A Logical Framework for Integrating Inconsistent Information in Multiple Databases*

Sandra de Amo[1], Walter A.Carnielli[2], and João Marcos[3]

[1] Faculty of Computer Science
Federal University of Uberlândia
Uberlândia, Brazil
deamo@ufu.br

[2] Group of Theoretical and Applied Logic
CLE/IFCH - State University of Campinas
Campinas, Brazil
carniell@cle.unicamp.br

[3] Centre for Logic and Philosophy of Science
Ghent University, Belgium
vegetal@cle.unicamp.br

Abstract. When integrating data coming from multiple different sources we are faced with the possibility of inconsistency in databases. In this paper, we use one of the paraconsistent logics introduced in [9, 7] (**LFI1**) as a logical framework to model possibly inconsistent database instances obtained by integrating different sources. We propose a method based on the sound and complete tableau proof system of **LFI1** to treat both the integration process and the evolution of the integrated database submitted to users updates. In order to treat the integrated database evolution, we introduce a kind of generalized database context, the *evolutionary databases*, which are databases having the capability of storing and manipulating inconsistent information and, at the same time, allowing integrity constraints to change in time. We argue that our approach is sufficiently general and can be applied in most circumstances where inconsistency may arise in databases.

1 Introduction

The treatment of inconsistencies arising from the integration of multiple sources has been a topic increasingly studied in the past years and has become an important field of research in databases. Since some pioneer work on database updates and belief revision in the eighties [17, 20], a great deal of work on multidatabases

* Author (1) was supported by an individual research grant from CNPq (Brazil). Author (2) was partially supported by a grant from the Alexander von Humboldt Foundation (Germany), by CAPES (Brazil) and by an individual research grant from CNPq (Brazil). Author (3) was supported by the Research Fund of Ghent University, project BOF2001/GOA/008.

and inconsistency management has been done during the last decade. Two basic approaches have been followed in solving the inconsistency problem in knowledge bases : belief revision ([21, 22]) and paraconsistent logic ([10, 12, 6]). The goal of the first approach is to make an inconsistent theory consistent, either by revising it or by representing it by a consistent semantics. So, the main concern there is to avoid contradictions. On the other hand, the paraconsistent approach allows reasoning in the presence of inconsistency, and contradictory information can be derived or introduced without trivialization. In this paper, we propose to treat inconsistencies arising from the integration of multiple databases by introducing a method based on the paraconsistent approach. We argue that in most situations inconsistent information can be useful, unavoidable and even desirable, like for instance in airline booking systems.

In recent work ([9, 7]), a family of paraconsistent logics called *Logics of Formal Inconsistency* (**LFI**s) has been introduced, and sound and complete axiomatic proof systems for this class of logics have been provided. The most important feature of these logics consists in the internalization of the concepts of consistency and inconsistency inside the object language. In this paper, we focus our attention in one of these logics, which we call **LFI1**, and use it as a logical framework to model integrated databases. We present the method Repair based on the inference mechanism of the sound and complete tableau system of **LFI1**, introduced in [8]. The method consists basically in constructing a repaired version of the integrated database where inconsistent information may appear. **LFI1** (with its 3-valued semantics) is used as the logical framework for the underlying model of this repaired version, which we call *paraconsistent databases*. We focus our attention on a particular class of integrity constraints and show that, as far as this particular class of constraints is concerned, the method is sound and complete : all paraconsistent databases returned by the method are repairs of the integrated database, i.e. they satisfy the integrity constraints and are as close as possible to the original (possibly) inconsistent integrated instance, and all possible repairs can be obtained through this procedure.

Example 1 (Running Example). Let us consider the local databases $\mathbf{R}_1 = \{R(a), Q(a), Q(b)\}$ and $\mathbf{R}_2 = \{R(c), Q(b)\}$. The first database verifies the condition $\mathrm{C}_1 = \forall x(\neg R(x) \vee Q(x))$ and the second one verifies $\mathrm{C}_2 = \forall x(\neg R(x) \vee \neg Q(x))$. However, the integrated database $\{R(a), R(c), Q(a), Q(b)\}$ violates both conditions C_1 and C_2. So, local databases may be consistent but when they are integrated, inconsistencies may appear. Even worse, the conditions may be mutually inconsistent or be only satisfied by an empty database as in the following situation : Let us consider a third local database $\mathbf{R}_3 = \{R(b), Q(b)\}$ and the condition $\mathrm{C}_3 = \forall x(\neg Q(x) \vee R(x))$. This database satisfies C_3 but the integrated database $\mathbf{I} = \{R(a), R(b), R(c), Q(a), Q(b)\}$ violates conditions C_1 and C_2. The three conditions C_1, C_2 and C_3 are rather incompatible in the sense that they are simultaneously satisfied only by empty databases.

The method Repair can be applied to $\mathbf{I}$ and produces the following database :

$$\mathbf{J} = \{\ R(a), R(b), \bullet R(c), \bullet Q(a), \bullet Q(b)\ \}$$

The symbol $\bullet$ preceding a ground atomic formula means that the information represented by the formula is controversial. Intuitively, the condition stated by C_1 enforces that each element in R must appear in Q. This condition is violated in the integrated database **I** because c belongs to R but not to Q. However, if the information "$c \in$ R" was taken as controversial, then C_1 would be verified (at least as far as the instantiation $x = c$ is concerned). In the same way, the condition stated by C_2 enforces that elements in R should not appear in Q. So, this condition is violated in **I** because a and b belong to R and Q simultaneously. If the two facts "$a \in$ Q" and "$b \in$ Q" were taken as controversial, then C_2 would be verified. The database **J** containing inconsistent information is called *paraconsistent database* and we can show that it satisfies (within **LFI1**) the integrity constraints IC. Besides, it constitutes a *repair* of the original instance, i.e., it is a paraconsistent database containing *minimal changes* w.r.t. the original integrated database **I** and which is consistent w.r.t. IC. As we will see in section 3, repairs are not unique, i.e., there are other paraconsistent databases satisfying the constraints and containing minimal changes w.r.t. the original (possibly inconsistent) database. For instance, $\mathbf{J}_1 = \{R(a), \bullet R(b), R(c), \bullet Q(a), Q(b), \bullet Q(c)\}$ is another repair of **I**. By using a *backtracking* mechanism, the method Repair can produce the set of repairs corresponding to a given database **I**.

Our notion of *repair* is more *refined* than the one introduced in [3], in the sense that it is closer to the original database. This follows from the fact that, contrarily to [3], in our approach inconsistent information are always kept inside the repaired database.

The method Repair is suitable to treat both static and dynamic aspects of inconsistency management in databases. The static aspect deals only with the integration of different database instances, by constructing the repaired version. The dynamic aspect of our approach deals with the evolution of the integrated databases submitted to user updates. In order to treat the dynamics of paraconsistent database evolution, we introduce a kind of generalized database context, the *evolutionary databases*, which are databases having the capability of storing and handling inconsistent information and, at the same time, allowing integrity constraints to change in time. The method Repair interacts with user updates (which may be a data or an integrity constraint update) in order to build a repaired version of the paraconsistent database produced after the user update.

We argue that our approach is sufficiently general and can be applied in most circumstances where inconsistency may arise in databases. It could be suitable for managing inconsistency in active and reactive databases and datawarehouses.

This paper is organized as follows : In section 2 we describe the syntax and the three-valued semantics of our Logic of Formal Inconsistency **LFI1** and present a sound and complete proof system for this logic. In section 3, we introduce the notion of *paraconsistent databases* and repairs. In section 4 we give the method for constructing repairs for paraconsistent databases obtained by the integration of different local consistent databases. In section 5 we generalize this method in order to treat the dynamic aspects of paraconsistent databases as well as more general situations where inconsistency may appear in a database context, these situations being captured by the notion of *evolutionary databases*.

Finally, in section 6 we discuss our perspectives for further work and compare our method with some other methods treating the problem of inconsistency in multiple databases. For lack of space, the proofs are just outlined.

2 LFI1 : A Three-valued Logic for Formal Inconsistency

In this section we describe the syntax and semantics of our Logic of Formal Inconsistency (**LFI1**). A detailed presentation can be found in our former paper [9].

Let **R** be a finite signature without functional symbols and **Var** a set of variables symbols. We assume the formulas of our logic to be defined in the usual way, as in the classical first-order logic, with the addition of a new symbol $\bullet$ (read "it is inconsistent"). So, a formula of **LFI1** is defined inductively by the following statements (and only by them) :

- If R is a predicate symbol of arity k and $x_1, ..., x_k$ are constants or variables, then $R(x_1, ..., x_k)$ and $x_1 = x_2$ are atomic formulas or atoms. The former is called a *relational* atom and the later an *equality* atom.
- If F, G are formulas and x is a variable then $F \vee G$, $\neg F$, $\forall x F$, $\exists x F$ and $\bullet F$ are formulas.

The notions of free and bound variables are defined as usual. If $x_1, \ldots, x_n$ are free variables of a formula F and $c_1, \ldots, c_n$ are constants or variables, we denote by $F[c_1, \ldots, c_n/x_1, \ldots, x_n]$ the formula obtained by replacing each occurrence of the variable x_i by c_i, for $i = 1, \ldots, n$. A *sentence* is a formula without free variables. In particular, an *atomic ground formula* or *ground atom* is an atomic formula which is a sentence. We denote by $\mathcal{G}$ and $\mathcal{S}$, the set of all ground atoms and the set of all sentences respectively.

We next define *interpretations* for formulas of **LFI1**, using *three-valued valuations* which are homomorphisms between sentences and the truth-values 0 (for "false"), 1 (for "true"), $\frac{1}{2}$ (for "partially true"). These homomorphisms are induced by the *connective matrices* and *distribution quantifiers* introduced below. It is important to notice that in a database context, one only considers *Herbrand interpretations*, those for which **Dom** (the set of constants of the language) is the domain of valuation of the variables and where each constant symbol is interpretated by itself.

Definition 1. *Let* **R** *be a finite signature. An* interpretation *over* **R** *is an application* $\delta : \mathcal{G} \rightarrow \{0$ *(false)*, 1 *(true)*, $\frac{1}{2}$ *(inconsistent)*$\}$.

An interpretation of ground atoms can be extended to the propositional sentences of $\mathcal{S}$ *in a natural way by using the connective matrices in figure 1(a). The connective* $\wedge$ *is derived from of* $\vee, \neg$: $A \wedge B \equiv \neg(\neg A \vee \neg B)$*. The derived matrix for* $\wedge$ *is given in figure 1 (b). The connective* $\rightarrow$ *is defined in* **LFI1** *as* $A \rightarrow B \equiv B \vee \neg(A \vee \bullet A)$.[1] *It is easy to show ([9]) that* $\bullet$ *cannot be derived from the other propositional connectives* $\vee$ *and* $\neg$*. So,* $\vee, \neg, \bullet, \forall$ *can be taken as the primitive logical symbols of our language.*

[1] In this paper, we omit the matrix for $\rightarrow$, since the class of **LFI1**-formulas we will be interested in (the *integrity constraints*) does not use this connective.

$\vee$	1	$\frac{1}{2}$	0
1	1	1	1
$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$
0	1	$\frac{1}{2}$	0

	$\neg$	$\bullet$
1	0	0
$\frac{1}{2}$	$\frac{1}{2}$	1
0	1	0

(a)

$\wedge$	1	$\frac{1}{2}$	0
1	1	$\frac{1}{2}$	0
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0
0	0	0	0

(b)

The extension of δ to the quantified sentences in $\mathcal{S}$ is obtained by means of the concept of distribution quantifiers, *introduced in [13]. Basically, this concept translates our basic intuition that an universal quantifier should work as a kind of unbounded conjunction and an existential quantifier as an unbounded disjunction. A* valuation *is an application v : $\mathbf{Var} \rightarrow \mathbf{Dom}$. We extend δ to the quantified sentences as follows :*

- *$\delta(\forall x A(x)) = 1$ iff for all valuations v we have $\delta(A[v(x)/x]) = 1$,*
- *$\delta(\forall x A(x)) = 0$ iff there exists a valuation v such that $\delta(A[v(x)/x]) = 0$,*
- *$\delta(\forall x A(x)) = \frac{1}{2}$ iff for all valuations v we have $\delta(A[v(x)/x]) = 1$ or $\frac{1}{2}$, and there exists a valuation v' such that $\delta(A[v'(x)/x]) = \frac{1}{2}$*
- *$\delta(\exists x A(x)) = 1$ iff there exists a valuation v such that $\delta(A[v(x)/x]) = 1$,*
- *$\delta(\exists x A(x)) = 0$ iff for all valuations v we have $\delta(A[v(x)/x]) = 0$,*
- *$\delta(\exists x A(x)) = \frac{1}{2}$ iff for all valuations v we have $\delta(A[v(x)/x]) = 0$ or $\frac{1}{2}$, and there exists a valuation v' such that $\delta(A[v'(x)/x]) = \frac{1}{2}$.*

It is easy to see that $\delta(\forall x A(x)) = \delta(\neg\exists x \neg A(x))$ as usual in classical first-order logic.

Definition 2. *Let $F(x_1, ..., x_n)$ be a formula of* **LFI1** *with free variables $x_1, \ldots$, x_n, v a valuation and δ an interpretation. We say that (δ, v)* satisfies *$F(x_1, ..., x_n)$ (denoted by $(\delta, v) \models F(x_1, ..., x_n)$) iff $\delta(F[v(x_1), ..., v(x_n)/x_1, ..., x_n])$ is 1 or $\frac{1}{2}$.*

Example 2. Let R be a binary predicate symbol. Let δ be the interpretation $\delta(R(a,b)) = 1, \delta(R(c,b)) = \frac{1}{2}$ and $\delta(R(p,q)) = 0$ for all (p,q) such that $p \neq c$ and $p \neq a$, or $q \neq b$. Then, $(\delta, v) \models (\exists x \bullet R(x,y) \wedge \neg\forall x R(x,y))$, where v is a valuation such that $v(y) = b$.

If $(\delta$,$v) \models F$ for each valuation v, we say that δ is a *model* of F (denoted $\delta \models F$). In this case, F is **LFI1**-*satisfiable*. A formula is **LFI1**-*valid* if for each interpretation δ, $\delta \models F$. An **LFI1** sentence F is a *logical consequence* of a set of **LFI1** sentences Γ if all models of F are also models of all formulas in Γ (we denote this by $\Gamma \models F$).

The logic **LFI1** is a *paraconsistent logic* since it does not verify the *principle of explosion*, i.e., A, $\neg A \not\models B$ for all B. In fact, if we take the interpretation δ of example 2, we see that $\delta \models R(c,b)$ and $\delta \models \neg R(c,b)$ but $\delta \not\models R(b,a)$.

A tableau proof system for **LFI1**. Before introducing our proof system, we need some definitions concerning the tableaux terminology :

Definition 3. *A* signed *formula of* **LFI1** *is an expression of the form $T(A)$ or $F(A)$, where A is a formula of* **LFI1**, *or the special symbol $\perp$. If A is atomic (resp. ground), the signed formula is said to be atomic (resp. ground).*

In what follows S is a finite set of atomic signed formulas, α and α_i ($i \in \{1,2\}$) are signed formulas. An inference rule *is an expression of one the following forms :*

$$\begin{array}{c} S:\alpha \\ | \\ S:\alpha_1,\ S:\alpha_2 \end{array}$$

and-rules : *from $S \cup \{\alpha\}$ we can infer $S \cup \{\alpha_1, \alpha_2\}$*

$$\begin{array}{ccc} & S:\alpha & \\ & | & \\ S:\alpha_1 & & S:\alpha_2 \end{array}$$

or-rules: *from $S \cup \{\alpha\}$ we can infer $S \cup \{\alpha_1\}$* or $S \cup \{\alpha_2\}$

If r is an **and-rule** *(resp. an* **or-rule***) then we define $r(S \cup \{\alpha\}) = S \cup \{\alpha_1, \alpha_2\}$ (resp. $r(S \cup \alpha) = S \cup \{\alpha_1\}$ or $r(S) = S \cup \{\alpha_2\}$).*

The inferences rules[2] of our proof system are listed in table 1.

and-rule				**or-rule**			
	α	α_1	α_2		α	α_1	α_2
(1)	T(A∧B)	T(A)	T(B)	(5)	F(A∧B)	F(A)	F(B)
(2)	F(A∨B)	F(A)	F(B)	(6)	T(A∨B)	T(A)	T(B)
(3)	F(¬A)	T(A)	F(•A)	(7)	T(¬A)	F(A)	T(•A)
(4)	T(¬¬A)	T(A)	T(A)	(8)	F(¬¬A)	F(A)	F(A)
(9)	T(•A)	T(A)	T(¬A)				
(10)	T(•• A)	$\perp$	$\perp$	(11)	F(•A)	F(A)	F(¬A)
(12)	F(•(A∧B))	F(•A∧B)	F(•B∧A)	(14)	T(•(A∧B))	T(•A∧B)	T(•B∧A)
(13)	F(•(A∨B))	F(•A∧¬B)	F(•B∧¬A)	(15)	T(•(A∨B))	T(•A∧¬ B)	T(•B∧¬A)
(16)	T($\forall x$A(x))	T(A(t))	T(A(t))	(20)	F($\forall x$A(x))	F(A(s))	F(A(s))
(17)	T($\exists x$A(x))	T(A(s'))	T(A(s'))	(21)	F($\exists x$A(x))	F(A(t))	F(A(t))
(18)	T(¬$\forall x$A))	T($\exists x$¬A)	T($\exists x$¬A)	(22)	F(¬$\forall x$A)	F($\exists x$¬A)	F($\exists x$¬A)
(19)	T(¬$\exists x$A)	T($\forall x$¬A)	T($\forall x$¬A)	(23)	F(¬$\exists x$A)	F($\forall x$¬A)	F($\forall x$¬A)
(24)	T(•($\forall x$A)	T($\exists x$•A)	T($\forall x$A)	(26)	F(•(∀xA))	F($\exists x$•A)	F($\forall x$A)
(25)	T(•($\exists x$A)	T($\exists x$•A)	T($\forall x$¬A)	(27)	F(•($\exists x$A))	F($\exists x$•A)	F($\forall x$¬A)
(28)	T ($A[x]$)	T($A[x/y]$)	T($A[x/y]$) (*)	(29)	F($A[x]$)	F($A[x/y]$)	F($A[x/y]$) (*)

t, t' are arbitrary terms; s is a new term w.r.t. $\forall x$ A(x), i.e., it does not appear in any branch containing $\forall x$ A(x); and s' is a new term w.r.t. $\exists x$ A(x), i.e., it does not appear in any branch containing $\exists x$ A(x).
(*) if T($x = y$) is in S.

Table 1. A tableau proof system for **LFI1**

We now describe the *tableaux method* underlying the proof system :

Definition 4. *A tableau for a set S of signed formulas is a (finite) tree $\mathcal{T}$ whose nodes are sets of signed formulas and which is constructed as follows :*

1. *the root of $\mathcal{T}$ is the set S.*
2. *a node is said to be* closed (open *otherwise) if it contains signed formulas of the form T(A) and F(A) for some A, or if it contains F($x = x$) or if it contains the special symbol $\perp$.*

[2] In fact, in order to simplify the presentation and for the purposes of the restricted class of sentences we treat in this paper, we have omitted the rules for $\to$ which are present in the original logic **LFI1** [9].

3. *a node S_2 is an* and-*successor (resp. an* or-*successor) of an* **open** *node S_1 if it is obtained by applying one* and-*rule (resp. one* or-*rule) r to an arbitrary element α of S_1. ($S_2 = (S_1 - \{\alpha\}) \cup r(S_1 : \alpha)$). Closed nodes have no successors.*

A tableau is closed *if all its leaves are closed. It is* open *if at least one leaf is open. A* proof *of a formula A is a closed tableau for the singleton $\{F(A)\}$. We say that A is* provable *(denoted by $\vdash A$) if there is a proof of A. A* derivation *of a formula A from a finite set Γ is a closed tableau for the set $\Gamma_T \cup F(A)$, where $\Gamma_T = \{T(X) \mid X \in \Gamma\}$. We say that A is* derived from *Γ (denoted $\Gamma \vdash A$) if there is a derivation of A from Γ.*

Example 3. $\vdash$ A $\vee$ $\neg$ A and $\bullet$A $\vdash$ $\neg$A. Indeed :

$$\begin{array}{c}
\underline{\mathrm{F(A\vee\neg A)}} \\
\mid \ \text{(by (2))} \\
\mathrm{F(A)},\ \underline{\mathrm{F(\neg A)}} \\
\mid \ \text{(by (3))} \\
\mathrm{F(A), T(A), F(\bullet A)} \\
\mid \\
\text{closed}
\end{array}
\qquad\qquad
\begin{array}{c}
\mathrm{F(\neg\, A)},\ \underline{\mathrm{T(\bullet A)}} \\
\mid \ \text{(by (9))} \\
\mathrm{F(\neg\, A),\ T(A),\ T(\neg\, A)} \\
\mid \\
\text{closed}
\end{array}$$

The following result guarantees the soundness and completeness of the proof system with respect to the logic **LFI1** :

Theorem 1 ([8, 13]). *Let Γ be a set of* **LFI1** *formulas and A be a* **LFI1** *formula. Then, $\Gamma \vdash A$ if and only if $\Gamma \models A$.*

Remark. We notice that in our paraconsistent logic **LFI1**, the third truth-value $\frac{1}{2}$ should not be read as "undefined" as in Kleene's logic, but rather as "overdefined". Our logic is paraconsistent, while Kleene's system is not. The two approaches are conceptually incomparable.

3 Paraconsistent Databases

In this section we use the logical formalism of **LFI1** to generalize the notion of database instance so as to allow the storage of inconsistent information in our databases. We assume the reader to be familiar with traditional database terminology [1].

Definition 5 (p-instance). *Let* **R** *be a database schema*[3]. *A* paraconsistent instance *(*p-instance*) over* **R** *is an interpretation* **I** *such that for each $R \in \mathbf{R}$ the set $\mathbf{I}_R = \{u : \mathbf{I}(R(u)) = 1$ or $\mathbf{I}(R(u)) = \frac{1}{2}\}$ is finite. So, an instance over* **R** *can be viewed as a finite set of* relations *where each relation is a finite set of tuples (those having truth-values 1 or $\frac{1}{2}$). A tuple u over R such that $\mathbf{I}(R(u)) = \frac{1}{2}$ is intended to be* controversial, *i.e. there may be evidence in favor of $R(u)$ and also*

[3] A set of relational names of a given arity.

evidence against $R(u)$.[4] *On the other hand, if $\mathbf{I}(R(u)) = 1$, $R(u)$ is intended to be a safe information. A p-instance where all tuples have truth-value 1 is called simply an* instance. *We denote by $adom(\mathbf{I})$ the active domain of $\mathbf{I}$, i.e. the set of constants appearing in the relations of $\mathbf{I}$. For the sake of simplification, we use the informal notation of example 1 for denoting p-instances, with the obvious translation.*

Definition 6 (Integrity Constraints). *An* integrity constraint *over a database schema* $\mathbf{R}$ *is an* **LFI1** *sentence of the form*

$$\forall x_1 \forall x_2 \dots \forall x_n (\bigvee_{i=0}^{p} R_i(u_i) \vee \bigvee_{j=1}^{q} \neg Q_j(v_j) \vee \bigvee_{l=0}^{s} \varphi_l)$$

where R_i and Q_j are relational atoms, u_i, v_j are tuples of variables appearing in $\{x_1, \dots, x_n\}$ and φ_l are equality atoms or negations of equality atoms.

Several important constraints that appear in databases fit into this form. Indeed, this class of sentences coincides with the class of *full dependencies* described in [1] (including functional dependencies, set inclusion dependencies, transitivity dependencies). However, inclusion dependencies of the form $\forall x(\neg P(x) \vee \exists y Q(x, y))$ do not belong to this class.

Example 4 (Running Example - Continued). Let IC = $\{\mathrm{C}_1, \mathrm{C}_2, \mathrm{C}_3\}$ and $\mathbf{I}$ be, respectively, the set of formulas and the integrated database instance mentioned in example 1. It is clear that each C_i is an integrity constraint (accordingly to definition 6). The two instances $\mathbf{J}$ and $\mathbf{J}_1$ introduced in this example are p-instances (where the notation $\bullet R(u)$ means $\mathbf{J}(R(u)) = \frac{1}{2}$). A simple calculation using the matrices in figure 1 will convince us that the p-instances $\mathbf{J}$ and $\mathbf{J}_1$ satisfy IC.

In section 4 we will present a method for repairing instances which are possibly inconsistent w.r.t. a given set of integrity constraints. This method is based on the tableau proof system for **LFI1** which has been introduced in the previous section. The soundness and completeness of this proof system w.r.t. *finite* structures (one reminds that a database instance is a finite structure) is essential for proving the soundness of the method. Theorem 1 guarantees the completeness of the tableau system but it is important to emphasize that this result is achieved when "$\Gamma \models$ A" means *all unrestricted interpretations (not necessarily finite) satisfying Γ also satisfy A.* Unfortunately, due to Trakhtenbrot's Theorem [23], this completeness result cannot be proven for finite structures in general. However, for the special class of integrity constraints one can prove the following theorem which is essential in the remainder of the paper :

Theorem 2. *Let IC be a set of integrity constraints and C be an integrity constraint over* $\mathbf{R}$. *Then, $IC \vdash C$ if and only if $IC \models_{fin} C$ (all* finite *models of IC are also models of C).*

[4] These tuples must be understood as "overdefined" instead of "undefined" as in Kleene's logic.

This follows from a well-known theorem for first-order logic (which can be extended to **LFI1** using the techniques of [19]), stating that the satisfiability problem (SAT) for the Bernays-Schöenfinkel class $\exists^*\forall^*$ is decidable and so unrestricted and finite satisfiability are equivalent [15].

The following definition will be helpful in the remainder of the paper :

Definition 7. *Let IC be a set of integrity constraints. A tableau is called* reduced *for F(IC) if (a) each of its leaves is either closed or is a set of signed formulas of the form T(X), F(X) or F($\bullet$X), where X is an atomic formula, (b) rule (11) has not been used in the derivation of the tableau nodes and (c) rules (28) and (29) cannot be applied to any leaf.*

The proof of the following result is straightforward and is omitted.

Proposition 1. *Let IC be a a set of integrity constraints. Then:*

1. *There is a unique reduced tableau for F(IC), which we call* the reduced tableau *for F(IC).*
2. *If $\mathcal{X}$ is an open leaf of the reduced tableau for F(IC) and T(X) $\in \mathcal{X}$ then F($\bullet$X) $\in \mathcal{X}$.*

We denote by reduction*(IC) the set* $\{L - \{F(\bullet X) \mid F(\bullet X) \in L\} \mid L$ *is an open leaf of the reduced tableau for F(IC)*$\}$ *(i.e., the set of the open leaves of the reduced tableau for F(IC) without the signed formulas of the form F($\bullet$X)).*

*Example 5 (*Running Example - Continued). Let C_1 be the integrity constraint $\forall x(\neg R(x) \vee Q(x))$ of example 1. A simple calculation shows that the reduced tableau for $\mathrm{F}(\mathrm{C}_1)$ contains only the leaf $\{\mathrm{T}(R(s)),\ \mathrm{F}(Q(s)),\ \mathrm{F}(\bullet R(s))\}$ Hence, reduction(C_1)= $\{\{\mathrm{T}(R(s)),\ \mathrm{F}(Q(s))\}\}$.

Repair Databases. Let us suppose the situation we have described in example 1 : we are given (1) a database specification which is the integration of several local databases, and (2) an instance which violates the integrity constraints. We want to build a repaired version "as close as possible" to the given instance which will verify the integrity constraints (w.r.t. the semantics of **LFI**). Our presentation generalizes the ideas presented in [3] which we have suitably adapted to our paraconsistent environment. The following definition aims at specifying what we mean by *as close as possible* :

Definition 8. *Let* $\mathbf{I}$ *and* $\mathbf{J}$ *be p-instances over a database schema* $\mathbf{R}$.

The distance *between* $\mathbf{I}$ *and* $\mathbf{J}$ *(a generalization of the well-known Hamming-distance) is given by :*

$$\mathrm{d}(\mathbf{I},\mathbf{J}) \ = \sum_{u \in \mathbf{I}_{\mathrm{R}} \cup \mathbf{J}_{\mathrm{R}}, \mathrm{R} \in \mathbf{R}} \mid \mathbf{I}(R(u)) - \ \mathbf{J}(R(u)) \mid$$

For p-instances $\mathbf{J}$ *and* $\mathbf{K}$*, we define* $\mathbf{J} \leq_{\mathbf{I}} \mathbf{K}$ *if* $\mathrm{d}(\mathbf{I},\mathbf{J}) \leq \mathrm{d}(\mathbf{I},\mathbf{K})$*.*

Obviously, our definition of *distance* satisfies the desirable properties of a *distance* in measure theory : (1) $\mathrm{d}(\mathbf{I},\mathbf{J}) = 0$ iff $\mathbf{I} = \mathbf{J}$, (2) $\mathrm{d}(\mathbf{I},\mathbf{J}) = \mathrm{d}(\mathbf{J},\mathbf{I})$ and (3) $\mathrm{d}(\mathbf{I},\mathbf{K}) \leq \mathrm{d}(\mathbf{I},\mathbf{J}) + \mathrm{d}(\mathbf{J},\mathbf{K})$.

Definition 9. *Let* **R** *be a database schema, IC a finite set of integrity constraints over* **R** *and* **I** *an instance over* **R** *(***I** *does not necessarily satisfy the integrity constraints in IC). A* repair *of* **I** *is a p-instance* **J** *satisfying IC which is* $\leq_{\mathbf{I}}$*-minimal among those satisfying IC.*

Example 6 (Running Example - Continued). Let us consider the situation described in example 1. A simple calculation yields : d(**J**,**I**) = 1.5, d($\mathbf{J}_1$,**I**) = 1.5. Let now consider the p-instance $\mathbf{J}_2 = \{\bullet R(a), \bullet R(b), R(c), \bullet Q(a), Q(b), \bullet Q(c)\}$. We can easily verify that $\mathbf{J}_2$ satisfies the integrity constraints IC and d($\mathbf{J}_2$,**I**) = 2. So, $\mathbf{J}_2$ is not a repair, even though it satisfies the constraints. It can be shown that **J** and $\mathbf{J}_1$ are repairs.

The definition of repair database we have just introduced satisfies some desirable properties. Firstly, we notice that if **I** satisfies the constraints IC then it does not need to be repaired (d(**I**,**I**) = 0). Moreover,

Proposition 2. *The repair of an instance always exists and, in general, is not unique.*

Proof. As IC is **LFI1**-satisfiable (see Theorem 5) there exists a p-instance **J** which satisfies IC. If **J** is a repair of **I**, we are done. If not, there exists a p-instance **J'** satisfying $\mathcal{C}$ such that d(**I**,**J'**) < d(**I**,**J**). We repeat the argument for **J'**. Eventually, we will find a repair of **I** (which can be **I** itself if **I** satisfies IC). The repair, in general, is not unique, as it is illustrated in example 6. Obviously, if $\mathbf{J}_1$ and $\mathbf{J}_2$ are repairs of **I** then d(**I**,$\mathbf{J}_1$) = d(**I**,$\mathbf{J}_2$).

We notice that in our definition of repair we have assumed that the instance being repaired is a (classical) instance (the integrity constraints are first-order sentences). We did so because our first concern was the development of a logical framework to treat inconsistencies arising from the integration of local databases. However, this assumption can be dropped and the notion of repair can be easily extended to p-instances.

4 A Method for Building Repair Databases

In this section we introduce a method to construct repair databases. This method can be viewed as a *static repairing process* because it concerns only the process of data integration : it takes as input a possibly inconsistent database instance resulting from the integration of several local consistent databases and produces a repaired version of this integrated instance. However, future updates over this (paraconsistent) repair version have to be monitored to insure a repairing process after each transaction. These *dynamic repairing process* will be treated in section 5, where we will generalize the "static" method we propose in the present section. The most important feature of our method relies on the fact that no information is lost in the repaired instance, but some information which was safe before may become controversial.

The method we propose here is based on the tableau proof system for **LFI1** presented in section 2. In general, the advantage of proof methods based on analytic tableaux is that, when a proof is not possible, in some cases counterexamples can be read from the derivation tree. We will explore this issue in order to construct a repair instance for a given database instance. For the sake of simplifying the presentation, we will consider only integrity constraints without constant symbols. Nevertheless, the method we present in this section can be extended to treat integrity constraints with constants.

Some Notations. Let $\mathbf{R}$ be a database schema and $\mathbf{I}$ be a p-instance over a relation schema $\mathrm{R} \in \mathbf{R}$. We denote by $\mathrm{S}(\mathbf{I})$ the set of the signed formulas obtained as follows : To each tuple v over R such that $\mathbf{I}(v) = 1$ (resp. 0, $\frac{1}{2}$), we associate the signed formula $\mathrm{T}(\mathrm{R}(v))$ (resp. $\mathrm{F}(\mathrm{R}(v))$, $\mathrm{T}(\bullet\mathrm{R}(v))$. If $\mathbf{I}$ is a p-instance over $\mathbf{R}$, we define $\mathrm{S}(\mathbf{I}) = \bigcup_{R\in\ \mathbf{R}} \mathrm{S}(\mathbf{I}(R)) \cup \{\mathrm{T}(a = a) \mid a \in \mathrm{adom}(\mathbf{I})\}$ $\cup$ $\{\mathrm{F}(a = b) \mid a, b \in \mathrm{adom}(\mathbf{I})$ and $a \neq b\}$. For instance, in example 1, $\mathrm{S}(\mathbf{I})$ $= \{\mathrm{T}(R(a)),\mathrm{T}(R(b)),\mathrm{T}(R(c)),\mathrm{T}(Q(a)),\mathrm{T}(Q(b)),\mathrm{F}(Q(c))\} \cup \{\mathrm{T}(a = a),\mathrm{T}(b = b),\mathrm{T}(c = c),\mathrm{F}(a = b),\mathrm{F}(a = c),...\}$.[5]

Conversely, let $\mathrm{D} \subseteq \mathbf{Dom}$ be a finite set of constants. To each set $\mathcal{S}$ of signed ground formulas over $\mathbf{R}$ and D satisfying the condition :

- if A is an atomic ground formula over $\mathbf{R}$ and D then $\mathcal{S}$ contains one and only one of the signed formulas T(A), F(A) or T(•A),

a p-instance is associated in the obvious way. We denote this p-instance by $\mathrm{S}^{-1}(\mathcal{S})$.

Let IC be a set of integrity constraints and $\mathbf{I}$ be an instance. Then reduction(IC) $= \{\mathrm{L}_1,\ldots,\mathrm{L}_m\}$[6], where each L_i is a set of signed atomic formulas without constants. For each valuation of variables v_j (within $\mathrm{adom}(\mathbf{I})$), let $\mathcal{X}^i_j = v_j(\mathrm{L}_i)$ be the set of signed ground formulas instantiated accordingly to v_j. So, for each i all the $\mathcal{X}^i_j$ have the same number k_i of signed atoms. Let $\mathcal{X}^i_1,...,\mathcal{X}^i_{j_i}$ be the set of instantiations of L_i. In what follows, we will fix an enumeration for the set $\{\mathrm{L}_1,\ldots,\mathrm{L}_m\}$ and an enumeration for each set of ground atoms $\mathcal{X}^i_j$.[7] So, we can assume that reduction(IC) is a list $[\mathrm{L}_1,\ldots,\mathrm{L}_m]$ and each $\mathcal{X}^i_j$ is also a list of signed atomic formulas.

Let L_i be a leaf in reduction(IC). A function $f : \{1, \ldots, j_i\} \to \{1, \ldots, k_i\}$ determines a choice of one signed ground atom in each instantiation $\mathcal{X}^i_j$ ($j = \{1, \ldots, j_i\}$) of L_i.

*Example 7 (*Running Example - Continued). Let us consider the set of integrity constraints IC $= \{\mathrm{C}_1, \mathrm{C}_2, \mathrm{C}_3\}$ and the integrated instance $\mathbf{I}$ introduced in example 1. A simple calculation yields :
reduction(IC) $= [[\mathrm{T}(R(x_1)),\mathrm{F}(Q(x_1))],[\mathrm{T}(R(x_2)),\mathrm{T}(Q(x_2))],[\mathrm{T}(Q(x_3)),\mathrm{F}(R(x_3))]]$.
Let us consider the first leaf, $[\mathrm{T}(R(x_1)), \mathrm{F}(Q(x_1))]$. As $\mathrm{adom}(\mathbf{I}) = \{a, b, c\}$, we have three instantiations for this leaf : $\mathcal{X}^1_1 = [\mathrm{T}(R(a)), \mathrm{F}(Q(a))]$, $\mathcal{X}^1_2 = [\mathrm{T}(R(b)),$

[5] In the remainder of the paper, we will omit the signed tuples $\mathrm{T}(a = a)$, $\mathrm{F}(a = b)$ (for a and $b \in \mathrm{adom}(\mathbf{I})$, $a \neq b$) in the description of $\mathrm{S}(\mathbf{I})$, presuming they are implicitly contained in this set.

[6] One reminds that reduction(IC) is the set of the open leaves of the reduced tableau for F(IC) without the signed formulas F(•X).

[7] It can be shown that these choices do not affect the result of the method Repair.

F($Q(b)$)], $\mathcal{X}_3^1$ = [T($R(c)$), F($Q(c)$)]. The function f such that $f(1) = 1$, $f(2) = 2$ and $f(3) = 2$ determines the choice of the first element in $\mathcal{X}_1^1$ (T($R(a)$)), the second element in $\mathcal{X}_2^1$ (F($Q(b)$)) and the second element in $\mathcal{X}_3^1$ (F($Q(c)$)).

We are ready now for the description of the method :

Input : a database schema **R**, a finite set IC = $\{C_1, \ldots, C_n\}$ of integrity constraints over **R**, an instance **I** over **R**.
Output : a set $\mathcal{K}$ of repairs of **I** and n = d(**I**,**J**), for all **J** $\in \mathcal{K}$.

Method Repair :
(1) Leaves := reduction(IC);
    % A list of lists containing signed atomic formulas T(X) or F(X)
(2) **If** Leaves = $\emptyset$ **then** return $\mathcal{K}$ = {**I**} and $n = 0$ % **I** *is a valid formula*
(3) **else** $\mathcal{K} := \emptyset$; $n := 0$;
(4) **For each** f_1, f_2, ..., f_m **do**
    % for each choice of one element in each instantiated leaf
(5) $\mathcal{S}$:= S(**I**); $d := 0$;
(6) **For each** $l = 1, ..., m$ **do** % *for each leaf*
(7) **For each** $k = 1, ..., j_l$ **do** % *for each of its instantiations*
(8) **If** $\mathcal{X}_k^l \subseteq \mathcal{S}$ **then**
(9) choose the $f_l(k)$-th element A $\in \mathcal{X}_k^l$
    (A = T(X) or A = F(X), X a relational atom);
(10) $\mathcal{S} := (\mathcal{S} -$ {A}) $\cup$ { T($\bullet$X) };
(11) $d := d + \frac{1}{2}$;
(12) **If** $n = 0$ or $d = n$ **then** $\mathcal{K} := \mathcal{K} \cup \{\mathrm{S}^{-1}(\mathcal{S})\}$; n:= d;
(13) **elseif** $d < n$ **then** $\mathcal{K} := \{\mathrm{S}^{-1}(\mathcal{S})\}$; n:= d

Example 8 (Running Example - Continued). Let us consider the situation of example 7. We have :
Leaves = [[T($R(x_1)$), F($Q(x_1)$)], [T($R(x_2)$), T($Q(x_2)$)], [T($Q(x_3)$), F($R(x_3)$)]]

$\mathcal{X}_1^1$ = [T($R(a)$, F($Q(a)$)] $\mathcal{X}_2^1$ = [T($R(b)$, F($Q(b)$)] $\mathcal{X}_3^1$ = [T($R(c)$, F($Q(c)$)]
$\mathcal{X}_1^2$ = [T($R(a)$, T($Q(a)$)] $\mathcal{X}_2^2$ = [T($R(b)$, T($Q(b)$)] $\mathcal{X}_3^2$ = [T($R(c)$, T($Q(c)$)]
$\mathcal{X}_1^3$ = [T($Q(a)$, F($R(a)$)] $\mathcal{X}_2^3$ = [T($Q(b)$, F($R(b)$)] $\mathcal{X}_3^3$ = [T($Q(c)$, F($R(c)$)]

The number m of leaves is 3, $k_1 = k_2 = k_3 = 2$ = number of atoms in each leaf and $j_1 = j_2 = j_3 = 3$ = number of instantiations for each leaf. Let us consider the following choice in step (4) : $f_1(1) = f_1(2) = f_1(3) = f_2(1) = f_2(2) = f_2(3) = f_3(1) = f_3(2) = f_3(3) = 1$. In step (5) we have : $\mathcal{S}$ = S(**I**) = {T($R(a)$), T($R(b)$), T($R(c)$), T($Q(a)$), T($Q(b)$), F($Q(c)$)}. Only the instantiations $\mathcal{X}_3^1$, $\mathcal{X}_1^2$ and $\mathcal{X}_2^2$ may verify the condition in step (8). The choice (f_1, f_2, f_3) (step (4)) implies that the first elements in each instantiation will be chosen. By repeating steps (10) and (11) for each of these instantiations, we obtain at the end of the iteration (7) $\mathcal{S}$ = S(**I**) = {$\bullet$T($R(a)$), $\bullet$T($R(b)$), $\bullet$T($R(c)$), T($Q(a)$), T($Q(b)$), F($Q(c)$)} and $d = 1.5$. As $n = 0$, the associated instance is inserted into $\mathcal{K}$ and n is instantiated with 1.5. All these calculations are repeated for each choice of

(f_1, f_2, f_3). If, for the next choice one obtains a value for d greater than 1.5, then the set $\mathcal{S}$ will not be included in $\mathcal{K}$. On the other hand, if d is smaller than 1.5, then $\mathcal{K}$ is instantiated with the unary set $\{\mathcal{S}\}$.

We notice that if IC is a **LFI1**-valid formula or if **I** satisfies IC then the repair returned by the method coincides with **I**. In fact, in the first case, the algorithm stops in step (2) and in the second case, the algorithm stops in step (8) for each iteration: as **I** satisfies IC, then in each element of Leaves and for all instantiations of its variables, we have a signed ground atomic formula T(A) (resp. F(A)) which matches with F(A) (resp. T(A)) in S(**I**). So T(A) (resp. F(A)) cannot appear in S(**I**). We notice also that the active domain of the repairs obtained by the method is the same as the one of the input instance **I** (no new constants are created).

Now, we will state and give sketchs of proofs of the main results of this section. The first one tells us that our method is sound, i.e. its result is a set of repairs of **I** and the second one guarantees that all repairs of **I** are contained in the output of Repair. The essential part of the proof of these two results is contained in Lemma 1 below. This lemma is a consequence of the fact that the tableau system for the class $\forall^*$ is sound and complete w.r.t. finite structures (Theorem 2). In what follows, **R** is a database schema, IC is a finite set of integrity constraints and **I** a safe instance over **R**.

Theorem 3. *All elements of the set $\mathcal{K}$ returned by executing the method* Repair *is a repair of* **I**.

The proof of this theorem follows immediately from Lemmas 1, 2 and 3 below.

Lemma 1. *Let* **J** *be a p-instance over* **R**. *Then, there exists a closed tableau for S(**J**) $\cup$ F(IC) if and only if **J** $\models$ IC.*

Proof. It can be shown that there exists a **LFI1**-sentence $\sigma_{\mathbf{J}}$ which characterizes **J** (a finite structure), i.e., for every arbitrary interpretation $\mathcal{I}$, $\mathcal{I} \models \sigma_{\mathbf{J}}$ if and only if $\mathcal{I}$ is isomorphic to **J** (this is due to an extension to **LFI1** of a well-known theorem for first-order logic ([18]). It can be shown that there is a closed tableau for $\mathrm{T}(\sigma_{\mathbf{J}}) \cup \mathrm{F(IC)}$ if and only if there exists a closed tableau for S(**J**) $\cup$ F(IC). The existence of a closed tableau for $\mathrm{T}(\sigma_{\mathbf{J}}) \cup \mathrm{F(IC)}$ is equivalent to affirming that $\sigma_{\mathbf{J}} \models_{fin}$ IC, by Theorem 2. Let us suppose that **J** $\models$ IC and let **J'** be a p-instance such that **J'** $\models \sigma_{\mathbf{J}}$. Then, **J'** and **J** are isomorphic and so, **J'** $\models$ IC. This proves that $\sigma_{\mathbf{J}} \models_{fin}$ IC. Conversely, suppose that $\sigma_{\mathbf{J}} \models_{fin}$ IC. Using the fact that **J** $\models \sigma_{\mathbf{J}}$, we can conclude that **J** $\models$ IC.

Lemma 2. *Let $\mathcal{K}$ be the output of the method* Repair *and* **J** $\in \mathcal{K}$. *Then, there exists a closed tableau for S(**J**) $\cup$ F(IC).*

Proof. Let us suppose without loss of generality that IC = {C}. So m = number of leaves = 1. If **J** = **I** : either Leaves = $\emptyset$ (the reduced tableau for F(IC) is closed) or for each instantiation there exists A $\in \mathcal{X}$ such that A $\notin$ S(**I**). Let A = T(X) (resp. F(X)). Then F(X) (resp. T(X)) is in S(**I**). So, in order to obtain a closed tableau for S(**I**) $\cup$ F(IC), we simply apply the rules of the reduced tableau for F(IC). If **J** $\neq$ **I** : For each instantiation $\mathcal{X}$ (step (7)) where there is a modification in $\mathcal{S}$, we have necessarily that $\mathcal{X} \subseteq \mathcal{S}$ (step (8)). A signed atomic formula A is chosen in $\mathcal{X}$ (step (9)). Let A = T(X). A is replaced by T(•X) in $\mathcal{S}$ (step (9)). From proposition 1(b), we know that F(•X) appears in the leaf corresponding to $\mathcal{X}$. So, the resulting tableau for S(**J**) ∪ F(IC) closes. Now, let A = F(X) (remind that A ∈ $\mathcal{S}$ and A ∈ $\mathcal{X}$). A is replaced by T(•X) in $\mathcal{S}$. We consider the same rules applied in order to obtain the reduced tableau for F(IC), and then we apply the rule (9) to T(•X), obtaining T(X) and T(¬ X). Hence, the resulting tableau for S(**J**) ∪ F(IC) closes.

Lemma 3. *Let $\mathcal{K}$ be the output of the method* Repair *and* **J** ∈ $\mathcal{K}$. *For each p-instance* **J'** *over* **R** *such that* **J'** ⊨ *IC we have that d(***J***,***I***) ≤ d(***J'***,***I***).*

Proof. If **J'** ⊨ IC then there exists a closed tableau for S(**J'**) ∪ F(IC) (Lemma 1). All p-instances **K** which are closest to **I** and for which there exists a closed tableau for S(**K**) ∪ F(IC) are obtained by the method. So, d(**J**,**I**) ≤ d(**J'**,**I**).

We conclude this section with the following theorem which states the completeness of the method :

Theorem 4. *Let* **I** *be an instance (safe) over* **R**. *If* **J** *is a repair of* **I** *then* **J** *is included in the output of the method* Repair.

Proof. If **J** is a repair of **I** then **J** ⊨ IC. By Lemma 1, there exists a closed tableau for S(**J**) ∪ F(IC). But all p-instances **J'** which are closest to **I** and for which there exists a closed tableau for S(**J'**) ∪ F(IC) are obtained by the method. So **J** is obtained by the method.

5 Updating Paraconsistent Databases

In this section we will study a dynamic repairing process for paraconsistent databases. This repairing process, which in fact is a simple adaptation of the method Repair for paraconsistent databases, will be executed after each update in order to control data inconsistencies. The method is sufficiently general and can treat two kinds of updates which possibly produce data inconsistencies : (1) data updates and (2) integrity constraints updates. It is important to notice that our update operations will not allow users to insert controversial information in the database. So, in our approach inconsistencies are viewed as an internal phenomenon and can only arise as a consequence of information conflict.

*Example 9 (*Running Example - Continued). Let **R**, **J** and IC be respectively the database schema, the integrated p-instance and the integrity constraints of example 1. Let us suppose that the user executes the operation $\text{ins}_{\text{R}}(d)$. After the update, the resulting p-instance violates the integrity constraint C_1. Let us now suppose that, instead of a data update, the user executes an integrity constraint update by inserting the new integrity constraint $\text{C}_4 = \forall x \forall y (\neg R(x) \vee \neg R(y) \vee x = y)$. The p-instance **J**, which satisfied the original integrity constraints IC before the update, now violates C_4.

The idea is to use the method Repair in order to build a repair for the updated inconsistent p-instance. So, the whole update process is composed of two steps : (1) the user update and (2) the repair process executed over the updated database.

Evolutionary Databases and Udpates. We introduce now the *evolutionary databases* which is a general database context where data and integrity constraints updates are allowed.

Definition 10. *Let* $\mathbf{R}$ *be a database schema. An* evolutionary instance (e-instance) *is a pair (*$\mathbf{I}$*,IC) where* $\mathbf{I}$ *is a p-instance and IC is a finite set of integrity constraints such that* $\mathbf{I} \models IC$*.*

Definition 11. *Let* $\mathbf{R}$ *be a database schema. An* update *over* $\mathbf{R}$ *is an expression of the form* $ins_{\mathrm{R}}(u)$, $del_{\mathrm{R}}(u)$, $ins(\varphi)$ *or* $del(\varphi)$*, where* $R \in \mathbf{R}$*,* φ *is an integrity constraint over* $\mathbf{R}$*, and* u *is a tuple over* $\mathbf{R}$*. The first two updates are called* data updates *and the other two are called* constraint updates.

The semantics of an update t is given by its effect over an e-instance ($\mathbf{I}$,IC). We define ($\mathbf{J}$,IC') = t($\mathbf{I}$,IC), where $\mathbf{J}$ and IC' are defined by the following table :

t	J	IC'
$\mathrm{ins}_{\mathrm{R}}(u)$	$\mathbf{J}(\mathrm{R}(u)) = 1$ if $\mathbf{I}(\mathrm{R}(u)) = 0$ $\mathbf{J}(\mathrm{R}(u)) = \mathbf{I}(\mathrm{R}(u))$ otherwise $\mathbf{J}(\mathrm{S}(v)) = \mathbf{I}(\mathrm{S}(v))$ for all tuples v over S if $\mathrm{S} \neq \mathrm{R}$	IC
$\mathrm{del}_{\mathrm{R}}(u)$	$\mathbf{J}(\mathrm{R}(u)) = 0$	IC
$\mathrm{ins}(\varphi)$	$\mathbf{I}$	$\mathrm{IC} \cup \{\varphi\}$
$\mathrm{del}(\varphi)$	$\mathbf{I}$	$\mathrm{IC} - \{\varphi\}$

This quite natural semantics implicitly presumes the following assumptions : (1) no inconsistent information can be inserted in the database by an user insertion, (2) an inconsistent information cannot become a safe information unless it is deleted by a user deletion and inserted as a safe information later on. We notice that the result ($\mathbf{J}$,IC') of a user update does not necessarily produce an e-instance, i.e., the integrity constraints IC' may be violated by the p-instance $\mathbf{J}$. Another important point is the following :

Theorem 5. *All sets of integrity constraints are* **LFI1***-satisfiable.*

So, the resulting set of integrity constraints IC' is always **LFI1**-satisfiable. This means that in our framework, inconsistencies can appear only at data level.

Proof of Theorem 5. First we find the reduced tableau for F(IC). As there is at least a negative relational atom $\neg\, \mathrm{R}(x_1, \ldots, x_n)$ in each sentence of IC, then all leaves in reduction(IC) contain a signed atomic formula of type $\mathrm{T}(\mathrm{R}(x_1, ..., x_n))$. For each leaf L_i, one considers one of these signed atoms A_i. Let $L = \{A_1, ..., A_k\}$. Let $\mathbf{Var} = \{y_1, ..., y_m\}$ be the set of variables in L. Let $v : \{y_1, ..., y_m\} \rightarrow \{1, ..., m\}$. We consider the instance built in the following way : for each $\mathrm{A}_i = \mathrm{T}(\mathrm{R}_i(z_1, ..., z_k))$, we define $\mathbf{I}(\mathrm{R}_i)(v(z_1), ..., v(z_k)) = \frac{1}{2}$. It is easy to see that $\mathrm{S}(\mathbf{I}) \cup \mathrm{F}(\mathrm{IC})$ has a closed tableau, using the same techniques of the proof of Lemma 2.

The Method Repair *as a Repairing Technique for p-instances.* Theorem 6 below guarantees that the method Repair executed over p-instances is correct and complete, and so, this method can be used as a repairing process after each update in order to control possible inconsistencies.

Theorem 6. *Let* $\mathbf{I}$ *be a p-instance and IC be a finite set of integrity constraints over a database schema* $\mathbf{R}$*. Let* $\mathbf{J} \in \mathcal{K}$*, where* $\mathcal{K}$ *is the output of the method* Repair*. Then* $\mathbf{J}$ *is a repair of* $\mathbf{I}$*. Conversely, if* $\mathbf{J'}$ *is a repair of* $\mathbf{I}$ *then* $\mathbf{J'}$ *is contained in* $\mathcal{K}$*.*

Proof. The completeness proof uses the same arguments of the proof of Theorem 4 and the soundness proof uses Lemmas 1, 2 and 3. Lemma 1 was proved for p-instances and the proofs of Lemmas 2 and 3 can be easily extended to p-instances. In fact, the only part of its proof which should be adapted is the case when $\mathbf{J} = \mathbf{I}$. In this case, either $\mathsf{Leaves} = \emptyset$ (the reduced tableau for F(IC) is closed) either for each instantiation there exists $A \in \mathcal{X}$ such that $A \notin S(\mathbf{I})$. Let $A = T(X)$ (resp. $F(X)$). Then $F(X)$ or $T(\bullet X)$ (resp. $T(X)$ or $T(\bullet X)$) is in $S(\mathbf{I})$. If it is the case that $T(\bullet X)$ is in $S(\mathbf{I})$ and $A = F(X)$, in order to obtain a closed tableau for $S(\mathbf{I}) \cup F(IC)$, we simply apply the rules of the reduced tableau for F(IC) and the rule (9) to $T(\bullet X)$. If it is the case that $T(\bullet X)$ is in $S(\mathbf{I})$ and $A = T(X)$: from proposition 1(b), we can affirm that $F(\bullet X)$ must also appear in the leaf corresponding to $\mathcal{X}$. The closure of $S(\mathbf{I}) \cup F(IC)$ is then achieved. The other cases are treated as in the proof of Lemma 2.

6 Conclusion and Further Work

In this paper we have introduced the method Repair for the integration of multiple databases where inconsistencies are not eliminated from the integrated database. The method produces a set of repair versions of the integrated database where inconsistencies are kept under control, i.e., one knows exactly what part of the whole information is inconsistent w.r.t. the integrity constraints specified in the local sources. The method is complete, i.e., all possible repairs of the integrated database can be obtained. Besides being an effective procedure for the integration of multiple databases it can also be employed to deal with the dynamics of the integrated database. It is sufficiently general to treat also other situations where inconsistencies may arise in databases, such as when integrity constraints changes during the lifetime of the database. The method can be generalized to treat other classes $\mathcal{C}$ of integrity constraints for which $IC \models_{fin} C$ is equivalent to $IC \models C$, where $IC \subseteq \mathcal{C}$ and $C \in \mathcal{C}$. For classes where this property is not satisfied, the method will not be complete and the p-instances produced would not necessarily verify the minimal distance.

We are currently pursuing the following lines of research : First, the complexity of the repair problem should be investigated and the adaptation of the method to a logic programming environment needs to be considered, by translating our tableau proof system into a resolution method. The logic **LFI1** may be used to specify a query language in a DATALOG style to query paraconsistent databases. It would be interesting to compare our three-valued semantics with the well-founded semantics of DATALOG$^\neg$ and obtain a fixpoint semantics for our paraconsistent query language. Finally, we plan to generalize our method for the creation and management of consistent materialized views (datawarehouses).

Some Related Work. In [10] we have proposed a paraconsistent logical framework having a two-valued semantics and a method for controlling inconsistencies in databases. Nevertheless, this method is not complete and does not guarantee that the instance produced contains minimal changes w.r.t. the original. In [3], a method for consistent answers to queries in relational databases (possibly violating a given set of integrity constraints) has been introduced. For this purpose, this paper introduces a notion of *repair* databases based on the minimal distance from the original database, similar to the one introduced in our approach. Consistent answers are then obtained from the minimally repaired versions of the original database. The class of integrity constraints treated there is the same considered in our approach. Even though the main purpose of our paper is not the problem of consistent query answer, we could point some differences between our approach and the one in [3] concerning repairs : (1) their method does not compute the repair databases; (2) our repairs are more *refined* in the sense that they are closer to the original database than the one in [3]; and (3) our repairs do not eliminate inconsistent information. In [4], the same authors present another method to compute consistent answers in possibly inconsistent databases where repairs are specified in a logic programming formalism. This method can treat a larger class of integrity contraints but does not compute the repairs. In [16, 5, 2], the semantics of integrating possibly inconsistent data is captured by the maximal consistent subsets of the union of the theories specifying the local sources. In [2] and [16], extensions of relational algebra and relational calculus, respectively, are introduced for manipulating and querying inconsistent data and inconsistent information is eliminated from the database. Our approach offers a much proper treatment of the whole question mainly because we do not waste information in the presence of contradiction.

References

1. Abiteboul, S., Hull, R. and Vianu, V.: Foundations of Databases, Addison-Wesley (1995).
2. Agarwal, S., Keller, A. M., Wiederhold, G., Saraswat, K. : Flexible Relation : An Approach for Integrating Data from Multiple, Possibly Inconsistent Databases. *Proceedings ICDE*, 1995.
3. Arenas, M., Bertossi, L., Chomicki, J. : Consistent Query Answers in Inconsistent Databases. In *Proc. of the 18th ACM Symposium on Principles of Database Systems*, June 1999, Philadelphia, USA, pp 68-79.
4. Arenas, M., Bertossi, L., Chomicki, J. : Specifying and Querying Database Repairs using Logic Programs with Exceptions. *In Proc. 4th International Conference on Flexible Query Aswering Systems*, October 2000, Warsaw, Poland, Springer-Verlag.
5. Baral,C., Kraus, S., Minker, J., Subrahmanian, V. S. : Combining knowledge bases consisting of first-order theories. *Computational Intelligence*, 8:45-71, 1992.

6. Blair, H., Subrahmanian, V.S. : Paraconsistent Logic Programming. *Theoretical Computer Science*, 68 : 135-154, 1989. Also in *Proc. Conf. on Foundations of Software Technology and Theoretical Computer Science*, (LNCS 287), 340-360, 1987.
7. Carnielli, W.A. and Marcos, J. : A taxonomy of C-systems To appear in: W. A. Carnielli, M. E. Coniglio, and I. M. L. D'Ottaviano, editors, Paraconsistency: The logical way to the inconsistent. *Proceedings of the II World Congress on Paraconsistency* (WCP2000). Marcel Dekker, 2001.
http://www.cle.unicamp.br/e-prints/abstract_5.htm
8. Carnielli, W.A. and Marcos, J. : Tableau systems for logics of formal inconsistency. In : H.R.Arabnia, editor, *Proceedings of the 2001 International Conference on Artificial Intelligence*, (IC-AI 2001), v. II, p. 848-852. CSREA Press, USA, 2001.
http://logica.rug.ac.be/~joao/tableauxLFIs.ps.zip
9. Carnielli, W.A., Marcos, J., de Amo, S. : Formal Inconsistency and evolutionary databases. To appear in *Logic and Logical Philosophy*, 7/8, 2000.
http://www.cle.unicamp.br/e-prints/abstract_6.htm
10. Carnielli, W.A., de Amo, S. : A logic-based system for controlling inconsistencies in evolutionary databases. *Proc. of the VI Workshop on Logic, Language, Information and Computation*, (WoLLIC 99), Itatiaia, Brazil, 1998, pp.89-101.
11. Carnielli, Walter A.: Many-valued logics and plausible reasoning. *Proceedings of International Symp. on Multiple-valued Logic*, Charlotte, U.S.A.,pp. 328-335 IEEE Computer Society Press(1990)
12. Carnielli, W. A., Lima–Marques, M.: Reasoning under Inconsistent Knowledge. *Journal of Applied Non-classical Logics*, Vol. 2 (1), 1992, pp. 49-79.
13. Carnielli, Walter A.: Systematization of the finite many-valued through the method of tableaux.
14. da Costa, Newton C.A.: On the theory of inconsistent formal system, *Notre Dame Journal of Formal Logic* , v. 11, pp. 497-510 (1974).
15. Dreben, B., Goldburg, W.D. : The Decision Problem : Solvable Classes of Quantificational Formulas. Addison-Wesley, 1979.
16. Dung, P.M. : Integrating Data from Possibly Inconsistency Databases. *International Conference on Cooperative Information Systems*, Brussels, Belgium, 1996.
17. Fagin, R., Ullman, J. D., Vardi, M. : On the semantics of updates in databases. In *2nd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages352-365, 1983.
18. Fagin, R. : Finite Model Theory : a Personal Perspective. *Theoretical Computer Science*, 116 (182):3-31, 1994.
19. Gallo, G., Rago, G. : The Satisfiability problem for the Schöenfinkel-Bernays fragment : Partial Instantiation and Hypergraph Algorithms TR 4/94, Dipartimento di Informatica, Università di Pisa, 1994.
20. Gärdenfors, P. : Knowledge in Flux - Modeling the Dynamics of Epistemic States. MIT Press, 1988.
21. Kifer, M., Lozinskii, E.L : A Logic for Reasoning with Inconsistency. *Journal of Automated Reasoning* 9 : 179-215, 1992.
22. Subrahmanian, V.S. : Amalgamating knowledge bases. *ACM Transactions on Database Systems*, 19(2)-1994.
23. Trakhtenbrot, B.A. : The impossibility of an algorithm for the decision problem for finite domains. (Russian), *Doklady Akademii Nauk*, SSSR (N.S.) 70, pp 569-572, 1950.