de actualización* (representados por fórmulas bien formadas), puede ser más general [1]. La forma más sencilla de satisfacer un requisito de actualización es añadir explícitamente el nuevo conocimiento a la base de datos; sin embargo, esto no es siempre lo más adecuado ya que, 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. Por ello, los métodos existentes proponen modificar el estado de base de datos aplicándole una transacción de forma que el nuevo estado obtenido satisfaga el requisito de actualización.

El problema puede entonces enunciarse de la forma siguiente:

⇒ Dados:

- el esquema (L, RI) de una base de datos deductiva,
- un estado de base de datos D de ese esquema tal que $\forall W \in RI$ se cumple que D satisface W, y
- un requisito de actualización U, representado por una fórmula cerrada de L, tal que U es falso en D

⇒ encontrar una transacción T tal que $\forall W \in RI$ se cumple que $D' = T(D)$ satisface W y U es cierto en D'.

Algunos problemas asociados son: la eficiencia del método, la definición de criterios para seleccionar la transacción más adecuada entre el conjunto de las obtenidas, la completitud del conjunto de transacciones y la generación de explicaciones sobre las soluciones obtenidas.

Los principales métodos propuestos en la literatura aparecen en [20], [28], [33], [35], [59] y [65]. Un resumen interesante se puede encontrar en [25].

2.1.5 Definición del problema de la restauración de la consistencia en bases de datos deductivas

La reacción más frecuente de los sistemas de bases de datos ante la violación de la integridad consiste en rechazar la transacción que la ha provocado, devolviendo la base de datos al estado anterior a su ejecución. Esta solución tan simple es, sin embargo, usualmente poco satisfactoria para sistemas reales ya que el rechazo de la transacción puede suponer la anulación de muchas actualizaciones, aun cuando la violación producida por la transacción pueda repararse con otras modificaciones que respeten los cambios originales. Como alternativa a esta solución, en la literatura se han propuesto distintos métodos con el objetivo de que el sistema reaccione ante una violación ejecutando una transacción compensatoria que repare la violación provocada por la ejecución de una transacción de usuario. Este problema formalmente se enuncia como sigue:

⇒ Dados:

- el esquema (L, RI) de una base de datos deductiva,
- un estado de base de datos D tal que $\forall W \in RI$ se cumple que D satisface W,
- una transacción $T = (T_{ins}, T_{bor})$, y
- un estado de base de datos $D' = T(D)$ tal que $\exists W \in RI$, D' viola W

⇒ encontrar una transacción $T^* = (T^*_{ins}, T^*_{bor})$, que cumpla:

- $T^*_{ins} \cap T^*_{bor} = \emptyset$,
- $T^*_{bor} \cap T_{ins} = \emptyset$,
- $T^*_{ins} \cap T_{bor} = \emptyset$

– D'' satisface W donde $D'' = (D' \cup T^*_{ins}) - T^*_{bor}$.

De este enunciado se deriva que el problema de la restauración de la consistencia es un caso particular del problema de la actualización del conocimiento en bases de datos en el que la fórmula que representa el conocimiento a actualizar es, en este caso, una restricción de integridad.

Los principales métodos propuestos en la literatura aparecen en [16], [42] y [45].

Debido a la importancia que para el desarrollo del presente trabajo tienen los dos últimos problemas presentados, en el capítulo 3 se realiza una revisión bibliográfica de los métodos principales para actualizar de forma segura una base de datos, bien comprobando las restricciones en el proceso de actualización, bien restaurando la integridad en caso de que sea violada.

2.2 Bases de datos activas

2.2.1 Introducción

Las bases de datos activas extienden la capacidad expresiva de las bases de datos relacionales incorporando reglas que modelan un comportamiento activo del sistema. Estas reglas permiten que el sistema reaccione de forma autónoma, es decir sin intervención del usuario, ante la ocurrencia de determinados eventos y se les denomina *reglas activas* [46], [63].

Una regla activa consta de tres componentes:

- Evento: define el suceso o estímulo que dispara la regla.
- Condición: define las condiciones de contexto que se deben producir para que la regla sea ejecutada.
- Acción: define la acción o acciones que el sistema debe ejecutar de forma autónoma.

Debido a su estructura, *E*vento-*C*ondición-*A*cción, otra forma de hacer referencia a una regla activa es con la expresión *regla ECA*. El significado de una regla *ECA* es el siguiente: "cuando se produce el *evento,* si la *condición* se cumple entonces la *acción* debe ser ejecutada". Cuando el sistema detecta que ha sucedido el evento de una regla, se dice que ésta se ha *disparado.* Cuando su condición se evalúa a cierto y se ejecuta su acción, se dice que la regla ha sido *procesada.*

A diferencia de otros campos de investigación, el área de las bases de datos activas se ha desarrollado sin que exista una sólida fundamentación teórica. Muchos de los resultados obtenidos se han aplicado ya a los sistemas comerciales sin que exista todavía un modelo formal aceptado. El resultado de esta situación es que se han desarrollado numerosos sistemas activos que, aunque comparten nociones y conceptos, utilizan lenguajes diferentes y proponen modelos de ejecución distintos. Resulta pues difícil definir una base de datos activa desde un punto de vista formal que se ajuste a todos los sistemas ya existentes. Es por este motivo por el que la presentación de las bases de datos activas de este capítulo va a consistir en resumir, sin concretar en

ningún sistema, las dos características fundamentales de un sistema activo que son las siguientes:

- El *modelo de conocimiento,* que define qué se puede expresar con una regla ECA; y
- El *modelo de ejecución,* que define cómo se procesa una regla ECA en tiempo de ejecución.

2.2.2 Modelo de conocimiento

Independientemente del lenguaje utilizado para su definición, los sistemas consideran las tres componentes de una regla antes comentados. Las formas más usuales que se pueden encontrar para la definición de eventos, condiciones y acciones son las siguientes [47]:

- Evento: los eventos de una regla pueden ser operaciones de base de datos (p.e. inserción de una tupla en una relación), operaciones predefinidas por el usuario (p.e. envío de un mensaje a un determinado objeto), operaciones de control de transacciones (p.e. ejecución de la sentencia *commit),* excepciones (p.e. intento de acceso a ciertos datos sin la autorización necesaria); pueden ser también eventos temporales (p.e. cada lunes), o incluso eventos *externos* (p.e. el hecho de que una temperatura ha alcanzado un cierto valor). Se dice que un evento es *primitivo* si consta nada más que de una ocurrencia de alguna de las categorías descritas. Un evento es *compuesto* si se obtiene como combinación de varios eventos primitivos y/o compuestos. La combinación puede realizarse mediante conectivas lógicas ($\vee, \wedge, \neg$), mediante la combinación secuencial, o por composición temporal.
- Condición: las condiciones de las reglas son expresiones lógicas escritas en el lenguaje de base de datos o procedimientos de tipo lógico escritos en un lenguaje de programación. La ausencia de condición o el uso de un predicado especial (*cierto* o *true*) significa que la acción de la regla debe ejecutarse siempre que ésta se dispare.
- Acción: La acción de una regla ECA puede consistir en operaciones de base de datos, llamadas a procedimientos, mensajes al usuario, operaciones de control de transacciones o la acción cambio de evento que sustituye el evento disparador por otro.

Algunos sistemas permiten que las reglas estén parametrizadas de forma que se puedan transmitir valores desde el evento a la condición y a la acción y de la condición a la acción.

2.2.3 Modelo de ejecución

Una vez se ha definido un conjunto de reglas ECA, el sistema debe procesarlas cuando sea oportuno. Este procesamiento puede ser diferente en cada sistema en función de las siguientes características [47]:

- *Modos de acoplamiento evento-condición* y *condición-acción.* El modo de acoplamiento evento-condición (resp. condición-acción) define la relación existente, con respecto a la transacción activa, entre la detección de un evento relevante para una regla y la evaluación de su condición (resp. la evaluación de la condición y la ejecución de la acción). Existen tres modos de acoplamiento:

  - *Inmediato*: en este modo, la condición es evaluada (resp. la acción es ejecutada) inmediatamente después de la detección del evento (resp. evaluación de la condición).
  - *Diferido*: en este modo, la condición es evaluada (resp. la acción es ejecutada) justo antes de que la transacción en la que se ha detectado el evento (resp. evaluado la condición) sea confirmada.
  - *Desacoplado*: en este modo, la condición es evaluada (resp. la acción es ejecutada) en una transacción distinta a aquélla en la que se detectó el evento (resp. se evaluó la condición).

- *Granularidad de las reglas.* Esta propiedad determina la relación entre la ocurrencia de un evento y la instanciación de la regla que dispara. Hay dos posibles granularidades a tener en cuenta:

  - *Orientada a la tupla*: en este caso, si el evento afecta a varias tuplas, la instancia de la regla asociada a cada tupla es procesada de forma independiente.
  - *Orientada al conjunto*: en este caso, aunque el evento afecte a varias tuplas, la regla que dispara debe procesarse sólo una vez para todas ellas.

- *Efecto de los eventos.* Esta propiedad establece si los eventos son considerados aisladamente o si se considera su efecto global dentro de la transacción; en este último caso se dice que el efecto es *neto.*

- *Política de procesamiento de las reglas.* Esta propiedad establece cuándo se procesan las reglas que son disparadas por la ejecución de otras reglas. Existen dos modos:

  - *Iterativo*: las reglas disparadas por cualquier motivo se van considerando una tras otra.
  - *Recursivo*: cuando el procesamiento de una regla supone el disparo de otras, éstas deben ser procesadas antes de que acabe aquélla que las disparó.

- *Resolución de conflictos.* Esta propiedad determina qué sucede cuando hay varias reglas disparadas. Posibles alternativas para resolver este problema son:

  - *Aleatoriedad:* la regla elegida para ser procesada se escoge arbitrariamente.
  - *Definición de prioridades:* a las reglas se les asigna una prioridad *absoluta* o *relativa a otras reglas.* En este caso, al elegir una regla para ser procesada no puede haber ninguna otra disparada de prioridad mayor.

2.2.4 Propiedades de un conjunto de reglas ECA

Dado un conjunto de reglas ECA, existen dos propiedades que resulta interesante estudiar:

- *Terminación*: Un conjunto de reglas cumple la propiedad de terminación si para cualquier estado de base de datos y para cualquier actualización inicial, el procesamiento de las reglas acaba en un tiempo finito.
- *Confluencia*: Un conjunto de reglas es confluente si para cualquier estado de base de datos y para cualquier actualización inicial, el estado de base de datos final que se alcanza tras el procesamiento de las reglas es único, i.e. este estado es independiente del orden en el que se disparan y procesan las reglas.

Estas dos propiedades han sido extensamente estudiadas habiéndose propuesto varios algoritmos (normalmente conservadores) para comprobarlas [3], [5].

Capítulo 3

Actualización segura de una base de datos deductiva

Como ya se ha comentado en el apartado 2.1.4, el objetivo de cualquier método de actualización es generar una transacción que modifique el estado actual de la base de datos de forma que el nuevo estado cumpla el requisito de actualización propuesto por el usuario. Evidentemente estos métodos deben generar transacciones de actualización seguras, es decir, tales que el nuevo estado satisfaga las restricciones de integridad presentes en el esquema. Para cumplir esta exigencia, los métodos de actualización siguen dos aproximaciones:

a) Aproximación restrictiva: las transacciones generadas por el método son validadas (aceptadas o rechazadas) en función de que se satisfagan o no las restricciones en el nuevo estado. Un método que sigue esta aproximación es el de Guessoum y Lloyd (apartado 3.3).

b) Aproximación restauradora: el método incorpora el mantenimiento de las restricciones en el proceso de generación de la transacción, incluyendo las acciones restauradoras necesarias. En esta aproximación se incluyen los métodos de Kakas y Mancarella, Decker, Wüthrich, Teniente y Olivé, y Pastor (apartados 3.2, 3.4, 3.6, 3.7 y 3.8).

Una alternativa a la "actualización segura" consiste en acoplar a cualquier método de actualización sin comprobación de restricciones (como p.e. el de Larson y Sheth del apartado 3.5) un método específico para la restauración de la consistencia en caso de violación (como el método de Ceri et al. del apartado 3.9). El método de Teniente y Olivé también plantea esta alternativa como posible.

El resto de este capítulo se dedica a presentar una caracterización precisa del problema de la actualización de una base de datos y a resumir, en orden cronológico, todos los métodos antes citados.

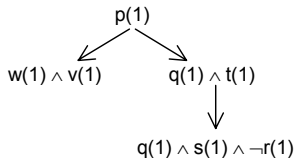
3.1 Caracterización del problema de la actualización de una base de datos deductiva

En el apartado 2.1.4 se enunció el problema de la actualización de la forma siguiente: "Dados el esquema (L, RI) de una base de datos deductiva, un estado de base de datos D de ese esquema tal que $\forall W \in RI$ se cumple que D satisface W, y dado un requisito de actualización U tal que U no es cierto en D entonces encontrar una transacción T tal que $\forall W \in RI$ se cumple que $D' = T(D)$ satisface W y U es cierto en D'". Para resolver este problema se han propuesto distintos métodos que estudian, siguiendo estrategias distintas, el árbol de derivación que tiene como raíz el conocimiento a actualizar concentrando la atención en la parte de la base de datos de la cual depende el nuevo conocimiento.

Ejemplo 1 *Sea una base de datos con el siguiente conjunto de reglas deductivas:*

$$1 : p(x) \leftarrow q(x) \wedge t(x)$$
$$2 : p(x) \leftarrow w(x) \wedge v(x)$$
$$3 : t(x) \leftarrow s(x) \wedge \neg r(x)$$

Donde $\{q, w, v, s, r\}$ son predicados básicos y $\{p, t\}$ son predicados derivados y sea el requisito de actualización $U = p(1)$. El árbol de derivación asociado a $p(1)$ en el que se representan los diferentes caminos por los que $p(1)$ podría derivarse en esa base de datos[1] *es el siguiente:*



En la figura se aprecia que $p(1)$ es cierto en la base de datos si se da una de las cuatro circunstancias siguientes:

- $\{w(1), v(1)\} \subseteq BDE$
- $\{q(1), s(1)\} \subseteq BDE$ *y* $\{r(1)\} \nsubseteq BDE$
- $\{p(1)\} \subseteq BDE$[2]
- $\{q(1), t(1)\} \subseteq BDE$

Así pues, los distintos métodos proponen estrategias para conseguir transacciones que aplicadas a la base de datos aseguren una de las cuatro condiciones anteriores.

Desde un punto de vista genérico y sin considerar ninguna estrategia de actualización particular, existen algunos problemas asociados a la actualización de una base de datos que es importante destacar antes de presentar los distintos métodos.

[1] No se presenta una definición formal del término *árbol de derivación* ya que para ello habría que elegir una semántica operacional concreta.

[2] Usualmente, en un esquema de base de datos se considera que el conjunto de predicados básicos y el conjunto de predicados derivados son disjuntos; por ello, ni esta solución ni la siguiente serían admitidas en este supuesto ya que proponen la existencia de un hecho sobre un predicado derivado.

Tiempo de generación de la solución. Esta propiedad hace referencia a cuándo se obtienen las posibles soluciones para un requisito de actualización. Hay tres posibilidades:

- Tiempo de ejecución: el árbol de derivación para el requisito de actualización se genera cuando la actualización es solicitada. En este caso, los requisitos de actualización pueden ser más generales ya que las soluciones se construyen a medida del requisito. Estos métodos suelen proporcionar la primera solución encontrada.

- Tiempo de definición: el árbol de derivación para un requisito de actualización se estudia cuando se define el esquema de la base de datos, lo que supone una mejora ya que determinadas tareas sólo se realizan una vez. En este caso los requisitos de actualización admitidos deben tener una forma restringida para que se ajusten a las actualizaciones que se han considerado. Estos métodos suelen generar todas las transacciones posibles, y es en tiempo de ejecución cuando se decide cuál ejecutar.

- Mixto: en este caso una parte de la solución se genera en tiempo de definición del esquema y se completa en tiempo de ejecución.

Ejemplo 2 *En el ejemplo 1, un método que obtuviese la solución en tiempo de ejecución estudiaría el árbol de derivación de la actualización* $p(1)$ *para encontrar una solución. Un método que trabajase en tiempo de definición de datos estudiaría el requisito genérico* $p(x)$ *para obtener soluciones que luego se instanciarían en tiempo de ejecución.*

Variables existencialmente cuantificadas. Dada una regla deductiva de una base de datos normal, a las variables que aparecen en el cuerpo de la regla y no aparecen en la cabeza se les denomina *variables existencialmente cuantificadas.* Esta denominación es debida a la equivalencia lógica existente entre las fórmulas: $\forall x_1 \ldots \forall x_i \ldots \forall x_m (A \leftarrow L_1 \wedge \ldots \wedge L_n)$ y $\forall x_1 \ldots \forall x_{i-1} \forall x_{i+1} \ldots \forall x_m (A \leftarrow \exists x_i (L_1 \wedge \ldots \wedge L_n))$ si las variables x_j $(1 \leq j \leq m)$ son libres en $(A \leftarrow L_1 \wedge \ldots \wedge L_n)$ y x_i no aparece en A.

La presencia de este tipo de variables en las reglas deductivas puede provocar la aparición del problema llamado *falta de valores* durante la generación de las transacciones que resuelven un requisito de actualización.

Ejemplo 3 *Sea una base de datos con una sola regla deductiva* $p(x) \leftarrow q(x,y) \wedge t(x,y)$ *y el requisito de actualización* $p(1)$*. En este caso aparece el problema de falta de valores, ya que para resolver este requisito se debería insertar una instancia de* $q(1,y)$ *y de* $t(1,y)$ *para un valor de "*y*" que se desconoce.*

Una solución sencilla a este problema consiste en utilizar el valor nulo para la variable de la que se desconoce el valor o bien un valor cualquiera proporcionado por el usuario o extraído sin criterio de la base de datos. Aunque en ocasiones esta solución es la única posible, en otras se puede elegir un valor tal que la transacción obtenida sea más sencilla.

Ejemplo 4 *Sea la base de datos del ejemplo 3 y considérese que el hecho* $t(1,2)$ *forma parte de* BDE*. Si se considera la solución trivial, para el requisito de actualización* $p(1)$ *se generaría una transacción con las operaciones* $\{insertar(q(1,?)), insertar(t(1,?))\}$ *donde el símbolo* ? *representa el valor nulo o con las operaciones* $\{insertar(q(1,c)), insertar(t(1,c))\}$ *donde "*c*" es una constante cualquiera. Sin embargo es fácil darse cuenta de que la transacción más evidente es* $\{insertar(q(1,2))\}$ *que puede obtenerse si se busca en la base de datos un valor para la variable "*y*" con criterios apropiados.*

Recursividad. La presencia de reglas recursivas en la base de datos puede complicar la generación de la transacción, ya que el árbol de derivación puede ser infinito para un determinado requisito de actualización, lo que supone la existencia de infinitas transacciones posibles para satisfacerlo.

Ejemplo 5 *Sea una base de datos con el siguiente conjunto de reglas deductivas:*

$$1 : p(x,y) \leftarrow q(x,y)$$
$$2 : p(x,y) \leftarrow q(x,z) \wedge p(z,y)$$

Y sea el requisito de actualización $U = p(1,1)$*. Para satisfacer este requisito hay infinitas transacciones posibles:*

$$T_1 : \{insertar(q(1,1))\}$$
$$T_2 : \{insertar(q(1,2)), insertar(q(2,1))\}$$
$$T_3 : \{insertar(q(1,2)), insertar(q(2,3)), insertar(q(3,1))\}$$
$$\ldots$$

Problema de la información asumida. En presencia de negación en las reglas deductivas, es posible que algunas soluciones que podrían parecer correctas no lo sean, ya que alguna información que se ha supuesto cierta (resp. falsa), durante la construcción de la solución pase a ser falsa (resp. cierta) debido a las actualizaciones propuestas más adelante.

Ejemplo 6 *Sea una base de datos con tan solo un hecho* $\{r(1,2)\}$ *y con el siguiente conjunto de reglas deductivas:*

$$1 : p(x) \leftarrow r(x,y) \wedge \neg s(x,y) \wedge q(x,y)$$
$$2 : s(1,2) \leftarrow q(1,2)$$

Y sea el requisito de actualización $U = p(1)$*. Para satisfacer este requisito es necesario que la fórmula* $\exists y(r(1,y) \wedge \neg s(1,y) \wedge q(1,y))$ *sea cierta. Supóngase un método que elige la substitución* $\{y/2\}$ *para construir la solución. El hecho* $r(1,2)$ *ya es cierto en la base de datos por lo que no es necesario insertarlo; por otra parte el hecho* $s(1,2)$ *no pertenece a la base de datos por lo tanto el método podría concluir que la transacción* $\{insertar(q(1,2))\}$ *es una buena solución al requisito de actualización. Sin embargo es fácil observar que no es una solución correcta ya que la inserción de* $q(1,2)$ *induce la inserción de* $s(1,2)$*. Una transacción correcta sería por ejemplo* $\{insertar(r(1,3)), insertar(q(1,3))\}$.

Tratamiento de las restricciones de integridad. La solución propuesta para un requisito de actualización puede suponer la violación de alguna restricción de integridad por lo que es interesante estudiar cómo integra cada método, si lo hace, su estrategia con la comprobación de las restricciones del esquema.

Ejemplo 7 *Sea una base de datos con una sola regla deductiva:*

$$1 : p(x) \leftarrow q(x) \wedge r(x)$$

Que debe satisfacer la restricción de integridad $W = \forall x(r(x) \rightarrow t(x))$. *Ante el requisito de actualización* $U = p(1)$ *un método que no se preocupe de la consistencia podría proponer como solución la transacción* $\{insertar(q(1)), insertar(r(1))\}$. *Esta solución debería, sin embargo, ser rechazada ya que su ejecución conduce a un estado de base de datos que viola la restricción de integridad* W. *Un método que integre la generación de las soluciones con la restauración de la integridad podría generar la transacción* $\{insertar(q(1)), insertar(r(1)), insertar(t(1))\}$.

A continuación se presentan algunos métodos que se han considerado relevantes. La presentación sigue el esquema que se expone a continuación:

1. Semántica asumida: la semántica declarativa determinará el conjunto de posibles soluciones al requisito de actualización y la semántica operacional constituirá la herramienta para computar esas soluciones.

2. Bases de datos: donde se indicará el tipo de bases de datos y de restricciones de integridad para los que está definido el método.

3. Requisitos de actualización: donde se indicará la forma sintáctica de los requisitos de actualización permitidos al usuario.

4. Transacciones generadas: donde se destacará el tipo de soluciones obtenidas y también si sólo se obtiene una o todas las soluciones posibles.

5. Descripción del método: donde se presentará la estrategia seguida para generar las transacciones.

6. Corrección y completitud del método.

7. Resumen: donde se destacará cómo el método resuelve los problemas antes mencionados.

3.2 Método de Kakas y Mancarella [33]

Este método se basa en el procedimiento de abducción de Eshghi y Kowalski definido en [24].

Semántica asumida.

La semántica declarativa asumida es la del modelo estable [27]. En cuanto a la semántica operacional se considera el procedimiento de abducción de Eshghi y Kowalski.

Bases de datos.

El método está definido para bases de datos localmente estratificadas sin restricciones de integridad específicas. Debido a la semántica operacional asumida, la base de datos se traduce a un marco abductivo. Dado un estado de base de datos $D = BDE \cup BDI$, el marco abductivo asociado con D es $< BDI^*, Ab, RI^* >$ donde:

- BDI^* es el conjunto de reglas deductivas obtenido a partir de BDI, reemplazando cada literal negativo $\neg q(\overrightarrow{t})$ por un nuevo literal positivo $q^*(\overrightarrow{t})$.
- Ab está formado por el conjunto de símbolos de predicados básicos junto con un nuevo predicado p^* por cada predicado básico o derivado p. Ab es el conjunto de predicados abducibles. Un átomo A es abducible si su predicado lo es.
- RI^* es un conjunto de restricciones de integridad tal que para todo conjunto Δ de hipótesis abducidas sobre predicados de Ab representadas por átomo abducibles, $BDI^* \cup \Delta$ debe satisfacer $RI^* = \{\leftarrow p(\overrightarrow{t}) \wedge p^*(\overrightarrow{t})$: p es un predicado de la base de datos$\} \cup \{p(\overrightarrow{t}) \vee p^*(\overrightarrow{t}) : p$ es un predicado de la base de datos$\}$[3].

Ejemplo 8 *Sea una base de datos que tiene el siguiente conjunto de reglas deductivas:*

$$1 : p(x) \leftarrow q(x) \wedge \neg r(x)$$
$$2 : p(x) \leftarrow t(x)$$
$$3 : r(x) \leftarrow s(x)$$

Donde $\{q, s, t\}$ son predicados básicos. El marco abductivo asociado a esta base de datos es:

$$\begin{aligned} BDI^* = \quad & \{p(x) \leftarrow q(x) \wedge r^*(x), \\ & \ p(x) \leftarrow t(x), \\ & \ r(x) \leftarrow s(x)\} \\ Ab = \quad & \{q, s, t, p^*, q^*, r^*, s^*, t^*\} \\ RI^* = \quad & \{\leftarrow p(x) \wedge p^*(x), p(x) \vee p^*(x), \\ & \leftarrow q(x) \wedge q^*(x), q(x) \vee q^*(x), \\ & \leftarrow r(x) \wedge r^*(x), r(x) \vee r^*(x), \\ & \leftarrow s(x) \wedge s^*(x), s(x) \vee s^*(x), \\ & \leftarrow t(x) \wedge t^*(x), t(x) \vee t^*(x)\} \end{aligned}$$

Requisitos de actualización.

Un requisito de actualización tiene la forma $insertar(A)$ (resp. $borrar(A)$) donde A es un átomo derivado base[4].

Transacción generada.

La transacción obtenida está formada por un conjunto de operaciones de inserción y borrado de hechos.

[3] Estas restricciones de integridad pretenden definir la equivalencia entre $p^*(\overrightarrow{t})$ y $\neg p(\overrightarrow{t})$.

[4] Ver apéndice C para la definición de átomo o literal básico o derivado y para la definición de átomo o literal base.

Descripción del método.

Dado un requisito de actualización $insertar(p(\overrightarrow{c}))$ (resp. $borrar(p(\overrightarrow{c}))$) el método distingue dos etapas:

- Paso A: Búsqueda de una derivación abductiva para $\leftarrow p(\overrightarrow{c})$ (resp. $\leftarrow p^*(\overrightarrow{c})$). Este paso devuelve un conjunto de hipótesis Δ tal que: $BDI^* \cup \Delta$ implica $p(\overrightarrow{c})$ (resp. $p^*(\overrightarrow{c})$) y tal que $BDI^* \cup \Delta$ es consistente y no viola RI^*.

- Paso B: Obtención de una transacción T sobre BDE tal que el estado de base de datos obtenido al aplicar la transacción T a D implique cualquier hipótesis en Δ.

A continuación se presentan estas dos etapas con más detalle. Para ello, dado un átomo $L = p(\overrightarrow{t})$ (resp. $L = p^*(\overrightarrow{t})$) con la expresión L^* se representará al átomo $p^*(\overrightarrow{t})$ (resp. $p(\overrightarrow{t})$), además se dice que una regla de selección R es *segura* si es una función parcial tal que dado un objetivo $\leftarrow L_1 \wedge \ldots \wedge L_k$ $(k \geq 1)$ devuelve un átomo L_j $(1 \leq j \leq k)$ tal que L_j no es abducible o L_j es abducible y base.

Paso A: Procedimiento de demostración abductivo. El procedimiento abductivo se basa en las definiciones de *derivación abductiva* y *derivación de consistencia*.

Derivación abductiva. Una derivación abductiva desde $(G_1\ \Delta_1)$ hasta $(G_n\ \Delta_n)$ vía una regla segura R es una secuencia $(G_1\ \Delta_1), (G_2\ \Delta_2), \ldots, (G_n\ \Delta_n)$ tal que $\forall i\ (1 \leq i \leq n)$ G_i tiene la forma $\leftarrow L_1 \wedge \ldots \wedge L_k$, Δ_i es un conjunto de hipótesis abducibles, la regla R selecciona el literal L_j $(1 \leq j \leq k)$ y $(G_{i+1}\ \Delta_{i+1})$ se obtiene de acuerdo con una de las siguientes reglas:

- Regla $A1$: 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 de BDI^* con G_i sobre el literal L_j.

- Regla $A2$: Si L_j es abducible y $L_j \in \Delta_i$ entonces $G_{i+1} := \leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k$ y $\Delta_{i+1} := \Delta_i$.

- Regla $A3$: Si L_j es abducible, básico, $L_j \notin \Delta_i$ y $L_j^* \notin \Delta_i$ entonces $G_{i+1} := \leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k$ y $\Delta_{i+1} := \Delta_i \cup \{L_j\}$.

- Regla $A4$: Si L_j es abducible, es derivado y $L_j \notin \Delta_i$ y hay una derivación de consistencia desde $(\{\leftarrow L_j^*\}\ \Delta_i \cup L_j)$ hasta $(\{\}\ \Delta')$ entonces $G_{i+1} := \leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k$ y $\Delta_{i+1} := \Delta'$.

Las reglas $A1$ y $A2$ de la derivación abductiva se corresponde con pasos de resolución SLD[5] utilizando respectivamente las reglas de BDI^* y las hipótesis abducidas. En las reglas $A3$ y $A4$ una nueva hipótesis es generada y añadida a Δ después de comprobar su consistencia (en la regla $A3$ esta prueba de consistencia es trivial).

[5] Procedimiento resolución Lineal con función de Selección para cláusulas Definidas [32].

Derivación de consistencia. Una derivación de consistencia desde $(F_1\ \Delta_1)$ hasta $(F_n\ \Delta_n)$ vía una regla segura R es una secuencia $(F_1\ \Delta_1)$, $(F_2\ \Delta_2)$, ...,$(F_n\ \Delta_n)$ tal que $\forall i\ (1 \leq i \leq n)\ F_i$ tiene la forma $\{\leftarrow L_1 \wedge \ldots \wedge L_k\} \cup F_i'$ siendo F_i' un conjunto de objetivos, Δ_i es un conjunto de hipótesis abducibles, la regla R selecciona el literal L_j $(1 \leq j \leq k)$ y $(F_{i+1}\ \Delta_{i+1})$ se obtiene de acuerdo con una de las siguientes reglas:

- Regla $C1$: Si L_j no es abducible entonces $F_{i+1} := C \cup F_i'$ donde C es el conjunto de todos los resolventes de cláusulas de BDI^* con $\leftarrow L_1 \wedge \ldots \wedge L_k$ sobre el literal L_j, $\square \notin C$ y $\Delta_{i+1} := \Delta_i$.

- Regla $C2$: Si L_j es abducible, $L_j \in \Delta_i$ y $k > 1$ entonces $F_{i+1} := \{\leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k\} \cup F_i'$ y $\Delta_{i+1} := \Delta_i$.

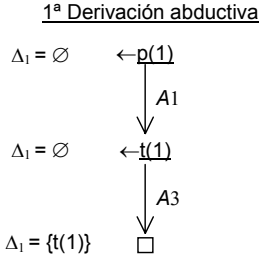
- Regla $C3$: Si L_j es abducible y $L_j^* \in \Delta_i$ entonces $F_{i+1} := F_i'$ y $\Delta_{i+1} := \Delta_i$.

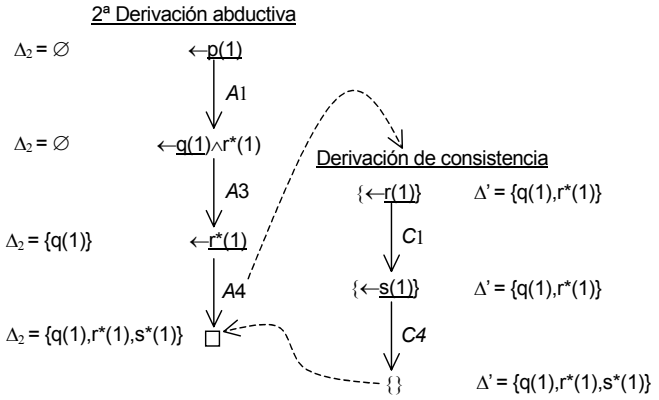
- Regla $C4$: Si L_j es abducible, básico, $L_j \notin \Delta_i$ y $L_j^* \notin \Delta_i$ entonces $F_{i+1} := F_i'$ y $\Delta_{i+1} := \Delta_i \cup \{L_j^*\}$.

- Regla $C5$: Si L_j es abducible, no es básico y existe una derivación abductiva desde $(\leftarrow L_j^*\ \Delta_i)$ hasta $(\square\ \Delta')$ entonces $F_{i+1} := F_i'$ y $\Delta_{i+1} := \Delta'$.

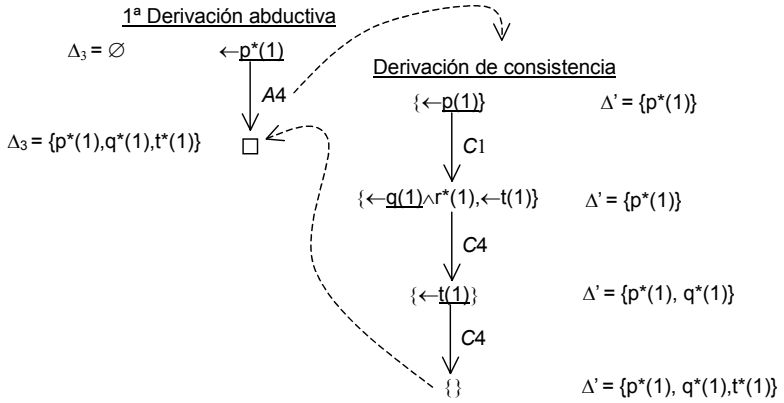
Al aplicar la regla $C1$ de la derivación de consistencia, la rama actual se divide en tantas ramas como resolventes de $\leftarrow L_1 \wedge \ldots \wedge L_k$ con cláusulas de BDI^* sobre L_j existen; si la cláusula vacía es uno de estos resolventes la comprobación de la consistencia falla. El caso de la regla $C2$ es similar al anterior pero se resuelve con una hipótesis. Al aplicar la regla $C3$ la rama es ya consistente y por lo tanto es eliminada. Las reglas $C4$ y $C5$ corresponden a casos de abducción de átomos que son básicos y que no son básicos respectivamente.

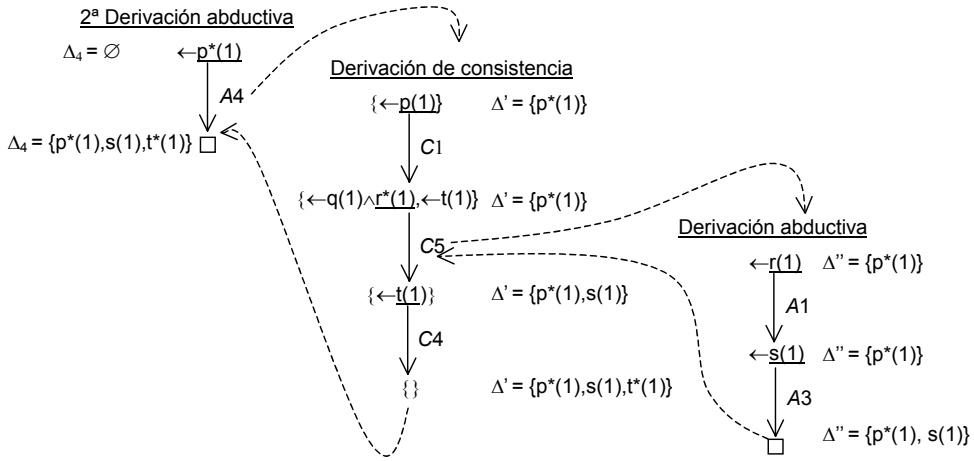
Ejemplo 9 *Sea la base de datos del ejemplo 8 y sea el requisito de actualización insertar*$(p(1))$*. Para ese requisito de actualización existen dos derivaciones abductivas que se muestran a continuación.*





Ejemplo 10 *Sea la base de datos del ejemplo 8 y sea el requisito de actualización* $borrar(p(1))$. *Para ese requisito de actualización existen de nuevo dos derivaciones abductivas que se muestran en las siguientes figuras.*





Paso B: Obtención de una transacción. Sea Δ un conjunto de hipótesis generadas por el procedimiento abductivo, T_Δ es una transacción asociada a Δ si dado un estado D, para cada átomo abducido básico $L \in \Delta$ (resp. $L^* \in \Delta$), $D' \vDash L$ (resp. $D' \vDash \neg L$) donde D' es el estado de base de datos obtenido al aplicar T_Δ a D.

En [33] no se presenta cómo llevar a cabo este paso en el que ya es necesario el acceso al conjunto de hechos. En el ejemplo siguiente se muestran las transacciones, obtenidas de forma intuitiva, que podrían resolver los requisitos de actualización de los ejemplos 9 y 10.

Ejemplo 11 *Supóngase que la base de datos del ejemplo 8 contiene sólo el hecho* $\{s(1)\}$. *Para el requisito de actualización* $insertar(p(1))$ *se habían obtenido dos conjuntos de hipótesis a partir de las cuales se podrían obtener dos transacciones:*

$$\Delta_1 = \{t(1)\} \qquad \Rightarrow T_{\Delta_1} = \{insertar(t(1))\}$$
$$\Delta_2 = \{q(1), r^*(1), s^*(1)\} \qquad \Rightarrow T_{\Delta_2} = \{insertar(q(1)), borrar(s(1))\}$$

Supóngase que ahora que la base de datos contiene los hechos $\{q(1), t(1)\}$. *Para el requisito de actualización* $borrar(p(1))$ *también se podrían construir dos transacciones a partir de los conjuntos de hipótesis obtenidos:*

$$\Delta_3 = \{p^*(1), q^*(1), t^*(1)\} \qquad \Rightarrow T_{\Delta_3} = \{borrar(q(1)), borrar(t(1))\}$$
$$\Delta_4 = \{p^*(1), s(1), t^*(1)\} \qquad \Rightarrow T_{\Delta_4} = \{insertar(s(1)), borrar(t(1))\}$$

Corrección y completitud.

La corrección y la completitud del método se enuncian en los siguientes términos:

- *Corrección*: Sea D una base de datos localmente estratificada, U un requisito $insertar(L)$ (resp. $borrar(L)$). Si hay una derivación abductiva desde $(\leftarrow L\ \{\})$ (resp. $(\leftarrow L^*\ \{\})$) hasta $(\square\ \Delta)$ entonces para cualquier parte extensional se cumple que $\vDash_{M_{D'}} L$ (resp. $\vDash_{M_{D'}} \neg L$) donde D' es el estado de base de datos obtenido al aplicar a D una transacción asociada a Δ y $M_{D'}$ es el modelo estable de D'.

- *Completitud*: Sea $D = BDI \cup BDE$ un estado de base de datos cuya parte intensional cumple las propiedades de ser acíclica y de no tener variables existencialmente cuantificadas[6], U un requisito $insertar(L)$ (resp. $borrar(L)$). Si existe una parte extensional BDE' tal que $\vDash_{M_{D'}} L$ (resp. $\vDash_{M_{D'}} \neg L$) donde $D' = BDI \cup BDE'$ y $M_{D'}$ es el modelo estable de D' entonces hay una derivación abductiva desde $(\leftarrow L\ \{\})$ (resp. $(\leftarrow L^*\ \{\})$) hasta $(\square\ \Delta)$ tal que $L' \in BDE'$ (resp. $L' \notin BDE'$) para cada átomo abducible básico $L' \in \Delta$ (resp. $L'^* \in \Delta$).

[6] En el artículo donde se presentó el método no era ésta la propiedad exigida sino otra que se llamaba *permitida* que no se correspondía con la propiedad comúnmente conocida con este término. Sin embargo uno de los autores nos comentó que lo que se quería controlar era que no hubiera variables "locales" (que a lo largo de la memoria se han llamado existencialmente cuantificadas) para evitar alcanzar un punto en las derivaciones en las que no se pudiera abducir ningún literal por no ser base. En esta memoria se ha preferido, por claridad y por entender que ésa era la intención de los autores, modificar el enunciado de corrección original por el que se ha presentado.

Resumen.

Las características más destacables de este método son las siguientes:

- Los requisitos de actualización sólo pueden ser inserciones o borrados de tuplas base sobre predicados derivados no admitiendo requisitos más generales.
- Obtiene las soluciones (conjunto Δ) en tiempo de ejecución pero sin acceder al conjunto de hechos.
- Debido a la regla de selección segura que utiliza el método, no se pueden elegir en las derivaciones las reglas deductivas con variables existencialmente cuantificadas por lo que no se aporta solución al problema de falta de valores.
- Debido al problema anterior, el método no puede tratar con reglas recursivas. En el caso de predicados derivados definidos recursivamente sólo se consideraría las reglas que definen el caso base encontrándose soluciones sólo en el caso de requisitos de inserción.
- En [33] se comenta que el método puede extenderse fácilmente para comprobar las restricciones de integridad durante el paso de obtención de las hipótesis pero en [57] se presenta un contraejemplo que muestra que algunas restricciones no son tenidas en cuenta.
- El problema de la información asumida es controlado por el método en la regla $A3$ de la derivación abductiva y en la regla $C4$ de la derivación de consistencia ya que en ellas la abducción de un literal se realiza tras comprobar que su complementario no pertenece ya al conjunto de hipótesis.

3.3 Método de Guessoum y Lloyd [28], [29]

Este método se basa en el procedimiento de resolución $SLDNF$[7].

Semántica asumida.

La semántica asumida es la de la compleción. La semántica operacional es la del procedimiento $SLDNF$ en el que se usa una definición más general de derivación $SLDNF$ ya que además de la derivación de éxito y la derivación de fallo se contempla el concepto de derivación *incompleta* en el sentido de que en cualquier punto se puede terminar la derivación al no seleccionarse ningún literal.

Base de Datos.

El método está definido para bases de datos localmente consistentes en llamada donde el conjunto de predicados básicos y el conjunto de predicados derivados pueden no ser disjuntos. Las restricciones de integridad se representan por fórmulas cerradas bien formadas.

[7] Extensión del procedimiento SLD con la regla de la Negación como Fallo.

Requisito de actualización.

Un requisito de actualización es de la forma $insertar(A)$ (resp. $borrar(A)$) donde A es un átomo. Si A es base representa una actualización simple; si no lo es representa una actualización múltiple.

Transacción generada.

Este método permite la inserción y el borrado de hechos y la inserción y el borrado de reglas deductivas. En el caso de la inserción de reglas deductivas, éstas sólo pueden ser de la forma $A \leftarrow$, donde A es un átomo que no es base (esto significa que se admite la inserción de reglas deductivas dependientes del dominio). Así pues la transacción obtenida es un conjunto de operaciones de la forma $insertar(C)$ donde C es un átomo que, si es base es un hecho y si no lo es representa una regla deductiva sin cuerpo, o de la forma $borrar(C)$ donde C es una sentencia de base de datos (hecho o regla deductiva).

Descripción del método.

El método de actualización se basa en los procedimientos *Borrado* e *Inserción* que utilizan a su vez los procedimientos básicos, *Borrado_Básico* e *Inserción_Básica,* que se llaman recursivamente entre sí. En estos procedimientos, que se presentan a continuación, se utiliza el concepto de árbol $SLDNF$ *trivial* que es aquél que sólo consta del nodo raíz.

```
ALGORITMO Borrado
ENTRADA
    D : Estado de base de datos;
    U = borrar(A) : Requisito de actualización de borrado;
    RI : Conjunto de restricciones de integridad;
SALIDA:
    τ : Conjunto de transacciones;
INICIO
SI comp(D) ⊨ ∃̄A
ENTONCES
| Borrado_Básico(D, A, τ0);
| τ := {T | T ∈ τ0, comp(T(D)) ⊭ ∃̄A y T(D) satisface RI}
FIN_SI
FIN.

ALGORITMO Borrado_Básico
ENTRADA
    D : Estado de base de datos;
    A : Átomo;
SALIDA:
    τ0 : Conjunto de transacciones;
INICIO
t := Árbol SLDNF finito que no sea trivial para D ∪ {← A};
τ0 := {[T1, ..., Tn] | existe un Ti (no necesariamente distinto) para
                       cada rama que no sea fallada de t, tal que
```

$T_i = borrar(C_i)$ donde C_i es una sentencia de D
utilizada como cláusula de entrada en una rama no fallada de t
o
$T_i \in \tau_i$ tal que $\neg B$ tiene éxito en una rama no fallada de t y τ_i es la salida de la llamada al procedimiento de $Inserción\_Básica$ con argumentos de entrada D y $B\}$

FIN.

ALGORITMO Inserción
ENTRADA
D : Estado de base de datos;
$U = insertar(A)$: Requisito de actualización de inserción;
RI : Conjunto de restricciones de integridad;
SALIDA
τ : Conjunto de transacciones;
INICIO
SI $comp(D) \not\models \overline{\exists} A$
ENTONCES
| $Inserción\_Básica(D, A, \tau_0)$;
| $\tau := \{T \mid T \in \tau_0, comp(T(D)) \models \overline{\exists} A$ y $T(D)$ satisface $RI\}$
FIN.

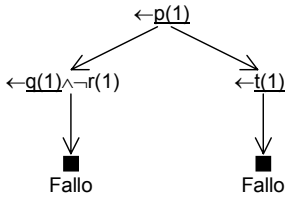
ALGORITMO Inserción_Básica
ENTRADA
D : Estado de base de datos;
A : Átomo;
SALIDA
τ_0 : Conjunto de transacciones;
INICIO
$t :=$ un árbol $SLDNF$ finito para $D \cup \{\leftarrow A\}$;
$\tau_0 := \{[T_1, \ldots, T_n] \mid \leftarrow L_1, \ldots, L_n$ es un objetivo en t tal que L_i
es base si es negativo y, o
$T_i = insertar(A_i)$ si $L_i = A_i$ (donde A_i es un átomo)
o
$T_i \in \tau_i$ si $L_i = \neg B_i$ (donde B_i es un átomo) y τ_i es la salida de la llamada al procedimiento $Borrado\_Básico$ con argumentos de entrada D y $B_i\}$
FIN.

En el siguiente ejemplo se muestra el funcionamiento de este método.

Ejemplo 12 *Sea D una base de datos que contiene los hechos $\{s(1)\}$ cuya parte intensional contiene el siguiente conjunto de reglas deductivas:*

$$1 : p(x) \leftarrow q(x) \wedge \neg r(x)$$
$$2 : p(x) \leftarrow t(x)$$
$$3 : r(x) \leftarrow s(x)$$

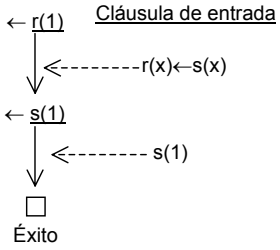
Y sea el requisito de actualización insertar$(p(1))$. El procedimiento Inserción llama al procedimiento Inserción_básica tras comprobar que comp$(D) \not\models p(1)$; supóngase que este procedimiento construye el árbol $SLDNF$ que se muestra a continuación:



A partir de los objetivos de este árbol el método construye las siguientes transacciones:

$$\begin{array}{lll} \leftarrow p(1) & \Rightarrow & T_1 = \{insertar(p(1))\} \\ \leftarrow t(1) & \Rightarrow & T_2 = \{insertar(t(1))\} \\ \leftarrow q(1) \wedge \neg r(1) & \Rightarrow & T_3 = \{insertar(q(1)), borrar(r(x) \leftarrow s(x))\} \\ \leftarrow q(1) \wedge \neg r(1) & \Rightarrow & T_4 = \{insertar(q(1)), borrar(s(1))\} \end{array}$$

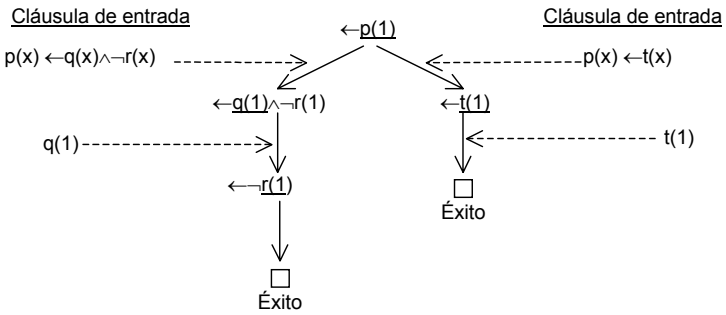
En las transacciones T_3 y T_4 la segunda operación se obtiene de la salida del procedimiento Borrado_básico con el átomo de entrada $r(1)$ que estudia el siguiente árbol $SLDNF$:



y que construye las transacciones:

$T_1' = \{borrar(r(x) \leftarrow s(x))\}$
$T_2' = \{borrar(s(1))\}$

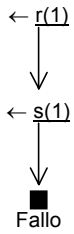
Ejemplo 13 *Sea de nuevo el conjunto de reglas deductivas del ejemplo anterior y supóngase que el conjunto de hechos que contiene la base de datos es $\{q(1), t(1)\}$ y sea el requisito de actualización borrar($p(1)$). El procedimiento Borrado llama al procedimiento Borrado_básico tras comprobar que $comp(D) \vDash p(1)$; supóngase que este procedimiento construye el árbol $SLDNF$ que se muestra a continuación:*



A partir de las ramas no-falladas de este árbol el método construye las siguientes transacciones:

$$T_1 = \{borrar(p(x) \leftarrow q(x) \wedge \neg r(x)), borrar(p(x) \leftarrow t(x))\}$$
$$T_2 = \{borrar(p(x) \leftarrow q(x) \wedge \neg r(x)), borrar(t(1))\}$$
$$T_3 = \{borrar(q(1)), borrar(p(x) \leftarrow t(x))\}$$
$$T_4 = \{borrar(q(1)), borrar(t(1))\}$$
$$T_5 = \{insertar(r(1)), borrar(p(x) \leftarrow t(x))\}$$
$$T_6 = \{insertar(r(1)), borrar(t(1))\}$$
$$T_7 = \{insertar(s(1)), borrar(p(x) \leftarrow t(x))\}$$
$$T_8 = \{insertar(s(1)), borrar(t(1))\}$$

En las cuatro últimas transacciones la primera operación se obtiene de la salida del procedimiento Inserción_básica con el átomo de entrada $r(1)$ que estudia el siguiente árbol SLDNF:



que construye las siguientes transacciones:

$$T'_1 = \{insertar(r(1))\}$$
$$T'_2 = \{insertar(s(1))\}$$

Corrección y completitud.

En el artículo donde se presenta el método sólo se dan resultados de corrección en los términos siguientes:

- *Corrección del procedimiento de inserción:* Sea D una base de datos normal localmente consistente en llamada, A un átomo y RI un conjunto de restricciones de integridad tal que $comp(D) \nvDash \overline{\exists} A$ y tal que D satisface RI. Sea T una transacción del conjunto τ que es la salida de la llamada al procedimiento *Inserción* con parámetros de entrada $(D, insertar(A), RI)$ entonces se cumple $comp(T(D)) \vDash \overline{\exists} A$.

- *Corrección del procedimiento de borrado:* Sea D una base de datos normal localmente consistente en llamada, A un átomo y RI un conjunto de restricciones de integridad tal que $comp(D) \vDash \overline{\exists} A$ y tal que D satisface RI. Sea T una transacción del conjunto τ que es la salida de la llamada al procedimiento *Borrado* con parámetros de entrada $(D, borrar(A), RI)$ entonces se cumple $comp(T(D)) \nvDash \overline{\exists} A$.

Resumen.

- Las soluciones obtenidas ante un requisito de actualización dependen del árbol $SLDNF$ obtenido en el primer paso del algoritmo *Inserción_básica* o del algoritmo *Borrado_básico* siendo necesario para obtener todas las soluciones la ejecución del mismo para cada uno de los posibles árboles

- El problema de la falta de valores no se resuelve con la elección de valores para las variables existencialmente cuantificadas sino al permitir la inserción de reglas deductivas sin cuerpo (que no cumplen, por lo tanto, la propiedad de ser independientes del dominio).

- La corrección de una transacción T obtenida para el requisito de actualización $insertar(A)$ (resp. $borrar(A)$) se comprueba en la última instrucción de los algoritmos *Inserción* (resp. *Borrado*) cuando verifica que $comp(T(D)) \models \overline{\exists} A$ (resp. $comp(T(D)) \not\models \overline{\exists} A$), también en este momento se comprueba si se satisfacen las restricciones de integridad. Esta verificación es muy costosa ya que en general se realizan tantos accesos a la base de datos extensional como accesos se hayan realizado en la generación de la solución.

- En el artículo se presenta una extensión del método propuesto que permite considerar bases de datos generales y requisitos de actualización representados por fórmulas bien formadas cualesquiera.

3.4 Método de Decker [19], [20]

Semántica asumida.

La semántica asumida es la de la compleción. La semántica operacional es la del procedimiento $SLDNF$.

Bases de datos.

Aunque el método se define para cualquier tipo de bases de datos advierte de que para resolver ciertos problemas es necesario exigir restricciones sintácticas.

Requisitos de actualización.

Un requisito de actualización tiene la forma $insertar(F)$ (resp. $borrar(F)$) donde F es una conjunción de literales.

Transacción generada.

Este método permite la inserción y el borrado de hechos y el borrado de reglas deductivas. Así pues la transacción obtenida es un conjunto de operaciones de la forma $insertar(A)$ donde A es un átomo base o de la forma $borrar(C)$ donde C es una sentencia de base de datos (hecho o regla deductiva).

Descripción del método.

El método se basa en los árboles $SLDNF$. Informalmente, la fórmula que representa el requisito de actualización de borrado (resp. de inserción) es la raíz de un árbol $SLDNF$ con al menos una refutación (resp. fallado finitamente). El problema es encontrar una transacción que modifique el estado de la base de datos de tal forma que en el nuevo estado haya una árbol fallado finitamente (resp. con al menos una refutación) cuya raíz sea el requisito de actualización. El nuevo estado deberá respetar las restricciones de integridad del esquema.

Para enunciar el método son necesarios algunos conceptos previos que se enuncian a continuación:.

- *Fallo inmediato de un literal:* Sea D un base de datos y L un literal, se dice que L falla inmediatamente si hay un árbol $SLDNF$ fallado finitamente para $D \cup \{\leftarrow L\}$ de profundidad 0.
- *Objetivo inmediato*[8]*:* un objetivo G de una derivación $SLDNF$ fallada es un objetivo inmediato si se cumplen las siguientes condiciones:
  - Cada literal positivo es básico,
  - Cada literal base falla inmediatamente; y
  - Cada literal negativo es base.
- *Literal seleccionable:* Sea D un base de datos y L un literal, se dice que L es seleccionable si L es bien positivo, bien negativo y base, y además existe una derivación $SLDNF$ finita para $D \cup \{\leftarrow L\}$.
- *Derivación justa*[9]*:* Una derivación $SLDNF$, $\aleph$, es justa si se cumple una de las siguientes condiciones:
  - $\aleph$ es una derivación no-fallada y para cada objetivo G de $\aleph$, cada literal seleccionable de G es seleccionado tras un número finito de pasos.
  - $\aleph$ es una derivación fallada y para cada objetivo G de $\aleph$ que contiene un literal seleccionable que nunca podría estar en un objetivo básico, entonces ese literal es seleccionado.

  Esta propiedad asegura que los literales derivados positivos y los literales seleccionables base que no fallan inmediatamente son elegidos en primer lugar.

  Un árbol $SLDNF$ es justo si cada derivación no-fallada en él es justa.
- *Subsumción de derivaciones:* Sea D una base de datos:
  - Sea $borrar(F)$ un requisito de actualización y sean $\aleph$, $\aleph'$ dos derivaciones $SLDNF$ no-falladas para $D \cup \{\leftarrow F\}$ entonces se dice que $\aleph$ subsume a $\aleph'$ si cada cláusula utilizada como cláusula de entrada en $\aleph'$ es utilizada también como cláusula de entrada en $\aleph$.

[8] *Base goal* en el artículo original. Se ha preferido sin embargo la expresión "objetivo inmediato" debido a que la característica que interesa destacar de esos objetivos es que se puede resolver en un sólo paso.

[9] En inglés, *fair derivation.*

  - Sea $insertar(F)$ un requisito de actualización y sean $\aleph$, $\aleph'$ dos derivaciones $SLDNF$ falladas para $D \cup \{\leftarrow F\}$ entonces se dice que $\aleph$ subsume a $\aleph'$ si para cada objetivo básico G de $\aleph'$ hay un objetivo en $\aleph$ que es una variante de G.

- *Árbol de inserción*: Sea D una base de datos e $insertar(F)$ un requisito de actualización. Un árbol de inserción t para $D \cup \{\leftarrow F\}$ tiene como raíz el objetivo $\leftarrow F$ y para cada objetivo G de t, los objetivos sucesores de G se obtienen como sigue:

  - Si G no es un objetivo inmediato, entonces se debe elegir un literal L de G de forma segura[10] y justa[11]; los objetivos sucesores de G se obtienen como en un árbol $SLDNF$ para $D \cup \{G\}$ donde se hubiera seleccionado L.
  - Si G es un objetivo inmediato, entonces para cada literal L que no sea base de G y para cada cláusula de D cuya cabeza unifique con L, G tiene un objetivo sucesor que se obtiene como en un árbol $SLDNF$ para $D \cup \{G\}$ donde se hubiera seleccionado L.
  - G no tiene otros objetivos sucesores.

Estrategia del método. El método propone estudiar las derivaciones $SLDNF$ para resolver los requisitos de actualización. Así:

- Dada una base de datos D y una fórmula F que es una conjunción de literales la solución al requisito de actualización $borrar(F)$ se obtiene haciendo fracasar cada derivación no-fallada $\aleph$ del árbol $SLDNF$ para $D \cup \{\leftarrow F\}$. Esto se puede conseguir de dos formas:

  - Borrando una cláusula utilizada como cláusula de entrada en $\aleph$, o
  - Resolviendo el requisito de actualización $insertar(A)$ donde A es un átomo tal que $\neg A$ ha sido utilizado como cláusula de entrada[12] en $\aleph$.

- Dada una base de datos D y una fórmula F que es una conjunción de literales la solución al requisito de actualización $insertar(F)$ se obtiene convirtiendo en refutación alguna derivación justa fallada $\aleph$ del árbol $SLDNF$ para $D \cup \{\leftarrow F\}$. Para ello si G es un objetivo inmediato de $\aleph$ entonces se deben realizar las dos siguientes tareas:

  - Insertar una instancia base de cada literal positivo de G; en caso de que los literales básicos no sean base, se debe elegir una valor para las variables, este valor podrá ser uno cualquiera ya utilizado en la derivación, uno proporcionado por el usuario o bien una constante de Skolem; y

[10] Una regla de selección es segura si no selecciona literales negativos que no sean base.

[11] Es decir que la derivación que se construye sea justa.

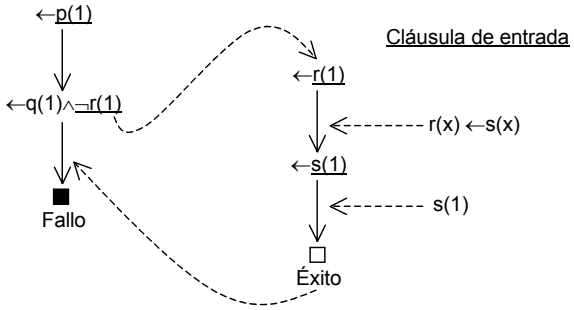
[12] Cuando en una derivación $SLDNF$ se selecciona un literal negativo base, $\neg A$, del objetivo $\leftarrow L_1 \wedge \ldots L_{i-1} \wedge \neg A \wedge L_{i+1} \wedge \ldots \wedge L_n$ entonces si existe un árbol fallado finitamente para $D \cup \{\leftarrow A\}$ el siguiente objetivo es $\leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \ldots \wedge L_n$. Por simplicidad en la presentación, se considerará que en este paso de la derivación, la cláusula de entrada ha sido $\neg A$.

– Resolver el requisito de actualización $borrar(A)$ donde A es un átomo que aparece negado en G.

Ejemplo 14 *Sea D una base de datos que contiene los hechos $\{s(1)\}$ cuya parte intensional contiene el siguiente conjunto de reglas deductivas:*

$1 : p(x) \leftarrow q(x) \wedge \neg r(x)$
$2 : p(x) \leftarrow t(x)$
$3 : r(x) \leftarrow s(x)$

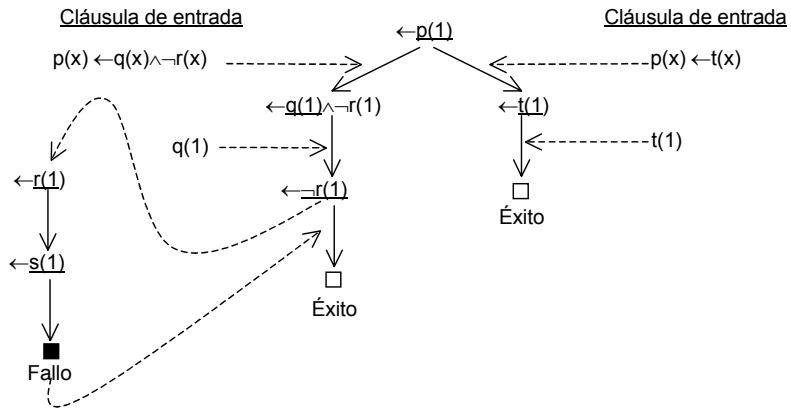
Y sea el requisito de actualización $insertar(p(1))$. La solución a este requisito podría obtenerse a partir de la siguiente derivación:



A partir del objetivo básico $\leftarrow q(1) \wedge \neg r(1)$ se obtendrían dos posibles soluciones

$T_1 = \{insertar(q(1))\ borrar(r(x) \leftarrow s(x))\}$;
$T_2 = \{insertar\ (q(1)),\ borrar(s(1))\}$

Ejemplo 15 *Sea el conjunto de reglas deductivas del ejemplo anterior y supóngase que el conjunto de hechos que contiene la base de datos es $\{q(1), t(1)\}$. Dado el requisito de actualización borrar($p(1)$), el árbol $SLDNF$ que contiene todas las refutaciones de $p(1)$ a partir del cual se obtendrán las soluciones es el siguiente:*



Las posibles soluciones al requisito serían:

$T_1 = \{borrar(p(x) \leftarrow q(x) \wedge \neg r(x)), borrar(p(x) \leftarrow t(x))\}$
$T_2 = \{borrar(p(x) \leftarrow q(x) \wedge \neg r(x)), borrar(t(1))\}$
$T_3 = \{borrar(q(1)), borrar(p(x) \leftarrow t(x))\}$
$T_4 = \{borrar(q(1)), borrar(t(1))\}$
$T_5 = \{insertar(s(1)),\ borrar(p(x) \leftarrow t(x))\}$
$T_6 = \{insertar(s(1)),\ borrar(t(1))\}$

El método enunciado en estos términos no encuentra soluciones en ciertos casos. Esencialmente hay dos motivos que pueden provocar esta situación. El primero es que se presentase el problema del *tropiezo*[13] en alguna derivación $SLDNF$; este problema puede evitarse si se restringe la aplicación del método a ciertas clases de bases de datos que aseguran que no se presenta ese problema (p.e. bases de datos de rango restringido). El segundo motivo para que no se encuentren soluciones es el problema de la no-terminación del método que puede aparecer al generarse infinitos requisitos de actualización subsidiarios.

Ejemplo 16 *Como ejemplo del primer problema sea una base de datos con la siguiente regla deductiva:*

$$1 : p(x) \leftarrow q(x) \wedge \neg r(x, y)$$

Y sea el requisito de actualización $insertar(p(1))$. *Cualquier derivación alcanzaría el objetivo* $\leftarrow q(1) \wedge \neg r(1, y)$ *en el que no se puede seleccionar ningún literal por lo que no se pueden obtener soluciones.*

Como ejemplo del segundo problema sea una base de datos con las reglas:

$$1 : p \leftarrow \neg q$$
$$2 : q \leftarrow \neg p \wedge r$$

Partiendo del requisito de actualización $insertar(p)$ *se generan infinitos requisitos de actualización subsidiarios.*

Otro problema que puede aparecer es que alguna solución obtenida no sea correcta. Esto puede darse por dos motivos:

- La presencia de negación puede invalidar alguna solución al no contemplar el problema de la información asumida tal como se mostró en el apartado 3.1.

- La presencia de restricciones de integridad puede invalidar alguna solución al proponerse cambios que conducen a un estado de la base de datos que no satisface las restricciones.

Debido a estos problemas hay que considerar que las soluciones obtenidas con este método son soluciones *posibles.* Una solución posible será una solución *válida* si y sólo se cumple que:

[13] En inglés *floundering.*

- La base de datos actualizada, D', satisface el requisito de actualización; para ello si el requisito era $insertar(F)$ (resp. $borrar(F)$) hay que comprobar que existe al menos una refutación (resp. un árbol $SLDNF$ fallado finitamente) para $D' \cup \{\leftarrow F\}$; y que

- La base de datos satisface todas las restricciones de integridad. En este caso se podría utilizar un método para la comprobación simplificada de la integridad o incluso integrar la comprobación con el requisito de actualización.

Corrección y completitud.

La corrección y la completitud del método se enuncian en los siguientes términos:

- *Corrección*: el artículo original no se enuncia este resultado, pero si se considera que las soluciones que devuelve el método son válidas entonces se puede afirmar que es correcto.

- *Completitud*: este resultado se enuncia en los dos teoremas que se presentan a continuación:

  - Completitud de los árboles $SLDNF$ para requisitos de actualización de borrado:

    "Sea D una base de datos, $borrar(F)$ un requisito de actualización y t un árbol $SLDNF$ justo para $D \cup \{\leftarrow F\}$ entonces para cada derivación $\aleph$ de $D \cup \{\leftarrow F\}$ hay una derivación en t que subsume a $\aleph$".

  - Completitud de los árboles de inserción para requisitos de actualización de inserción:

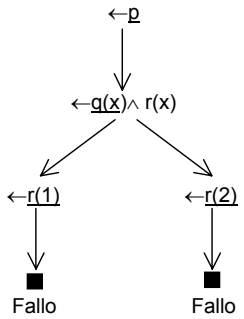
    "Sea D una base de datos, $insertar(F)$ un requisito de actualización y t un árbol de inserción para $D \cup \{\leftarrow F\}$ entonces para cada derivación fallada $\aleph$ de $D \cup \{\leftarrow F\}$ hay una derivación en t que subsume a $\aleph$".

Es importante darse cuenta de que el resultado de completitud en el caso de un requisito de inserción se enuncia para árboles de inserción y no simplemente para árboles $SLDNF$ normales. Esto es debido a que en un árbol $SLDNF$ fallado finitamente las derivaciones dependen de cómo se han elegido los literales por lo que podrían no encontrarse todas las soluciones. Este problema se ilustra en el ejemplo siguiente:

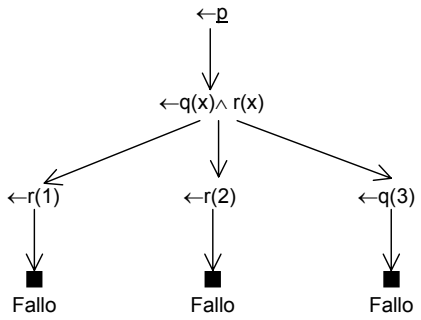
Ejemplo 17 *Sea D una base de datos que contiene los hechos $\{q(1), q(2), r(3)\}$ cuya parte intensional se define con la siguiente regla deductiva:*

$$1 : p \leftarrow q(x) \wedge r(x)$$

Y sea el requisito de actualización $insertar(p)$. Un árbol $SLDNF$ fallado para $D \cup \{\leftarrow p\}$ se muestra en la figura siguiente:



A partir de los objetivos de este árbol se podrían proponer como soluciones la inserción de $\{r(1)\}$ *o la inserción de* $\{r(2)\}$*. Es fácil observar que la solución insertar* $\{q(3)\}$ *no se puede obtener de ese árbol. Sin embargo si se construye el árbol de inserción, que se muestra más abajo, se pueden obtener todas las soluciones.*



Resumen.

Las características más destacables de este método son las siguientes:

- El método está enunciado para bases de datos de cualquier tipo aunque para asegurar que se encuentran soluciones y que el método termina es necesario exigir propiedades que aseguren que el procedimiento de resolución $SLDNF$ tiene ese comportamiento.

- Genera las transacciones en tiempo de ejecución.

- Para resolver el problema de *falta de valores* el método plantea la posible consulta al usuario o el uso de cualquier valor, además al ser completo podrían obtenerse todas las transacciones posibles.

- Para asegurar la consistencia el método propone convertir cada requisito de actualización F al requisito de actualización $F \wedge ri$ donde ri es un predicado derivado que no aparece en la base de datos y que se define por las reglas $ri \leftarrow W_1, \ldots, ri \leftarrow W_n$ donde $\{W_1, ..., W_n\}$ es el conjunto de restricciones de integridad del esquema.

- Las soluciones obtenidas por el método son las mismas que el método de Guessoum y Lloyd excepto que el conjunto de predicados básicos y derivados son disjuntos por lo que no se permite la inserción de reglas sin cuerpo ni la inserción de hechos sobre predicados derivados.

3.5 Método de Larson y Sheth [35]

La inclusión de este método en la memoria está justificado por el hecho de que el método para la restauración de la consistencia presentado en [16] (ver apartado 3.9) lo propone para extender el suyo permitiendo la inclusión de vistas en las bases de datos que considera. Por otra parte, el enfoque algebraico que propone resulta original.

Semántica asumida.

La semántica declarativa asumida es la del modelo mínimo de la base de datos, M_D.

Bases de datos.

Este método trabaja con bases de datos jerárquicas en las que un predicado derivado se define con una operación del álgebra relacional sobre predicados básicos o derivados. El método asume que se conoce el conjunto de atributos que constituyen el esquema de cada relación o vista (es decir de cada predicado).

Los operadores del álgebra relacional considerados son las siguientes:

- Diferencia: $r - s$ donde r y s son relaciones del mismo esquema;
- Unión: $r \cup s$ donde r y s son relaciones del mismo esquema;
- Intersección: $r \cap s$ donde r y s son relaciones del mismo esquema;
- Selección: r *donde* F siendo r una relación y F una condición definida sobre los atributos de r;
- Proyección: $r[A_{i_1}, \ldots, A_{i_n}]$: donde $A_{i_1}, \ldots, A_{i_n}$ es un subconjunto no vacío de los atributos de r;
- Concatenación: $r \bowtie s$ definida sobre los atributos del mismo nombre de r y s.

Ejemplo 18 *Sea D una base de datos que tiene el siguiente conjunto de reglas deductivas:*

$$1 : p(x) \leftarrow q(x) \wedge \neg r(x)$$
$$2 : p(x) \leftarrow t(x)$$
$$3 : r(x) \leftarrow s(x)$$

Si se supone que el esquema de todos los predicados es $\{A$: dom_$A\}$ *la definición de los predicados derivados p y r utilizando el álgebra relacional sería el siguiente:*

$$1 : p_1 \equiv q - r$$
$$2 : p \equiv p_1 \cup t$$

$3 : r \equiv s$[14]

donde p_1 *es un predicado auxiliar.*

Requisitos de actualización.

Un requisito de actualización es una fórmula cerrada de la forma $[\neg]p(\overrightarrow{c})$ donde p es un predicado derivado y $\vec{c}$ es un vector de constantes.

Transacción generada.

Este método resuelve los requisitos de actualización generando una transacción que sólo modifica la parte extensional de la base de datos. En esta transacción se pueden incluir operaciones de inserción, borrado y modificación de hechos aunque en este resumen sólo se consideren los dos primeros tipos de operaciones. La presencia de variables en las operaciones de borrado representa borrados múltiples.

Descripción del método.

La generación de las transacciones se va a iniciar en la fase de definición de datos, para ello, a cada predicado derivado se le intenta asociar una transacción de inserción y otra de borrado para cada situación que pueda darse. Estas situaciones se definen con las llamadas *condiciones de aplicabilidad* (C_{Apl}) que determinan en qué circunstancias se puede utilizar una transacción para resolver un requisito de actualización. Cuando, para una determinada condición, hay varias transacciones posibles, se dice que se ha presentado un *problema de ambigüedad semántica* (SAP[15]).

Para presentar cómo se generan estas transacciones sean:

- $D = BDE \cup BDI$ una base de datos donde BDI incluye la definición de los predicados derivados mediante una operación del álgebra relacional y BDE es el conjunto de hechos,
- q y s los nombre de dos predicados básicos ambos de esquema: (r_1: dom_r_1,..., r_n: dom_r_n),
- t y v los nombres de dos predicados básicos de esquemas respectivos: (r_1: dom_r_1, ..., r_n: dom_r_n, r_{n+1}: dom_r_{n+1}, ..., r_m: dom_r_m) y (r_1: dom_r_1, ..., r_n: dom_r_n, r_{n+m+1}: dom_r_{n+m+1}, ..., r_{n+m+p}: dom_r_{n+m+p}), y
- F una condición definida sobre los atributos del predicado q.

Sea p un predicado derivado y sea U un requisito de actualización de inserción (resp. borrado) de la forma $p(\overrightarrow{c})$ (resp. $\neg p(\overrightarrow{c})$); las reglas de traducción que obtienen las transacciones para satisfacer U dependiendo de qué operador del álgebra define p se presentan a continuación.

- Regla 1. Inserción en una vista definida con la diferencia: $p \equiv q - s$ y $U = p(\overrightarrow{c})$

[14] Siendo rigurosos, la definición del predicado r debería ser $s[A]$.
[15] Semantic Ambiguity Problem.

- C_{Apl}: $\models_{M_D} q(\overrightarrow{c}) \wedge \models_{M_D} s(\overrightarrow{c})$
  $T_{ins}^{-,1} := \{borrar(s(\overrightarrow{c}))\}$[16]
- C_{Apl}: $\not\models_{M_D} q(\overrightarrow{c}) \wedge \models_{M_D} s(\overrightarrow{c})$
  $T_{ins}^{-,2} := \{insertar(q(\overrightarrow{c})), borrar(s(\overrightarrow{c}))\}$
- C_{Apl}: $\not\models_{M_D} q(\overrightarrow{c}) \wedge \not\models_{M_D} s(\overrightarrow{c})$
  $T_{ins}^{-,3} := \{insertar(q(\overrightarrow{c}))\}$

- Regla 2. Borrado en una vista definida con la diferencia: $p \equiv q - s$ y $U = \neg p(\overrightarrow{c})$. Se presenta un problema de ambigüedad semántica al haber tres transacciones posibles para la misma condición de aplicabilidad (SAP_{-}) :

  - C_{Apl}: $\models_{M_D} q(\overrightarrow{c}) \wedge \not\models_{M_D} s(\overrightarrow{c})$
    $T_{bor}^{-,1} := \{insertar(s(\overrightarrow{c}))\}$
    $T_{bor}^{-,2} := \{borrar(q(\overrightarrow{c}))\}$
    $T_{bor}^{-,3} := \{borrar(q(\overrightarrow{c})), insertar(s(\overrightarrow{c}))\}$

- Regla 3. Inserción en una vista definida con la unión ($SAP_{\cup}$): $p \equiv q \cup s$ y $U = p(\overrightarrow{c})$

  - C_{Apl}: $\not\models_{M_D} q(\overrightarrow{c}) \wedge \not\models_{M_D} s(\overrightarrow{c})$
    $T_{ins}^{\cup,1} := \{insertar(q(\overrightarrow{c}))\}$
    $T_{ins}^{\cup,2} := \{insertar(s(\overrightarrow{c}))\}$
    $T_{ins}^{\cup,3} := \{insertar(q(\overrightarrow{c})), insertar(s(\overrightarrow{c}))\}$

- Regla 4. Borrado en una vista definida con la unión: $p \equiv q \cup s$ y $U = \neg p(\overrightarrow{c})$

  - C_{Apl}: $\models_{M_D} q(\overrightarrow{c}) \wedge \models_{M_D} s(\overrightarrow{c})$
    $T_{bor}^{\cup,1} := \{borrar(q(\overrightarrow{c})), borrar(s(\overrightarrow{c}))\}$
  - C_{Apl}: $\models_{M_D} q(\overrightarrow{c}) \wedge \not\models_{M_D} s(\overrightarrow{c})$
    $T_{bor}^{\cup,2} := \{borrar(q(\overrightarrow{c}))\}$
  - C_{Apl}: $\not\models_{M_D} q(\overrightarrow{c}) \wedge \models_{M_D} s(\overrightarrow{c})$
    $T_{bor}^{\cup,3} := \{borrar(s(\overrightarrow{c}))\}$

- Regla 5. Inserción en una vista definida con la intersección: $p \equiv q \cap s$ y $U = p(\overrightarrow{c})$

  - C_{Apl}: $\models_{M_D} q(\overrightarrow{c}) \wedge \not\models_{M_D} s(\overrightarrow{c})$
    $T_{ins}^{\cap,1} := \{insertar(s(\overrightarrow{c}))\}$
  - C_{Apl}: $\not\models_{M_D} q(\overrightarrow{c}) \wedge \models_{M_D} s(\overrightarrow{c})$
    $T_{ins}^{\cap,2} := \{insertar(q(\overrightarrow{c}))\}$
  - C_{Apl}: $\not\models_{M_D} q(\overrightarrow{c}) \wedge \not\models_{M_D} s(\overrightarrow{c})$
    $T_{ins}^{\cap,1} := \{insertar(q(\overrightarrow{c})), insertar(s(\overrightarrow{c}))\}$

[16] $T_{ins}^{-,i}$ (resp. $T_{bor}^{-,i}$) es la transacción *iésima* que inserta (resp. borra) una tupla en una vista definida como diferencia de dos relaciones básicas.

- Regla 6. Borrado en una vista definida con la intersección ($SAP_{\cap}$): $p \equiv q \cap s$ y $U = \neg p(\overrightarrow{c})$

  - C_{Apl}: $\models_{M_D} q(\overrightarrow{c}) \wedge \models_{M_D} s(\overrightarrow{c})$
    $T_{bor}^{\cap,1} := \{borrar(q(\overrightarrow{c}))\}$
    $T_{bor}^{\cap,2} := \{borrar(s(\overrightarrow{c}))\}$
    $T_{bor}^{\cap,3} := \{borrar(q(\overrightarrow{c})), borrar(s(\overrightarrow{c}))\}$

- Regla 7. Inserción en una vista definida con la selección: $p \equiv q$ *donde* F y $U = p(\overrightarrow{c})$

  - C_{Apl}: $\not\models_{M_D} q(\overrightarrow{c})$ y se cumple $F(\overrightarrow{c})$
    $T_{ins}^{donde,1} := \{insertar(q(\overrightarrow{c}))\}$

- Regla 8: Borrado en una vista definida con la selección: $p \equiv q$ *donde* F y $U = \neg p(\overrightarrow{c})$

  - C_{Apl}: $\models_{M_D} q(\overrightarrow{c})$ y se cumple $F(\overrightarrow{c})$
    $T_{bor}^{donde,1} := \{borrar(q(\overrightarrow{c}))\}$

- Regla 9[17]: Inserción en una vista definida con la proyección ($SAP_{[]}$)[18]: $p \equiv t[r_1, \ldots r_n]$ y $U = p(c_1, \ldots, c_n)$

  - C_{Apl}: $\not\models_{M_D} \overline{\exists} t(c_1, \ldots, c_n, x_{n+1}, \ldots, x_m) \wedge (c_{n+1}, \ldots, c_m)$ son valores proporcionados por el usuario o administrador de la base de datos:
    $T_{ins}^{[],1} := \{insertar(t(c_1, \ldots, c_n, c_{n+1}, \ldots, c_m))\}$

- Regla 10: Borrado en una vista definida con la proyección: $p \equiv t[r_1, \ldots r_n]$ y $U = \neg p(c_1, \ldots, c_n)$

  - C_{Apl}: $\models_{M_D} \overline{\exists} t(c_1, \ldots, c_n, x_{n+1}, \ldots, x_m)$:
    $T_{bor}^{[],1} := \{borrar(t(c_1, \ldots, c_n, x_{n+1}, \ldots, x_m))\}$

- Regla 11[19]: Inserción en una vista definida con la concatenación: $p \equiv t \bowtie v$ y $U = p(c_1, \ldots, c_n, c_{n+1}, \ldots, c_{n+m}, c_{n+m+1}, \ldots, c_{n+m+p})$

  - C_{Apl}: $\models_{M_D} t(c_1, \ldots, c_n, c_{n+1}, \ldots, c_{n+m}) \wedge$
    $\not\models_{M_D} v(c_1, \ldots, c_n, c_{n+m+1}, \ldots, c_{n+m+p})$:
    $T_{ins}^{\bowtie,1} := \{insertar(v(c_1, \ldots, c_n, c_{n+m+1}, \ldots, c_{n+m+p}))\}$

[17] La transacciones se han obtenido considerando que al proyectar se conservan todos los atributos que constituyen la clave principal de la relación básica.

[18] Aunque en este caso sólo hay una transacción posible, también se dice que hay un *SAP* debido a que se presenta el problema de falta de valores.

[19] En el artículo que presenta este método se consideran distintos casos en función de que los atributos sobre los que se concatena sean o no clave de las relaciones básicas. Caso de no serlo, las transacciones presentadas pueden tener efectos laterales no deseables como la inserción o borrado de más tuplas que la solicitada.

– C_{Apl}: $\nvDash_{M_D} t(c_1, \ldots, c_n, c_{n+1}, \ldots, c_{n+m}) \wedge$
$\vDash_{M_D} v(c_1, \ldots, c_n, c_{n+m+1}, \ldots, c_{n+m+p})$:
$T^{\bowtie,2}_{ins} := \{insertar(t(c_1, \ldots, c_n, c_{n+1}, \ldots, c_{n+p}))\}$

– C_{Apl}: $\nvDash_{M_D} t(c_1, \ldots, c_n, c_{n+1}, \ldots, c_{n+m}) \wedge$
$\nvDash_{M_D} v(c_1, \ldots, c_n, c_{n+m+1}, \ldots, c_{n+m+p})$:
$T^{\bowtie,3}_{ins} := \{insertar(t(c_1, \ldots, c_n, c_{n+1}, \ldots, c_{n+p})),$
$insertar(v(c_1, \ldots, c_n, c_{n+m+1}, \ldots, c_{n+m+p}))\}$

- Regla 12: Borrado en una vista definida con la concatenación ($SAP_{\bowtie}$): $p \equiv t \bowtie v$ y $U = \neg p(c_1, \ldots, c_n, c_{n+1}, \ldots, c_{n+m}, c_{n+m+1} \ldots, c_{n+m+p})$

– C_{Apl}: $\vDash_{M_D} t(c_1, \ldots, c_n, c_{n+1}, \ldots, c_m) \wedge$
$\vDash_{M_D} v(c_1, \ldots, c_n, c_{n+m+1}, \ldots, c_{n+m+p})$:
$T^{\bowtie,1}_{bor} := \{borrar(v(c_1, \ldots, c_n, c_{n+m+1}, \ldots, c_{n+m+p}))\}$
$T^{\bowtie,2}_{bor} := \{borrar(t(c_1, \ldots, c_n, c_{n+1}, \ldots, c_{n+p}))\}$
$T^{\bowtie,3}_{bor} := \{borrar(t(c_1, \ldots, c_n, c_{n+1}, \ldots, c_{n+p})),$
$borrar(v(c_1, \ldots, c_n, c_{n+m+1}, \ldots, c_{n+m+p}))\}$

Una vez establecido cómo resolver requisitos de actualización de predicados derivados definidos sobre predicados básicos, el algoritmo general tiene tres pasos algunos de los cuales se ejecutan en tiempo de definición de datos y otros en tiempo de ejecución:

- Paso 1: Definición de predicados derivados y resolución de ambigüedades. En este paso, que se realiza cuando se define una vista, se intenta asociar a la misma una transacción de inserción y otra de borrado; para ello se utilizan las restricciones semánticas que se conocen en tiempo de definición[20]. Si no es posible eliminar los SAP se interactúa con el administrador de la base de datos para que elija una entre las posibles. Si no se tiene conocimiento suficiente para realizar esta elección, entonces la ambigüedad se resolverá más adelante.

- Paso 2: Ejecución de una actualización. Este paso se realiza en tiempo de ejecución cuando un usuario solicita la actualización de una vista. El sistema evalúa las condiciones de aplicabilidad asociadas a la actualización. Si la transacción asociada es única entonces ésta se ejecuta. Obsérvese que si alguna de los predicados que definen el predicado derivado es a su vez derivado, este paso se ejecutará recursivamente.

- Paso 3: Resolución de la ambigüedad. Este paso se ejecuta cuando la ambigüedad no ha sido resuelta en el paso 1. La elección de la transacción que debe ejecutarse la va a realizar finalmente el usuario que solicitó la actualización de la vista.

[20] Estas restricciones pueden ser de varios tipos: dependencias de inclusión, dependencias de no inclusión, dependencias funcionales, restricciones de clave, y restricciones de cardinalidad. Para obtener más detalles de cómo las restricciones pueden invalidar alguna transacción consúltese [35].

Ejemplo 19 *Sea la base de datos definida en el ejemplo 18. Las transacciones de inserción y borrado para los tres predicados derivados deberían especificarse al definir el esquema. Al elegir las transacciones para el predicado r no se presenta ningún SAP ya que tiene el mismo esquema que el predicado s. Las transacciones, definidas en forma de algoritmo, serían las siguientes:*

ALGORITMO Insertar_$R(x)$
INICIO
SI $\not\models_{M_D} s(x)$
ENTONCES
| $insertar(s(x))$
FIN_SI
FIN.

ALGORITMO Borrar_$R(x)$
INICIO
SI $\models_{M_D} s(x)$
ENTONCES
| $borrar(s(x))$
FIN_SI
FIN.

Para el predicado derivado p_1 se presenta un SAP en la transacción de borrado que, supóngase, resuelve el administrador de la base de datos al elegir la primera de las tres opciones posibles ($T^{-,1}_{bor}$). Las transacciones serían entonces las siguientes:

ALGORITMO Insertar_$P_1(x)$
INICIO
SI $\models_{M_D} q(x) \wedge \models_{M_D} r(x)$
ENTONCES
| $Borrar\_R(x)$
SI NO
| SI $\not\models_{M_D} q(x) \wedge \models_{M_D} r(x)$
| ENTONCES
| | $insertar(q(x))$;
| | $Borrar\_R(x)$;
| SI NO
| | SI $\not\models_{M_D} q(x) \wedge \not\models_{M_D} r(x)$
| | ENTONCES
| | | $insertar(q(x))$
| | FIN_SI
| | FIN_SI
FIN_SI
FIN.

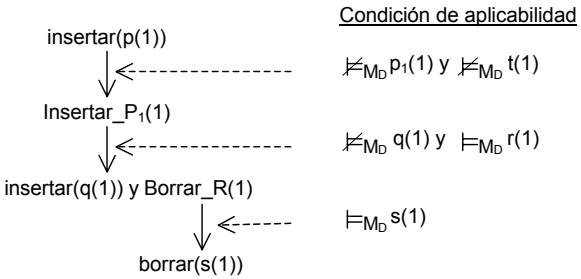
ALGORITMO Borrar_$P_1(x)$
INICIO
SI $\models_{M_D} q(x) \wedge \not\models_{M_D} r(x)$
ENTONCES
| $Insertar\_R(x)$
FIN_SI
FIN.

Por último, para el predicado derivado p se presenta un SAP en la transacción de inserción que, supóngase de nuevo, resuelve el administrador de la base de datos eligiendo la primera de las tres opciones posibles ($T_{ins}^{\cup,1}$). Las transacciones serían entonces las siguientes:

ALGORITMO Insertar_$P(x)$
INICIO
SI $\not\models_{M_D} p_1(x) \wedge \not\models_{M_D} t(x)$
ENTONCES
| $Insertar\_P_1(x)$
FIN_SI
FIN.

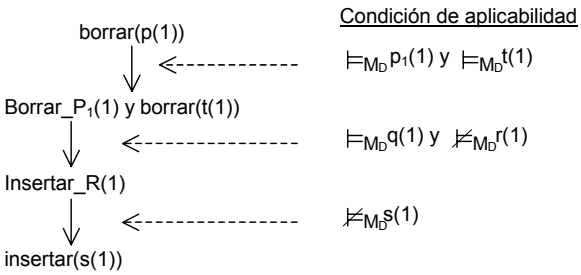
ALGORITMO Borrar_$P(x)$
INICIO
SI $\models_{M_D} p_1(x) \wedge \models_{M_D} t(x)$
ENTONCES
| $Borrar\_P_1(x)$;
| $borrar(t(x))$
SI NO
| SI $\not\models_{M_D} p_1(x) \wedge \models_{M_D} t(x)$
| ENTONCES
| | $borrar(t(x))$;
| SI NO
| | SI $\models_{M_D} p_1(x) \wedge \not\models_{M_D} t(x)$
| | ENTONCES
| | | $Borrar\_P_1(x)$
| | FIN_SI
| FIN_SI
FIN_SI
FIN.

Ejemplo 20 *Sea la base de datos del ejemplo 18 con las transacciones definidas en el ejemplo 19 y supóngase que contiene los hechos $\{s(1)\}$. Dado el requisito de actualización insertar$(p(1))$, a continuación se muestra cómo se resolvería este requisito:*



Así pues el requisito original se ha resuelto con operaciones insertar$(q(1))$ y borrar$(s(1))$.

Ejemplo 21 *Sea de nuevo la base de datos del ejemplo 18 y supóngase que contiene los hechos* $\{q(1), t(1)\}$. *El requisito de actualización* $borrar(p(1))$ *se resolvería como sigue:*



Así pues el requisito de actualización $borrar(p(1))$ *se resuelve con operaciones* $borrar(t(1))$ *e* $insertar(s(1))$.

Corrección y completitud del método.

En el artículo no se presentan estos resultados, pero tal como está enunciado el método, éste podría no ser correcto ya que no controla la información asumida.

Resumen.

Las características más destacables de este método son las siguientes:

- Genera las transacciones en tiempo de definición. La elección de la transacción adecuada en cada caso puede posponerse hasta que se ejecute una actualización.
- No permite la presencia de reglas recursivas.
- Para resolver el problema de *falta de valores* que se identifica en el $SAP_{[]}$ el método no especifica ninguna estrategia.
- El método no comenta cómo controlar el problema de la información asumida por lo que podría generar transacciones que no sean correctas.
- En este método las restricciones de integridad se utilizan para elegir transacción en caso de que haya varias posibles para satisfacer una actualización; no hace hincapié sin embargo en cómo integrar su método con la comprobación de las mismas.

3.6 Método de Wüthrich [65]

En el artículo se presenta una técnica para resolver el problema de las actualizaciones en una base de datos deductiva sin considerar la restricciones de integridad. Más tarde se extiende la propuesta inicial para tenerlas en cuenta.

Semántica asumida.

La semántica declarativa asumida en este método es la del modelo estándar [2], [15]. Con esta semántica, una restricción de integridad W se satisface en un estado de base de datos D si y sólo si $M_D \vDash W$ donde M_D es el modelo estándar de D.

Bases de datos.

El método trabaja con bases de datos permitidas y estratificadas aunque limita el esquema recursivo que se puede utilizar ya que sólo permite la definición de un predicado de forma recursiva con dos reglas similares a las siguientes:

$$p(x, y) \leftarrow s(x, y)$$
$$p(x, y) \leftarrow s(x, z) \wedge p(z, y)$$

Otra condición que se impone a las reglas deductivas es que en la cabeza no aparezcan ni constantes ni repeticiones de un mismo símbolo de variable. Esta limitación no quita generalidad al método ya que un esquema en el que no se cumple esta condición puede ser fácilmente modificado para que la cumpla.

Requisitos de actualización.

Un requisito de actualización, U, es una fórmula cerrada y permitida de la forma:

$$U = [\overline{\exists}]((\neg U_{1,1} \vee \ldots \vee \neg U_{1,m_1}) \wedge \ldots \wedge (\neg U_{n,1} \vee \ldots \vee \neg U_{n,m_n}))$$

donde se cumplen las siguientes condiciones:

- $n \geq 1$,
- $m_i \geq 1$ $(1 \leq i \leq n)$,
- U_{i,k_i} $(1 \leq i \leq n,\ 1 \leq k_i \leq m_i)$ es una fórmula con la estructura de un requisito de actualización.

Ejemplo 22 *Sea D una base de datos que tiene el siguiente conjunto de reglas deductivas:*

$$1 : p(x) \leftarrow q(x) \wedge \neg r(x)$$
$$2 : p(x) \leftarrow t(x)$$
$$3 : r(x) \leftarrow s(x)$$

Las siguientes expresiones constituyen requisitos de actualización:

$$U_1 = \exists x\ p(x)$$
$$U_2 = p(1)$$

Transacción generada.

Este método resuelve los requisitos de actualización generando, en tiempo de ejecución, una transacción formada por dos conjuntos de átomos, T_{ins} y T_{bor}, que representan los hechos que van a ser insertados y borrados respectivamente por la transacción.

Descripción del método.

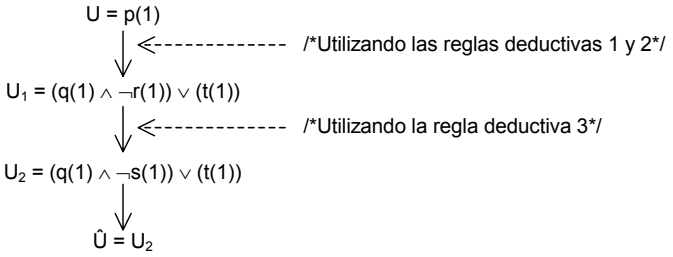
La generación de una solución para un requisito de actualización U en un estado de base de datos D se realiza en dos pasos:

1. Despliegue: este paso despliega los literales construidos con un predicado derivado que aparecen en U generando una fórmula, denotada por $\hat{U}$, que es lógicamente equivalente a U.
2. Generación de la transacción: en este paso se obtiene una transacción formada por dos conjuntos de átomos (T_{ins} y T_{bor}) tal que $\hat{U}$ es cierto en $(D \cup T_{ins}) - T_{bor}$.

A continuación se presentan estos dos pasos con detalle.

Despliegue del requisito de actualización. Este paso tiene como objetivo sustituir cada literal construido con un predicado derivado que aparece en el requisito de actualización U por la disyunción de los cuerpos de las reglas deductivas que definen ese predicado. Este paso es aplicado sucesivamente a las expresiones que se van obteniendo hasta que no aparezcan literales derivados.

Ejemplo 23 *Sea el requisito de actualización $U = p(1)$ planteado en el ejemplo 22. El despliegue sucesivo de U genera los siguientes requisitos:*



El despliegue de una actualización se consigue con el siguiente algoritmo:

```
ALGORITMO Desplegar
ENTRADA
    U:  Requisito de actualización;
    BDI:  Conjunto de reglas deductivas;
SALIDA
    Û:  requisito de actualización desplegado;
INICIO
MIENTRAS ∃ un átomo en U que unifique con la cabeza de
|                          alguna regla de BDI HACER
| Seleccionar un átomo A de U que unifique con la
|                          cabeza de alguna regla de BDI;
| R := ∅;
| PARA CADA regla deductiva refrescada r_i ∈ BDI (r_i = A_i ← C_i) tal
| |                              ∃mgu(A, A_i) HACER
| | SI A ocurre negativamente en U y r_i es una regla recursiva
| | ENTONCES
```

```
| | | Sustituir en el literal recursivo del cuerpo de r_i cada
| | |         ocurrencia de variable que no aparezca en la cabeza de r_i
| | |         por un nuevo símbolo de variable;
| | FIN_SI;
| | r_i^* := A_i ← ∃y_{i,1} ... ∃y_{i,m_i} Ci donde {y_{i,1}, ... y_{i,m_i}} son todas las variables
| |             que aparecen en C_i y no aparecen en A_i;
| | R := R ∪ {r_i^*};
| FIN_PARA;
| θ := mgu{A, A_1^*, ... , A_k^*} donde {A_1^*, ... , A_k^*} son las cabezas de todas
|               las reglas de R^*;
| Substituir en U la ocurrencia de A por la disyunción C_1^* ∨ ... ∨ C_k^*
|               donde {C_1^*, ... , C_k^*} son los cuerpos de todas
|               las reglas de R^*;
| U := Uθ;
FIN_MIENTRAS;
Û := U;
FIN.
```

Es fácil observar que la estructura de la fórmula $\hat{U}$ obtenida por el algoritmo *Desplegar* es la misma que la de un requisito de actualización. Por otra parte se puede demostrar la equivalencia lógica entre $\hat{U}$ y U.

En presencia de reglas recursivas, este algoritmo podría no terminar nunca, por ello el autor propone, informalmente, la siguiente regla de parada: "cada átomo que va a ser desplegado puede tener como máximo un átomo predecesor construido con el mismo predicado".

Generación de la transacción. El segundo paso busca una solución al requisito de actualización U generando una transacción T que genere un nuevo estado de base de datos en el que $\hat{U}$ sea cierta. Para obtener T el método propone los procedimientos mutuamente recursivos *Insertar* y *Borrar*. La tarea del procedimiento *Insertar* (resp. *Borrar*) es hacer cierta (resp. falsa) una fórmula. Estos dos procedimientos tienen cinco parámetros $(U, T_{ins}, T_{bor}, \Psi, Parar)$ donde:

- U es un requisito de actualización,
- T_{ins} y T_{bor} son conjuntos de hechos que representan respectivamente lo que se va a insertar y lo que se va a borrar para satisfacer el requisito de actualización,
- Ψ es un conjunto de fórmulas que o bien ya eran falsas o bien que se hacen falsas con el procedimiento de borrado. Estas fórmulas tiene la misma forma que un requisito de actualización,
- $Parar$ es una variable lógica que si toma el valor *cierto* indica que los procedimientos no han resuelto el requisito de entrada.

```
ALGORITMO Insertar
ENTRADA
    U:  Requisito de actualización;
    T_ins, T_bor:  Conjuntos de átomos;
    Ψ:  Conjunto de fórmulas;
SALIDA
```

```
    T_ins, T_bor:  Conjuntos de átomos;
    Parar:  Lógico;
INICIO
Parar := falso;
SI U = ¬U'
ENTONCES
| Borrar(U', T_ins, T_bor, Ψ, Parar)
SI NO /*U = [∃](∧_i ∨_{k_i} [¬]U_{i,k_i})  (1 ≤ i ≤ n, 1 ≤ k_i ≤ m_i)*/
|       /*El requisito es una conjunción de disyunciones*/
| Eliminar todos los cuantificadores existenciales que no estén
|           en el alcance de una negación;
| PARA cada i ELEGIR un k_i, sea k'_i /*Elegir un disyuntor de cada conjuntor*/
| | U = ∧_i[¬]U_{i,k'_i}  (1 ≤ i ≤ n)
| FIN_PARA;
| ELEGIR una substitución base para las variables libres de U, sea δ;
| U := Uδ;
| SI U es base
| ENTONCES
| | U := Forma_Normal_Disyuntiva(U)
| |       /*U es una disyunción de conjunciones de literales base*/
| | ELEGIR un disyuntor G de U
| | PARA cada átomo A en G HACER
| | |   SI A ∈ T_bor
| | |   ENTONCES Parar:=cierto
| | |   SI NO
| | |   | T_ins := T_ins ∪ {A}
| | |   FIN_SI
| | FIN_PARA;
| | PARA cada átomo negativo ¬A en G HACER
| | |   SI A ∈ T_ins
| | |   ENTONCES Parar:=cierto
| | |   SI NO
| | |   | T_bor := T_bor ∪ {A}
| | |   FIN_SI
| | FIN_PARA
| SI NO /*U no es base*/
| | i := 1;
| | MIENTRAS i ≤ n y NO Parar HACER
| | |    /*Cada conjuntor de U se va a resolver por separado*/
| | | G := [¬]U_{i,k'_i}
| | | (T_ins, T_bor, Parar) := Insertar(G, T_ins, T_bor, Ψ, Parar);
| | | i := i + 1
| | FIN_ MIENTRAS
| FIN_SI;
FIN_SI;
SI ∃ elemento f en Ψ|(BDE ∪ T_ins) − T_bor ⊨ ∃̄f
|    /*Una fórmula que era falsa, pasa a ser cierta con las
|        actualizaciones propuestas.  Ver el ejemplo 26*/
ENTONCES Parar := cierto
FIN_SI
FIN.
```

```
ALGORITMO Borrar
ENTRADA
    U: Requisito de actualización,
    T_ins, T_bor: Conjuntos de átomos,
    Ψ: Conjunto de fórmulas,
SALIDA
    T_ins, T_bor: Conjuntos de átomos,
    Parar: Lógico
INICIO
Parar:=falso;
SI U = ¬U'
ENTONCES
| Insertar(U', T_ins, T_bor, Ψ, Parar)
SI NO /*U = [∃](∧_i ∨_{k_i} [¬]U_{i,k_i}) (1 ≤ i ≤ n, 1 ≤ k_i ≤ m_i)*/
|        /*El requisito es una conjunción de disyunciones*/
| SI U es base
| ENTONCES
| | U := Forma_Normal_Conjuntiva(U);
| |    /*U es una conjunción de disyunciones de literales base*/
| | ELEGIR un conjuntor G de U;
| | PARA cada átomo A en G HACER
| | |   SI A ∈ T_ins
| | |   ENTONCES Parar:=cierto
| | |   SI NO
| | |   | T_bor := T_bor ∪ {A}
| | |   FIN_SI
| | FIN_PARA;
| | PARA cada átomo negativo ¬A en G HACER
| | |   SI A ∈ T_bor
| | |   ENTONCES Parar:=cierto
| | |   SI NO
| | |   | T_ins := T_ins ∪ {A}
| | |   FIN_SI
| | FIN_PARA
| SI NO /*U no es base*/
| | Eliminar todos los cuantificadores existenciales que no estén en el
| | |     alcance de una negación y tal que la fórmula obtenida es permitida;
| | (T*_ins, T*_bor) := (T_ins, T_bor);
| | Θ := {θ|(BDE ∪ T_ins) − T_bor ⊨ Uθ};
| |     /*En Θ están todas las instancias del requisito U
| |             que hay que borrar*/
| | PARA cada θ ∈ Θ HACER
| | | ELEGIR un conjuntor de U, sea ∨_{k_i'} [¬]U_{i',k_i'} (1 ≤ k_i' ≤ m_i');
| | | j := 1;
| | | MIENTRAS j ≤ m_i' y NO Parar HACER
| | | | G := ([¬]U_{i',j})θ;
| | | | (T_ins, T_bor, Parar) := Borrar(G, T_ins, T_bor, Ψ, Parar);
| | | | j := j + 1
| | | FIN_MIENTRAS
| | FIN_PARA;
| | SI (algún átomo de (T_ins − T*_ins) unifica con un átomo que aparezca en
```

```
| | |    un literal positivo de U) ∨ (algún átomo de (T_bor − T*_bor) unifica con
| | |    un átomo que aparezca en un literal negativo de U)
| | ENTONCES
| | |     /*Las actualizaciones propuestas pueden insertar lo que
| | |              se pretende borrar.*/
| | | SI (BDE ∪ T_ins) − T_bor ⊨ ∃U /*En efecto lo inserta*/
| | | ENTONCES
| | | | Parar := cierto
| | | FIN_SI
| | SI NO
| | | Ψ := Ψ ∪ {U}
| | FIN_SI;
| FIN_SI;
| SI hay un elemento f ∈ Ψ|(BDE ∪ T_ins) − T_bor ⊨ ∃̄f
| |     /*Una fórmula que era falsa, pasa a ser cierta con las
| |             actualizaciones propuestas.  Ver el ejemplo 27*/
| ENTONCES
| | Parar := cierto
| FIN_SI
FIN_SI
FIN.
```

Con estos dos procedimientos, la generación de la transacción que resuelve el requisito de actualización se consigue con la llamada $Insertar$(U,Ø,Ø,Ø, $falso$). Si el parámetro lógico que devuelve el procedimiento toma el valor *cierto* entonces significa que no se ha podido resolver el requisito de actualización con las elecciones realizadas. Si no es así, entonces se dice que los conjuntos (T_{ins}, T_{bor}) constituyen la *solución* de una *ejecución* de U en el estado de base de datos de entrada.

Algunos aspectos a destacar de los dos algoritmos anteriores son:

- La elección que debe hacerse en algunos puntos de los dos algoritmos anteriores (donde se especifique *elegir*) puede resolverse interrogando al usuario o bien es el propio sistema el que decide utilizando para ello criterios predefinidos.
- Si la elección realizada falla (es decir que la ejecución del algoritmo termina con *Parar* igual a *cierto*), entonces se debería intentar buscar otra solución cambiando alguna de las elecciones realizadas[21].

En los siguientes ejemplos se ilustra el funcionamiento de los algoritmos anteriores.

Ejemplo 24 *Sea una base de datos del ejemplo 22 y sea también el requisito de actualización:*

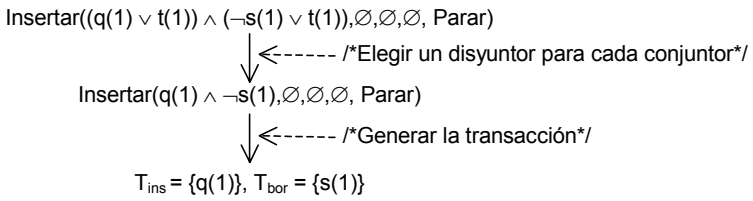
$$U = p(1).$$

El despliegue de U genera el siguiente requisito:

$$\hat{U} = (q(1) \vee t(1)) \wedge (\neg s(1) \vee t(1))$$

A continuación se incluye una traza simplificada de la ejecución del algoritmo Insertar para el requisito $\hat{U}$:

[21] Este *backtracking* no está contemplado en los algoritmos.

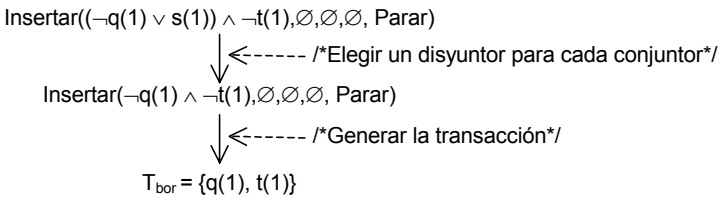


En este ejemplo se puede observar que la transacción que se genera va a depender de la elección de los disyuntores que se realice.

Ejemplo 25 *Sea de nuevo el conjunto de reglas deductivas del ejemplo anterior y sea el requisito de actualización:*

$$U = \neg p(1)).$$

A continuación se incluye una traza simplificada de la ejecución del algoritmo Insertar para la forma normal conjuntiva del requisito desplegado $\hat{U} = (\neg q(1) \vee s(1)) \wedge \neg t(1)$*:*



En dos ejemplos siguientes se ilustra el uso del conjunto Ψ que permite detectar soluciones incorrectas.

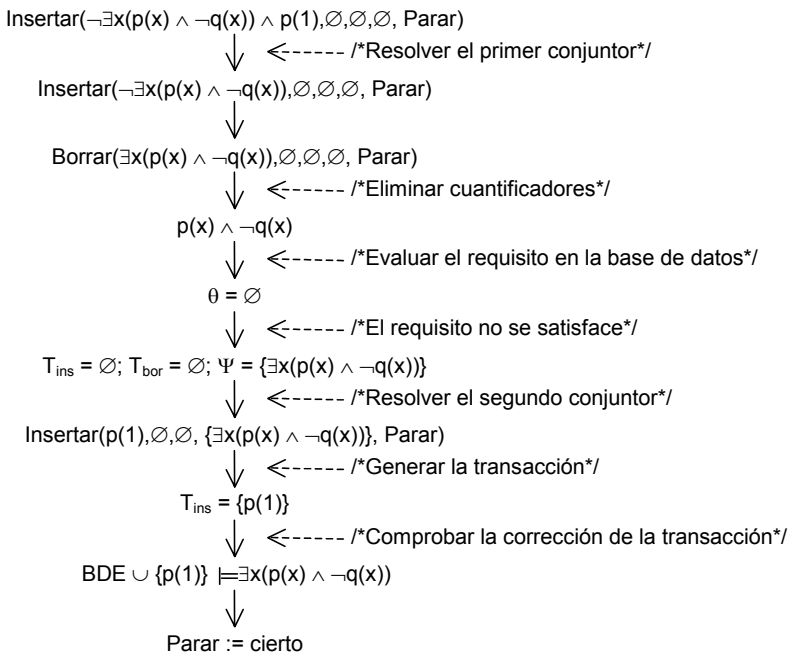
Ejemplo 26 *Sea una base de datos que no tiene reglas deductivas y cuyo conjunto de hechos es*

$$BDE = \{p(2), q(2)\}.$$

Y sea también el siguiente requisito de actualización:

$$U = \neg\exists x(p(x) \wedge \neg q(x)) \wedge p(1).$$

A continuación se incluye una traza simplificada de la ejecución del algoritmo Insertar para el requisito U que no es necesario desplegar ya que no contiene literales derivados:



Ejemplo 27 *Sea una base de datos con una regla deductiva:*

$$1 : p(x, y) \leftarrow q(x, z, w, y) \wedge r(z) \wedge \neg r(w)$$

Y el siguiente conjunto de hechos:

$$BDE = \{q(1, 2, 3, 4), q(1, 5, 2, 4), r(2), r(5)\}$$

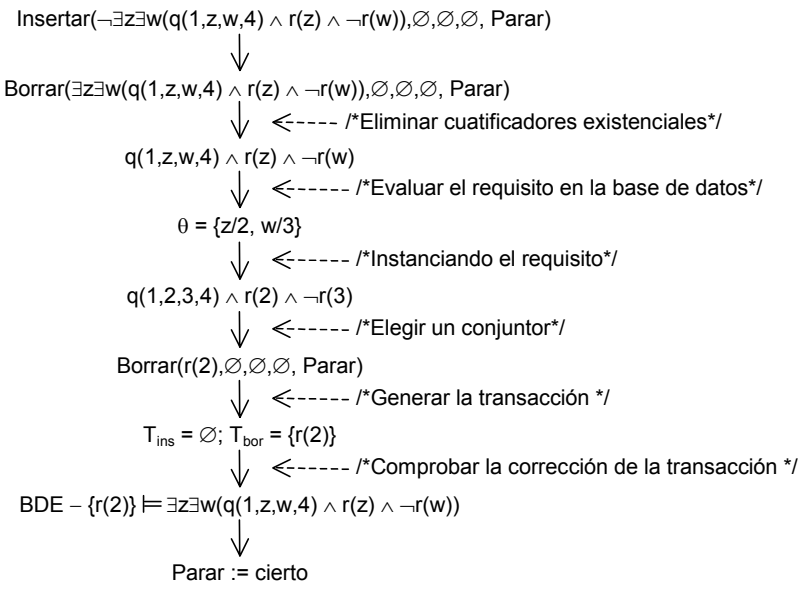
Y sea también el siguiente requisito de actualización:

$$U = \neg p(1, 4)$$

El despliegue de U genera el siguiente requisito:

$$\hat{U} = \neg\exists z \exists w(q(1, z, w, 4) \wedge r(z) \wedge \neg r(w))$$

A continuación se incluye una traza simplificada de la ejecución del algoritmo Insertar para el requisito $\hat{U}$:



Corrección y completitud.

Para presentar estos resultados primero se introduce el concepto de *realización*. Dados un estado de base de datos $D = BDE \cup BDI$ y una fórmula F, se dice que F es una realización de U en D si y sólo si se cumple:

1. F es permitida y contiene sólo predicados básicos, y
2. Para cada ejecución de F en D con solución (T_{ins}, T_{bor}), se cumple que $D' = (D \cup T_{ins}) - T_{bor}$ es consistente y que U es cierta en D'.

Teniendo en cuenta este concepto las propiedades de este método se pueden enunciar como sigue:

- *Corrección*: Sea D un estado de base de datos, U un requisito de actualización y $\hat{U}$ el despliegue de U entonces se cumple que $\hat{U}$ es una realización de U en D.
- *Completitud*: Sea D una base de datos definida, U una fórmula positiva y $\hat{U}$ el despliegue de U entonces U tiene alguna realización en D sólo si hay una ejecución de $\hat{U}$ en D.

Resumen.

Las características más destacables de este método son las siguientes:

- Genera las transacciones en tiempo de ejecución.
- En presencia de recursividad no siempre encuentra una solución para resolver un requisito con negación aunque ésta exista; por otra parte limita el esquema recursivo permitido.

- Para resolver el problema de *falta de valores* el método no especifica ninguna estrategia como se puede observar en el algoritmo de inserción en la instrucción que indica que se debe "elegir" una substitución para las variables libres del requisito sin comentar ningún criterio para ello.

- La actualización del conocimiento en presencia de restricciones de integridad puede resolverse con el método propuesto. Para ello si el conjunto de restricciones de integridad del esquema es $RI = \{W_1, \ldots, W_n\}$ y el requisito de actualización es U entonces el método se aplica a la fórmula $U \wedge W_1 \wedge \ldots \wedge W_n$ de forma que se controla que las restricciones de integridad se satisfagan tras las modificaciones propuestas. Evidentemente así usado, el método no aprovecha el hecho de que la base de datos es íntegra. Para mejorarlo, en el artículo se propone modificar el algoritmo *Inserción*.

- El problema de la información asumida es controlado por el método tanto en el algoritmo de inserción como en el de borrado cuando se comprueba que ningún átomo que se va a incluir en T_{ins} (resp. T_{bor}) pertenece ya a T_{bor} (resp. T_{ins}). Además también se controla cuando se comprueba si alguna fórmula del conjunto Ψ pasa a ser cierta con las modificaciones que se proponen.

3.7 Método de Teniente y Olivé [59]

Semántica asumida.

La semántica declarativa asumida es la de la compleción. La semántica operacional es una extensión del procedimiento $SLDNF$ que permite manejar todos los literales que pueden aparecer en una base de datos *aumentada* (este concepto se presenta en el apartado siguiente).

Bases de datos.

El método trabaja con bases de datos permitidas y estratificadas. El esquema de la base de datos se aumenta con las llamadas reglas de transición y reglas de eventos [44] que son reglas que definen la diferencia entre estados consecutivos. Para la especificación de estas reglas el lenguaje del esquema se extiende con tres nuevos símbolos de predicado por cada símbolo de predicado *viejo* (i.e. presente en el esquema original). Para un símbolo de predicado viejo p, estos predicados son los siguientes :

- *Predicado evento de inserción*: $\iota p(\overrightarrow{x})$ que representa, si p es básico, la inserción de hechos en p y si p es derivado la inserción inducida de instancias de $p(\overrightarrow{x})$.

- *Predicado evento de borrado*: $\delta p(\overrightarrow{x})$ que representa si p es básico, el borrado de hechos de p y si p es derivado el borrado inducido de instancias de $p(\overrightarrow{x})$.

- *Predicado nuevo*: p^n que denota al mismo predicado que p pero evaluado en el estado de base de datos obtenido tras la ejecución de una transacción.

Con estos nuevos predicados, el esquema de la base de datos se extiende con las siguientes reglas deductivas:

- Reglas de transición: definen la extensión de un predicado nuevo en términos del predicado viejo y de los eventos que hayan ocurrido en la transición de un estado a otro. Estas reglas son las siguientes:
  - $p^n(\overrightarrow{x}) \leftarrow p(\overrightarrow{x}) \wedge \neg\delta p(\overrightarrow{x})$
  - $p^n(\overrightarrow{x}) \leftarrow \iota p(\overrightarrow{x})$
- Reglas de eventos: que definen exactamente las inserciones y borrados de hechos de un predicado p inducidos por una transacción:
  - $\iota p(\overrightarrow{x}) \leftarrow p^n(\overrightarrow{x}) \wedge \neg p(\overrightarrow{x})$
  - $\delta p(\overrightarrow{x}) \leftarrow p(\overrightarrow{x}) \wedge \neg p^n(\overrightarrow{x})$

A la base de datos que incluye todas estas reglas se le denomina *base de datos aumentada* $A(D)$[22].

Dependiendo del predicado utilizado para construirlo, se distinguirán tres tipos de literales llamados *eventos, literales nuevos* y *literales viejos*. Se dice que un evento es básico (resp. derivado) si está construido con un predicado básico (resp. derivado).

Por otra parte, se consideran restricciones de integridad estáticas y de transición que se representarán en forma negada[23].

Ejemplo 28 *Sea una base de datos que tiene el siguiente conjunto de reglas deductivas:*

$$1 : p(x) \leftarrow q(x) \wedge \neg r(x)$$
$$2 : p(x) \leftarrow t(x)$$
$$3 : r(x) \leftarrow s(x)$$

La base de datos aumentada asociada a ese esquema se presenta a continuación. En ella se han introducido algunos predicados auxiliares que permiten simplificar el esquema:

$$1 : p_1(x) \leftarrow q(x) \wedge \neg r(x)$$
$$2 : p_2(x) \leftarrow t(x)$$
$$3 : r_1(x) \leftarrow s(x)$$
$$4 : p^n_1(x) \leftarrow p^n_{11}(x)$$
$$5 : p^n_1(x) \leftarrow p^n_{12}(x)$$
$$6 : p^n_1(x) \leftarrow p^n_{13}(x)$$
$$7 : p^n_1(x) \leftarrow p^n_{14}(x)$$
$$8 : p^n_2(x) \leftarrow p^n_{21}(x)$$
$$9 : p^n_2(x) \leftarrow p^n_{22}(x)$$
$$10 : p^n_{11}(x) \leftarrow q(x) \wedge \neg\delta q(x) \wedge \neg r(x) \wedge \neg\iota r(x)$$
$$11 : p^n_{12}(x) \leftarrow q(x) \wedge \neg\delta q(x) \wedge \delta r(x)$$
$$12 : p^n_{13}(x) \leftarrow \iota q(x) \wedge \neg r(x) \wedge \neg\iota r(x)$$
$$13 : p^n_{14}(x) \leftarrow \iota q(x) \wedge \delta r(x)$$

[22] En [57] se simplifican estas reglas para que el método sea más eficiente. En este resumen no se muestra esta simplificación por claridad.

[23] Ver la definición 6 en el apéndice C.

$14 : p^n_{21}(x) \leftarrow t(x) \wedge \neg\delta t(x)$
$15 : p^n_{22}(x) \leftarrow \iota t(x)$
$16 : r^n_1(x) \leftarrow r^n_{11}(x)$
$17 : r^n_1(x) \leftarrow r^n_{12}(x)$
$18 : r^n_{11}(x) \leftarrow s(x) \wedge \neg\delta s(x)$
$19 : r^n_{12}(x) \leftarrow \iota s(x)$
$20 : \iota p(x) \leftarrow p^n_{12}(x) \wedge \neg p_2(x)$
$21 : \iota p(x) \leftarrow p^n_{13}(x) \wedge \neg p_2(x)$
$22 : \iota p(x) \leftarrow p^n_{14}(x) \wedge \neg p_2(x)$
$23 : \iota p(x) \leftarrow p^n_{22}(x) \wedge \neg p_1(x)$
$24 : \iota r(x) \leftarrow r^n_{12}(x)$
$25 : \delta p(x) \leftarrow p_1(x) \wedge \neg p^n_1(x) \wedge \neg p^n_2(x)$
$26 : \delta p(x) \leftarrow p_2(x) \wedge \neg p^n_1(x) \wedge \neg p^n_2(x)$
$27 : \delta r(x) \leftarrow r_1(x) \wedge \neg r^n_1(x)$

Requisitos de actualización.

Un requisito de actualización es una conjunción de la forma $U = L_1 \wedge \ldots \wedge L_i \wedge \ldots \wedge L_n$ donde L_i $(1 \leq i \leq n)$ es un literal.

En el requisito inicial del usuario los literales suelen ser eventos positivos.

Transacción generada.

Este método resuelve un requisito de actualización generando una transacción que sólo modifica el conjunto de hechos de la base de datos. La transacción está formada por un conjunto de eventos básicos positivos base. Estos literales definen la modificación que hay que realizar para satisfacer el requisito de actualización, así siendo q un predicado básico:

- Si el evento $\iota q(\overrightarrow{c})$ pertenece a la transacción, el estado de base de datos debe modificarse añadiendo el hecho $q(\overrightarrow{c})$,
- Si el evento $\delta q(\overrightarrow{c})$ pertenece a la transacción, el estado de base de datos debe modificarse eliminando el hecho $q(\overrightarrow{c})$;

Descripción del método.

Para la obtención de la transacción que satisface un requisito de actualización, el método alterna dos actividades:

- Satisfacer el requisito incluyendo eventos positivos básicos base en la transacción; y
- Comprobar que las actualizaciones inducidas por estos eventos no son contradictorias con el requisito solicitado.

Estas dos actividades se realizan respectivamente durante las *derivaciones constructiva* y de *consistencia* que se presentan a continuación. En ellas se utiliza una regla de selección *segura* que es aquélla que nunca selecciona un literal negativo que no sea base.

Derivación constructiva. Una derivación constructiva desde$(G_1\ T_1\ C_1)$ hasta $(G_n\ T_n\ C_n)$ a través de una regla de selección segura R es una secuencia $(G_1\ T_1\ C_1)$, $(G_2\ T_2\ C_2), \ldots, (G_n\ T_n\ C_n)$ tal que $\forall i\ G_i$ tiene la forma $\leftarrow L_1 \wedge \ldots \wedge L_j \wedge \ldots \wedge L_k$, la regla R selecciona el literal L_j y $(G_{i+1}\ T_{i+1}\ C_{i+1})$ se obtiene con las siguientes reglas:

- Regla A_1: L_j es positivo y no es un evento básico.
  - Si S es el resolvente de G_i con alguna cláusula de $A(D)$ sobre el literal L_j entonces $G_{i+1} := S$; $T_{i+1} := T_i$; $C_{i+1} := C_i$.
- Regla A_2: L_j es un evento básico positivo base.
  - Si $L_j \in T_i$ entonces $G_{i+1} :=\leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k$, $T_{i+1} := T_i$, $C_{i+1} := C_i$.
- Regla A_3: L_j es un evento básico positivo base.
  - Si $L_j \notin T_i$ y se cumple que $L_j = \iota p(\overrightarrow{c})$ (resp. $L_j = \delta p(\overrightarrow{c})$) y $p(\overrightarrow{c})$ no es cierto (resp. $p(\overrightarrow{c})$ es cierto) en la base de datos, y si $C_i = \{\leftarrow Q_1, \ldots, \leftarrow Q_m, \ldots, \leftarrow Q_p\}$ y existen derivaciones de consistencia:
    desde $(\{\leftarrow Q_1\}\ T_i \cup \{L_j\}\ C_i)$ hasta $(\{\}\ T^1\ C^1)$,
    $\ldots$,
    desde $(\{\leftarrow Q_m\}\ T^{m-1}\ C^{m-1})$ hasta $(\{\}\ T^m\ C^m)$,
    $\ldots$,
    desde $(\{\leftarrow Q_p\}\ T^{p-1}\ C^{p-1})$ hasta $(\{\}\ T^p\ C^p)$

    entonces $G_{i+1} :=\leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k$, $T_{i+1} := T^p$, $C_{i+1} := C^p$. En caso de que $C_i = \emptyset$ entonces $G_{i+1} :=\leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k$, $T_{i+1} := T_i \cup \{L_j\}$, $C_{i+1} := C_i$.
- Regla A_4: L_j es un evento básico positivo que no es base.
  - Si $L_j = \iota p$ entonces elegir una substitución de las variables de L_j, sea σ, tal que $p\sigma$ no es cierto en la base de datos. Si $L_j = \delta p$ entonces sea σ, una substitución de las variables de L_j, tal que $p\sigma$ es cierto en la base de datos; y si $C_i = \{\leftarrow Q_1, \ldots, \leftarrow Q_m, \ldots, \leftarrow Q_p\}$ y existen derivaciones de consistencia:
    desde $(\{\leftarrow Q_1\}\ T_i \cup \{L_j\sigma\}\ C_i)$ hasta $(\{\}\ T^1\ C^1)$,
    $\ldots$,
    desde $(\{\leftarrow Q_m\}\ T^{m-1}\ C^{m-1})$ hasta $(\{\}\ T^m\ C^m)$,
    $\ldots$,
    desde $(\{\leftarrow Q_p\}\ T^{p-1}\ C^{p-1})$ hasta $(\{\}\ T^p\ C^p)$

    entonces $G_{i+1} := (\leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k)\sigma$, $T_{i+1} := T^p$, $C_{i+1} := C^p$. En caso de que $C_i = \emptyset$ entonces $G_{i+1} := (\leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k)\sigma$, $T_{i+1} := T_i \cup \{L_j\sigma\}$, $C_{i+1} := C_i$.
- Regla A_5: L_j es viejo y negativo.

  - Si el objetivo $\leftarrow L_j$ tiene éxito entonces $G_{i+1} :=\leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k$, $T_{i+1} := T_i$, $C_{i+1} := C_i$. $\leftarrow L_j$ tiene éxito si existe un árbol $SLDNF$ fallado finitamente para $A(D) \cup \{\leftarrow \neg L_j\}$.

- Regla A_6: L_j es un evento básico negativo base.

  - Si $\neg L_j \notin T_i$ entonces $G_{i+1} :=\leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k$, $T_{i+1} := T_i$, $C_{i+1} := C_i \cup \{\leftarrow \neg L_j\}$

- Regla A_7: L_j es negativo y no es viejo.

  - Si existe una derivación de consistencia desde $(\{\leftarrow \neg L_j\}\ T_i\ C_i)$ hasta $(\{\}\ T'\ C')$ entonces $G_{i+1} :=\leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k$, $T_{i+1} := T'$, $C_{i+1} := C'$.

Derivación de consistencia. Una derivación de consistencia desde $(F_1\ T_1\ C_1)$ hasta $(F_n\ T_n\ C_n)$ a través de una regla de selección segura R es una secuencia $(F_1\ T_1\ C_1), (F_2\ T_2\ C_2), \ldots, (F_n\ T_n\ C_n)$ tal que $\forall i\ F_i$ tiene la forma $\{\leftarrow L_1 \wedge \ldots \wedge L_j \wedge \ldots \wedge L_k\} \cup F'_i$, la regla R selecciona el literal L_j y $(F_{i+1}\ T_{i+1}\ C_{i+1})$ se obtiene con las reglas que se presentan a continuación.

- Regla B_1: L_j es positivo y no es un evento básico.

  - Si S' es el conjunto de todos los resolvente de F_i con alguna cláusula de $A(D)$ sobre el literal L_j y $\square \notin S'$entonces $F_{i+1} := S' \cup F'_i$, $T_{i+1} := T_i$, $C_{i+1} := C_i$.

- Regla B_2: L_j es positivo y no es un evento básico.

  - Si no hay cláusulas en $A(D)$ que puedan unificar con el literal L_j entonces $F_{i+1} := F'_i$, $T_{i+1} := T_i$, $C_{i+1} := C_i$.

- Regla B_3: L_j es un evento básico positivo base.

  - Si $L_j \in T_i$ y $k > 1$ entonces $F_{i+1} := \{\leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k\} \cup F'_i$, $T_{i+1} := T_i$, $C_{i+1} := C_i$.

- Regla B_4: L_j es un evento básico positivo base.

  - Si $L_j \notin T_i$ entonces $F_{i+1} := F'_i$, $T_{i+1} := T_i$, $C_{i+1} := C_i \cup \{F_i\}$.

- Regla B_5: L_j es evento básico positivo que no es base.

  - Si S' es el conjunto de todos los resolvente de F_i con alguna cláusula de T_i sobre el literal L_j y $\square \notin S'$ entonces $F_{i+1} := S' \cup F'_i$, $T_{i+1} := T_i$, $C_{i+1} := C_i \cup \{F_i\}$.

- Regla B_6: L_j es evento básico positivo que no es base.

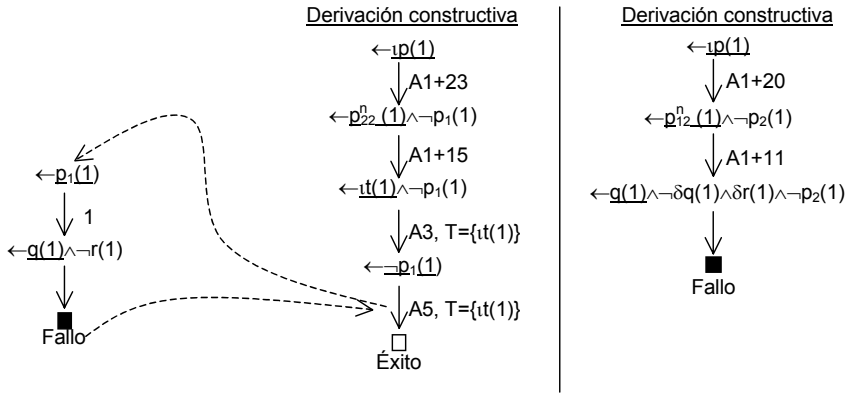
  - Si no hay hechos en T_i que puedan unificar con el literal L_j entonces $F_{i+1} := F'_i$, $T_{i+1} := T_i$, $C_{i+1} := C_i \cup \{F_i\}$.

- Regla B_7: L_j es viejo y negativo.
  - Si el objetivo $\leftarrow L_j$ tiene éxito y $k > 1$ entonces $F_{i+1} := \{\leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k\} \cup F'_i$, $T_{i+1} := T_i$, $C_{i+1} := C_i$. $\leftarrow L_j$ tiene éxito si existe un árbol $SLDNF$ fallado finitamente para $A(D) \cup \{\leftarrow \neg L_j\}$.
- Regla B_8: L_j es viejo y negativo.
  - Si el objetivo $\leftarrow \neg L_j$ tiene éxito entonces $F_{i+1} := F'_i$, $T_{i+1} := T_i$, $C_{i+1} := C_i$. $\leftarrow \neg L_j$ tiene éxito si existe una refutación $SLDNF$ para $A(D) \cup \{\leftarrow \neg L_j\}$.
- Regla B_9: L_j es un evento básico negativo base.
  - Si $\neg L_j \in T_i$ entonces $F_{i+1} := F'_i$, $T_{i+1} := T_i$, $C_{i+1} := C_i$.
- Regla B_{10}: L_j es un evento básico negativo base.
  - Si $\neg L_j \notin T_i$ y $k > 1$ entonces $F_{i+1} := \{\leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k\} \cup F'_i$, $T_{i+1} := T_i$, $C_{i+1} := C_i$.
- Regla B_{11}: L_j es un evento básico negativo base.
  - Si $\neg L_j \notin T_i$ y si existe una derivación constructiva desde $(\leftarrow \neg L_j\ T_i\ C_i)$ hasta $(\square\ T'\ C')$ entonces $F_{i+1} := F'_i$, $T_{i+1} := T'$, $C_{i+1} := C'$.
- Regla B_{12}:. L_j es un evento derivado negativo, o un literal nuevo negativo.
  - Si $k > 1$ y existe una derivación de consistencia desde $(\{\leftarrow \neg L_j\}\ T_i\ C_i)$ hasta $(\{\}\ T'\ C')$ entonces $F_{i+1} := \{\leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k\} \cup F'_i$, $T_{i+1} := T'$, $C_{i+1} := C'$.
- Regla B_{13}: L_j es un evento derivado negativo, o un literal nuevo negativo.
  - Si existe una derivación constructiva desde $(\leftarrow \neg L_j\ T_i\ C_i)$ hasta $(\square\ T'\ C')$ entonces $F_{i+1} := F'_i$, $T_{i+1} := T'$, $C_{i+1} := C'$.

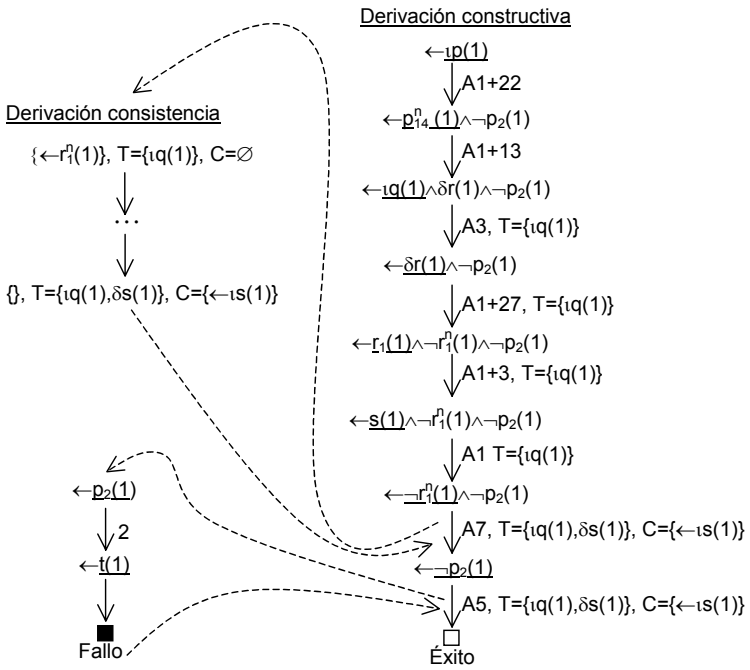
El conjunto C generado en estas derivaciones está formado por aquellos objetivos alcanzados durante las derivaciones cuya consistencia no se puede garantizar completamente con la transacción obtenida hasta ese momento. En general esto se producirá cuando el literal seleccionado sea un evento básico que no es base o cuando sea un evento básico que no pertenece a la transacción.

El método resuelve un requisito de actualización U utilizando estas dos derivaciones. Así si existe una derivación constructiva desde $(\leftarrow U\ \emptyset\ \emptyset)$ hasta $(\square\ T\ C)$ entonces T es una transacción que aplicada al estado de base de datos actual genera un estado de base de datos en la que se satisface U.

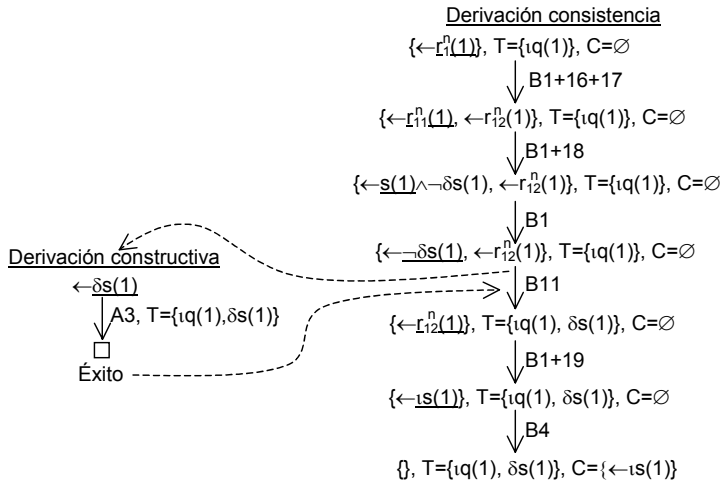
Ejemplo 29 *Sea la base de datos aumentada del ejemplo 28 y supóngase que contiene los hechos* $\{s(1)\}$. *A continuación se muestran varias derivaciones constructivas para resolver el requisito de actualización* $\iota p(1)$.



De la derivación constructiva de la izquierda se ha obtenido la solución $\iota t(1)$ *(es decir insertar el hecho* $t(1)$*). De la de la derecha no se obtiene solución ya que no termina con éxito. Por último, sea la siguiente derivación constructiva:*

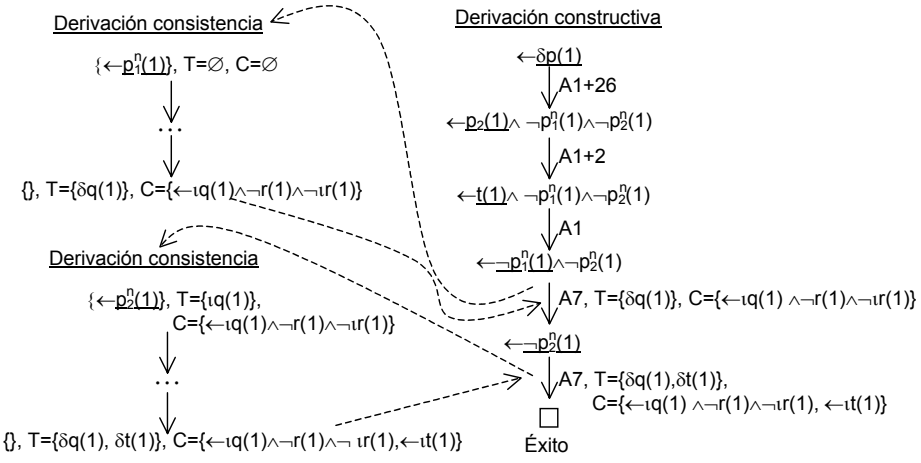


La derivación de consistencia que no se ha mostrado completa en la figura anterior es la siguiente:

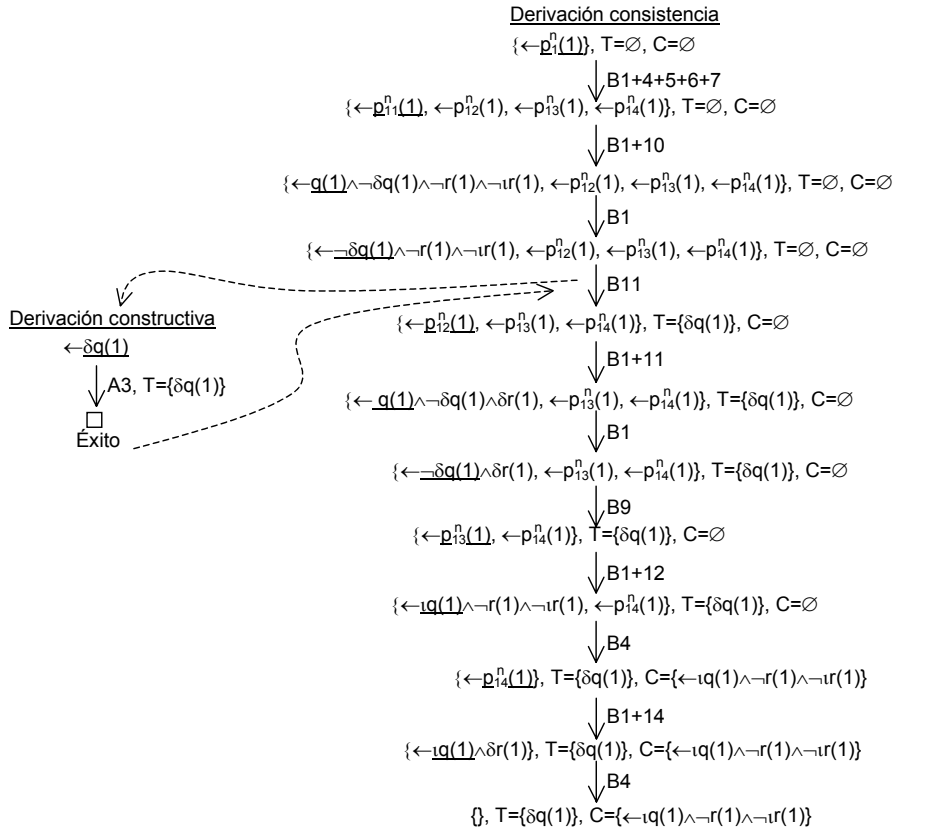


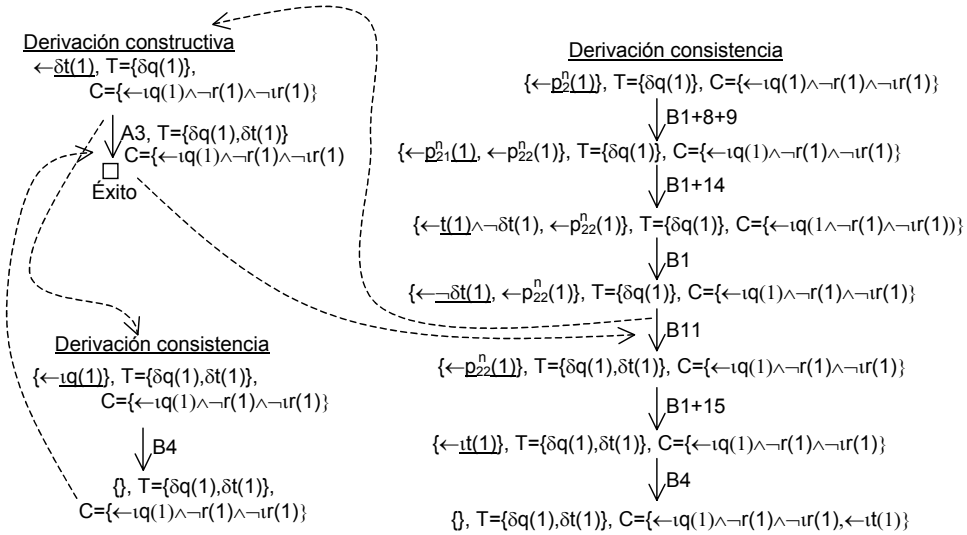
De esta derivación constructiva se obtiene la solución $\{\iota q(1), \delta s(1)\}$ *(es decir* $insertar(q(1))$ *y* $borrar(s(1))$*).*

Ejemplo 30 *Sea de nuevo la base de datos aumentada del ejemplo 28 con los hechos* $\{q(1), t(1)\}$ *Y sea el requisito de actualización* $\delta(p(1))$*.*



De esta derivación constructiva se obtiene la solución $\{\delta q(1), \delta t(1)\}$ *(es decir* $borrar(q(1))$ *y* $borrar(t(1))$*). Las dos derivaciones de consistencia que no se han presentado completas pueden verse en las siguientes figuras.*


Derivación consistencia
{←p1n(1)}, T=∅, C=∅
B1+4+5+6+7
{←p11n(1), ←p12n(1), ←p13n(1), ←p14n(1)}, T=∅, C=∅
B1+10
{←q(1)∧¬δq(1)∧¬r(1)∧¬ιr(1), ←p12n(1), ←p13n(1), ←p14n(1)}, T=∅, C=∅
B1
{←¬δq(1)∧¬r(1)∧¬ιr(1), ←p12n(1), ←p13n(1), ←p14n(1)}, T=∅, C=∅
B11
{←p12n(1), ←p13n(1), ←p14n(1)}, T={δq(1)}, C=∅
B1+11
{← q(1)∧¬δq(1)∧δr(1), ←p13n(1), ←p14n(1)}, T={δq(1)}, C=∅
B1
{←¬δq(1)∧δr(1), ←p13n(1), ←p14n(1)}, T={δq(1)}, C=∅
B9
{←p13n(1), ←p14n(1)}, T={δq(1)}, C=∅
B1+12
{←ιq(1)∧¬r(1)∧¬ιr(1), ←p14n(1)}, T={δq(1)}, C=∅
B4
{←p14n(1)}, T={δq(1)}, C={←ιq(1)∧¬r(1)∧¬ιr(1)}
B1+14
{←ιq(1)∧δr(1)}, T={δq(1)}, C={←ιq(1)∧¬r(1)∧¬ιr(1)}
B4
{}, T={δq(1)}, C={←ιq(1)∧¬r(1)∧¬ιr(1)}
Derivación constructiva
←δq(1)
A3, T={δq(1)}
Éxito


Derivación constructiva
←δt(1), T={δq(1)},
C={←ιq(1)∧¬r(1)∧¬ιr(1)}
A3, T={δq(1),δt(1)}
C={←ιq(1)∧¬r(1)∧¬ιr(1)
Éxito
Derivación consistencia
{←ιq(1)}, T={δq(1),δt(1)},
C={←ιq(1)∧¬r(1)∧¬ιr(1)}
B4
{}, T={δq(1),δt(1)},
C={←ιq(1)∧¬r(1)∧¬ιr(1)}
Derivación consistencia
{←p2n(1)}, T={δq(1)}, C={←ιq(1)∧¬r(1)∧¬ιr(1)}
B1+8+9
{←p21n(1), ←p22n(1)}, T={δq(1)}, C={←ιq(1)∧¬r(1)∧¬ιr(1)}
B1+14
{←t(1)∧¬δt(1), ←p22n(1)}, T={δq(1)}, C={←ιq(1∧¬r(1)∧¬ιr(1))}
B1
{←¬δt(1), ←p22n(1)}, T={δq(1)}, C={←ιq(1)∧¬r(1)∧¬ιr(1)}
B11
{←p22n(1)}, T={δq(1),δt(1)}, C={←ιq(1)∧¬r(1)∧¬ιr(1)}
B1+15
{←ιt(1)}, T={δq(1),δt(1)}, C={←ιq(1)∧¬r(1)∧¬ιr(1)}
B4
{}, T={δq(1),δt(1)}, C={←ιq(1)∧¬r(1)∧¬ιr(1),←ιt(1)}

Corrección y completitud.

Las propiedades de corrección y completitud de este método se pueden enunciar en los siguientes términos:

- *Corrección*: Sea D una base de datos deductiva, $A(D)$ la base de datos aumentada, U un requisito de actualización tal que $comp(A(D)) \nvDash U$ y sea T una transacción obtenida con el método descrito; entonces $comp(A(D')) \vDash U$ donde $D' = D \cup \{A|\iota A \in T\} - \{A|\delta A \in T\}$.
- *Completitud*: Sea D una base de datos estratificada, $A(D)$ la base de datos aumentada y U un requisito de actualización tal que $comp(A(D)) \nvDash U$. Entonces para toda transacción T tal que $comp(A(D')) \vDash U$ siendo $D' = D \cup \{A|\iota A \in T\} - \{A|\delta A \in T\}$ hay una derivación constructiva desde $(\leftarrow U\ \emptyset\ \emptyset)$ hasta $(\square\ T\ C)$.

Las demostraciones de estas propiedades se pueden encontrar en [57].

Resumen.

Las características más destacables de este método son las siguientes:

- Genera las transacciones en tiempo de ejecución.
- Trabaja con bases de datos estratificadas.
- La *falta de valores* no plantea problemas en este método ya que al ser completo encuentra todas las transacciones posibles.
- Con este método, la comprobación de la integridad en la base de datos actualizada podría realizarse de dos formas distintas:
  - Utilizando un método para la comprobación simplificada de la integridad que rechace las transacciones obtenidas que no generan un estado consistente de la base de datos. Dado que se trabaja con la base de datos aumentada un método adecuado sería el propuesto en [44].
  - Integrando la comprobación de la integridad con la obtención de la transacción. Para ello, un requisito de actualización de usuario U se extiende con el literal $\neg\iota inc$ donde ιinc es un predicado que se define por la regla deductiva $\iota inc \leftarrow \iota inc_1 \wedge \ldots \wedge \iota inc_n$ siendo n es el número de restricciones de integridad y donde inc_i son predicados utilizados para la representación de las restricciones de integridad en forma negada. En este caso, durante la derivación constructiva se controla que no se viole ninguna restricción de integridad.

3.8 Método de Pastor [45]

Semántica asumida.

La semántica declarativa asumida es la de la compleción. La semántica operacional es una extensión del procedimiento $SLDNF$.

Bases de datos.

El método trabaja con bases de datos permitidas y jerárquicas. Se consideran restricciones de integridad estáticas y de transición que se representarán en forma negada.

Al igual que en el método descrito en la sección 3.7, el esquema de base de datos se aumenta con las reglas de transición y las reglas de eventos que son reglas que definen la diferencia entre dos estados de bases de datos consecutivos. La principal diferencia con este método es que las soluciones se pueden obtener en tiempo de definición de la base de datos.

Requisitos de actualización.

En este método, un requisitos de actualización U es la conjunción de tres componentes:

- *Requisito de postcondición*: se representa mediante una fórmula de la forma: $[\neg]p^n(\overrightarrow{t})$ donde p^n es un predicado nuevo básico o derivado y $\overrightarrow{t}$ es un vector de términos[24]. Esta componente representa la modificación deseada por el usuario, con $p^n(\overrightarrow{t})$ (resp. $\neg p^n(\overrightarrow{t})$) se demanda la inserción (resp. el borrado) de hechos en el predicado p. En el caso de que la modificación deseada implique a varios predicados, el esquema de la base de datos se extenderá, momentáneamente, con una nueva regla que defina un predicado derivado que represente la modificación.

- *Requisito de aplicabilidad*: se representa mediante una fórmula de la forma: $[\neg]q_1(\overrightarrow{t_1}) \wedge \ldots \wedge [\neg]q_m(\overrightarrow{t_m})$ donde q_i $(1 \leq i \leq m)$ es un predicado viejo básico o derivado. Esta componente representa una condición que debe satisfacerse en el estado de base de datos original para que se resuelva la petición.

- *Operaciones prohibidas*: se representa mediante una fórmula: $[\neg]r_1(\overrightarrow{t_1}) \wedge \ldots \wedge [\neg]r_l(\overrightarrow{t_l})$ donde r_i $(1 \leq i \leq l)$ es un predicado evento básico o derivado. Esta componente representa aquellas operaciones que no deben ser propuestas para satisfacer la petición.

Transacciones generadas.

A partir de un requisito como el descrito, el método propuesto genera, en tiempo de definición, un programa cuya ejecución resuelve la petición inicial respetando, además de las condiciones impuestas por el usuario, las restricciones de integridad definidas en el esquema de la base de datos. Así pues, las transacciones generadas por el método están embebidas en un programa que incluye entre sus sentencias eventos positivos base que constituyen la transacción propuesta como solución. Estos programas, una vez generados podrán utilizarse, con parámetros actuales, para actualizar la base de datos.

Descripción del método.

El método obtiene un programa P a partir del esquema de la base de datos y de un requisito inicial, que refleja la intención del usuario. Este programa puede, en tiempo

[24] En este método por *término* se entiende un parámetro, una variable o una constante.

de ejecución, actualizar la base de datos mediante actualizaciones de los predicados básicos sin violar las restricciones de integridad.

El programa es generado con la aplicación de pasos de derivación secuenciales. El punto de partida es un triplete $(G_0\ C_0\ P_0)$ donde G_0 es el requisito de actualización del usuario, C_0 es el conjunto de condiciones de inconsistencia (inicialmente vacío) y P_0 el programa que debe ser generado. En cada paso, una regla de derivación refina P en programas más pequeños, $P_1, \ldots, P_n$, embebidos en las instrucciones de control adecuadas. Cada uno de estos programas es entonces el punto de partida de otros pasos hasta que se llegue a programas que consistan sólo de operaciones elementales ($insertar$, $borrar$, $fallar$, y $parar$).

Existen dos clases de pasos de derivación dependiendo de la naturaleza de la post-condición G_i. Si G_i es una condición que debe satisfacerse después de la ejecución de P_i entonces es un *paso de traducción* (el primer paso siempre es un paso de traducción). Si G_i es una condición que no debe satisfacerse después de la ejecución de P_i entonces es un *paso de reajuste*.

El conjunto de derivaciones puede verse como un árbol llamado *árbol de traducción*. En este árbol, la raíz, los nodos intermedios y las hojas tienen la forma $(G_i\ C_i\ P_i)$, cada rama es una derivación lineal, $(G_1\ C_1\ P_1)$, $(G_2\ C_2\ P_2)$,..., $(\{\}\ C_n\ P_n)$, construida a través de una regla de selección segura R, tal que $\forall i\ G_i$ tiene la forma $L_1 \wedge \ldots \wedge L_j \wedge \ldots L_k$, la regla R selecciona el literal L_j y $(G_{i+1}C_{i+1}P_{i+1})$ se obtiene con la aplicación de una de las reglas de traducción o de reajuste que por cuestiones de espacio no se comentan en este resumen. Para ilustrar, sin embargo, el método, a continuación se presenta un ejemplo sencillo en el que se puede observar el aspecto de los programas obtenidos por este método. En este programa se utilizan los predicados nuevos y de eventos propio de la base de datos aumentada.

Ejemplo 31 *Sea una base de datos con el siguiente conjunto de reglas deductivas:*

$$1 : p(x) \leftarrow q(x) \wedge \neg r(x)$$
$$2 : p(x) \leftarrow t(x)$$
$$3 : r(x) \leftarrow s(x)$$

Dado el requisito inicial: $p^n(x)$, el programa obtenido por el método se muestra a continuación (en él X es un parámetro):

```
TREK_TEXT ([p^n(X)])
IF p(X)
THEN parar
ELSE
| EITHER
| | IF q(X)
| | THEN
| | | δs(X)
| | ELSE
| | | IF ¬s(X)
| | | THEN
| | | | ιq(X)
| | | ELSE
| | | | δs(X);
| | | | ιq(X)
```

| | | END_IF
| | END_IF
| OR
| | $\iota t(X)$
| END_EITHER
END_IF
END OF TREK TEXT.

Con este programa se podría conseguir la inserción de instancias de $p(x)$. Si se considera ahora el requisito de actualización $\neg p^n(x)$ el programa que lo resuelve es el siguiente:

TREK_TEXT ([$\neg p^n(X)$])
IF $q(X) \wedge \neg r(X)$
THEN
| EITHER
| | $\delta q(X)$
| OR
| | $\iota s(X)$
| END_EITHER
END_IF;
IF $t(X)$
THEN
| $\delta t(X)$
END_IF
END OF TREK TEXT.

En el ejemplo anterior puede apreciarse que el programa contempla todas las posibles situaciones que pueden darse en la base de datos y en cada caso propone una actualización para llegar a un estado en el que se satisfaga el requisito inicial. Las reglas de derivación propuestas están definidas de forma que los programas que se generan nunca incluyen operaciones de actualización que puedan inducir las operaciones prohibidas del requisito inicial.

El método propuesto vigila las restricciones de integridad mediante la inclusión del literal $\neg\iota Inc$ en el conjunto de operaciones prohibidas para cada requisito. De esta forma en cada paso de derivación se comprueba que las operaciones de actualización propuestas en el programa nunca supongan la inserción de un camino de derivación para el átomo de inconsistencia.

Corrección y completitud.

En [45] se muestra cómo cada regla de derivación del método siempre deriva un programa correcto P. Es decir, el programa P se ha derivado de forma que su ejecución consigue el objetivo propuesto o falla. Cada regla de derivación deriva un programa correcto a partir de otro programa correcto producido por una regla de derivación.

Dado que el programa final es obtenido por la aplicación de una única regla de derivación, se concluye que este programa es correcto.

3.9 Método de Ceri, Fraternali, Paraboschi y Tanca [16]

Como ya se ha comentado en la introducción de este capítulo, además de definir métodos para la actualización segura de una base de datos, otra aproximación consiste en incorporar al sistema algún método para la restauración automática de la consistencia de la base de datos. En este apartado se presenta uno de estos métodos. El método tiene como objetivo restaurar una base de datos cuando como consecuencia de una actualización se ha llegado a un estado inconsistente; para ello genera, a partir del conjunto de restricciones de integridad, un conjunto de reglas de activas que pueden restaurar la consistencia de la base de datos. Estas reglas sólo constarán de condición y de acción. La restauración se realiza mediante las acciones de estas reglas que consisten en una operación de actualización (inserción, borrado o modificación) de un hecho. Dado el objetivo de estas reglas, a partir de ahora se les denominará *reglas restauradoras*.

Semántica asumida.

La semántica declarativa asumida es la del modelo mínimo de la base de datos, M_D, por lo que una restricción de integridad W se satisface en un estado de base de datos D si y sólo si $\vDash_D W$.

Bases de datos.

Las bases de datos consideradas por el método son bases de datos relacionales sin vistas. En el lenguaje de definición de la base de datos se distingue un tipo especial de predicados llamados *predicados restrictivos* que son aquéllos cuya extensión o su complementario es infinita si se aplica a un dominio infinito (p.e. los predicados de comparación).

Las restricciones de integridad se van a representar por fórmulas bien formadas en *forma conjuntiva estándar* que son fórmulas de la forma siguiente:

$$W = \forall \overrightarrow{x} \exists \overrightarrow{y} \neg(A_1 \wedge \ldots \wedge A_n \wedge \neg B_1 \wedge \ldots \wedge \neg B_m \wedge G)$$

donde se cumple:

- A_i $(1 \leq i \leq n)$ es un átomo construido con un predicado que no es restrictivo en el que todas las variables que aparecen están universalmente cuantificadas[25],
- B_j $(0 \leq j \leq m)$ es un átomo construido con un predicado que no es restrictivo,
- G es una fórmula que sólo contiene predicados restrictivos y en la que todas las variables que aparecen están universalmente cuantificadas,
- W es evaluable, para ello toda variable universalmente cuantificada debe aparecer al menos una vez en un átomo A_i $(1 \leq i \leq n)$.

[25] Así pues, la fórmula anterior es lógicamente equivalente a $\forall \overrightarrow{x}(A_1 \wedge \ldots \wedge A_n \rightarrow (\neg G \vee \exists \overrightarrow{y}(B_1 \vee \ldots \vee B_m)))$

Las restricciones de integridad con esta forma tienen la propiedad de que pueden restaurarse con una sola operación de actualización de la base de datos.

Asociado a cada restricción de integridad el método considera el *requisito de violación*. Este requisito es una consulta cuya evaluación en un estado de base de datos devuelve un conjunto de tuplas, llamado *conjunto de violación,* cada una de las cuales representa una violación de la restricción de integridad. Si la consulta no devuelve tuplas, entonces la restricción de integridad se satisface. El requisito de violación de una restricción de integridad ($W = \forall \overrightarrow{x} \exists \overrightarrow{y} \neg C$) se obtiene negando la fórmula que la representa ($\neg W = \neg \forall \overrightarrow{x} \exists \overrightarrow{y} \neg C \equiv \exists \overrightarrow{x} \overrightarrow{\forall} \overrightarrow{y} C$) y liberando todas las variables existencialmente cuantificadas. Dada una restricción de integridad $W = \forall \overrightarrow{x} \exists \overrightarrow{y} \neg C$ su requisito de violación W^* es $W^* = \forall \overrightarrow{y} C$ ya que:

$$
\begin{array}{l}
W = \forall \overrightarrow{x} \exists \overrightarrow{y} \neg C \\
\qquad \downarrow \qquad \text{/*Negando la restricción*/} \\
\neg W = \neg \forall \overrightarrow{x} \exists \overrightarrow{y} \neg C \equiv \exists \overrightarrow{x} \overrightarrow{\forall} \overrightarrow{y} C \\
\qquad \downarrow \qquad \text{/*Eliminando los cuantificadores*/} \\
W^* = \forall \overrightarrow{y} C
\end{array}
$$

Descripción del método.

Antes de presentar cómo se obtienen las reglas restauradoras, a continuación se introduce brevemente el lenguaje utilizado para definirlas así como su significado.

Sintaxis de las reglas restauradoras. La operaciones de actualización permitidas en estas reglas son $insertar(p(\overrightarrow{t}))$, $borrar(p(\overrightarrow{t}))$, $sustituir(p(\overrightarrow{t}), t_k \leftarrow t'_k))$ y *abortar* donde p es un predicado y $\overrightarrow{t}$ es un vector de términos $t_1, \ldots, t_n$ donde cada término t_i o t'_i es una variable, una constante o el símbolo especial "?" que denota el valor nulo o un valor elegido al azar. A partir de ahora, la expresión $O(\overrightarrow{y})$ denotará una operación cuyos términos variables se representan en el vector de variables $\overrightarrow{y}$. Si la operación no contiene términos variables se representará por $O(\overrightarrow{c})$.

Una regla restauradora es una expresión de la forma $F(\overrightarrow{x}) \rightarrow \{O_1(\overrightarrow{y_1}), \ldots, O_n(\overrightarrow{y_n})\}$ donde $F(\overrightarrow{x})$ es una fórmula evaluable que representa la condición de la regla y $\{O_1(\overrightarrow{y_1}), \ldots, O_n(\overrightarrow{y_n})\}$ es un conjunto operaciones de actualización de la base de datos que representa la acción de la regla donde se cumple que $\forall i \ (1 \leq i \leq n)$ $\overrightarrow{y_i}$ es un subconjunto de $\overrightarrow{x}$.

Semántica de las reglas reparadoras. La semántica de ejecución de las reglas se va a especificar operacionalmente, para ello es necesario establecer en primer lugar la semántica de cada operación elemental. Sea O una operación de actualización base. El significado de O es la transformación de un estado de base de datos D definido por un conjunto de hechos en otro estado D' ($D' = O(D)$) definido como sigue:

- Si $O = insertar(p(\overrightarrow{c}))$ entonces $D' = D \cup \{p(\overrightarrow{c})\}$
- Si $O = borrar(p(\overrightarrow{c}))$ entonces $D' = D - \{p(\overrightarrow{c})\}$
- Si $O = substituir(p(c_1, \ldots, c_n, c_k \leftarrow c'_k))$ entonces $D' = D - \{p(c_1, \ldots, c_n)\} \cup \{p(c_1, \ldots, c_{k-1}, c'_k, c_{k+1}, \ldots, c_n)\}$

- Si $O = abortar$ entonces D' es el estado de base de datos $D_{inicial}$ que había antes de la transacción del usuario.

La semántica operacional del conjunto de reglas se define en términos de transiciones de bases de datos. Dados un estado de base de datos D_i y un conjunto de reglas restauradoras $\Re$, una *transición* de estados es la aplicación del paso de producción ilustrado en el algoritmo *Procesar_Regla* a D_i para generar un nuevo estado de base de datos D_{i+1}. Una *computación* es el proceso de repetir la aplicación del paso de producción hasta que se alcance un estado en el que no haya posible evolución.

```
ALGORITMO Procesar_Regla
ENTRADA
    D_inicial:  Estado inicial de la base de datos;
    D_i:  Estado actual de la base de datos;
    ℜ:  Conjunto de reglas restauradoras;
SALIDA
    D_{i+1}:  Estado de base de datos;
INICIO
R_r := {Rα | R ∈ ℜ, R = F(x→) → {O_1(y_1→), ... , O_n(y_n→)} y α es una tupla obtenida
|        al evaluar F(x→) en D_i}
|        /*R_r es el conjunto de reglas relevantes*/
SI R_r = Ø /*El estado D_i que se ha alcanzado es íntegro*/
ENTONCES D_{i+1} := D_i;
SI NO
| Seleccionar una regla r = F(c→) → {O_1(c_1→), ... , O_n(c_n→)} de R_r;
| SI abortar ∈ {O_1(c_1→), ... , O_n(c_n→)} /*La restauración no es posible*/
| ENTONCES D_{i+1} := D_inicial
| SI NO
| | D_{i+1} := {O_1(c_1→), ... , O_n(c_n→)}(D_i)
| |      /*Las operaciones de la regla se aplican a D_i*/
| FIN_SI
FIN_SI
FIN.
```

Obtención de las reglas restauradoras. El método que se está estudiando genera reglas como las definidas anteriormente que contienen una única operación de actualización. En función de la operación incluida, se diferencian reglas restauradoras de inserción y de borrado, y reglas restauradoras de substitución. En los algoritmos que se presentan a continuación, que generan estas reglas, se hace uso de los siguientes conceptos:

- *Literales gemelos*: dos literales son gemelos si están construidos con el mismo predicado.

- *Variables de enlace*[26]: Las variables que aparecen en un literal sobre un predicado que no es restrictivo y otro predicado cualquiera se denominan variables de enlace.

[26] *Join-variable* en el artículo original.

```
ALGORITMO Generar_Reglas_Borrado
ENTRADA
    W = ∀x⃗∃y⃗¬(A1∧...∧An∧¬B1∧...∧¬Bm∧G)):  Restricción de integridad;
SALIDA
    ℜBor:  Conjunto de reglas restauradoras de borrado;
INICIO
ℜBor := ∅;
PARA CADA Ai de W (1 ≤ i ≤ n) HACER
| SI ¬∃Bj(1 ≤ j ≤ m) gemelo de Ai
| ENTONCES
| | ℜBor := ℜBor ∪ {∀y⃗(A1 ∧ ... ∧ An ∧ ¬B1 ∧ ... ∧ ¬Bm ∧ G) → borrar(Ai)}
| FIN_SI
FIN_PARA
FIN.
```

En este algoritmo al igual que en el de generación de reglas de inserción, la condición de que no haya literales gemelos de signo contrario controla la posibilidad de que una regla elimine una violación introduciendo otra como se muestra en el ejemplo 32.

Ejemplo 32 *Sea la restricción de integridad* $\forall x \forall y \exists z \neg(p(x,y) \wedge \neg p(y,z))$ *cuyo requisito de violación es* $\forall z(p(x,y) \wedge \neg p(y,z))$. *La regla restauradora de borrado generada a partir de ella sin tener en cuenta la limitación de los literales gemelos sería la siguiente:*

$$\Re_{Bor} = \{\forall z(p(x,y) \wedge \neg p(y,z)) \rightarrow borrar(p(x,y))\}$$

Esta regla no es considerada por el método ya que la eliminación de una violación puede introducir otras. Si se supone que la base de datos contiene los hechos $\{p(1,2), p(2,3)\}$ *entonces la condición de la regla se cumple para la substitución* $\{x/2, y/3\}$ *pudiendo eliminarse esta violación con la operación* $borrar(p(2,3))$. *Sin embargo esta operación ha introducido una nueva violación para los valores* $\{x/1, y/2\}$.

```
ALGORITMO Generar_Reglas_Inserción
ENTRADA
    W = ∀x⃗∃y⃗¬(A1∧...∧An∧¬B1∧...∧¬Bm∧G):  Restricción de integridad;
SALIDA
    ℜIns:  Conjunto de reglas restauradoras de inserción;
INICIO
ℜIns := ∅;
PARA CADA ¬Bj de W (1 ≤ j ≤ m) HACER
| SI ¬∃Ai(1 ≤ i ≤ n) gemelo de Bj
| ENTONCES
| | α:= Substitución para las variable de Bj que están
|             existencialmente cuantificadas en W ;
| ℜIns := ℜIns ∪ {∀y⃗(A1 ∧ ... ∧ An ∧ ¬B1 ∧ ... ∧ ¬Bm ∧ G)) → insertar(Bjα)}
| FIN_SI
FIN_PARA
FIN.
```

Para instanciar las variables existencialmente cuando sea necesario el método comenta varias estrategias como son: elegir el valor nulo, preguntar al usuario, definir una consulta que devuelva el valor o dejar que sea el sistema el que lo elija aleatoriamente.

Ejemplo 33 *Sea la restricción de integridad* $W = \forall x \exists y \neg(p(x) \wedge q(x) \wedge \neg r(x,y) \wedge \neg s(x,y))$ *cuyo requisito de violación es* $\forall y(p(x) \wedge q(x) \wedge \neg r(x,y) \wedge \neg s(x,y))$. *Para esta restricción se definirían las siguientes reglas de inserción y borrado:*

$$\begin{aligned}\Re_{Bor} = \{&\forall y(p(x) \wedge q(x) \wedge \neg r(x,y) \wedge \neg s(x,y)) \rightarrow borrar(p(x)),\\ &\forall y(p(x) \wedge q(x) \wedge \neg r(x,y) \wedge \neg s(x,y)) \rightarrow borrar(q(x))\}\\ \Re_{Ins} = \{&\forall y(p(x) \wedge q(x) \wedge \neg r(x,y) \wedge \neg s(x,y)) \rightarrow insertar(r(x,1)),\\ &\forall y(p(x) \wedge q(x) \wedge \neg r(x,y) \wedge \neg s(x,y)) \rightarrow insertar(s(x,2))\}\end{aligned}$$

En las reglas de inserción se han elegido las constantes 1 y 2 para instanciar las variables existencialmente cuantificadas.

Para la obtención de las reglas restauradoras de substitución, las operaciones se van a realizar sólo sobre variables de enlace:

ALGORITMO Generar_Reglas_Substitución
ENTRADA
    $W = \forall \overrightarrow{x} \exists \overrightarrow{y} \neg(A_1 \wedge \ldots \wedge A_n \wedge \neg B_1 \wedge \ldots \wedge \neg B_m \wedge G))$: Restricción de integridad;
SALIDA
    $\Re_{Sub}$: Conjunto de reglas restauradoras de substitución;
INICIO
$\Re_{Sub} := \emptyset$;
$W^* := \forall \overrightarrow{y}(A_1 \wedge \ldots \wedge A_n \wedge \neg B_1 \wedge \ldots \wedge \neg B_m \wedge G))$;
PARA CADA A_i de W $(1 \leq i \leq n)$ HACER
| PARA CADA Variable de enlace x que aparezca en A_i HACER
| | $\overrightarrow{x_i} :=$ Variables universalmente cuantificadas de A_i en W;
| | $W^{**} :=$Liberar $\overrightarrow{x_i}$ en W;
| | $W^{**} :=$ Eliminar A_i de W^{**};
| | $W^{**} :=$ Substituir x en W^{**} por una nueva variable z;
| | *Condición* $:= W^* \wedge W^{**}$;
| | SI *Condición* es evaluable
| | ENTONCES
| | | $\Re_{Sub} := \Re_{Sub} \cup \{Condición \rightarrow substituir(A_i, x \leftarrow z)\}$
| | FIN_SI;
| FIN_PARA
FIN_PARA
FIN.

Ejemplo 34 *Sea la restricción* $W = \forall x \exists y \neg(p(x) \wedge q(x) \wedge \neg r(x,y)))$ *cuyo requisito de violación es* $\forall y(p(x) \wedge q(x) \wedge \neg r(x,y))$. *Para esta restricción se definirían las siguientes reglas de substitución:*

$$\begin{aligned}\Re_{Sub} = \{&\forall y(p(x) \wedge q(x) \wedge \neg r(x,y)) \wedge \exists y \neg(q(z) \wedge \neg r(z,y))\\ &\rightarrow substituir(p(x), x \leftarrow z),\\ &\forall y(p(x) \wedge q(x) \wedge \neg r(x,y)) \wedge \exists y \neg(p(z) \wedge \neg r(z,y))\\ &\rightarrow substituir(q(x), x \leftarrow z)\}\end{aligned}$$

En el caso de que la condición obtenida no sea evaluable, el artículo propone extenderla con literales que permitan darle rango a la variable de enlace. Esta situación se ilustra informalmente en el ejemplo 35.

Ejemplo 35 *Sea la restricción de integridad* $W = \forall x \forall y \neg(p(x,y) \wedge y > 1)$ *cuyo requisito de violación es* $(p(x,y) \wedge y > 1)$. *Para esta restricción se definiría la siguiente regla de substitución utilizando la variable de enlace* y*:*

$$\Re_{Sub} = \{p(x,y) \wedge y > 1 \wedge \neg(z > 1) \rightarrow substituir(p(x,y), y \leftarrow z)\}$$

En esta regla, puede observarse que la condición no es evaluable para la variable z*. Las distintas soluciones a este problema se basan en añadir a la condición algún literal que dé rango a esa variable. Para ello podría utilizarse por ejemplo el propio predicado* p*, con lo que la regla quedaría:*

$$\Re_{Sub} = \{p(x,y) \wedge y > 1 \wedge p(x',z) \wedge \neg(z > 1) \rightarrow substituir(p(x,y), y \leftarrow z)\}$$

Finalmente, la obtención de todas las reglas restauradoras para un conjunto de restricciones de integridad se obtendría con el siguiente algoritmo:

```
ALGORITMO Generar_Reglas
ENTRADA
    RI : Conjunto de restricciones de integridad;
SALIDA
    ℜ : Conjunto de reglas restauradoras;
INICIO
ℜ := ∅;
PARA CADA W ∈ RI HACER
| ℜ := ℜ ∪ Generar_Reglas_Inserción(W);
| ℜ := ℜ ∪ Generar_Reglas_Borrado(W);
| ℜ := ℜ ∪ Generar_Reglas_Substitución(W);
FIN_PARA
FIN.
```

El método que se ha presentado podría extenderse con un método de actualizaciones, para permitir la presencia de vistas en la base de datos. En concreto en [16] se propone la extensión con el método [35] presentado en la sección 3.5.

Corrección y completitud.

La corrección del método propuesto se puede establecer a partir de la propiedad de *independencia de la base de datos* que se introduce a continuación:

"Una regla restauradora R de una restricción de integridad W es independiente del estado de base de datos si y sólo si para cada estado D tal que el conjunto de violaciones de W en D, $\surd_{D,W}$, no sea vacío, entonces se cumple $\surd_{D',W} \subset \surd_{D,W}$ donde D' es el estado de base de datos obtenido al aplicar la acción de R a D."

Si una regla restauradora de la restricción de integridad W es independiente del estado de base de datos, entonces cada vez que se ejecute su acción se elimina al menos una violación de W.

Se puede demostrar que las reglas de inserción y borrado antes presentadas son independientes de la base de datos.

A partir de esta propiedad se puede establecer el siguiente resultado:

- *Corrección*: Si cada restricción de integridad del esquema tiene al menos una regla restauradora independiente del estado de base de dato, y la ejecución de las reglas restauradoras empezando en un estado $D_{inicial}$ termina y genera un estado D_{final}, entonces D_{final} satisface todas las restricciones de integridad.
- *Completitud*: el método no es completo debido a la limitación de la generación de reglas en presencia de literales gemelos. Este problema podía eliminarse definiendo reglas cuya acción fuera *abortar* para aquellas restricciones sin reglas restauradoras independientes del estado de base de datos.

Comentarios.

El método tal y como se ha planteado tiene una limitación importante ya que no toda restricción de integridad puede ser expresada mediante una fórmula en forma conjuntiva estándar.

Ejemplo 36 *La restricción de integridad representada por la fórmula* $\forall x(p(x) \rightarrow \exists y (r(x,y) \wedge s(x,y))$ *no puede expresarse en forma conjuntiva estándar.*

Para resolver este hecho se propone permitir la presencia de vistas en la base de datos.

Ejemplo 37 *Si en la base de datos del ejemplo 36 se define la vista* $q(x) \leftarrow r(x,y) \wedge s(x,y)$ *entonces la restricción de integridad se puede escribir como* $\forall x \neg(p(x) \wedge \neg q(x))$ *que está en forma conjuntiva estándar. La regla restauradora de borrado asociado a esa restricción sería* $p(x) \wedge \neg q(x) \rightarrow borrar(q(x))$.

Si se considera esta solución, el método debería incorporar una estrategia para la actualización de vistas.

Otra característica importante es la ausencia de evento en las reglas restauradoras. Este hecho implica que ante una actualización de la base de datos no se sabrá qué reglas hay que procesar lo que obliga a evaluar la condición de todas ellas con la evidente carga que esto supone para el sistema.

3.10 Actualización y mantenimiento de la integridad

Ya que el objetivo de este trabajo es la definición de un método de restauración de la consistencia en caso de violación y dado que ésta sólo se produce por una actualización de usuario, es interesante destacar de qué forma los métodos de actualizaciones contemplan el mantenimiento de la consistencia.

- El método de Kakas y Mancarella se puede extender, según sus autores, para comprobar y restaurar la integridad de la base de datos mientras se generan soluciones al requisito de actualización. Para ello en el conjunto RI^* del marco abductivo se incluirían las restricciones de integridad en forma negada.
- El método de Guessoum y Lloyd rechaza cualquier transacción obtenida que no satisfaga las restricciones de integridad pero no propone ninguna restauración.

- En el método de Decker se puede extender el requisito de actualización con las restricciones de forma que la solución obtenida es correcta respecto al requisito y además genera un estado de base de datos consistente con las restricciones de integridad. En la solución propuesta se incluirán operaciones restauradoras si es necesario.
- El método de Larson y Sheth no se plantea el problema de la consistencia.
- El método de Wüthrich plantea, en la misma línea que el de Decker, que el requisito inicial se puede extender con las restricciones de integridad de forma que la transacción obtenida no sólo resuelve el requisito de actualización sino que también restaura si es necesario.
- El método de Teniente y Olivé estudia las dos alternativas, integra la restauración con la actualización o rechazar las soluciones que no generen un estado consistente.
- El método de Pastor restaura la posibles inconsistencias generadas al resolver el requisito de actualización.

Se puede concluir pues que los métodos estudiados adoptan tres posiciones frente al problema de la consistencia:

1. No considerar las restricciones de integridad (ignorar).
2. Comprobar las restricciones después del proceso de actualización (comprobar).
3. Integrar en el proceso de actualización la restauración de las restricciones violadas (restaurar).

Capítulo 4

Un método para la obtención de reglas restauradoras de la consistencia en bases de datos deductivas

4.1 Definición del contexto

Como ya se ha comentado en el capítulo 1, el trabajo que se presenta a continuación se enmarca en los campos de las bases de datos deductivas y de las bases de datos activas, por lo que se considera como ámbito de aplicación un sistema de bases de datos deductivo-activo. Asumiendo la formalización lógica de una base de datos deductiva presentada en el apartado 2.1.2, las características fundamentales del sistema en los aspectos deductivo y activo son las que se presentan a continuación.

- En cuanto a la capacidad deductiva, se asume que:
  - El esquema de la base de datos está definido por un par (L, RI) donde L es un lenguaje de primer orden con un conjunto finito de símbolos, sin símbolos de función y con conjuntos disjuntos de símbolos de predicados básicos y derivados. RI es el conjunto de restricciones de integridad, fórmulas cerradas de L.
  - Las restricciones de integridad se representan en forma negada. Dada una restricción de integridad W su forma negada es $\leftarrow inc_W$ donde inc_W es un nuevo símbolo de predicado. Si RI es el conjunto $\{W_1, \ldots, W_m\}$, entonces la representación de las restricciones de integridad en forma negada exige la definición de las siguientes reglas deductivas:
    * $inc_{W_j} \leftarrow \neg W_j \ (1 \leq j \leq m)$.
    * $inc \leftarrow inc_{W_1} \vee \ldots \vee inc_{W_m} \ (1 \leq j \leq m)$.

A los nuevos predicados inc_{W_j} $(1 \leq j \leq m)$ se les denomina *predicados de inconsistencia*. A las reglas con un predicado de inconsistencia en la cabeza se les denomina *reglas de inconsistencia.*

- Un estado de base de datos D consta de dos subconjuntos BDE y BDI donde:
  - BDE es la parte extensional de la base de datos y sus elementos se denominan *hechos:* $BDE \subseteq \{A : A$ es un átomo básico base[1]$\}$.
  - BDI es la parte intensional de la base de datos y sus elementos se denominan *reglas deductivas:* $BDI \subseteq \{A \leftarrow L_1 \wedge \ldots \wedge L_n : A$ es un átomo derivado y L_i $(1 \leq i \leq n)$ es un literal básico o derivado$\}$. Entre las reglas de BDI se incluyen las reglas deductivas que se obtienen al normalizar el conjunto de reglas definido para la representación de las restricciones en forma negada[2].
- La base de datos debe ser permitida, estratificada y estricta[3].
- Las transacciones son conjuntos de operaciones de inserción y/o borrado de hechos, i.e. $insertar(p(\overrightarrow{c}))$ o $borrar(p(\overrightarrow{c}))$ donde p es un predicado básico y $\overrightarrow{c}$ es un vector de constantes.
- La semántica declarativa es la compleción de la base de datos.
- La semántica operacional es el procedimiento de resolución $SLDNF$.
- El concepto de satisfacción de la integridad coincide con el punto de vista de la demostración que, en la semántica asumida, se concreta como sigue: dados un estado de base de datos D y una restricción de integridad W entonces:
  - D satisface W si y sólo si $comp(D) \models W$; y
  - D viola W si y sólo si $comp(D) \not\models W$.

- En cuanto a la capacidad activa, se asume que:
  - Una regla ECA tiene la forma R(*Evento: E, Condición: C, Acción: A*) donde:
    - R es el nombre de la regla.
    - E es una expresión de la forma $insertar(p(\overrightarrow{t}))$ o $borrar(p(\overrightarrow{t}))$ donde p es un predicado básico y $\overrightarrow{t}$ es un vector de términos.
    - C es una conjunción de literales o el predicado *cierto*[4].
    - A es una estructura de datos en árbol (que se presentará más adelante) algunos de cuyos nodos incluyen una operación de la forma *abortar*, $insertar(p(\overrightarrow{t}))$ o $borrar(p(\overrightarrow{t}))$ donde p es un predicado básico y $\overrightarrow{t}$ es un vector de términos.

[1] Ver las definiciones 1 y 2 del apéndice C para la definición de átomo o literal básico o derivado y para la definición de átomo o literal base.

[2] Para ello se puede utilizar el algoritmo de normalización de Lloyd y Topor [37].

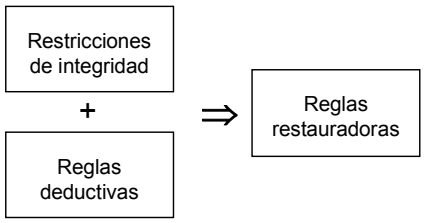
[3] Ver las definiciones 22, 26 y 28 del apéndice C.

[4] *cierto* es un predicado de aridad cero que se evalúa a cierto en cualquier semántica.

- El modo de acoplamiento entre el evento y la condición es diferido y entre la condición y la acción es inmediato.
- La granularidad de las reglas es orientada a la tupla.
- La política de procesamiento de reglas es iterativa.
- Los conflictos al procesar las reglas se resolverán de forma aleatoria.

4.2 Introducción al método

A partir del conjunto de restricciones de integridad y del conjunto de reglas deductivas, el método que se propone genera un conjunto de reglas de actividad que restaura la consistencia de la base de datos cuando, como consecuencia de la ejecución de una transacción de usuario, se ha producido la violación de alguna restricción. Este conjunto de reglas, a las que se denominará *reglas restauradoras*[5], se integran en la base de datos.



Para entender cómo se pueden generar automáticamente las reglas restauradoras se puede realizar la reflexión siguiente: si W es una restricción de integridad que se satisface en un estado de la base de datos y su predicado de inconsistencia asociado es inc_W, entonces la violación de la restricción W se produce cuando la ejecución de una transacción de usuario induce la inserción del átomo inc_W. Con esta visión de la violación de la integridad, el problema de la generación automática de las reglas restauradoras para W podría resolverse en dos pasos:

1. Obtención del conjunto de operaciones de actualización sobre predicados básicos que puedan suponer la inserción del átomo de inconsistencia inc_W.
2. Obtención, para cada operación del conjunto anterior, de una transacción que pueda inducir el borrado del átomo de inconsistencia inc_W.

Si O es una de las operaciones detectadas en el primer paso y T la transacción asociada generada en el segundo paso, entonces la siguiente regla restauradora podría reparar la violación de la restricción W:

R:
`Evento:` O
`Condición:` inc_W
`Acción:` T

El significado de una regla como la anterior es el siguiente: "Cuando la operación O unifique con una operación de una transacción, si inc_W es cierto en el estado posterior a la transacción, entonces debe ejecutarse la transacción T".

[5] Por inspiración del método presentado en el apartado 3.9.

Ejemplo 38 *Sea un esquema de base de datos con* $\{p\}$ *como predicado derivado y* $\{q, s, t, v\}$ *como predicados básicos donde el conjunto de restricciones de integridad,* $\{W = \forall x_1(q(x_1) \rightarrow \neg p(x_1))\}$*, consta de una sola restricción cuya forma negada es* $\leftarrow inc_W$ *donde* inc_W *se define en la regla deductiva 1. Y sea D una base de datos del esquema anterior cuyo conjunto de reglas deductivas es:*

$$1 : inc_W \leftarrow q(x_1) \wedge p(x_1)$$
$$2 : p(x_2) \leftarrow t(x_2)$$
$$3 : p(x_3) \leftarrow s(x_3) \wedge \neg v(x_3)$$

Observando las reglas deductivas de esta base de datos, es fácil determinar, por una parte, que el borrado de una instancia del predicado v[6] *puede suponer la violación de la restricción de integridad W al inducir, a través de las reglas deductivas 1 y 3, la inserción del átomo* inc_W*; y, por otra parte, que el borrado de la misma instancia del predicado q puede reparar la consistencia al inducir el borrado del átomo de inconsistencia* inc_W*. La siguiente regla podría ser entonces una regla restauradora para reparar la violación de la restricción de integridad W:*

R:

`Evento:`	$borrar(v(x))$
`Condición:`	inc_W
`Acción:`	$\{borrar(q(x))\}$

Evidentemente, si se considera como condición de las reglas restauradoras el átomo de inconsistencia asociado a la restricción que se pretende restaurar, entonces la evaluación de la condición puede ser muy costosa, ya que no se utiliza el conocimiento que se tiene sobre cuál ha sido la operación causante de la violación, conocimiento que puede simplificar dicha condición. Por este motivo, una mejora evidente consiste en considerar como condición de la regla una forma simplificada de la restricción de integridad obtenida a partir de dicha operación.

Ejemplo 39 *En la evaluación de la condición de la regla del ejemplo 38 se utiliza la regla deductiva 2 a pesar de que, si el evento que ha disparado la regla es el borrado de instancias del predicado v, entonces la violación de la restricción de integridad sólo se produce a través de las reglas deductivas 1 y 3. Teniendo esta información en cuenta, la regla R puede simplificarse reescribiéndose como sigue:*

R:

`Evento:`	$borrar(v(x))$
`Condición:`	$q(x) \wedge s(x)$
`Acción:`	$\{borrar(q(x))\}$

En esta regla puede observarse que la condición es menos costosa de evaluar que la original.

El resto de este capítulo se organiza como sigue. En el apartado 4.3 se va a ilustrar cómo se pueden generar, en tiempo de definición, estas reglas restauradoras. Para que la presentación de las ideas del método sea más sencilla, en primer lugar se considera sólo la obtención del evento y la condición; primero, para bases de datos jerárquicas

[6] Con la expresión "instancia del predicado v" se quiere hacer referencia a un "hecho construido con el predicado v". Este abuso del lenguaje se justifica ya que, en ocasiones, simplifica el discurso.

(apartado 4.3.1) y después para bases de datos estratificadas (apartado 4.3.2). A continuación, se muestra cómo obtener las acciones de las reglas para bases de datos jerárquicas (apartado 4.3.3) y para bases de datos estratificadas (apartado 4.3.4). Por último en el apartado 4.4 se presenta un algoritmo completo para la generación de las reglas restauradoras de un esquema de bases de datos y en el apartado 4.5 se dan unas orientaciones para su implementación en *SQL*3.

4.3 Estrategia para la obtención del conjunto de las reglas restauradoras

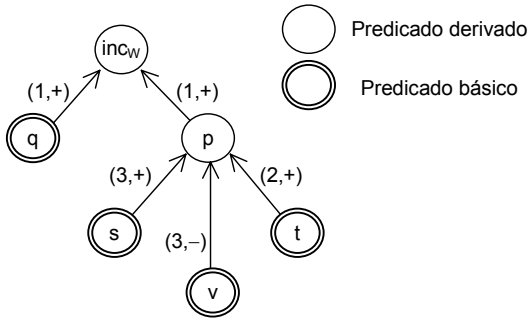
4.3.1 Obtención del evento y de la condición para bases de datos jerárquicas

Para presentar de forma intuitiva cómo se puede obtener el evento y la condición de las reglas restauradoras resulta muy útil construir el *grafo de dependencias* entre los predicados de la base de datos.

Definición 1 *Grafo de dependencias*[7]*.*

El grafo de dependencias entre los predicados de la base de datos está formado por nodos, etiquetados con un símbolo de predicado, unidos por arcos doblemente etiquetados con una regla deductiva y con un signo (+ o −). En el grafo hay un arco del nodo A al nodo B etiquetado con $(r, +)$ (resp. $(r, -)$) si el predicado B aparece en la cabeza de la regla deductiva r, en cuyo cuerpo aparece el predicado A en un literal positivo (resp. negativo).■

Ejemplo 40 *En la siguiente figura se presenta el grafo de dependencias del esquema de la base de datos del ejemplo 38:*



Definición 2 *Camino en el grafo de dependencias.*

Dado un grafo de dependencias y dos de sus nodos A y B, se dice que hay un camino desde A hasta B si existe una secuencia de nodos $P_1, \ldots, P_n$, tal que $P_1 = A$, $P_n = B$ y para todo i $(1 \leq i < n)$ existe un arco desde P_i hasta P_{i+1}. El nodo A es

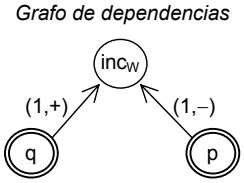
[7] En el apéndice A se ha incluido un índice de todas las definiciones, teoremas y algoritmos presentados en este capítulo.

el *nodo inicial* del camino. El camino es *positivo* (resp. *negativo*) si aparecen cero o un número par (resp. impar) de signos − en los arcos que lo componen.■

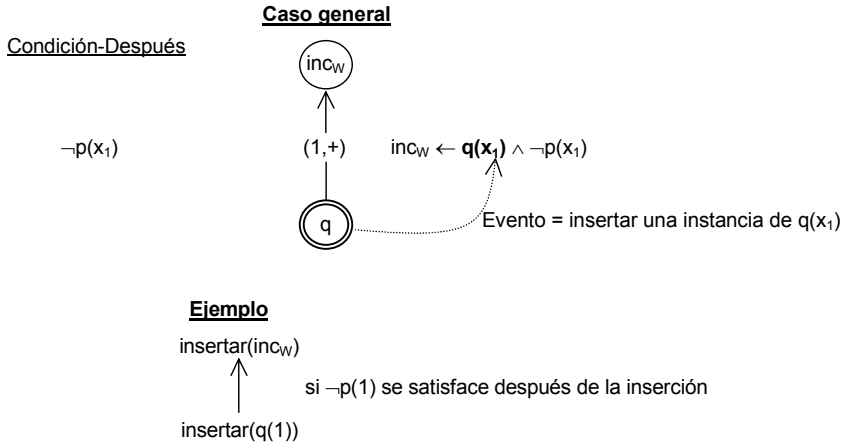
A partir de este grafo de dependencias se puede obtener el evento y la condición de las reglas restauradoras tal como se ilustra en los siguientes ejemplos.

Ejemplo 41 *Sea un esquema de base de datos con* $\{p, q\}$ *como predicados básicos donde el conjunto de restricciones de integridad,* $\{W = \forall x_1(q(x_1) \rightarrow p(x_1))\}$*, consta de una sola restricción cuya forma negada es* $\leftarrow inc_W$ *donde* inc_W *es un nuevo símbolo de predicado que se define por la regla deductiva 1:*

$$1 : inc_W \leftarrow q(x_1) \wedge \neg p(x_1)$$

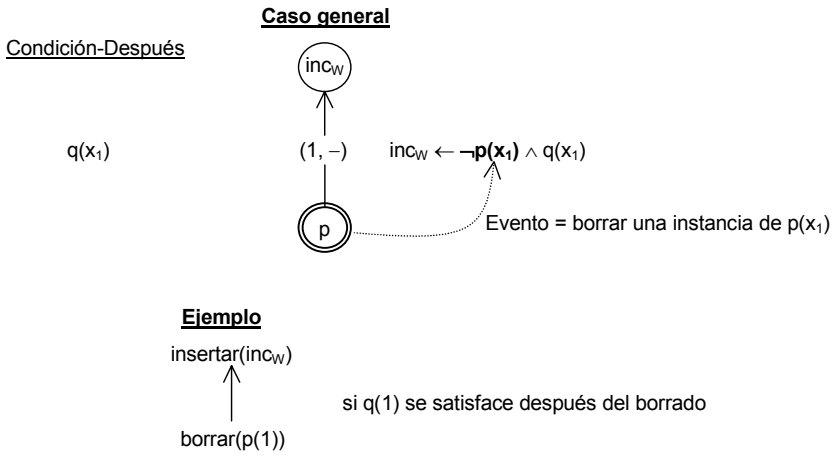


Considérese el camino desde el predicado q hasta inc_W*. Estudiando el único arco de este camino, se puede deducir que la inserción de hechos en el predicado q puede, potencialmente, inducir la inserción del átomo de inconsistencia* inc_W*. La condición que debe cumplirse para que esta inserción sea real es que el resto del cuerpo de la regla deductiva 1,* $\neg p(x_1)$*, se satisfaga (i.e. sea cierto en la semántica asumida) en el estado de base de datos obtenido después de la inserción; por este motivo, esta condición se denomina Condición-Después. Esta reflexión se muestra gráficamente para el caso general y para un ejemplo concreto.*



De este camino puede deducirse que: "La inserción de una instancia del átomo $q(x_1)$*, por ejemplo* $q(1)$*, induce la inserción de un camino de derivación para el átomo* inc_W *si* $\neg p(1)$ *se satisface después de la operación de inserción".*

Considérese ahora el camino desde el predicado p hasta el predicado inc_W*:*



De donde se puede deducir que: "El borrado de una instancia del átomo $p(x_1)$, por ejemplo $p(1)$, induce la inserción de un camino de derivación para el predicado inc_W si $q(1)$ se satisface después de la operación de inserción".

Del ejemplo 41 se pueden extraer ya algunas conclusiones que ayudan a automatizar la generación de las reglas restauradoras. Obviamente, cada camino del grafo desde un predicado básico hasta un predicado de inconsistencia representa un posible camino de violación de la restricción representada por este predicado. Del nodo inicial del camino y del signo de éste, se puede deducir la operación que puede desencadenar la violación. Si el nodo inicial es el predicado p y el camino hasta el predicado de inconsistencia es positivo entonces las operaciones peligrosas son las inserciones de átomos base construidos con el predicado p; si el nodo inicial es el predicado p y el camino hasta el predicado de inconsistencia es negativo entonces las operaciones peligrosas son los borrados de átomos base construidos con el predicado p. Por otra parte, la conjunción que se ha llamado Condición-Después representa una condición que debe satisfacerse después de la operación para poder asegurar que la actualización ha inducido realmente una violación. Las reglas restauradoras, aún sin acción, generadas a partir de los caminos estudiados en el ejemplo 41 se muestran en el ejemplo 42.

Ejemplo 42 *La regla restauradora asociada al camino desde q hasta inc_W sería:*

R_1:
`Evento:` $insertar(q(x_1))$
`Condición:` $\neg p(x_1)$
`Acción:` ...

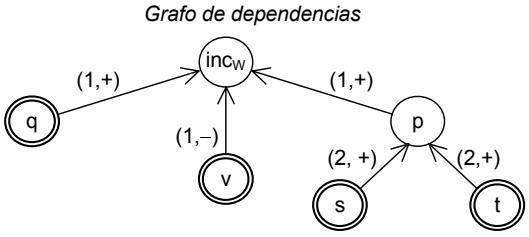
Y la asociada al camino desde p hasta inc_W:

R_2:
`Evento:` $borrar(p(x_1))$
`Condición:` $q(x_1)$
`Acción:` ...

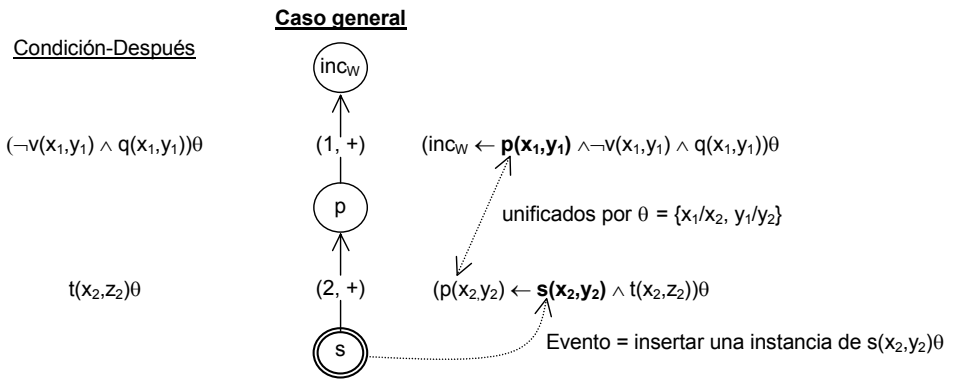
A continuación se presentan algunos ejemplos más; en todos los caminos que se estudian, se muestra la regla que se obtiene a partir del camino tal como se ha hecho en el ejemplo 42. En primer lugar se van a considerar los casos más sencillos. Éstos se corresponden con caminos en los que no hay arcos negativos y la violación se produce exclusivamente a través de inserciones, o bien con caminos en los que el único arco negativo es el primero, es decir, la violación se produce a través de inserciones provocadas por un borrado inicial (como puede apreciarse, los caminos estudiados en el ejemplo 41 son de este tipo). Más adelante se presentan otros casos.

Ejemplo 43 *Sea el siguiente conjunto de reglas deductivas*[8]*:*

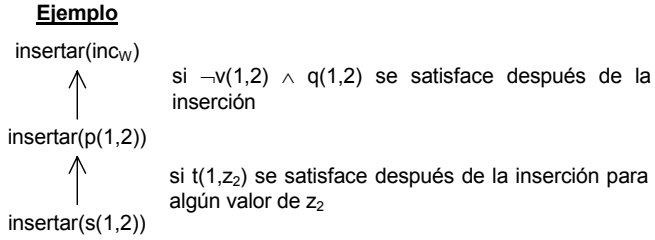
$$1 : inc_W \leftarrow q(x_1, y_1) \wedge \neg v(x_1, y_1) \wedge p(x_1, y_1)$$
$$2 : p(x_2, y_2) \leftarrow s(x_2, y_2) \wedge t(x_2, z_2)$$



Considérese el camino desde s hasta inc_W:



[8] A partir de aquí, y por simplicidad, en los ejemplos sólo se presentará el conjunto de reglas deductivas no haciendo explícito el esquema de la base de datos. Los predicados que no aparezcan en la cabeza de una regla deductiva son predicados básicos.



De nuevo, de este camino puede deducirse que: "La inserción de una instancia de $s(x_2,y_2)$, por ejemplo $s(1,2)$, induce la inserción de un camino de derivación para $p(1,2)$ si hay una substitución para z_2, p.e. $\{z_2/3\}$ tal que $t(1,3)$ se satisface después de la operación de inserción; y la inserción de $p(1,2)$ induce la inserción de un camino de derivación para inc_W si $\neg v(1,2) \wedge q(1,2)$ se satisface después de la operación". Generalizando, se puede decir que la inserción de una instancia de $s(x_2,y_2)$ induce la inserción de inc_W si $t(x_2,z_2) \wedge \neg v(x_2,y_2) \wedge q(x_2,y_2)$ se satisface después de la operación para dicha instancia y algún valor de z_2.

La regla generada a partir de este camino es la siguiente:

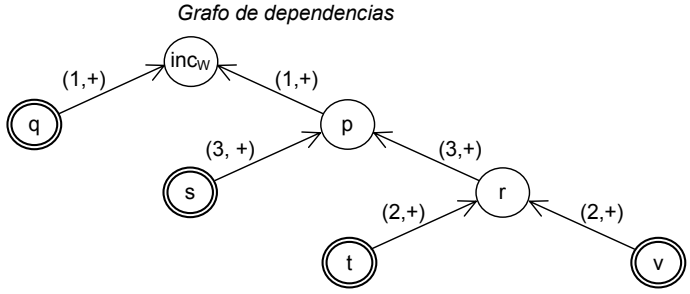
R:

`Evento:`	$insertar(s(x_2,y_2))$
`Condición:`	$t(x_2,z_2) \wedge \neg v(x_2,y_2) \wedge q(x_2,y_2)$
`Acción:`	...

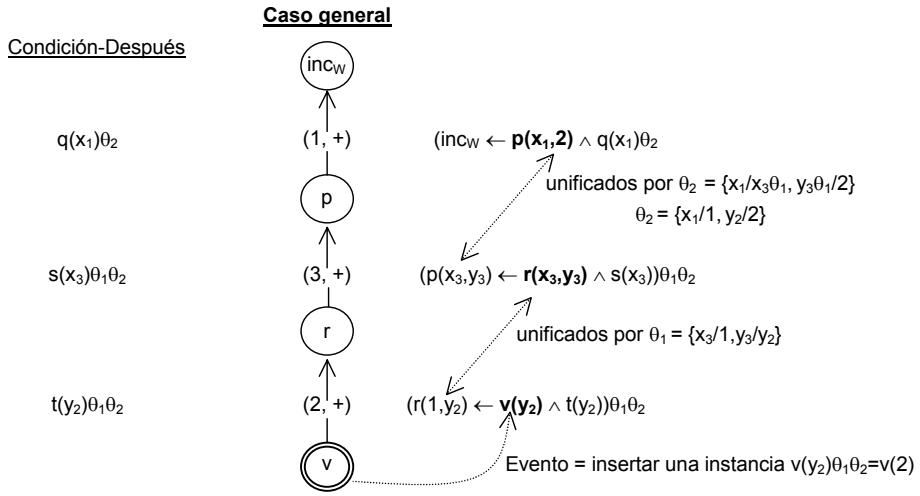
En el ejemplo 44 se ilustra la necesidad de transmitir los unificadores a lo largo de todo el camino debido a la posible presencia de constantes en las reglas deductivas.

Ejemplo 44 *Sea el siguiente conjunto de reglas deductivas:*

$1 : inc_W \leftarrow q(x_1) \wedge p(x_1,2)$
$2 : r(1,y_2) \leftarrow t(y_2) \wedge v(y_2)$
$3 : p(x_3,y_3) \leftarrow s(x_3) \wedge r(x_3,y_3)$



Considérese el camino desde v hasta inc_W:



La regla generada a partir de este camino es la siguiente:

R:

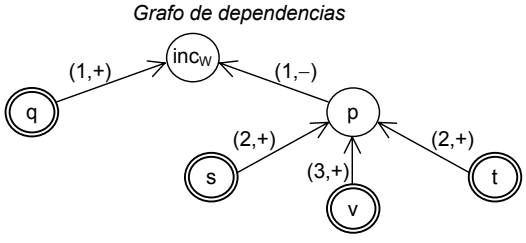
`Evento:`	$insertar(v(2))$
`Condición:`	$q(1) \wedge t(2) \wedge s(1)$
`Acción:`	...

En la que puede observarse que el evento y la condición están instanciados debido a la presencia de constantes en la reglas deductivas.

Considérense ahora caminos en los que aparecen arcos negativos en cualquier punto del camino. En este caso, la violación se produce a través de borrados sobre predicados derivados.

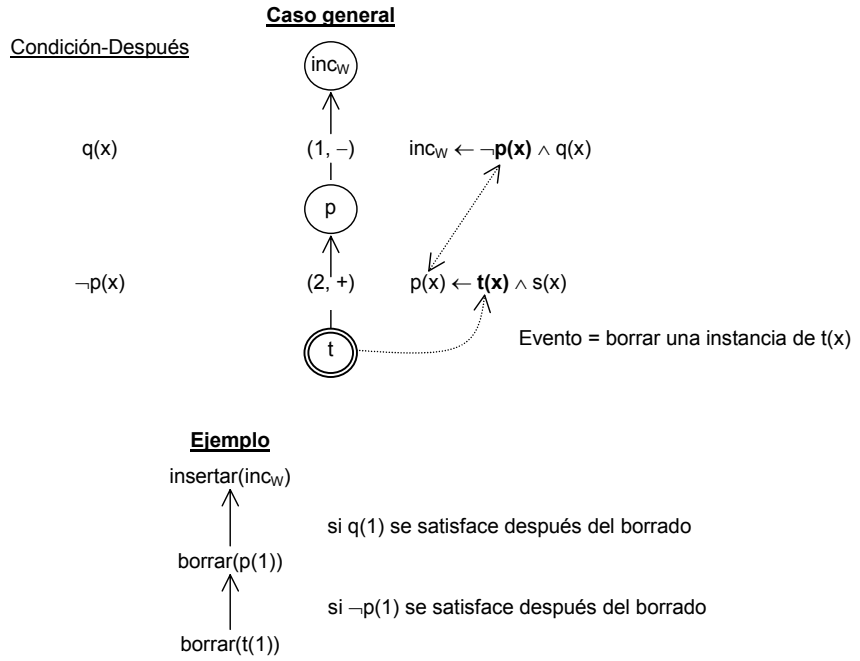
Ejemplo 45 *Sea el siguiente conjunto de reglas deductivas*[9]*:*

$1 : inc_W \leftarrow q(x) \wedge \neg p(x)$
$2 : p(x) \leftarrow s(x) \wedge t(x)$
$3 : p(x) \leftarrow v(x)$



[9] A partir de este punto, y también por claridad en la exposición, se van a eliminar, cuando sea posible, los subíndices en las variables de las reglas deductivas con lo que, en muchos casos, la unificación de literales será evidente no siendo necesario hacerla explícita.

Considérese el camino desde t hasta inc_W:



De este camino puede deducirse que: "El borrado de una instancia de $t(x)$, por ejemplo $t(1)$, puede inducir el borrado de un camino de derivación para $p(1)$, y por lo tanto inducir también el borrado de $p(1)$ siempre que no queden otros caminos de derivación para este átomo. Así, para seguir induciendo actualizaciones a partir de este nodo será necesario comprobar que $\neg p(1)$ se satisface después de la operación. Por otra parte, el borrado de $p(1)$ induce la inserción de inc_W si $q(1)$ se satisface después de la operación". Resumiendo, el borrado de $t(1)$, induce la inserción de un camino de derivación para inc_W si $q(1) \wedge \neg p(1)$ se satisface después de la operación. Así pues, se puede concluir que el borrado de instancias de $t(x)$ puede suponer la inserción de inc_W si la condición $q(x) \wedge \neg p(x)$ se satisface después de la operación.

La regla generada a partir de este camino es la siguiente:

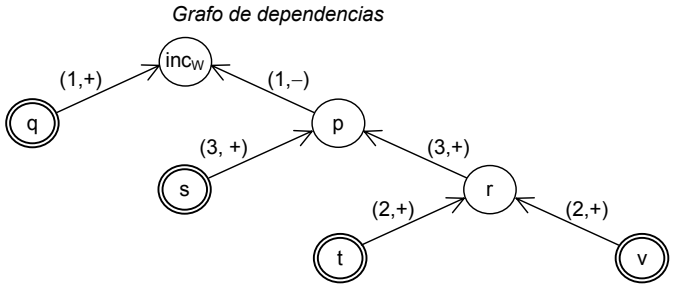
R:

`Evento:` $borrar(t(x))$
`Condición:` $q(x) \wedge \neg p(x)$
`Acción:` ...

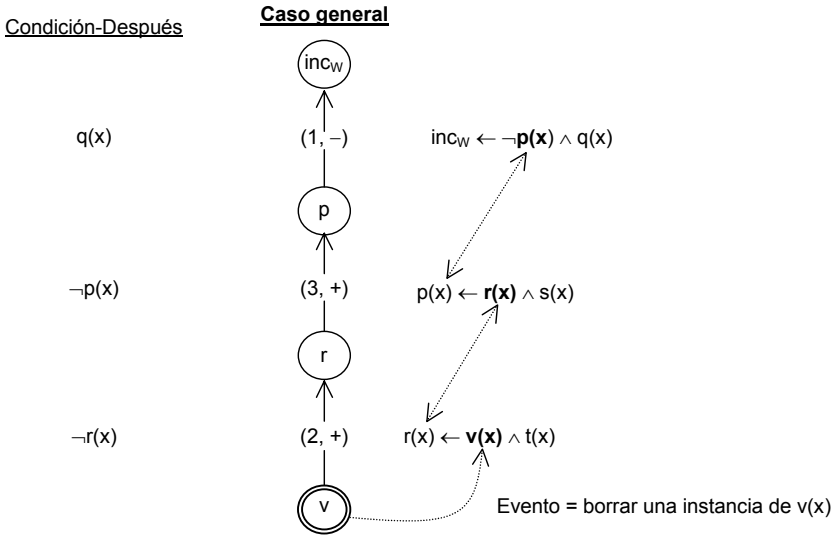
Del ejemplo 45 hay que destacar que cuando en el camino se produce un borrado inducido sobre un predicado derivado, la Condición-Después no puede incluir solamente el resto del cuerpo de la regla como se ha venido haciendo hasta ahora, ya que el borrado de un camino de derivación para un átomo sólo supone su borrado si no existe otro camino de derivación para él.

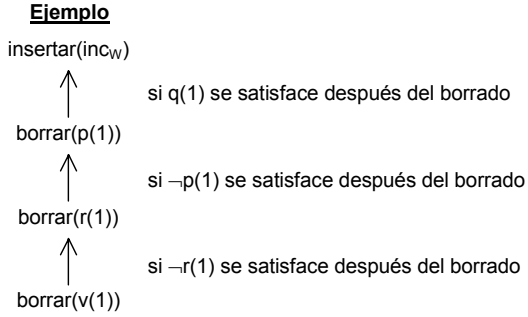
Ejemplo 46 *Sea el siguiente conjunto de reglas deductivas:*

$1 : inc_W \leftarrow q(x) \wedge \neg p(x)$
$2 : r(x) \leftarrow t(x) \wedge v(x)$
$3 : p(x) \leftarrow s(x) \wedge r(x)$



Considérese el camino desde v hasta inc_W:





De este camino puede deducirse que: "El borrado de una instancia de $v(x)$, por ejemplo $v(1)$, puede inducir el borrado de un camino de derivación para $r(1)$, y por lo tanto inducir también el borrado de $r(1)$ siempre que no queden otros caminos de derivación para este átomo; así, para seguir induciendo actualizaciones a partir de este nodo será necesario comprobar que $\neg r(1)$ se satisface después de la operación. Por otra parte, el borrado $r(1)$ puede inducir el borrado de un camino de derivación para $p(1)$, de nuevo habrá que comprobar que $\neg p(1)$ se satisface después de la operación. Por último, el borrado de $p(1)$ induce la inserción de inc_W si $q(1)$ se satisface después de la operación". Resumiendo, el borrado de $v(1)$, induce la inserción de un camino de derivación para inc_W si $q(1) \wedge \neg p(1) \wedge \neg r(1)$ se satisface después de la operación. Esta condición puede simplificarse eliminando el literal $\neg r(1)$ ya que la presencia del literal $\neg p(1)$ hace innecesaria su comprobación. Así pues, se puede concluir que el borrado de instancias de $v(x)$ puede suponer la inserción de inc_W si la condición $q(x) \wedge \neg p(x)$ se satisface después de la operación.

La regla generada a partir de este camino es la siguiente:

R:

`Evento:`	$borrar(v(x))$
`Condición:`	$q(x) \wedge \neg p(x)$
`Acción:`	...

Del ejemplo 46 hay que destacar que si el camino que se está considerando es $P_1, \ldots, P_i, P_{i+1}, \ldots, P_n$ y el arco desde P_i hasta P_{i+1} es el último arco negativo del camino entonces para construir la Condición-Después sólo se utilizan los cuerpos de las reglas que aparecen desde P_i hasta P_n.

Las reglas restauradoras que se han presentado hasta el momento pueden obtenerse a partir de los conjuntos de *actualizaciones inducidas* que se presentan a continuación.

Definición 3 *Actualizaciones Inducidas (AI) en bases de datos jerárquicas.*

Los conjuntos de actualizaciones inducidas llamados AI_{POS} y AI_{NEG} que capturan respectivamente información sobre las posibles inserciones y borrados inducidos sobre predicados derivados, se definen a partir del conjunto de reglas deductivas de la base de datos y están formados por elementos que son tripletes de la forma (P, O, C_D) donde:

- P es un átomo derivado;
- O es una operación de la forma $insertar(Q)$ o $borrar(Q)$ donde Q es un átomo básico; y
- C_D es una conjunción de literales o el predicado *cierto*.

Estos conjuntos se definen inductivamente de la forma siguiente:

$AI^0_{POS} = \{(P, insertar(A), L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_m \text{ (resp. } cierto))|(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge A \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada[10], A es un átomo básico y $m > 1$ (resp. $m = 1$)$\}$

[10] *Refrescar* una regla deductiva consiste en substituir cada uno de los símbolos de variable que aparecen en ella por un nuevo símbolo de variable.

$\cup$

$\{(P, borrar(A), L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_m$ (resp. $cierto))|(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg A \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada, A es un átomo básico y $m > 1$ (resp. $m = 1$)$\}$

$AI^0_{NEG} = \{(P, insertar(A), cierto)|(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg A \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada y A es un átomo básico$\}$

$\cup$

$\{(P, borrar(A), cierto)|(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge A \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada y A es un átomo básico$\}$

$AI^{n+1}_{POS} = AI^{n+1}_{POS,1} \cup AI^{n+1}_{POS,2}$ siendo $n \geq 0$ y los subconjuntos $AI^{n+1}_{POS,i}$ $(1 \leq i \leq 2)$ los siguientes:

$AI^{n+1}_{POS,1} = \{(P\theta, E\theta, (C_D \wedge L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_m)\theta)|(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge B \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada, B es un átomo, $(A, E, C_D) \in AI^n_{POS}$ y $\theta = mgu(A, B)\}$

$AI^{n+1}_{POS,2} = \{(P\theta, E\theta, (L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg B \wedge L_{i+1} \wedge \ldots \wedge L_m)\theta)|(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg B \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada, B es un átomo, $(A, E, cierto) \in AI^n_{NEG}$ y $\theta = mgu(A, B)\}$

$AI^{n+1}_{NEG} = AI^{n+1}_{NEG,1} \cup AI^{n+1}_{NEG,2}$ siendo $n \geq 0$ y los subconjuntos $AI^{n+1}_{NEG,i}$ $(1 \leq i \leq 2)$ los siguientes:

$AI^{n+1}_{NEG,1} = \{(P\theta, E\theta, cierto)|(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge B \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada, B es un átomo, $(A, E, cierto) \in AI^n_{NEG}$ y $\theta = mgu(A, B)\}$

$AI^{n+1}_{NEG,2} = \{(P\theta, E\theta, cierto)|(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg B \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada, B es un átomo, $(A, E, C_D) \in AI^n_{POS}$ y $\theta = mgu(A, B)\}$

La generación de estos conjuntos termina en un j $(j \geq 0)$ tal que $AI^j_{POS} = \emptyset$ y $AI^j_{NEG} = \emptyset$. La existencia de ese valor para j está asegurada ya que la base de datos es jerárquica. Una vez obtenidos todos estos conjuntos, las actualizaciones inducidas son:

$AI_{POS} = \cup_{n \geq 0} AI^n_{POS}$

$AI_{NEG} = \cup_{n \geq 0} AI^n_{NEG}$ ■

Las reglas restauradoras, aún sin acción, pueden obtenerse a partir del conjunto AI_{POS} tal y como se muestra en el siguiente algoritmo. En él se define una regla restauradora para cada elemento (P, O, C_D) de AI_{POS} tal que el predicado de P sea un predicado de inconsistencia.

Algoritmo 1 *Algoritmo de generación del evento y la condición de las reglas restauradoras en bases de datos jerárquicas.*

```
ALGORITMO Generación_Reglas_Restauradoras_1
ENTRADA
    AI_POS : Conjunto de actualizaciones inducidas según la definición 3;
SALIDA
    ℜ : Conjunto de reglas restauradoras sin acción;
VARIABLES
    i : Entero;
INICIO
ℜ := ∅;
i := 1;
PARA CADA (P, O, C_D) ∈ AI_POS HACER
| SI el predicado de P es el predicado de inconsistencia inc_W
| ENTONCES
| | ℜ := ℜ ∪ {R_i_inc_W(Evento: O, Condición: C_D, Acción:...)};
| | i := i + 1;
| FIN_SI
FIN PARA
FIN.■
```

En una regla restauradora obtenida con el algoritmo anterior, el evento O es una operación relevante para la restricción W y la condición C es una forma simplificada de W. Así, la regla se disparará cuando se produzca una operación relevante para W pero sólo se ejecutará cuando se compruebe (de forma simplificada) que W realmente se viola en el nuevo estado.

Ejemplo 47 *Sea el conjunto de reglas deductivas del ejemplo 46. Los conjuntos AI_{POS} y AI_{NEG} son los siguientes*[11]*:*

$$AI^0_{POS} = \{(inc_W, insertar(q(x)), \neg p(x)), (r(x), insertar(v(x)), t(x)), (r(x), insertar(t(x)), v(x)), (p(x), insertar(s(x)), r(x))\}$$

$$AI^0_{NEG} = \{(inc_W, borrar(q(x)), cierto), (r(x), borrar(v(x)), cierto), (r(x), borrar(t(x)), cierto), (p(x), borrar(s(x)), cierto))\}$$

$$AI^1_{POS} = \{(p(x), insertar(v(x)), t(x) \wedge s(x)), (p(x), insertar(t(x)), v(x) \wedge s(x)), (inc_W, borrar(s(x)), q(x) \wedge \neg p(x))\}$$

$$AI^1_{NEG} = \{(p(x), borrar(v(x)), cierto), (p(x), borrar(t(x)), cierto), (inc_W, insertar(s(x)), cierto)\}$$

[11] En la generación de estos conjunto cada vez que se utilice una regla deductiva, debería utilizarse una variante refrescada, sin embargo, por claridad y mientras no sea necesario, se utilizarán las variables de la regla.

$$AI^2_{POS} = \{(inc_W, borrar(v(x)), q(x) \wedge \neg p(x)), (inc_W, borrar(t(x)), q(x) \wedge \neg p(x))\}$$

$$AI^2_{NEG} = \{(inc_W, insertar(v(x)), cierto), (inc_W, insertar(t(x)), cierto)\}$$

$$AI^3_{POS} = \emptyset$$

$$AI^3_{NEG} = \emptyset$$

$$AI_{POS} = AI^0_{POS} \cup AI^1_{POS} \cup AI^2_{POS}$$

$$AI_{NEG} = AI^0_{NEG} \cup AI^1_{NEG} \cup AI^2_{NEG}$$

Las reglas restauradoras que se generan son las siguientes:

$R_1\_inc_W$:
Evento: $insertar(q(x))$
Condición: $\neg p(x)$
Acción: ...

$R_2\_inc_W$:
Evento: $borrar(s(x))$
Condición: $q(x) \wedge \neg p(x)$
Acción: ...

$R_3\_inc_W$:
Evento: $borrar(v(x))$
Condición: $q(x) \wedge \neg p(x)$
Acción: ...

$R_4\_inc_W$:
Evento: $borrar(t(x))$
Condición: $q(x) \wedge \neg p(x)$
Acción: ...

Teorema 1 *Propiedad de los elementos de AI_{POS} en bases de datos jerárquicas.*

$\Rightarrow$ Sean (P, O, C_D) un elemento de AI_{POS}, O' una operación de inserción o borrado de un hecho y D un estado de base de datos tal que:

- $\alpha = mgu(O, O')$,
- D' es el estado de base de datos obtenido al ejecutar O' en D, y
- existe una refutación $SLDNF$ para $D' \cup \{\leftarrow C_D\alpha\}$ con respuesta computada β,

$\Rightarrow$ entonces existe una refutación $SLDNF$ para $D' \cup \{\leftarrow P\alpha\beta\}$.

Demostración:

La demostración se va a realizar por inducción sobre el nivel n del conjunto AI^n_{POS} al que pertenece el elemento (P, O, C_D).

Caso base: $n = 0$

Sea (P, O, C_D) un elemento de AI^0_{POS}; se pueden distinguir dos casos en función de que O' sea una operación de inserción o de borrado:

- Si O' es una operación de inserción, entonces tiene la forma $O' = insertar(A')$ donde A' es un hecho; además, por el enunciado del teorema se cumple que $D' = D \cup \{A'\}$, que en el elemento (P, O, C_D), $O = insertar(A)$, $A\alpha = A'$, y $C_D = L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_m$, y que existe la regla deductiva $P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge A \wedge L_{i+1} \wedge \ldots \wedge L_m$.

  En este caso, si existe una refutación $SLDNF$ para $D' \cup \{(\leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_m)\alpha\}$ con respuesta computada β, y además $A\alpha \in D'$ $(A\alpha = A')$, entonces existe una refutación $SLDNF$ para $D' \cup \{(\leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge A \wedge L_{i+1} \wedge \ldots \wedge L_m)\alpha\beta\}$ **(1)**.

  Teniendo en cuenta la regla deductiva $P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge A \wedge L_{i+1} \wedge \ldots \wedge L_m$ y el resultado (1) entonces se puede afirmar que existe una refutación $SLDNF$ para $D' \cup \{\leftarrow P\alpha\beta\}$.

- Si O' es una operación de borrado, entonces tiene la forma $O' = borrar(A')$ donde A' es un hecho; además, por el enunciado del teorema se cumple que $D' = D - \{A'\}$, que en el elemento (P, O, C_D), $O = borrar(A)$, $A\alpha = A'$, y $C_D = L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_m$, y que existe la regla deductiva $P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg A \wedge L_{i+1} \wedge \ldots \wedge L_m$.

  Dado que $A\alpha \notin D'$, ya que el hecho A' ha sido borrado, entonces existe un árbol $SLDNF$ fallado finitamente para $D' \cup \{\leftarrow A\alpha\}$ **(2).**

  Si existe una refutación $SLDNF$ para $D' \cup \{(\leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_m)\alpha\}$ con respuesta computada β, entonces teniendo en cuenta el resultado (2) existe una refutación $SLDNF$ para $D' \cup \{(\leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg A \wedge L_{i+1} \wedge \ldots \wedge L_m)\alpha\beta\}$ **(3)**.

  A partir de la regla deductiva $P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg A \wedge L_{i+1} \wedge \ldots \wedge L_m$ y del resultado (3) se puede afirmar que existe una refutación $SLDNF$ para $D' \cup \{\leftarrow P\alpha\beta\}$.

Paso de inducción: $n > 0$

Si la propiedad se cumple para los elementos de AI^{n-1}_{POS} entonces se cumple para AI^n_{POS}. De nuevo se distinguen dos casos en función de que el elemento pertenezca a $AI^n_{POS,1}$ o a $AI^n_{POS,2}$:

- Si el elemento pertenece a $AI^n_{POS,1}$ entonces, por definición, tiene la forma $(P\theta, E\theta, (C_D \wedge L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_m)\theta)$ y existen un elemento $(A, E, C_D) \in AI^{n-1}_{POS}$ y una regla deductiva $P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge B \wedge L_{i+1} \wedge \ldots \wedge L_m$ tal que $\theta = mgu(A, B)$; además, por el enunciado del teorema, $\alpha = mgu(O', E\theta)$ ($E\theta\alpha$ es base).

Si la propiedad se cumple para un elemento (A, E, C_D) de AI^{n-1}_{POS} entonces también se cumple para cualquier instancia, por lo que se cumple para $(A\theta, E\theta, C_D\theta)$. Por hipótesis de inducción, si existe una refutación $SLDNF$ para $D' \cup \{\leftarrow C_D\theta\alpha\}$ con respuesta computada β entonces existe una refutación $SLDNF$ para $D' \cup \{\leftarrow A\theta\alpha\beta\}$ $(\alpha = mgu(O', E\theta))$ **(4)**.

Si existe una refutación $SLDNF$ para $D' \cup \{\leftarrow (C_D \wedge L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_m)\theta\alpha\}$ con respuesta computada β entonces, teniendo en cuenta el resultado (4), existe una refutación $SLDNF$ para $D' \cup \{\leftarrow (C_D \wedge L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_m \wedge A)\theta\alpha\}$ con respuesta computada β **(5)**.

Si $P\theta \leftarrow (L_1 \wedge \ldots \wedge L_{i-1} \wedge B \wedge L_{i+1} \wedge \ldots \wedge L_m)\theta$ es una instancia de la regla deductiva $P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge B \wedge L_{i+1} \wedge \ldots \wedge L_m$ siendo $\theta = mgu(A, B)$ entonces, teniendo en cuenta el resultado (5), se cumple que si existe una refutación $SLDNF$ para $D' \cup \{\leftarrow (C_D \wedge L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_m)\theta\alpha\}$ con respuesta computada β existe una refutación $SLDNF$ para $D' \cup \{\leftarrow P\theta\alpha\beta\}$.

- Si el elemento pertenece a $AI^{n}_{POS,2}$ entonces, por definición, tiene la forma $(P\theta, E\theta, (L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg B \wedge L_{i+1} \wedge \ldots \wedge L_m)\theta)$ y existen un elemento $(A, E, cierto) \in AI^{n-1}_{NEG}$ y una regla deductiva $P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg B \wedge L_{i+1} \wedge \ldots \wedge L_m$ tal que $\theta = mgu(A, B)$; además, por el enunciado del teorema, $\alpha = mgu(O', E\theta)$ ($E\theta\alpha$ es base).

  Si existe una refutación $SLDNF$ para $D' \cup \{\leftarrow (L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg B \wedge L_{i+1} \wedge \ldots \wedge L_m)\theta\alpha\}$ con respuesta computada β teniendo en cuenta la regla deductiva $P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg B \wedge L_{i+1} \wedge \ldots \wedge L_m$, entonces existe una refutación $SLDNF$ para $D' \cup \{\leftarrow P\theta\alpha\beta\}$.■

Corolario 1 *Corrección del algoritmo de generación del evento y la condición de las reglas restauradoras en bases de datos jerárquicas.*

⇒ Sean $R(Evento\colon O, Condición\colon C, Acción\colon \ldots)$ una regla restauradora obtenida con el algoritmo 1 asociada a una restricción de integridad, W, cuyo predicado de inconsistencia es inc_W, O' una operación de inserción o borrado de un hecho, y D un estado de base de datos tal que:

- $\alpha = mgu(O, O')$,
- D' es el estado de base de datos obtenido al ejecutar O' en D, y
- existe una refutación $SLDNF$ para $D' \cup \{\leftarrow C\alpha\}$,

⇒ entonces el estado D' viola la restricción de integridad W.

Demostración:

Si inc_W es el predicado de inconsistencia asociado a la restricción de integridad W, entonces en la base de datos existe la regla deductiva $inc_W \leftarrow \neg W$ (o el resultado de normalizar esta regla) **(1)**.

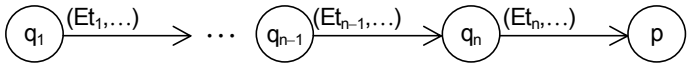
Dado el procedimiento de generación de las reglas restauradoras, si $R(Evento\colon O, Condición\colon C, Acción\colon \ldots)$ es una regla restauradora asociada a una restricción de

integridad cuyo predicado de inconsistencia es inc_W entonces debe existir un elemento de la forma (inc_W, O, C) en AI_{POS}. Si $\alpha = mgu(O, O')$ y existe una refutación $SLDNF$ para $D' \cup \{\leftarrow C\alpha\}$ entonces por el teorema 1 existe una refutación $SLDNF$ para $D' \cup \{\leftarrow inc_W\}$ y teniendo en cuenta el resultado (1) existe una refutación $SLDNF$ para $D' \cup \{\leftarrow \neg W\}$ y por tanto un árbol fallado finitamente para $D' \cup \{\leftarrow W\}$ por lo que $comp(D') \not\models W$ con lo que D' viola la restricción W.■

4.3.2 Obtención del evento y la condición para bases de datos estratificadas

La presencia de recursividad en las reglas deductivas implica, debido a la aparición de ciclos, la existencia de infinitos caminos en el grafo de dependencias por lo que la generación de los conjuntos de actualizaciones inducidas es un proceso infinito. Para resolver este problema es necesario modificar la generación de las actualizaciones inducidas de forma que se pueda asegurar que se alcanzará un j $(j > 0)$ tal que $AI^j_{POS} = \emptyset$ y $AI^j_{NEG} = \emptyset$.

La solución que se propone para resolver este problema consiste en la detección, durante el proceso de generación, de aquellas actualizaciones que subsumen a otras ya generadas. Para implementar esta solución, en cada actualización inducida se guarda el camino que se ha seguido para su obtención; este camino se va a representar como una lista de elementos de la forma (q, Et) donde q es un símbolo de predicado y Et es la etiqueta asociada a una regla deductiva. Así, si el elemento $(p(\overrightarrow{t}), O(q_1(\overrightarrow{t'})), C_D, [(q_1, Et_1), \ldots, (q_n, Et_n)])$ es una actualización inducida, esto significa que en el grafo de dependencias existe el siguiente camino:

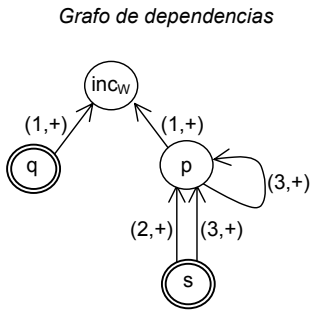


y que la actualización sobre q_1 puede inducir, a través de ese camino, una actualización sobre p.

A partir de este punto, cada actualización inducida se extenderá con una lista de este tipo. Estas ideas se presentan en el siguiente ejemplo.

Ejemplo 48 *Sea el siguiente conjunto de reglas deductivas:*

$$1 : inc_W \leftarrow p(x_1, y_1) \wedge q(x_1, y_1)$$
$$2 : p(x_2, y_2) \leftarrow s(x_2, y_2)$$
$$3 : p(x_3, y_3) \leftarrow s(x_3, z_3) \wedge p(z_3, y_3)$$



Los conjuntos AI_{POS} *y* AI_{NEG} de nivel 0 y 1 *son los siguientes*[12]*:*

$$AI^0_{POS} = \{(inc_W, insertar(q(x^0_1, y^0_1)), p(x^0_1, y^0_1), [(q,1)]),$$
$$(p(x^0_2, y^0_2), insertar(s(x^0_2, y^0_2)), cierto, [(s,2)]),$$
$$(p(x^0_3, y^0_3), insertar(s(x^0_3, z^0_3)), p(z^0_3, y^0_3), [(s,3)])\}$$

$$AI^0_{NEG} = \{(inc_W, borrar(q(x^1_1, y^1_1)), cierto, [(q,1)]),$$
$$(p(x^1_2, y^1_2), borrar(s(x^1_2, y^1_2)), cierto, [(s,2)]),$$
$$(p(x^1_3, y^1_3), borrar(s(x^1_3, z^1_3)), cierto, [(s,3)])\}$$

$$AI^1_{POS} = \{(inc_W, insertar(s(x^2_1, y^2_1)), q(x^2_1, y^2_1), [(s,2),(p,1)]),$$
$$(inc_W, insertar(s(x^3_1, z^0_3)), p(z^0_3, y^3_1) \wedge q(x^3_1, y^3_1), [(s,3),(p,1)]),$$
$$(p(x^2_3, y^2_3), insertar(s(z^2_3, y^2_3)), s(x^2_3, z^2_3), [(s,2),(p,3)]),$$
$$(p(x^3_3, y^3_3), insertar(s(z^3_3, z^0_3)), p(z^0_3, y^3_3) \wedge s(x^3_3, z^3_3), [(s,3),(p,3)])\}$$

$$AI^1_{NEG} = \{(inc_W, borrar(s(x^4_1, y^4_1)), cierto, [(s,2),(p,1)]),$$
$$(inc_W, borrar(s(x^5_1, z^1_3)), cierto, [(s,3),(p,1)]),$$
$$(p(x^4_3, y^4_3), borrar(s(z^4_3, y^4_3)), cierto, [(s,2),(p,3)]),$$
$$(p(x^5_3, y^5_3), borrar(s(z^5_3, z^1_3)), cierto, [(s,3),(p,3)])\}$$

...

Dado que AI^1_{POS} *(resp.* AI^1_{NEG}*) contiene dos elementos que representan la inserción (resp. el borrado) inducido de instancias de p y teniendo en cuenta la regla deductiva* 3*, el conjunto* AI^2_{POS} *(resp.* AI^2_{NEG}*) también contendrá otros dos elementos que representen la inserción (resp. el borrado) de instancias de p. Esta situación, que se repite infinitamente, se puede detectar estudiando el camino a través del cual se ha inducido cada elemento ya que el predicado p aparece en él. Para resolver el problema, estos elementos recursivos van a substituirse por otros que representan actualizaciones inducidas más generales, que detectan la inserción (resp. el borrado) de instancias de p utilizando un número cualquiera de veces la regla deductiva recursiva* 3*. En estos elementos se utiliza el símbolo subguión (_) para representar variables nuevas. Así pues, los conjuntos* AI^1_{POS} *y* AI^1_{NEG} *quedarían de la forma siguiente:*

$$AI^1_{POS} = \{(inc_W, insertar(s(x^2_1, y^2_1)), q(x^2_1, y^2_1), [(s,2),(p,1)]),$$
$$(inc_W, insertar(s(x^3_1, z^0_3)), p(z^0_3, y^3_1) \wedge q(x^3_1, y^3_1), [(s,3),(p,1)]),$$
$$(p(x^2_3, y^2_3), insertar(s(\_,\_)), p(x^2_3, y^2_3), [(s,2),(p,3)]),$$
$$(p(x^3_3, y^3_3), insertar(s(\_,\_)), p(x^3_3, y^3_3), [(s,3),(p,3)])\}$$

[12]Las variantes de refresco de cada regla deductiva utilizadas en la generación de los elementos se indican mediante el uso de superíndices.

$$AI^1_{NEG} = \{(inc_W, borrar(s(x^4_1, y^4_1)), cierto, [(s,2),(p,1)]),$$
$$(inc_W, borrar(s(x^5_1, z^1_3)), cierto, [(s,3),(p,1)]),$$
$$(p(x^4_3, y^4_3), borrar(s(\_,\_)), cierto, [(s,2),(p,3)]),$$
$$(p(x^5_3, y^5_3), borrar(s(\_,\_)), cierto, [(s,3),(p,3)])\}$$

Generalizando estas ideas, cuando en una actualización inducida (P, O, C_D, L) de AI^i_{POS} (resp. $(P, O, cierto, L)$ de AI^i_{NEG}), donde L es la lista que representa el camino seguido para generarla, el predicado de P aparece en L, entonces el elemento debe substituirse por (P, O^*, P, L) (resp. $(P, O^*, cierto, L)$) donde O^* se obtiene a partir de O substituyendo cualquier término que aparezca por un nuevo símbolo de variable (que como ya se ha dicho va a representarse por un subguión). De esta forma se consigue aislar la operación de las actualizaciones que pueda inducir. A este tipo de actualizaciones inducidas se les denominará *genéricas.*

Una vez introducido el concepto de actualización inducida genérica habrá que modificar la forma de obtener los conjuntos AI_{POS} y AI_{NEG} para evitar que su generación sea infinita. Inicialmente podría considerarse que una buena idea consiste en no incluir en los conjuntos de actualizaciones inducidas un elemento en cuyo camino aparezca dos veces el mismo par (q, Et) tal y como puede observarse en el ejemplo siguiente.

Ejemplo 49 *Los conjuntos de actualizaciones inducidas de nivel superior a 1 para el ejemplo 48 son los siguientes:*

$$AI^2_{POS} = \{(inc_W, insertar(s(\_,\_)), p(x^6_1, y^6_1) \wedge q(x^6_1, y^6_1), [(s,2),(p,3),(p,1)]),$$
$$(inc_W, insertar(s(\_,\_)), p(x^7_1, y^7_1) \wedge q(x^7_1, y^7_1), [(s,3),(p,3),(p,1)])\}$$

$$AI^2_{NEG} = \{(inc_W, borrar(s(\_,\_)), cierto, [(s,2),(p,3),(p,1)]),$$
$$(inc_W, borrar(s(\_,\_)), cierto, [(s,3),(p,3),(p,1)])\}$$

En estos conjuntos no se incluye ninguna actualización inducida sobre p ya que en su camino aparcería dos veces el par $(p, 3)$.

$AI^3_{POS} = \emptyset$

$AI^3_{NEG} = \emptyset$

Las reglas restauradoras obtenidas a partir de las inserciones inducidas sobre inc_W *son las siguientes:*

R_1:
`Evento:` $insertar(q(x^0_1, y^0_1))$
`Condición:` $p(x^0_1, y^0_1)$
`Acción:` ...

R_2:
`Evento:` $insertar(s(x^2_1, y^2_1))$
`Condición:` $q(x^2_1, y^2_1)$
`Acción:` ...

R_3:
`Evento:` $insertar(s(x^3_1, z^0_3))$
`Condición:` $p(z^0_3, y^3_1) \wedge q(x^3_1, y^3_1)$
`Acción:` ...

R_4:

Evento:	$insertar(s(\_,\_))$.
Condición:	$p(x_1^6, y_1^6) \wedge q(x_1^6, y_1^6)$
Acción:	...

R_5:

Evento:	$insertar(s(\_,\_))$
Condición:	$p(x_1^7, y_1^7) \wedge q(x_1^7, y_1^7)$
Acción:	...

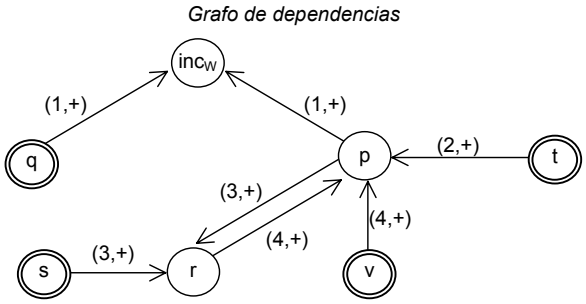
La regla R_1 restaura la inconsistencia inducida, por la inserción inicial de hechos en el predicado q, a través de la regla deductiva 1; la regla R_2 restaura la inconsistencia inducida, por la inserción inicial de hechos en el predicado s, a través de las reglas deductivas 2 y 1; la regla R_3 restaura la inconsistencia inducida, por la inserción inicial de hechos en el predicado s, a través de las reglas deductivas 3 y 1; las reglas R_4 y R_5 restauran la inconsistencia inducida, por la inserción inicial de hechos en el predicado s, utilizando cualquier número de veces la regla deductiva 3 y, finalmente la regla deductiva 1.

Como puede observarse en el ejemplo anterior, debido al uso de actualizaciones inducidas genéricas, es posible que se generen varias reglas restauradoras "equivalentes". Este punto será considerado más adelante.

La modificación sugerida en la generación de las actualizaciones inducidas presentada en el ejemplo anterior puede no ser, sin embargo, adecuada cuando la longitud del ciclo en el grafo de dependencias es superior a 1. Este problema se ilustra en el ejemplo siguiente.

Ejemplo 50 *Sea el siguiente conjunto de reglas deductivas:*

$1 : inc_W \leftarrow p(x_1) \wedge q(x_1)$
$2 : p(x_2) \leftarrow t(x_2)$
$3 : r(x_3) \leftarrow s(x_3, y_3) \wedge p(y_3)$
$4 : p(x_4) \leftarrow v(x_4) \wedge r(x_4)$



Los conjuntos de actualizaciones inducidas para este ejemplo aplicando la estrategia comentada anteriormente son los siguientes[13]*:*

[13] Debido a la ausencia de literales negativos en los cuerpos de las reglas deductivas de este ejemplo, las inserciones inducidas sobre el predicado de inconsistencia son independientes de los borrados por ello no se presentan los conjuntos AI^i_{NEG}.

$$AI^0_{POS} = \{(inc_W, insertar(q(x^0_1)), p(x^0_1), [(q,1)]),\ (p(x^0_2), insertar(t(x^0_2)), cierto, [(t,2)]),\ (r(x^0_3), insertar(s(x^0_3, y^0_3)), p(y^0_3), [(s,3)]),\ (p(x^0_4), insertar(v(x^0_4)), r(x^0_4), [(v,4)])\}$$

$$AI^0_{NEG} = \{\dots\}$$

$$AI^1_{POS} = \{(inc_W, insertar(t(x^1_1)), q(x^1_1), [(t,2),(p,1)]),\ (r(x^1_3), insertar(t(y^1_3)), s(x^1_3, y^1_3), [(t,2),(p,3)]),\ (p(x^1_4), insertar(s(x^1_4, y^0_3)), p(y^0_3) \wedge v(x^1_4), [(s,3),(r,4)]),\ (inc_W, insertar(v(x^2_1)), r(x^2_1) \wedge q(x^2_1), [(v,4),(p,1)]),\ (r(x^2_3), insertar(v(y^2_3)), r(y^2_3) \wedge s(x^2_3, y^2_3), [(v,4),(p,3)])\}$$

$$AI^1_{NEG} = \{\dots\}$$

$$AI^2_{POS} = \{(p(x^2_4), insertar(t(\_)), p(x^2_4), [(t,2),(p,3),(r,4)]),\ (inc_W, insertar(s(x^3_1, y^0_3)), p(y^0_3) \wedge v(x^3_1) \wedge q(x^3_1), [(s,3),(r,4),(p,1)]),\ (r(x^3_3), insertar(s(\_,\_)), r(x^3_3), [(s,3),(r,4),(p,3)]),\ (p(x^3_4), insertar(v(\_)), p(x^3_4), [(v,4),(p,3),(r,4)])\}$$

$$AI^2_{NEG} = \{\dots\}$$

$$AI^3_{POS} = \{(inc_W, insertar(t(\_)), p(x^4_1) \wedge q(x^4_1), [(t,2),(p,3),(r,4),(p,1)]),\ (inc_W, insertar(v(\_)), p(x^5_1) \wedge q(x^5_1), [(v,4),(p,3),(r,4),(p,1)])\}$$

$$AI^3_{NEG} = \{\dots\}$$

$$AI^4_{POS} = \emptyset$$

$$AI^4_{NEG} = \emptyset$$

En el conjunto AI^3_{POS} no se incluye ninguna actualización sobre p ya que en sus listas aparecería dos veces el par $(r,4)$, ni sobre r porque se repetiría el par $(p,3)$. Las reglas restauradoras obtenidas a partir de las inserciones inducidas sobre inc_W son las siguientes:

R_1:
- Evento: $insertar(q(x^0_1))$
- Condición: $p(x^0_1)$
- Acción: ...

R_2:
- Evento: $insertar(t(x^1_1))$
- Condición: $q(x^1_1)$
- Acción: ...

R_3:
- Evento: $insertar(v(x^2_1))$
- Condición: $r(x^2_1) \wedge q(x^2_1)$
- Acción: ...

R_4:
- Evento: $insertar(s(x^3_1, y^0_3))$
- Condición: $p(y^0_3) \wedge v(x^3_1) \wedge q(x^3_1)$
- Acción: ...

R_5:
Evento: $insertar(t(\_))$
Condición: $p(x_1^4) \wedge q(x_1^4)$
Acción: ...

R_6:
Evento: $insertar(v(\_))$
Condición: $p(x_1^5) \wedge q(x_1^5)$
Acción: ...

En este conjunto de reglas no se controla la posible violación inducida por la inserción inicial de instancias de s utilizando cualquier número de veces las reglas deductivas (recursivas) 3 y 4. Así, si se supone una base de datos que contiene los hechos $\{t(1), q(3), v(2), v(3), s(3,2)\}$*, la violación inducida por la inserción del hecho* $s(2,1)$ *no es detectada por ninguna regla.*

Del ejemplo anterior se puede concluir que, con la modificación propuesta para la generación de las actualizaciones inducidas, es posible que no se detecten todas las violaciones. Para resolver este problema hay que asegurar que para cada predicado básico desde el que se alcanza un ciclo en el grafo, hay una actualización inducida genérica (de inserción o borrado según el caso) sobre cada uno de los predicados que forman parte del ciclo. Para asegurar que se obtienen todas estas actualizaciones genéricas es necesario recorrer cada ciclo al menos dos veces.

Definición 4 *Regla de parada en la generación de las actualizaciones inducidas.*

En los conjuntos de actualizaciones inducidas no se incluirá ningún elemento en cuyo camino se haya utilizado el mismo par (q, Et) más de dos veces.■

Ejemplo 51 *Los conjuntos de actualizaciones inducidas del ejemplo 50 utilizando el nuevo criterio son los siguientes:*

$$\begin{aligned}
AI^3_{POS} = \quad & \{(inc_W, insertar(t(_)), p(x_1^4) \wedge q(x_1^4), [(t,2),(p,3),(r,4),(p,1)]),\\
& (r(x_3^4), insertar(t(_)), r(x_3^4), [(t,2),(p,3),(r,4),(p,3)]),\\
& (p(x_4^4), insertar(s(_,_)), p(x_4^4), [(s,3),(r,4),(p,3),(r,4)]),\\
& (inc_W, insertar(v(_)), p(x_1^5) \wedge q(x_1^5), [(v,4),(p,3),(r,4),(p,1)]),\\
& (r(x_3^5), insertar(v(_)), r(x_3^5), [(v,4),(p,3),(r,4),(p,3)])\}\\
AI^3_{NEG} = \quad & \{\dots\}\\
AI^4_{POS} = \quad & \{(p(x_4^5), insertar(t(_)), p(x_4^5), [(t,2),(p,3),(r,4),(p,3),(r,4)]),\\
& (inc_W, insertar(s(_,_)), p(x_1^6) \wedge q(x_1^6),\\
& \qquad [(s,3),(r,4),(p,3),(r,4),(p,1)]),\\
& (r(x_3^5), insertar(s(_,_)), r(x_3^5), [(s,3),(r,4),(p,3),(r,4),(p,3)]),\\
& (p(x_4^6), insertar(v(_)), p(x_4^6), [(v,4),(p,3),(r,4),(p,3),(r,4)])\}\\
AI^4_{NEG} = \quad & \{\dots\}\\
AI^5_{POS} = \quad & \{(inc_W, insertar(t(_)), p(x_1^7) \wedge q(x_1^7),\\
& \qquad [(t,2),(p,3),(r,4),(p,3),(r,4),(p,1)]),
\end{aligned}$$

$$(inc_W, insertar(v(\_)), p(x_1^8) \wedge q(x_1^8), \\ [(v,4),(p,3),(r,4),(p,3),(r,4),(p,1)])\}$$

$AI^5_{NEG} = \quad \{\dots\}$

$AI^6_{POS} = \emptyset$

$AI^6_{NEG} = \emptyset$

A partir de estos elementos se definen tres reglas restauradoras más. Las dos últimas son equivalentes a otras ya obtenidas (R_8 a R_5 y R_9 a R_6)*; la primera controla la situación que se había detectado:*

R_7:
`Evento:` $insertar(s(\_,\_))$
`Condición:` $p(x_1^6) \wedge q(x_1^6)$
`Acción:` ...

R_8:
`Evento:` $insertar(t(\_))$
`Condición:` $p(x_1^7) \wedge q(x_1^7)$
`Acción:` ...

R_9:
`Evento:` $insertar(v(\_))$
`Condición:` $p(x_1^8) \wedge q(x_1^8)$
`Acción:` ...

A continuación se presenta de nuevo el concepto de actualización inducida pero teniendo en cuenta la posible presencia de recursividad en las reglas deductivas. Estos conjuntos se definen inductivamente siguiendo las ideas de la definición 3 extendidas con la regla de parada de la definición 4.

Definición 5 *Actualizaciones Inducidas (AI) en bases de datos estratificadas.*

Los conjuntos de actualizaciones inducidas AI_{POS} y AI_{NEG} se definen a partir del conjunto de reglas deductivas de la base de datos y están formados por elementos que son cuádruplas de la forma (P, O, C_D, L) donde:

- P es un átomo derivado;
- O es una operación de la forma $insertar(Q)$ o $borrar(Q)$ donde Q es un átomo básico;
- C_D es una conjunción de literales o el predicado *cierto*; y
- L es una lista de la forma $[(q_1, Et_1), \dots, (q_{\tilde{n}}, Et_{\tilde{n}})]$ donde $\tilde{n} \geq 1$, Et_i $(1 \leq i \leq \tilde{n})$ es la etiqueta de una regla deductiva y q_i $(1 \leq i \leq \tilde{n})$ es un símbolo de predicado.

Los conjuntos de actualizaciones inducidas se definen como sigue:

$AI^0_{POS} =$

$\{(P, insertar(q(\overrightarrow{t})), L_1 \wedge \dots \wedge L_{i-1} \wedge L_{i+1} \wedge \dots \wedge L_m \text{ (resp. } cierto), [(q, Et)]) \mid$

$(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge q(\overrightarrow{t}) \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada cuya etiqueta es Et, q es un predicado básico y $m > 1$ (resp. $m = 1$)$\}$

$\cup$

$\{(P, borrar(q(\overrightarrow{t})), L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_m$ (resp. $cierto$), $[(q, Et)]) \mid$

$(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg q(\overrightarrow{t}) \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada cuya etiqueta es Et, q es un predicado básico y $m > 1$ (resp. $m = 1$)$\}$

$AI^0_{NEG} =$

$\{(P, insertar(q(\overrightarrow{t})), cierto, [(q, Et)]) \mid$

$(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg q(\overrightarrow{t}) \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada cuya etiqueta es Et y q es un predicado básico$\}$

$\cup$

$\{(P, borrar(q(\overrightarrow{t})), cierto, [(q, Et)]) \mid$

$(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge q(\overrightarrow{t}) \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada cuya etiqueta es Et y q es un predicado básico$\}$

$AI^{n+1}_{POS} = AI^{n+1}_{POS,1} \cup AI^{n+1}_{POS,2} \cup AI^{n+1}_{POS,3} \cup AI^{n+1}_{POS,4}$ siendo $n \geq 0$ y los subconjuntos $AI^{n+1}_{POS,i}$ $(1 \leq i \leq 4)$ los siguientes:

$AI^{n+1}_{POS,1} =$

$\{(P\theta, O(q'(\overrightarrow{t'}))\theta, (C_D \wedge L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_m)\theta, L + [(q, Et)])|$[14]

$(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge q(\overrightarrow{t}) \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada cuya etiqueta es Et; $(A, O(q'(\overrightarrow{t'})), C_D, L) \in AI^n_{POS}$; $\theta = mgu(A, q(\overrightarrow{t}))$; (q, Et) aparece como mucho una vez en L; y el predicado de P no aparece en los predicados de $L + [(q, Et)]\}$

$AI^{n+1}_{POS,2} =$

$\{(P\theta, O(q'(\overrightarrow{t'}))\theta, (L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg q(\overrightarrow{t}) \wedge L_{i+1} \wedge \ldots \wedge L_m)\theta, L + [(q, Et)]) \mid$

$(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg q(\overrightarrow{t}) \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada cuya etiqueta es Et; $(A, O(q'(\overrightarrow{t'})), cierto, L) \in AI^n_{NEG}$; $\theta = mgu(A, q(\overrightarrow{t}))$; (q, Et) aparece como mucho una vez en L; y el predicado de P no aparece en los predicados de $L + [(q, Et)]\}$

[14] Para representar el operador que concatena dos listas se ha utilizado el símbolo $+$.

$AI^{n+1}_{POS,3} =$

$\{(P\theta, O(q'(\overrightarrow{x})), P\theta, L + [(q, Et)]) \mid$

$(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge q(\overrightarrow{t}) \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada cuya etiqueta es Et; $(A, O(q'(\overrightarrow{t'})), C_D, L) \in AI^n_{POS}$; $\theta = mgu(A, q(\overrightarrow{t}))$; $\overrightarrow{x}$ es un vector de variables nuevas; (Et, q) aparece como mucho una vez en L; y el predicado de P aparece en los predicados de $L + [(q, Et)]\}$

$AI^{n+1}_{POS,4} =$

$\{(P\theta, O(q'(\overrightarrow{x})), P\theta, L + [(q, Et)]) \mid$

$(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg q(\overrightarrow{t}) \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada cuya etiqueta es Et; $(A, O(q'(\overrightarrow{t'})), cierto, L) \in AI^n_{NEG}$; $\theta = mgu(A, q(\overrightarrow{t}))$; $\overrightarrow{x}$ es un vector de variables nuevas; (q, Et) aparece como mucho una vez en L; y el predicado de P aparece en los predicados de $L + [(q, Et)]\}$

$AI^{n+1}_{NEG} = AI^{n+1}_{NEG,1} \cup AI^{n+1}_{NEG,2} \cup AI^{n+1}_{NEG,3} \cup AI^{n+1}_{NEG,4}$ siendo $n \geq 0$ y los subconjuntos $AI^{n+1}_{NEG,i}$ $(1 \leq i \leq 4)$ los siguientes:

$AI^{n+1}_{NEG,1} =$

$\{(P\theta, O(q'(\overrightarrow{t'}))\theta, cierto, L + [(q, Et)]) \mid$

$(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge q(\overrightarrow{t}) \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada cuya etiqueta es Et; $(A, O(q'(\overrightarrow{t'})), cierto, L) \in AI^n_{NEG}$; $\theta = mgu(A, q(\overrightarrow{t}))$; (q, Et) aparece como mucho una vez en L; y el predicado de P no aparece en los predicados de $L + [(q, Et)]\}$

$AI^{n+1}_{NEG,2} =$

$\{(P\theta, O(q'(\overrightarrow{t'}))\theta, cierto, L + [(q, Et)]) \mid$

$(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg q(\overrightarrow{t}) \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada cuya etiqueta es Et; $(A, O(q'(\overrightarrow{t'})), C_D, L) \in AI^n_{POS}$; $\theta = mgu(A, q(\overrightarrow{t}))$; (q, Et) aparece como mucho una vez en L; y el predicado de P no aparece en los predicados de $L + [(q, Et)]\}$

$AI^{n+1}_{NEG,3} =$

$\{(P\theta, O(q'(\overrightarrow{x})), cierto, L + [(q, Et)]) \mid$

$(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge q(\overrightarrow{t}) \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada cuya etiqueta es Et; $(A, O(q'(\overrightarrow{t'})), cierto, L) \in AI^n_{NEG}$; $\theta = mgu(A, q(\overrightarrow{t}))$; $\overrightarrow{x}$ es un vector de variables nuevas; (q, Et) aparece como mucho una vez en L; y el predicado de P aparece en los predicados de $L + [(q, Et)]\}$

$AI^{n+1}_{NEG,4} =$

$\{(P\theta, O(q'(\overrightarrow{x})), cierto, L + [(q, Et)]) \mid$

$(P \leftarrow L_1 \wedge \ldots \wedge L_{i-1} \wedge \neg q(\overrightarrow{t}) \wedge L_{i+1} \wedge \ldots \wedge L_m)$ es una regla deductiva refrescada cuya etiqueta es Et; $(A, O(q'(\overrightarrow{t'})), C_D, L) \in AI^n_{POS}$; $\theta = mgu(A, q(\overrightarrow{t}))$; $\overrightarrow{x}$ es un vector de variables nuevas; (q, Et) aparece como mucho una vez en L; y el predicado de P aparece en los predicados de $L + [(q, Et)]\}$

La generación de estos conjuntos termina en un j ($j \geq 0$) tal que $AI^j_{POS} = \emptyset$ y $AI^j_{NEG} = \emptyset$. La existencia de ese valor para j está asegurada por la regla de parada aplicada en la generación de los conjuntos. Una vez obtenidos todos estos conjuntos, las actualizaciones inducidas son:

$AI_{POS} = \cup_{n \geq 0} AI^n_{POS}$

$AI_{NEG} = \cup_{n \geq 0} AI^n_{NEG}$ ■

Como ya se ha visto en los ejemplos 48 y 50, la necesidad de recorrer cada ciclo del grafo al menos dos veces para asegurar que se controla cualquier violación supone la generación de actualizaciones inducidas *equivalentes* en el sentido de que las reglas restauradoras que se obtienen a partir de ellas detectan exactamente el mismo conjunto de violaciones. Esta propiedad se define más formalmente a continuación.

Definición 6 *Actualizaciones inducidas equivalentes.*

Sean $I_1 = (P, O, C_D, L)$ e $I_2 = (P', O', C'_D, L')$ dos elementos de AI_{POS}; se dice que I_1 e I_2 son equivalentes si existe una substitución α de variable por variable tal que $P\alpha = P'\alpha$ y $O\alpha = O'\alpha$ y $C_D\alpha = C'_D\alpha$.■

De todos los elementos de AI_{POS} sólo será necesario tener en cuenta, para la generación de las reglas restauradoras, un elemento por cada subconjunto de actualizaciones inducidas que sean equivalentes.

Las reglas restauradoras, aún sin acción, pueden generarse a partir del conjunto AI_{POS} obtenido de acuerdo con la definición 5 tal y como se muestra en el siguiente algoritmo. El algoritmo, en primer lugar, elimina las actualizaciones inducidas equivalentes generando un subconjunto de AI_{POS} que se ha denominado $AI^{reducido}_{POS}$; después define una regla restauradora para cada elemento (P, O, C_D, L) de ese conjunto que cumpla que el predicado de P es un predicado de inconsistencia.

Algoritmo 2 *Algoritmo de generación del evento y la condición de las reglas restauradoras en bases de datos estratificadas.*

```
ALGORITMO Generación_Reglas_Restauradoras_2
ENTRADA
    AI_POS : Conjunto de actualizaciones inducidas según la definición 5;
SALIDA
    ℜ : Conjunto de reglas restauradoras sin acción;
VARIABLES
    AI_POS^reducido : Conjunto de actualizaciones inducidas;
    i : Entero;
INICIO
ℜ := ∅;
i := 1;
AI_POS^reducido := ∅;
PARA CADA (P, O, C_D, L) ∈ AI_POS HACER
| SI no existe un (P', O', C'_D, L') en AI_POS^reducido y una substitución α de
| |    variable por variable tal que Pα = P'α y Oα = O'α y C_Dα = C'_Dα
| ENTONCES
| | AI_POS^reducido := AI_POS^reducido ∪ {(P, O, C_D, L)}
| FIN_SI
FIN_PARA;
PARA CADA (P, O, C_D, L) ∈ AI_POS^reducido HACER
| SI el predicado de P es el predicado de inconsistencia inc_W
| ENTONCES
| | ℜ := ℜ ∪ {R_i_inc_W(Evento: O, Condición: C_D, Acción:...)};
| | i := i + 1
| FIN_SI
FIN_PARA
FIN.■
```

Teorema 2 *Propiedad de los elementos de AI_{POS} en bases de datos estratificadas.*

⇒ Sean (P, O, C_D, L) un elemento de AI_{POS}, O' una operación de inserción o borrado de un hecho y D un estado de base de datos tal que:

- $\alpha = mgu(O, O')$,
- D' es el estado de base de datos obtenido al ejecutar O' en D, y
- existe una refutación $SLDNF$ para $D' \cup \{\leftarrow C_D\alpha\}$ con respuesta computada β,

⇒ entonces existe una refutación $SLDNF$ para $D' \cup \{\leftarrow P\alpha\beta\}$.

Demostración:

La demostración se va a realizar de nuevo por inducción sobre el nivel n del conjunto AI^n_{POS} al que pertenece el elemento (P, O, C_D, L).

Caso base: $n = 0$

Sea (P, O, C_D, L) un elemento de AI^0_{POS}. El caso base del teorema 2 coincide con el caso base del teorema 1 por lo que la propiedad queda demostrada.

Paso de inducción: $n > 0$

Si la propiedad se cumple para los elementos de AI^{n-1}_{POS} entonces se cumple para AI^{n}_{POS}. La demostración se realiza teniendo en cuenta a cuál de los cuatro subconjuntos $AI^{n}_{POS,i}$ $(1 \leq i \leq 4)$ pertenece el elemento:

- $(P, O, C_D, L) \in AI^{n}_{POS,1}$

  La definición de $AI^{n}_{POS,1}$ para bases de datos estratificadas coincide con la definición del conjunto $AI^{n}_{POS,1}$ para bases de datos jerárquicas (sin tener en cuenta el último elemento de la cuádrupla) por lo que teniendo en cuenta el teorema 1 la propiedad queda demostrada para los elementos de este subconjunto.

- $(P, O, C_D, L) \in AI^{n}_{POS,2}$

  La definición de $AI^{n}_{POS,2}$ para bases de datos estratificadas coincide con la definición del conjunto $AI^{n}_{POS,2}$ para bases de datos jerárquicas (sin tener en cuenta el último elemento de la cuádrupla) por lo que teniendo en cuenta el teorema 1 la propiedad queda demostrada para los elementos de este subconjunto.

- $(P, O, C_D, L) \in AI^{n}_{POS,3}$

  El elemento es de la forma $(P\theta, O(q'(\overrightarrow{x})), P\theta, L + [(q, Et)])$ por lo que la demostración es inmediata ya que $C_D = P\theta$ **(1)**.

  Si existe una refutación $SLDNF$ para $D' \cup \{\leftarrow C_D\alpha\}$ con respuesta computada β entonces, teniendo en cuenta la igualdad (1), existe una refutación $SLDNF$ para $D' \cup \{\leftarrow P\theta\alpha\beta\}$.

- $(P, O, C_D, L) \in AI^{n}_{POS,4}$

  Similar al caso anterior.■

Corolario 2 *Corrección del algoritmo de generación del evento y la condición de las reglas restauradoras en bases de datos estratificadas.*

⇒ Sean $R(Evento\colon O, Condición\colon C, Acción\colon \dots)$ una regla restauradora obtenida con el algoritmo 2 asociada a una restricción de integridad, W, cuyo predicado de inconsistencia es inc_W, O' una operación de inserción o borrado de un hecho y D un estado de base de datos tal que:

- $\alpha = mgu(O, O')$,
- D' es el estado de base de datos obtenido al ejecutar O' en D, y
- existe una refutación $SLDNF$ para $D' \cup \{\leftarrow C\alpha\}$,

⇒ entonces el estado D' viola la restricción de integridad W.

Demostración:

La demostración de este corolario es la misma que la del corolario 1.■

4.3.3 Obtención de la acción en base de datos jerárquicas

La acción de cada regla restauradora va a obtenerse a partir de su condición. Las acciones más sencillas que se pueden generar son las asociadas a condiciones que constan de un solo literal básico ya que en este caso la única operación posible se definirá sobre este predicado. Este caso se ilustra en el siguiente ejemplo.

Ejemplo 52 *Sea la siguiente regla restauradora:*

R:
Evento: $insertar(s(x))$
Condición: $t(x,z)$
Acción: ...

Si se supone que t es un predicado básico, de la condición de esa regla se puede deducir la transacción $\{borrar(t(x,z)\}$ podría restaurar la inconsistencia detectada por la regla:

R:
Evento: $insertar(s(x))$
Condición: $t(x,z)$
Acción: $\{borrar(t(x,z))\}$

Antes de presentar casos más complejos, para aclarar el funcionamiento de las reglas restauradoras durante el normal funcionamiento del sistema, en el ejemplo 53 se muestra cómo se procesaría la regla antes obtenida.

Ejemplo 53 *Sean la regla restauradora del ejemplo 52 y el siguiente conjunto de hechos $\{t(1,2),\ t(1,3),\ t(1,4)\}$. Supóngase que una transacción ejecuta la operación $insertar(s(1))$. Esta operación dispara la regla ya que unifica con su evento a través de la substitución $\{x/1\}$. Para determinar si se debe ejecutar la acción hay que evaluar la condición $t(1,z)$ en el nuevo estado. La condición se evalúa a cierto para las instancias $\{z/2\}$, $\{z/3\}$ y $\{z/4\}$ por lo que debe ejecutarse la acción. El conjunto de operaciones que se ejecutarán será: $\{borrar(t(1,2)),\ borrar(t(1,3)),\ borrar(t(1,4))\}$.*

Del ejemplo 53 debe concluirse que la acción de una regla restauradora debe ejecutarse para cada una de las instancias que se obtengan en la evaluación de la condición.

Para estudiar cómo obtener la acción en un caso más complejo hay que tener en cuenta que debido a la posible presencia de literales derivados en la condición, puede no ser suficiente con una simple operación para conseguir que la condición pase a ser falsa, por lo que hay que considerar que la acción va a ser una transacción que puede incluir varias operaciones sobre predicados básicos.

A continuación se presenta cómo se van a obtener y representar estas acciones; en primer lugar se supondrá que en las reglas deductivas que no son de inconsistencia no aparecen variables existencialmente cuantificadas[15] y más adelante se contemplará esta posibilidad.

[15] El problema que supone la presencia de este tipo de variables se ilustró en el apartado 3.1 concretamente en el ejemplo 3.

Acciones generadas en presencia de reglas deductivas sin variables existencialmente cuantificadas

En los siguientes ejemplos se muestra cómo se van a obtener y a representar las transacciones que constituyen la acción de las reglas restauradoras en el caso de que en las reglas deductivas que no son de inconsistencia no aparezcan variables existencialmente cuantificadas. En estos ejemplos, la condición de la regla consta nada más de un literal. Como se verá más adelante, sin embargo, las ideas presentadas se extienden fácilmente a condiciones con cualquier número de literales.

Ejemplo 54 *Sea el siguiente conjunto de reglas deductivas:*

$$1 : inc_W \leftarrow q(x) \wedge \neg p(x)$$
$$2 : p(x) \leftarrow s(x) \wedge \neg v(x)$$
$$3 : p(x) \leftarrow t(x) \wedge u(x)$$

Y considérese la siguiente regla restauradora:

R:
Evento: $insertar(q(x))$
Condición: $\neg p(x)$
Acción: ...

En este caso la condición sólo contiene un literal derivado negativo, por lo que la acción debe restaurar la consistencia insertando una instancia de $p(x)$. Considerando la regla deductiva 2 por ejemplo, podría inducirse la inserción de una instancia de $p(x)$ insertando $s(x)$ (si es falso) y borrando después $v(x)$ (si es cierto). Teniendo en cuenta todas las reglas deductivas y todos los órdenes posibles entre los literales del cuerpo de cada regla, las transacciones posibles que permitirían la inserción de instancias de $p(x)$ serían las siguientes:

```
Transacción T_Insertar_p_1(x)
Inicio
| SI ¬s(x)
| ENTONCES
| | insertar(s(x));
| | SI v(x)
| | ENTONCES
| | | borrar(v(x))
| | FIN_SI
| FIN_SI
Fin_Transacción.

Transacción T_Insertar_p_2(x)
Inicio
| SI v(x)
| ENTONCES
| | borrar(v(x));
| | SI ¬s(x)
| | ENTONCES
| | | insertar(s(x))
| | FIN_SI
| FIN_SI
```

Fin_Transacción.

Transacción $T\_Insertar\_p\_3(x)$
Inicio
| SI $\neg t(x)$
| ENTONCES
| | $insertar(t(x))$;
| | SI $\neg u(x)$
| | ENTONCES
| | | $insertar(u(x))$
| | FIN_SI
| FIN_SI
Fin_Transacción.

Transacción $T\_Insertar\_p\_4(x)$
Inicio
| SI $\neg u(x)$
| ENTONCES
| | $insertar(u(x))$;
| | SI $\neg t(x)$
| | ENTONCES
| | | $insertar(t(x))$
| | FIN_SI
| FIN_SI
Fin_Transacción.

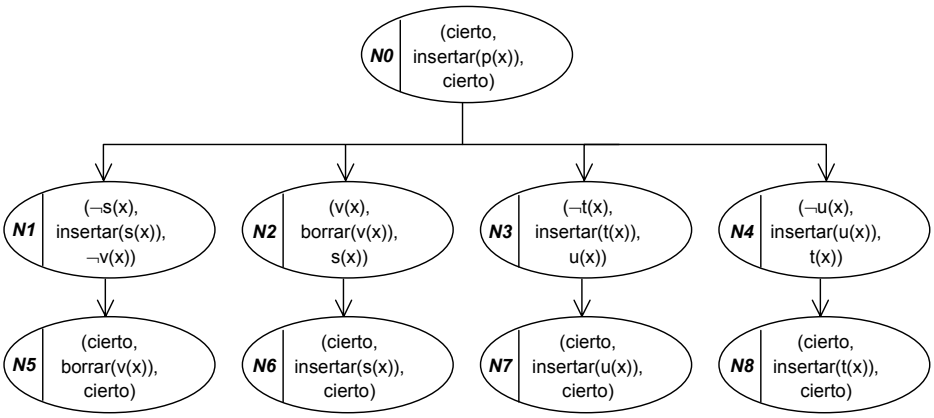
Es fácil darse cuenta de que tanto las dos primeras transacciones como las dos últimas podrían haberse integrado en una sola:

Transacción $T\_Insertar\_p\_1\_y\_2(x)$
Inicio
| SI $\neg s(x)$
| ENTONCES
| | $insertar(s(x))$;
| FIN_SI;
| SI $v(x)$
| ENTONCES
| | $borrar(v(x))$
| FIN_SI
Fin_Transacción.

Transacción $T\_Insertar\_p\_3\_y\_4(x)$
Inicio
| SI $\neg t(x)$
| ENTONCES
| | $insertar(t(x))$;
| FIN_SI;
| SI $\neg u(x)$
| ENTONCES
| | $insertarr(u(x))$
| FIN_SI
Fin_Transacción.

Pero teniendo en cuenta la estructura que se ha diseñado para representarlas y el algoritmo de procesamiento, resulta más sencillo considerar las transacciones inicialmente presentadas.

La estructura diseñada para representar las transacciones es un árbol, tal y como se muestra en la figura siguiente, en el que por claridad se ha incluido un nodo raíz cuya operación se deriva directamente del literal de la condición:



En un árbol como el del ejemplo, cada camino desde la raíz hasta una hoja representa una transacción que satisface la operación de inserción del nodo raíz:

- $T\_Insertar\_p\_1$ *se corresponde con el camino* $N0 - N1 - N5$
- $T\_Insertar\_p\_2$ *se corresponde con el camino* $N0 - N2 - N6$
- $T\_Insertar\_p\_3$ *se corresponde con el camino* $N0 - N3 - N7$
- $T\_Insertar\_p\_4$ *se corresponde con el camino* $N0 - N4 - N8$

Ejemplo 55 *Sea el siguiente conjunto de reglas deductivas:*

$$1 : inc_W \leftarrow q(x) \wedge p(x)$$
$$2 : p(x) \leftarrow s(x) \wedge \neg v(x)$$
$$3 : p(x) \leftarrow t(x) \wedge u(x)$$

Y la siguiente regla restauradora:

R:

`Evento:`	$insertar(q(x))$
`Condición:`	$p(x)$
`Acción:`	...

En este caso la condición sólo contiene un literal derivado positivo, por lo que la acción debe restaurar la consistencia borrando una instancia de $p(x)$ lo que significa eliminar todos los caminos por los que sea derivable. Teniendo en cuenta todas las reglas deductivas, las transacciones posibles que permitirían el borrado de instancias de $p(x)$ serían las siguientes:

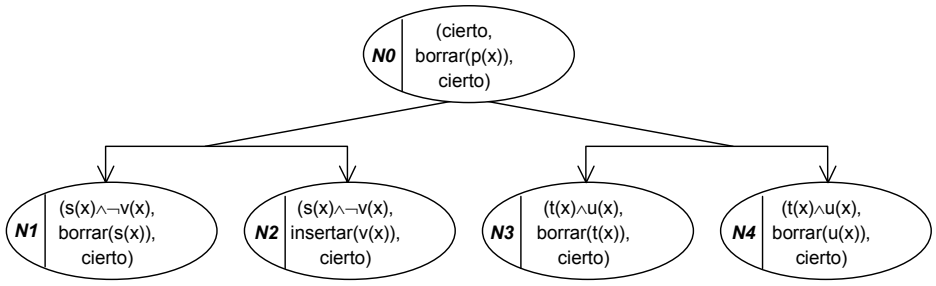
Transacción $T\_Borrar\_p\_1(x)$
Inicio
| SI $s(x) \wedge \neg v(x)$
| ENTONCES
| | $borrar(s(x))$
| FIN_SI;
| SI $t(x) \wedge u(x)$
| ENTONCES
| | $borrar(t(x))$
| FIN_SI
Fin_Transacción.

Transacción $T\_Borrar\_p\_2(x)$
Inicio
| SI $s(x) \wedge \neg v(x)$
| ENTONCES
| | $insertar(v(x))$
| FIN_SI;
| SI $t(x) \wedge u(x)$
| ENTONCES
| | $borrar(t(x))$
| FIN_SI
Fin_Transacción.

Transacción $T\_Borrar\_p\_3(x)$
Inicio
| SI $s(x) \wedge \neg v(x)$
| ENTONCES
| | $borrar(s(x))$
| FIN_SI;
| SI $t(x) \wedge u(x)$
| ENTONCES
| | $borrar(u(x))$
| FIN_SI
Fin_Transacción.

Transacción $T\_Borrar\_p\_4(x)$
Inicio
| SI $s(x) \wedge \neg v(x)$
| ENTONCES
| | $insertar(v(x))$
| FIN_SI;
| SI $t(x) \wedge u(x)$
| ENTONCES
| | $borrar(u(x))$
| FIN_SI
Fin_Transacción.

Representando también estas transacciones en forma de árbol se tiene:



En este caso, del nodo raíz parten tantos arcos como reglas deductivas existen con el predicado p en la cabeza. Cada camino desde la raíz hasta una hoja permite eliminar un camino de derivación para $p(x)$. Para asegurar que el borrado es definitivo es necesario recorrer un camino para cada arco que parte de la raíz. Así pues en este ejemplo, cada transacción está representada en dos caminos, uno por cada arco que parte del nodo raíz:

- $T\_Borrar\_p\_1$ *se corresponde con los caminos* $N0 - N1$ *y* $N0 - N3$
- $T\_Borrar\_p\_2$ *se corresponde con los caminos* $N0 - N2$ *y* $N0 - N3$
- $T\_Borrar\_p\_3$ *se corresponde con los caminos* $N0 - N1$ *y* $N0 - N4$
- $T\_Borrar\_p\_4$ *se corresponde con los caminos* $N0 - N2$ *y* $N0 - N4$

De los ejemplos anteriores se concluye que a partir del literal positivo (resp. negativo) de la condición de una regla se va a construir un árbol en el que se representarán todas las posibles transacciones que puedan suponer el borrado (resp. la inserción) de instancias de ese literal. A partir de ahora a un árbol de este tipo se le denominará *T-árbol* (*Transacción en árbol*). Esta nueva estructura se presenta a continuación.

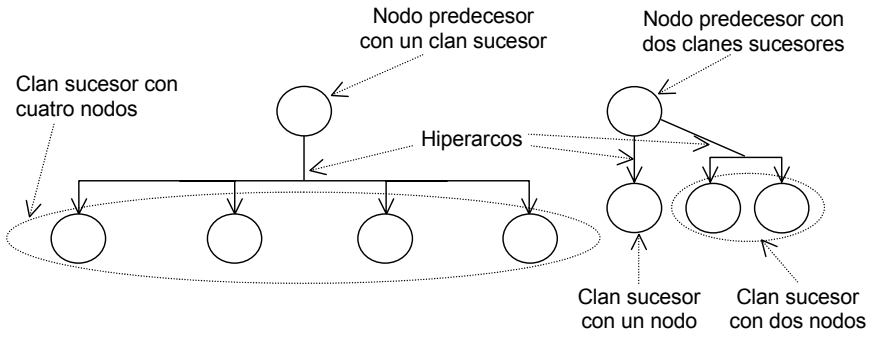
Estructura *T*-árbol.

Un T-árbol es una estructura en la que:

a) Los nodos son tripletes (C_A, O, C_D) donde C_A y C_D son el predicado *cierto* o una conjunción de literales, llamadas respectivamente *Condición-Antes* y *Condición-Después*[16] y O es la operación *insertar*(A) o *borrar*(A), donde A es un átomo, o la operación especial *abortar*.

b) Los nodos se relacionan mediante *hiperarcos* que conectan un nodo *predecesor* con un conjunto de nodos que se denomina *clan sucesor*.

[16] Se ha elegido el mismo nombre para esta condición que el de la condición generada para la obtención del evento y la condición de las reglas restauradoras ya que como se verá más adelante estas dos condiciones se construyen estudiando qué debe ser cierto antes y después de la ejecución de la operación del nodo.



Cada hoja de un T-árbol distinta del nodo raíz debe cumplir dos propiedades:

1. La operación es sobre un predicado básico, y
2. La Condición-Después es el predicado *cierto*.

La ejecución de la acción de una regla restauradora consiste en el procesamiento (recorrido) de su T-árbol en tiempo de ejecución. En este contexto, el recorrido de un T-árbol no significa, como en el caso clásico de recorrido de un árbol, la visita de todos los nodos de éste, sino la visita de los nodos necesarios para construir una transacción restauradora. Este recorrido se hará con los siguientes criterios:

1. Después de visitar un nodo, se debe intentar recorrer, para cada uno de sus clanes sucesores, un T-árbol que tenga como nodo raíz a uno de los elementos del clan.
2. A partir de un nodo, se podrá elegir cualquier nodo sucesor cuya Condición-Antes sea cierta.
3. El recorrido de un camino termina en un nodo con la operación sobre un predicado básico cuya Condición-Después sea cierta.

Aunque más adelante se presentarán estas ideas más formalmente, una propiedad importante de los T-árboles es la de que en su recorrido nunca habrá que retroceder porque se haya alcanzado un punto en el que la restauración no es posible. Esto se puede asegurar ya que un nodo sólo se puede visitar si su Condición-Antes es cierta lo que asegura que el camino que se inicia en ese nodo es bueno para la restauración. En el siguiente ejemplo se muestra el recorrido de los árboles de los ejemplos anteriores.

Ejemplo 56 *En el árbol del ejemplo 54 del nodo raíz parte un hiperarco que le conecta con un clan que tiene cuatro nodos. Supóngase que la base de datos contiene los siguientes hechos:* $\{s(1), v(1)\}$ *y que el usuario ejecuta una transacción que incluye la operación* $insertar(q(1))$*. Esta operación dispara la regla* R *que resulta ser relevante ya que su condición* $\neg p(1)$ *se satisface. Como ya se ha comentado, la acción de esa regla está representada en un* $T-$*árbol que incluye cuatro transacciones. El recorrido del árbol se inicia en el nodo* $N0$ (*cuya Condición-Antes es cierta*)*. A partir de este nodo hay que recorrer un camino de su único clan. El camino que sigue en* $N1$ *no es elegible ya que no se cumple su Condición-Antes* ($\neg s(1)$ *es falso en la base de datos*)*. Véanse los otros caminos:*

- *Camino* $N0 - N2 - N6$*:*

  - *Nodo* $N2$:
    $\Longrightarrow$*Condición-Antes* $= v(1)$ *es cierta* $\Longrightarrow$ $borrar(v(1))$
    $\Longrightarrow$*Condición-Después* $= s(1)$ *es cierta* $\Longrightarrow$ *Terminar*

  Este camino restaura la consistencia borrando $v(1)$ *con lo que se induce la inserción de* $p(1)$ *a través de la regla deductiva 1.*

- *Camino* $N0 - N3 - N7$*:*

  - *Nodo* $N3$:
    $\Longrightarrow$*Condición-Antes* $= \neg t(1)$ *es cierta* $\Longrightarrow$ $insertar(t(1))$
    $\Longrightarrow$*Condición-Después* $= u(1)$ *es falsa* $\Longrightarrow$ *Seguir*
  - *Nodo* $N7$:
    $\Longrightarrow$*Condición-Antes* $=$ *cierto* $\Longrightarrow$ $insertar(u(1))$
    $\Longrightarrow$*Condición-Después* $=$ *cierto* $\Longrightarrow$ *Terminar*

  Este camino restaura la consistencia insertando $t(1)$ *y* $u(1)$ *con lo que se induce la inserción de* $p(1)$ *a través de la regla deductiva 2.*

- *Camino* $N0 - N4 - N8$*:*

  - *Nodo* $N4$:
    $\Longrightarrow$*Condición-Antes* $= \neg u(1)$ *es cierta* $\Longrightarrow$ $insertar(u(1))$
    $\Longrightarrow$*Condición-Después* $= t(1)$ *es falsa* $\Longrightarrow$ *Seguir*
  - *Nodo* $N8$:
    $\Longrightarrow$*Condición-Antes* $=$ *cierto* $\Longrightarrow$ $insertar(t(1))$
    $\Longrightarrow$*Condición-Después* $=$ *cierto* $\Longrightarrow$ *Terminar*

  Este camino restaura la consistencia insertando $u(1)$ *y* $t(1)$ *con lo que inserta* $p(1)$ *a través de la regla deductiva 2.*

Puede observarse que los dos últimos caminos representan transacciones restauradoras que sólo difieren en el orden en el que se realizan las operaciones, por lo que podrían parecer redundantes. No es así sin embargo, dada la forma en la que se recorre el árbol. Supóngase por ejemplo que la base de datos contiene sólo un hecho, $\{u(2)\}$*, y que el usuario ejecuta una transacción que contiene la operación* $insertar(q(2))$*. Esta operación dispara la regla* R *que resulta ser relevante ya que su condición* $\neg p(2)$ *se satisface. En este caso los únicos caminos posibles para restaurar la consistencia son:* $N0 - N1 - N5$*,* $N0 - N2 - N6$ *y* $N0 - N3 - N7$ *no siendo elegible el camino* $N0 - N4 - N8$ *ya que no se cumple su Condición-Antes* ($\neg u(2)$ *es falso en la base de datos*).

En el árbol del ejemplo 55*, del nodo raíz parten dos hiperarcos que le conectan con dos clanes cada uno con dos nodos. Supóngase que la base de datos contiene los siguientes hechos:* $\{s(1), v(1), t(1), u(1)\}$ *y que el usuario ejecuta una transacción que*

contiene la operación insertar$(q(1))$*. Esta operación dispara la regla R que resulta ser relevante ya que su condición* $p(1)$ *se satisface. El recorrido del árbol se inicia en el nodo* $N0$ (*cuya Condición-Antes es cierta*)*. A partir de este nodo hay que recorrer, si es posible, un camino para cada uno de sus dos clanes. Si no es posible encontrar el camino es porque no es necesario recorrer el clan. Para el primer clan no hay camino posible ya que la Condición-Antes no se satisface ni para* $N1$ *ni para* $N2$ (*lo que es razonable ya que* $p(1)$ *no es derivable por la regla deductiva que generó ese clan*)*. Para el segundo clan cualquier nodo es elegible.*

- *Camino* $N0 - N3$*:*
  - *Nodo* $N3$:
    $\Longrightarrow$*Condición-Antes* $= t(1) \wedge u(1)$ *es cierta* $\Longrightarrow$ *borrar*$(t(1))$
    $\Longrightarrow$*Condición-Después* $=$ *cierto* $\Longrightarrow$ *Terminar*

  Este camino restaura la consistencia borrando $t(1)$ *con lo que elimina* $p(1)$ *que era derivable a través de la regla deductiva 2.*

- *Camino* $N0 - N4$*:*
  - *Nodo* $N4$:
    $\Longrightarrow$*Condición-Antes* $= t(1) \wedge u(1)$ *es cierta* $\Longrightarrow$ *borrar*$(u(1))$
    $\Longrightarrow$*Condición-Después* $=$ *cierto* $\Longrightarrow$ *Terminar*

  Este camino restaura la consistencia borrando $u(1)$ *con lo que elimina* $p(1)$ *que era derivable a través de la regla deductiva 2.*

Así pues, la acción de cada regla restauradora es un T-árbol que se obtiene a partir de su condición como se muestra a continuación.

Sea $L_1 \wedge \ldots \wedge L_n$ la condición de la regla donde $n \geq 1$, entonces:

- Si $n = 1$ y $L_1 =$ *cierto*: el T-árbol asociado consta de un solo nodo que es (*cierto*, *abortar*, *cierto*).
- Si $n = 1$ y L_1 es un literal positivo básico: el T-árbol asociado consta de un solo nodo que es (*cierto*, *borrar*(L_1), *cierto*).
- Si $n = 1$ y L_1 es un literal negativo básico, $L_1 = \neg A_1$: el T-árbol asociado consta de un solo nodo que es (*cierto*, *insertar*(A_1), *cierto*).
- Si $n = 1$ y L_1 es un literal positivo derivado: el T-árbol se obtendrá a partir del nodo raíz (*cierto*, *borrar*(L_1), *cierto*).
- Si $n = 1$ y L_1 es un literal negativo derivado, $L_1 = \neg A_1$: el T-árbol se obtendrá a partir del nodo raíz (*cierto*, *insertar*(A_1), *cierto*).
- Si $n > 1$: el T-árbol se obtendrá a partir del nodo raíz (*cierto*, *borrar* ($Cond(x_1, \ldots, x_m)$), *cierto*) donde $\{x_1, \ldots, x_m\}$ son todas las variables que aparecen en $L_1 \wedge \ldots \wedge L_n$ y *Cond* es un nuevo símbolo de predicado que se define por la regla deductiva auxiliar $Cond(x_1, \ldots, x_m) \leftarrow L_1 \wedge \ldots \wedge L_n$[17].

[17] La inclusión de este predicado y de esta regla deductiva permiten tratar el caso de una condición con más de un literal como si sólo tuviera uno.

En los tres últimos casos debe generarse el árbol completo a partir de la raíz usando las reglas deductivas. Antes de presentar más formalmente cómo obtener el árbol es necesario introducir algunos conceptos previos.

Definición 7 *Variable marcada.*

Una variable marcada es aquélla que se trata como una constante en el proceso de unificación y se va a destacar con un *. Si L es un literal, con L^* se representa al mismo literal en el que se han marcado todas las variables.■

Definición 8 *Substitución restringida.*

Una substitución restringida θ es una substitución de la forma $\{v_1/t_1, \ldots, v_n/t_n\}$ donde v_i es una variable que no está marcada y cada término t_i es una constante o una variable marcada.■

Definición 9 *T-árbol en bases de datos jerárquicas.*

Un nodo de un T-árbol es un triplete de la forma (C_A, Op, C_D) donde C_A y C_D son el predicado *cierto* o una conjunción de literales y Op es $O(A)$ donde O es la operación *insertar* o *borrar* y A es un átomo, o bien Op es la operación especial *abortar*.

Dado un nodo, (C_A, Op, C_D) de un T-árbol, sus clanes sucesores se definen por las siguientes reglas:

1. Si $Op = abortar$, entonces el nodo no tiene clanes sucesores (i.e. es una hoja).
2. Si $Op = O(A)$, A es un átomo básico y $C_D = cierto$, entonces el nodo no tiene clanes sucesores (i.e. es una hoja).
3. Si $Op = O(A)$, A es un átomo básico y $C_D = M$, donde M es un literal, entonces el nodo tiene un clan sucesor, C, con un solo nodo generado por el literal M de la forma:
   (a) Si $M = \neg B$ entonces $C = \{(cierto, borrar(B), cierto)\}$
   (b) Si $M = B$ entonces $C = \{(cierto, insertar(B), cierto)\}$
4. Si $Op = O(A)$, A es un átomo básico y $C_D = A_1 \wedge \ldots \wedge A_n \wedge \neg B_1 \wedge \ldots \wedge \neg B_m$ donde $n + m > 1$ y tanto A_i $(1 \leq i \leq n)$ como B_j $(1 \leq j \leq m)$ son átomos entonces el nodo tiene sólo un clan sucesor, sea C, con $n + m$ nodos:
   $C = \{(\neg A_i, insertar(A_i), A_1 \wedge \ldots \wedge A_{i-1} \wedge A_{i+1} \wedge \ldots \wedge A_n \wedge \neg B_1 \wedge \ldots \wedge \neg B_m) | (1 \leq i \leq n)\} \cup \{(B_j, borrar(B_j), A_1 \wedge \ldots \wedge A_n \wedge \neg B_1 \wedge \ldots \wedge \neg B_{j-1} \wedge \neg B_{j+1} \wedge \ldots \wedge \neg B_m) | (1 \leq j \leq m)\}$.
5. Si $Op = O(A)$, A es un átomo derivado y hay p $(p \geq 1)$ reglas deductivas (refrescadas) de la forma $H_k \leftarrow A_{k,1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,m_k}$ $(1 \leq k \leq p$, $n_k + m_k \geq 1$ y tanto $A_{k,i}$ $(1 \leq i \leq n_k)$ como $B_{k,j}$ $(1 \leq j \leq m_k)$ son átomos) cuyas cabezas unifican con A^* a través de las substituciones restringidas θ_k, entonces:

(a) Si $O = insertar$ el nodo tiene sólo un clan sucesor C. El clan C tiene exactamente $\sum_{1\leq k\leq p}(n_k + m_k)$ nodos:
$C = \{(\neg A_{k,i}\theta_k, insertar(A_{k,i}\theta_k), (C_D \wedge A_{k,1} \wedge \ldots \wedge A_{k,i-1} \wedge A_{k,i+1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,m_k})\theta_k) \mid (1 \leq k \leq p)$ y $(1 \leq i \leq n_k)\}$ $\cup$
$\{(B_{k,j}\theta_k, borrar(B_{k,j}\theta_k), (C_D \wedge A_{k,1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,j-1} \wedge \neg B_{k,j+1} \wedge \neg B_{k,m_k})\theta_k) \mid (1 \leq k \leq p)$ y $(1 \leq j \leq m_k)\}$.

(b) Si $O = borrar$ entonces el nodo tiene p clanes sucesores, C_k $(1 \leq k \leq p)$. El clan C_k tiene $(n_k + m_k)$ nodos:
$C_k = \{((A_{k,1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,m_k})\theta_k, borrar(A_{k,i}\theta_k), C_D\theta_k) \mid (1 \leq i \leq n_k)\}$ $\cup$
$\{((A_{k,1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,m_k})\theta_k,\ insertar(B_{k,j}\theta_k), C_D\theta_k) \mid (1 \leq j \leq m_k)\}$.

6. Si $Op = O(A)$, A es un átomo derivado y no hay ninguna regla deductiva cuya cabeza unifica con A^* entonces el nodo tiene un clan sucesor, C, con un solo un nodo: $C = \{(cierto, abortar, cierto)\}$.■

Es evidente que los T-árboles (de acuerdo con la definición 9) en el caso de bases de datos jerárquicas son finitos.

Teniendo en cuenta la última regla incluida en la definición 9, algunos de los caminos de un T-árbol pueden terminar en un nodo cuya operación sea *abortar*. Dado que estos caminos no son capaces de restaurar la consistencia de la base de datos, van a ser eliminados del T-árbol mediante un proceso de depuración que se presenta en el apartado 4.3.5. Tras esta depuración, la presencia de la operación especial *abortar* sólo se permitirá en el nodo raíz de un T-árbol. A partir de este punto se supondrá que un T-árbol ha sido depurado para eliminar cualquier camino que incluya la operación *abortar*. En el siguiente ejemplo se ilustra esta situación.

Ejemplo 57 *Sea el siguiente conjunto de reglas deductivas donde el predicado s es un predicado derivado pero no hay reglas deductivas que lo definan*[18]*:*

$1 : inc_W \leftarrow q(x) \wedge \neg p(x)$
$2 : p(x) \leftarrow s(x) \wedge t(x)$

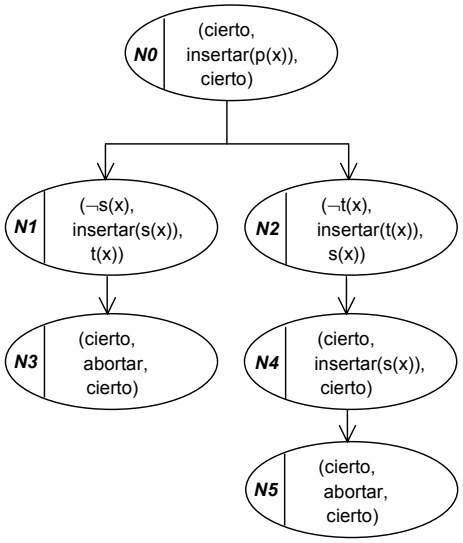
Y sea la siguiente regla restauradora:

R:

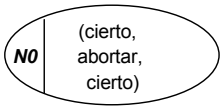
`Evento:`	$insertar(q(x))$
`Condición:`	$\neg p(x)$
`Acción:`	...

La acción de esta regla está representada en el siguiente árbol:

[18] Otra posible forma de abordar este problema habría sido prohibir esta situación por no considerarla "razonable"; es decir se exigiría que para todo predicado derivado hubiera al menos una regla deductiva que lo definiera.



Ninguno de los dos caminos de este T-árbol es capaz de restaurar la consistencia ya que ambos incluyen un nodo con la operación abortar. Tras el proceso de depuración, el T-árbol de esta regla será el siguiente:



El recorrido de un T-árbol generado según la definición 9, en el supuesto de que no aparezcan variables existencialmente cuantificadas en las reglas deductivas que no son de inconsistencia, se realiza con el algoritmo recursivo que se presenta a continuación. Entre los parámetros de entrada de este algoritmo se incluyen el estado de base de datos que había antes de la ejecución de la transacción del usuario, $D_{inicial}$, (éste es el estado que devuelve el algoritmo si la restauración no es posible) y el estado de base de datos que se obtiene tras la aplicación de esta transacción ,D.

Algoritmo 3 *Algoritmo de recorrido de un T-árbol en bases de datos jerárquicas (versión 1).*

```
ALGORITMO Recorrer_T-Árbol_1
ENTRADA
    (C_A^0, O^raíz, C_D^0) : Raíz de un T-árbol;
    D : Estado de base de datos;
    D_inicial : Estado inicial de base de datos;
SALIDA
    D' : Estado de base de datos;
VARIABLES
    C : Clan sucesor;
INICIO /*Se visita el nodo raíz*/
SI O^raíz = O^0(A^0)
```

```
ENTONCES
| SI el predicado de A^0 es derivado
| ENTONCES
| | PARA CADA clan sucesor de la raíz, C, HACER
| | | Elegir un nodo de C, sea (C_A^1, O^1(A^1), C_D^1) tal que C_A^1
| | |     sea cierta en D;
| | | Recorrer_T-Árbol_1((C_A^1, O^1(A^1), C_D^1), D, D_inicial, D')
| | FIN_PARA
| SI NO /*El predicado de A^0 es básico
| |       y el nodo tiene un solo clan sucesor*/
| | SI O^0 = insertar
| | ENTONCES
| | | D' := D ∪ {A^0}
| | SI NO /*O^0 = borrar*/
| | | D' := D − {A^0}
| | FIN_SI;
| | SI C_D^0 es cierta en D'
| | ENTONCES
| | | Parar
| | SI NO
| | | D := D';
| | | Elegir un nodo del clan sucesor de la raíz, sea
| | |     (C_A^1, O^1(A^1), C_D^1) tal que C_A^1 sea cierta en D;
| | | Recorrer_T-Árbol_1((C_A^1, O^1(A^1), C_D^1), D, D_inicial, D')
| | FIN_SI
| FIN_SI
SI NO    /*O^raíz = abortar*/
| D' := D_inicial;
FIN_SI
FIN.■
```

En este algoritmo es importante tener en cuenta que el átomo de la operación del nodo de entrada (si ésta es distinta de la operación *abortar*) es siempre un átomo base en tiempo de ejecución; esto es debido a que, dado que no hay variables existencialmente cuantificadas en las reglas deductivas, todas las variables que aparecen en un nodo de un T-árbol aparecen en su nodo raíz y ese nodo es instanciado, si no es base, por la evaluación de la condición de la regla restauradora. En caso de que el nodo raíz del T-árbol contenga la operación *abortar*, el estado de base de datos devuelto por el algoritmo es el estado (inicial) que había antes del inicio de la transacción de usuario. También es importante destacar que el recorrido de un camino de un T-árbol termina cuando se alcanza un nodo cuya operación es sobre un predicado básico y cuya Condición-Después se evalúa a cierto en el estado posterior a la ejecución de su operación.

Teorema 3 *Corrección del algoritmo de recorrido de un T-árbol en bases de datos jerárquicas (versión 1).*

⇒ Sea D una base de datos jerárquica en cuyas reglas deductivas que no son de inconsistencia no aparecen variables existencialmente cuantificadas, y sea D' la salida de la llamada al algoritmo *Recorrer_T-Árbol_1*$((C_A^0,$ *insertar*$(A^0), C_D^0)$,

D, $D_{inicial}$, D') (resp. *Recorrer_T-Árbol_1*$((C_A^0, borrar(A^0), C_D^0), D, D_{inicial}, D'))$ donde A^0 es base y C_A^0 es cierto en D,

$\Rightarrow$ entonces se cumple que $comp(D') \models A^0$ (resp. $comp(D') \models \neg A^0$).

Demostración:
Para demostrar este teorema, primero se establece la equivalencia entre cada paso del recorrido de un T-árbol según el algoritmo 3 y un paso de derivación $SLDNF$.

Paso 1:

Sea $N^n = (C_A^n, insertar(A^n), C_D^n)$ un nodo de un T-árbol, que no es hoja, donde A^n es un átomo básico base y sea A^{n+1} (resp. $\neg A^{n+1}$) un literal de C_D^n siendo C'_D el resto de la conjunción (que puede ser el predicado *cierto*); entonces un nodo sucesor en el recorrido es de la forma $N^{n+1} = (C_A^{n+1}, insertar(A^{n+1}), C'_D)$ (resp. $N^{n+1} = (C_A^{n+1}, borrar(A^{n+1}), C'_D)$.

Por el algoritmo 3, el recorrido del arco $N^n - N^{n+1}$ representa el paso de derivación $SLDNF$ entre el par (G^n: $\leftarrow A^n \wedge C_D^n$, L^n: A^n)[19] y (G^{n+1}: $\leftarrow A^{n+1} \wedge C'_D$, L^{n+1}: A^{n+1}) (resp. (G^{n+1}: $\leftarrow \neg A^{n+1} \wedge C'_D$, L^{n+1}: $\neg A^{n+1}$)) con cláusula de entrada A^n ($A^n \in D'$ porque es insertado por el algoritmo de recorrido).

Paso 2:

Sea $N^n = (C_A^n, insertar(A^n), C_D^n)$ un nodo de un T-árbol donde A^n es un átomo derivado base; entonces un nodo sucesor en el recorrido es de la forma $N^{n+1} = (C_A^{n+1}, insertar(A^{n+1}), C'_D)$ (resp. $N^{n+1} = (C_A^{n+1}, borrar(A^{n+1}), C'_D)$ donde $A^n \leftarrow L_1 \wedge \ldots \wedge L_i \wedge \ldots \wedge L_m$ es la regla deductiva que define A^n utilizada en la creación del nodo N^{n+1}, $L_i = A^{n+1}$ (resp. $L_i = \neg A^{n+1}$) y $C'_D = L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_m \wedge C_D^n$.

Por el algoritmo 3, el recorrido del arco $N^n - N^{n+1}$ representa el paso de derivación $SLDNF$ entre el par (G^n: $\leftarrow A^n \wedge C_D^n$, L^n: A^n) y (G^{n+1}: $\leftarrow A^{n+1} \wedge C'_D$, L^{n+1}: A^{n+1}) (resp. (G^{n+1}: $\leftarrow \neg A^{n+1} \wedge C'_D$, L^{n+1}: $\neg A^{n+1}$)) con cláusula de entrada $A^n \leftarrow L_1 \wedge \ldots \wedge L_i \wedge \ldots \wedge L_m$ y siendo $L_i = A^{n+1}$ (resp. $L_i = \neg A^{n+1}$) y $C'_D = L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_m \wedge C_D^n$.

Paso 3:

Sea $N^n = (C_A^n, borrar(A^n), C_D^n)$ un nodo de un T-árbol, que no es hoja, donde A^n es un átomo básico base y sea A^{n+1} (resp. $\neg A^{n+1}$) un literal de C_D^n siendo C'_D el resto de la conjunción (que puede ser el predicado *cierto*); entonces un nodo sucesor en el recorrido es de la forma $N^{n+1} = (C_A^{n+1}, insertar(A^{n+1}), C'_D)$ (resp. $N^{n+1} = (C_A^{n+1}, borrar(A^{n+1}), C'_D)$.

Por el algoritmo 3, el recorrido del arco $N^n - N^{n+1}$ representa el paso de derivación $SLDNF$ entre el par (G^n: $\leftarrow \neg A^n \wedge C_D^n$, L^n: $\neg A^n$) y (G^{n+1}: $\leftarrow A^{n+1} \wedge C'_D$, L^{n+1}: A^{n+1}) (resp. (G^{n+1}: $\leftarrow \neg A^{n+1} \wedge C'_D$, L^{n+1}: $\neg A^{n+1}$)) ya que existe un árbol $SLDNF$ fallado finitamente para $D' \cup \{\leftarrow A^n\}$ porque $A^n \notin D'$ (A^n ha sido borrado por el algoritmo de recorrido).

[19] G^i es el objetivo del paso de derivación y L^i el literal seleccionado en ese paso.

Paso 4:

Sea $N^n = (C_A^n, borrar(A^n), C_D^n)$ un nodo de un T-árbol donde A^n es un átomo derivado base definido por p reglas deductivas; entonces N^n tiene p nodos sucesores en el recorrido de la forma $N_j^{n+1} = (C_{A,j}^{n+1}, insertar(A_j^{n+1}), C_D^n)$ (resp. $N_j^{n+1} = (C_{A,j}^{n+1}, borrar(A_j^{n+1}), C_D^n)$) $(1 \leq j \leq p)$ donde $A^n \leftarrow L_{j,1} \wedge \ldots \wedge L_{j,i} \wedge \ldots \wedge L_{j,m_j}$ es la regla deductiva jotaésima que define A^n utilizada en la creación del nodo N_j^{n+1} y $L_{j,i} = \neg A_j^{n+1}$ (resp. $L_{j,i} = A_j^{n+1}$).

Por el algoritmo 3, el recorrido del subárbol de la raíz N^n hace fracasar los p caminos de derivación de A^n a través de las p reglas que lo definen y además sigue la derivación para la conjunción C_D^n, así el recorrido del subárbol de raíz N^n es equivalente al paso de derivación $SLDNF$ entre el par (G^n: $\leftarrow \neg A^n \wedge C_D^n$, L^n: $\neg A^n$) y (G^{n+1}: $\leftarrow C_D^n$, L^{n+1}: A^{n+1}) siendo A^{n+1} un literal de la conjunción C_D^n.

Según el algoritmo 3 y teniendo en cuenta las equivalencias anteriores, el recorrido de un T-árbol de raíz $(C_A^0, insertar(A^0), C_D^0)$ (resp. $(C_A^0, borrar(A^0), C_D^0)$) es equivalente a la refutación $SLDNF$ para $D' \cup \{\leftarrow A^0 \wedge C_D^0\}$ (resp. $D' \cup \{\leftarrow \neg A^0 \wedge C_D^0\}$) con repuesta computada θ.**(1)**

Considerando que A^0 es base, el resultado (1) y los resultados de corrección para el procedimiento de resolución $SLDNF$, se puede afirmar que $comp(D') \vDash A^0$ (resp. $comp(D') \vDash \neg A^0$).■

Acciones generadas en presencia de reglas deductivas con variables existencialmente cuantificadas

La obtención del T-árbol utilizando reglas deductivas en las que aparezcan variables existencialmente cuantificadas se realiza siguiendo la misma propuesta que para el caso más simple. Es posible sin embargo, que durante el recorrido del árbol se llegue a algún nodo cuya operación no esté completamente instanciada. Cuando se da esta situación, se dice que *faltan valores*[20]. Para resolver este problema se pueden utilizar distintas estrategias (aunque no todas son válidas en todos los casos). Estas estrategias son las siguientes:

1. Encontrar un valor adecuado en la base de datos. Esto puede realizarse de dos formas:
   (a) Evaluando la Condición-Después;
   (b) Evaluando la Condición-Antes;
2. Encontrar un valor cualquiera. Hay tres posibles formas:
   (a) Elegir un valor de la base de datos al azar;
   (b) Pedir un valor al usuario; o
   (c) Utilizar el valor nulo.

[20] Este problema se ilustró en el apartado 3.1, ejemplo 3.

La elección de una estrategia dependerá de la situación concreta y en el caso de múltiples alternativas de la decisión del diseñador. En el método que se propone no se considerarán las estrategias 2a) ni 2c) por los problemas de implementación (en el primer caso) y de interpretación (en el segundo) que suponen.

El problema de la falta de valores al procesar un nodo del árbol puede aparecer en nodos con distintas propiedades siendo en cada caso distinta la solución más adecuada. Suponiendo que el nodo que se está procesando es $(C_A, O(A), C_D)$ y que su nodo predecesor es $(C_A^P, O^P(A^P), C_D^P)$, los casos que resulta interesante estudiar son los siguientes:

- Caso 1: A^P es un átomo derivado, $O^P = Insertar$ y $O = Insertar$
- Caso 2: A^P es un átomo derivado, $O^P = Insertar$ y $O = Borrar$
- Caso 3: A^P es un átomo básico y $O = Insertar$
- Caso 4: A^P es un átomo básico y $O = Borrar$
- Caso 5: A^P es un átomo derivado y $O^P = Borrar$

A continuación se presentan varios ejemplos en los que aparecen todos estos casos.

Ejemplo 58 *Caso 1 del problema de la falta de valores.*

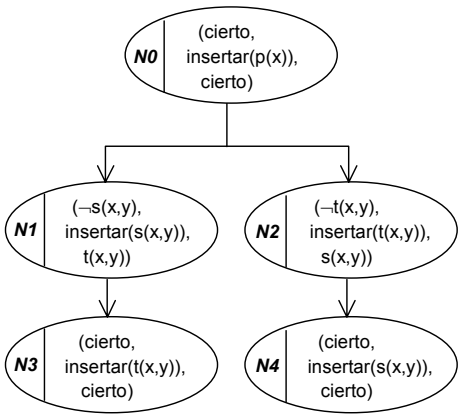
Sea el siguiente conjunto de reglas deductivas:

$1 : inc_W \leftarrow q(x) \wedge \neg p(x)$
$2 : p(x) \leftarrow s(x,y) \wedge t(x,y)$

Y sea la siguiente regla restauradora:

R:
`Evento:` $insertar(q(x))$
`Condición:` $\neg p(x)$
`Acción:` `...`

Cuya acción está representada en el siguiente árbol:



Supóngase que en la base de datos se tienen los siguientes hechos: $\{t(1,1), t(1,2)\}$ *y que se ejecuta una transacción que incluye la operación* $insertar(q(1))$. *Esta operación dispara la regla* R *detectándose la violación de la consistencia ya que la evaluación de la condición de la regla para ese evento,* $\neg p(1)$, *es cierta. Para restaurar la consistencia existen dos caminos posibles en el árbol:* $N0-N1-N3$ *y* $N0-N2-N4$. *Estudiando el primero de esos caminos se puede observar que el literal de la operación del nodo* $N1$ *no está instanciado; en este caso el problema podría resolverse utilizando la estrategia 1a) de la siguiente forma: la evaluación de la Condición-Después del nodo en el estado actual proporciona un valor adecuado para la variable y. Concretamente, la evaluación de* $t(1,y)$ *devuelve dos valores posibles* $\{y/1\}$ *e* $\{y/2\}$. *Cualquiera de ellos es adecuado para inducir la inserción de* $p(1)$.

Supóngase ahora que se ejecuta una transacción que incluye la operación insertar$(q(3))$. *Esta operación dispara la regla* R *detectándose la violación de la consistencia ya que la evaluación de la condición de la regla para ese evento,* $\neg p(3)$, *es cierta. De nuevo, en el camino* $N0-N1-N3$ *se puede observar que el literal de la operación del nodo* $N1$ *no está instanciado. En este caso no es posible el uso de la estrategia 1a) ya que la evaluación de la Condición-Después del nodo no proporciona ningún valor. En este caso es la estrategia 2b) la que debe ser utilizada.*

Ejemplo 59 *Caso 2 del problema de la falta de valores.*

Sea el siguiente conjunto de reglas deductivas:

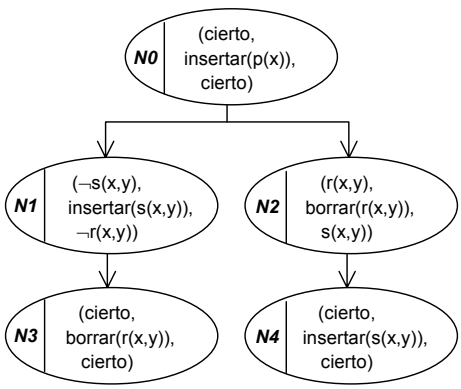
$$1 : inc_W \leftarrow q(x) \wedge \neg p(x)$$
$$2 : p(x) \leftarrow s(x,y) \wedge \neg r(x,y)$$

Y la siguiente regla restauradora:

R:

`Evento:` $insertar(q(x))$
`Condición:` $\neg p(x)$
`Acción:` ...

Cuya acción está representada en el siguiente árbol:



Supóngase que la base de datos tiene los hechos: $\{r(3,1), r(3,2), r(1,2), s(1,2)\}$ *y que se ejecuta una transacción que incluye la operación* $insertar(q(1))$. *Esta operación dispara la regla R detectándose la violación de la consistencia ya que la evaluación de la condición de la regla para ese evento,* $\neg p(1)$, *es cierta. Para restaurar la consistencia existen dos caminos posibles ambos con el problema de la falta de valores.*

En el camino $N0 - N1 - N3$ *se da el caso 1 y en este caso la estrategia 1a) no es elegible ya que la Condición-Después del nodo no es conjunción permitida*[21]*; habría que utilizar la 2b).*

Restaurando por el otro camino, el problema de la falta de valores se presenta en el nodo $N2$ *ya que no se sabe qué instancia de* $r(1,y)$ *hay que borrar. En este caso podría utilizarse la estrategia 1a) ya que la evaluación de la Condición-Después devuelve un valor de y (el 2) que satisface la Condición-Antes del nodo por lo que sería adecuada. La operación ejecutada sería* $borrar(r(1,2))$.

Supóngase ahora que se ejecuta una transacción que inserta $q(3)$ *por lo que es necesaria la inserción de* $p(3)$ *para restaurar. Siguiendo de nuevo el camino* $N0 - N2 - N4$, *el problema de la falta de valores se presenta en el nodo* $N2$; *en este caso la evaluación de la Condición-Después* $(s(3,y))$ *no devuelve ninguna instancia. Se puede utilizar ahora la estrategia 1b) ya que la evaluación de la Condición-Antes* $(r(3,y))$ *proporciona dos posibles valores para la variable y, el 1 y el 2. Si se elige por ejemplo el valor 1 se ejecuta la operación* $borrar(r(3,1))$. *En el nodo* $N4$ *se insertará* $s(3,1)$ *induciéndose con estas dos operaciones la inserción de* $p(3)$.

Por último, supóngase una transacción que incluye la operación $insertar(q(4))$. *En este caso para instanciar la operación del nodo* $N2$ *no son útiles ni la estrategia 1a) ni la 1b); la 2b) no es útil ya que ningún valor proporcionado por el usuario hará cierta la Condición-Antes del nodo (si existiera ese valor la estrategia 1b) lo habría encontrado). Así, hay que concluir que este camino no es elegible, y hay que seguir por el camino* $N0 - N1 - N3$.

Las soluciones al caso 3 y al caso 4 van a ser las mismas que en el caso 1 y en el caso 2 como se muestra en los siguientes ejemplos.

Ejemplo 60 *Caso 3 del problema de la falta de valores.*

Sea el siguiente conjunto de reglas deductivas:

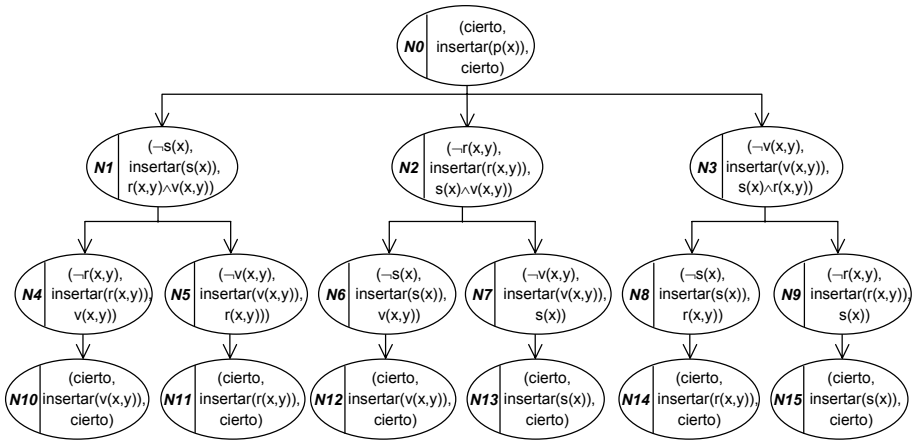
$$1 : inc_W \leftarrow q(x) \wedge \neg p(x)$$
$$2 : p(x) \leftarrow s(x) \wedge r(x,y) \wedge v(x,y)$$

R:

`Evento:`	$insertar(q(x))$
`Condición:`	$\neg p(x)$
`Acción:`	...

Cuya acción está representada en el siguiente árbol:

[21] Es decir, tal que toda variable ocurre al menos en un literal positivo.



Supóngase que en la base de datos se tiene el hecho $\{v(1,2)\}$ *y que se ejecuta una transacción que incluye la operación* $insertar(q(1))$. *Esta operación dispara la regla* R *detectándose la violación de la consistencia ya que la evaluación de la condición de la regla para ese evento,* $\neg p(1)$, *es cierta. Para restaurar la consistencia todos los caminos que parten de la raíz son elegibles al cumplirse su Condición-Antes.*

En el camino $N0 - N1 - N4 - N10$ *el algoritmo inserta* $s(1)$ *en el nodo* $N1$ *pero al llegar al nodo* $N4$ *se presenta el problema de la falta de valores. En este caso es fácil observar que la solución puede ser la misma que en el caso 1 ya que la evaluación de la Condición-Después,* $v(x,y)$, *proporciona un valor adecuado para la variable* y, $\{y/2\}$.

En el camino $N0 - N1 - N5 - N11$ *el algoritmo inserta de nuevo* $s(1)$ *en el nodo* $N1$ *pero al llegar al nodo* $N5$ *se presenta el problema de la falta de valores. En este caso, la estrategia 1a) no es útil al no devolver valores la evaluación de la Condición-Después debiendo usarse entonces la estrategia 2b).*

Ejemplo 61 *Caso 4 del problema de falta de valores.*

La solución a este caso va a ser la misma que en el caso 2; para mostrarlo sea el siguiente conjunto de reglas deductivas:

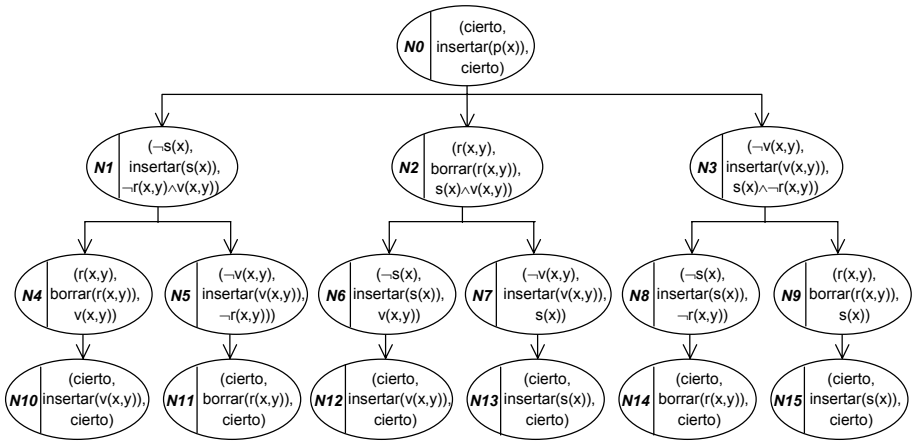
$$1 : inc_W \leftarrow q(x) \wedge \neg p(x)$$
$$2 : p(x) \leftarrow s(x) \wedge \neg r(x,y) \wedge v(x,y)$$

Y sea la siguiente regla restauradora:

R:

`Evento:`	$insertar(q(x))$
`Condición:`	$\neg p(x)$
`Acción:`	...

Cuya acción está representada en el siguiente árbol:



Supóngase que en la base de datos se tienen los siguientes hechos: $\{r(1,2), r(1,3), v(1,3)\}$ *y que se ejecuta una transacción que incluye la operación* $insertar(q(1))$. *Esta operación dispara la regla* R *detectándose la violación de la consistencia ya que la evaluación de la condición de la regla para ese evento,* $\neg p(1)$, *es cierta.*

En el camino $N0-N1-N4-N10$ *el algoritmo inserta* $s(1)$ *en el nodo* $N1$ *pero al llegar al nodo* $N4$ *se presenta el problema de la falta de valores que se puede resolver como en el caso 2. En este ejemplo el uso de la estrategia 1a) propondría el borrado de* $r(1,3)$ *habiéndose obtenido el valor para la variable* y *al evaluar la Condición-Después,* $v(1,y)$. *Si se utiliza la estrategia 1b) la evaluación de la Condición-Antes devolvería dos posibles valores* $\{y/2\}$ *e* $\{y/3\}$ *que generarían dos soluciones posibles* $\{borrar(r(1,2), insertar(v(1,2)\}$ *o* $\{borrar(r(1,3)\}$.

Ejemplo 62 *Caso 5 del problema de la falta de valores.*

Sea el siguiente conjunto de reglas deductivas:

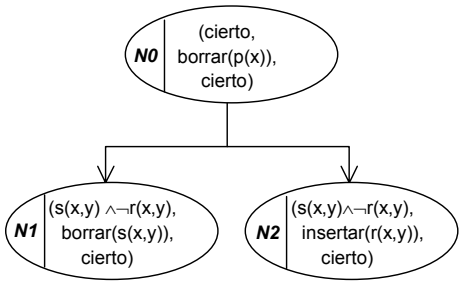
$$1 : inc_W \leftarrow q(x) \wedge p(x)$$
$$2 : p(x) \leftarrow s(x,y) \wedge \neg r(x,y)$$

Y sea la siguiente regla restauradora:

R:

`Evento:`	$insertar(q(x))$
`Condición:`	$p(x)$
`Acción:`	...

Cuya acción está representada en el siguiente árbol:



Supóngase que en la base de datos se tienen los siguientes hechos: $\{s(1,2), s(1,3)\}$ *y que se ejecuta una transacción que incluye la operación* $insertar(q(1))$. *Esta operación dispara la regla* R *detectándose la violación de la consistencia ya que la evaluación de la condición de la regla para ese evento,* $\neg p(1)$, *es cierta. Recorriendo el camino* $N0-N1$ *se llega al nodo* $N1$ *cuya operación no está instanciada. En este caso la única estrategia posible es la 1b) ya que las instancias de* $s(1,y)$ *que hay que borrar son todas aquéllas que hacen cierta la Condición-Antes. Así se deberían ejecutar las operaciones* $borrar(s(1,2))$ *y* $borrar(s(1,3))$ *para restaurar la consistencia.*

Si se sigue el camino $N0-N2$ *se deben ejecutar las operaciones* $insertar(r(1,2))$ *e* $insertar(r(1,3))$ *que de nuevo se pueden obtener evaluando la Condición-Antes.*

De los ejemplos anteriores se pueden extraer las siguientes conclusiones:

- Caso 1: $O^P = Insertar$, A^P es un átomo derivado y $O = Insertar$. El conjunto de instancias consta nada más de una substitución que se puede obtener evaluando la Condición-Después en la base de datos si se cumple que esta condición es una conjunción permitida. Si falta algún valor entonces el usuario deberá proporcionarlo. En cualquier caso la Condición-Antes instanciada con la substitución elegida debe ser cierta en la base de datos.

- Caso 2: $O^P = Insertar$, A^P es un átomo derivado y $O = Borrar$. En este caso la Condición-Después siempre es una conjunción permitida en tiempo de ejecución y además todas las variables de la operación a instanciar aparecen en ella[22]. El conjunto de instancias consta nada más de una substitución que se puede obtener evaluando la Condición-Después en la base de datos si la Condición-Antes instanciada con la substitución elegida es cierta en la base de datos. Si no es posible se evaluará Condición-Antes. Si ambas estrategias fallan el nodo no es elegible y habrá que seguir otro camino.

- Caso 3: A^P es un átomo básico y $O = Insertar$. La solución es la misma que en el caso 1.

[22] La afirmación realizada en el caso 2 es evidente si se recuerda cómo se genera un nodo sucesor de otro cuya operación es una inserción sobre un predicado derivado. Así, si el nodo predecesor es $(C_A, insertar(A), C_D)$ donde A es derivado y él nodo sucesor es $(B\theta, borrar(B\theta), (C_D \wedge L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_n)\theta)$, que ha sido generado con la regla deductiva $A' \leftarrow L_1 \wedge \ldots \wedge L_i \wedge \ldots \wedge L_n$ (donde $L_i = \neg B$ y B es un átomo), y con la substitución restringida θ entre A^* y A' sobre el literal L_i, entonces todas las variables que aparecen en B aparecen también en $L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_n$ ya que las reglas deductivas son permitidas. Además, cuando se alcance ese nodo en tiempo de ejecución, C_D estará instanciado por el nodo anterior por lo que la conjunción $(C_D \wedge L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_n)\theta$ no tendrá ninguna variable que sólo aparezca en un literal negativo (i.e. será una conjunción permitida en tiempo de ejecución).

- Caso 4: A^P es un átomo básico y $O = Borrar$. La solución es la misma que en el caso 2.
- Caso 5: $O^P = Borrar$ y A^P es un átomo derivado. Sea cual sea la operación del nodo, el conjunto de instancias se obtiene evaluando la Condición-Antes en la base de datos de entrada.

El recorrido de un T-árbol generado según la definición 9, en el supuesto de que puedan aparecer variables existencialmente cuantificadas en cualquier regla deductiva, se realiza con el algoritmo recursivo que se presenta a continuación. Este algoritmo es una versión revisada del algoritmo 3 que resuelve el problema de la falta de valores.

Algoritmo 4 *Algoritmo de recorrido de un T-árbol en bases de datos jerárquicas (versión 2).*

```
ALGORITMO Recorrer_T-Árbol
ENTRADA
    (C_A^0, O^raíz, C_D^0) : Raíz de un T-árbol;
    D : Estado de base de datos;
    D_incial : Estado inicial de base de datos;
SALIDA
    D' : Estado de base de datos;
VARIABLES
    C : Clan sucesor;
    c : Nodo de un T-árbol;
    Θ : Conjunto de substituciones;
    θ : Substitución;
INICIO
/*Se visita la raíz*/
SI O^raíz = O^0(A^0)
ENTONCES
| SI el predicado de A^0 es derivado
| ENTONCES
| | PARA cada clan, C, sucesor de la raíz HACER
| | | Θ := ∅; /*Θ es el conjunto de substituciones que hay que aplicar
| | |         a la operación del miembro del clan C que se va a visitar*/
| | | MIENTRAS Θ = ∅ Y quedan miembros en C HACER
| | | | Elegir un miembro del clan, c = (C_A^1, O^1(A^1), C_D^1);
| | | | SI A^1 no es base
| | | | ENTONCES
| | | | | Instanciar((C_A^1, O^1(A^1), C_D^1), O^0(A^0), D, Θ)
| | | | SI NO /*A^1 es base*/
| | | | | SI ∃ una refutación para D ∪ {← C_A^1}
| | | | | ENTONCES
| | | | | | Θ := ε    /*ε es la substitución identidad*/
| | | | | FIN_SI
| | | | FIN_SI;
| | | | C := C − {c}
| | | FIN_MIENTRAS;
| | | PARA CADA θ ∈ Θ HACER /*Se recorre el árbol para
| | | |                       cada instancia de Θ*/
```

| | | | Aplicar θ a todo el árbol;
| | | | Recorrer_T-Árbol$((C_A^1\theta, O^1(A^1\theta), C_D^1\theta), D, D_{inicial}, D')$
| | | FIN_PARA
| | FIN_PARA
| SI NO /*El predicado de A^0 es básico
| | y el nodo tiene un solo clan sucesor*/
| | SI $O^0 = insertar$
| | ENTONCES
| | | $D' := D \cup \{A^0\}$
| | SI NO /*$O^0 = borrar$*/
| | | $D' := D - \{A^0\}$
| | FIN_SI;
| | SI C_D^0 es cierta en D'
| | ENTONCES
| | | Parar
| | SI NO
| | | $D := D'$;
| | | $\Theta := \emptyset$;
| | | $C :=$ Clan sucesor de la raíz;
| | | MIENTRAS $\Theta = \emptyset$ HACER
| | | | Elegir un miembro de C, sea $c = (C_A^1, O^1(A^1), C_D^1)$;
| | | | SI A^1 no es base
| | | | ENTONCES
| | | | | Instanciar$((C_A^1, O^1(A^1), C_D^1), O^0(A^0), D, \Theta)$
| | | | SI NO
| | | | | SI $\exists$ una refutación para $D \cup \{\leftarrow C_A^1\}$
| | | | | ENTONCES
| | | | | | $\Theta := \varepsilon$
| | | | | FIN_SI
| | | | FIN_SI;
| | | | $C := C - \{c\}$;
| | | FIN_MIENTRAS;
| | | PARA CADA $\theta \in \Theta$ HACER
| | | | Aplicar θ a todo el árbol;
| | | | Recorrer_T-Árbol$((C_A^1\theta, O^1(A^1\theta), C_D^1\theta), D, D_{inicial}, D')$
| | | FIN_PARA
| | FIN_SI
| FIN_SI
SI NO /*$O^{raíz} = abortar$*/
| $D' := D_{inicial}$
FIN_SI
FIN.■

A continuación se presenta el algoritmo Instanciar. Este algoritmo recibe como entrada un nodo junto con la operación de su nodo predecesor y un estado de base de datos, y devuelve un conjunto de substituciones que instancian la operación del nodo que hay que procesar de acuerdo con los casos antes analizados. Las substituciones obtenidas satisfacen siempre la Condición-Antes del nodo.

Algoritmo 5 *Algoritmo para instanciar la operación de un nodo de un T-árbol.*

```
ALGORITMO Instanciar
ENTRADA
    (C_A, O(A), C_D) : Nodo de un T-árbol;
    O^P(A^P) : Operación del nodo predecesor;
    D : Estado de base de datos;
SALIDA
    Θ : Conjunto de substituciones;
VARIABLES
    Encontrada : Lógico;
    Δ, Φ : Conjunto de substituciones;
    α, β, δ, φ, θ : Substitución;
INICIO
Encontrada := falso;
Θ := ∅;
/*Caso 5*/
SI (A^P es derivado Y O^P = Borrar)
ENTONCES
| Θ := {θ | θ es una respuesta computada para D ∪ {← C_A}}
FIN_SI;
/*Caso 1 y Caso 3*/
SI (A^P es derivado Y O^P = Insertar Y O = Insertar) O
|        (A^P es básico Y O = Insertar)
ENTONCES
| SI C_D es una conjunción permitida Y Var[23](A) ∩ Var(C_D) ≠ ∅
| ENTONCES
| | Δ := {δ|δ es una respuesta computada para D ∪ {← C_D}};
| | MIENTRAS NO Encontrada Y Δ ≠ ∅ HACER
| | | α := Elegir un elemento de Δ;
| | | SI Aα no es base
| | | ENTONCES
| | | | β :=Substitución proporcionada por el usuario
| | | |            para las variables libres de Aα;
| | | | θ := αβ
| | | FIN_SI;
| | | SI existe una respuesta computada para D ∪ {← C_Aθ}
| | | ENTONCES
| | | | Encontrada := cierto;
| | | | Θ := {θ}
| | | SI NO
| | | | Δ := Δ − {α}
| | | FIN _SI
| | FIN_MIENTRAS;
| FIN_SI;
| SI (NO Encontrada)
| ENTONCES
| | θ := Substitución proporcionada por el usuario para las variables
| |        libres de A tal que existe una respuesta computada para
```

[23] $Var(E)$, donde E es una operación o una condición, es una función que devuelve el conjunto de variables que aparecen en E.

| | $D \cup \{\leftarrow C_A\theta\}$;
| | $\Theta := \{\theta\}$
| FIN_SI
FIN_SI;
/*Caso 2 y Caso 4*/
SI (A^P es derivado Y $O^P = Inserción$ Y $O = Borrar$) O
| (A^P es básico Y $O = Borrar$)
ENTONCES
| $\Delta := \{\delta | \delta$ es una respuesta computada para $D \cup \{\leftarrow C_D\}\}$;
| MIENTRAS NO Encontrada Y $\Delta \neq \emptyset$ HACER
| | $\theta :=$ Elegir un elemento de Δ;
| | SI existe una respuesta computada para $D \cup \{\leftarrow C_A\theta\}$
| | ENTONCES
| | | Encontrada := cierto;
| | | $\Theta := \{\theta\}$
| | SI NO
| | | $\Delta := \Delta - \{\theta\}$
| | FIN_SI
| FIN_MIENTRAS;
| SI (NO Encontrada)
| ENTONCES
| | $\Phi := \{\phi | \phi$ es una respuesta computada para $D \cup \{\leftarrow C_A\}\}$;
| | SI $\Phi \neq \emptyset$
| | ENTONCES
| | | $\theta :=$ Elegir un elemento de Φ;
| | | $\Theta := \{\theta\}$
| | FIN_SI
| FIN_SI
FIN_SI
FIN.■

Teorema 4 *Corrección del algoritmo de recorrido de un T-árbol en bases de datos jerárquicas (versión 2).*

⇒ Sea D una base de datos jerárquica y sea D' la salida de la llamada al algoritmo *Recorrer_T-Árbol*($(C_A^0, insertar(A^0), C_D^0)$, D, $D_{inicial}$, D') (resp. *Recorrer_T-Árbol*($(C_A^0, borrar(A^0), C_D^0)$, D, $D_{inicial}$, D')) donde A^0 es base y C_A^0 es cierto en D,

⇒ entonces se cumple que $comp(D') \vDash A^0$ (resp. $comp(D') \vDash \neg A^0$).

Demostración:

La diferencia entre el algoritmo 3 y el algoritmo 4 reside en la incorporación en este último del procedimiento Instanciar que instancia, antes de ser visitado (procesado), la operación de un nodo cuando ésta no es base. Una vez instanciado el nodo, los pasos en el recorrido de un T-árbol según el algoritmo 4 coinciden con los del algoritmo 3, por lo tanto la propiedad de corrección para este último (teorema 3) sigue siendo válida para el algoritmo 4 (teorema 4) siempre que las instancias producidas por el algoritmo Instanciar sean las correctas, extremo éste que se justifica a continuación.

La corrección de una instanciación depende de las características del nodo a instanciar, pudiendo diferenciarse, como aparece en el algoritmo Instanciar, cinco casos.

- En los casos 1 y 3, el nodo cuya operación hay que instanciar tiene la forma $(\neg A, insertar(A), C_D)$ donde A no es base. Cualquier substitución α de las variables libres de A que haga cierto $\neg A\alpha$ es adecuada. En esta línea el algoritmo Instanciar encuentra la substitución α buscando valores en la base de datos para poder automatizar la búsqueda (uso de la Condición-Antes o la Condición-Después del nodo) o consultando al usuario, en cualquier caso habrá que comprobar que $\neg A\alpha$ se satisface. Se puede afirmar por tanto que la instanciación producida por Instanciar en estos casos es correcta.

- En los casos 2 y 4, el nodo cuya operación hay que instanciar tiene la forma $(A, borrar(A), C_D)$ donde A no es base. De nuevo cualquier substitución α de las variables libres de A que haga cierto $A\alpha$ es adecuada. Para encontrar una substitución α que haga cierto $A\alpha$ los valores hay que buscarlos siempre en la base de datos; por esto, el algoritmo Instanciar no consulta al usuario y encuentra esta substitución buscando estos valores con el uso de la Condición-Antes o la Condición-Después del nodo, de nuevo habrá que comprobar que $A\alpha$ se satisface. Si esta substitución no se encuentra entonces el nodo no es elegible. Se puede afirmar por tanto que la instanciación producida por Instanciar en estos casos es correcta.

- En el caso 5, la operación del nodo predecesor O^P es *borrar* y A^P es derivado, es decir el nodo tiene la forma $N^P = (C_A^P, borrar(A^P), C_D^P)$. El procesamiento de este nodo significa el recorrido de todos los subárboles que representen un camino de derivación para A^P a través de las reglas deductivas que lo definen con el objetivo de eliminarlos. Los nodos raíces de estos subárboles tienen la forma $N = (C_A, O(A), C_D^P)$ donde O puede ser *insertar* o *borrar* y C_A es el cuerpo de una regla deductiva que define A^P y A es un átomo de C_A; si A no es base, el algoritmo Instanciar debe obtener "todas las instancias" de sus variables libres que hacen cierto C_A y procesar el T-árbol de raíz N para cada una de ellas con el fin de asegurar el borrado de A^P. De nuevo se comprueba que el algoritmo Instanciar produce "todas las instancias" y se puede afirmar que en este caso la instanciación también es correcta.

Una vez justificado que las instancias producidas por el algoritmo Instanciar son correctas en todos los casos, la propiedad de corrección del algoritmo de recorrido de un T-árbol para bases de datos jerárquicas cuando se produce el problema de la falta de valores (teorema 4) queda demostrada ya que coincide con la demostración del teorema 3.■

Para finalizar con el problema de la falta de valores, a continuación se presenta un ejemplo más complejo.

Ejemplo 63 *Sea el siguiente conjunto de reglas deductivas:*

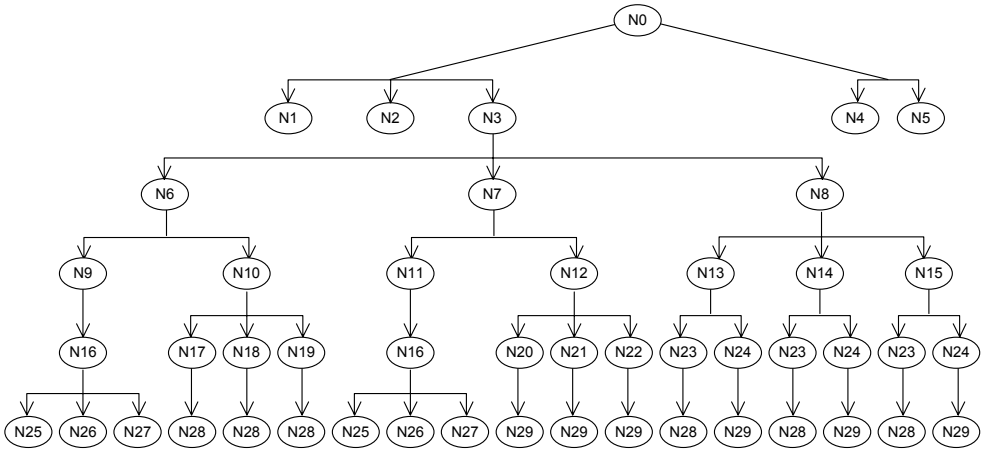
$$1 : inc_W \leftarrow q(x, y) \wedge p(x, y)$$
$$2 : p(x, y) \leftarrow k(x, y) \wedge s(z_1, y) \wedge \neg r(x, z_1)$$

$3 : p(x, y) \leftarrow l(x, y) \wedge \neg j(x, y)$
$4 : r(x, z_1) \leftarrow h(x, z_1, z_2) \wedge t(z_1, z_2) \wedge \neg n(z_1, z_2)$
$5 : n(z_1, z_2) \leftarrow d(z_1, z_2) \wedge g(z_1, z_3) \wedge \neg f(z_1, z_3)$

Y sea la siguiente regla restauradora:

R:
`Evento:` $insertar(q(x, y))$
`Condición:` $p(x, y)$
`Acción:` ...

Cuya acción está representada en el siguiente árbol:



Donde los nodos son los siguientes tripletes:

Nodo raíz:
$N0 = (cierto, borrar(p(x, y)), cierto)$

Clanes sucesores del nodo $N0$:
1^{er} Clan:
$N1 = (k(x, y) \wedge s(z_1, y) \wedge \neg r(x, z_1), borrar(k(x, y)), cierto)$
$N2 = (k(x, y) \wedge s(z_1, y) \wedge \neg r(x, z_1), borrar(s(z_1, y)), cierto)$
$N3 = (k(x, y) \wedge s(z_1, y) \wedge \neg r(x, z_1), insertar(r(x, z_1)), cierto)$

2^o Clan:
$N4 = (l(x, y) \wedge \neg j(x, y), borrar(l(x, y)), cierto)$
$N5 = (l(x, y) \wedge \neg j(x, y), insertar(j(x, y)), cierto)$

Clanes sucesores del nodo $N3$:
Clan único:
$N6 = (\neg h(x, z_1, z_2), insertar(h(x, z_1, z_2)), t(z_1, z_2) \wedge \neg n(z_1, z_2))$
$N7 = (\neg t(z_1, z_2), insertar(t(z_1, z_2)), h(x, z_1, z_2) \wedge \neg n(z_1, z_2))$
$N8 = (n(z_1, z_2), borrar(n(z_1, z_2)), h(x, z_1, z_2) \wedge t(z_1, z_2))$

Clanes sucesores del nodo $N6$:

Clan único:

$N9 = (\neg t(z_1, z_2), insertar(t(z_1, z_2)), \neg n(z_1, z_2))$

$N10 = (n(z_1, z_2), borrar(n(z_1, z_2)), t(z_1, z_2))$

Clanes sucesores del nodo $N7$:

Clan único:

$N11 = (\neg h(x, z_1, z_2), insertar(h(x, z_1, z_2)), \neg n(z_1, z_2))$

$N12 = (n(z_1, z_2), borrar(n(z_1, z_2)), h(x, z_1, z_2))$

Clanes sucesores del nodo $N8$:

1^{er} *Clan*:

$N13 = (d(z_1, z_2) \wedge g(z_1, z_3) \wedge \neg f(z_1, z_3), borrar(d(z_1, z_2)),$
$h(x, z_1, z_2) \wedge t(z_1, z_2))$

$N14 = (d(z_1, z_2) \wedge g(z_1, z_3) \wedge \neg f(z_1, z_3), borrar(g(z_1, z_3)),$
$h(x, z_1, z_2) \wedge t(z_1, z_2))$

$N15 = (d(z_1, z_2) \wedge g(z_1, z_3) \wedge \neg f(z_1, z_3), insertar(f(z_1, z_3)),$
$h(x, z_1, z_2) \wedge t(z_1, z_2))$

Clanes sucesores de los nodos $N9$ y $N11$[24]:

Clan único

$N16 = (cierto, borrar(n(z_1, z_2)), cierto)$

Clanes sucesores del nodo $N10$:

1^{er} *Clan*:

$N17 = (d(z_1, z_2) \wedge g(z_1, z_3) \wedge \neg f(z_1, z_3), borrar(d(z_1, z_2)), t(z_1, z_2))$

$N18 = (d(z_1, z_2) \wedge g(z_1, z_3) \wedge \neg f(z_1, z_3), borrar(g(z_1, z_3)), t(z_1, z_2))$

$N19 = (d(z_1, z_2) \wedge g(z_1, z_3) \wedge \neg f(z_1, z_3), insertar(f(z_1, z_3)), t(z_1, z_2))$

Clanes sucesores del nodo $N12$:

1^{er} *Clan*:

$N20 = (d(z_1, z_2) \wedge g(z_1, z_3) \wedge \neg f(z_1, z_3), borrar(d(z_1, z_2)), h(x, z_1, z_2))$

$N21 = (d(z_1, z_2) \wedge g(z_1, z_3) \wedge \neg f(z_1, z_3), borrar(g(z_1, z_3)), h(x, z_1, z_2))$

$N22 = (d(z_1, z_2) \wedge g(z_1, z_3) \wedge \neg f(z_1, z_3), insertar(f(z_1, z_3)), h(x, z_1, z_2))$

Clanes sucesores de los nodos $N13, N14$ y $N15$:

Clan único:

$N23 = (\neg h(x, z_1, z_2), insertar(h(x, z_1, z_2)), t(z_1, z_2))$

$N24 = (\neg t(z_1, z_2), insertar(t(z_1, z_2)), h(x, z_1, z_2))$

Clanes sucesores del nodo $N16$:

1^{er} *Clan*:

$N25 = (d(z_1, z_2) \wedge g(z_1, z_3) \wedge \neg f(z_1, z_3), borrar(d(z_1, z_2)), cierto)$

$N26 = (d(z_1, z_2) \wedge g(z_1, z_3) \wedge \neg f(z_1, z_3), borrar(g(z_1, z_3)), cierto)$

$N27 = (d(z_1, z_2) \wedge g(z_1, z_3) \wedge \neg f(z_1, z_3), insertar(f(z_1, z_3)), cierto)$

Clanes sucesores de los nodos $N17, N18, N19$ y $N23$:

Clan único:

$N28 = (cierto, insertar(t(z_1, z_2)), cierto)$

[24] El que los hijos de $N9$ y $N11$ sean los mismos no convierte el árbol en un grafo sino que implica que en el árbol hay nodos iguales. Se presenta así por abreviar.

Clanes sucesores de los nodos $N20, N21, N22$ y $N24$:

Clan único:

$N29 = (cierto, insertar(h(x, z_1, z_2)), cierto)$

En este T-árbol, el problema de la falta de valores se presenta en los siguientes nodos:

- *Caso 1:* $N6$ y $N7$
- *Caso 2:* $N8$
- *Caso 5:* $N2, N3, N14, N15, N18, N19, N21, N22, N26$ y $N27$

Para terminar con el ejemplo, supóngase una base de datos con los siguientes hechos: $\{k(1,2), s(3,2), s(4,2), t(3,4), t(3,2), h(1,3,2), d(3,2), g(3,3)\}$ *que permite deducir los siguientes hechos derivados:* $\{p(1,2), n(3,2)\}$ *y supóngase una transacción que ejecute la operación insertar*$(q(1,2))$; *la regla R se dispara y la condición* $p(1,2)$ *se hace cierta por lo que hay que procesar el T-árbol. Dado que el nodo raíz tiene dos clanes hay que recorrer, si es posible, un camino para cada clan. En este caso, para el segundo clan no hay camino posible ya que* $p(1,2)$ *no es derivable a través de la regla 3. Para el primer clan se podrían elegir entre otros los siguientes caminos:*

- *Camino* $N0 - N2$:
  - *Nodo N2* $\Longrightarrow$ *borrar*$(s(z_1, 2))$. *Se detecta el caso 5 del problema de la falta de valores. Las instancias que deben borrarse se obtienen evaluando la Condición-Antes.*

  Este camino restaura con la transacción $\{borrar(s(3,2)), borrar(s(4,2))\}$

- *Camino* $N0 - N3 - N6 - N9 \ldots$ *:*
  - *Nodo N3* $\Longrightarrow$ *insertar*$(r(1, z_1))$. *Caso 5 del problema de la falta de valores. Las instancias se obtienen evaluando la Condición-Antes. Hay pues que insertar* $r(1,3)$ *y* $r(1,4)$ *para evitar que* $p(1,2)$ *sea derivable a través de la regla deductiva 2. Obsérvese que ahora el recorrido del resto del árbol debe hacerse una vez para cada instancia que se pretende insertar.*

  Substitución $\{z_1/3\}$

  - *Nodo N6* $\Longrightarrow$ *insertar*$(h(1,3,z_2))$. *Caso 1 del problema de la falta de valores. Las instancias se obtienen evaluando la Condición-Después en el estado de base de datos actual que devuelve la substitución* $\{z_2/4\}$ *que satisface la Condición-Antes ya que* $\neg h(1,3,4)$ *es cierto en la base de datos ejecutándose la operación insertar*$(h(1,3,4)) \Longrightarrow$ *Condición-Después cierta* $\rightarrow$ *Terminar el recorrido.*

  Substitución $\{z_1/4\}$

- *Nodo N6* $\Longrightarrow$ $insertar(h(1,4,z_2))$. *Caso 1 del problema de la falta de valores. La evaluación de la Condición-Después en el estado de base de datos actual no proporciona ninguna instancia, habrá que utilizar la estrategia 2b). Supóngase que consultado el usuario, éste proporciona la substitución* $\{z_2/5\}$*; dado que la Condición-Antes instanciada con esa substitución es cierta se ejecuta la operación* $insertar(h(1,4,5))$ $\Longrightarrow$ *Condición-Después falsa* $\rightarrow$ *Seguir el recorrido.*
- *Nodo N9* $\Longrightarrow$ $\neg t(4,5)$ *es cierto* $\Longrightarrow$ $insertar(t(4,5))$
  $\Longrightarrow$ *Condición-Después cierta* $\Longrightarrow$ *Terminar el recorrido*

Este camino restaura con la transacción $\{insertar(h(1,3,4))$, $insertar(h(1,4,5)$, $insertar(t(4,5))\}\}$

- *Camino* $N0 - N3 - N8 \ldots$ *:*
  - *Nodo N3* $\Longrightarrow$ $insertar(r(1,z_1))$. *La evaluación de la Condición-Antes proporciona las instancias* $r(1,3)$ *y* $r(1,4)$.

  Substitución $\{z_1/3\}$
  - *Nodo N8* $\Longrightarrow$ $borrar(n(3,z_2))$. *Caso 2 del problema de la falta de valores. Las instancias se pueden obtener evaluando la Condición-Después en el estado de base de datos actual que devuelve la substitución* $\{z_2/2\}$ *que satisface la Condición-Antes* $\Longrightarrow$ *Condición-Después falsa* $\Longrightarrow$ *Seguir el recorrido.*
  - *Nodo N13* $\Longrightarrow$ *Condición-Antes cierta* $\Longrightarrow$ $borrar(d(3,2))$
    $\Longrightarrow$ *Condición-Después cierta* $\Longrightarrow$ *Terminar el recorrido.*

  Substitución $\{z_1/4\}$
  - *Nodo N8* $\Longrightarrow$ $borrar(n(4,z_2))$. *Caso 2 del problema de la falta de valores que en este caso no proporciona la Condición-Después ni tampoco la Condición-Antes. Ningún valor proporcionado por el usuario hará cierta la Condición-Antes por lo que no se puede seguir este camino para* $insertar(r(1,4))$ $\Longrightarrow$ *Condición-Antes falsa* $\Longrightarrow$ *Buscar otro camino.*
  - *Nodo N7* $\Longrightarrow$ $insertar(t(4,z_2)$. *Faltan valores que en este caso no proporciona la Condición-Después ni tampoco la Condición-Antes. El usuario deberá proporcionar una valor, sea éste* $\{z_2/1\}$; *la Condición-Antes* $\neg t(4,1)$ *es cierta y se ejecuta la operación* $insertar(t(4,1))$ $\Longrightarrow$ *Condición-Después falsa* $\Longrightarrow$ *Seguir el recorrido.*
  - *Nodo N11* $\Longrightarrow$ *Condición-Antes cierta* $\Longrightarrow$ $insertar(h(1,4,1))$
    $\Longrightarrow$ *Condición-Después cierta* $\Longrightarrow$ *Terminar el recorrido.*

    Este camino restaura con la transacción $\{borrar(d(3,2))$, $insertar(t(4,1))$, $insertar(h(1,4,1))\}$.

4.3.4 Obtención de la acción en bases de datos estratificadas

La obtención de la acción para las reglas restauradoras en bases de datos estratificadas va a seguir las mismas directrices que en el caso de bases de datos jerárquicas. El problema, en este caso, es que la presencia en la condición de literales construidos con *predicados recursivos* podría dar lugar a T-árboles infinitos. El concepto de predicado recursivo se define a continuación.

Definición 10 *Predicado recursivo.*

Dado un conjunto de reglas deductivas, los predicados recursivos son aquéllos que forman parte de un ciclo del grafo de dependencias asociado a ese conjunto de reglas.■

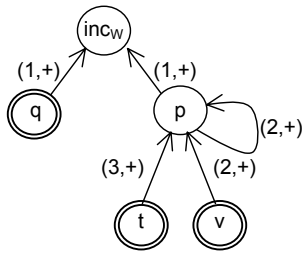
Ejemplo 64 *Sea el siguiente conjunto de reglas deductivas:*

$1 : inc_W \leftarrow p(x) \wedge q(x)$
$2 : p(x) \leftarrow v(x, y, z) \wedge p(y) \wedge p(z)$
$3 : p(x) \leftarrow t(x)$

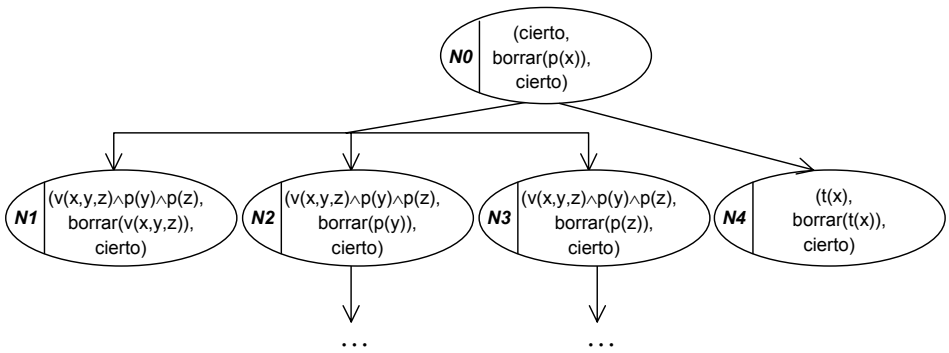
Y sea también la siguiente regla restauradora:

R:
`Evento:` $insertar(q(x))$
`Condición:` $p(x)$
`Acción:` ...

El único predicado recursivo de este ejemplo es p, como puede observarse en el grafo de dependencias:



El T-árbol de la regla restauradora R (según la definición 9) sería el siguiente:



Este T-árbol es infinito ya que los nodos N2 y N3 tendrían ambos dos clanes sucesores en uno de los cuales habrá dos nodos con la operación borrar(p(_)) generándose por tanto infinitos clanes sucesores.

Para evitar esta situación es necesario introducir un criterio de parada en la generación del árbol. Sea cual sea el criterio elegido, una consecuencia evidente de su aplicación es que el método no obtendrá todas las posibles transacciones restauradoras; dado que esta situación es inevitable, el criterio de parada elegido ha sido el más sencillo posible y consiste en evitar el crecimiento del árbol utilizando los predicados recursivos. Para ello se introduce el concepto de *nodo permitido.*

Definición 11 *Nodo permitido en un T-árbol.*

El nodo $(C_A, O(A), L_1 \wedge \ldots \wedge L_n)$ es permitido en un T-árbol si no se da ninguna de las dos condiciones siguientes:

1. El predicado de A es un predicado recursivo y además en el camino desde ese nodo hasta la raíz hay otro nodo $(C'_A,\ O'(A'), C'_D)$ tal que el predicado de A' es el mismo que el de A;

2. Existe un L_i $(1 \leq i \leq n)$ tal que el predicado de L_i es un predicado recursivo y además en el camino desde ese nodo hasta la raíz hay otro nodo $(C'_A, O'(A'), C'_D)$ tal que el predicado de A' es el mismo que el de L_i.■

Ejemplo 65 *Dada la regla restauradora del ejemplo 64, para la obtención del árbol se generarían los siguientes nodos:*

Nodo raíz:

$$N0 = (cierto, borrar(p(x)), cierto)$$

Clanes sucesores del nodo N0:

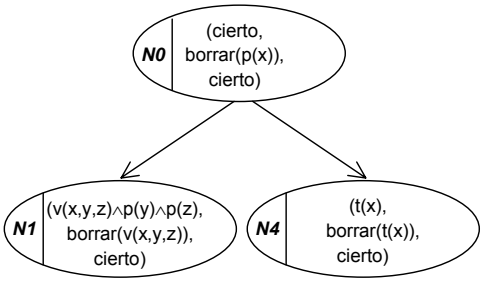
1^{er} *Clan:*

$$N1 = (v(x,y,z) \wedge p(y) \wedge p(z), borrar(v(x,y,z)), cierto)$$
$$N2 = (v(x,y,z) \wedge p(y) \wedge p(z), borrar(p(y)), cierto) \Rightarrow No - permitido$$
$$N3 = (v(x,y,z) \wedge p(y) \wedge p(z), borrar(p(z)), cierto) \Rightarrow No - permitido$$

2^o *Clan*:

$$N4 = (t(x), borrar(t(x)), cierto)$$

Así, el T-árbol de la regla restauradora sería el siguiente:



Esta regla restaura la consistencia borrando las instancias $v(x, y, z)$ que permiten generar la instancia de $p(x)$ que se quiere borrar (a un primer nivel) y borrando la instancia de $t(x)$ asociada.

Ejemplo 66 *Sea el siguiente conjunto de reglas deductivas:*

$$1 : inc_W \leftarrow \neg p(x) \wedge q(x)$$
$$2 : p(x) \leftarrow v(x, y, z) \wedge p(y) \wedge p(z)$$
$$3 : p(x) \leftarrow t(x)$$

Y sea la siguiente regla restauradora:

R:

`Evento:`	$insertar(q(x))$
`Condición:`	$\neg p(x)$
`Acción:`	...

Para la obtención del árbol se generarían los siguientes nodos:

Nodo raíz:

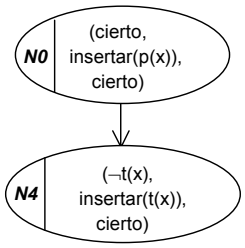
$$N0 = (cierto, insertar(p(x)), cierto)$$

Clanes sucesores del nodo N0:

Clan único:

$$N1 = (\neg v(x, y, z), insertar(v(x, y, z)), p(y) \wedge p(z)) \rightarrow No - permitido$$
$$N2 = (\neg p(y), insertar(p(y)), v(x, y, z) \wedge p(z)) \rightarrow No - permitido$$
$$N3 = (\neg p(z), insertar(p(z)), v(x, y, z) \wedge p(y)) \rightarrow No - permitido$$
$$N4 = (\neg t(x), insertar(t(x)), cierto)$$

El árbol de la regla restauradora sería el siguiente:



Esta regla restaura la consistencia insertando la instancia de $p(x)$ que se quiere a través de la regla que no es recursiva.

El criterio elegido para evitar el crecimiento infinito de un T-árbol supone que para la inserción de instancias de un predicado recursivo sólo se utilizarán las reglas deductivas no recursivas. En el caso del borrado se restaurará utilizando tan solo los predicados no recursivos de todas las reglas deductivas.

A continuación se presenta la definición de un T-árbol considerando sólo nodos permitidos. La definición coincide con la ya presentada (definición 9) excepto en los reglas 5a) y 5b) en las que, antes de introducir un nodo en un clan, se comprueba que éste sea permitido.

Definición 12 *T-árbol en bases de datos estratificadas.*

Un nodo de un T-árbol es un triplete de la forma (C_A, Op, C_D) donde C_A y C_D son el predicado *cierto* o una conjunción de literales y Op es $O(A)$ donde O es la operación *insertar* o *borrar* y A es un átomo, o bien Op es la operación especial *abortar*.

Dado un nodo, (C_A, Op, C_D) de un T-árbol, sus clanes sucesores se definen por las siguientes reglas:

1. Si $Op = abortar$, entonces el nodo no tiene clanes sucesores (i.e. es una hoja).
2. Si $Op = O(A)$, A es un átomo básico y $C_D = cierto$, entonces el nodo no tiene clanes sucesores (i.e. es una hoja).
3. Si $Op = O(A)$, A es un átomo básico y $C_D = M$, donde M es un literal, entonces el nodo tiene un clan sucesor, C, con un solo nodo generado por el literal M de la forma:
   (a) Si $M = \neg B$ entonces $C = \{(cierto, borrar(B), cierto)\}$
   (b) Si $M = B$ entonces $C = \{(cierto, insertar(B), cierto)\}$
4. Si $Op = O(A)$, A es un átomo básico y $C_D = A_1 \wedge \ldots \wedge A_n \wedge \neg B_1 \wedge \ldots \wedge \neg B_m$ donde $n + m > 1$ y tanto A_i $(1 \leq i \leq n)$ como B_j $(1 \leq j \leq m)$ son átomos entonces el nodo tiene sólo un clan sucesor, sea C, con $n + m$ nodos:
   $C = \{(\neg A_i, insertar(A_i), A_1 \wedge \ldots \wedge A_{i-1} \wedge A_{i+1} \wedge \ldots \wedge A_n \wedge \neg B_1 \wedge \ldots \wedge \neg B_m) | (1 \leq i \leq n)\} \cup \{(B_j, borrar(B_j), A_1 \wedge \ldots \wedge A_n \wedge \neg B_1 \wedge \ldots \wedge \neg B_{j-1} \wedge \neg B_{j+1} \wedge \ldots \wedge \neg B_m) | (1 \leq j \leq m)\}$.
5. Si $Op = O(A)$, A es un átomo derivado y hay p $(p \geq 1)$ reglas deductivas (refrescadas) de la forma $H_k \leftarrow A_{k,1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,m_k}$ $(1 \leq k \leq p$, $n_k + m_k \geq 1$ y tanto $A_{k,i}$ $(1 \leq i \leq n_k)$ como $B_{k,j}$ $(1 \leq j \leq m_k)$ son átomos) cuyas cabezas unifican con A^* a través de las substituciones restringidas θ_k, entonces:
   (a) Si $O = insertar$ el nodo tiene sólo un clan sucesor C. El clan C tiene N_C nodos $(N_C \leq \sum_{1 \leq k \leq p}(n_k + m_k))$:
   $C = \{(\neg A_{k,i}\theta_k, insertar(A_{k,i}\theta_k), (C_D \wedge A_{k,1} \wedge \ldots \wedge A_{k,i-1} \wedge A_{k,i+1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,m_k})\theta_k) \mid (\neg A_{k,i}\theta_k, insertar(A_{k,i}\theta_k), (C_D \wedge A_{k,1} \wedge \ldots \wedge A_{k,i-1} \wedge A_{k,i+1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,m_k})\theta_k)$ es un nodo permitido, $(1 \leq k \leq p)$ y $(1 \leq i \leq n_k)\}$ $\cup$
   $\{(B_{k,j}\theta_k, borrar(B_{k,j}\theta_k), (C_D \wedge A_{k,1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,j-1} \wedge \neg B_{k,j+1} \wedge \neg B_{k,m_k})\theta_k) \mid (B_{k,j}\theta_k, borrar(B_{k,j}\theta_k), (C_D \wedge A_{k,1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,j-1} \wedge \neg B_{k,j+1} \wedge \neg B_{k,m_k})\theta_k)$ es un nodo permitido, $(1 \leq k \leq p)$ y $(1 \leq j \leq m_k)\}$.
   (b) Si $O = borrar$ entonces el nodo tiene c clanes sucesores $(c \leq p)$, C_k $(1 \leq k \leq c)$. El clan C_k tiene N_{C_k} nodos $(N_{C_k} \leq (n_k + m_k))$:

$C_k = \{((A_{k,1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,m_k})\theta_k, borrar(A_{k,i}\theta_k), C_D\theta_k) \mid ((A_{k,1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,m_k})\theta_k, borrar(A_{k,i}\theta_k), C_D\theta_k)$ es un nodo permitido y $(1 \leq i \leq n_k)\}$ $\cup$
$\{((A_{k,1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,m_k})\theta_k,\ insertar(B_{k,j}\theta_k), C_D\theta_k) \mid ((A_{k,1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,m_k})\theta_k,\ insertar(B_{k,j}\theta_k), C_D\theta_k)$ es un nodo permitido y $(1 \leq j \leq m_k)\}$.

6. Si A es un átomo derivado y no hay reglas deductivas (refrescadas) cuya cabeza unifica con A^* entonces el nodo tiene un clan sucesor, C, con un solo un nodo: $C = \{(cierto, abortar, cierto)\}$.■

Es evidente que los T-árboles (de acuerdo con la definición 12) en el caso de bases de datos estratificadas son finitos al no admitirse la presencia de nodos que no sean permitidos. De nuevo en estos T-árboles pueden aparecer caminos que terminen en un nodo cuya operación sea *abortar*. Estos caminos deberán eliminarse depurando el T-árbol como ya se ha comentado anteriormente. Otra situación que obliga a depurar un T-árbol puede darse al no admitirse la inclusión de nodos que no sean permitidos, con lo que el T-árbol obtenido puede que no sea capaz de restaurar la consistencia. Esta situación se ilustra en el ejemplo siguiente.

Ejemplo 67 *Sea el siguiente conjunto de reglas deductivas:*

$1 : inc_W \leftarrow \neg p \wedge q(x)$
$2 : p \leftarrow p$

Y sea la siguiente regla restauradora:

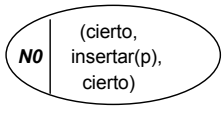
R:
`Evento:` $insertar(q(x))$
`Condición:` $\neg p$
`Acción:` ...

Para la obtención del árbol se generarían los siguientes nodos:

Nodo raíz:
$N0 = (cierto, insertar(p), cierto)$

Clanes sucesores del nodo N0:
Clan único:
$N1 = (\neg p, insertar(p), cierto) \rightarrow No - permitido$

Con lo que el T-árbol obtenido (de acuerdo con la definición 12) sería el que se muestra (es evidente que este T-árbol no restaura la consistencia).



Para eliminar todos los caminos de un T-árbol que no son capaces de restaurar la consistencia es necesario depurarlo. Este proceso se presenta en el apartado 4.3.5.

4.3.5 Depuración de un T-árbol

La depuración de un T-árbol es el proceso que elimina de éste los caminos que no son capaces de restaurar la consistencia; este proceso será necesario tanto en bases de datos jerárquicas como estratificadas. Antes de presentarlo, se introduce el concepto de *operación incompatible* y dos ejemplos más que ilustran otras situaciones que exigen la depuración de un T-árbol.

Definición 13 *Operación incompatible.*

Dadas dos operaciones O y O' se dice que O es incompatible con O' si el predicado de O coincide con el predicado de O' y la operación de O es *insertar* (resp. *borrar*) y la de O' es *borrar* (resp. *insertar*).■

Ejemplo 68 *Sea el siguiente conjunto de reglas deductivas:*

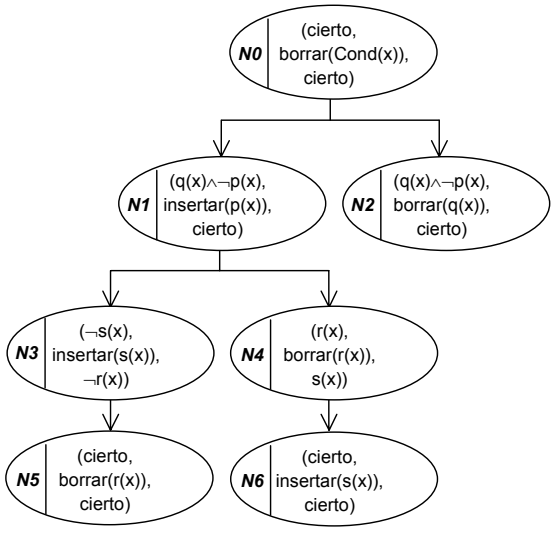
$$1 : inc_W \leftarrow q(x) \wedge \neg p(x)$$
$$2 : p(x) \leftarrow s(x) \wedge \neg r(x)$$

Y sea la siguiente regla restauradora:

R:

`Evento:` $insertar(r(x))$
`Condición:` $q(x) \wedge \neg p(x)$
`Acción:` ...

Cuya acción está representada en el siguiente T-árbol:



En este T-árbol, resulta evidente que los caminos $N0 - N1 - N3 - N5$ y $N0 - N1 - N4 - N6$ no son adecuados, ya que para restaurar deshacen la operación que disparó la regla ($insertar(r(x))$, por lo que habría que eliminarlos.

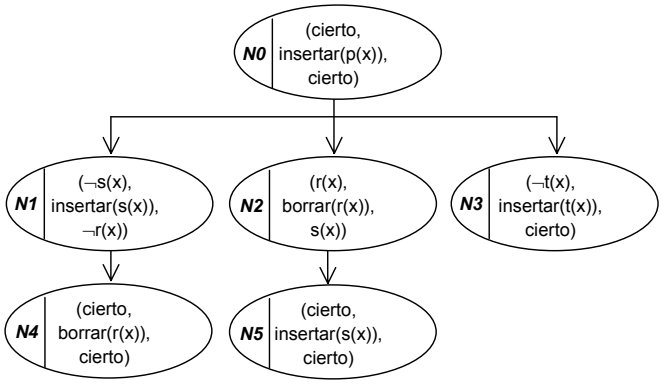
Ejemplo 69 *Sea una base de datos que contiene los hechos $\{s(1), r(1)\}$ y con el siguiente conjunto de reglas deductivas:*

$1 : inc_W \leftarrow q(x) \wedge \neg p(x)$
$2 : p(x) \leftarrow s(x) \wedge \neg r(x)$
$3 : p(x) \leftarrow t(x)$

Y sea la siguiente regla restauradora:

R:
`Evento:` $insertar(q(x))$
`Condición:` $\neg p(x)$
`Acción:` ...

Cuya acción está representada en el siguiente T-árbol:



La regla anterior se dispara tras la ejecución de una transacción que incluya las operaciones $\{borrar(s(1)), insertar(q(1))\}$. En este caso, los caminos $N0-N1-N4$ y $N0-N2-N5$ no son adecuados ya que para restaurar deshacen una de las operaciones incluidas en la transacción, $(borrar(s(1))$, por lo que habría que eliminarlos.

Resumiendo, el proceso de depuración de un T-árbol es necesario cuando se dé alguno de los cuatro casos siguientes:

1. En el T-árbol hay nodos distintos del raíz cuya operación es *abortar* (ejemplo 57).
2. En la generación del T-árbol ha habido nodos que no se han incluido al no ser permitidos evitando así caminos infinitos (ejemplo 67).
3. Alguna de las operaciones del T-árbol es incompatible con el evento que dispara la regla del T-árbol (ejemplo 68).
4. Alguna de las operaciones del T-árbol es incompatible con alguna de las operaciones ejecutadas desde el inicio de la transacción del usuario (ejemplo 69).

La depuración de un T-árbol, T, puede realizarse en cuatro pasos:

- Paso 1: Eliminar los nodos de T de la forma (C_A, O, C_D) que cumplen que O es incompatible con el evento de la regla a la que pertenece el T-árbol. También se deben eliminar los descendientes de estos nodos.

- Paso 2: Eliminar los nodos de T que no lleven a ninguna hoja *correcta,* que es aquella cuya operación es sobre un predicado básico y con la Condición-Después igual al predicado *cierto*.
- Paso 3: Eliminar los nodos de T que hayan perdido un clan completo en la definición o en pasos anteriores. También se deben eliminar los descendientes de estos nodos.
- Paso 4: Si T se ha quedado vacío entonces la restauración no es posible y la regla debe abortar la transacción del usuario.

Estos cuatro pasos deben repetirse hasta que ya no se pueda eliminar ningún nodo. A continuación se presenta un algoritmo que implementa este proceso.

Algoritmo 6 *Algoritmo para depurar un T-árbol.*

```
ALGORITMO Depurar_T-Árbol
ENTRADA
    T : T-árbol;
    C : Conjunto de operaciones prohibidas;
    seguir : Lógico;
SALIDA
    T : T-árbol;
    seguir : Lógico;
INICIO
seguir := falso;
/*Paso 1*/
SI existe alguna operación de C incompatible con alguna
| operación de algún nodo de T
ENTONCES
| seguir := cierto;
| T := Eliminar junto con sus descendientes los nodos de T
|     cuya operación sea incompatible con alguna operación de C
FIN_SI;
/*Paso 2*/
SI existen nodos en T que no llevan a ninguna hoja correcta
ENTONCES
| seguir := cierto;
| T := Eliminar los nodos de T que no llevan a ninguna hoja correcta
FIN_SI;
/*Paso 3*/
SI existen nodos en T que han perdido un clan completo
ENTONCES
| seguir := cierto;
| T := Eliminar junto con sus descendientes los nodos de T
|     que han perdido un clan completo
FIN_SI;
/*Paso 4*/
SI T se ha quedado vacío
ENTONCES
| seguir := falso;
```

| $T :=$ ($cierto, abortar, cierto$)
FIN_SI;
SI *seguir*
ENTONCES
| Depurar_T-Árbol($T, \emptyset, seguir$)
FIN_SI
FIN.■

En cada caso de los enumerados anteriormente, el T-árbol se depuraría como sigue:

caso 1) La depuración puede realizarse en tiempo de definición con la llamada *Depurar_T-Árbol*(T,$\emptyset$, $cierto$)

caso 2) La depuración puede realizarse en tiempo de definición con la llamada *Depurar_T-Árbol*(T,$\emptyset$, $cierto$)

caso 3) La depuración puede realizarse en tiempo de definición con la llamada *Depurar_T-Árbol*($T, \{E\}, cierto$) donde E es el evento de la regla restauradora del T-árbol;

caso 4) La depuración depende de la transacción que haya realizado el usuario por lo que debería realizarse en tiempo de ejecución antes de procesar la regla con la llamada *Depurar_T-Árbol*($T, C, cierto$) donde C es el conjunto de operaciones realizadas por la transacción del usuario hasta ese momento.

Ejemplo 70 *El T-árbol de la regla del ejemplo 57 quedaría, como ya se mostró, como un T-árbol con un sólo nodo ($cierto, abortar, cierto$), ya que no hay restauración posible.*

Ejemplo 71 *El T-árbol de la regla del ejemplo 67 se depuraría como se muestra a continuación:*

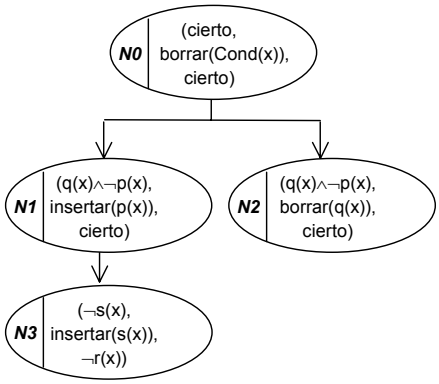
- *Paso 1: No es aplicable en este caso al estar vacío el conjunto de operaciones prohibidas.*
- *Paso 2: El $N0$ no lleva a ninguna hoja correcta por lo que debe eliminarse.*
- *Paso 3: No es aplicable.*
- *Paso 4: El T-árbol se ha quedado vacío por lo que se sustituye por uno con un único nodo ($cierto$,$abortar$,$cierto$).*

La regla definitiva es la siguiente:

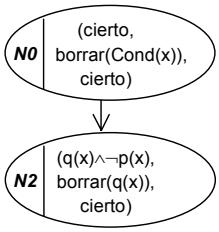
R:
Evento: $insertar(q(x))$
Condición: $\neg p$
Acción: ($cierto, abortar, cierto$)

Ejemplo 72 *El T-árbol de la regla del ejemplo 68 se depuraría como se muestra a continuación:*

- *Paso 1: Los nodos $N4$ y $N5$ deben eliminarse ya que su operación es incompatible con el evento de la regla. El nodo $N6$ también debe eliminarse ya que es descendiente de $N4$.*



- *Paso 2: Los nodos $N1$ y $N3$ no llevan a ninguna hoja correcta por lo que deben eliminarse.*

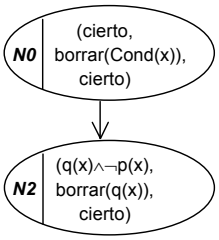


- *Los pasos 3 y 4 no son aplicables en este ejemplo.*

La regla definitiva es entonces la siguiente:

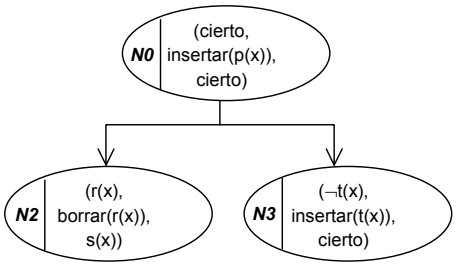
R:

`Evento:` $insertar(r(x))$
`Condición:` $q(x) \wedge \neg p(x)$
`Acción:`

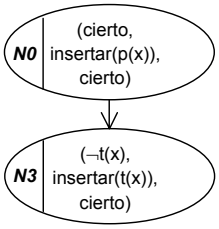


Ejemplo 73 *Dada la transacción $\{borrar(s(1)), insertar(q(1))\}$, el T-árbol de la regla del ejemplo 69 se depuraría como se muestra a continuación:*

- *Paso 1: Los nodos $N1$ y $N5$ deben eliminarse ya que su operación es incompatible con una de las operaciones de la transacción. El nodo $N4$ también debe eliminarse ya que es descendiente de $N1$.*



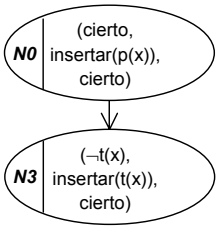
- *Paso 2: El nodo $N2$ no lleva a ninguna hoja correcta por lo que debe eliminarse.*



- *Los pasos 3 y 4 no son aplicables en este ejemplo.*

La regla definitiva es la siguiente:

R:
Evento: $insertar(q(x))$
Condición: $\neg p(x)$
Acción:



Dado que en los ejemplo anteriores no se ha visto la necesidad del paso 3 del procedimiento de depuración, éste se muestra a continuación.

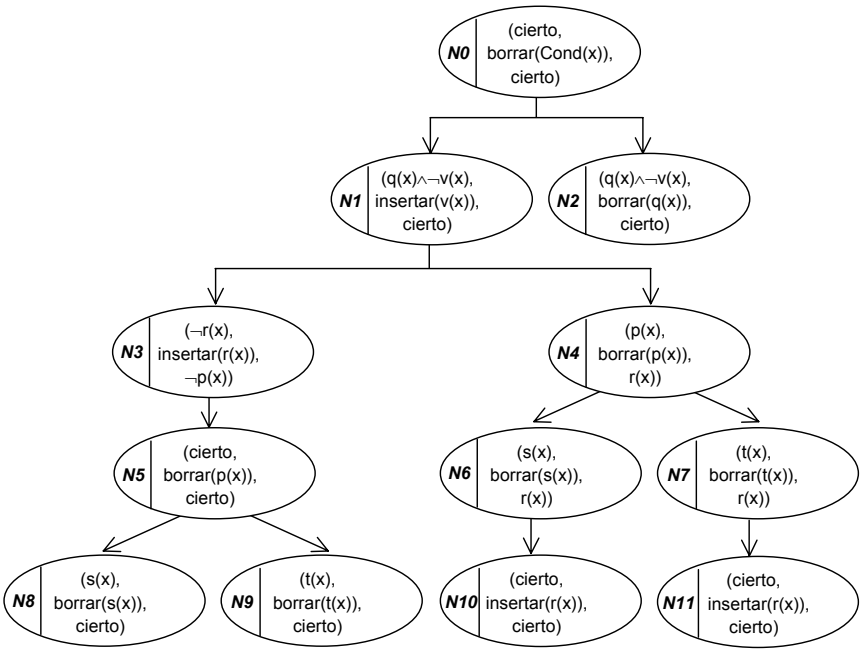
Ejemplo 74 *Sea el siguiente conjunto de reglas deductivas:*

$$1 : inc_W \leftarrow q(x) \wedge \neg v(x)$$
$$2 : v(x) \leftarrow r(x) \wedge \neg p(x)$$
$$3 : p(x) \leftarrow s(x)$$
$$4 : p(x) \leftarrow t(x)$$

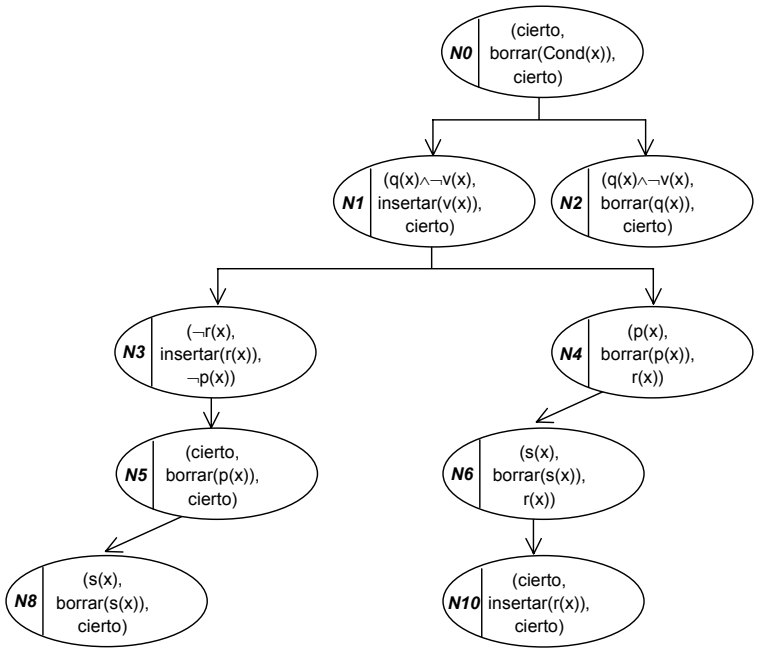
Y sea la siguiente regla restauradora:

R:
Evento: $insertar(t(x))$
Condición: $q(x) \wedge \neg v(x)$
Acción: ...

Cuya acción está representada en el siguiente T-árbol:

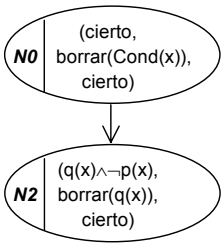


Aplicando el procedimiento de depuración sin el paso 3 se obtiene el siguiente T-árbol:



Si la base de datos contiene los hechos $\{r(1), q(1)\}$*, ante la inserción del hecho* $t(1)$ *es fácil ver que este T-árbol no es correcto ya que los caminos que empiezan en*

$N1$ no restaurarían la consistencia. Aplicando el paso 3, los nodos $N4$ y $N5$ y todos sus descendientes deben desaparecen ya que han perdido un clan completo. En una nueva iteración del algoritmo de depuración desaparecen los nodos $N3$ y $N1$ porque no llevan a ninguna hoja correcta. El T-árbol definitivo obtenido sería:



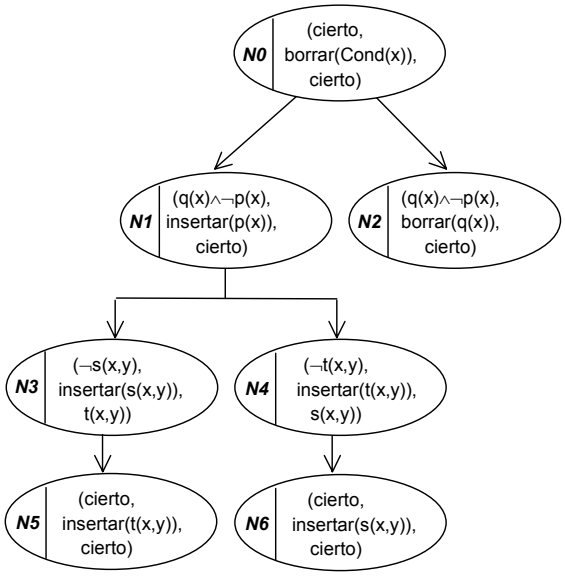
El criterio de depuración elegido es conservador en el sentido de que se van a eliminar caminos que podrían restaurar la consistencia con operaciones sobre el mismo predicado que el evento pero con instancias distintas.

Ejemplo 75 *Sea el siguiente conjunto de reglas deductivas:*

$$1 : inc_W \leftarrow q(x) \wedge \neg p(x)$$
$$2 : p(x) \leftarrow s(x,y) \wedge t(x,y)$$

Y sea la siguiente regla restauradora en la que aún no se ha depurado el T-árbol:

R:
`Evento:` $borrar(t(x,y))$
`Condición:` $q(x) \wedge \neg p(x)$
`Acción:`



Supóngase que la base de datos contiene los siguientes hechos:

$$\{s(1,2), s(1,4), t(1,2), t(1,3), q(1)\}$$

Y supóngase que la regla anterior se dispara por la ejecución de una transacción que incluya la operación borrar$(t(1,2))$*. El camino formado por los nodos* $N0-N1-N3-N5$ *podría restaurar la consistencia insertando* $s(1,3)$ *y el camino formado por los nodos* $N0-N1-N4-N6$ *podría restaurar la consistencia insertando* $t(1,4)$*. Para ambos caminos, la instancia* $\{x/1, y/2\}$ *debe estar prohibida. Tras la poda del T-árbol, estas restauraciones no se encuentran.*

Para permitir las restauraciones proporcionadas por caminos con nodos incompatibles con el evento, el algoritmo 5 de instanciación presentado anteriormente tendría que revisarse para proporcionar instancias diferentes a la que disparó la regla; por este motivo se ha preferido la depuración conservadora.

4.3.6 Recorrido de un T-árbol en bases de datos estratificadas

El recorrido de un T-árbol generado según la definición 12 para bases de datos estratificadas contemplando la posibilidad de que haya variables existencialmente cuantificadas, se puede realizar sin problemas con el algoritmo 4 ya que el hecho de permitir bases de datos deductivas estratificadas no modifica la forma de recorrer los T-árboles, sólo la forma de generarlos. El resultado de corrección del recorrido de un T-árbol en este caso se enuncia a continuación.

Teorema 5 *Corrección del algoritmo de recorrido de un T-árbol en bases de datos estratificadas.*

⇒ Sea D una base de datos estratificada y sea D' la salida de la llamada al algoritmo *Recorrer_T-Árbol*$((C_A^0,\ insertar(A^0),\ C_D^0),\ D,\ D_{inicial},\ D')$ (resp. *Recorrer_T-Árbol*$((C_A^0,\ borrar(A^0),\ C_D^0),\ D,\ D_{inicial},\ D')$) donde A^0 es base y C_A^0 es cierto en D,

⇒ entonces se cumple que $comp(D') \vDash A^0$ (resp. $comp(D') \vDash \neg A^0$).

Demostración:

La demostración de este teorema coincide con la demostración del teorema 4.■

4.4 Generación del conjunto de reglas restauradoras

4.4.1 Algoritmo de generación de las reglas restauradoras

En el apartado 4.3.2 se presentó un algoritmo (algoritmo 2) que permitía la obtención del conjunto de reglas restauradoras (sin incluir la acción) para un esquema deductivo. A continuación, se presenta la versión definitiva de este algoritmo en el que se contempla ya la obtención de la acción siguiendo las ideas expuestas a lo largo de todo este capítulo.

Algoritmo 7 *Algoritmo de generación de reglas restauradoras.*

```
ALGORITMO Generación_Reglas_Restauradoras_3
ENTRADA
    AI_POS : Conjunto de actualizaciones inducidas según la definición 5;
    BDI : Conjunto de reglas deductivas;
SALIDA:
    ℜ : Conjunto de reglas restauradoras;
VARIABLES
    AI_POS^reducido : Conjunto de actualizaciones inducidas;
    ℜ_acción^sin : Conjunto de reglas restauradoras sin acción;
    T : T-árbol;
    i : Entero;
INICIO
ℜ_acción^sin := ∅;
i := 1;
AI_POS^reducido := ∅;
PARA CADA (P, O, C_D, L) ∈ AI_POS HACER
| SI no existe un (P', O', C'_D, L') en AI_POS^reducido y una substitución α de
| |    variable por variable tal que Pα = P'α y Oα = O'α y C_Dα = C'_Dα
| ENTONCES
| | AI_POS^reducido := AI_POS^reducido ∪ {(P, O, C_D, L)}
| FIN_SI
FIN_PARA;
PARA CADA (P, O, C_D, L) ∈ AI_POS^reducido HACER
| SI el predicado de P es el predicado de inconsistencia inc_W
| ENTONCES
| | ℜ_acción^sin := ℜ_acción^sin ∪ {R_i_inc_W(Evento:O, Condición:C_D, Acción:...)};
| | i := i + 1;
| FIN_SI
FIN_PARA;
ℜ := ∅;
PARA CADA R(Evento:O, Condición:C_D, Acción:...) ∈ ℜ_acción^sin HACER
| T := árbol_vacío[25];
| Generar_T-Árbol(T, C_D, BDI);
| Depurar_T-Árbol(T, O, cierto);
| ℜ := ℜ ∪ {R(Evento:O, Condición:C_D, Acción:T)};
FIN_PARA
FIN.■
```

A continuación se presenta el algoritmo para la generación de T-árboles a partir de la Condición-Después de una regla restauradora.

Algoritmo 8 *Algoritmo de generación de un T-árbol.*

```
ALGORITMO Generar_T-Árbol
ENTRADA
    T : T-árbol;
    C : Condición de una regla restauradora;
    BDI : Conjunto de reglas deductivas;
SALIDA
```

[25] *Árbol_vacío* es una constante de tipo T-árbol que representa un T-árbol sin nodos.

```
    T : T-árbol;
INICIO
SI C = cierto
ENTONCES
| T := (cierto, abortar, cierto)
SI NO
| SI C = L y L es un átomo positivo básico
| ENTONCES
| | T := (cierto, borrar(L), cierto)
| SI NO
| | SI C = L y L es un átomo negativo básico L = ¬A
| | ENTONCES
| | | T := (cierto, insertar(A), cierto)
| | SI NO
| | | SI C = L y L es un átomo positivo derivado
| | | ENTONCES
| | | | T := (cierto, borrar(L), cierto);
| | | | Generar_clanes((cierto, borrar(L), cierto), T, BDI)
| | | SI NO
| | | | SI C = L y L es un átomo negativo derivado L = ¬A
| | | | ENTONCES
| | | | | T := (cierto, insertar(A), cierto);
| | | | | Generar_clanes((cierto, insertar(A), cierto), T, BDI)
| | | | SI NO
| | | | | SI C = L1 ∧ ... ∧ Ln, n > 1 y {x1, ..., xm} son todas las
| | | | | |    variables que aparecen en C
| | | | | ENTONCES
| | | | | | BDI := BDI ∪ {Cond(x1, ..., xm) ← L1 ∧ ... ∧ Ln};
| | | | | | T := (cierto, borrar(Cond(x1, ..., xm)), cierto);
| | | | | | Generar_clanes((cierto, borrar(Cond(x1, ..., xm)), cierto), T, BDI)
| | | | | FIN_SI
| | | | FIN_SI
| | | FIN_SI
| | FIN_SI
| FIN_SI
FIN_SI;
Depurar_T-Árbol(T,∅,cierto);
FIN.■
```

El algoritmo Generar_clanes utilizado, extiende un T-árbol generando todos los clanes sucesores del nodo de entrada siguiendo la definición 12. En él, se hace uso de otro algoritmo llamado *Incluir_clan*. El algoritmo Incluir_clan, que no se hace explícito, recibe como entrada un T-árbol, T, un nodo de ese T-árbol, N, y un conjunto de nodos, C, y devuelve el T-árbol T modificado al incluir el conjunto de nodos C como clan sucesor del nodo N.

Algoritmo 9 *Algoritmo de generación de clanes sucesores de un nodo en un T-árbol.*

```
ALGORITMO Generar_clanes
ENTRADA
```

$N = (C_A, O(A), C_D)$: Nodo de un T-árbol;
T : T-árbol;
BDI : Conjunto de reglas deductivas;
SALIDA
T : T-árbol;
VARIABLES
$Clan$: Conjunto de nodos;
INICIO
SI A es un átomo positivo básico y $C_D = cierto$
ENTONCES
| $Parar$ /*El nodo no tiene clanes sucesores*/
FIN_SI;
SI A es un átomo positivo básico, $C_D = L$ y L es un literal
ENTONCES
| SI $L = \neg B$ y B es un átomo
| ENTONCES
| | Incluir_clan$(T, N, \{(cierto, borrar(B), cierto)\})$;
| | Generar_clanes$((cierto, borrar(B), cierto), T, BDI)$
| SI NO /*$L = B$ y B es un átomo*/
| | Incluir_clan$(T, N, \{(cierto, insertar(B), cierto)\})$;
| | Generar_clanes$((cierto, insertar(B), cierto), T, BDI)$
| FIN_SI
FIN_SI;
SI A es un átomo positivo básico, $C_D = A_1 \wedge \ldots \wedge A_n \wedge \neg B_1 \wedge \ldots \wedge \neg B_m$
| donde $n + m > 1$ y tanto A_i $(1 \leq i \leq n)$ como B_j $(1 \leq j \leq m)$ son átomos
ENTONCES
| $Clan := \{(\neg A_i, insertar(A_i), A_1 \wedge \ldots \wedge A_{i-1} \wedge A_{i+1} \wedge \ldots \wedge A_n \wedge \neg B_1 \wedge \ldots \wedge \neg B_m) |$
| $(1 \leq i \leq n)\} \cup$
| $\{(B_j, borrar(B_j), A_1 \wedge \ldots \wedge A_n \wedge \neg B_1 \wedge \ldots \wedge \neg B_{j-1} \wedge \neg B_{j+1} \wedge \ldots \wedge \neg B_m) |$
| $(1 \leq j \leq m)\}$;
| Incluir_clan$(T, N, Clan)$;
| PARA CADA nodo M de $Clan$ HACER
| | Generar_clanes(M, T, BDI)
| FIN_PARA
FIN_SI;
SI A es un átomo derivado y hay p $(p \geq 1)$ reglas deductivas (refrescadas):
| $H_k \leftarrow A_{k,1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,m_k}$ cuyas cabezas unifican con
| A^* a través de las substituciones restringidas θ_k $(1 \leq k \leq p,\ n_k + m_k \geq 1$
| y tanto $A_{k,i}$ $(1 \leq i \leq n_k)$ como $B_{k,j}$ $(1 \leq j \leq m_k)$ son átomos)
ENTONCES
| SI $O = insertar$
| ENTONCES
| | $Clan := \{(\neg A_{k,i}\theta_k, insertar(A_{k,i}\theta_k), (C_D \wedge A_{k,1} \wedge \ldots \wedge A_{k,i-1} \wedge A_{k,i+1} \wedge \ldots \wedge A_{k,n_k} \wedge$
| | $\neg B_{k,1} \wedge \ldots \wedge \neg B_{k,m_k})\theta_k) \mid (\neg A_{k,i}\theta_k, insertar(A_{k,i}\theta_k), (C_D \wedge A_{k,1} \wedge \ldots \wedge A_{k,i-1} \wedge$
| | $A_{k,i+1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,m_k})\theta_k)$ es un nodo permitido,
| | $(1 \leq k \leq p)$ y $(1 \leq i \leq n_k)\} \cup$
| | $\{(B_{k,j}\theta_k, borrar(B_{k,j}\theta_k), (C_D \wedge A_{k,1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,j-1} \wedge \neg B_{k,j+1} \wedge$
| | $\neg B_{k,m_k})\theta_k) \mid (B_{k,j}\theta_k, borrar(B_{k,j}\theta_k), (C_D \wedge A_{k,1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge$
| | $\neg B_{k,j-1} \wedge \neg B_{k,j+1} \wedge \neg B_{k,m_k})\theta_k)$ es un nodo permitido, $(1 \leq k \leq p)$ y
| | $(1 \leq j \leq m_k)\}$;
| | Incluir_clan$(T, N, Clan)$;

| | PARA CADA nodo M de $Clan$ HACER
| | | Generar_clanes(M, T, BDI)
| | FIN_PARA
| SI NO /*$O = borrar$*/
| | PARA k DESDE 1 HASTA p HACER
| | | $Clan := \{((A_{k,1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,m_k})\theta_k, borrar(A_{k,i}\theta_k), C_D\theta_k) \mid$
| | | $((A_{k,1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,m_k})\theta_k, borrar(A_{k,i}\theta_k), C_D\theta_k)$ es
| | | un nodo permitido y $(1 \leq i \leq n_k)\}$ $\cup$
| | | $\{((A_{k,1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,m_k})\theta_k,\ insertar(B_{k,j}\theta_k), C_D\theta_k) \mid$
| | | $((A_{k,1} \wedge \ldots \wedge A_{k,n_k} \wedge \neg B_{k,1} \wedge \ldots \wedge \neg B_{k,m_k})\theta_k,\ insertar(B_{k,j}\theta_k), C_D\theta_k)$ es
| | | un nodo permitido y $(1 \leq j \leq m_k)\}$;
| | | Incluir_clan($T, N, Clan$);
| | | PARA CADA nodo M de $Clan$ HACER
| | | | Generar_clanes(M, T, BDI)
| | | FIN_PARA
| | FIN_PARA
| FIN_SI
FIN_SI;
SI A es un átomo derivado y no hay reglas deductivas (refrescadas)
| cuya cabeza unifique con A^*
ENTONCES
| $T := (cierto, abortar, cierto)$;
FIN_SI;
FIN.■

A continuación, en el teorema 6, se demuestra la corrección de una regla restauradora obtenida con el algoritmo 7. Este resultado significa que el procesamiento del T-árbol de una regla disparada cuya condición es cierta en un estado de base de datos genera un estado en el cual esa condición ya no se cumple. Dado que cuando la condición de una regla disparada se cumple es porque existe un camino de derivación para el átomo de inconsistencia asociado a esa regla, se puede concluir que el procesamiento del T-árbol elimina ese camino.

Teorema 6 *Corrección de una regla restauradora.*

⇒ Sean $R(Evento{:}\ O, Condición{:}\ C, Acción{:}\ T)$ una regla restauradora obtenida con el algoritmo 7, $(cierto, O^0(A^0), cierto)$ el nodo raíz de T y O' una operación de inserción o borrado de un hecho donde:

- α es una substitución tal que $O\alpha = O'$,
- D es el estado de base de datos resultante de aplicar la operación O' a un estado $D_{inicial}$,
- existe una refutación $SLDNF$ para $D \cup \{\leftarrow C\alpha\}$ con respuesta computada β, y
- D' es el estado de salida de la llamada *Recorrer_T-Árbol*$((cierto, O^0(A^0\alpha\beta), cierto), D, D_{inicial}, D')$,

⇒ entonces existe un árbol $SLDNF$ fallado finitamente para $D' \cup \{\leftarrow C\alpha\beta\}$

Demostración:

La demostración se va a realizar estudiando el nodo raíz del T-árbol T que se obtiene a partir de la condición de la regla.

- Si $C = A_1$ y A_1 es un átomo básico entonces el T-árbol asociado consta de un solo nodo que es (*cierto*, $borrar(A_1)$, *cierto*) y la llamada que se realiza es *Recorrer_T-Árbol*((*cierto*, $borrar(A_1\alpha\beta)$, *cierto*), D, $D_{inicial}$, D'). Por el teorema 5, se cumple que $comp(D') \vDash \neg A_1\alpha\beta$ por lo que existe un árbol $SLDNF$ fallado finitamente para $D' \cup \{\leftarrow A_1\alpha\beta\}$ y por tanto para $D' \cup \{\leftarrow C\alpha\beta\}$.
- Si $C = \neg A_1$ y A_1 es un átomo básico entonces el T-árbol asociado consta de un solo nodo que es (*cierto*, $insertar(A_1)$, *cierto*) y la llamada que se realiza es *Recorrer_T-Árbol*((*cierto*, $insertar(A_1\alpha\beta)$, *cierto*), D, $D_{inicial}$, D'). Por el teorema 5, se cumple que $comp(D') \vDash A_1\alpha\beta$ por lo que existe un árbol $SLDNF$ fallado finitamente para $D' \cup \{\leftarrow \neg A_1\alpha\beta\}$ y por lo tanto para $D' \cup \{\leftarrow C\alpha\beta\}$.
- Si $C = A_1$ y A_1 es un átomo derivado entones el T-árbol se obtendrá a partir del nodo raíz (*cierto*, $borrar(A_1)$, *cierto*) y la llamada que se realiza es *Recorrer_T-Árbol*((*cierto*, $borrar(A_1\alpha\beta)$, *cierto*), D, $D_{inicial}$, D'). Por el teorema 5, se cumple que $comp(D') \vDash \neg A_1\alpha\beta$ por lo que existe un árbol $SLDNF$ fallado finitamente para $D' \cup \{\leftarrow A_1\alpha\beta\}$ y por tanto para $D' \cup \{\leftarrow C\alpha\beta\}$.
- Si $C = \neg A_1$ y A_1 es un átomo derivado entonces el T-árbol se obtendrá a partir del nodo raíz (*cierto*, $insertar(A_1)$, *cierto*) y la llamada que se realiza es *Recorrer_T-Árbol*((*cierto*, $insertar(A_1\alpha\beta)$, *cierto*), D, $D_{inicial}$, D'). Por el teorema 5, se cumple que $comp(D') \vDash A_1\alpha\beta$ por lo que existe un árbol $SLDNF$ fallado finitamente para $D' \cup \{\leftarrow \neg A_1\alpha\beta\}$ y por lo tanto para $D' \cup \{\leftarrow C\alpha\beta\}$.
- Si $C = L_1 \wedge \ldots \wedge L_n$ y $n > 1$ y la regla deductiva auxiliar asociada a la condición es $Cond(x_1, \ldots, x_m) \leftarrow L_1 \wedge \ldots \wedge L_n$ entonces el T-árbol se obtendrá a partir del nodo raíz (*cierto*, $borrar(Cond(x_1, \ldots, x_m)\alpha\beta)$, *cierto*) y la llamada que se realiza es *Recorrer_T-Árbol*((*cierto*, $borrar(Cond(x_1, \ldots, x_m)\alpha\beta)$, *cierto*), D, $D_{inicial}$, D'). Por el teorema 5, se cumple que $comp(D') \vDash \neg Cond(x_1, \ldots, x_m)\alpha\beta$; teniendo en cuenta la regla deductiva $Cond(x_1, \ldots, x_m) \leftarrow L_1 \wedge \ldots \wedge L_n$ entonces existe un árbol $SLDNF$ fallado finitamente para $D' \cup \{\leftarrow L_1 \wedge \ldots \wedge L_n\alpha\beta\}$ y por lo tanto para $D' \cup \{\leftarrow C\alpha\beta\}$.■

4.4.2 Semántica operacional de una regla restauradora

Para terminar la presentación del método propuesto, en este apartado se muestra cómo es procesada una regla restauradora obtenida con el algoritmo 7 en tiempo de ejecución.

Sean D una base de datos deductiva, R(Evento: E, Condición: C, Acción: T) una regla restauradora, (C_A, O, C_D) el nodo raíz de T, y $Tr = \{O_1, \ldots, O_n\}$ una transacción donde O_i $(1 \leq i \leq n)$ es una operación básica base; entonces el sistema debe realizar las dos tareas que se describen a continuación:

1. Disparo de la regla. Si existe una operación en Tr, sea O_i, y una substitución α tal que $E\alpha = O_i$ entonces la regla R se dispara.

2. Procesamiento de la regla durante la ejecución del algoritmo de procesamiento de reglas del sistema activo. Cuando la regla R es seleccionada el sistema debe:

   (a) Evaluar la condición de la misma en la base de datos. Sea $\Theta = \{\theta | \theta$ es una respuesta computada para $D \cup \{\leftarrow C\alpha\}\}$.

   (b) Ejecutar la acción. Si $\Theta \neq \emptyset$ entonces para cada $\theta \in \Theta$ hay que aplicar la substitución $\alpha\theta$ a todos los nodos del T-árbol T y ejecutar la llamada Recorrer_T-Árbol($(C_A\alpha\theta, O\alpha\theta, C_D\alpha\theta)$, $Tr(D)$, $D_{inicial}$, D') (algoritmo 4).

4.5 Implementación del método en $SQL3$

En este apartado se pretende mostrar cómo se podría integrar el método propuesto en un sistema de bases de datos relacional compatible con el estándar $SQL3$ [31]. Para ello es necesario especificar cómo se van a representar las reglas restauradoras y cómo se van a procesar. Para ilustrar este punto se va a utilizar el siguiente ejemplo:

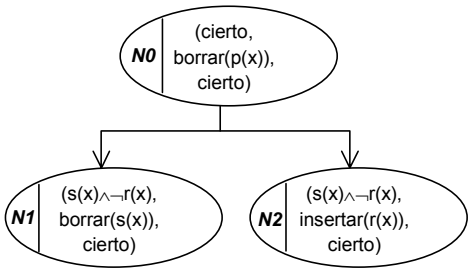
Ejemplo 76 *Sea el siguiente conjunto de reglas deductivas:*

$$1 : inc_W \leftarrow q(x) \wedge p(x)$$
$$2 : p(x) \leftarrow s(x) \wedge \neg r(x)$$

Las reglas restauradoras que se generarán para ese esquema son las siguientes:

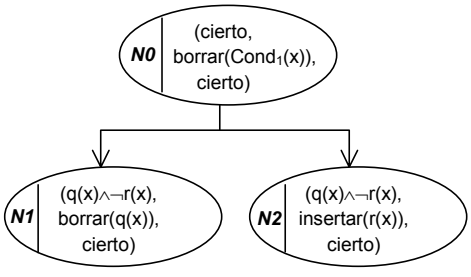
R_1:

`Evento:` $insertar(q(x))$
`Condición:` $p(x)$
`Acción:`

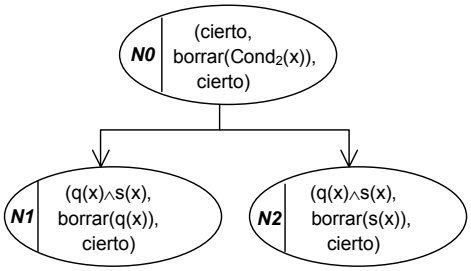


R_2:

`Evento:` $insertar(s(x))$
`Condición:` $q(x) \wedge \neg r(x)$
`Acción:`



R_3:
Evento: $borrar(r(x))$
Condición: $q(x) \wedge s(x)$
Acción:



Cada regla restauradora va a suponer la definición de un disparador (*trigger*) del tipo *"for each row"* (i.e. con granularidad orientada a la tupla) tal y como se muestra en los siguientes apartados. Estos disparadores estarán definidos en un esquema relacional por lo que habrá que traducir el esquema deductivo en esos términos.

Ejemplo 77 *El esquema relacional asociado al esquema del ejemplo 76 podría ser el siguiente:*

```
CREATE SCHEMA Ejemplo
CREATE TABLE q (a NUMBER PRIMARY KEY)
CREATE TABLE s (a NUMBER PRIMARY KEY)
CREATE TABLE r (a NUMBER PRIMARY KEY)
CREATE VIEW p AS (SELECT a
                  FROM s
                  WHERE NOT EXISTS (SELECT a FROM r
                                    WHERE r.a = s.a))
```

4.5.1 Representación del evento de una regla en un disparador

Esta traducción es la más sencilla ya que el evento de las reglas restauradoras se puede trasladar directamente al disparador.

Ejemplo 78 *Las tres reglas restauradoras del ejemplo 76 darían lugar a los tres disparadores que se muestran, inacabados, a continuación:*

```
CREATE TRIGGER R1
AFTER INSERT ON q
FOR EACH ROW
...
CREATE TRIGGER R2
AFTER INSERT ON s
FOR EACH ROW
...
CREATE TRIGGER R3
AFTER DELETE ON r
FOR EACH ROW
...
```

4.5.2 Representación de la condición de una regla en un disparador

Dado que la condición de un disparador en *SQL3* sólo permite expresar propiedades de la tupla que se ha insertado o borrado y teniendo en cuenta que las condiciones de las reglas restauradoras son más generales, éstas van a incluirse en el cuerpo del procedimiento del disparador.

Ejemplo 79 *Los tres disparadores del ejemplo 78 incluyendo ya la condición de las reglas serían los siguientes:*

```
CREATE TRIGGER R1
AFTER INSERT ON q
FOR EACH ROW
DECLARE
| aux NUMBER;
| condición BOOLEAN;
BEGIN
| SELECT COUNT(*) INTO aux
| FROM Dual[26]
| WHERE EXISTS (SELECT * FROM p
| |              WHERE p.a = new.a);
| condición := (aux > 0);
| IF condición
| THEN /*La condición de la regla se cumple*/
| | Ejecutar la acción
| END IF;
END.

CREATE TRIGGER R2
AFTER INSERT ON s
FOR EACH ROW
DECLARE
| aux NUMBER;
| condición BOOLEAN;
BEGIN
| SELECT COUNT(*) INTO aux
| FROM Dual
| WHERE EXISTS (SELECT * FROM q
| |              WHERE q.a = new.a AND
| |                     NOT EXISTS (SELECT * FROM r
| |                                 WHERE r.a = new.a));
| condición := (aux > 0);
| IF condición
| THEN /*La condición de la regla se cumple*/
| | Ejecutar la acción
| END IF;
END.
```

[26] Recuérdese que *Dual* es una tabla virtual del *SQL3* que permite interrogar una base de datos.

```
CREATE TRIGGER R3
AFTER DELETE ON r
FOR EACH ROW
DECLARE
| aux NUMBER;
| condición BOOLEAN;
BEGIN
| SELECT COUNT(*) INTO aux
| FROM FROM Dual
| WHERE EXISTS (SELECT * FROM q
| |               WHERE q.a = old.a AND
| |                   EXISTS (SELECT * FROM s
| |                           WHERE s.a = old.a));
| condición := (aux > 0);
| IF condición
| THEN /*La condición de la regla se cumple*/
| | Ejecutar la acción
| END IF;
END.
```

Así pues esta tarea consiste en la traducción de la condición de la regla restauradora, que está representada por una conjunción de literales, en una sentencia *select*.

4.5.3 Representación de la acción de una regla en un disparador

Para resolver esta parte, en primer lugar hay que determinar cómo se va a almacenar el T-árbol de cada regla. Una posible solución a este problema consiste en almacenar los árboles en una tabla de forma que cada nodo esté representado por una tupla de esa tabla y en el que las relaciones de un nodo con sus clanes se mantengan por referencias explícitas. Esta solución exige un orden entre los clanes sucesores de un nodo así como entre los nodos que forma un clan. Esa tabla, a la que se ha llamado *Bosque*, se cargará al definir la base de datos y tendrá los siguientes atributos:

- *Regla*: identificador de la regla restauradora a la que pertenece el nodo.
- *Nodo*: identificador del nodo dentro de la regla.
- *Caso*: puede tomar un valor entre 0 y 5 según el caso del problema de la falta de valores que se presenta en el nodo. El valor 0 indica que no se presenta este problema.
- C_A: sentencia de consulta a la base de datos que representa la Condición-Antes del nodo. También puede tomar el valor "cierto".
- *Operación*: sentencia de SQL de inserción o borrado sobre una relación básica. En caso de que el predicado de la operación del nodo sea derivado, este atributo tomará el valor "I" (resp. "B") para indicar una inserción (resp. un borrado).
- *Relación*: nombre de la relación a la que se va a aplicar la operación.

- C_D: sentencia de consulta a la base de datos que representa la Condición-Después del nodo. También puede tomar el valor "cierto".
- *Sucesor*: identificador del primer nodo del primer clan sucesor del nodo representado en esa tupla o la constante "*null*" que indica que no existe ese nodo.
- *Hermano*: identificador del siguiente nodo en el mismo clan al que pertenece el nodo representado en esa tupla o la constante "*null*" que indica que no existe ese nodo.
- *Clan*: identificador del primer nodo del siguiente clan sucesor del nodo representado en esa tupla o la constante "*null*" que indica que no existe ese nodo.

Ejemplo 80 *Los árboles del ejemplo 76 se representarían por las siguientes tuplas (las condiciones y operaciones que no se presentan completas en la tabla se pueden encontrar después de ésta):*

Regla	Nodo	Caso	C_A	Op.	Rel.	C_D	Suc.	Herm.	Cla
1	0	0	cierto	B	p	cierto	1	*null*	*null*
1	1	0	C_1	O_1	s	cierto	*null*	2	*null*
1	2	0	C_2	O_2	r	cierto	*null*	*null*	*null*
2	0	0	cierto	B	$Cond_1$	cierto	1	*null*	*null*
2	1	0	C_3	O_3	q	cierto	*null*	2	*null*
2	2	0	C_4	O_4	r	cierto	*null*	*null*	*null*
3	0	0	cierto	B	$Cond_2$	cierto	1	*null*	*null*
3	1	0	C_5	O_5	q	cierto	*null*	2	*null*
3	2	0	C_6	O_6	s	cierto	*null*	*null*	*null*

Donde las condiciones completas son las siguientes:

- Consultas C_1 y C_2:

```
SELECT a AS X
FROM s
WHERE a = new.a AND
    NOT EXISTS(SELECT * FROM r
                WHERE a = new.a)
```

- Consultas C_3 y C_4:

```
SELECT a INTO X
FROM q
WHERE a = new.a AND
    NOT EXISTS(SELECT * FROM r
                WHERE a = new.a)
```

- Consultas C_5 y C_6:

```
SELECT a INTO X
FROM q
WHERE a = old.a AND
    EXISTS(SELECT a FROM s
            WHERE a = old.a)
```

Las operaciones que no se han especificado completas son las siguientes:

- O_1 : DELETE FROM s WHERE $a = new.a$
- O_2 : INSERT INTO r VALUES ($new.a$)
- O_3 : DELETE FROM q WHERE $a = new.a$
- O_4 : INSERT INTO r VALUES ($new.a$)
- O_5 : DELETE FROM q WHERE $a = old.a$
- O_6 : DELETE FROM s WHERE $a = old.a$

Los disparadores definitivos son los siguientes:

```
CREATE TRIGGER R1
AFTER INSERT ON q
FOR EACH ROW
DECLARE
| nom_regla CONSTANT NUMBER := 1;
| TYPE Nodo IS RECORD
| | (Regla NUMBER, Nodo NUMBER, Caso NUMBER(1),
| |  CA VARCHAR, Operación VARCHAR, Relación VARCHAR,
| |  CD VARCHAR, Sucesor NUMBER, Hermano NUMBER, Clan NUMBER);
| TYPE Árbol IS TABLE OF Nodo;
| aux NUMBER;
| condición BOOLEAN;
| Tárbol ÁRBOL;
BEGIN
| SELECT COUNT(*) INTO aux
| FROM Dual
| WHERE EXISTS (SELECT * FROM p
| |              WHERE p.a = new.a);
| condición := (aux > 0);
| IF condición
| THEN /*La condición de la regla se cumple*/
| | Cargar(Tárbol,nom_regla);
| | Procesar (Tárbol);
| END IF;
END.
```

```
CREATE TRIGGER R2
AFTER INSERT ON s
FOR EACH ROW
DECLARE
| nom_regla CONSTANT NUMBER := 2;
| TYPE Nodo IS RECORD
| | (Regla NUMBER, Nodo NUMBER, Caso NUMBER(1),
| |  CA VARCHAR, Operación VARCHAR, Relación VARCHAR,
| |  CD VARCHAR, Sucesor NUMBER, Hermano NUMBER, Clan NUMBER);
| TYPE Árbol IS TABLE OF Nodo;
| aux NUMBER;
| condición BOOLEAN;
| Tárbol ÁRBOL;
BEGIN
| SELECT COUNT(*) INTO aux
| FROM Dual
```

```
| WHERE EXISTS (SELECT * FROM q
| |              WHERE q.a = new.a AND
| |                     NOT EXISTS (SELECT * FROM r
| |                                 WHERE r.a = new.a));
| condición := (aux > 0);
| IF condición
| THEN /*La condición de la regla se cumple*/
| | Cargar(Tárbol,nom_regla);
| | Procesar (Tárbol);
| END IF;
END.
```

```
CREATE TRIGGER R3
AFTER DELETE ON r
FOR EACH ROW
DECLARE
| nom_regla CONSTANT NUMBER := 3;
| TYPE Nodo IS RECORD
| | (Regla NUMBER, Nodo NUMBER, Caso NUMBER(1),
| |  CA VARCHAR, Operación VARCHAR, Relación VARCHAR,
| |  CD VARCHAR, Sucesor NUMBER, Hermano NUMBER, Clan NUMBER);
| TYPE Árbol IS TABLE OF Nodo;
| aux NUMBER;
| condición BOOLEAN;
| Tárbol ÁRBOL;
BEGIN
| SELECT COUNT(*) INTO aux
| FROM FROM Dual
| WHERE EXISTS (SELECT * FROM q
| |              WHERE q.a = old.a AND
| |                  EXISTS (SELECT * FROM s
| |                          WHERE s.a = old.a));
| condición := (aux > 0);
| IF condición
| THEN /*La condición de la regla se cumple*/
| | Cargar(Tárbol,nom_regla);
| | Procesar(Tárbol);
| END IF
END.
```

En el ejemplo anterior la ejecución de la acción de la regla restauradora se realiza con dos procedimientos:

- Cargar: este procedimiento recupera todos los nodos del árbol de la regla al inicializar la variable *Tárbol* con todas las filas de la tabla *Bosque* cuyo atributo *regla* coincida con el valor del parámetro *nom_regla*.

- Procesar: este procedimiento es el que realmente ejecuta la acción de la regla restauradora al recorrer el árbol de ésta (representado por las tuplas que se han cargado en *Tárbol*).

Estos dos procedimientos no se presentan en esta memoria, en el caso del primero por su sencillez y en el del segundo porque, dada la forma elegida para representar las condiciones y la operación de la regla, su tarea fundamental será el manejo de cadenas de caracteres que, aun siendo complejo, se aparta del interés del trabajo.

Para terminar este apartado hay que comentar que el ejemplo utilizado representa el caso más sencillo que se puede presentar por dos motivos:

- La acción de la regla se ejecutará nada más una vez, ya que la evaluación de la condición, si se cumple, sólo devuelve la substitución identidad[27];
- La ausencia de variables existencialmente cuantificadas simplifica la definición de las condiciones y de la operación del nodo al no ser necesario introducir variables.

En el caso más general, para la definición de los disparadores sería necesario el uso de cursores y de variables de tipo tabla para transmitir las instancias de las variables así como para resolver el problema de la falta de valores. De nuevo este punto no se ilustra por requerir una tarea de programación compleja en cuanto al manejo de cadenas de caracteres.

[27] Recuérdese que, tal y como se mostró en el ejemplo 52, la ejecución de la acción de una regla restauradora (i.e. el recorrido de su T-árbol) debe realizarse para cada instancia que devuelva la evaluación de su condición.

Capítulo 5

Conclusiones

En un sistema de base de datos, el cambio en el mundo real se modela mediante la ejecución de transacciones de usuario que modifican un estado generando otro que representa la nueva situación. Este nuevo estado puede violar las restricciones de integridad del esquema, restricciones que representan las propiedades de ese mundo. La reacción más frecuente de los sistemas de bases de datos ante la violación de la integridad consiste en rechazar la transacción que la ha provocado, devolviendo la base de datos al estado anterior a su ejecución. Esta solución tan simple es, sin embargo, usualmente poco satisfactoria para sistemas reales. Una alternativa a este comportamiento consiste en que el sistema modifique el estado inconsistente de forma que se repare la violación provocada por la transacción de usuario respetando los cambios propuestos por ésta. Se dice entonces que el sistema ha restaurado la consistencia de la base de datos.

En esta tesis se ha propuesto un método para la restauración de la consistencia en bases de datos relacionales con vistas (o bases de datos deductivas) que utiliza el lenguaje de reglas de un sistema activo. Así, a partir del conjunto de restricciones de integridad y del conjunto de reglas deductivas del esquema, el método genera un conjunto de reglas de actividad que restaura la consistencia de la base de datos cuando, como consecuencia de la ejecución de una transacción de usuario, se ha producido la violación de alguna restricción. Estas reglas se han denominado *reglas restauradoras*.

Para realizar el trabajo, y dada la estrecha relación que el problema de la actualización de una base de datos tiene con el problema de la restauración, otro objetivo cubierto por esta tesis ha sido la revisión de los métodos más relevantes que se han propuesto en la literatura para actualizar una base de datos deductiva.

Las características más destacables del nuevo método son:

- La obtención del conjunto de reglas restauradoras se realiza en tiempo de definición de la base de datos, descargando al sistema en tiempo de ejecución.
- Cuando es posible, se resuelve el problema de la falta de valores sin implicar al usuario.
- El método se pueden trasladar a un sistema relacional definido en $SQL3$, donde se utilizarían los disparadores (*triggers*) para implementar las reglas restauradoras.

- El método es correcto, ya que el procesamiento del T-árbol de una regla disparada cuya condición es cierta en un estado de base de datos genera un estado en el cual esa condición ya no se cumple.
- El método representa una mejora respecto a las propuestas existentes ya que obtiene las soluciones en tiempo de definición de datos contemplando, además, la presencia de reglas deductivas recursivas.

Como líneas de trabajo futuro se podrían considerar las siguientes:

- Extender el método para que sea capaz de considerar bases de datos más generales.
- Suavizar el proceso de depuración de los T-árboles para que no se pierdan restauraciones.
- Implementar el método en $SQL3$ (siguiendo las ideas esbozadas en el apartado 4.5), lo que permitiría transferir los resultados obtenidos a los sistemas comerciales. El desarrollo de esta idea exigiría la construcción de una herramienta de diseño que permitiese al diseñador simular el funcionamiento del conjunto de reglas restauradoras generadas y elegir el subconjunto de éstas más adecuado para mantener la consistencia. Para ello sería necesario permitir establecer prioridades entre las reglas e incluso entre los nodos de un T-árbol.
- Adaptar el método para resolver el problema de la actualización de vistas en una base de datos. Para ello sería necesario definir el T-árbol de inserción (resp. de borrado) para cada predicado derivado. En ese T-árbol estarán representadas todas las formas de inducir la inserción (resp. el borrado) de instancias de ese predicado.
- Trasladar los resultados obtenidos al campo del mantenimiento de almacenes de datos ("*data warehouse*") visto como un problema de materialización de vistas.

Apéndice A

Índice de definiciones, teoremas y algoritmos

A.1 Definiciones

En el capítulo 4 se han incluido las siguientes definiciones:

A.2 Teoremas y corolarios

En la presentación del método en el capítulo 4 se han propuesto los siguientes resultados:

A.3 Algoritmos

En la presentación del método en el capítulo 4 se han propuesto los siguientes algoritmos:

Apéndice B

Notación

A continuación se presentan algunos expresiones y abreviaturas que han sido utilizadas en el texto.

1. ? Representa el valor nulo.
2. $\overline{\forall}$ Cierre universal de una fórmula. Si $\{x_1, \ldots, x_n\}$ son las únicas variables libres de F entonces $\overline{\forall}F$ representa la fórmula $\forall x_1 \ldots \forall x_n F$.
3. $\overline{\exists}$ Cierre existencial de una fórmula. Si $\{x_1, \ldots, x_n\}$ son las únicas variables libres de F entonces $\overline{\exists}F$ representa la fórmula $\exists x_1 \ldots \exists x_n F$.
4. $\square$ Representa la cláusula vacía.
5. $A \leftarrow$ Sentencia de base de datos sin cuerpo. Esta expresión representa la fórmula $\overline{\forall}A$.
6. $\leftarrow F$ Esta expresión representa la fórmula $\overline{\forall}(\neg F)$.
7. $[\ldots]$ Los corchetes en una fórmula denotan opcionalidad.
8. $\overrightarrow{x}$ representa un vector de variables; $\overrightarrow{c}$ representa un vector de constantes; y $\overrightarrow{t}$ representa un vector de términos constantes o variables.
9. $comp(D)$ representa la compleción de D (ver el apartado D.2.1 en el apéndice D).
10. B_L es la Base de Herbrand del lenguaje de primer orden L que está formada por todos los átomos base que se puedan construir con los símbolos de L.
11. $\models_M F$ Representa que la fórmula F es cierta en el modelo M.
12. $Tr \models F$ Representa que la fórmula F es consecuencia lógica de la teoría Tr.
13. $\blacksquare$ Indica el fin de un definición, teorema, corolario o algoritmo.

Abreviaturas.

1. et al.: *et allii*, y otra gente.
2. i.e.: *id est*, esto es.
3. p.e.: por ejemplo.
4. resp.: respectivamente.

Apéndice C

Definiciones

En este apartado se presentan algunas definiciones de propiedades sintácticas para bases de datos que han sido utilizadas a lo largo de la tesis. Con el fin de hacer una presentación más homogénea en algunos casos no se ha respetado la definición original.

A continuación se presenta la relación de las propiedades definidas indicando el número de definición correspondiente.

- Aplicación de nivel: definiciones 12 y 13.
- Átomo positivo (negativo): definición 5.
- Átomo/Literal base: definición 1.
- Átomo/Literal básico (derivado):definición 2.
- Base de datos acíclica: definición 25.
- Base de datos de rango restringido: definición 29.
- Base de datos definida: definición 18.
- Base de datos estratificada: definición 22.
- Base de datos estricta: definición 26.
- Base de datos general: definición 20.
- Base de datos jerárquica: definición 21.
- Base de datos localmente consistente en llamada: definición 27
- Base de datos localmente estratificada: definición 24.
- Base de datos localmente jerárquica: definición 23.
- Base de datos normal: definición 19.
- Base de datos permitida: definición 28.

- Cláusula de base de datos: definición 4.
- Cláusula admisible: definición 17.
- Cláusula permitida: definición 15.
- Conjunción permitida: definición 16
- Dependencias entre átomos: definición 10
- Dependencias entre símbolos de predicado: definición 8.
- Forma negada de una restricción de integridad: definición 6.
- Grafo de dependencias entre los átomos de un conjunto de cláusulas: definición 9.
- Grafo de dependencias entre los símbolos de predicados de un conjunto de sentencias: definición 7.
- Hecho: definición 3.
- Regla deductiva: definición 3.
- Relación $\gg_C$: definición 11.
- Sentencia de base de datos: definición 3.
- Variable permitida en una cláusula: definición 14.

1. **Átomo/Literal base.**

   Con la expresión átomo o literal base se hace referencia a un átomo o literal que no contiene variables. Por extensión, una fórmula bien formada F es base si no contiene variables.

2. **Átomo/Literal básico (derivado)**

   Con la expresión átomo o literal básico (resp. derivado) se hace referencia a un átomo o literal construido con un predicado básico (resp. derivado).

3. **Sentencia de base de datos.**

   Una sentencia de base de datos (abreviadamente *sentencia*) es una fórmula de la forma A [$\leftarrow F$] donde A es un átomo y F, si existe, es una fórmula bien formada. En una sentencia todas las variables libres se consideran universalmente cuantificadas. La *cabeza* de la sentencia es A y el *cuerpo* es W. Si la sentencia no tiene cuerpo y es base entonces se llama *hecho,* en caso contrario es una *regla deductiva.*

4. **Cláusula de base de datos.**

   Una cláusula de base de datos (abreviadamente *cláusula*) es una sentencia de base de datos cuyo cuerpo es una conjunción de literales.

5. **Átomo positivo (negativo).**

   Un átomo A es positivo (resp. negativo) en la fórmula A (resp. $\neg A$).

   Si un átomo A es positivo (resp. negativo) en una fórmula W entonces también es positivo (resp. negativo) en $\exists xW$, en $\forall xW$, en $W \wedge V$, en $W \vee V$, y en $W \leftarrow V$.

   Si un átomo A es positivo (resp. negativo) en una fórmula W entonces es negativo (resp. positivo) en $\neg W$ y en $V \leftarrow W$.

6. **Forma negada de una restricción de integridad.**

   Dada una restricción de integridad de la forma W donde W es una fórmula bien formada cualquiera su forma negada es $\leftarrow inc_W$ donde inc_W es un nuevo símbolo de predicado que se define por la regla deductiva $inc_W \leftarrow \neg W$. Las cláusulas resultantes de aplicar al algoritmo de normalización de Lloyd y Topor [37] a esa regla deductiva deben ser permitidas y formar parte de cualquier estado de la base de datos. A los nuevos predicados inc_W se les denomina *predicados de inconsistencia*. A las reglas con un predicado de inconsistencia en la cabeza se les denomina *reglas de inconsistencia.*

   Con esta representación de las restricciones de integridad, un estado de base de datos D viola la restricción W si y sólo si satisface inc_W.

7. **Grafo de dependencias entre los símbolos de predicados de un conjunto de sentencias.**

   Sea S un conjunto de sentencias. El grafo de dependencias de S es un grafo dirigido en el que cada símbolo de predicado de S es un nodo. En el grafo hay un arco del nodo p al nodo q etiquetado con $(+)$ (resp. $(-)$) si el predicado q aparece en la cabeza de una sentencia de S en cuyo cuerpo aparece el predicado p en un átomo positivo (resp. negativo)[1].

8. **Dependencias entre símbolos de predicado.**

   Sea S un conjunto de sentencias y p y q dos símbolos de predicado de S. Se dice que p depende positivamente (resp. negativamente) de q si en el grafo de dependencias entre los símbolos de predicado de S hay al menos un camino desde p hasta q con un número par (resp. impar) de arcos etiquetados con el signo $(-)$.

9. **Grafo de dependencias entre los átomos de un conjunto de cláusulas.**

   Sea C un conjunto de cláusulas. El grafo de dependencias entre los átomos de C es un grafo dirigido definido como sigue:

   − Los nodos del grafo son los átomos base de la Base de Herbrand de C (B_C).

   − Sean A y B dos átomos base de B_C. Hay un arco etiquetado con el signo $(+)$ (resp. $(-)$) desde A hasta B si y sólo si hay una instancia base de una cláusula de C tal que A es la cabeza y B (resp. $\neg B$) aparece en el cuerpo de la instancia de la cláusula.

[1] Esta definición es la misma que se ha introducido en el capítulo 4 excepto que, por no ser relevante, el arco no se ha etiquetado con el número de la sentencia que lo genera.

10. **Dependencias entre átomos.**

    Sea C un conjunto de cláusulas y A y B dos átomos base de la Base de Herbrand de C (B_C). Se dice que A depende positivamente (resp. negativamente) de B si en el grafo de dependencias entre los átomos de C hay al menos un camino desde A hasta B con un número par (resp. impar) de arcos etiquetados con el signo $(-)$.

11. **Relación $\gg_C$**

    Sea C un conjunto de cláusulas. La relación $\gg_C$ se define entre los átomos de la Base de Herbrand de C (B_C) como sigue: para átomos base A y B de B_C, $A \gg_C B$ si A depende tanto positiva como negativamente de B.

12. **Aplicación de nivel de los predicados de una base de datos.**

    Sea D una base de datos. Una aplicación de nivel de los predicados de D es una aplicación de su conjunto de símbolos de predicado en el conjunto de los número ordinales. El *nivel* de un símbolo de predicado es el valor que le asigna esta aplicación.

13. **Aplicación de nivel de la Base de Herbrand de una base de datos.**

    Sea D una base de datos. Una aplicación de nivel de la Base de Herbrand de D es una aplicación de su Base de Herbrand en el conjunto de los número ordinales.

14. **Variable permitida en una cláusula.**

    Dada una cláusula F y una variable x, x es permitida en F si x ocurre en un literal positivo del cuerpo de F.

15. **Cláusula permitida.**

    Una cláusula F es permitida si cada variable de F es permitida en F.

16. **Conjunción permitida.**

    Una conjunción C es permitida si cada variable de C ocurre en un literal positivo de C.

17. **Cláusula admisible.**

    Una cláusula F es admisible si para cada variable x de F se cumple que x ocurre en la cabeza de F o x es permitida en F.

18. **Base de datos definida.**

    Una base de datos es definida si sus sentencias son cláusulas cuyo cuerpo sólo contiene literales positivos.

19. **Base de datos normal.**

    Una base de datos es normal si sus sentencias son cláusulas.

20. **Base de datos general.**

    Una base de datos es general si sus sentencias son fórmulas de la forma $A\ [\leftarrow W]$ donde A es un átomo y W es una fórmula cualquiera.

21. **Base de datos jerárquica.**

    Una base de datos es jerárquica si existe una aplicación de nivel de sus predicados tal que en cada sentencia de base de datos $A \leftarrow W$ el nivel de cada símbolo de predicado de W es menor que el nivel del símbolo de predicado de A.

22. **Base de datos estratificada.**

    Una base de datos es estratificada si existe una aplicación de nivel de sus predicados tal que en cada sentencia de base de datos $A \leftarrow W$ se cumple que

    − el nivel de cada símbolo de predicado de un átomo positivo de W es menor o igual que el nivel del símbolo de predicado de A,

    − el nivel de cada símbolo de predicado de un átomo negativo de W es menor que el nivel del símbolo de predicado de A.

23. **Base de datos localmente jerárquica.**

    Una base de datos D es localmente jerárquica si existe una aplicación de nivel de la Base de Herbrand de para D tal que, para toda instancia base $A \leftarrow B_1 \wedge \ldots \wedge B_n \wedge \neg C_1 \wedge \ldots \wedge \neg C_m$ de una cláusula de D, se cumple que $nivel(B_i) < nivel(A)$ $(1 \leq i \leq n)$ y que $nivel(C_j) < nivel(A)$ $(1 \leq j \leq m)$.

24. **Base de datos localmente estratificada.**

    Una base de datos D es localmente estratificada si existe una aplicación de nivel de la Base de Herbrand de D tal que, para toda instancia base $A \leftarrow B_1 \wedge \ldots \wedge B_n \wedge \neg C_1 \wedge \ldots \wedge \neg C_m$ de una cláusula de D, se cumple que $nivel(B_i) \leq nivel(A)$ $(1 \leq i \leq n)$ y que $nivel(C_j) < nivel(A)$ $(1 \leq j \leq m)$.

25. **Base de datos acíclica.**

    Una base de datos D es acíclica si es localmente jerárquica con respecto a una aplicación de nivel que asigna a cada átomo de la base de Herbrand de D un ordinal finito.

26. **Base de datos estricta.**

    Una base de datos D es estricta si no existen un par de símbolos de predicado p y q de D tal que p depende tanto positiva como negativamente de q.

27. **Base de datos localmente consistente en llamada.**

    Una base de datos D es localmente consistente en llamada si la relación $\gg_C$ es un orden parcial bien fundado.

28. **Base de datos permitida.**

    Una base de datos D es permitida si cada una de sus sentencias es una cláusula permitida. Una base de datos permitida es independiente del dominio.

29. **Base de datos de rango restringido.**

    Una base de datos D es de rango restringido si es permitida. Una base de datos de rango restringido es independiente del dominio.

Apéndice D

Semánticas de una base de datos deductiva

D.1 Introducción

Dar semántica a una base de datos significa definir cuál es la información implícitamente almacenada en ella. Existen dos tipos de semánticas, la semántica declarativa que define cuál es esta información y la semántica operacional que dice cómo computarla. Ambas semánticas están relacionadas por los conceptos de *corrección* y *completitud*. Se dice que un procedimiento es correcto si toda la información que computa es cierta en la semántica declarativa asociada, y es completo si toda la información cierta en la semántica declarativa asociada puede ser computada. En el apartado D.2 de este apéndice se presentan las semánticas declarativas propuestas para bases de datos definidas y normales y en el apartado D.3 las semánticas operacionales asociadas a ellas.

D.2 Semántica declarativa de una base de datos

Como ya se ha dicho, un estado de base de datos es un conjunto de sentencias de base de datos. Dada esta definición, de una base de datos no es posible derivar información negativa ya que dado un estado D y un átomo base A, $D \not\models \neg A$ ya que $D \cup \{A\}$ siempre es satisfactible. En bases de datos relacionales sin vistas, formadas sólo por hechos, este problema se resuelve aplicando la regla de inferencia conocida como *regla del mundo cerrado* (RMC) [51] que expresa la idea de que la información que no está explícitamente almacenada en la base de datos debe considerarse falsa:

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

En bases de datos deductivas sin embargo, la información contenida en la base de datos no está explícitamente almacenada; en éstas la aplicación de la RMC para

derivar $\neg A$ de D, exige demostrar que $D \nvdash A$ que es un problema indecidible en lógica de primer orden. Así, 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) [17]:

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

Las reglas de inferencia anteriores permiten derivar información negativa de una base de datos, pero hay que especificar cuál es la semántica declarativa respecto a la cual son correctas estas reglas, es decir cuál es 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 $\neg A$ es *cierto* en D si y sólo si $Tr \models \neg A$. La semántica en esta aproximación viene definida por el conjunto de literales base que son consecuencia lógica de Tr, $Sem\_Tr = \{L \mid L \text{ es un literal base tal que } Tr \models L\}$. A esta aproximación pertenece la semántica de la compleción [17].

- Aproximación por modelo minimal: consiste en seleccionar un modelo M entre todos los modelos minimales de Herbrand de D e interpretar la negación en dicho modelo, es decir $\neg A$ es *cierto* en D si y sólo si $\models_M \neg A$. La semántica en esta aproximación viene definida por el conjunto de literales base cierto en M, $Sem\_M = \{L \mid L \text{ es un literal base tal que } \models_M \neg A\}$. A esta aproximación pertenece la semántica del punto fijo iterado [2], la semántica del modelo perfecto [48], la semántica del modelo estable [27] y [56] y la semántica bien fundada [61].

D.2.1 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 que el conjunto de sentencias que tienen el mismo símbolo de predicado en la cabeza se interpreta como la definición *completa* de dicho predicado.

La compleción de una base de datos se obtiene añadiendo a un estado D los axiomas de la compleción para cada símbolo de predicado de su esquema 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 la compleción para un símbolo de predicado p en un estado de base de datos D se definen de la forma siguiente:

a) Si en D existen k ($k \geq 1$) sentencias con el predicado p en la cabeza de la forma: $p(t_{i,1}, \ldots, t_{i,n}) \leftarrow L_{i,1} \wedge \ldots \wedge L_{i,m_i}$ $(1 \leq i \leq k)$, entonces cada sentencia se substituye por $p(x_1, \ldots, x_n) \leftarrow \exists y_{i,1} \ldots \exists y_{i,j_i}(x_1 = t_{i,1} \wedge \ldots \wedge x_n = t_{i,n} \wedge L_{k,1} \wedge \ldots \wedge L_{i,m_i})$ donde $\{y_{i,1}, \ldots, y_{i,j_i}\}$ son las variables de la sentencia original.

Con estas transformaciones, el axioma de la compleción para p se define como $\forall x_1 \ldots \forall x_n(p(x_1, \ldots, x_n) \rightarrow (E_1 \vee \ldots \vee E_k))$ donde E_i $(1 \leq i \leq k)$ tiene la forma general: $E_i = \exists y_{i,1} \ldots \exists y_{i,j_i}(x_1 = t_{i,1} \wedge \ldots, x_n = t_{i,n} \wedge L_{i,1} \wedge \ldots \wedge L_{i,m_i})$.

b) Si en D no existe ninguna sentencia con el predicado p en la cabeza, entonces el axioma de compleción para p es: $\forall x_1 \ldots \forall x_n(\neg p(x_1, \ldots, x_n))$.

Los axiomas de la teoría de la igualdad son:

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

La teoría Tr que representa un estado de base de datos D en la semántica de la compleción es: $Tr = comp(D) = D \cup$ {axiomas de la compleción para cada predicado del esquema de D} $\cup$ {axiomas de la igualdad}. La semántica de la compleción viene definida por el conjunto de literales base que son consecuencia lógica de $Tr, COMP(D) = \{L \mid L$ es un literal base tal que $Tr \models L\}$.

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

- Si D es un estado de una base de datos definida y A es un átomo base, entonces se cumple:
  - $comp(D)$ es consistente,
  - $comp(D) \models A$ si y sólo si $D \models A$,
  - $comp(D) \models \neg A$ si y sólo si $A \in F_D$ donde F_D es el conjunto de fallo finito de D.
- Si D es un estado de una base de datos normal, entonces se cumple:
  - $comp(D)$ puede no ser consistente,
  - $comp(D) \models D$.

La semántica de la compleción fue introducida por Clark [17] 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 fallado finitamente para $D \cup \{R\}$ entonces $comp(D) \models R$".

D.2.2 Semántica del modelo minimal

La semántica por modelo minimal se basa en la idea de seleccionar entre todos los modelos minimales de Herbrand de una base de datos D aquél que captura la *semántica supuesta* de D. A continuación se presenta esta semántica en primer lugar para bases de datos definidas y luego para bases de datos normales.

Bases de datos definidas

Si D es una base de datos definida, existe un modelo mínimo de Herbrand de D, M_D, tal que $M_D = \{A \mid A \in B_L \text{ y } D \vDash A\}$. Este modelo 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: $T_D : 2^{B_L} \rightarrow 2^{B_L}$ y $T_D(I) = \{A \mid A \in B_L \text{ y } A \leftarrow A_1 \wedge \ldots \wedge A_n$ es una instancia base de una sentencia de D, y $\{A_1, \ldots, A_n\} \subseteq I\}$ siendo I una interpretación de Herbrand de D representada por un conjunto de átomos base. Si D es una base de datos definida y M_D es su modelo mínimo de Herbrand, entonces 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. Esta semántica se va a representar por el conjunto de literales base ciertos en M_D: $MIN(D) = \{L \mid L \text{ es un literal base y } \vDash_{M_D} L\}$.

La relación entre la semántica de la compleción y la semántica del modelo mínimo para bases de datos definidas es la siguiente: $COMP(D) \subseteq MIN(D)$.

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. Por este motivo, la semántica por modelo minimal para bases de datos normales viene definida por el conjunto de literales base ciertos en todo modelo minimal de D: $MIN(D) = \{L \mid L$ es un literal base y $\vDash_M L$ para todo modelo minimal de M de $D\}$.

Esta semántica es consistente con la regla de inferencia denominada *regla del mundo cerrado generalizada* ($RMCG$) [41] que consiste en:

$$RMCG: \quad \frac{\nvDash_M A \text{ para todo modelo minimal } M \text{ de } D}{\neg A}$$

La semántica del modelo minimal definida de esta forma no es sin embargo satisfactoria.

Ejemplo 81 *Sea D un estado de base de datos con una regla deductiva $p(x) \leftarrow \neg q(x) \wedge r(x)$ y un hecho $r(a)$. D tiene dos modelos minimales $M_1 = \{r(a), q(a)\}$ y $M_2 = \{r(a), p(a)\}$ por lo que $MIN(D) = \{r(a)\}$ sin embargo la semántica supuesta para esta base de datos viene dada por el modelo M_2, es decir puesto que no hay motivo evidente para suponer que $q(a)$ es cierto en D, se supone que es falso y se deriva $p(a)$.*

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, se dice que I es soportado si y sólo si para todo átomo A de I existe una instancia base de una sentencia de D de la forma $A \leftarrow L_1 \wedge \ldots \wedge L_n$ tal que $L_1 \wedge \ldots \wedge L_n$ es cierto en I. La cuestión consiste ahora en elegir entre todos los modelos minimales de D uno que sea soportado. A continuación se estudian distintas bases de datos para las cuales existe un modelo minimal soportado.

Semántica del punto fijo iterado [2] Esta semántica está definida para bases de datos estratificadas. La división en niveles $D_1, D_2, \ldots, D_k$ 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 minimal de D_1)
$D_2 \cup D_1$ M_2 (Modelo minimal y $M_1 \subseteq M_2$)
...
$D_k \cup D_{k-1} \cup \ldots \cup D_1$ M_K (Modelo minimal y $M_{k-1} \subseteq M_k$)

Sea $\Psi_D = M_k$. La semántica del punto fijo iterado se define por el conjunto de literales base ciertos en Ψ_D: $ESTRA(D) = \{L \mid L$ es un literal base y $\vDash_{\Psi_D} L\}$.

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

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

$-M_0 = \emptyset$
$-M_j = \Upsilon_{D_j} \uparrow w(M_{j-1})$ para $1 \leq j \leq k$

entonces se cumple que:

$-M_k$ es un modelo minimal y soportado de D.
$-M_k$ es independiente de la estratificación elegida para D.
$-M_k = \Psi_D$.

La relación entre esta semántica del punto fijo iterado y la de la compleción es la siguiente: $COMP(D) \subseteq ESTRA(D)$.

Semántica del modelo perfecto [48] La semántica del punto fijo iterado puede extenderse a bases de datos localmente estratificadas que son aquéllas que aunque no son estratificadas su instancia base (conjunto de todas las instancias base de las sentencias de la base de datos) sí que los es. 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.

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

Para toda base de datos localmente estratificada existe un modelo minimal soportado Π_D, llamado *modelo perfecto* de D, que coincide con el menor punto fijo del operador Υ_D para la instancia base de D. La semántica del modelo perfecto viene definida como sigue: $ESTRA\_L = \{L \mid L$ es un literal base y $\vDash_{\Pi_D} L\}$.

La relación entre la semántica del modelo perfecto y la semántica de la compleción es la siguiente: $COMP(D) \subseteq ESTRA\_L(D)$.

Semántica del modelo estable [27], [56] Existen bases de datos que no siendo localmente estratificadas tienen una semántica supuesta clara.

Ejemplo 82 *Sea D un estado de base de datos con una regla deductiva $p(x) \leftarrow q(x,y) \wedge \neg p(y)$ y un hecho $q(a,b)$. La instancia base de D contiene la sentencia $p(a) \leftarrow q(a,a) \wedge \neg p(a)$, 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 $\{q(a,b), p(a)\}$.*

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 átomo 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 para una lógica bivaluada (modelo estable total) y extendida posteriormente para una lógica trivaluada en los trabajos de Sacca y Zaniolo (modelo estable parcial). A continuación se presenta este último ya que el modelo estable total es un caso particular.

Sea B la instancia base de una base de datos D y F un conjunto de átomos base y sea $\Gamma(B,F)$ la base de datos obtenida a partir de B aplicando las siguiente 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 e F.

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

i) $T = Con(B,F)$

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

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

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

Ejemplo 83 *Sea D una base de datos con una sola regla deductiva $p \leftarrow p$. D tiene tres modelos parciales: $M_1 = <\emptyset, \{p\}>$, $M_2 = <\{p\}, \emptyset >$ y $M_3 = <\emptyset, \emptyset >$. M_1 y M_3 son admisibles y M_2 es soportado. El modelo que mejor captura 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: $ESTAB(D) = \{L \mid L$ es un literal base y $\models_M L$ para todo modelo estable $M\}$.

Ejemplo 84 *El modelo estable de D en el ejemplo 82 es $M = <\{q(a,b), p(a)\}, \{p(b), q(a,a), q(b,b), q(b,a)\}>$. Este modelo representa la semántica supuesta de D.*

Un modelo estable $< T, F >$ será total si $T \cup F$ cubre la Base de Herbrand del lenguaje de D. 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 Ψ_D.

Semántica del modelo bien fundado [61] 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 intenta seleccionar un modelo admisible de D tal que el conjunto de átomos F cumpla ciertas propiedades "razonables". Estas propiedades se basen 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. Se dice que X es infundado con respecto a I si para toda sentencia de B de la forma $A \leftarrow L_1 \wedge \ldots \wedge L_n$ con $A \in X$ se cumple al menos una de las siguientes condiciones:

i) $L_1 \wedge \ldots \wedge L_n$ es falso en I.

ii) $L_1 \wedge \ldots \wedge L_n$ 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 restante 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 85 *Sea D_1 una base de datos con una sola regla deductiva $p \leftarrow p$. El conjunto $\{p\}$ es infundado con respecto a <Ø, Ø>. La derivabilidad de p exige asumir p como cierto. Sea D_2 una base de datos con dos reglas deductivas $\{p \leftarrow q, q \leftarrow p\}$. El conjunto $\{p, q\}$ es infundado con respecto a <Ø, Ø>. La derivabilidad de p exige asumir q como cierto y viceversa.*

Dada una interpretación I los átomos que están en un conjunto X infundado con respecto a I no pueden ser considerados ciertos sin violar la condición i) de la definición de interpretación admisible, por ello puede extenderse 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 ($I = <$Ø, Ø$>$).

Dada una interpretación I de B se denomina:

$-U_D(I)$ a la unión de todos los conjuntos infundados respecto a I,

$-T_D(I)$ al conjunto de los átomos A tales que existe una sentencia $A \leftarrow L_1 \wedge \ldots \wedge L_n$ en B con $L_1 \wedge \ldots \wedge L_n$ cierto en I,

$-W_D(I)$ a la interpretación $< T_D(I), U_D(I) >$.

El operador W_D es monótono y su menor punto fijo es el modelo bien fundado $\aleph_D$ 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 $\aleph_D$: $B\_F = \{L \mid L$ es un literal base y $\models_{\aleph_D} L\}$.

Ejemplo 86 *Sea D una base de datos con dos reglas deductivas $\{p \leftarrow \neg q, q \leftarrow \neg p\}$. 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 versiones minimalista y maximalista de la propiedad de admisibilidad. El modelo bien fundado es admisible mientras que el modelo estable es admisible y minimal. La relación entre la semántica bien fundada y la semántica del modelo estable es la siguiente: $B\_F(D) \subseteq ESTAB(D)$.

Cuando el modelo bien fundado es total, éste coincide con el único modelo estable. Para bases de datos localmente estratificadas, las semánticas $ESTRA\_L$, $ESTAB$ y B_F coinciden

D.3 Semántica operacional de una base de datos

Desde un punto de vista práctico, una semántica declarativa debe tener asociada una semántica operacional que sea correcta y completa con respecto a ella, es decir que sea "válida" para la obtención de respuestas a un requerimiento. Dado un estado de base de datos D y un requerimiento R, una respuesta al requerimiento R es una substitución θ de las variables libres de R, tal que $Tr \models R\theta$ donde Tr es la teoría que representa el estado D en la semántica asumida. Las semánticas operacionales propuestas para bases de datos deductivas pueden clasificarse en dos aproximaciones: la aproximación *top-down* y la aproximación *bottom-up.*

- Aproximación *top-down*: Los procedimientos propuestos en esta aproximación se basan en *Resolución lineal* [32]. Estos procedimientos parten del requerimiento y obtienen respuestas al mismo aplicando las reglas deductivas hasta acceder a los hechos de la base de datos. Algunas propuestas en esta aproximación son:
  - El procedimiento $SLDNF$ para la semántica de la compleción [17], [34].El procedimiento *Query-Subquery* [62] es una extensión del anterior en el que se almacenan resultados intermedios para evitar la aparición de ramas infinitas.
  - El procedimiento SLS [49] para la semántica del modelo minimal (estándar, perfecto, estable o bien fundado) que es válido para ciertas bases de datos.

  - Un procedimiento para la semántica del modelo estable basado en abducción que generaliza el $SLDNF$ y que es más completo que éste. Es el procedimientos propuesto por Eshghi y Kowalski en [24]

- Aproximación *bottom-up*: Los procedimientos propuestos en esta aproximación consisten en implementaciones del operador consecuencia inmediata T_D definido por van Emden y Kowalski [60] para programas lógicos definidos. Estos procedimientos parten de los hechos de la base de datos y van aplicando las reglas deductivas hasta obtener las tuplas de las relaciones derivadas implicadas en el requerimiento.

- El método de Alexander [53] y el método de los conjuntos mágicos [12] son implementaciones eficientes del operador consecuencia inmediata para bases de datos definidas. Para bases de datos normales, existen numerosas extensiones de dicho operador que permiten obtener un modelo minimal; una de las más interesantes es la propuesta en [2], que ca1cula el modelo estándar para bases de datos estratificadas.

Bibliografía

[1] Abiteboul, S. *Updates, a new frontier*. International Conference on Database Theory, 1988, págs. 1-18.

[2] Apt, K. J; Blair, H. A; Walker, A. *Towards a theory of declarative knowledge*. En "Foundations of deductive databases and logic programming" (Minker, J. ed.), Morgan Kaufman, 1988.

[3] Aiken, A.; Hellerstein, J.; Widom, J. *Static analysis techniques for predicting the behavior of active database rules*. ACM Transactions on Database Systems, Vol. 20, N^{o}1, 1995, págs. 3-41.

[4] Asirelli, P.; Inverardi, P.; Mustaro, A. *Improving integrity constraint checking in deductive databases*. Proc. 2^{nd} International Conference on Database Theory (Gyssens M. et al. eds), Springer-Verlag, 1988, págs. 72-86.

[5] Baralis, E. *Rule analysis.* En "Active rules in database systems" (Patton ed.) Springer, 1998. págs. 51-65.

[6] Baralis, E.; Ceri, S.; Fraternali, P.; Paraboschi, S. *Support environment for active rule design*. Journal of Intelligent Information Systems, Vol. 7, 1996, págs. 129-149.

[7] Baralis, E.; Ceri, S.; Paraboschi, S. *Improved rule analysis by means of triggering and activation graphs*. Proc. 2^{nd} Workshop on Rules in Database Systems, 1995, Springer, págs. 165-181.

[8] Baralis, E.; Ceri, S.; Widom, J. *Better termination analysis for active databases*. Proc. 1^{st} Workshop on Rules in Database Systems, Springer-Verlag, 1993, págs. 163-179.

[9] Bry, F.; Decker, H. *Préserver l'intégrité d'une base de données déductive: une méthode et son implémentation*. 4^{emes} Journées de Bases de Donées Avancées, 1988.

[10] Bry, F.; Decker, H.; Manthey, R. *A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases*. Proc. 1^{st} International Conference Extending Data Base Technology (Schmidt, J. et al eds.), Springer Verlag LNCS 303, 1988, págs. 488-505.

[11] Bry, F.; Manthey, R.; Martens, B. *Integrity verification in knowledge bases.* Informe Interno D.2.1.a del ECRC (Munich), 1990.

[12] Bancilhon, F.; Maier, D.; Sagiv, Y.; Ullman, J. *Magic sets and other strange ways to implement logic programs.* Proc. 5^{th} ACM SIGMOD-SIGACT, Symposium Principles of Database Systems (PODS), 1986.

[13] Celma, M.; Casamayor, J.C; Mota, L.; Pastor, M.A; Marques, F. *Comprobación de la integridad en bases de datos deductiva: una aproximación basada en metaprogramación.* Proc. Sesto Convegno sulla Programmazione Logica, 1991.

[14] Celma, M.; Casamayor, J.C; Mota, L.; Pastor, M.A; Marques, F. *A derivation path recording method for integrity checking in deductive databases* Proc. 2^{nd} Workshop on the Deductive Approach to Information Systems and Databases, 1991.

[15] Celma, M. *Comprobación de la integridad en bases de datos deductivas: un método basado en el almacenamiento de los caminos de derivación.* Tesis doctoral Universidad Politécnica de Valencia, 1992.

[16] Ceri, S.; Fraternali, P.; Paraboschi, S.; Tanca, L. *Automatic generation of production rules for integrity maintenance.* ACM Transactions on Database Systems, Vol. 19, N^{o}3 , 1994, págs. 367-422.

[17] Clark, P. *Negation as failure.* En "Logic and databases" (H.Gallaire y J. Minker eds.), Plenum, 1978.

[18] Decker, H. *Integrity enforcement on deductive databases.* Proc. 1^{st} International Conference on Expert Database Systems (en Kerschberg, L. ed.), 1986.

[19] Decker, H. *Drawing updates from derivation.* Versión extendida, ECRC IR-KB-65, 1990.

[20] Decker, H. *Drawing updates from derivation.* Proc. 3^{th} International Conference on Database Theory, 1990, págs. 437-451.

[21] Díaz, Óscar *Deriving active rules for constraint maintenance in an object-oriented database.* Proc. International. Conference on Database and Expert Systems, 1992.

[22] Das, S.K.; Williams, M.H. *A path finding method for constraint checking in deductive databases.* Data & Knowledge Engineering, Vol. 4, 1989, págs. 233-244.

[23] Das, S.K.; Williams, M.H. *Integrity checking methods in deductive databases: a comparative evaluation.* Proc. 5^{th} British National Conference on Databases, 1989.

[24] Eshghi, K.; Kowalski, R.A. *Abduction compared with negation by failure.* Proc. 6^{th} International Conference on Logic Programming, MIT Press, 1989.

[25] Fraternali, P.; Paraboschi, S. *A review of repairing techniques for integrity maintenance.* En "Rules in Database Systems, Workshops in Computing", Springer, 1993, págs. 333-346.

[26] Gertz, M. *Specifying reactive integrity control for active databases.* IEEE RIDE Proc. 4th International Workshop on Research Issues in Database Engineering, 1994.

[27] Gelfond, M.; Lifschitz, V. The stable model semantics for logic programming, Proc. 5^{th} International Conference on Logic Programming, 1988.

[28] Guessoum, A.; Lloyd, J. W. *Updating knowledge bases.* New Generation Computing,Vol. 8, 1990, págs. 71-89.

[29] Guessoum, A.; Lloyd, J. W. *Updating knowledge bases* II. New Generation Computing,Vol. 10, 1991, págs. 73-100.

[30] Gallaire, H.; Minker, J.; Nicolas, J.M. *Logic and databases: a deductive approach.* Computing Surveys, Vol. 16, N°2, 1984, págs. 153-185.

[31] Gulutzan, P. Pelzer, T. *SQL-99 Complete Really, An Example-Based Reference Manual of the New Standard,* R&D Books, Miller Freman, 1999.

[32] Kowalski, R.; Kuehner, D. *Linear resolution with selection function.* Artificial Intelligence, Vol.2, 1971, págs. 227-260.

[33] Kakas, A. C.; Mancarella, P. *Database updates through abduction.* Proc. 16^{th} International Conference on Very Large Databases, 1990, págs. 650-661.

[34] Lloyd, J.W. *Foundations of logic programming* (2^{nd} edition). Springer Verlag, 1987.

[35] Larson, J. A.; Sheth, A. P. *Updating relational views using knowledge at view definition and view update time.* Information Systems, Vol. 16 n°2, 1991, págs. 145-168.

[36] Lloyd, J.W; Sonenberg, E.A.; Topor, R.W. *Integrity constraint checking in stratified databases.* Journal of Logic Programming, Vol. 4, 1987, págs. 331-343.

[37] Lloyd, J.W; Topor, R.W. *Making PROLOG more expressive.* Journal of Logic Programming, Vol. 3, 1984, págs. 225-240.

[38] Mota, L.; Celma, M. *Métodos para la comprobación de la integridad en bases de datos deductivas.* Qüestiió, Vol. 17, N°1, 1993, págs. 75-101.

[39] Mota, L.: Celma, M. *Automatic generation of trigger rules for integrity enforcement in relational databases with view definition* Proc. 3^{th} International Conference on Flexible Query Answering Systems, Springer, LNCS 1495, 1998, págs. 286-297.

[40] Mota, L.: Celma, M.; Decker, H *Transaction trees for planning knowledge revision* Proc. 4^{th} International Conference on Flexible Query Answering Systems, Physica-Springer, págs. 182-191, 2000.

[41] Minker, J. *On indefinite databases and the closed world assumption.* LNCS 138, Springer Verlag, 1982.

[42] Lockemann, P. C.; Moerkotte, G. *Reactive consistency control in deductive databases*. ACM TODS, Vol. 16, N°4, 1991, págs. 670-702.

[43] Nicolas, J.M. *Logic for improving integrity checking in relational databases*. Acta Informática, Vol. 18, 1982, págs. 227-253.

[44] Olivé, A. *Integrity checking in deductive databases*. Proc. 17^{th} International Conference on Very Large Data Bases, 1991.

[45] Pastor, J.A. *Automatic synthesis of update transaction programs in deductive databases*. Tesis doctoral Universidad Politécnica de Cataluña, 1996.

[46] Paton, N. W. *Active rules in databases systems*. Springer, 1998.

[47] Paton, N. W.; Díaz, O. *Introduction*. En "Active rules in database systems" (Patton ed.) Springer, 1998. págs. 3-26.

[48] Przymusinski, T. C. *On the declarative semantics of deductive databases and logic programs*. En "Foundations of deductive databases and logic programming", (J. Minker ed.), Morgan Kaufman, 1988.

[49] Przymusinski, T. C. *On the declarative and procedural semantics of logic programs*. Journal of automated Reasoning, Vol. 5, págs. 167-205, 1989.

[50] Qian, X. *The deductive synthesis of database transactions*. ACM Transactions on database Systems, Vol. 18, N°4, 1993, págs. 626-677.

[51] Reiter, R. *On closed world assumption* En "Logic and databases" (H.Gallaire y J. Minker eds.), Plenum, 1978.

[52] Reiter, R. *Towards a logical reconstruction of relational database theory*. En "On Conceptual Modelling" (Brodie, Mylopoulos y Schmit eds.), Springer-Verlag, 1984.

[53] Rohmer, J.; Lescoeur, R.; Kerisit, J. M. *The Alexander method. A technique for the processing of recursive axioms in deductive databases*. New Generation Computing, Vol 4, 1986, págs. 276-285.

[54] Robinson, J. A. *A machine oriented logic based on the resolution principle* Journal of ACN, N°12, 1965.

[55] Sadri, F.; Kowalski, R. *A theorem-proving approach to database integrity*. Proc. Workshop on Foundations of Deductive Databases and Logic Programming, 1987.

[56] Sacca, D.; Zaniolo, C. *Stable models and Non-determinism for logic programs with negation*. Proc. of the ACM SIGMOD-SIGACT Symposium on Principles of Databases Systems, 1990.

[57] Teniente, E. *El mètode dels esdeveniments per a l'actualització de vistes en bases de dades deductives*. Tesis doctoral Universidad Politécnica de Cataluña, 1992.

[58] Teniente, E. *Actualització de vistes en bases de dades relacionals: estat de la qüestió*. Novática, Mayo-Junio 93, págs. 41-52.

[59] Teniente, E.; Olivé, A. *Updating knowledge bases while maintaining consistency.* The VLDB Journal, Vol. 4, N^{o}2, 1995, págs. 193-241.

[60] van Emdem, M. H.; Kowalski, R. A. *The semantics of predicate logic as a programming language.* Journal of ACM, Vol. 23, 1976, págs. 733-742.

[61] van Gelder, A.; Ross, K. A.Shilipf, J. S. *Unfounded sets and well-founded semantics for general logic programs.* Proc. 7^{th} Symposium on Principles of Database Systems, 1988.

[62] Vielle, L. *Recursive query processing: the power of logic.* Theoretical Computer Science, Vol. 68, N^{o}2, 1989.

[63] Widom, J.; Ceri, S. *Active database systems.* Morgan-Kaufmann, 1996.

[64] Wallace, M. *Compiling integrity checking into update procedures.* Proc. 12^{th} International Joint Conference on Artificial Intelligence, 1991, págs. 903-910

[65] Wüthrich, B. *On updates and inconsistency repairing in knowledge bases.* Proc. 9^{th} IEEE International Conference on Data Engineering, 1993, págs. 608-615.