Database Repairs and Analytic Tableaux

Leopoldo Bertossi[1] and **Camilla Schwind**[2]

[1] Carleton University
School of Computer Science
Ottawa, Canada K1S 5B6
bertossi@scs.carleton.ca

[2] MAP-CNRS, Ecole d'architecture de Marseille
183, Avenue de Luminy,
13288 Marseille, Cedex 9, France
schwind@map.archi.fr

Abstract. In this article, we characterize in terms of analytic tableaux the repairs of inconsistent relational databases, that is databases that do not satisfy a given set of integrity constraints. For this purpose we provide closing and opening criteria for branches in tableaux that are built for database instances and their integrity constraints. We use the tableaux based characterization as a basis for consistent query answering, that is for retrieving from the database answers to queries that are consistent wrt the integrity constraints.

1 Introduction

The notion of consistent answer to a query posed to an inconsistent database was defined in [1]: A tuple is a consistent answer if it is an answer, in the usual sense, in every possible repair of the inconsistent database. A repair is a new database instance that satisfies the integrity constraints and differs from the original instance by a minimal set of changes wrt set inclusion.

A computational methodology to obtain such consistent answers was also presented in [1]. Nevertheless, it has some limitations in terms of the syntactical form of integrity constraints and queries it can handle. In particular, it does not cover the case of existential queries and constraints.

In classical logic, analytic tableaux [8] are used as a formal deductive system for propositional and predicate logic. Similar in spirit to resolution, but with some important methodological and practical differences [18], they are mainly used for producing formal refutations from a contradictory set of formulas. Starting from a set of formulas, the system produces a tree with formulas in its nodes. The set of formulas is inconsistent whenever all the branches in the tableau can be closed. A branch closes when it contains a formula and its negation.

In this paper we extend the tableaux methodology to deal with a relational database instance plus a set of integrity constraints that the first fails to satisfy. Consequently, both inputs together can be considered as building an inconsistent

set of sentences. In this situation, we give criteria for closing branches in a tableau for a relational database instance.

The technique of "opening tableaux" was introduced in [23] for a solution to the frame problem, and in [35,36] for applying tableaux methods to default logic. In this paper we show how to open tableaux for database instances plus their constraints, and this notion of opening is applied to characterize and represent by means of a tree structure all the repairs of the original database. Finally, we sketch how this representation could be used to retrieve consistent query answers. At least at the theoretical level, the methodology introduced in this paper could be applied to any kind of first order (FO) queries and constraints.

This paper is organized as follows. In section 2, we define our notion of repair of a inconsistent database. Section 3 recalls the definition of analytic tableaux and shows how databases and their repairs can be characterized as openings of closed tableaux. In section 4 we show the relationship between consistent query answering and Winslett's approach to knowledge base update; this allows us to obtain some complexity results for our methodology. Section 5 shows how consistent answers to queries posed to an inconsistent database can be obtained using the analytic tableaux. In section 6 we show the relationship of consistent query answering with minimal entailment, more specifically, in section 6.1, with circumscriptive reasoning. This yields a method for implementing the approach, which is studied in section 6.2.

2 Inconsistent Databases and Repairs

In this paper a database instance is given by a finite set of finite relations on a database schema. A database schema can be represented in logic by a typed first-order language, $\mathcal{L}$, containing a finite set of sorted database predicates and a fixed infinite set of constants D. The language contains a predicate for each database relation and the constants in D correspond to the elements in the database domain, that will be also denoted by D. That is every database instance has an infinite domain D. We also have a set of integrity constraints IC expressed in language $\mathcal{L}$. These are first-order formulas which the database instances are expected to satisfy. In spite of this, there are realistic situations where a database may not satisfy its integrity constraints [1]. If a database instance satisfies IC, we say that it is consistent (wrt IC), otherwise we say it is inconsistent. In any case, we will assume from now on that IC is a consistent set of first order sentences.

A database instance r can be represented by a finite set of ground atoms in the database language, or alternatively, as a Herbrand structure over this language, with Herbrand domain D [27]. In consequence, we can say that a database instance r is consistent, wrt IC, when its corresponding Herbrand structure is a model of IC, and we write $r \models IC$.

The active domain of a database instance r is the set of those elements of D that explicitly appear (in the extensions of the database predicates) in r. The active domain is always finite and we denote it by $Act(r)$. We may also have a set of built-in (or evaluable) predicates, like equality, arithmetical relations, etc.

In this case, we have the language $\mathcal{L}$ possibly extended with these predicates. In all database instances each of these predicates has a fixed and possibly infinite extension. Of course, since we defined database instances as finite sets of ground atoms, we are not considering these built-in atoms as members of database instances.

In database applications, it is usually the case that an inconsistent database[1] has "most" of its data contents still consistent wrt IC and can still provide "consistent answers" to queries posed to it. The notion of consistent answer was defined and analyzed in [1]. This was done on the basis of considering all possible changes to r, in such a way that it becomes a consistent database instance. A consistent answer is an answer that can be retrieved from all those repairs that differ from the original instance in a minimal way.

The notion of minimal change, defined in [1], is based on the notion of minimal distance between models using symmetric set difference Δ of sets of database tuples.

Definition 1. [1] Given databases instances[2] r, r' and r'', we say that r' is closer to r than r'' iff $r\Delta r' \subseteq r\Delta r''$. This is denoted by $r' \leq_r r''$. □

It is easy to see that $\leq_r$ is an order relation. Only database predicates are taken into account for the notion of distance. This is because built-in predicates are not subject to change; and then they have the same extension in all database instances. Now we can define the "repairs" of an inconsistent database instance.

Definition 2. [1]
(a) Given database instances r and r', r' is a *repair* of r, if $r' \models IC$ and r' is a minimal element in the set of instances wrt the order $\leq_r$.
(b) Given a database instance r, a set IC and a first order query $Q(\bar{x})$, we say that a ground tuple $\bar{t}$ is a consistent answer to Q in r wrt IC iff $r' \models Q[\bar{t}]$ for every repair r' of r (wrt IC). □

Example 1. Consider the integrity constraint

$$IC: \ \forall x, y, z(Supply(x, y, z) \ \wedge \ Class(z, T_4) \ \rightarrow \ x = C),$$

stating that C is the only provider of items of class T_4; and the inconsistent database $r = \{Supply(C, D_1, It_1), Supply(D, D_2, It_2), Class(It_1, T_4), Class(It_2, T_4)\}$. We have only two possible (minimal) repairs of the original database instance, namely $r_1 = \{Supply(C, D_1, It_1), Class(It_1, T_4), Class(It_2, T_4)\}$ and $r_2 = \{Supply(C, D_1, It_1), Supply(D, D_2, It_2), Class(It_1, T_4)\}$.

Given the query $Q(x, y, z) : Supply(x, y, z)?$, the tuple (C, D_1, It_1) is a consistent answer because it can be obtained from every repair, but (D, D_2, It_2) is not, because it cannot be retrieved from r_1. □

[1] Sometimes we will simply say "database" instead of "database instance".
[2] We are assuming here and everywhere in the paper that all database instances have the same predicates and domain.

It is possible to prove [1] that for every database instance r and set IC of integrity constraints, there is always a repair r'. If r is already consistent, then r is the only repair. The following lemma, easy to prove, will be useful.

Lemma 1.

1. If $r' \leq_r r''$, then $r \cap r'' \subseteq r \cap r'$.
2. If $r' \subseteq r$, then $r \Delta r' = r \setminus r'$. □

We have given a semantic definition of consistent answer to a query in an inconsistent database. We would like to compute consistent answers, but via computing all possible repairs and checking answers in common in **all** of them. Actually there may be an exponential number of repairs in the size of the database [3].

In [1,11] a mechanism for computing and checking consistent query answers was considered. It does not produce/use the repairs, but it queries the only explicitly available inconsistent database instance. Given a FO query Q, to obtain the consistent answers wrt a finite set of FO ICs IC, Q is qualified with appropriate information derived from the interaction between Q and IC. More precisely, if we want the consistent answers to $Q(\bar{x})$ in r, the query is rewritten into a new query $\mathcal{T}(Q(\bar{x}))$; and then the (ordinary) answers to $\mathcal{T}(Q(\bar{x}))$ are retrieved from r.

Example 2. (example 1 continued) Consider the query $Q : Supply(x, y, z)$? about the items supplied together with their associated information. In order to obtain the consistent answers, the query $\mathcal{T}(Q) : Supply(x, y, z) \wedge (Class(z, T_4) \rightarrow x = C)$ is generated and posed to the original database. The extra conjunct in it is the "residue" obtained from the interaction between the query and the constraint. Residues can be obtained automatically [1]. □

In general, $\mathcal{T}$ is an iterative operator. There are sufficient conditions on queries and ICs for soundness, completeness and termination of operator $\mathcal{T}$; and natural and useful syntactical classes satisfy those conditions. There are some limitations though: $\mathcal{T}$ can not be applied to existential queries like $Q(X) : \exists Y\ Supplies(X, Y, It_1)$?. However, this query does have consistent answers at the semantic level. Furthermore, the methodology presented in [1] assumes that the ICs are (universal) constraints written in clausal form.

There are fundamental reasons for the limitations of the query rewriting approach. If a FO query can be always rewritten into a new FO query, then the problem of *consistent query answering* (CQA) would have polynomial time data complexity. From the results in this paper (see also [13]), we will see that CQA is likely to have a higher computational complexity.

Notice that $\mathcal{T}$ is based on the interaction between the queries and the ICs. It does not consider the interaction between the ICs and the database instance. In this paper we concentrate mostly on this second form of interaction. In particular, we wonder if we can obtain an implicit and compact representation of the database repairs.

Furthermore, the database seen as a set of logical formulas plus IC is an inconsistent first order theory; and we know that such an inconsistency can be detected and represented by means of an analytic tableau.

An analytic tableau is a syntactically generated tree-like structure that, starting from a set of formulas placed at the root, has all its branches "closed" when the initial set of formulas is inconsistent. This tableaux can show us how to repair inconsistencies, because closed branches can be opened by removing literals.

In this work, we show how to generate, close and open tableaux for database instances with their constraints; and we apply the notion of opening to characterize and represent by means of a tree structure all the repairs of the original database. Finally, we sketch how this representation could be used to retrieve consistent query answers. At least at the theoretical level, the methodology introduced here could be applied to any kind of first order queries and constraints.

3 Database Repairs and Analytic Tableaux

In order to use analytic tableaux to represent database repairs and characterize consistent query answers, we need a special form of tableaux, suitable for representing database instances and their integrity constraints.

Given a database instance r and a finite set of integrity constraints IC, we first compute the tableau, $TP(IC \cup r)$, for IC and r. This tableau has as root node the set of formulas $IC \cup r$. This tableau should be closed, that is the tableau has only closed branches, if and only if r is inconsistent. By removing database literals in every closed branch we can transform r into a consistent database instance and thus obtain a repair of the database. For all this to work, we must take into account, when computing the tableau, that r represents a database instance and not just a set of formulas, in particular, that the absence of positive information means negative information, etc. (see section 3.2). Next, we give a brief review of classical first order analytic tableaux [8,37,19].

3.1 Analytic tableaux

The tableau of a set of formulas is obtained by recursively *breaking down* the formulas into subformulas, obtaining sets of sets of formulas. These are the usual Smullyan's classes of formulas:

α	α_1 α_2	β	β_1 β_2
$f \wedge g$	f g	$f \vee g$	f g
$\neg(f \vee g)$	$\neg f$ $\neg g$	$\neg(f \wedge g)$	$\neg f$ $\neg g$
$\neg(f \rightarrow g)$	f $\neg g$	$f \rightarrow g$	$\neg f$ g

γ	$\gamma(p), p$ any constant	δ	$\delta(p), p$ a fresh constant
$(\forall x)f$	$f[x/p]$	$(\exists x)f$	$f[x/p]$
$\neg(\exists x)f$	$\neg f[x/p]$	$\neg(\forall x)f$	$\neg f[x/p]$

A tableaux prover produces a formula tree. An α-rule adds new formulas to branches, a β-rule splits the tableau and adds a new branch. Given a formula φ, we denote by $TP(\varphi)$ the tree produced by the tableaux system. We can think of this tree as the set of its branches, that we usually denote with $X, Y, \ldots$.

Notice that the original set of constants in the language, in our case, D, is extended with a set of new constants, P, the so-called Skolem functions or parameters. These parameters, that we will denote by $p, p_1, \ldots$, have to be new at the point of their introduction in the tree in the sense that they have not appeared so far in the (same branch of the) tableau. When applying the γ-rule, the parameter can be any of the old or new constants.

A tableau branch is *closed* if it contains a formula and its negation, otherwise it is *open*. Every open branch corresponds to a model of the formula: If a branch $B \in TP(\varphi)$ is open and finished, then the set of ground atoms on B is a model of φ. If the set of initial formulas is inconsistent, it does not have models, and then all branches (and thus the tableau) have to be closed. Actually, the completeness theorem for tableaux theorem proving [37] states that: F is a theorem iff $TP(\{\neg F\})$ is closed.

The intuitive idea of finished branch, of one to which no tableaux rule can be applied obtaining something new and relevant, is captured by means of the notion of *saturated branch*: this is a branch where all possible rules have been applied.

Definition 3. A branch B is *saturated* iff it satisfies

1. If $\neg\neg\varphi \in B$, then $\varphi \in B$
2. If $(\varphi \vee \psi) \in B$, then $\varphi \in B$ or $\psi \in B$
3. If $(\varphi \wedge \psi) \in B$, then $\varphi \in B$ and $\psi \in B$
4. If $\exists x\varphi \in B$, then $\varphi[c] \in B$ for some constant c
5. If $\forall x\varphi \in B$, then $\varphi[c] \in B$ for any constant c.[3] □

A branch is called *Hintikka* if it is saturated and not closed [19]. It is easy to see that a saturated branch is Hintikka iff it does not contain any atomic formula A and its negation $\neg A$. From now on, tableaux branches will be assumed to be saturated. Nevertheless, sometimes we talk about branches even when they are partially developed only.

We consider TP not only as a theorem prover (or consistency checker) for formulae but also as an application from (sets of) formulas to trees which has some useful properties. Thus, operations on tableaux can be defined on the basis of the logical connectives occurring inside the formulas involved.

Lemma 2. Let φ and ψ be any formulae. Then TP has the following properties.

1. $TP(\{\varphi \vee \psi\}) = TP(\{\varphi\}) \cup TP(\{\psi\})$
2. $TP(\{\varphi \wedge \psi\}) = \{X \cup Y : X \in TP(\{\varphi\})$ and $Y \in TP(\{\psi\})\}$

[3] If the language had function symbols, we would have replace constants by ground terms in this definition.

3. If $B \in TP(\varphi \wedge \psi)$ then $B = B' \cup B''$ and $B' \in TP(\varphi)$ and $B'' \in TP(\psi)$. □

Property 3. follows directly from properties 1. and 2. The properties in the lemma motivate the following definition.

Definition 4. Given tableaux T and T', each of them identified with the set of its branches, the combined tableaux is $T \otimes T' = \{X \cup Y : X \in T \text{ and } Y \in T'\}$. □

Remark 1. The properties in lemma 2 can be used to check whether a formula φ derives from a theory A. $A \models \varphi$ iff $(A \rightarrow \varphi)$ is a theorem, what will be proved if we derive a contradiction from assuming $\neg(A \rightarrow \varphi)$. Therefore we will have to compute $TP(\{\neg(A \rightarrow \varphi)\})$ and check for closure. Using the second property, we will check $TP(\{A\}) \otimes TP(\{\neg\varphi\})$ for closure, allowing us to compute $TP(A)$ only once for any number of requests. □

The following relationship between the open branches of the tableaux for a formula and the its models has been shown, among others by [7,36].

Theorem 1. Let $B \in TP(\{\phi\})$ be an open branch of the tableau for ϕ. Then there is a model M of ϕ, which satisfies B, i.e. $B \subseteq M$. More precisely, there is Herbrand model of φ such that the ground atoms in B belong to M. □

3.2 Representing database instances by tableaux

In database theory, we usually make the following assumptions[4]: (a) Unique Names Assumption (UNA): If a and b are different constants in D, then $a \neq b$ holds in r. (b) Closed World Assumption (CWA): If r is a database instance, then for any ground database atom $P(c)$, if $P(c) \notin r$, then $\neg P(c)$ holds for r, more precisely, implicitly $\neg P(c)$ belongs to r.

In consequence, if we see the relational database as the set of its explicit atoms plus its implicit negative atoms, we can always repair the database by removing ground database literals.

When computing a tableau for a database instance r, we do not add explicitly the formulas corresponding to the UNA and CWA, rather we keep them implicit, but taking them into account when computing the tableau. This means, for example, that the presence on a tableau branch of a formula $a = b$, for different constants a, b in D, closes the branch.

Given a database r and integrity constraints IC, we will generate the tableau $TP(IC \cup r)$. Notice that every branch B of this tableau will be of the form $I \cup r$, where $I \in TP(IC)$ (see lemma 2). I is the "IC-part" of the branch.

Notice also that a tableau for IC only will never be closed, because IC is consistent. The same happens with any tableau for r. Only the combination of r and IC may produce a closed tableau.

[4] Actually, it is possible to make all these assumptions explicit and transform the database instance into a first-order theory [33].

$TP(IC \cup r)$ is defined as in section 3.1, but we still have to define the closure conditions for tableaux associated to database instances. Before, we present some motivating examples.

Example 3. (example 1 continued) In this case, $TP(IC \cup r)$ is the tree in figure 1. The last branch is closed because $D = C$ is false in the database (alternatively,

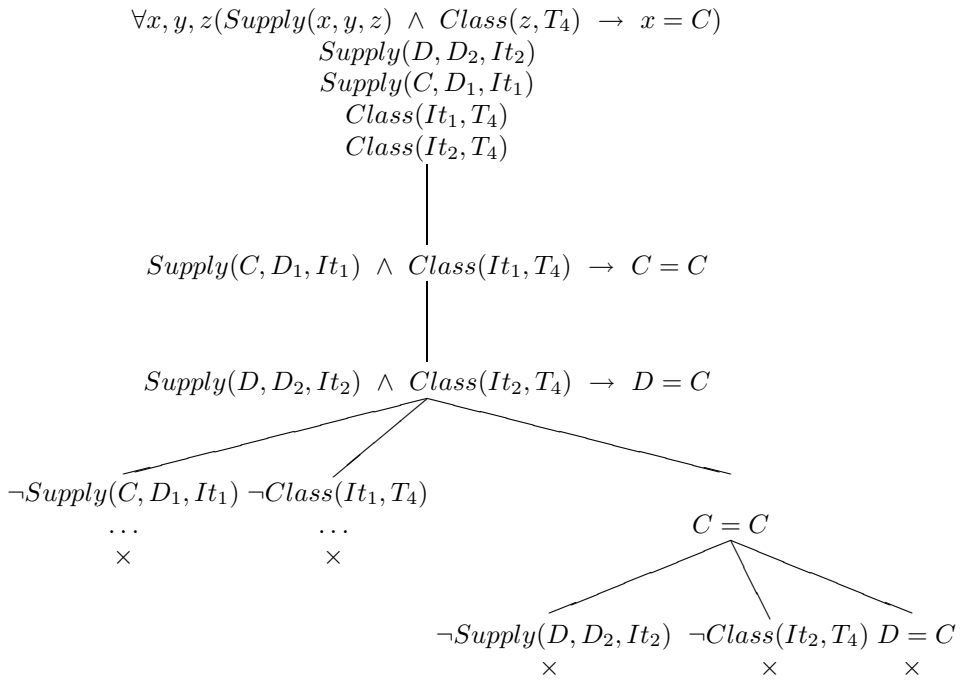
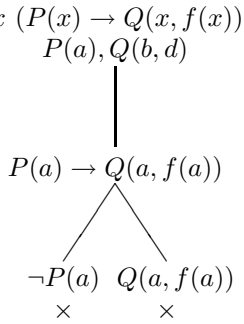


Fig. 1. Tableau for Example 3

because $D \neq C$ is implicitly in the database). We can see that $TP(IC \cup r)$ is closed. r is inconsistent wrt IC. The nodes $(Supply(C, D_1, It_1) \wedge Class(It_1, T_4) \rightarrow C = C)$ and $(Supply(D, D_2, It_2) \wedge Class(It_2, T_4) \rightarrow D = C)$ are obtained by applying the γ-rule to $\forall x, y, z(Supply(x, y, z) \wedge Class(z, T_4) \rightarrow x = C)$. Application of the β-rule to $(Supply(D, D_2, It_2) \wedge Class(It_2, T_4) \rightarrow D = C)$ produces the same subtree for all three leaves: $\neg Supply(C, D_1, It_1)$, $\neg Class(It_1, T_4)$ and $C = C$. In the figure, we indicate this subtree by "...". We will see later (see section 3.3) that, in some cases, we can omit the development of subtrees that should develop under branches that are already closed. Here we can omit the explicit further development of the subtree from the first two leftmost branches, because these branches are already closed. □

In tableaux with equality, we need extra rules. We will assume that we can always introduce equalities of the form $t = t$, for a term t, and that we can replace a term t in a predicate P by t' whenever $t = t'$ belongs to the same tableau branch (paramodulation, [19]). It will be simpler to define the closure rules for database tableaux, if we skolemize existential formulas before developing the tableau [18]. We assume from now on that all integrity constraints are skolemized by means of a set of Skolem constants (the parameters in P) and new function symbols.

Example 4. Consider the referential $IC : \ \forall x \ (P(x) \rightarrow \exists y \ Q(x, y))$, and the inconsistent database instance $r = \{P(a), Q(b, d)\}$, for $a, b, c \in D$. With an initial skolemization, we can develop the following tableau $TP(IC \cup r)$. In this tableau, the second branch closes because $Q(a, f(a))$ does not belong to the database instance. There is no x in the active database domain, such that r contains $Q(a, x)$. Implicitly, by the CWA, r contains then $\neg Q(a, x)$ for any x. Hence the branch containing $Q(a, f(a))$ closes and r is inconsistent for IC.



Example 5. Consider the inconsistent database $r_1 = \{Q(a), Q(b)\}$ wrt the IC: $\exists x \ P(x)$. After having skolemized $\exists x \ P(x)$ into $P(p)$, a tableau proof for the inconsistency is the following

$$\begin{array}{c} P(p) \\ Q(a), Q(b) \\ \times \end{array}$$

This branch closes because there is no x in D such that $P(x) \in r$ and therefore $\neg P(x)$ belongs to r for any x in D. $P(p)$ cannot belong to this database. □

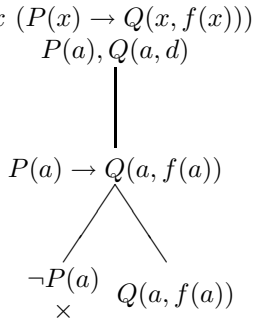
Example 6. Let us now change the database instance in example 5 to $r_2 = \{P(a), P(b)\}$, keeping the integrity constraint. Now, the database is consistent, and we have the following tableau $TP(IC \cup r_2)$:

$$\begin{array}{c} P(p) \\ P(a), P(b) \end{array}$$

This time we do not want the tableau to close, and thus sanctioning the inconsistency of the database. The reason is that we could make p take any of the values in the active domain $\{a, b\} \subseteq D$ of the database. □

A similar situation can be found in a modified version of example 4.

Example 7. Change the database instance in example 4 to $\{P(a), Q(a,d)\}$. Now it is consistent wrt the same IC. We obtain



Now we do not close the rightmost branch because we may define f as a function from the active domain into itself that makes $Q(a, f(a))$ become a member of the database, actually by defining $f(a) = d$. □

Example 8. Consider $IC: \exists x\ \neg P(x)$ and the consistent database instance $r = \{P(a)\}$. The tableau $TP(IC \cup r)$ after skolemization of IC is:

$$\begin{array}{c} \neg P(p) \\ P(a) \end{array}$$

This tableau cannot be closed, because p must be a new parameter, not occurring in the same branch of the tableau and it is not the case that $P(p) \in r$ (alternatively, we may think of p as a constant that can be defined as any element in $D \setminus \{a\}$, that is in the complement of the active domain of the database). □

In general, a tableau branch closes whenever it contains a formula and its negation. However, in our case, it is necessary to take into account that not all literals are explicit on branches due to the UNA and CWA. The following definition of closed branch modifies the standard definition, and considers those assumptions.

Definition 5. Let B be a tableau branch for a database instance r with integrity constraints IC, say $B = I \cup r$. B is closed iff one of the following conditions holds:

1. $a = b \in B$ for different constants a, b in D.
2. (a) $P(\bar{c}) \in I$ and $P(\bar{c}) \notin r$, for a ground tuple $\bar{c}$ containing elements of D only.
   (b) $P(\bar{c}) \in I$ and there is no substitution σ for the parameters in $\bar{c}$ such that $P(\bar{c})\sigma \in r$.[5]

[5] A substitution is given as a pair $\sigma = (p, t)$, where p is a variable (parameter) and t is a term. The result of applying σ to formula F, noted $F\sigma$, is the formula obtained by replacing every occurrence of p in F by t.

3. $\neg P(\bar{c}) \in I$ and $P(\bar{c}) \in r$ for a ground tuple $\bar{c}$ containing elements of D only.
4. $\varphi \in B$ and $\neg\varphi \in B$, for an arbitrary formula φ.
5. $\neg t = t \ \in B$ for any term t. □

Condition 1. takes UNA into account. Notice that it is restricted to database constants, so that it does not apply to new parameters[6]. Condition 2(a) takes CWA into account. Alternative condition 2(b) (actually it subsumes 2(a)) gives an account of examples 4, 5, 6, and 7.

In condition 3. one might miss a second alternative as in condition 2., something like "$\neg P(\bar{c}) \in I$ for a ground tuple containing Skolem symbols, when there is no way to define them considering elements of $D \setminus Act(r)$ in such a way that $P(\bar{c}) \notin r$". This condition can be never satisfied because we have an infinite database domain D, but a finite active domain $Act(r)$. So, it will never apply. This gives an account of example 8. Conditions 4. and 5. are the usual closure conditions. Conditions 2(a) and 3. are special cases of 4.

Now we can state the main properties of tableaux for database instances and their integrity constraints.

Proposition 1. For a database instance r and integrity constraints IC, it holds:

1. r is inconsistent wrt to IC iff the tableau $TP(IC \cup r)$ is closed (i.e. each of its branches is closed).
2. $TP(IC \cup r)$ is closed iff r does not satisfy IC (i.e. $r \not\models IC$). □

3.3 Opening tableaux

The inconsistency of a database r wrt IC is characterized by a tableau $TP(IC \cup r)$ which has only closed branches. In order to obtain a repair of r, we may remove the literals in the branches which are "responsible" for the inconsistencies, even implicit literals corresponding to the CWA. Every branch which can be "opened" in this way will possibly yield a repair. We can only repair inconsistencies due to literals in r. We cannot remove literals in I because, according to our approach, integrity constraints are rigid, we are not willing to give them up; we only allow changes in the database instances. We cannot suppress equalities $a = b$ neither built-in predicates.

Remark 2. According to Definition 5, we can repair inconsistencies due only to cases 2. and 3. More precisely, given a closed branch B in $TP(IC \cup r)$:

1. If B is closed because of the CWA, it can be opened by inserting $\sigma P(\bar{c})$ into r, or, equivalently, by removing the implicit literal $\neg\sigma P(\bar{c})$ from r for any substitution σ from the parameters into D (case 2(b) in Def. 5).
2. If B is closed because of contradictory literals $\neg P(\bar{c}) \in I$ and $P(\bar{c}) \in r$, then it can be opened by removing $P(\bar{c})$ from r (case 3 in Def. 5) . □

[6] That is, elements of P are treated as null values in Reiter's logical reconstruction of relational databases [33].

Example 9. (example 3 continued) The tableau has 9 closed branches: (we display the literals within the branches only)

B_1	B_2	B_3
$Supply(C, D_1, It_1)$	$Supply(C, D_1, It_1)$	$Supply(C, D_1, It_1)$
$Supply(D, D_2, It_2)$	$Supply(D, D_2, It_2)$	$Supply(D, D_2, It_2)$
$Class(It_1, T_4)$	$Class(It_1, T_4)$	$Class(It_1, T_4)$
$Class(It_2, T_4)$	$Class(It_2, T_4)$	$Class(It_2, T_4)$
$\neg Supply(C, D_1, It_1)$	$\neg Supply(C, D_1, It_1)$	$\neg Supply(C, D_1, It_1)$
$\neg Supply(D, D_2, It_2)$	$\neg Class(It_2, T_4)$	$D = C$

B_4	B_5	B_6
$Supply(C, D_1, It_1)$	$Supply(C, D_1, It_1)$	$Supply(C, D_1, It_1)$
$Supply(D, D_2, It_2)$	$Supply(D, D_2, It_2)$	$Supply(D, D_2, It_2)$
$Class(It_1, T_4)$	$Class(It_1, T_4)$	$Class(It_1, T_4)$
$Class(It_2, T_4)$	$Class(It_2, T_4)$	$Class(It_2, T_4)$
$\neg Class(It_1, T_4)$	$\neg Class(It_1, T_4)$	$\neg Class(It_1, T_4)$
$\neg Supply(D, D_2, It_2)$	$\neg Class(It_2, T_4)$	$D = C$

B_7	B_8	B_9
$Supply(C, D_1, It_1)$	$Supply(C, D_1, It_1)$	$Supply(C, D_1, It_1)$
$Supply(D, D_2, It_2)$	$Supply(D, D_2, It_2)$	$Supply(D, D_2, It_2)$
$Class(It_1, T_4)$	$Class(It_1, T_4)$	$Class(It_1, T_4)$
$Class(It_2, T_4)$	$Class(It_2, T_4)$	$Class(It_2, T_4)$
$C = C$	$C = C$	$C = C$
$\neg Supply(D, D_2, It_2)$	$\neg Class(It_2, T_4)$	$D = C$

The first four tuples in every branch correspond to the initial instance r. Each branch B_i consists of an I-part and the r-part, say $B_i = r \cup I_i$. And we have

I_1	I_2	I_3
$\neg Supply(C, D_1, It_1)$	$\neg Supply(C, D_1, It_1)$	$\neg Supply(C, D_1, It_1)$
$\neg Supply(D, D_2, It_2)$	$\neg Class(It_2, T_4)$	$D = C$

I_4	I_5	I_6
$\neg Class(It_1, T_4)$	$\neg Class(It_1, T_4)$	$\neg Class(It_1, T_4)$
$\neg Supply(D, D_2, It_2)$	$\neg Class(It_2, T_4)$	$D = C$

I_7	I_8	I_9
$C = C$	$C = C$	$C = C$
$\neg Supply(D, D_2, It_2)$	$\neg Class(It_2, T_4)$	$D = C$

In order to open this closed tableau, we can remove literals in the closed branches. Since a tableau is open whenever it has an open branch, each opened branch of the closed tableau might produce one possible transformed open tableau. Since we want to modify the database r, which should become con-

sistent, we should try to remove a minimal set of literals in the r-part of the branches in order to open the tableau. This automatically excludes branches B_3, B_6 and B_9, because they close due to the literals $D = C$, which do not correspond to database literals, but come from the constraints.

In this example we observe that the sets of database literals of some of the I_j are included in others. Let us denote by I'_j the set of literals in I_j that are database literals (i.e. not built-in literals), e.g. $I'_1 = I_1, I'_7 = \{\neg Supply(D, D_2, It_2)\}$. We have then $I'_1 \supset I'_7$, $I'_2 \supset I'_8$, $I'_3 \supset I'_9$, $I'_4 \supset I'_7$, $I'_5 \supset I'_8$, $I'_6 \supset I'_9$. This shows, for example, that in order to open B_1, we have to remove from r a superset of the set of literals that have to be removed from r for opening B_7. Hence, we can decide that the branches whose database part contains the database part of another branch can be ignored because they will not produce any (minimal) repairs. This allows us not to consider B_1 through B_6 in our example, and B_7 and B_8 are the only branches that can lead us to repairs. □

The following lemma tells us that we can ignore branches with subsumed I-parts, because those branches cannot become repairs.

Lemma 3. If $r'' \subseteq r' \subseteq r$, then $r' \leq_r r''$. □

Moreover, as illustrated above, where the tableau tree is shown, sometimes we can detect possible subsuming branches without fully developing the tableau. In example 3 the first formula has been split by a tableau rule and we have already closed two branches. When we apply another rule, we know then, that the branch $C = C$, which is not closed yet, will be not be closed or will be closed by a subset of the database literals appearing in the first two branches.

Definition 6. Let $B = I \cup r$ be a closed branch of the tableau $TP(IC \cup r)$.

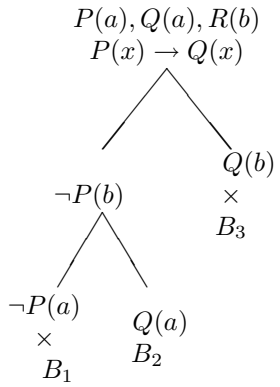
(a) If I is not closed, i.e the branch is closed due to database literals only, we say that B is *data closed*.
(b) Let $B = I \cup r$ be a data closed branch in the tableau $TP(IC \cup r)$, we define $op(B) := (r \setminus L(B)) \cup K(B)$, where
   1. $L(B) = \{l \mid l \in r \text{ and } \neg l \in I\}$
   2. $K(B) = \tau\{l \mid l \text{ is a ground atom in} I \text{ and there is no substitution } \sigma \text{ such that } l\sigma \in r\}$, where τ is any substitution of the parameters into D.
(c) An instance r' is called an *opening* of r iff $r' = op(B)$ for a data closed branch B in $TP(IC \cup r)$. □

If the branch B is clear from the context, we simply write $r' = (r \setminus L) \cup K$. If no parameters have been introduced in the branch, then we do not need to consider the substitutions above. In this case, for an opening $I \cup r'$ of a branch $I \cup r$ it holds:(a) If $P(\bar{c}) \in I$ and $P(\bar{c}) \notin r$, then $P(\bar{c}) \in r'$. (b) If $\neg P(\bar{c}) \in I$ and $P(\bar{c}) \in r$, then $P(\bar{c}) \notin r'$. Notice that we only open branches which are closed because of conflicting database literals.

When $r \models IC$, then $TP(IC \cup r)$ will have (finished) open branches B. For any of those branches $op(B)$ can be defined exactly as in Definition 6. It is easy to verify that in this case $op(B)$ coincides with the original instance r.

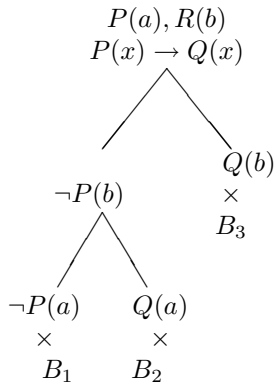
Proposition 2. Let r' be an opening of r. Then r' is consistent with IC, i.e. $r' \models IC$. □

Example 10. Consider $r = \{P(a), Q(a), R(b)\}$ and $IC = \{\forall x(P(x) \rightarrow Q(x)\}$. Here $r \models IC$ and $TP(IC \cup r)$ is



The first branch, B_1, is closed and $op(B_1) = \{Q(a), R(b)\}$ that satisfies IC. The second branch, B_2, is open and $op(B_2) = r$. The third branch, B_3, is closed and $op(B_3) = \{P(a), Q(a), Q(b), R(b)\}$ that satisfies IC. Notice that we could further develop the last node there, obtaining the same tree that is hanging from $\neg P(b)$ in the tree on the LHS. If we do this, we obtain closed branches B_4, B_5, with $op(B_4) = \{Q(a), Q(b), R(b)\}$, and $op(B_5) = \{P(a), Q(a), Q(b), R(b)\}$. With these last two openings we do not get any closer to r than with $op(B_3)$, that is still not as close to r as the only repair, r, obtained with branch B_2. □

Example 11. Consider IC as in example 10, but now $r = \{P(a), R(b)\}$, that does not satisfy IC. $TP(IC \cup r)$ is



For the first branch B_1, we obtain $op(B_1) = \{R(b)\}$, that is a repair. Branch B_2 gives $op(B_2) = \{P(a), R(b), Q(a)\}$, the other repair.

For the closed branch B_3 we have $op(B_3) = \{P(a), Q(b), R(b)\}$. This is not a model of IC, apparently contradicting Proposition 2, in particular, it is not a

repair of r. If we keep developing node $Q(b)$ exactly as $\neg P(b)$ on the LHS, we obtain extended (closed) branches, with associated instances $\{Q(b), R(b)\}$ and $\{P(a), Q(a), Q(b), R(b)\}$. Both of them satisfy IC, but are non minimal; and then they are not repairs of r. This example shows the importance of having the open and closed branches (maybe not explicitly) saturated (see Definition 3). □

We can see that every opening is related to a possibly non minimal repair of the original database instance[7]. For repairs, we are only interested in "minimally" opened branches, i.e. in open branches which are as close as possible to r. In consequence, we may define a *minimal opening* r' as an opening such that $r \Delta r'$ is minimal under set inclusion.

Openings of r are obtained by deletion of literals from r, or, equivalently, by deletion/insertion of atoms from/into r. In order to obtain minimal repairs, we have to make a minimal set of changes, therefore we do not keep openings associated to an r'', such that $r' \Delta r \subsetneq r'' \Delta r$, where r' is associated to another opening. We will show subsequently that these are the openings where L and K are minimal in the sense of set inclusion wrt all other openings in the same tree.

The following theorem establishes a relationship between the order of repairs defined in Definition 1 and the set inclusion of the database atoms that have been inserted or deleted when opening a database instance.

Lemma 4. For any opening $r' = (r \setminus L) \cup K$, we have $r \Delta r' = L \cup K$. □

Proposition 3. Let $r_1 = (r \setminus L_1) \cup K_1$ and $r_2 = (r \setminus L_2) \cup K_2$. Then r_1 is closer to r than r_2, i.e. $r_1 \leq_r r_2$ iff $L_1 \subseteq L_2$ and $K_1 \subseteq K_2$. □

Theorem 2. Let r be an inconsistent database wrt IC. Then r' is a repair of r iff there is an open branch I of $TP(IC)$, such that $I \cup r$ is closed and $I \cup r'$ is a minimal opening of $I \cup r$ in $TP(IC \cup r)$. □

Example 12. (example 9 continued) $TP(IC \cup r)$ has two minimal openings:

r'_7	r'_8
$Supply(C, D_1, I_1)$	$Supply(C, D_1, I_1)$
$Class(I_1, T_4)$	$Class(I_1, T_4)$
$Class(I_2, T_4)$	$Supply(D, D_2, I_2)$

The rightmost closed branch cannot be opened because it is closed by the atom $D = C$ which is not a database predicate. □

[7] Strictly speaking, we should not say "non minimal repair", because repairs are minimal by definition. Instead, we should talk of database instances that differ from the original one and satisfy the ICs. In any case, we think there should be no confusion if we relax the language in this sense.

4 Repairs, Knowledge Base Updates and Complexity

Our definition of repairs is based on a minimal distance function as used by Winslett for knowledge base update [39]. More precisely, Winslett in her "possible models approach" defines the knowledge base change operator $\circ$ for the update of a propositional knowledge base K by a propositional formula p by

$$Mod(K \circ p) = \bigcup_{m \in Mod(K)} \{m' \in Mod(p) : m \triangle m' \in Min_{\subseteq}(\{m \triangle m' : m' \in Mod(p)\})\}$$

In [16], Eiter and Gottlob present complexity results for propositional knowledge base revision and update. According to these results, Winslett's update operator is on the second level of the polynomial hierarchy in the general case (i.e. without any syntactic restriction on the propositional formulas): the problem of deciding whether a formula q is a logical consequence of the update by p of a knowledge base T is Π_2^P-complete.

Update	General case	General case	Horn	Horn
	arbitrary p	$\| p \| \leq k$	arbitrary p	$\| p \| \leq k$
$T \circ p \rightarrow q$	Π_2^P-complete	co-NP-complete	co-NP-complete	$O(\| T \| \cdot \| q \|)$

In the above table, we resume the results reported in [16]. The table contains five columns. In the general case (columns two and three), T is a general propositional knowledge base. In the Horn-case (columns four and five), it is assumed that p and q and all formulas in T are conjunctions of Horn-clauses. Columns two and four account for cases where no bound is imposed on the length of the update formula p, while columns three and five describe the case where the length of p is bounded by a constant k. The table illustrates that the general problem in the worst case (arbitrary propositional formulas without bound on the size) is intractable, whereas it becomes very well tractable (linear in the size of T and query q) in the case of Horn formulas with bounded size.

How are these results related to CQA? If r is a database which is inconsistent with respect to the set of integrity constraints IC, the derivation of a consistent answer to a query Q from r corresponds to the derivation of Q from the data base r updated by the integrity constraints IC. Hence, the (inconsistent) knowledge base instance r, which is just a conjunction of literals, corresponds to the propositional knowledge base T. The integrity constraints IC correspond to the update formula p And deriving an answer to query Q from r (and IC) corresponds to the derivation of Q from r updated by IC.

Update is defined for propositional formulas. Update is defined by means of models of the knowledge base r and the update formula IC. In our case, r is a finite conjunction of grounded literals, i.e. r is a propositional Horn formula. The update formulas however (integrity constraints IC) are FO formulas. However, the Herbrand universe of the database is a finite set of constants. Therefore, we can consider instead of IC the finite set of instantiations of the formulas in IC by database constants. Let us denote the conjunction of these instantiations by

ic. Note that ic is Horn whenever all formulas in IC are Horn, what is common for database ICs.

It is then easy to see that the following relationship holds between update and repairs and CQA. It follows straightforwardly from the definitions of repairs and update.

Theorem 3. Given a database instance r and a set of integrity constraints IC with their propositional database representation ic:
(a) r' is a repair of r wrt IC iff $r' \in Mod(r \circ ic)$.
(b) If Q is a query, $\bar{t}$ is a consistent answer to Q wrt IC iff every model of $r \circ ic$ is a model of $Q(\bar{t})$, i.e. $Mod(r \circ ic) \subseteq Mod(Q(\bar{t}))$. □

In consequence, the results given by Eiter and Gottlob apply directly to CQA.

The number of branches of a fully developed tableaux is very high: in the worst case, it contains $o(2^n)$ branches where n is the length of the formula. Moreover, we have to find minimal elements within this exponential set, what increases the complexity. Theorem 3 tells us that we do not need to compare the entire branches but only parts of them, namely the literals which have been removed in order to open the tableau. This reduces the size of the sets we have to compare, but not their number. Let us reconsider in example 3 the point just before applying the tableaux rule which develops formula $Supply(D, D_2, I_2) \wedge Class(I_2, T_4) \rightarrow D = C$. As we pointed out in the discussion of example 3, under some conditions, it is possible to avoid the development of closed branches because we know in advance, without developing them, that they will not be minimal.

Example 13. (example 3 continued) In this case, $TP(IC \cup r)$ is the tree in Figure 2. This tree has two closed branches, B_1 and B_2, and one open branch B_3. Each of these branches will receive an identical subtree due to the application of the tableaux rules to the formulas not yet developed on the tree, namely $(Supply(D, D_2, It_2) \wedge Class(It_2, T_4) \rightarrow D = C)$. We know at this stage of the development that B_1 is closed due to $\neg Supply(C, D_1, It_1)$ and B_2 is closed due to $\neg Class(It_1, T_4)$; B_3 is not closed. □

In this example, we can see that if we further develop the tree, every B_i will have the same sets of sub-branches, say L_1, L_2, ..., where L_i is a set of literals. The final fully developed tableau will then consist of the branches $B_1 \cup L_1$, $B_1 \cup L_2$, ..., $B_2 \cup L_1$, $B_2 \cup L_2$, ... $B_3 \cup L_1$, $B_3 \cup L_2$, ..., ... If the final tableau is closed, since B_3 is not closed, every $B_3 \cup L_j$ will be closed due to literals within L_j, say K_j.

We have then two cases: either the literals in K_j close due to literals in r (which is the original inconsistent database instance) or they close due to literals in the part of B_3 not in r. In the first case, these literals from K_j will close every branch of the tree (also B_1 and B_2). Since B_1 and B_2 were already closed, they will be closed due to a set of literals that is strictly bigger than before, and therefore they will not produce minimally closed branches (and no repairs). In this situation, those branches can immediately be ignored and not further

developed. This can considerably reduce the size of the tableau. In this example, at the end of the development, only B_3 will produce repairs (see example 3).

In the second case, the literals in K_j close due to literals in the part of B_3 that are not in r. If these literals are not database literals (we have called them built-in predicates), the branch cannot be opened, we cannot repair inconsistencies that are not due to database instances. Then, we only have to consider the case of database literals that are not in r.

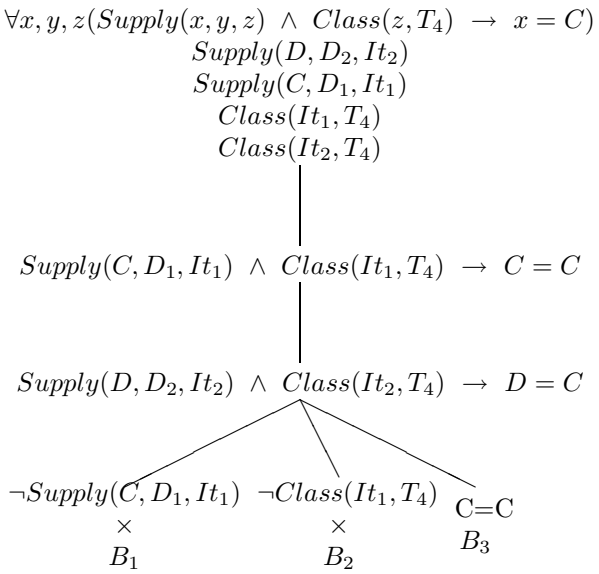


Fig. 2.

Since B_3 is open, those literals are negative literals (in the other case, B_3 would not have been open, due to condition 2. in Definition 5). This is the only situation where the sub-branches which are closed at a previous point of development may still become minimal. In consequence, a reasonable heuristics will be to suspend the explicit development of already closed branches unless we are sure that this case will not occur.

5 Consistent Query Answering

In order to determine consistent answers to queries, we can also use, at least at the theoretical level, a tableaux theorem prover to produce $TP(IC \cup r)$ and its openings. Let us denote by $op(TP(IC \cup r))$ the tableau $TP(IC \cup r)$, with

its minimal openings: All branches which cannot be opened or which cannot be minimally opened are pruned and all branches which can be minimally opened are kept (and opened). (We reconsider this pruning process in section 6.2.)

According to Definition 2 and Theorem 2, $\bar{t}$ is a consistent answer to the open query $Q(\bar{x})$ when the combined tableau $op(\,TP(IC \cup r)) \otimes TP(\neg Q(\bar{t}))$ (c.f. Definition 4) is, again, a closed tableau. In consequence, we might use the tableau $op(\,TP(IC \cup r)) \otimes TP(\neg Q(\bar{x}))$ in order to retrieve those values for $\bar{x}$ that restore the closure of all the opened branches in the tableau.

Example 14. Consider the functional dependency

$$IC: \quad \forall(x, y, z, u, v)(Student(x, y, z) \wedge Student(x, u, v) \;\rightarrow\; y = u \wedge z = v);$$

and the inconsistent students database instance

$$r = \{Student(S_1, N_1, D_1), Student(S_1, N_2, D_1), Course(S_1, C_1, G_1), \\ Course(S_1, C_2, G_2)\},$$

which has the two repairs, namely

$$r_1 = \{Student(S_1, N_1, D_1), Course(S_1, C_1, G_1), Course(S_1, C_2, G_2)\},$$

$$r_2 = \{Student(S_1, N_2, D_1), Course(S_1, C_1, G_1), Course(S_1, C_2, G_2)\}.$$

We can distinguish two kinds of queries. The first one corresponds to a first order formula containing free variables (not quantified), and then expects a (set of database) tuple(s) as answer. For example, we want the consistent answers to the query "$Course(x, y, z)$?". Here we have that $op(\,TP(IC \cup r)) \otimes TP(\neg Course(x, y, z))$ is closed for the tuples (S_1, C_1, G_1) and (S_1, C_2, G_2).

A second kind of queries corresponds to queries without free variables, i.e. to sentences. They should get the answer "yes" or "no". For example, consider the query "$Course(S_1, C_2, G_2)$?". Here $op(\,TP(IC \cup r)) \otimes TP(\,\neg Course(S_1, C_2, G_2))$ is closed. The answer is "yes", meaning that the sentence is true in all repairs.

Now, consider the query "$Student(S_1, N_2, D_1)$?". The tableau $op(\,TP(IC \cup r)) \otimes TP(\neg Student(S_1, N_2, D_1))$ is not closed, and $Student(S_1, N_2, D_1)$ is not a member of both repairs. The answer is "no", meaning that the query is not true in all repairs. □

The following example shows that, as opposed to [1], we are able to treat existential queries in a proper way.

Example 15. Consider the query "$\exists x Course(x, C_2, G_2)$?" for the database in example 14. Here we have that $op(\,TP(IC \cup r)) \otimes TP(\neg \exists x\; Course(x, C_2, G_2))$ is closed. The second tableau introduces the formulas $\neg Course(p, C_2, G_2)$, for every $c \in D \cup P$ in every branch. The answer is "yes". This answer has been obtained by replacing p by the same constant S_1 in both branches. This does not need to be always the case. For example, with the query "$\exists x\; Student(S_1, x, D_1)$?", that introduces the formulas $\neg Student(S_1, p, D_1)$ in every branch of $op(\,TP(IC \cup$

$r)) \otimes TP(\neg\exists x\ Student(S_1, x, D_1))$, the tableau closes, the answer is "yes", but one repair has been closed for $p = N_1$ and the other repair has been closed for $p = N_2$.

We can also handle open existential queries. Consider now the query with y as the free variable "$\exists z Course(S_1, y, z)$?". The tableaux for $op(TP(IC \cup r)) \otimes TP(\neg\exists z Course(S_1, y, z))$, which introduces the formulas $\neg Course(S_1, y, p)$ in every branch, is closed, actually by $y = C_1$, and also by $y = C_2$, but for two different values for p, namely G_1 and G_2, resp. □

Theorem 4. Let r be an inconsistent database wrt to the set of integrity constraints IC.

1. Let $Q(\bar{x})$ be an open query with the free variables $\bar{x}$. A ground tuple $\bar{t}$ is a consistent answer to $Q(\bar{x})$ iff $op(TP(IC \cup r)) \otimes TP(\neg Q(\bar{x}))$ is closed for the substitution $\bar{x} \mapsto \bar{t}$.
2. Let Q be query without free variables. The answer is "yes", meaning that the query is true in all repairs, iff $op(TP(IC \cup r)) \otimes TP(\neg Q)$ is closed.

6 CQA, Minimal Entailment and Tableaux

As the following example shows, CQA is a form of *non-monotonic entailment*, i.e. given a relational database instance r, a set of ICs IC, and a consistent answer $P(\bar{a})$ wrt IC, i.e. $r \models_c \varphi$, it may be the case that $r' \not\models_c P(\bar{a})$, for an instance r' that extends r.

Example 16. The database containing the table

Employee	*Name*	*Salary*
	J.Page	5000
	V.Smith	3000
	M.Stowe	7000

is consistent wrt the FD $f_1 : Name \rightarrow Salary$. In consequence, the set of consistent answers to the query $Q(x, y) :\ Employee(x, y)$ is $\{(J.Page,\ 5000), (V.Smith,\ 3000), (M.Stowe,\ 7000)\}$. If we add the tuple $(J.Page,\ 8000)$ to the database, the set of consistent answers to the same query is reduced to $\{(V.Smith,\ 3000), (M.Stowe, 7000)\}$. □

We may be interested in having a logical specification $Spec_r$ of the repairs of the database instance r. In this case, we could consistently answer a query $Q(\bar{x})$, by asking for those $\bar{t}$ such that

$$Spec_r \mid\!\approx\ Q(\bar{t}) \quad \equiv \quad r \models_c Q(\bar{t}), \tag{1}$$

where $\mid\!\approx$ is a new, suitable consequence relation, that, as the example shows, has to be non-monotonic.

6.1 A circumscriptive characterization of CQA

Notice that with CQA we have a minimal entailment relation in the sense that consistent answer are true of certain minimal models, those that minimally differ from the original instance. This is a more general reason for obtaining a non-monotonic consequence relation. Actually, the database repairs can be specified by means of a circumscription axiom [28,26] that has the effect of minimizing the set of changes to the original database performed in order to satisfy the ICs.

Let $P_1, \ldots, P_n$ be the database predicates in $\mathcal{L}$. In the original instance r, each P_i has a finite extension that we also denote by P_i. Let $R_1, \ldots, R_n$ be new copies of $P_1, \ldots, P_i$, standing for the corresponding tables in the database repairs. Define, for $i = 1, \ldots, n$,

$$\forall \bar{x}[P_i^{in}(\bar{x}) \quad _{def}\longleftrightarrow \quad (R_i(\bar{x}) \wedge \neg P_i(\bar{x}))], \tag{2}$$

$$\forall \bar{x}[P_i^{out}(\bar{x}) \quad _{def}\longleftrightarrow \quad (P_i(\bar{x}) \wedge \neg R_i(\bar{x}))]. \tag{3}$$

Consider now the theory Σ consisting of axioms (2), (3) plus r, i.e. the (finite) conjunction of the atoms in the database, plus $IC(P_1/R_1, \cdots, P_n/R_n)$, i.e. the set of ICs, but with the original database predicates replaced by the new predicates; and possibly, axioms for the built-in predicates, e.g. equality.

In order to minimize the set of changes, we circumscribe in parallel the predicates P_i^{in}, P_i^{out} in the theory Σ, with variable predicates $R_1, \ldots, R_n$, and fixed predicates $P_1, \ldots, P_n$ [25], that is, we consider the following circumscription

$$Circum(\Sigma; P_1^{in}, \ldots P_n^{out}; R_1, \ldots, R_n; P_1, \ldots, P_n). \tag{4}$$

The semi-colons separate the theory, the predicates minimized in parallel, the variable predicates and the fixed predicate, in that order.

We want to minimize the differences between a database repair and the original database instance. For this reason we need the R_i to be flexible in the minimization process. The original predicates P_is are not subject to changes, because the changes can be read from the R_i (or from their differences with the P_i).

Example 17. Consider $r = \{P(a)\}$ and $IC = \{\forall x(P(x) \rightarrow Q(x))\}$. In this case, Σ consists of the following sentences: $P(a), \forall x(R_P(x) \rightarrow R_Q(x)), \forall x(P^{in}(x) \leftrightarrow R_P(x) \wedge \neg P(x)), \forall x(P^{out}(x) \leftrightarrow P(x) \wedge \neg R_P(x)), \forall x(Q^{in}(x) \leftrightarrow R_Q(x) \wedge \neg Q(x)), \forall x(Q^{out}(x) \leftrightarrow Q(x) \wedge \neg R_Q(x))$. Here the new database predicates are R_P and R_Q. They vary when $P^{in}, P^{out}, Q^{in}, Q^{out}$ are minimized.

The models of th circumscription are the minimal (classical) models of the theory Σ. A model $\mathfrak{M} =< M, (P^{in})^M, (P^{out})^M, (Q^{in})^M, (Q^{out})^M, R_P^M, R_Q^M, P^M, Q^M, a^M >$ is minimal if there is no other model with the same domain M that interprets P, Q, a in the same way as $\mathfrak{M}$ and has at least one of the interpretations of P^{in}, P^{out}, Q^{in}, Q^{out} strictly included in the corresponding in $\mathfrak{M}$ and the others (not necessarily strictly) included in the corresponding in $\mathfrak{M}$. □

Circumscription (4) can be specified by means of a second-order axiom

$$\Sigma(P_1^{in}, \ldots, P_n^{in}, P_1^{out}, \ldots, P_n^{out}, R_1, \ldots, R_n) \ \wedge \tag{5}$$
$$\forall X_1 \cdots \forall X_n \forall Y_1 \cdots \forall Y_n \forall Z_1 \cdots \forall Z_n (\Sigma(X_1, \ldots, X_n, Y_1, \ldots, Y_n, Z_1, \ldots, Z_n) \ \wedge$$
$$\bigwedge_1^n X_i \subseteq P_i^{in} \ \wedge \ \bigwedge_1^n Y_i \subseteq P_i^{out} \ \longrightarrow \ \bigwedge_1^n P_i^{in} \subseteq X_i \ \wedge \ \bigwedge_1^n P_i^{out} \subseteq Y_i).$$

The first conjunct emphasizes the fact that the theory is expressed in terms of the predicates shown there. Those predicates are replaced by second-order variables in the Σ in the quantified part of the formula. The circumscription axiom says that the change predicates R_i^{in}, R_i^{out} have the minimal extension under set inclusion among those that satisfy the ICs. It is straightforward to prove that the database repairs are in one to one correspondence with the restrictions to $R_1, \ldots, R_n$ of those Herbrand models of the circumscription that have domain D and the extensions of the predicates $P_1, \ldots, P_n$ as in the original instance r.

An alternative to externally fixing the domain D consists in minimizing the finite active domain, that is a subset of D. This can be achieved by means of a circumscription as well, and then that domain can be extended to the whole of D. Notice that in order to capture the unique names assumption of databases, the equality predicate could be minimized. Furthermore, if we want the minimal models to have the extensions for the P_i as in r, we can either include in Σ predicate closure axioms of the form $\forall \bar{x}(P_i(\bar{x}) \leftrightarrow \bigvee_1^{k_i} \bar{x}_j = \bar{a}_j)$ if P_i's extension is non-empty and $\forall \bar{x}(P_i(\bar{x})) \leftrightarrow \bar{x} \neq \bar{x})$ if it is empty; or apply to those predicates the closed world assumption, that can also be captured by means of circumscription. See [26] for details. Another alternative is to fix the domain D and replace everywhere r in Σ by the first-order sentence, $\sigma(r)$, corresponding to Reiter's logical reconstruction of database instance r [33]. We do not do any of this explicitly, but leave it as something to be captured at the implementation level.

Example 18. (example 17 continued) The minimal model of the circumscription of the theory are $< D, \emptyset, \{a\}, \emptyset, \emptyset, \emptyset, \emptyset, \{a\}, \emptyset >$ and $< D, \emptyset, \emptyset, \{a\}, \emptyset, \{a\}, \{a\}, \{a\}, \emptyset >$, that show first the domain and next the extensions of $P^{in}, P^{out}, Q^{in}, Q^{out}, R_P, R_Q, P, Q >$, in this order. The first model corresponds to repairing the database by deleting $P(a)$; the second, to inserting $Q(a)$. □

By playing with different kinds of circumscription, e.g. introducing priorities [25], or considering only some change predicates, e.g. only P_i^{out}'s (only deletions), preferences for some particular kinds of database repairs could be captured. We do not explore here this direction any further.

The original theory Σ can be written as $\Sigma' \wedge r$, where Σ' is formed by all the conjunctions in Σ, except for r. It is easy to see that the circumscription $\mathit{Circum}(\Sigma; R_1^{in}, \ldots R_n^{out}; R_1, \ldots, R_n; P_1, \ldots, P_n)$ is logically equivalent to $r \wedge \mathit{Circum}(\Sigma'; R_1^{in}, \ldots R_n^{out}; R_1, \ldots, R_n; P_1, \ldots, P_n)$. In consequence, we can replace (1) by

$$r \wedge \mathit{Circum}(\Sigma'; R_1^{in}, \ldots R_n^{out}; R_1, \ldots, R_n; P_1, \ldots, P_n) \models Q(\bar{t}) \quad \equiv \quad r \models_c Q(\bar{t}). \tag{6}$$

We can see that in this case the nonmonotonic consequence relation $|\approx$ corresponds then to classical logical consequence, but with the original data put in conjunction with a second-order theory.

Some work has been done on detecting conditions and developing algorithms for the collapse of a (second-order) circumscription to a first-order theory [25,15]. The same for collapsing circumscription to logic programs [20]. In our case, this would not be surprising. In [2,22,6], direct specifications of database repairs by means of logic programs are presented.

In our case, there is not much hope in having the circumscription collapse to a first-order sentence, φ_{Circ}. If this were the case, CQA would be feasible in polynomial time in the size of the database, because then for a query Q, the query $(\varphi_{Circ} \rightarrow Q)$ could be posed to the original instance r. As shown in [13], CQA can be coNP-complete, even with simple functional dependencies and (existentially quantified) conjunctive queries. Actually, in the general case CQA is indecidable (to appear in an extended version of [1]).

Under those circumstances, it seems a natural idea to explore to what extent semantic tableaux can be used for CQA. Actually, some implementations to nomonotonic reasoning, more precisely to minimal entailment, based on semantic tableaux have been proposed in [31,29,30,32,10].

6.2 Towards implementation

The most interesting proposal for implementing first order circumscriptive reasoning with semantic tableaux is offered by Niemela in [30], where optimized techniques for developing tableaux branches and checking their minimality are introduced. The techniques presented there, that allow minimized, variable and fixed predicates, could be applied in our context, either directly, appealing to the circumscriptive characterization of CQA we gave before, or adapting Niemela's techniques to the particular kind of process we have at hand, in terms of minimal opening of branches in the tableau $TP(IC \cup r)$.[8] We will briefly explore this second alternative.

As in [30], we assume in this section that (a) the semantic tableaux are applied to formulas in clausal form, and (b) only Herbrand models are considered, what in our case represents no limitation, because our openings, repairs, etc. are all Herbrand structures. Furthermore, if IC contains *safe* formulas [38], what is commonly required in database applications, we can restrict the Herbrand domain to be the finite active domain of the database.

As seen in section 5, consistently answering query Q from instance r wrt IC, can be based on the combination of $op(TP(IC \cup r))$ and $TP(\neg Q(\bar{x}))$. Nevertheless, explicitly having the first, pruned, tableau amounts to having also explicitly all possible repairs of the original database. Moreover, this requires having verified the property of minimality in the data closed branches, possibly comparing

[8] Notice that the input theory in this case differs from the theory to which the circumscription is applied in the previous section.

different branches wrt to inclusion. It is more appealing to check minimality as the tableau $TP(IC \cup r)$ is developed.

Notice that if a finished branch $B \in TP(IC \cup r)$, opened after a preliminary data closure was reached, remains open for $\bar{x} = \bar{t}$ when combined with $TP(\neg Q(\bar{x}))$, then $op(B)$ is a model of IC and $\neg Q(\bar{t})$, and in consequence $op(B)$ provides a counterexample to $IC \models Q(\bar{t})$. However, this is classical entailment, and we are interested in those models of IC that minimally differ from r, in consequence, $op(B)$ may not be a counterexample for our problem of CQA, because the it may not correspond to a repair of the original instance. Such branches that would lead to a non minimal opening in $TP(IC \cup r)$ should be closed, and left closed exactly as those branches that were closed due to built-ins.

As we can see, what is needed is a methodology for developing the tableaux such that: (a) Each potential counterexample is explored, and hopefully at most once. (b) Being a non minimal opening is treated as a closure condition (because, as we just saw, they do not provide appropriate counterexamples). (c) The minimality condition is checked locally, without comparison with other branches, what is much more efficient in terms of space.

Such methodology is proposed in [30], with two classical rules for generating tableaux, a kind of hyper-type rule, and a kind of cut rule. The closure conditions are as in the classical case, but a new closure condition is added, to close branches that do not lead to minimal models. This is achieved by means of a "local" minimality test, that can also be found in [29,17]. We can adapt and adopt such a test in our framework on the basis of the definition of *grounded* model given in [30] and our circumscriptive characterization of CQA given above.

Let B be a data closed branch in $TP(IC \cup r)$, with $op(B) = (r \setminus L) \cup K$. We associate to B a Herbrand structure $M(B)$ over the first order language $\mathcal{L}(\bar{K}, \bar{L}, \bar{P}, \bar{R})$, where $\bar{R} =< R_1, \ldots, R_n >$ is the list of original database predicates, $\bar{P} =< P_1, \ldots, P_n >$ is the list of predicates for the repaired versions of the R_is, $\bar{L} =< L_1, \ldots, L_n >, \bar{K} =< K_1, \ldots, K_n >$ are predicates for $R_i \setminus P_i$ and $P_i \setminus R_i$, resp. (Then it makes sense to identify the list of predicates $\bar{L}$ and $\bar{K}$ with the sets of differences K and L in the branch B). $M(B) =< Act(r), \bar{L}^B, \bar{K}^B, \bar{P}^B, \bar{R}^B >$ is defined through (and can be identified with) the subset $\Lambda := \bigcup_1^n L_i^B \cup \bigcup_1^n K_i^B \cup \bigcup_1^n P_i^B \cup \bigcup_i^n R_i^B$ of the Herbrand base $\mathcal{B}$, where $\bigcup_1^n R_i^B$ coincides with the database contents r, and the elements in $\bigcup_1^n P_i^B$ are taken from $op(B)$.

Now we can reformulate for our context the notion of grounded Herbrand structure given in [30].

Definition 7. (adapted from [30]) An opening $op(B)$ is *grounded* iff for all $p \in \bar{K} \cup \bar{L}$ with $p(\bar{t}) \in \Lambda$ it holds

$$IC(P_1/R_1, \ldots, P_n/R_n) \ \cup \ \{\bigwedge_i^n (L_i = R_i \setminus P_i), \bigwedge_i^n K_i = P_i \setminus R_i)\} \qquad (7)$$
$$\cup \ N^{<\bar{L},\bar{K};\bar{R}>}(\Lambda) \ \models \ p(\bar{t}),$$

where $N^{<\bar{L},\bar{K};\bar{R}>}(\Lambda) := \{\neg q(\bar{t}) \mid q \in \bar{L} \cup \bar{K} \cup \bar{R} \text{ and } q(\bar{t}) \in \mathcal{B} \setminus \Lambda\} \cup \{q(\bar{t}) \mid q \in \bar{R} \text{ and } q(\bar{t}) \in \Lambda\}$. □

Notice that the first set in the union that defines $N^{<\bar{L},\bar{K};\bar{R}>}(\Lambda)$ corresponds to the CWA applied to the minimized predicates. i.e. those in $\bar{L}, \bar{K}$, and the fixed predicates, i.e. those in $\bar{R}$. The second set coincides with the original database contents r. From the results in [30] and our circumscriptive characterization of CQA, we obtain the following theorem.

Theorem 5. An opening $op(B)$ corresponds to a database repair iff $M(B)$ is a grounded model of (7). □

Ungrounded models can be discarded, and then ungroundedness can be used as an additional closure condition on branches. Notice that the test is local to a branch and can be applied at any stage of the development of a branch, even when it is not finished yet. The test is based on classical logical consequence, and then not on any kind of minimal entailment.

Example 19. (example 11 continued) We need some extra predicates. P_P, P_Q, P_R stand for the repaired versions of P, Q, R, resp. $L_P, L_Q, L_R, K_P, K_Q, K_R$ stand for $P \setminus P_P, \ldots, P_R \setminus R$, resp. Here $\bar{L} =< L_P, L_Q, L_R >, \bar{K} =< K_P, K_Q, K_R >, \bar{P} =< P_P, P_Q, P_R >, \bar{R} =< P, Q, R >$.

In order to check groundedness for branches, we have the underlying theory $\Sigma = \{\forall x(P_P(x) \rightarrow P_Q(x)), \forall x(L_P(x) \leftrightarrow (P(x) \wedge \neg P_P(x))), \ldots, \forall x(K_R(x) \leftrightarrow (P_R(x) \wedge \neg R(x)))\}$, corresponding to (7).

In order to check the minimality of branch B_1, we consider $M(B_1)$, that is determined by the set of ground atoms $\Lambda(B_1) = \{P(a), R(b), L_P(a), R_R(b)\}$. First, this structure satisfies Σ. Now, for this branch

$$\begin{aligned} N^{<\bar{L},\bar{K};\bar{R}>}(\Lambda(B_1)) = \{&\neg L_P(b), \neg L_Q(a), \neg L_Q(b), \neg L_R(a), \neg L_R(b), \neg K_P(a), \\ &\neg K_P(b), \neg K_Q(a), \neg K_Q(b), \neg K_R(a), \neg K_R(b), \neg P(b), \\ &\neg Q(a), \neg Q(b), \neg R(a)\} \cup \{P(a), R(b)\}. \end{aligned}$$

For groundedness, we have to check if $L_P(a)$ is a classical logical consequence of $\Sigma \cup N^{<\bar{L},\bar{K};\bar{R}>}(\Lambda(B_1))$. This is true, because, from $\neg K_Q(a)$, we obtain $\neg P_Q(a)$. Using the contrapositive of the IC in Σ, we obtain, $\neg P_P(a)$.

In consequence, the opening corresponding to branch B_1 is a repair of the original database.

Consider now the unfinished branch B_3, for which $\Lambda(B_3) = \{P(a), R(b), K_Q(b), R_P(a), R_Q(b), R_R(b)\}$, and

$$\begin{aligned} N^{<\bar{L},\bar{K};\bar{R}>}(\Lambda(B_3)) = \{&\neg L_P(a), \neg L_P(b), \neg L_Q(a), \neg L_Q(b), \neg L_R(a), \neg L_R(b), \\ &\neg K_P(a), \neg K_P(b), \neg K_Q(a), \neg K_R(a), \neg K_R(b), \neg P(b), \\ &\neg Q(a), \neg Q(b), \neg R(a)\} \cup \{P(a), R(b)\}. \end{aligned}$$

We have to apply the groundedness test to $K_Q(b)$. In this case it is not possible to derive this atom from $\Sigma \cup N^{<\bar{L},\bar{K};\bar{R}>}(\Lambda(B_3))$, meaning that the set of literal is not grounded. If we keep developing that branch, the set N can only shrink. In consequence, we will not derive the atom in the extensions. We can stop developing branch B_3 because we will not get a minimal opening. □

7 Conclusions

We have presented the theoretical basis for a treatment of consistent query answering in relational databases by means of analytic tableaux. We have mainly concentrated on the interaction of the database instance and the integrity constraints; and in the problem of representing database repairs by means of opened tableaux. However, we also showed how the analytic tableaux methodology could we also used for consistent query answering.

We established the connections between the problem of consistent query answering and knowledge base update, on one side, and circumscriptive reasoning, on the other. This is not surprising, since the relationship between knowledge base update and circumscription has already been studied by Winslett [41,40] (see also [24]).

The connection of CQA to updates and minimal entailment allowed us to apply know complexity results to our scenario. Furthermore, we have seen that the reformulation of the problem of CQA as one of computing circumscription opens the possibility of applying established methodologies for semantic tableaux based methodologies for circumscriptive reasoning.

As we have seen, there are several similarities between our approach to consistency handling and those followed by the belief revision/update community. Database repairs coincide with revised models defined by Winslett in [39]. The treatment in [39] is mainly propositional, but a preliminary extension to first order knowledge bases can be found in [14]. Those papers concentrate on the computation of the models of the revised theory, i.e., the repairs in our case, but not on query answering. Comparing our framework with that of belief revision, we have an empty domain theory, one model: the database instance, and a revision by a set of ICs. The revision of a database instance by the ICs produces new database instances, the repairs of the original database.

Nevertheless, our motivation and starting point are quite different from those of belief revision. We are not interested in computing the repairs *per se*, but in answering queries, hopefully using the original database as much as possible, possibly posing a modified query. If this is not possible, we look for methodologies for representing and querying simultaneously and implicitly all the repairs of the database. Furthermore, we work in a fully first-order framework. Other connections to belief revision/update can be found in [1].

To the best of our knowledge, the first treatment of CQA in databases goes back to [9]. The approach is based on a purely proof-theoretic notion of consistent query answer. This notion, described only in the propositional case, is more restricted than the one we used in this paper. In [12], Cholvy presents a general logic framework for reasoning about contradictory information which is based on an axiomatization in modal propositional logic. Instead, our approach is based on classical first order logic.

Other approaches to consistent query answering based on logic programs with stable model semantics were presented in [2,6,22]. They can handle general first order queries with universal ICs.

There are many open issues. One of them has to do with the possibility of obtaining from the tableaux for instances and ICs the right "residues" that can be used to rewrite a query as in [1]. The theoretical basis of CQA proposed in [1] were refined and implemented in [11]. Comparisons of the tableaux based methodology for CQA and the "rewriting based approach" presented in those papers is an open issue. However, query rewriting can not be applied to existential queries like the one in example 15, whereas the tableaux methodology can be used. Perhaps, an appropriate use of tableaux could make possible an extension of the rewriting approach to syntactically richer queries and ICs.

Another interesting open issue has to do with the fact that we have treated Skolem parameters as null values. It would be interesting to study the applicability in our scenario of methodologies for query evaluation in databases in the presence of null values like the one presented in [34].

In this paper we have concentrated mostly on the theoretical foundations of a methodology based on semantic tableaux for querying inconsistent databases. Nevertheless, the methodology for CQA requires further investigation. In this context, the most interesting open problems have to do with implementation issues. More specifically, the main challenge consists in developing heuristics and mechanisms for using a tableaux theorem prover to generate/store/represent $TP(IC \cup r)$ in a compact form with the purpose of: (a) applying the database assumptions, (b) interacting with a DBMS on request, in particular, without replicating the whole database instance at the tableau level, (c) detecting and producing the minimal openings (only), (d) using a theorem prover (in combination with a DBMS) in order to consistently answer queries.

An important issue in database applications is that usually queries have free variables and then answer sets have to be retrieved as a result of the automated reasoning process. Notice that once we have $op(TP(IC \cup r))$, we need to be able to: (a) use it for different queries Q, (b) process the combined tableau $op(TP(IC \cup r)) \otimes TP(\neg Q)$ in an "reasonable and practical" way. We have seen that existing methodologies and algorithms like the one presented in [30], can be used in this direction. However, producing a working implementation, considering all kinds of optimizations with respect to representation and development of the tableaux, grounding techniques, database/theorem-prover interaction, etc. is a major task that deserves separate investigation.

Acknowledgments: Work supported by FONDECYT Grant # 1000593; ECOS /CONICYT Grant C97E05, Carleton University Start-Up Grant 9364-01, and NSERC Grant 250279-02. Preliminary versions of this paper appeared in [4,5]; we are grateful to anonymous referees for their remarks.

References

1. Arenas, A., Bertossi, L. and Chomicki, J. Consistent Query Answers in Inconsistent Databases. Proc. ACM Symposium on Principles of Database Systems (ACM PODS'99). ACM Press, 1999, pp. 68–79.
2. Arenas, M.; Bertossi, L. and Chomicki, J. Specifying and Querying Database Repairs using Logic Programs with Exceptions. In *Flexible Query Answering Systems. Recent Developments*, H.L. Larsen, J. Kacprzyk, S. Zadrozny, H. Christiansen (eds.). Springer-Verlag, 2000, pp. 27–41.
3. Arenas, A., Bertossi, L. and Chomicki, J. Scalar Aggregation in FD-Inconsistent Databases. In Database Theory - ICDT 2001 (Proc. International Conference on Database Theory, ICDT'2001). Springer LNCS 1973, 2001, pp. 39 – 53.
4. Bertossi, L. and Schwind, C. B. An Analytic Tableaux based Characterization of Database Repairs for Consistent Query Answering (preliminary report). In Working Notes of the IJCAI'01 Workshop on Inconsistency in Data and Knowledge. AAAI Press, 2001, pp. 95 – 106.
5. Bertossi, L. and Schwind, C. B. Analytic Tableaux and Database Repairs: Foundations. In *Foundations of Information and Knowledge Systems (Proc. FoIKS 2002)*, Eiter, T. and Schewe, K.-D. (eds.). Springer LNCS 2284, 2002, pp. 32-48.
6. Barcelo, P. and Bertossi, L. Repairing Databases with Annotated Predicate Logic. In *Proc. Ninth International Workshop on Non-Monotonic Reasoning (NMR'2002). Special session on Changing and Integrating Information: From Theory to Practice*. S. Benferhat and E. Giunchiglia (eds.). Morgan Kaufmann Publishers, 2002, pp. 160 – 170.
7. Belleannée, C., Kuhna, P., Lamarre, P., Schwind, C., Thiébaux, S., Vialard, V. and Vorc'h, R. Méthodes Sémantiques de Démonstration pour Logiques Non-standards. In *PRC GDR Intelligence artificielle, Actes des 5èmes Journées Nationales*, Nancy 2-5, février 1995.
8. Beth, E. W. *The Foundations of Mathematics*. North Holland, 1959.
9. Bry, F. Query Answering in Information Systems with Integrity Constraints. In Proc. IFIP WG 11.5 Working Conference on Integrity and Control in Information Systems, Chapman & Hall, 1997.
10. Bry, F. and Yahya, A.H. Positive Unit Hyperresolution Tableaux and Their Application to Minimal Model Generation. *Journal of Automated Reasoning*, 25(1) (2000) 35–82.
11. Celle, A. and Bertossi, L. Querying Inconsistent Databases: Algorithms and Implementation. In 'Computational Logic - CL 2000', J. Lloyd et al. (eds.). Stream: 6th International Conference on Rules and Objects in Databases (DOOD'2000). Springer LNAI 1861, 2000, pp. 942 – 956.
12. Cholvy, L. A General Framework for Reasoning about Contradictory Information and some of its Applications. In Proceedings of ECAI Workshop "Conflicts among Agents", Brighton, England, August 1998.
13. Chomicki, J. and Marcinkowski, J. On the Computational Complexity of Consistent Query Answers. Submitted in 2002 (CoRR paper cs.DB/0204010).
14. Chou, T. and Winslett, M. A Model-Based Belief Revision System. *J. Automated Reasoning*, 12 (1994) 157–208.
15. Doherty, P., Lukaszewicz, W. and Szalas, A. Computing Circumscription Revisited: A Reduction Algorithm. *Journal of Automated Reasoning*, 18(3) (1997) 297–336.
16. Eiter, T. and Gottlob, G. On the Complexity of Propositional Knowledge Base Revision, Updates, and Counterfactuals. *Artificial Intelligence*, 57 (1992) 227-270.

17. Eiter, T. and Gottlob, G. Propositional Circumscription and Extended Closed World Assumption are Π_2^p-complete. *Theoretical Computer Science*, 114 (1993) 231-245.
18. Fitting, M. First Order Modal Tableaux. *Journal of Automated Reasoning*, 4(2) (1988) 191–213.
19. Fitting, M. *First Order Logic and Automated Theorem Proving*. Texts and Monographs in Computer Science. Springer-Verlag, 2nd Edition, 1996.
20. Gelfond, G. and Lifschitz, V. Compiling Circumscriptive Theories into Logic Programs. In *Non–Monotonic Reasoning*. Springer LNAI 346, 1989, pp. 74–99.
21. Gottlob, G. Complexity Results for Nonmonotonic Logics. *Journal of Logic and Computation*, 2(3) (1992).
22. Greco, G.; Greco, S. and Zumpano, E. A Logic Programming Approach to the Integration, Repairing and Querying of Inconsistent Databases. In *Proc. 17th International Conference on Logic Programming (ICLP'01)*, Ph. Codognet (ed.). Springer LNCS 2237, 2001, pp. 348–364.
23. Lafon, E. and Schwind, C. B. A Theorem Prover for Action Performance. In Y. Kodratoff, editor, *Proceedings of the 8th European Conference on Artificial Intelligence*, Pitman Publishing, 1988, pp. 541–546.
24. Liberatore, P. and Schaerf, M. Reducing Belief Revision to Circumscription (and vice versa). *Artificial Intelligence*, 93 (1997) 261–296.
25. Lifschitz, V. Computing Cirscumscription. In Proc. IJCAI'85, 1985, pp. 121-127.
26. Lifschitz, V. Circumscription. In Handbook of Logic in Artificial Intelligence and Logic Programming, Vol. 3. Oxford University Press, 1994, pp. 297–352.
27. Lloyd, J.W. *Foundations of Logic Programming*. Springer-Verlag, 1987.
28. McCarthy, J. Applications of Circumscription to Formalizing Common Sense Knowledge. *Artificial Intelligence*, 26(3) (1986) 89–118.
29. Niemela, I. A Tableau Calculus for Minimal Model Reasoning. In Proc. Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods. Springer LNCS 1071, 1996, pp. 278-294.
30. Niemela, I. Implementing Circumscription Using a Tableau Method. Proc. ECAI 1996, pp. 80–84.
31. Olivetti, N. Tableaux and Sequent Calculus for Minimal Entailment. *Journal of Automated Reasoning*, 9(1) (1992) 99–139.
32. Olivetti, N. Tableaux for Nonmonotonic Logics. In Handbook of Tableaux Methods. Kluwer Publishers, 1999, pp. 469–528.
33. Reiter, R. Towards a Logical Reconstruction of Relational Database Theory. In 'On Conceptual Modeling', Brodie, M. L. and Mylopoulos, J. and Schmidt, J. W. (eds.). Springer-Verlag, 1984, pp. 191–233.
34. Reiter, R. A Sound and Sometimes Complete Query Evaluation Algorithm for Relational Databases with Null Values. *Journal of the ACM*, 33(2) (1986) 349–370.
35. Schwind, C. B. A Tableau-based Theorem Prover for a Decidable Subset of Default Logic. In M. E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*. Springer LNAI 449, 1990, pp. 541–546.
36. Schwind, C. B. and Risch, V. Tableau-based Characterisation and Theorem Proving for Default Logic. *Journal of Automated Reasoning*, 13(4) (1994) 223–242.
37. Smullyan, R. M. *First Order Logic*. Springer-Verlag, 1968.
38. Ullman, J. *Principles of Database and Knowledge-Base Systems, Vol. I*. Computer Science Press, 1988.
39. Winslett, M. Reasoning about Action with a Possible Models Approach. In *Proceedings of the 8th National Conference on Artificial Intelligence*, 1988, pp. 89–93.

40. Winslett, M. Cirscumscriptive Semantics for Updating Knowledge Bases. *Annals of Mathematics and Artificial Intelligence*, 3(2-4) (1991) 429–.
41. Winslett, M. Sometimes Updates are Circumscription. *Proceeding of the International Joint Conference on Artificial Intelligence (IJCAI'89)*, 1989, pp. 859-863.

Appendix: Proofs

Proof of Lemma 3

We have by Lemma 1 $r'\Delta r = r \setminus r'$ and $r''\Delta r = r \setminus r''$. Then $l \in r'\Delta r$ iff $l \in r \setminus r'$, i.e. $l \in r$ and $l \notin r'$ from which it follows that $l \in r$ and $l \notin r''$. Hence $l \in r \setminus r'' = r''\Delta r$.

Proof of Lemma 4

Let r' be an opening of r. Then $r' = (r \setminus L) \cup K$, where $L = \{l : l \in r \text{ and } \neg l \in I\}$ and $K = \{l : l \in I \text{ and there is no substitution } \sigma \text{ such that } l\sigma \in r\}$. Let us first observe that $L \cap K = \emptyset$ since $L \subseteq r$ and for $l \in K$, $l \notin r$. We show that $r\Delta r' = L \cup K$. Let be $x \in r\Delta r'$.
1. Case $x \in r$ and $x \notin r'$. Then $x \notin K$ and $x \notin (r \setminus L)$. But from this, we get $x \in L$, hence $x \in L \cup K$
2. Case $x \notin r$ and $x \in r'$, iff $x \notin r$ and (($x \in r$ and $x \notin L$) or $x \in K$), iff $x \notin r$ or $x \in K$ from which it follows $x \in K \cup L$.

On the other hand, let be $x \in L \cup K$. Again, we consider two cases:
1. Case $x \in L$, then by definition, $\neg x \in I$. Then, $x \notin r \setminus L$ and, since I is open, $x \notin I$. From this, we get $x \notin K$ and, since $r' = (r \setminus L) \cup K$, $x \notin r'$, from which it follows that $x \in r\Delta r'$.
2. Case $x \in K$, then $x \in I$ and $x \notin r$. But then $x \in r'$ and therefore $x \in r\Delta r'$.

Proof of Proposition 3

By Lemma 4, we have $r\Delta r_1 = L_1 \cup K_1$ and $r\Delta r_2 = L_2 \cup K_2$. From $r_1 \leq_r r_2$ we get then $L_1 \cup K_1 \subseteq L_2 \cup K_2$. Since $L_i \cap K_i = \emptyset$, we have $L_1 \subseteq L_2$ and $K_1 \subseteq K_2$.

Proof of Theorem 2

Let r' be a repair of r. Then $r' \models IC$ and $r' \in Min_{leq_r}(ic)$. Since r' is a model of IC, by Theorem 1, r' contains an open branch I of the tableau $TP(IC)$ for IC. We have $r' = (r \setminus L) \cup K$ and since r' is minimal wrt $\leq_r$, there is no r'' closer to r than r'. i.e. there is no $r'' = (r \setminus L') \cup K'$ such that $L' \subset L$ and $K' \subset K$. Hence $r' \cup I$ is a minimal opening of $r \cup I$.

On the other hand, let $I \cup r'$ be a minimal opening of $I \cup r$ in $TP(IC \cup r)$ where I is an open branch of $TP(IC)$. Then, by Definition 6, $r' = (r \setminus L) \cup K$ where $L = \{l : l \in r \text{ and } \neg l \in I \text{ and } K = \{l : l \in I \text{ and there is no substitution } \sigma$ such that $l\sigma \in r\}$. By Lemma 4, we have $r\Delta r' = L \cup K$. Since $I \cup r$ is a minimal opening of $I \cup r'$, we have by Theorem 3, that there is no r'', L'' and K'' such that r'' is an opening of r and $r'' = (r \setminus L'') \cup K''$ and $L'' \subset L$ and $K'' \subset K$. By Lemma 4, this means that there is no r'' such that $r\Delta r'' \subset r\Delta r'$, i. e. r' is a minimal element of $Mod(IC)$ wrt the order $\leq_r$.