The Intensional Events Method for Consistent View Updating

Dolors Costal
Ernest Teniente
Toni Urpí

Report LSI–96–44–R

The Intensional Events Method for Consistent View Updating

Dolors Costal
Ernest Teniente
Toni Urpí

Universitat Politècnica de Catalunya
LSI, Facultat d'Informàtica
Pau Gargallo 5
E-08028 Barcelona -- Catalonia

fax: +34 - 3 - 401 70 14
e-mail: [dolors | teniente | urpi]@lsi.upc.es

Abstract

An important amount of research has been devoted to consistent view updating. In this paper we propose a method that follows a new approach to deal with this problem. Our approach is aimed at obtaining intensional translations that satisfy a view update request, instead of obtaining extensional translations. Intuitively, our translations are intensional in the sense that they characterise multiple possible values for a set of base fact updates such that the request is satisfied when the updates are applied using these values. Each characterised set of updates constitutes an extensional translation. The main advantages of following this approach are to improve the meaningfulness of the translations to the users; to increase the efficiency of obtaining them; and to facilitate the treatment of infinite domains.

We also show in the paper how our intensional method can be applied to database schema validation and we point out the advantages of using an intensional method in this case.

RELEVANT AREAS: Extensible and Active Databases/Knowledge Base Systems

July 1996

1. Introduction

Deductive databases generalise relational databases by including not only base facts and integrity constraints, but also view predicates defined by means of deductive rules. Using these rules, new facts (view facts) may be derived from facts explicitly stored. In this framework, an important problem is that of *consistent view updating*.

Consistent view updating is concerned with determining how a request to update a view can be appropriately translated into a set of updates of the underlying base facts that satisfy both the requested update and the integrity constraints of the database. Integrity constraints are handled in a consistency-maintaining way, i.e., when a constraint is violated, database consistency is restored by considering additional updates of base facts. In general, several translations that satisfy the request may exist.

The interest of consistent view updating has increased in the recent years since several proposals have shown how it can be applied to solve many different problems. For instance, [DTU96] reports how to tackle database schema validation by view updating, [DMMP91, DMB93, Cos93] use view updating as a core of their corresponding planning systems, [TO95] outlines how a view updating method can be used to repair an inconsistent database. In fact, [TU95] argues that several database updating problems (including the previous ones) may be solved in the same way since they belong to a more general class of problems called downward problems.

Much research has been devoted to consistent view updating during last years (see for example [KM90, Wüt93, TO95, CHM95, Dec96]). These methods differ in several aspects such as the kind of databases considered, the type of handled updates or the approach taken to deal with the problem. However, all of them are only concerned with obtaining *extensional* translations to satisfy a view update request. An extensional translation can be understood as an exhaustive enumeration of the individual base fact updates to be performed.

We propose in this paper a new approach aimed at obtaining *intensional* translations for a view update request. Intuitively, our translations are intensional in the sense that they characterise multiple possible values for a set of base fact updates such that the request is satisfied when the updates are applied using these values. Each characterised set of updates constitutes an extensional translation. The following simple example illustrates the difference between extensional and intensional translations.

Consider a deductive database consisting of a single base predicate Sal(e,s) which indicates that employee e has salary s, a single view predicate Hsal(e) defining employees with a salary greater or equal than 700, and an integrity constraint which prevents salaries of employees to be greater than 1000:

$$\text{Hsal}(e) \leftarrow \text{Sal}(e,s) \wedge s \geq 700$$
$$\text{Ic1} \leftarrow \text{Sal}(e,s) \wedge s > 1000$$

Given the consistent view update request of inserting Hsal(Joan), the extensional translations would be: {insert Sal(Joan,700)}, {insert Sal(Joan,701)}, {insert Sal(Joan,702)}, ..., {insert Sal(Joan,1000)}. That is, 301 different extensional translations exist. All these extensional translations can be characterised by the single intensional translation: {insert Sal(Joan,s) such that $s \geq 700 \wedge s \leq 1000$}.

Intensional translations provide several advantages over the extensional ones. First, the intensional translations obtained are more meaningful to the users since they provide them with additional insight into the nature of the extensional ones. At the same time, since each intensional translation characterises multiple extensional ones, the user is able to understand all of them at once.

Second, the search space needed to generate the intensional translations is considerably reduced since the extensional translations characterised by a corresponding single intensional translation are processed at once. So, the translations are obtained in a more efficient way.

Third, intensional translations facilitate the treatment of infinite domains since in this case an intensional translation may characterise infinite extensional ones and, obviously, an infinite set of translations cannot be generated in finite time. For instance, if we do not consider integrity constraint Ic1 in the previous example infinite extensional translations exist. All these translations are characterised by the intensional translation: {insert Sal(Joan,s) such that $s \geq 700$}.

In this paper, we propose the Intensional Events Method for obtaining intensional translations that satisfy a view update request. This method is an evolution of our previous extensional Events Method [TO95] that improves it according to the advantages outlined in the previous paragraphs.

As we have mentioned previously, as far as we know no method with these capabilities exists for dealing with consistent view updating in databases. However, as a related work, we must mention Kakas and Michael [KM95] who have taken a similar approach for integrating Abductive Logic Programming and Constraint Logic Programming and who have discussed the application of this framework to temporal reasoning and to scheduling. In addition to the contrasts due to the different field of concern, the main drawback of this proposal is that it is only applicable to restricted classes of logic programs.

The work reported here can be seen as parallel to the one of *intensional query answering* [CD86, Mot89, BJM93] which aims at obtaining intensional answers to a query that characterise a set of database values that satisfy that query. The main difference is that while these methods are concerned with the problems related to query processing, we deal with the different field of update processing.

This paper is organized as follows. Next section reviews basic concepts needed in the rest of the paper. Section 3 presents our method for obtaining intensional solutions that satisfy a view update request. Section 4 shows how this method can be applied to database schema validation and indicates the advantages of using an intensional method instead of an extensional one in this case. Finally, section 5 presents our conclusions and points out further work.

2. Base Concepts

In this section, we briefly review some definitions related to Deductive Databases [Llo87, Ull88]. We also survey the concept of the Augmented Database, as presented in [Oli91, UO92], which is needed to describe in section 3 the main features of the Intensional Events Method. Finally, we shortly describe the basic concepts of Constraint Logic Programming [JMSY92, JM94, Stu91], that we also use in section 3 for the procedural definition of our method, and we state the correspondence between them and deductive databases.

2.1 Deductive Databases

Throughout the paper, we consider a first order language with a universe of constants, a set of variables, a set of predicate names and no function symbols. We will use names beginning with a capital letter for predicate symbols and constants (with the exception that constants are also permitted to be numbers) and names beginning with a lower case letter for variables.

A *term* is a variable symbol or a constant symbol. If P is an m-ary predicate symbol and t_1, ..., t_m are terms, then $P(t_1, ..., t_m)$ is an *atom*. The atom is *ground* if every t_i (i = 1, ..., m) is a constant. A *literal* is defined as either an atom or a negated atom.

A *fact* is a formula of the form: $P(t_1, ..., t_m) \leftarrow$, where $P(t_1, ..., t_m)$ is a ground atom.

A *deductive rule* is a formula of the form:

$$P(t_1, ..., t_m) \leftarrow L_1 \wedge ... \wedge L_n \quad \text{with } m \geq 0, n \geq 1$$

where $P(t_1, ..., t_m)$ is an atom denoting the *conclusion* and $L_1, ..., L_n$ are literals representing *conditions*. $P(t_1, ..., t_m)$ is called the *head* and $L_1 \wedge ... \wedge L_n$ the *body* of the deductive rule. Variables in the conclusion or in the conditions are assumed to be universally quantified over the whole formula. The definition of a predicate P is the set of all rules in the deductive database that have P in their head.

An *integrity constraint* is a formula that every state of the deductive database is required to satisfy. We deal with constraints in *denial* form:

$$\leftarrow L_1 \wedge ... \wedge L_n \quad \text{with } n \geq 1$$

where the L_i are literals and all variables are assumed to be universally quantified over the whole formula. More general constraints can be transformed into this form by first applying the range form transformation [Dec89] and then using the procedure described in [LT84].

For the sake of uniformity, we associate to each integrity constraint an inconsistency predicate Icn, with or without terms, and thus they have the same form as the deductive rules. We call them *integrity rules*. Then, we would rewrite the former denial as $Ic1 \leftarrow L_1 \wedge ... \wedge L_n$.

We also define an standard auxiliary predicate Ic with the following rules: $Ic \leftarrow Ic1, ..., Ic \leftarrow Icn$. A fact Ic will indicate that there is an integrity constraint which is not satisfied.

A *deductive database* D is a triple D = (EDB, DR, IC) where EDB is a set of facts, DR a set of deductive rules, and IC a set of integrity constraints. The set EDB of facts is called the *extensional* part of the deductive database and the set DR of deductive rules is called the *intensional* part.

We also assume that deductive database predicates are partitioned into base and derived (view) predicates. A base predicate appears only in the extensional part and (eventually) in the body of deductive rules. A derived predicate appears only in the intensional part. Every deductive database can be defined in this form [BR86]. Furthermore, predicates in the body of deductive rules may be ordinary or evaluable ("built-in"). The former are base or derived predicates, while the latter are predicates such as the comparison or arithmetic predicates, that can be evaluated without accessing a database.

Example 2.1: The following example of deductive database will be used throughout the paper:

Dpt(Sales)
Wst(Sales)

$Lucky\_emp(e) \leftarrow Wks(e,d) \wedge Hsal(e) \wedge \neg Wst(d)$
$Hsal(e) \leftarrow Sal(e,s) \wedge s \geq 700$
$Ic1 \leftarrow Sal(e,s) \wedge s > 1000$
$Ic2 \leftarrow Wks(e,d) \wedge \neg Dpt(d)$
$Ic \leftarrow Ic1$
$Ic \leftarrow Ic2$

There are 4 base predicates. Wks(e,d) indicates that employee e works at department d, Dpt(d) means that d is a department, Wst(d) states that d is the worst department and Sal(e,s) indicates that employee e has salary s. There are 2 view predicates. Hsal(e) defines employees with a salary greater than 700, and Lucky_emp(e) defines employees who work in a department which is not the worst and who have a high salary. Finally, integrity constraint Ic1 prevents salaries of employees to be greater than 1000 and integrity constraint Ic2 forbids an employee to work in a department that does not exist.

2.2 The Augmented Database

The Intensional Events Method is based on a set of rules that define the difference between two consecutive database states. This set of rules together with the original database form the Augmented Database, denoted by A(D), which explicitly defines the insertions and deletions induced by a transaction which consists of a set of updates of the extensional database. We refer the reader to [Oli91, UO92] for a further description of this concept.

The Augmented Database is strongly based on the concept of *event*. For each predicate P in the underlying language of a given deductive database D, a distinguished *insertion event predicate* ιP and a distinguished *deletion event predicate* δP are used to define the precise difference of deducible facts of consecutive database states.

If P is a base predicate, ιP and δP facts represent insertions and deletions of base facts, respectively. We call them base event facts. Moreover, we assume that a transaction T consists of a set of base event facts. If P is a derived predicate, ιP and δP facts represent induced insertions and induced deletions, respectively. If P is an inconsistency predicate, ιP repesents a violation of the corresponding integrity constraint.

The definition of ιP and δP depends on the definition of P in D, but is independent of any transaction T and of the extensional part of D. For each derived or inconsistency predicate P, the Augmented Database contains the rules about ιP and δP, called *event rules*, which define exactly the insertions and deletions of facts about P that are induced by some transaction T:

$$\iota P(\mathbf{x}) \leftarrow P^n(\mathbf{x}) \wedge \neg P(\mathbf{x})$$
$$\delta P(\mathbf{x}) \leftarrow P(\mathbf{x}) \wedge \neg P^n(\mathbf{x})$$

where P refers to a predicate evaluated in the old state of the database, P^n refers to the predicate P evaluated in the new state of the database and $\mathbf{x}$ is a vector of variables.

The Augmented Database also contains a set of *transition rules* associated to each predicate P which define it in the new state of the database. These transition rules define the new state of predicate P (denoted by P^n) in terms of the old state of the database and the events that occur in the transition between the old and the new state of the database. We illustrate event and transition rules by means of an example.

Example 2.2: Consider the derived predicate Hsal from example 2.1 defined by the rule $Hsal(e) \leftarrow Sal(e,s) \wedge s \geq 700$. Event and transition rules associated to predicate Hsal are the following:

$$\iota Hsal(e) \leftarrow Hsal^n(e) \wedge \neg Hsal(e)$$
$$\delta Hsal(e) \leftarrow Hsal(e) \wedge \neg Hsal^n(e)$$
$$Hsal^n(e) \leftarrow Sal(e,s) \wedge \neg\delta Sal(e,s) \wedge s \geq 700$$
$$Hsal^n(e) \leftarrow \iota Sal(e,s) \wedge s \geq 700$$

Transition rules for $Hsal^n(e)$ define all possible ways of having facts about Hsal(e) in the new state. The first one corresponds to the case that a fact of Salary with S greater or equal than 700 was true in the old state and has not been deleted by the transaction, while the second one refers to the case where a fact of Salary with S greater or equal than 700 has been inserted by the transaction.

Given a deductive database D, the Augmented Database A(D) consists of D, its transition rules and its event rules. Description and discussion of the procedure for automatically deriving an Augmented Database from a database can be found in [Oli91, UO92]. These references also describe several syntactical simplifications of transition and event rules.

The following example shows part of the Augmented Database for the database of example 2.1, after simplification.

Example 2.3:

$$\iota Hsal(e) \leftarrow \iota Sal(e,s) \wedge s \geq 700$$
$$\iota Lucky\_emp(e) \leftarrow \iota Wks(e,d) \wedge \iota Hsal(e) \wedge \neg Wst(d) \wedge \neg\iota Wst(d)$$
$$\iota Lucky\_emp(e) \leftarrow \iota Wks(e,d) \wedge \iota Hsal(e) \wedge \delta Wst(d)$$
$$\iota Ic1 \leftarrow \iota Sal(e,s) \wedge s > 1000$$
$$\iota Ic2 \leftarrow \iota Wks(e,d) \wedge \neg Dpt(d) \wedge \neg\iota Dpt(d)$$
$$\iota Ic \leftarrow \iota Ic1$$
$$\iota Ic \leftarrow \iota Ic2$$

2.3 Constraint Logic Programming

In the following, we shortly describe the basic concepts of Constraint Logic Programming [JM94, Stu91] and we state the correspondence between them and deductive databases.

Roughly, Constraint Logic Programming can be said to involve the incorporation of constraints and constraint "solving" methods in a logic-based language.

A Constraint Logic Programming language CLP(S), exists in the context of a particular structure S which determines the meaning of the function and (constraint) relation symbols of some language L. *Constraints* in the structure are relations upon terms of the structure. For instance, $s \geq 700$, is a basic constraint in the domain of integers.

Constraint logic programs differ from logic programs by allowing constraints in bodies of rules and goals. A constraint logic program is, thus, a finite set of rules of the form:

$$P \leftarrow A_1 \wedge \ldots \wedge A_n \wedge c \quad \text{with } n \geq 0$$

where P and A_i are atoms and c is a constraint.

A goal takes the form:

$$\leftarrow A_1 \wedge ... \wedge A_n \mid c \text{ with } n \geq 1$$

where c is the collected constraint formula and A_i are atoms. We shall understand the goal $\leftarrow A_1 \wedge ... \wedge A_n \mid c$ to represent the formula $\leftarrow A_1 \wedge ... \wedge A_n \wedge c$.

Given a computation rule R, *CLP resolvents* from a goal G of the form:

$$\leftarrow A_1 \wedge ... \wedge A_i \wedge ... \wedge A_n \mid c$$

in a derivation tree for G are given by the following. If A_i is selected by the computation rule then there is a CLP resolvent

$$\leftarrow A_1 \wedge ... \wedge D_1 \wedge ... \wedge D_m \wedge ... \wedge A_n \mid c''$$

for each clause $D \leftarrow D_1 \wedge ... \wedge D_m \wedge c'$ such that $c'' = c \wedge \{A_i=D\} \wedge c'$ is satisfiable in S, where $\{A_i=D\}$ is a set of constraints equating the arguments of atoms in A_i and D.

A derivation of G is a path on the derivation tree of G. A derivation is *successful* if it is finite and its last goal contains no atoms i.e. it can be written as $\leftarrow [] \mid c$. A derivation is *finitely failed* if it is finite and not successful. The *answer constraint* of a successful derivation is the collected constraint c appearing in its last goal.

Constructive Negation for Constraint Logic Programming generalises this approach to handle negative literals in the body of the rules. It extends negation as failure to treat non-ground negative subgoals in a constructive manner. Roughly, it entails the following procedure: nodes of the subderivation for the non-ground negative subgoal are collected as a disjunction and negated giving a formula equivalent to the negative subgoal. See [Stu91] for further details on how to obtain Constructive Negation resolvents.

Deductive databases and their corresponding Augmented Databases, as defined in sections 2.1 and 2.2, fit in this framework. Deductive rules and integrity constraints can be seen as constraint logic program rules, where the constraint corresponds to the conjunction of evaluable literals in their body. Base facts are constraint logic program rules of the form: $P \leftarrow$ true, where P is a ground atom. For simplicity we will denote them by P.

3. The Intensional Events Method

In this section we present the Intensional Events Method for performing consistent view updating. This method extends our previous work in the field: the Events Method [TO95], which is a sound and complete method for updating deductive databases while maintaining their consistency. The method we present in this paper improves the Events Method by obtaining intensional answers to a view update request, while the Events Method (as well as the rest of the methods proposed up to date) is only able to obtain extensional solutions.

The main contributions of this paper are related to the advantges of obtaining intensional translations over obtaining extensional ones. That is, to increase the meaningfulness of the translations to the users, to reduce the search space needed to obtain these translations and to facilitate the treatment of infinite domains. An extensive argumentation of these advantages has already been provided in the introduction of this paper.

3.1 Declarative Definition of the Method

Consistent view updating is concerned with determining how a request to update a view can be appropriately translated into a set of updates of the underlying base facts that satisfy both the requested update and the integrity constraints of the database. Integrity constraints are handled in a consistency-maintaining way, i.e., when a constraint is violated, database consistency is restored by considering additional updates of base facts.

In general, there exist several possible ways of satisfying a view update request. Our approach consists of generating minimal translations for a given request. A translation is *minimal* when it does not exist a subset of it which is also a translation. In general, several minimal translations may exist.

The following definition formalises the kind of update requests that the Intensional Events Method is able to handle.

Definition 3.1: Update Request.

An *update request* is described by means of a derived predicate U, defined by a rule of the form $U \leftarrow u \mid c_u$, where u is a conjunction of event literals and c_u is a constraint formula.

For example, the update request of inserting the fact that Mary is a lucky employee would correspond to: $U \leftarrow \iota Lucky\_emp(e) \mid e=Mary$. The update request of inserting the fact that Mary is a lucky employee and deleting the fact that Sales is the worst department without inserting the fact that John works in the Staff department would correspond to: $U \leftarrow \iota Lucky\_emp(e) \wedge \delta Wst(d) \wedge \neg \iota Wks(e',d') \mid e=Mary \wedge d=Sales \wedge e'=John \wedge d'=Staff$. Finally, the update request of inserting a salary greater than 2000 to somebody would correspond to: $U \leftarrow \iota Sal(e,s) \mid s>2000$.

Intensional answers characterise possible values of a set of base facts to insert and/or to delete in order to satisfy the update request. Each of the characterised possible values of the insertions and/or deletions constitutes an extensional answer. In the following, we first define the concept of extensional translation and after that we generalise it to define an intensional translation.

Definition 3.2: Extensional Translation.

Let A(D) be an Augmented Database and let U be an update request. Then, an *extensional translation* of U is a set T of ground base events such that:

(1) $A(D) \cup T \models U$

(2) $A(D) \cup T \not\models \iota Ic$

The first condition states that the update request is a logical consequence of the database updated according to T, while the second condition states that the updated database will remain consistent since no insertion of Ic will be induced. Note that, as usual, we assume that the database is consistent before the update.

Definition 3.3: Intensional Translation.

Let A(D) be an Augmented Database and let U be an update request. Then, an *intensional translation* of U has the form $T \mid c_T$, where T is a set of non-ground base event literals and c_T is a constraint formula such that:

Its corresponding Augmented Database is the following:

$$\iota Hsal(e) \leftarrow \iota Sal(e,s) \wedge s \geq 700$$
$$\iota Ic \leftarrow \iota Sal(e,s) \wedge s > 1000$$

In Figure 3.1 we show how we obtain the intensional translations that satisfy the update request U ← ιHsal(e) | e=Joan. Circled labels appearing in the left of a derivation are references to the rules of the method defined in Section 3.3, while underlined literals in each goal correspond to the selected literals.

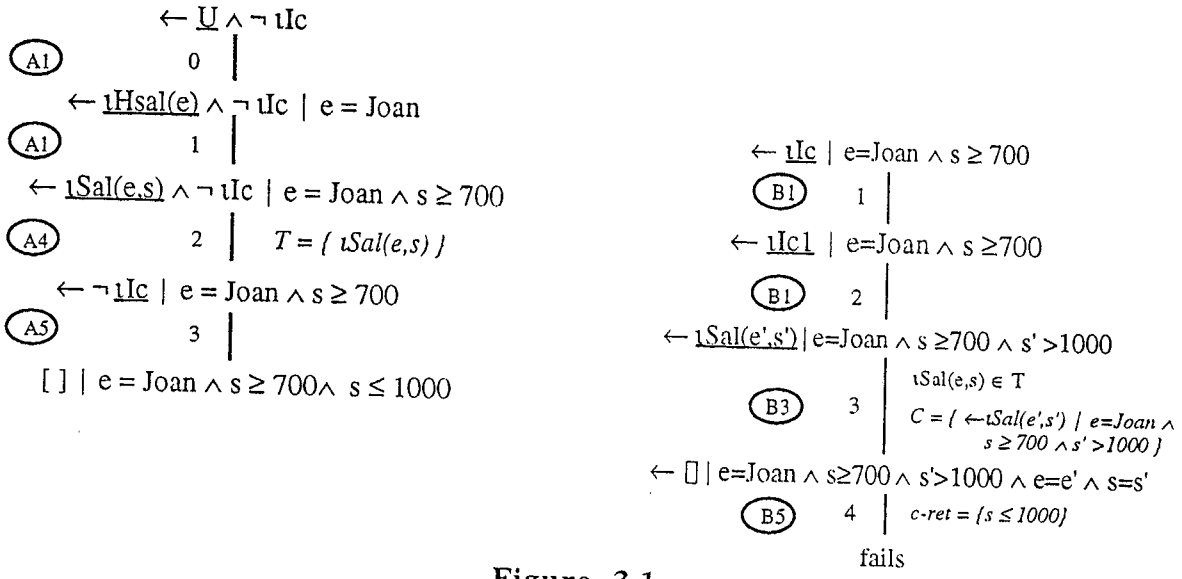

Figure 3.1

Given our update request U, the root goal is ← U ∧ ¬ιIc. Thus, we start from this goal in the tree at the upper left of the previous figure. Steps 0 and 1 of this tree are CLP resolution steps. At step 2, the selected literal is ιSal(e,s), which is a positive base event. To get a successful derivation, we must include it in the input set and use it as input clause. Therefore, it is included in the intensional translation set T. At step 3 the selected literal is ¬ ιIc. To get a successful derivation, this step requires finitely failure of the subsidiary tree rooted at ← ιIc | e = Joan ∧ s ≥ 700. This finitely failed tree ensures maintenance of integrity constraints Ic1 and it is shown in the bottom right of the previous figure.

Steps 1 and 2 of the subsidiary tree are CLP resolution steps. Step 3 is a CLP resolution step where T acts as input set. Moreover, to guarantee failure for this branch, ιSal(e',s') must fail for all possible values of e' and s' that satisfy the constraint formula. We use an auxiliary set C, which we call *condition set*, in order to check that later on, during the derivation process, the goal "←ιSal(e',s') | e = Joan ∧ s ≥ 700" does not cease to fail due to some inclusion in T. Thus, in step 3, the current goal is included in C.

In general, a condition set C is a set of goals that must fail during the derivation process. Since inclusions to T are the only way to make failed goals in C succeed, before including a base event in T we must ensure that all goals in C continue failing.

At step 4, failure of the branch is achieved by eliminating local variables from the constraint formula and negating the resulting expression. Usually this will result in a disjunction of several constraints. Each of these possibilites guarantees failure for the branch. We use an auxiliary constraint formula c-ret to keep trace of the chosen possibility. In the example.

$s \leq 1000$ is included in c-ret which guarantees that the obtained translation will not induce a violation of integrity constraint Ic1.

Once the tree rooted at $\leftarrow \iota Ic \mid e = Joan \wedge s \geq 700$ fails finitely, c-ret constraints are added to the constraints of step 3 of the initial derivation of Figure 3.1. Then, we get the empty clause in this initial derivation since the resulting constraint formula is solvable. Therefore, the process finishes and we obtain the intensional translation:

$$\{ \iota Sal(e,s) \} \quad | \quad e=Joan \wedge s \geq 700 \wedge s \leq 1000$$

which both satisfies the update request and maintains the integrity constraint. Note that this intensional translation coincides with the translation pointed out in the introduction.

Example 3.2: Consider now the database example defined in Section 2. We show in Figures 3.2 and 3.3 how the intensional translations that satisfy the update request $U \leftarrow \iota Lucky\_emp(e) \mid e=Mary$ are obtained by our procedure.

$\leftarrow U \wedge \neg \iota Ic$

(A1) 0 |

$\leftarrow \iota Lucky\_emp(e) \wedge \neg \iota Ic \mid e = Mary$

(A1) 1 |

$\leftarrow \iota Wks(e,d) \wedge \iota Hsal(e) \wedge \neg Wst(d) \wedge \neg \iota Wst(d) \wedge \neg \iota Ic \mid e = Mary$

(A1) 2 |

$\leftarrow \iota Wks(e,d) \wedge \iota Sal(e,s) \wedge \neg Wst(d) \wedge \neg \iota Wst(d) \wedge \neg \iota Ic \mid e = Mary \wedge s \geq 700$

(A4) 3 | $T = \{ \iota Wks(e,d) \}$

$\leftarrow \iota Sal(e,s) \wedge \neg Wst(d) \wedge \neg \iota Wst(d) \wedge \neg \iota Ic \mid e = Mary \wedge s \geq 700$

(A4) 4 | $T = \{ \iota Wks(e,d), \iota Sal(e,s) \}$

$\leftarrow \neg Wst(d) \wedge \neg \iota Wst(d) \wedge \neg \iota Ic \mid e = Mary \wedge s \geq 700$

(A2) 5 |

$\leftarrow \neg \iota Wst(d) \wedge \neg \iota Ic \mid e = Mary \wedge s \geq 700 \wedge d \neq Sales$

(A5) 6 | $C = \{ \leftarrow \iota Wst(d) \mid e = Mary \wedge s \geq 700 \wedge d \neq Sales \}$

$\leftarrow \neg \iota Ic \mid e = Mary \wedge s \geq 700 \wedge d \neq Sales$

(A5) 7 | $T = \{ \iota Wks(e,d), \iota Sal(e,s), \iota Dpt(d'') \}$

$[\,] \mid e = Mary \wedge s \geq 700 \wedge d \neq Sales \wedge s \leq 1000 \wedge d = d''$

Figure 3.2

Steps 0, 1 and 2 are CLP resolution steps. At step 3, the selected literal is $\iota Wks(e,d)$, which is a positive base event. To get a successful derivation, we must include it in the input set and use it as input clause. Therefore, it is added to the intensional translation set T. Step 4 is similar to step 3 and results in the inclusion of the base event $\iota Sal(e,s)$ to T. Step 5 is a Constructive Negation step which results in the addition of the conjunctand $d \neq Sales$ to the constraint formula.

At step 6, the selected literal is $\neg \iota Wst(d)$. To get success for this branch, $\iota Wst(d)$ must fail for all possible values of d that satisfy the constraint formula. Thus, in this step the goal $\leftarrow \iota Wst(d) \mid e = Mary \wedge s \geq 700 \wedge d \neq Sales$ is included in C to guarantee that subsequent inclusions in T will not make it succeed.

At step 7 the selected literal is ¬ ιIc. This step requires that the subsidiary tree rooted at ←ιIc | e = Mary ∧ s ≥ 700 ∧ d≠Sales fails finitely. This finitely failed tree ensures maintenance of integrity constraints Ic1 and Ic2. Part of this subsidiary tree is shown in the following figure.

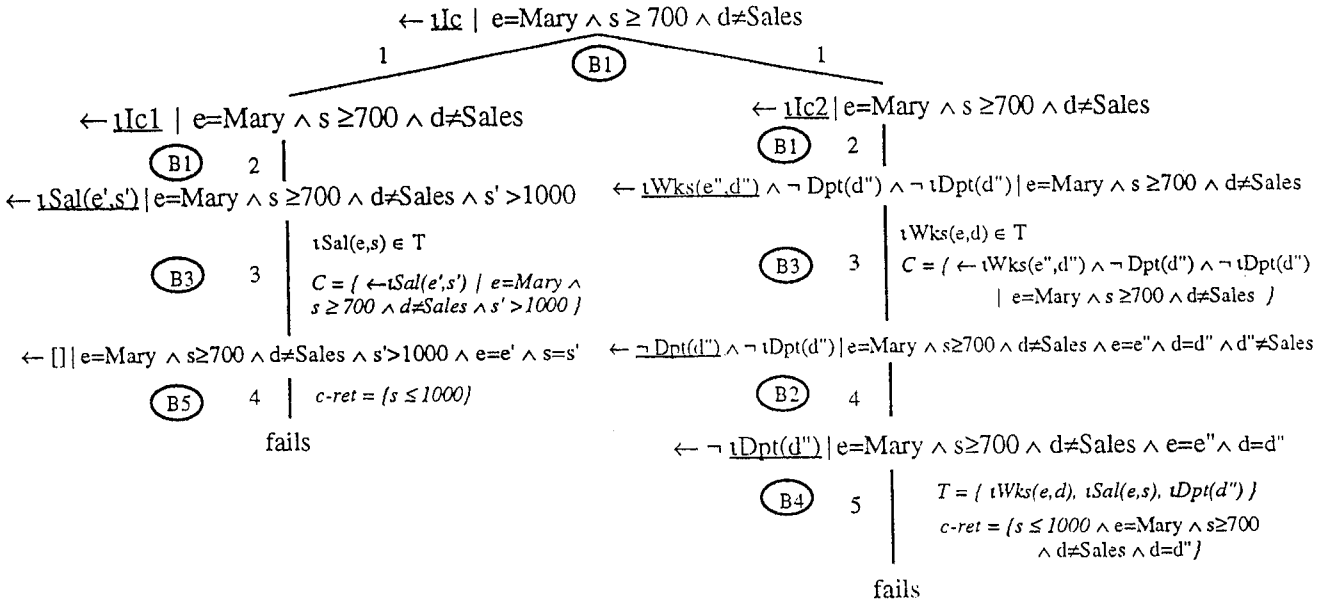

Figure 3.3

Steps 1 and 2 in the left branch are CLP resolution steps. Step 3 is a CLP resolution step where T acts as input set. Moreover, the current goal is included in the condition set C to guarantee that subsequent inclusions in T will not make it succeed. At step 4, failure of this branch is achieved by eliminating local variables from the constraint formula and negating the resulting expression. In the example, s≤1000 is included in c-ret which guarantees that the obtained translation will not induce a violation of integrity constraint Ic1.

Steps 1, 2 and 3 in the right branch are similar to the same steps of the left branch. Step 4 is a Constructive Negation step. Finally, at step 5 the selected literal is ¬ ιDpt(d"). This right branch will fail if it is possible to make the subsidiary tree rooted at: ← ιDpt(d") | e=Mary ∧ s>700 ∧ d≠Sales ∧ s≤1000 succeed. This latter derivation (not shown here) results in the addition of ιDpt(d") to T and of additional constraints to c-ret. Note that this insertion is needed to guarantee that Ic2 is not violated by the intensional translation T.

Once the derivation of ← ιIc | e = Mary ∧ s ≥ 700 ∧ d≠Sales fails finitely, c-ret constraints are added to the constraints of step 7 of the initial derivation of Figure 3.2. Then, we get the empty clause in this initial derivation since the resulting constraint formula is solvable. Therefore, the process finishes and we obtain the intensional translation:

$$\{\iota Wks(e,d), \iota Sal(e,s), \iota Dpt(d")\} \quad | \quad e=Mary \wedge s\geq 700 \wedge d\neq Sales \wedge s\leq 1000 \wedge d=d"$$

which both satisfies the update request and maintains the integrity constraints.

Selecting at step 1 of Figure 3.2 the clause ιLucky_emp(e) ← ιWks(e,d) ∧ ιHsal(e) ∧ δWst(d) instead of the selected clause, our method would obtain the intensional translation {ιWks(e,d), ιSal(e,s), δWst(d)} | e=Mary ∧ d=Sales ∧ s≥700 ∧ s≤1000, which satisfies also the update request U ← ιLucky_emp(e) | e=Mary.

3.3 Formalisation of the Procedural Definition

As has been pointed out in the previous section, the Intensional Events Method is an interleaving of two activities: 1) satisfying an update request by including base events in the intensional translation set, and 2) ensuring that changes induced by these base events maintain the integrity constraints and are not contradictory with the requested update. These two activities are performed during *constructive* and *consistency* derivations, respectively, as defined below.

Let U be an update request. In our method, $T \mid c_T$ will be an intensional translation of U if there exists a constructive derivation from $(\leftarrow U \wedge \neg \iota Ic \ \varnothing \ \varnothing)$ to $([] \mid c_T \ T \ C)$. Constraint formula c_T characterises possible values of the updates given by T that satisfy the update request without violating any integrity constraint. In order to obtain all possible translations that satisfy an update request U, we have to consider all possible constructive derivations.

Predicates appearing in A(D) may be (we include some examples from the previous section):

- old base predicates (from the old state of the database): Wks, Dpt, Wst
- old derived predicates (from the old state of the database): Hsal, Lucky_emp
- base events: δWks, ιWks, δDpt, ιWst
- derived events: δHsal, ιLucky_emp

The rules applied in constructive and consistency derivations depend on the type of the selection. We summarize in figure 3.3 which rule is applied in each case:

Constructive	Positive	Negative
Old base pred. Old derived pred.	A1	A2
Base event	A3, A4	A5
Derived event	A1	A5

Consistency	Positive	Negative
Old base pred. Old derived pred.	B1	B2
Base event	B3	B4
Derived event	B1	B4
Const. formula	B5	

Figure. 3.3

Constructive Derivation

A *constructive derivation* from $(G_1 \ T_1 \ C_1)$ to $(G_n \ T_n \ C_n)$ via a computation rule R is a sequence:

$$(G_1 \ T_1 \ C_1), (G_2 \ T_2 \ C_2), \ldots, (G_n \ T_n \ C_n)$$

such that for each $i \geq 1$, G_i has the form $\leftarrow L_1 \wedge \ldots \wedge L_k \mid c$, where c is a constraint formula, $R(G_i) = L_j$ and $(G_{i+1} \ T_{i+1} \ C_{i+1})$ is obtained according to one of the following rules:

A1) If L_j is positive, it is not a base event, and S is a CLP resolvent of some clause in A(D) with G_i on the selected literal L_j, then $G_{i+1} = S$, $T_{i+1} = T_i$ and $C_{i+1} = C_i$.

A2) If L_j is a base or derived predicate, negative and old, and S is a Constructive Negation[1] resolvent of some clause in A(D) with G_i on the selected literal L_j, then G_{i+1}= S, T_{i+1}= T_i and C_{i+1}=C_i.

A3) If L_j is a positive base event, and S is a CLP resolvent of some clause in T_i with G_i on the selected literal L_j, then G_{i+1}= S, T_{i+1}= T_i and C_{i+1}=C_i.

A4) If L_j is a positive base event ιP (resp. δP), [] | c' is a Constructive Negation resolvent of some clause in T_i with the goal $\leftarrow \neg L_j \mid c$, [] | c" is a Constructive Negation resolvent (resp. a CLP resolvent) of some clause in A(D) with the goal $\leftarrow \neg P \mid c'$ (resp. $\leftarrow P \mid c'$), C'_i is the condition set restricted[2] by c", there exists a consistency derivation from (C'_i $T_i \cup \{L_j\}$ C'_i ∅) to ({} T" C" c-ret), and c"∧ c-ret is solvable, then $G_{i+1} = \leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k \mid c" \wedge$ c-ret, T_{i+1}=T" and C_{i+1}= C".
Note that if $C_i = \varnothing$, then $G_{i+1} = \leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k \mid c"$, T_{i+1} = $T_i \cup \{L_j\}$ and C_{i+1}=C_i.

A5) If L_j is a negative, new or event predicate and there exists a consistency derivation from ($\{\leftarrow \neg L_j \mid c\}$ T_i C_i ∅) to ({} T' C' c'-ret), and c ∧ c'-ret is solvable, then G_{i+1}= $\leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k \mid c \wedge$ c'-ret, T_{i+1}=T' and C_{i+1}= C'.

Rules A1) and A3) are CLP-resolution steps where A(D) or T act as input set, respectively. Rule A2) is a Constructive Negation step where A(D) acts as input set.

In rule A4) the selected base event is included in T_i, after restricting the possible values for which the event is applicable according to the the events of the same type already contained in T and to the event definition, and provided that we can ensure that this will not violate the consistency of any condition in C_i. Note that if $C_i = \varnothing$, consistency derivations must not be performed because there is no condition to satisfy.

In rule A5), we get the next goal if we can ensure consistency for the selected literal.

Consistency Derivation

A *consistency derivation* from (F_1 T_1 C_1 c_1-ret) to (F_n T_n C_n c_n-ret) via a computation rule R is a sequence:

$$(F_1\ T_1\ C_1\ c_1\text{-ret}), (F_2\ T_2\ C_2\ c_2\text{-ret}),\ldots, (F_n\ T_n\ C_n\ c_n\text{-ret})$$

such that for each $i \geq 1$, F_i has the form $H_i \cup F'_i$, where $H_i = \leftarrow L_1 \wedge \ldots \wedge L_k \mid c_i$, and, *for some* j=1...k or c_i, (F_{i+1} T_{i+1} C_{i+1} c_{i+1}-ret) is obtained according to one of the following rules:

B1) If L_j is positive, it is not a base event, S' is the set of all CLP resolvents of clauses in A(D) with H_i on the literal L_j, then F_{i+1}= S'∪F'_i, T_{i+1}= T_i, C_{i+1}= C_i, and c_{i+1}-ret=c_i-ret.

---

[1] Following [Stu91], we assume that the complete constraint formula c_i is passed to the negative subderivation and that the frontier considered by Constructive Negation is the empty clause.

[2] By a condition set C restricted by a constraint formula c we mean that c is added to the constraint part of all conditions in C.

B2) If L_j is a base or derived predicate, negative and old, S' is the set of all Constructive Negation resolvents of clauses in A(D) with H_i on the literal L_j, then $F_{i+1}= S' \cup F'_i$, $T_{i+1}= T_i$, $C_{i+1}= C_i$ and c_{i+1}-ret=c_i-ret.

B3) If L_j is a positive base event, S' is the set of all CLP resolvents of clauses in T_i with H_i on the literal L_j, then $F_{i+1} = S' \cup F'_i$, $T_{i+1} =T_i$, $C_{i+1} =C_i \cup \{H_i\}$ and c_{i+1}-ret=c_i-ret.

B4) If L_j is negative, new or event predicate, and there exists a constructive derivation from $(\{\leftarrow \neg L_j \mid c_i\}\ T_i\ C_i)$ to $([] \mid c'\ T'\ C')$ and c_g is a constraint formula obtained by eliminating local variables[3] from c' and simplifying the resulting expression, then $F_{i+1} = F'_i$, $T_{i+1} =T'$, $C_{i+1} =C'$ and c_{i+1}-ret = c_i-ret $\wedge$ c_g.

B5) If c_i is selected, c_g is a constraint formula obtained by eliminating local variables from c_i and simplifying the resulting expression, $\neg c_g \leftrightarrow c_{g1} \vee \ldots \vee c_{gk} \vee \ldots \vee c_{gm}$, then $F_{i+1} = F'_i$, $T_{i+1} =T_i$, $C_{i+1} =C_i$ and c_{i+1}-ret = c_i-ret $\wedge$ c_{gk}.

Rule B1) is a CLP-resolution step where A(D) acts as input set. Rule B2) is a Constructive Negation step where A(D) acts as input set.

In rule B3) the selected literal can be unified with one or more base events in T_i. We must then verify that each resulting branch fails and include the current goal in the condition set C to guarantee that subsequent inclusions in T will not make it succeed.

In rule B4) the current branch is dropped if there exists a constructive derivation for the negation of the selected literal. If this is the case, we must keep trace of the constraints that guarantee failure of the current branch (that is, those constraints for which the constructive derivation exists). These constraints are obtained by eliminating local variables from the constraint formula returned by the constructive derivation and simplifying the resulting expression and are stored in an auxiliary constraint formula c-ret.

In rule B5) failure of the current branch is achieved by selecting one of the disjunctands resulting of eliminating local variables from the constraint formula and simplifying the resulting expression. The chosen disjunctand must be added to c-ret to guarantee that it will be returned to the original constructive derivation.

Consistency derivations do not rely on the particular order in which selection rule R selects literals or the constraint formula since, in general, all possible ways in which a conjunction "$\leftarrow L_1 \wedge \ldots \wedge L_k \mid c$" can fail should be explored. Each one may lead to a different translation.

4. Applications of the Method

As we pointed out in [TU95] and in [DTU96], we believe that the improvements in the consistent view updating area can also be adopted to be applied in many different problems that can be formulated in terms of view updating like database schema validation, repairing of inconsistent databases or as a core of a planning system. In this sense, we show in this section how the Intensional Events Method can be applied to database schema validation and we point out the advantages of using an intensional method instead of an extensional one in this case.

---

[3] Local variables in a constraint formula are those variables that do not appear in the original root derivation nor in any element of the translation set T.

4.1 Database Schema Validation

In [DTU96] we showed that any method for view updating can be used also for validating database schema specifications. In that work, we defined a set of desirable properties that a database schema should satisfy (satisfiability of the schema, liveliness of a predicate, non-redundancy of integrity constraints and reachability of partially specified states) and we showed how these properties could be validated by using view updating.

Roughly, the main idea of that paper is to define, for each property, a distinguished view predicate which describes declaratively the accomplishment of the property. An attempt to check a property can then be made by attempting to satisfy the request of inserting the predicate corresponding to that property.

Consider, for instance, the definition of the liveliness property [DTU96]: a predicate P is lively in a certain schema if there exists a state of that schema that satisfies all the integrity constraints and in which at least one fact about p is true. Assume that the view predicate Lively_P expresses declaratively this definition *(see the example below for the precise definition)*. Then, a successful attempt to insert Lively_P shows that in the database schema on which the insert request has been executed, P is lively; a finitely failed attempt to insert Lively_P shows that the extension of predicate P will be empty in each sound state.

In [DTU96] we showed that this idea is independent of and applicable to any particular method for view updating, and we illustrated it by using the Events Method [TO95]. Although effectively any method could in fact be used for validating a database schema, we claim that the search space is significantly reduced if an intensional method is considered.

The intuition behind this claim is that when we have to prove that a certain property is not satisfied, we need to take into account all possible values of the domain of the variables. If we use an extensional method we will have a derivation for each possible solution, whereas if we use an intensional method we will have a derivation for each possible characterisation of the solutions. We will show it by means of a simple example of the liveliness property.

Example 4.1: Consider the following database schema with only one derived predicate and one integrity constraint:

$$P(x) \leftarrow Q(x) \wedge R(x)$$
$$Ic \leftarrow R(x)$$

Following [DTU96], the declarative definition of liveliness of predicate P is expressed as:

$$Lively\_P \leftarrow P(x) \wedge \neg Ic$$

and the corresponding request to check this property is $\iota Lively\_P$.

In this example, predicate P is clearly not lively since the integrity constraint prevents any fact about P to be true. That is, the existence of a fact of P would require the existence of a fact of R, which is not possible due to the integrity constraint.

Relevant rules of the Augmented Database corresponding to this database schema are the following. Note that these rules are not as simplified as rules used in section 3 since when validating a database schema it is not possible to assume that the database is consistent before the update.

$$\iota P(x) \leftarrow Q(x) \wedge \neg\delta Q(x) \wedge \iota R(x)$$
$$\iota P(x) \leftarrow \iota Q(x) \wedge R(x) \wedge \neg\delta R(x)$$

$\iota P(x) \leftarrow \iota Q(x) \wedge \iota R(x)$
$\iota Ic \leftarrow \iota R(x) \wedge \neg Ic$
$\delta Ic \leftarrow \delta R(x) \wedge \neg Ic_1' \wedge \neg Ic_2'$
$Ic_1' \leftarrow \iota R(x)$
$Ic_2' \leftarrow R(x) \wedge \neg \delta R(x)$
$\iota Lively\_P \leftarrow P(x) \wedge \neg \delta P(x) \wedge \delta Ic$
$\iota Lively\_P \leftarrow \iota P(x) \wedge \neg Ic \wedge \neg \iota Ic$
$\iota Lively\_P \leftarrow \iota P(x) \wedge \delta Ic$

Figures 4.2 and 4.3 below show the search space of our procedure to determine that no solution that satisfies the update request $U \leftarrow \iota Lively\_P$ exists[4]. That is, the search space needed to determine that predicate P is not lively.

More concretely, figure 4.2 shows that it does not exist any constructive derivation for the update request U since all branches fail. In particular, notice that the central branch fails since the consistency derivation associated to the literal $\neg\iota Ic$, showed in figure 4.3, returns the constraint false, which is obviously not solvable.

If we had used an extensional method we would have had a similar search space for each value of the domain of P. That is, we would have obtained the proposed translations {ιQ(a), ιR(a)}, {ιQ(b), ιR(b)}, {ιQ(c), ιR(c)}, ..., and each one of them would had failed since no consistency derivation would exist for the literal $\neg\iota Ic$.

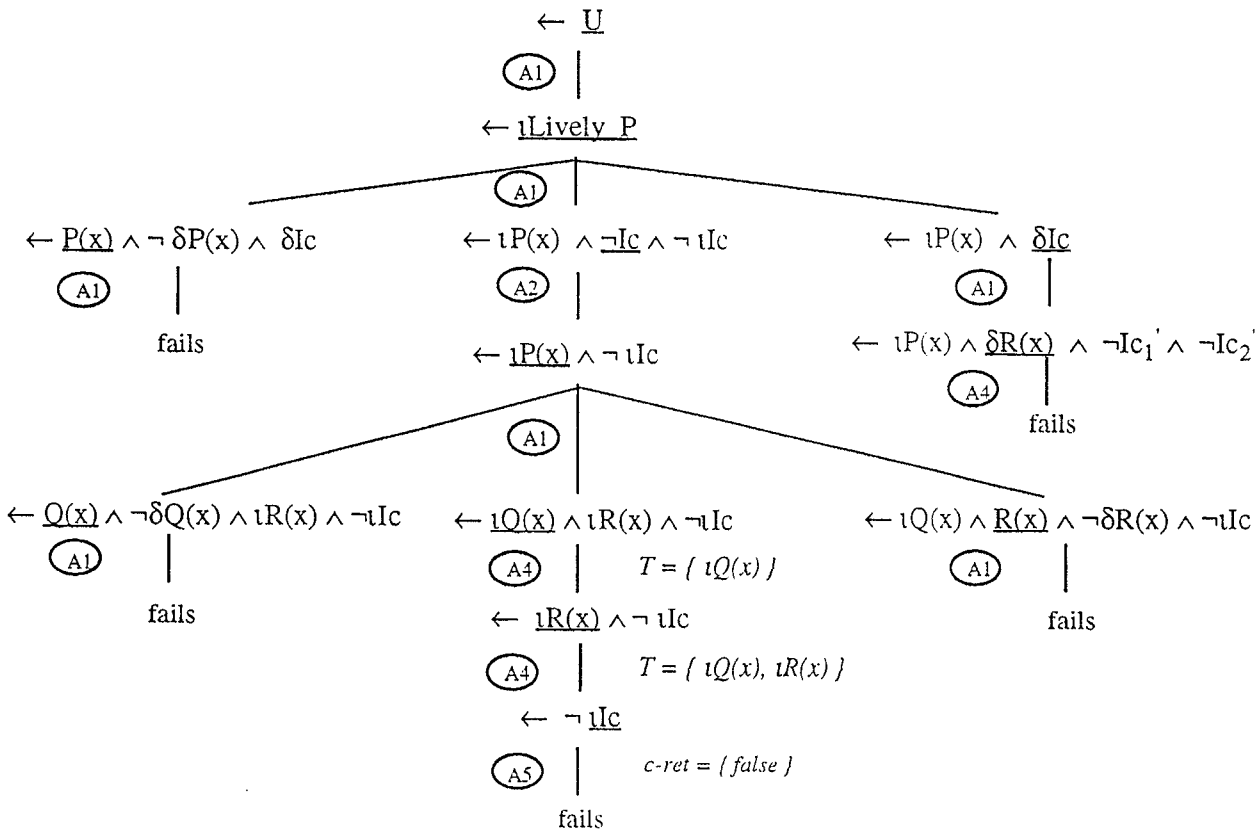

Figure 4.1

[4] Note that in this example the top clause is just ← U instead of ← U ∧ ¬ιIc. We can omit ¬ιIc since following [DTU96] the maintenance of integrity constraints is ensured due to definition of Lively_P.

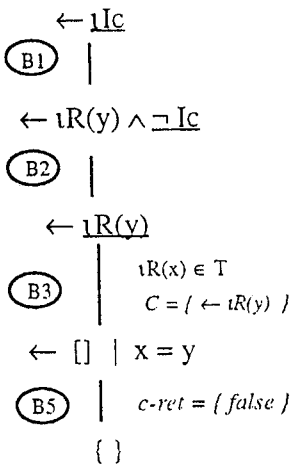

Figure 4.2

5. Conclusions and Further Work

In this paper we have presented a new method, the Intensional Events Method, for dealing with the problem of consistent view updating in deductive databases. The main contribution of this method is that of obtaining *intensional translations* that satisfy an update request instead of obtaining *extensional translations*, which has been the traditional approach followed by the up to date research in the field. An intensional translation characterises possible values of base fact updates that satisfy an update request; while each characterised set of updates constitutes an extensional translation.

We have shown that intensional translations provide several advantages over the extensional ones. First, intensional translations are more meaningful to the users since they provide them with additional insight into the nature of the extensional ones. Second, intensional translations are obtained in a more efficient way since the search space needed to generate them is considerably reduced. Finally, intensional translations facilitate the treatment of infinite domains since an intensional translation may characterise infinite extensional ones.

As we pointed out in [TU95] and in [DTU96], we believe that the improvements in the consistent view updating area can also be adopted to be applied in many different problems that can be formulated in terms of view updating like database schema validation, repairing of inconsistent databases or as a core of a planning system. In this sense, we have shown in this paper how the Intensional Events Method can be applied to database schema validation and we have pointed out the advantages of using an intensional method instead of an extensional one in this case.

We believe that the Intensional Events Method represents a first important step towards providing methods that obtain intensional answers. Nevertheless, we believe that more research is needed in this direction. In particular, an extension of the method able to handle rules such as $P(x) \leftarrow x>1 \wedge x\leq 3$ or $P(x) \leftarrow x=2 \vee x=3$ would allow to obtain in some cases translations with a higher intensionality degree.

Acknowledgements

We would like to thank E.Mayol, A.Olivé, J.A.Pastor and M.R.Sancho for many useful comments and discussions. This work has been partially supported by PRONTIC CICYT program projects TIC94-0512 and TIC95-0735.

References

[BJM93] Brodsky, A.; Jaffar, J,; Maher, M. "Towards Practical Constraint Databases", *Proc. of the 19th VLDB Conference*, Dublin, Ireland, 1993, pp. 567-580.

[BR86] Bancilhon, F.; Ramakrishnan, R, "An Amateur's Introduction to Recursive Query Processing", *Proc. of the ACM SIGMOD Int. Conf. on the Management of Data,*, Washington D.C., 1986.

[CD86] Cholvy, L.; Demolombe, R. "Querying a Rule Base", *Proc. of the 1st Int. Conf. on Expert Database Systems*, 1986, pp. 365-371.

[CHM95] Chen, I.A.; Hull, R.; McLeod, D. "An Execution Model for Limited Ambiguity Rules and Its Application to Derived Data Update", *ACM Transactions on Database Systems,*, Vol. 20, N° 4, December 1995, pp. 365-413.

[Cos93] Costal, D. "A New Plan Generation Method for Deductive Conceptual Model Validation", *Forth Int. Workshop on the Deductive Approach to Information Systems and Databases*, 1993, pp. 175-200.

[Dec89] Decker, H. "The Range Form of databases or: How to avoid Floundering", *Proc. 5th ÖGAI*, Springer-Verlag, 1989.

[Dec96] Decker, H. "An Extension of SLD by Abduction and Integrity Maintenance for View Updating in Deductive Databases", *To appear in Joint International Conference and Symposium on Logic Programming (JICSLP'96)*, Bonn (Germany), 1996.

[DMB93] Denecker, M.; Missiaen, L.; Bruynooghe, M. "Temporal Reasoning with Abductive Event Calculus", *Proc. of the 10th ECAI*, 1993, pp. 384-388.

[DMMP91] Decker, M.; Moerkotte, G.; Müller, H.; Possega,J. "Consistency Driven Planning", *Proc. of the 5th Portuguese Conference on Artificial Intelligence*, 195-209, Albufeira, Portugal, 1991.

[DTU96] Decker, H.; Teniente, E.; Urpí, T. "How to Tackle Schema Validation by View Updating", *Proc. of the EDBT'96*, Avignon, France, 1996, pp. 535-549.

[JMSY92] Jaffar, J,; Michaylov, S.; Stuckey, P.J.; Yap, R. "The CLP(R) Language and System", *ACM Transactions on Programming Languages and Systems*, Vol. 14, No. 3, 1992, pp. 339-395.

[JM94] Jaffar, J,; Maher, M. "Constraint Logic Programming: A Survey", *Journal of Logic Programming*, Vol. 19, No. 20, 1994, pp. 503-581.

[KM90] Kakas, A.; Mancarella, P. "Database Updates through Abduction", *Proc. of the 16th VLDB Conference*, Brisbane, Australia, 1990, pp. 650-661.

[KM90] Kakas, A.; Michael, A. "Integrating Abductive and Constraint Logic Programming", *Proc. of the 12th ICLP Conference*, Kanagawa, Japan, 1995, pp. 399-413.

[Llo87] Lloyd, J.W. *Foundations on Logic Programming*, 2nd edition, Springer, 1987.

[LT84] Lloyd, J.W.; Topor, R.W. "Making Prolog More Expressive", *Journal of Logic Programming*, 1984, No. 3, pp. 225-240.

[Mot89] Motro, A. "Using Integrity Constraints to Provide Intensional Answers to Relational Queries", *Proc. of the 15th VLDB Conference*, Amsterdam, 1989, pp. 237-246.

[Oli91] Olivé, A. "Integrity Checking in Deductive Databases", *Proc. of the 17th VLDB Conference*, Barcelona, Catalonia, 1991, pp. 513-523.

[Stu91] Stuckey, P.J. "Constructive Negation for Constraint Logic Programming", *Proc. of the 6th IEEE Symposium on Logic in Computer Science*, Amsterdam, 1991, pp. 328-339.

[TO95] Teniente, E.; Olivé. A. "Updating Knowledge Bases while Maintaining their Consistency", *The VLDB Journal*, Vol. 4, Num. 2, 1995, pp. 193-241.

[TU95] Teniente, E., Urpi, T. "A Common Framework for Classifying and Specyfing Deductive Database Updating Problems". *Proc. Eleventh International Conference on Data Engineering (ICDE'95).* Taipei, 1995, pp. 173-183.

[Ull88] Ullman, J.D. *Principles of Database and Knowledge-Base Systems*, Computer Science Press, New York, 1988.

[UO92] Urpí, T.; Olivé, A. "A Method for Change Computation in Deductive Databases", *Proc. of the 18th VLDB Conference*, Vancouver, 1992, pp. 225-237.

[Wüt93] Wüthrich, B. "On Updates and Inconsistency Repairing in Deductive databases", *Int. Conf. on Data Engineering (ICDE'93)*, Vienna, 1993, pp. 608 - 615.

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Research Reports – 1996

LSI–96–1–R "(Pure) Logic out of Probability", Ton Sales.

LSI–96–2–R "Automatic Generation of Multiresolution Boundary Representations", C. Andújar, D. Ayala, P. Brunet, R. Joan-Arinyo, and J. Solé.

LSI–96–3–R "A Frame-Dependent Oracle for Linear Hierarchical Radiosity: A Step towards Frame-to-Frame Coherent Radiosity", Ignacio Martin, Dani Tost, and Xavier Pueyo.

LSI–96–4–R "Skip-Trees, an Alternative Data Structure to Skip-Lists in a Concurrent Approach", Xavier Messeguer.

LSI–96–5–R "Change of Belief in SKL Model Frames (Automatization Based on Analytic Tableaux)", Matías Alvarado and Gustavo Núñez.

LSI–96–6–R "Compressibility of Infinite Binary Sequences", José L. Balcázar, Ricard Gavaldà, and Montserrat Hermo.

LSI–96–7–R "A Proposal for Word Sense Disambiguation using Conceptual Distance", Eneko Agirre and German Rigau.

LSI–96–8–R "Word Sense Disambiguation Using Conceptual Density", Eneko Agirre and German Rigau.

LSI–96–9–R "Towards Learning a Constraint Grammar from Annotated Corpora Using Decision Trees", Lluis Màrquez and Horacio Rodríguez.

LSI–96–10–R "POS Tagging Using Relaxation Labelling", Lluís Padró..

LSI–96–11–R "Hybrid Techniques for Training HMM Part-of-Speech Taggers", Ted Briscoe, Greg Grefenstette, Lluís Padró, and Iskander Serail.

LSI–96–12–R "Using Bidirectional Chart Parsing for Corpus Analysis", A. Ageno and H. Rodríguez.

LSI–96–13–R "Limited Logical Belief Analysis", Antonio Moreno.

LSI–96–14–R "Logic as General Rationality: A Survey", Ton Sales.

LSI–96–15–R "A Syntactic Characterization of Bounded-Rank Decision Trees in Terms of Decision Lists", Nicola Galesi.

LSI–96–16–R "Algebraic Transformation of Unary Partial Algebras I: Double-Pushout Approach", P. Burmeister, F. Rosselló, J. Torrens, and G. Valiente.

LSI-96-17-R "Rewriting in Categories of Spans", Miquel Monserrat, Francesc Rosselló, Joan Torrens, and Gabriel Valiente.

LSI-96-18-R "Strong Law for the Depth of Circuits", Tatsuie Tsukiji and Fatos Xhafa.

LSI-96-19-R "Learning Causal Networks from Data", Ramon Sangüesa i Solé.

LSI-96-20-R "Boundary Generation from Voxel-based Volume Representations", R. Joan-Arinyo and J. Solé.

LSI-96-21-R "Exact Learning of Subclasses of CDNF Formulas with Membership Queries", Carlos Domingo.

LSI-96-22-R "Modeling the Thermal Behavior of Biosphere 2 in a Non-Controlled Environment Using Bond Graphs", Angela Nebot, François E. Cellier, and Francisco Mugica.

LSI-96-23-R "Obtaining Synchronization-Free Code with Maximum Parallelism", Ricard Gavaldá, Eduard Ayguadé, and Jordi Torres.

LSI-96-24-R "Memoisation of Categorial Proof Nets: Parallelism in Categorial Processing", Glyn Morrill.

LSI-96-25-R "Decision Trees Have Approximate Fingerprints", Víctor Lavín and Vijay Raghavan.

LSI-96-26-R "Visible Semantics: An Algebraic Semantics for Automatic Verification of Algorithms", Vicent-Ramon Palasí Lallana.

LSI-96-27-R "Massively Parallel and Distributed Dictionaries on AVL and Brother Trees", Joaquim Gabarró and Xavier Messeguer.

LSI-96-28-R "A Maple package for semidefinite programming", Fatos Xhafa and Gonzalo Navarro.

LSI-96-29-R "Bounding the expected length of longest common subsequences and forests", Ricardo A. Baeza-Yates, Ricard Gavaldà, and Gonzalo Navarro.

LSI-96-30-R "Parallel Computation: Models and Complexity Issues", Raymond Greenlaw and H. James Hoover.

LSI-96-31-R "ParaDict, a Data Parallel Library for Dictionaries (Extended Abstract)", Joaquim Gabarró and Jordi Petit i Silvestre.

LSI-96-32-R "Neural Networks as Pattern Recognition Systems", Lourdes Calderón.

LSI-96-33-R "Semàntica externa: una variant interessant de la semàntica de comportament" (written in Catalan), Vicent-Ramon Palasí Lallana.

LSI-96-34-R "Automatic verification of programs: algorithm ALICE", V.R. Palasí Lallana.

LSI-96-35-R "Multiresolution Approximation of Polyhedral Solids", D. Ayala, P. Brunet, R. Joan-Arinyo, I. Navazo.

LSI-96-36-R "Algebraic Transformation of Unary Partial Algebras II: Single-Pushout Approach", P. Burmeister, M. Montserrat, F. Rosselló, and G. Valiente.

LSI-96-37-R "Probabilistic Conditional Independence: A Similarity-Based Measure and its Application to Causal Network Learning", Ramon Sangüesa Solé, Joan Cabós Fabregat, and Ulises Cortés García.

LSI-96-38-R "Analysing the Process of Enforcing Integrity Constraints", Enric Mayol and Ernest Teniente.

LSI-96-39-R "Reducció de l'equivalència inicial visible a teoremes inductius" (written in Catalan), Vicent-Ramon Palasí Lallana.

LSI-96-40-R "A Compendium of Problems Complete for Symmetric Logarithmic Space", Carme Àlvarez and Raymond Greenlaw.

LSI-96-41-R "Semàntica algebraica del llenguatge AL: l'algorisme α" (written in Catalan), V.R. Palasí Lallana.

LSI-96-42-R "Partial Occam's Razor and its Applications", Carlos Domingo, Tatsuie Tsujiki, and Osamu Watanabe.

LSI-96-43-R "Transparent Distributed Problem Resolution in the MAKILA Multi-Agent System", Karmelo Urzelai.

LSI-96-44-R "The Intensional Events Method for Consistent View Updating", Dolors Costal, Ernest Teniente, and Toni Urpí.

Hardcopies of reports can be ordered from:

Nuria Sánchez
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo, 5
08028 Barcelona, Spain
secrelsi@lsi.upc.es

See also the Department WWW pages, `http://www-lsi.upc.es/www/`