Answers that Have Integrity

Hendrik Decker

Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Spain

Note:
This is a revised version of a paper with the same title, published in a collection of revised papers that were selected from the 4th International Workshop on Semantics in Data and Knowledge Bases (SDKB), Bordeaux, France, July 5, 2010, pp. 54–72. The copyright owner of the original paper is Springer-Verlag Berlin Heidelberg. Consult that paper at the following reference:

Hendrik Decker:
Answers that Have Integrity.
In *Semantics in Data and Knowledge Bases* (*SDKB 2010*), 4th International Workshop, Revised Selected Papers. Proceedings edited by Klaus-Dieter Schewe and Bernhard Thalheim, published in Springer Lecture Notes in Computer Science, vol. 6834, 2011.

Answers that Have Integrity

Hendrik Decker *

Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Spain

Abstract. Answers to queries in possibly inconsistent databases may not have integrity. We formalize 'has integrity' on the basis of a definition of 'causes'. A cause of an answer is a minimal excerpt of the database that explains why the answer has been given. An answer has integrity if one of its causes does not overlap with any cause of integrity violation.

1 Introduction

We continue the development of 'answers that have integrity' (in short, *AHI*) in databases that may suffer from extant violations of integrity. It has begun in [6], for databases, queries, constraints and answers without negation. In this paper, definitions and results are generalized to be applicable also if there is negation.

Consistent query answering (CQA) [1] is a popular approach to provide useful answers in inconsistent databases. Roughly, CQA defines an answer to be consistent if the answer is true in each minimal repair. Unfortunately, the consistency of answers is not invariant under different notions of minimality.

We elaborate the alternative idea of answers that 'have integrity', i.e, answers that are reasonably correct in the presence of integrity violation. This idea is based on 'causes', i.e., certain extracts of the database that explain why an answer is given, or why a constraint is violated. Intuitively, an answer has integrity if one of its causes does not overlap with any cause of integrity violation.

Arguably, AHI does not suffer from any ambivalence of minimality, nor from several other shortcomings associated to CQA. However, while computing AHI for definite databases and queries is very simple, it seems to be as complex as computing CQA in general.

Apart from some background of database logic, we broach, in Section 2, the only-if halves of predicate completions as a basis for defining causes of negative answers. Section 3 contains the main definitions for characterizing causes. In Section 4, we define and discuss how to compute AHI. In Section 5, we compare AHI to related work. In Section 6, we conclude with an outlook to further work.

2 Preliminaries

In 2.1, we address the foundations of the database logic upon which the remainder of the paper is built. In 2.2, we make explicit an implicit part of the database, viz. ground instances of the only-if halves of predicate definitions, since they may contribute to causes for explaining negative answers.

* supported by FEDER and the Spanish grants TIN2009-14460-C03, TIN2010-17139

2.1 Background

As a formal framework, we opt for *datalog*. We denote logical consequence by $\models$. We use the abbreviation *iff* for 'if and only if'.

We assume a universal language denoted by $\mathcal{L}$ that contains a finite universal domain $\mathcal{L}^c$ of constant terms over which each attribute variable in each database ranges. W.l.o.g., we represent the elements in $\mathcal{L}^c$ by natural numbers. By overloading, we use $=$ as the identity predicate in $\mathcal{L}$, as the assignment symbol in substitutions of variables with terms, and as meta-level equality. Since ',' is used as the conjunction operator between literals in the body of clauses, we use ';' as the delimiter between elements of sets of clauses.

As in [21], we call each database without any occurrence of negative literals in the body of its clauses a *definite* database, and each database without self-recursive definitions of predicates a *hierarchical* database. As opposed to definite databases, which may contain self-recursive definitions of predicates, hierarchical database clauses may contain negative literals in their body. Unless mentioned otherwise, each database considered in this paper is assumed to be hierarchical. Answers that have integrity in definite databases are studied in [6].

We say that a formula F is a *conjunctive sentence* if F is a universally closed non-empty conjunction of literals. If all literals in F are positive, we also say that F is a *definite sentence*.

For each database D and each conjunctive sentence F, we write $D(F) = \mathit{true}$ and say 'F is *true* in D' if F is a logical consequence of the theory associated to D (e.g., the completion of D, or some standard model theory such as the set of stable models of D). Otherwise, we write $D(F) = \mathit{false}$ and say 'F is *false* in D'.

Let D be a database and L a ground literal such that $D(L) = \mathit{true}$. We say that L is *terminal* in D if the atom of L does not match the head of any clause with non-empty body in D. For instance, $\sim p(1,2)$ is terminal in $\{p(x,1) \leftarrow q(x);\ q(1)\}$. If the predicates in $\mathcal{L}$ are partitioned, as usual, into extensional and intensional ones, then each extensional fact is terminal.

We assume, w.l.o.g., that each query is a conjunctive query, possibly with negative literals, where the predicates in the query are defined by clauses in the database. Similarly, we assume that each integrity constraint (shortly, constraint) is a *denial*, i.e., a clause with empty head and a conjunction of (possibly negative) literals as its body. Each finite set of constraints is called an *integrity theory*.

We assume each clause C to be *range-restricted*, i.e., each variable in C occurs is a positive literal in the body of C.

We do not flatten databases by materializing the definition of predicates or by representing the database by some standard model, since such an unfolding of database clauses with non-empty body may lead to a loss of causal information.

Example 1. The two databases $D_1 = \{p \leftarrow r;\ r;\ s\}$ and $D_2 = \{p \leftarrow s;\ r;\ s\}$ have the same flattened version: $\{p;\ r;\ s\}$. That version, however, does not provide to identify that r is part of the cause of the truth of p in D_1 and s is not, nor that s is part of the cause of the truth of p in D_2 and r is not.

Definition 1. For each database D, let D^+ denote the set of ground instances of clauses in D.

2.2 Only-if Halves

In [6], we have defined causes for positive answers in definite databases. For that, it was sufficient to define a cause very simply as a minimal subset of D^+ of which the answer is a consequence. Now, we are going to deal also with negation.

Since causes are meant to be some sort of explanation, causes of negative answers and positive consequences thereof need to recur on the negative information that is conveyed by the database. Negative consequences of databases usually are justified by some closure axioms. For negative consequences of definite databases, the closed world assumption *CWA* [23] usually is the closure axiom of choice. For each hierarchical database D, the completion $comp(D)$ (which coincides with CWA for definite databases without recursion) is the standard axiomatization for inferring negative consequences from D.

For each predicate p in $\mathcal{L}$, the well-known *iff-completion* (also called *predicate completion* or, in short, *completion*) of p is defined in [5, 21].

Example 2. Let $D = \{p(x,1) \leftarrow r(x);\ p(1,y) \leftarrow s(y,z);\ s(1,2);\ s(2,3)\}$. Further, let us assume that p, r and s are the only predicate symbols in $\mathcal{L}$. Then, $comp(D)$ consists of the following iff-completions, each of which is a universally closed sentence with existentially quantified 'local' variables that do not occur in the head of any clause. We omit the universal quantifier prenex for all non-local variables in the completions below, and also the equality theory associated to $comp(D)$ that interprets $=$ as identity.

$p(x,y) \ \leftrightarrow\ (y=1 \wedge r(x) \ \vee\ x=1 \wedge \exists z(s(y,z)))$

$\sim r(x)$

$s(x,y) \ \leftrightarrow\ (x=1 \wedge y=2 \ \vee\ x=2 \wedge y=3)$

Definition 2. Let D be a database, $p(x_1, \ldots, x_n)$ an atom in $\mathcal{L}$, $\underline{p}$ the iff-completion of p in $comp(D)$, n ($n \geq 0$) the arity of p, θ a (not necessarily ground) substitution of the variables $x_1, \ldots, x_n$ and $A = p(x_1, \ldots, x_n)\theta$.

a) The *iff-completion* $\underline{A}$ of A is obtained by applying θ to the $\forall$-quantified variables in $\underline{p}$.

b) The sentence obtained by replacing $\leftrightarrow$ in $\underline{A}$ with $\rightarrow$ is called the *only-if half* of $\underline{A}$, and A is called the *head* of that sentence.

c) Let D^- denote the set of only-if halves of all ground atoms in $\mathcal{L}$. Assume that D^- is factorized modulo renamings of variables.

For easy reading, we are going to represent elements of D^- in a simplified form, if possible. It is obtained by replacing equations with their truth value and applying common equivalence-preserving laws for the composition of subformulas with *true* or *false*. Elements of D^- the simplification of which is *true* are omitted.

Example 3. For D as in Example 2, the only-if half of $\underline{p}$ is

$p(x,y) \rightarrow (y=1 \wedge r(x) \vee x=1 \wedge \exists z(s(y,z)))$.

Its instance

$p(1,1) \rightarrow (1=1 \wedge r(1) \vee 1=1 \wedge \exists z(s(1,z)))$,

obtained by the substitution $\{x=1,\ y=1\}$, is obviously equivalent to

$p(1,1) \rightarrow (\mathit{true} \wedge r(1) \vee \mathit{true} \wedge \exists z(s(1,z)))$

which can be further simplified to

$p(1,1) \rightarrow (r(1) \vee \exists z(s(1,z)))$.

Similarly, the instance

$p(2,3) \rightarrow (3=1 \wedge r(2) \vee 2=1 \wedge \exists z(s(3,z)))$

obtained by the substitution $\{x=2,\ y=3\}$ of x and y is equivalent to

$p(2,3) \rightarrow (\mathit{false} \wedge r(2) \vee \mathit{false} \wedge \exists z(s(3,z)))$

which simplifies to

$p(2,3) \rightarrow \mathit{false}$

which finally is equivalent to

$\sim p(2,3)$.

Similarly, the instance $s(2,3) \rightarrow (2=1 \wedge 3=2 \ \vee \ 2=2 \wedge 3=3)$ of the only-if half of the completion of s first simplifies to $s(2,3) \rightarrow (\mathit{false} \wedge \mathit{false} \vee \mathit{true} \wedge \mathit{true})$, which is equivalent to $s(2,3) \rightarrow \mathit{true}$, which is equivalent to true.

3 Excerpts, Explanation Bases, Causes

We develop a formal notion of our intuition of 'cause' in three steps.

First, we define an 'excerpt' of a database D to consist of a 'positive' and a 'negative excerpt', i.e., a subset of D^+ (Definition 1) and, resp., a subset of D^- (Definition 2*c*).

Second, we define, for each database D and each sentence F that is *true* in D, an 'explanation base' of F in D to be an excerpt E of D such that F is a logical consequence of E.

Third, we define a 'cause' of F in D to be a minimal explanation base of F in D. Thus, a cause is an explanation base that is free of superfluous elements.

Similarly, we define the causes of an answer substitution θ of the variables in a query $\leftarrow B$ in D to be the causes of $\forall(B\theta)$, i.e., the universal closure of $B\theta$, in D, and the causes of the violation of a denial constraint $\leftarrow B$ as the causes of the answer *yes* (i.e., the identity substitution) to the query $\leftarrow \mathit{violated}$ in $D \cup \{\mathit{violated} \leftarrow B\}$, where *violated* is a distinguished 0-ary predicate that does not occur in D.

In section 4, we will then define that an answer to a query in a database D 'has integrity' if it has a cause that does not overlap with any cause of the violation of any constraint in D.

3.1 Excerpts

Definition 3. Let D be a database and P a set of ground sentences. P is called a *positive excerpt* of D if $P \subseteq D^+$.

The following result entails that the truth of a definite sentence in a positive excerpt P of a definite database cannot be non-monotonically changed by adding possibly non-definite clauses to P.

Theorem 1. For each positive excerpt P of some database and each conjunctive sentence F such that $P \models F$, F is *true* in each database of which P is a positive excerpt.

To prove Theorem 1, it is useful to observe that the premise $P \models F$ entails that F necessarily is a definite sentence and that there is a definite database P' such that $P' \subseteq P$ and $P'(F) = \mathit{true}$. Then, Theorem 1 is a direct consequence of Definition 3 and the monotonicity of definite databases.

As soon as an excerpt is supposed to capture not only positive, but also negative consequences of some database D, the latter need to be made explicit. That motivates Definition 4, below, which recurs on ground only-if halves in D^-.

Definition 4. Let D be a database and N a set of ground sentences. N is called a *negative excerpt* of D if $N \subseteq D^-$.

As in Example 2, we assume that the usual equality theory of $\mathit{comp}(D)$ is associated by default to each negative excerpt.

Now, we are in the position to formalize our intuition of 'excerpt'.

Definition 5. Let D be a database, P a positive excerpt of D and N a negative excerpt of D. Then, the tuple $E = (P, N)$ is called an *excerpt* of D. P is called the *positive part*, and N the *negative part* of E. For each excerpt E, we denote its positive part by E^+, its negative part by E^-, and the union of E^+ and E^- by $\hat{E}$. We say that two excerpts E, E' *overlap* if $\hat{E} \cap \hat{E}' \neq \emptyset$.

The generalization of Theorem 1 below follows from Definitions 3 and 5.

Theorem 2. For each excerpt E of some database and each conjunctive sentence F such that $\hat{E} \models F$, F is *true* in each database of which E is an excerpt.

A partial order $\leq$ is defined in a natural way for excerpts, as follows.

Definition 6. Let D be a database and $E = (P, N)$, $E' = (P', N')$ be two excerpts of D. We define that $E \leq E'$ if $\hat{E} \subseteq \hat{E}'$, i.e., if $P \subseteq P'$ and $N \subseteq N'$. We write $E < E'$ and say E *is smaller than* E' if $E \leq E'$ and not $E' \leq E$ holds.

3.2 Explanation Bases

An 'explanation base' E of the truth of a conjunctive sentence F (shortly, an 'explanation base' of F) in a database D is going to be defined as an excerpt E of D such that F is a logical consequence of $\hat{E}$. For definite databases and definite conjunctive sentences, a similar but simpler definition has been presented in [6].

Example 4. Let $D = \{p \leftarrow q;\ p \leftarrow r;\ q;\ r;\ s\}$. Clearly, $P_1 = \{p \leftarrow q;\ q\}$ is a positive excerpt of D that serves to explain why p is *true* in D. Also $P_2 = \{p \leftarrow r;\ r\}$ is a positive excerpt of D that explains why p is *true* in D.

It is easy to see that no negative excerpts are needed in order to explain positive consequences of any definite database D; positive excerpts of D are sufficient, as illustrated in Example 4. However, for causally explaining negative consequences or, more generally, consequences of databases with negation in the body of clauses, positive excerpts are not sufficient.

Indeed, it is not possible to infer any negative logical consequence, such as sentences represented by negative literals or denials, from any positive excerpt. Of course, negative consequences can be inferred from the completion of each positive excerpt, but not each such consequence is necessarily *true* in the given database. For instance, all ground negative literals are *true* in the empty database, which, by Definition 8, is a positive excerpt of each database D. Moreover, Example 5 below shows that the invariance results expressed in Theorems 1 and 2 cease to hold if negation is allowed in D or F.

Example 5. Consider again $D = \{p \leftarrow q;\ p \leftarrow r;\ q;\ r;\ s\}$ and P_i $(i = 1, 2)$ as in example 4. While each definite sentence that is *true* in P_i is also *true* in each database D' of which P_i is a positive excerpt, negative consequences of P_i do not necessarily hold in each such D'. For instance, the conjunctive sentence $s \wedge \sim t$ is *true* in D and each P_i, but is *false* in $D' = D \cup \{t\}$, although each P_i is a positive excerpt also of D'. Hence, P_i is not sufficient for explaining p.

The circumstance that positive excerpts are insufficient in general for explaining negative consequences of hierarchical databases can be seen by an even simpler example.

Example 6. It is not possible to characterize a cause of $\sim q$ in $D = \{p\}$ by any subset of D. In particular, none of the positive excerpts $\emptyset$ and $\{p\}$ of D may explain $\sim q$ in D since each of them is also a positive excerpt of $D' = D \cup \{q\}$, while the truth value of $\sim q$ is not the same in D and D'.

In the preceding example, the absence of q in D must be captured explicitly for explaining why $\sim q$ is *true* in D. Thus, each explanation base is going to consist of a positive and a negative excerpt, as defined subsequently.

Definition 7. Let D be a database, E an excerpt of D, and F a conjunctive sentence such that $D(F) = \mathit{true}$. E is called an *explanation base* of F in D if $\hat{E} \models F$.

Example 7.
a) Let $D = \{p \leftarrow q, s;\ q;\ r\}$ and $F = \sim p$. Clearly, $(\emptyset,\ \{p \rightarrow q, s;\ \sim s\})$ is an explanation base of F in D. However, neither $E_1 = (\{p \leftarrow q, s\},\ \{\sim s\})$ nor $E_2 = (\{p \leftarrow q, s;\ q\},\ \{\sim s\})$ is an explanation base of F in D, since F is not a logical consequence of $\hat{E}_i$ $(i = 1, 2)$. Note that both excerpts E_i of D are also excerpts of $D' = D \cup \{p \leftarrow r\}$, but $D'(F) = \mathit{false}$.

b) Let $D = \{p(x) \leftarrow q(x), \sim r(x);\ q(1);\ q(2);\ r(3)\}$. Then, $D(p(1)) = \mathit{true}$, but $p(1)$ is not explainable by any positive excerpt of D alone, since each subset of D^+ is also a positive excerpt of $D' = \{p(x) \leftarrow q(x), \sim r(x);\ q(1);\ q(2);\ r(1)\}$, but $D'(p(1)) = \mathit{false}$. However, $p(1)$ can be explained by the excerpt $E = (\{p(1) \leftarrow q(1), \sim r(1);\ q(1)\},\ \{\sim r(1)\})$ of D, since $\hat{E} \models p(1)$, and $p(1)$ is indeed *true* in each database D' of which E is an excerpt.

The result below is an immediate consequence of Definition 7 and Theorem 2.

Theorem 3. For each database D, each conjunctive sentence F, each explanation base E of F in D, and each database D' of which E is an excerpt, it follows that $D'(F) = \mathit{true}$.

Theorem 3 expresses that the truth of each conjunctive sentence F in D that is explained by some explanation base of F in D is independent of any clause that is present or absent in D but not captured by E. Note that this result holds although database negation is non-monotonic. Also the consequences of Definition 7 in Proposition 1, below, hold in spite of non-monotonicity.

Proposition 1. Let D be a database, F a conjunctive sentence such that $D(F) = \mathit{true}$, and E an explanation base of F in D. Then, the following holds.

a) $E^+(F) = \mathit{true}$.

b) For each literal L that is terminal in D, $(\{L\},\ \emptyset)$ is the only explanation base of L in D if L is positive, and $(\emptyset,\ \{L\})$ is the only explanation base of L in D if L is negative.

c) For each excerpt E' of D such that $E \leq E'$, E' is an explanation base of F in D.

Part *a* of Proposition 1 says that F is *true* in the positive part of each explanation base E of F in D. However, as we have seen in examples 6 and 7*b*, F is not necessarily *true* of each database of which E^+ is a positive excerpt. Yet, F is *true* of each database of which E is an excerpt, according to Theorem 3. Part *b* says that each ground literal with extensional predicate explains itself. Part *c* says that each excerpt which contains an explanation base of F in D also is an explanation base of F in D. Thus, each explanation base is sufficient for inferring the explained sentence from it, but its truth does not necessarily depend on each element in a given explanation base, in general. Such a condition of necessity is going to be added in the definition of causes, in 3.3.

3.3 Causes

As shown by Proposition 1*c*, there may be superfluous literals in an explanation base, i.e., the presence or absence of such literals in an explanation base of some formula F in some database D is irrelevant for explaining the truth of F in D.

Example 8. Clearly, $(\{p(1) \leftarrow r(1,2);\ r(1,2);\ s(3)\},\ \emptyset)$ is an explanation base of $p(1)$ in $D = \{p(x) \leftarrow r(x,y);\ r(1,1);\ r(1,2);\ r(2,2);\ s(3)\}$, but $s(3)$ is superfluous for explaining $p(1)$ in D.

Thus, a cause of F in D is going to be defined as a minimal explanation base, without irrelevant elements, as follows.

Definition 8. For each database D, each conjunctive sentence F and each explanation base E of F in D, E is called a *cause* of F in D if there is no explanation base of F in D that is smaller than E.

Thus, a cause of a sentence F in a database D is an explanation base E of F in D without superfluous clauses, i.e., it is not possible to explain F with any extract obtained by dropping any element from E.

In Definitions 9–11, Definition 8 is extended to positive and negative answers, constraint violation and satisfaction, and to integrity violation and satisfaction.

Definition 9. Let D be a database, B a conjunction of literals, θ a substitution of the variables in B such that $\forall(B\theta) = \mathit{true}$, and E a cause of $\forall(B\theta)$ in D.

a) If $\leftarrow B$ is a query, we say: E is a *cause of the answer* θ to $\leftarrow B$ in D.

b) If $\leftarrow B$ is a constraint, we say: E is a *cause of the violation* of $\leftarrow B$ in D.

Definition 10. Let D be a database, B a conjunction of literals, and E a cause of the answer *yes* (i.e., the identity substitution) to the query $\leftarrow \sim \mathit{answer}$ in $D \cup \{\mathit{answer} \leftarrow B\}$, where *answer* be a 0-ary predicate not occurring in D.

a) If $\leftarrow B$ is a query, we say: E is a *cause of the answer no* to $\leftarrow B$ in D.

b) If $\leftarrow B$ is a constraint, we say: E is a *cause of the satisfaction* of $\leftarrow B$ in D.

The following results are immediate consequences of Definitions 9 and 10.

Proposition 2. Let D be a database, E an excerpt of D, A a ground atom, B a conjunction of literals, θ a substitution, and *answer* a predicate that does not occur in D.

a) E is a cause of the answer *yes* (resp., *no*) to $\leftarrow A$ in D iff
E is a cause of the answer *no* (resp., *yes*) to $\leftarrow \sim A$ in D.

b) E is a cause of the answer θ to $\leftarrow B$ in D iff
$(E^+ \cup \{\mathit{answer} \leftarrow B\theta\gamma \mid \gamma \text{ is a substitution, } B\theta\gamma \text{ is ground}\},\ E^-)$ is a cause of the answer *yes* to $\leftarrow$ *answer* in $D \cup \{\mathit{answer} \leftarrow B\}$.

c) If $\leftarrow B$ is a query, then E is a cause of the answer *no* to $\leftarrow B$ in D iff $(E^+,\ E^- \cup \{\mathit{answer} \rightarrow \exists B\})$ is a cause of $\sim$*answer* in $D \cup \{\mathit{answer} \leftarrow B\}$.

d) If $\leftarrow B$ is a constraint, then E is a cause of the satisfaction of $\leftarrow B$ in D iff $(E^+,\ E^- \cup \{\mathit{answer} \rightarrow \exists B\})$ is a cause of $\sim$*answer* in $D \cup \{\mathit{answer} \leftarrow B\}$.

Definition 11. Let D be a database, $IC = \{I_1, \ldots, I_n\}$ an integrity theory containing n constraints of the form $I_i = \leftarrow B_i$ ($1 \leq i \leq n$, $n > 0$), and E a cause of the answer *no* (resp., *yes*) to $\leftarrow \sim answer$ in $D \cup \{answer \leftarrow B_i \mid 1 \leq i \leq n\}$, where *answer* be a 0-ary predicate not occurring in D. Then, we say that E is a *cause of the violation of integrity* (resp., a *cause of the satisfaction of integrity*) in (D, IC), or simply, a *cause of the violation* (resp., *satisfaction*) of IC in D.

Below, Examples 9a, b illustrate Definitions 8 and 9a. Examples 9c, d illustrate Definition 9b. Examples 9e, f illustrate Definition 10a. Examples 9g–i illustrate Definition 10b; 9i also is an example for Definition 11. Finally, Example 9j shows that causes of the violation of integrity (Definition 11) may not be composed of the causes of the violation of constraints in that theory (Definition 9b).

Example 9.

a) Let $D = \{p \leftarrow q(1,2);\ q(2,y) \leftarrow r(y);\ r(1)\}$. The only cause of $\sim p$ in D, as well as of the answer *yes* to $\leftarrow \sim p$ in D, is $(\emptyset,\ \{p \rightarrow q(1,2);\ \sim q(1,2)\})$.

b) Let $D = \{p \leftarrow \sim q;\ q \leftarrow \sim r;\ q \leftarrow \sim s\}$. The two causes of $\sim p$ and of the answer *yes* to $\leftarrow \sim p$ in D are $(\{q \leftarrow \sim r\}, \{p \rightarrow \sim q;\ \sim r\})$ and $(\{q \leftarrow \sim s\}, \{p \rightarrow \sim q;\ \sim s\})$.

c) Let $D = \{p \leftarrow q;\ p \leftarrow \sim q\}$ and $I = \leftarrow p$. The two causes of the violation of I in D are $(\{p \leftarrow \sim q\}, \{\sim q\})$ and $(D, \emptyset)$.

d) Let $D = \{p(x) \leftarrow r(x); r(1)\}$ and $I = \exists x(r(x) \wedge \sim p(x))$ be a constraint. Clearly, $D(I) = false$ (in fact, I is violated in each database that contains $p(x) \leftarrow r(x)$). A denial form of I is $\leftarrow violated$, where *violated* is defined by $\{violated \leftarrow \sim q;\ q \leftarrow r(x), \sim p(x)\}$ (q is a fresh 0-ary predicate). Thus, the causes of the violation of I in D are the causes of *violated* in $D' = D \cup \{violated \leftarrow \sim q;\ q \leftarrow r(x), \sim p(x)\}$. Thus, for each $\mathcal{K} \subseteq \mathcal{L}^c$ such that $1 \in \mathcal{K}$, a cause of *violated* in D' is given by $(\{violated \leftarrow \sim q\} \cup \{p(i) \leftarrow r(i) \mid i \in \mathcal{K}\}, \{q \rightarrow \exists x(r(x) \wedge \sim p(x))\} \cup \{\sim r(i) \mid i \notin \mathcal{K}\})$. There are no other causes of *violated* in D'.

e) Let $D = \{p \leftarrow q(1,x);\ q(2,y) \leftarrow r(y);\ r(1)\}$. The only cause of the answer *no* to the query $\leftarrow p$ in D is $(\emptyset,\ \{p \rightarrow \exists x\, q(1,x)\} \cup \{\sim q(1,i) \mid i \in \mathcal{L}^c\})$.

f) Let $D = \{p \leftarrow r(x), s(x); r(1); s(2)\}$. Each cause E of the answer *no* to $\leftarrow p$ in D contains the excerpt $E_0 = (\emptyset, \{p \rightarrow \exists x(r(x) \wedge s(x)); \sim s(1); \sim r(2)\})$ of D. Each E also contains, for each $i \in \mathcal{L}^c$, $i > 2$, either $\sim r(i)$ or $\sim s(i)$, and nothing else.

g) Let $D = \{p \leftarrow q;\ p \leftarrow \sim q;\ q\}$ and $I = \leftarrow \sim p$. The two causes of the satisfaction of I in D (i.e., of p in D) are $(\{p \leftarrow q;\ q\}, \emptyset)$ and $(\{p \leftarrow q;\ p \leftarrow \sim q\}, \emptyset)$.

h) Let $D = \{r(1),\ r(3),\ s(2),\ s(4)\}$ and $I = \leftarrow r(x), s(x)$ a constraint. Depending on the extent of $\mathcal{L}^c$, there may be many causes of the satisfaction of I in D. The positive part of each such cause is empty, and each contains the excerpt $(\emptyset,\ \{violated \rightarrow \exists x\,(r(x) \wedge s(x))\} \cup \{\sim s(1),\ \sim s(3),\ \sim r(2),\ \sim r(4)\})$ of $D \cup \{violated \leftarrow r(x), s(x)\}$. (Recall that the cause of the satisfaction of I in D is the cause of $\sim violated$ in $D \cup \{violated \leftarrow r(x), s(x)\}$.) Moreover, the negative part of each cause of the satisfaction of I in D contains, for each $i > 4$ in $\mathcal{L}^c$, either $\sim r(i)$ or $\sim s(i)$, and no other element.

i) Let $D = \{p \leftarrow q, \sim q\}$, $I_1 = \leftarrow p$ and $I_2 = \leftarrow \sim p$. The only cause of the satisfaction of I_1 as well as of the violation of I_2 in D is $(\emptyset, \{p \rightarrow q \wedge \sim q\})$, which also is the only cause of the violation of $IC = \{I_1, I_2\}$ in D.

j) Let $D = \{r(1,1);\ s(1)\}$, $I_1 = \leftarrow r(x,x)$, $I_2 = \leftarrow r(x,y), s(y)$ and $IC = \{I_1;\ I_2\}$. The only cause of the violation of IC in D is $(\{r(1,1)\}, \emptyset)$, which is smaller than the single cause $(D, \emptyset)$ of the violation of I_2 in D.

Note that examples 9 *c*, *g*, *i* feature constraints and an integrity theory that, together with the rules in the database, are either tautological (i.e., always satisfied) or contradictory (i.e., always violated), no matter which facts are in the database. Interestingly, some of the causes in the mentioned examples are not devoid of database facts, as one could suspect, since the truth of tautological satisfaction and the falsity of contradictory violation is independent of the presence or absence of facts in the database. The use of facts for explaining the violation of contradictory constraints also means that our concept of causes is applicable even in databases with unsatisfiable integrity theories, as opposed to CQA.

4 Defining and Computing AHI

In 4.1, we define that an answer to a query in a database D 'has integrity' if one of its causes does not overlap with any cause of the violation of integrity in D. In 4.2, we show how to compute explanation bases and causes. In 4.3, we address the problem of computing sufficiently many causes for checking if answers have integrity.

4.1 Defining AHI

Definition 12. Let D be a database, IC an integrity theory, $\leftarrow B$ a query and α an answer to $\leftarrow B$ in D. (i.e., α is either a substitution or the answer *no*).

a) We say that α *has weak integrity* in (D, IC) if there is a cause of the answer in D that does not overlap with any cause of the violation of IC in D.

b) We say that α *has strong integrity* if there is a cause of the answer in D that does not overlap with any cause of the violation of any constraint in IC in D.

Example 10. Consider D as in Example 9*f* and let $\leftarrow \sim p$ be a constraint. According to Proposition 2, the causes of its violation in D are the same as the causes of the answer *no* to the query $\leftarrow p$. From Definition 12, it follows that the answers $x = 1$ and $x = 2$ to the queries $\leftarrow r(x)$ and, resp., $\leftarrow s(x)$ both have strong integrity, while the answer *no* to $\leftarrow p$ does not even have weak integrity.

It is easy to see that an answer has weak integrity if it has strong integrity. Thus, we may say that an answer *does not have integrity* if it does not have weak integrity. However, weak and strong integrity of answers according to Definition 12 are not equivalent, as shown by the following example.

Example 11. Let $D = \{r(1,1);\ s(1)\}$, $IC = \{\leftarrow r(x,x);\ \leftarrow r(x,y), s(y)\}$. The answer $x=1$ to the query $\leftarrow s(x)$ in D has weak integrity, since its only cause $(\{s(1)\}, \emptyset)$ does not overlap with the only cause $(\{r(1,1)\}, \emptyset)$ of the violation of IC in D. However, it does not have strong integrity, since $(\{s(1)\}, \emptyset)$ overlaps with the cause $(\{r(1,1); s(1)\}, \emptyset)$ of the violation of the constraint $\leftarrow r(x,y), s(y)$. The answer $\{x{=}1, y{=}1\}$ to $\leftarrow r(x,y)$ in D does not have integrity, since its only cause $(\{r(1,1)\}, \emptyset)$ overlaps with some cause of the violation of IC in D (in fact, it is identical with the only cause of the violation of IC in D).

Example 11 illustrates the differentiation of weak and strong integrity in Definition 12. In fact, a distinction between different grades of integrity is desirable, since, for instance, the violation of $\leftarrow r(x,y), s(y)$, which is caused, among others, by the presence of $s(1)$ in D, may cast some doubt on the integrity of the answer $x=1$ to the query $\leftarrow s(x)$ in D. Even more refined differentiations of the integrity of answers are possible, as shown in [6] for definite databases and conjunctive queries without negative literals.

4.2 Computing Explanations and Causes

As seen in [6], SLD resolution [18, 21] provides an easy way to compute causes of positive answers and integrity violation of definite queries and denials in definite databases. Each cause of each answer corresponds to a refutation R that computes the answer: input clauses of R, instantiated with the substitution computed by R, are the elements of the cause. Hence, AHI can be computed by comparing causes drawn from refutations with causes of integrity violation. If the latter have been computed ahead of query time, then checking for overlaps can already be done while the answer is computed.

Similarly, for each SLDNF refutation R and each finitely failed SLDNF tree T of some query in a database, an explanation for the answer computed by R or, resp., T can be obtained as described below in Definition 14. To prepare this definition, we first recall some basic SLDNF issues from [21] and ask the reader to agree on some denotations.

Let D be a database and $\leftarrow B$ a query. An *SLDNF computation* of $D \cup \{\leftarrow B\}$ is either an SLDNF refutation or a finitely failed tree of $D \cup \{\leftarrow B\}$. Each SLDNF computation involves one top-rank computation and possibly several subsidiary computations of lower rank, spawned by the selection of ground negative literals in goals of derivations.

It is easy to see that no finitely failed tree of rank n-1 that is subsidiary to some finitely failed tree T of rank n could contribute anything to explain the answer *no* computed by T. Thus, such subsidiary trees are ignored in Definition 13. It characterizes the set of computations involved in an SLDNF computation S that contribute to an explanation of the answer to the root of S.

Definition 13. Let S be an SLDNF computation of rank n $(n \geq 0)$.

a) The set S_r of *explanatory refutations* of S consists of each refutation R of rank k involved in S such that either $k=n$ and $R=S$, or $k<n$ and R is subsidiary to a tree in S_t of rank $k+1$.

b) The set S_t of *explanatory trees* of S consists of each finitely failed tree T of rank k involved in S such that either $k=n$ and $T=S$, or $k<n$ and T is subsidiary to a refutation in S_r of rank $k+1$.

Note that the mutual recursion of parts a and b in Definition 13 does not pose any problem since D is hierarchical, i.e., the rank of each computation is bounded by the rank of the top-rank computation, and the rank of each subsidiary computation decreases iteratively until the lowest rank without subsidiary inferences is reached.

An SLDNF computation S is called *fair* if, in each tree $T \in S_t$ and each goal G in T, one of the most recently introduced literals is selected in G.

For each refutation R, let θ_R denote the substitution computed by R. The projection of θ_R to the variables in the root of R is the *computed answer* of R. For each database D, each query $\leftarrow B$ and each finitely failed tree T of $D \cup \{\leftarrow B\}$, the *computed answer* of T is *no*.

For each clause C, each only-if half H and each substitution θ, let $C\theta$, resp., $H\theta$ denote the formula obtained by applying θ to the $\forall$-quantified variables in C or, resp., H. For each only-if half H, let $h(H)$ denote the head of H.

Now, we are in the position to define computed explanations.

Definition 14. For each SLDNF computation S, the *computed explanation* E_S of S consists of

$$E_S^+ = \{C\theta_R \mid C \in D,\ R \in S_r,\ C \text{ is input clause in } R\}$$

and

$$E_S^- = \{H\gamma \mid H\gamma \in D^-,\ \gamma \text{ is a ground substitution, } h(H) \text{ is selected in some node of some tree in } S_t\}.$$

Thus, E_S^+ is obtained by instantiating the positive input clauses of each refutation $R \in S_r$ with θ_R. E_S^- is obtained by collecting the only-if-halves of all ground instances of each positive literal selected in any node of any tree in S_t.

Example 12. Let $D = \{p(x) \leftarrow q(x,x);\ q(1,2);\ q(2,3)\}$. The answer *no* to the query $\leftarrow p(x)$ is computed by a finitely failed tree consisting of a single branch rooted at $\leftarrow p(x)$, which is reduced to the goal $\leftarrow q(x,x)$, which fails. The only-if halves of the two selected positive literals in the tree are $p(x) \rightarrow q(x,x)$ and $q(x,x) \rightarrow (x{=}1 \wedge x{=}2 \vee x{=}2 \wedge x{=}3)$. The latter obviously is equivalent to $\sim q(x,x)$. Thus, $(\emptyset,\ \{p(i) \rightarrow q(i,i) \mid i \in \mathcal{L}^c\} \cup \{\sim q(i,i) \mid i \in \mathcal{L}^c\})$ is the computed explanation of the tree, which in fact is also a cause of $\sim p(x)$ in D.

Theorem 4 is easily inferred from Definition 14.

Theorem 4. For each database D, each query $\leftarrow B$ and each SLDNF computation S of $D \cup \{\leftarrow B\}$, the computed explanation of S is an explanation base of the answer computed by S. If $S_t = \emptyset$, then the computed explanation of S is a cause of the answer computed by S.

The following example illustrates that explanations computed by finitely failed SLDNF trees are not necessarily causes, since they may not be minimal explanation bases.

Example 13. Let $D = \{p \leftarrow q, r;\ r \leftarrow s\}$. Depending on the selection function, there are three SLDNF trees of $D \cup \{\leftarrow p\}$. In each, the goal $\leftarrow q, r$ is derived from the root. Then, if q is selected, the computation terminates with failure. If, instead, r is selected, the derived goal $\leftarrow q, s$ may fail in two ways, after selecting either q or s. Hence, depending on the selection, precisely one of the following three explanation bases $(\emptyset,\ \{p \rightarrow q \wedge r;\ \sim q\})$, $(\emptyset,\ \{p \rightarrow q \wedge r;\ r \rightarrow s;\ \sim q\})$, $(\emptyset,\ \{p \rightarrow q \wedge r;\ r \rightarrow s;\ \sim s\})$ can be drawn from the respective SLDNF tree. Only the first and the last of these explanation bases are causes of the answer *no* to the query $\leftarrow p$ in D, while the middle one is not because it properly contains the first one and thus is not minimal.

If, in Example 13, a fair selection policy is employed, the computation of a non-minimal explanation is avoided. However, fair selection alone is not enough, in general, as shown by the following example.

Example 14. Let $D = \{p \leftarrow q(x), r;\ q(1)\}$. Both left-to-right and right-to-left selection is fair for computing the answer *no* to $\leftarrow p$ in D. However, only the latter yields a minimal explanation, while the former computes the explanation $(\emptyset,\ \{p \rightarrow \exists x(q(x) \wedge r)\} \cup \{\sim q(i) \mid i \in \mathcal{L}^c,\ i \neq 1\} \cup \{\sim r\})$, in which each $\sim q(i)$ is superfluous.

Thus, each explanation E computed by a SLDNF computation must eventually be minimized, by checking if any proper subset of E satisfies Definition 8. Also, selection strategies such as those proposed in [9] can be used to obtain finitely failed trees with minimal explanations.

4.3 Computing Causes for AHI

As stated by Theorem 4, each finitely failed SLDNF tree computes a single explanation base, from which a cause for explaining its root can be drawn. However, Definition 12 entails that *all* causes of violations may be needed for deciding if the answer *no* to a given query has integrity or not. Thus, SLDNF computations may be incomplete for computing AHI. That is illustrated in the following continuation of Example 13.

Example 15. Depending on whether q or r is selected in the goal $\leftarrow q, r$, the cause drawn from a fair finitely failed SLDNF tree of $D \cup \{\leftarrow p\}$ is either $E_1 = (\emptyset,\ \{p \rightarrow q \wedge r;\ \sim q\}$ or $E_2 = (\emptyset,\ \{p \rightarrow q \wedge r;\ r \rightarrow s;\ \sim s\}$. Now, let $\leftarrow \sim q$ be a constraint. Clearly, the cause $(\emptyset, \{\sim q\})$ of its violation in D overlaps with E_1 but not with E_2. Hence, each SLDNF computation of $D \cup \{\leftarrow p\}$ that selects literals from left to right fails to detect that the answer *no* to $\leftarrow p$ in D has integrity.

The incompleteness of SLDNF for computing all minimal explanations of answers had already been identified in [9]. A solution similar to that in [9] can be applied for attaining completeness. Essentially, it consists in fairly selecting and trying to resolve not just one, but several literals in each goal derived from the root, such that several finitely failed trees are obtained. In Example 15,

for instance, both q and r have to be selected and processed for obtaining two finitely failed SLDNF trees, from which the two causes E_1 and E_2 can be drawn.

At worst, each fairly selectable literal in each goal of an explanatory finitely failed SLDNF tree may have to be selected and attempted to be resolved, when aiming to draw sufficiently many causes from SLDNF computations, for computing AHI. Thus, the number of explanatory finitely failed trees to be built may grow linearly with the number of selectable literals, polynomially with the rank of computations and exponentially with the depth of explanatory finitely failed trees. This indicates a worst case complexity of computing AHI like that of CQA.

Example 16, a continuation of Example 9f, shows that not each cause can be drawn from the set of explanatory SLDNF computations obtained by selecting and resolving (if possible) each literal in the goals of explanatory finitely failed trees.

Example 16. Clearly, two SLDNF trees are built when selecting and resolving both $r(x)$ and $s(x)$ in the goal $\leftarrow r(x), s(x)$ derived from $\leftarrow p$. Using input clauses $r(1)$ or, resp., $s(2)$, the respective resolvents $\leftarrow s(1)$ and $\leftarrow r(2)$ are derived. Each of them fails when selected and processed, and so does the respective tree. From the two trees, $E_1 = (\emptyset, \{p \rightarrow \exists x(r(x), s(x))\} \cup \{\sim r(i) \mid i \in \mathcal{L}^c,\ i \neq 1\} \cup \{\sim s(1)\})$ and, resp., $E_2 = (\emptyset, \{p \rightarrow \exists x(r(x), s(x))\} \cup \{\sim s(i) \mid i \in \mathcal{L}^c,\ i \neq 2\} \cup \{\sim r(2)\})$ can be drawn as a cause of the answer *no* of $D \cup \{\leftarrow p\}$.

Depending on $\mathcal{L}^c$, many more causes of that answer may exist. For instance, $E = (\emptyset, \{p \rightarrow \exists x(r(x), s(x))\} \cup \{\sim r(i) \mid i \in \mathcal{L}^c,\ i \notin \{1; 3\}\} \cup \{\sim s(1); \sim s(3)\})$ cannot be drawn from any of the two trees above, according to Definition 13. However, E_1 and E_2 are sufficient for determining if the computed answer has integrity or not, since each possible overlap with any cause of the answer *no* to $\leftarrow p$ in D contains at least one of the elements in $\hat{E}_1 \cup \hat{E}_2$.

For instance, consider the constraint $I = \leftarrow \sim r(3)$. The cause of its violation in D is $(\emptyset, \{\sim r(3)\})$. That overlaps with E_1, but not with E_2. Hence, the two trees of $D \cup \{\leftarrow p\}$ obtained as described above yield that the answer *no* to $\leftarrow p$ has integrity in $(D, \{I\})$.

The preceding example illustrates that the number of causes for explaining an answer may far exceed the number of causes that can be drawn from SLDNF computations, even if not just one, but several finitely failed trees are built for obtaining sufficiently many causes of negative answers. Nevertheless, the following lemma can be shown, by induction on the rank of computations. The symbol α below may either stand for some answer substitution or the answer *no*. In words, the lemma states that each element in each cause is contained in some cause drawn from a computed explanation.

Lemma For each database D, each query $\leftarrow B$, each answer α to $\leftarrow B$ in D, each cause E of α and each $e \in \hat{E}$, there is an SLDNF computation S of $D \cup \{\leftarrow B\}$ with computed answer α such that $e \in \hat{E}'$, where $\hat{E}'$ is a cause drawn from $\hat{E}_S$.

From that, the subsequent result about the soundness and completeness of computing AHI with SLDNF can be inferred.

Theorem 5. Let D be a database, IC an integrity theory, $\leftarrow B$ a query, and α an answer to $\leftarrow B$ in D.

a) α has weak integrity iff there is an SLDNF computation S of $D \cup \{\leftarrow B\}$ with computed answer α and a cause E of α drawn from S that does not overlap with any cause drawn from any SLDNF computation of the violation of IC in D.

b) α has strong integrity iff there is an SLDNF computation S of $D \cup \{\leftarrow B\}$ with computed answer α and a cause E of α drawn from S that does not overlap with any cause drawn from any SLDNF computation of the violation of any constraint $I \in IC$ in D.

5 Related Work

In 5.1, we take first steps of comparing AHI with CQA. In 5.2, we relate belief revision, knowledge assimilation and abduction to AHI. In 5.3, we address other related work.

5.1 AHI vs CQA

There is yet no comprehensive analysis of commonalities and differences between AHI and CQA. However, each of the following paragraphs identifies a point in favour of answers that have integrity.

The simplicity of the basic definitions of AHI appears to be on a par with the simplicity of the basic definitions of CQA. However, the minimality of explanation bases required in Definition 8 does not suffer from the ambivalence of the minimality of repairs in CQA.

As seen in 4.1, Definition 12 provides for a differentiation of various degrees of consistency. Such a differentiation is not provided by CQA. For instance, recall that, in Example 11, the answer $x = 1$ to the query $\leftarrow s(x)$ in D has weak but not strong integrity. According to CQA, the same answer also is a consistent answer to the same query, since the only minimal repair of the violation of IC is $D' = \{s(1)\}$, obtained by deleting $r(1, 1)$ from D. However, CQA does not distinguish different grades of the consistency of answers, such as the distinction of weak and strong integrity in Definition 12. In fact, some doubts may be raised on the consistency of the answer $y = 1$ to the query $\leftarrow s(y)$ in D, as claimed by CQA, since the violation of the constraint $\leftarrow r(x, y), s(y)$ is caused, among others, by the presence of $s(1)$ in D.

Inconsistency can be measured in accordance with AHI, simply by counting causes, or by comparing sets of causes of the violation of constraints. A similar way to measure inconsistency in accordance with CQA could be to quantify the minimality of repairs, but that appears to be less simple, again due to the multiplicity of notions of minimality.

By definition, an answer has integrity only if it is *true* in the given database. More precisely, if, for an answer substitution θ of a query $\leftarrow B$ in a database D, $\forall(B\theta)$ has integrity in D, then $D(\forall(B\theta)) = true$. Similarly, if the answer *no* to $\leftarrow B$ in D has integrity, then $D(\forall(\sim B)) = true$. As opposed to that, consistent answers according to CQA may be *false* in the given database. For instance, let $D = \{p\}$ be a database and $IC = \{\leftarrow p\}$ an integrity theory. Clearly, the answer *yes* to the query $\leftarrow p$ does not have integrity and is not consistent. However, this answer as well as the information about its lack of integrity is given only by AHI, not by CQA. In fact, the answer to $\leftarrow p$ in D given by CQA is *no*, since the only minimal repair is the empty database. Arguably, the information conveyed by the answer according to AHI (that p is *true* in D but does not have integrity) is more useful than the answer *no* according to CQA.

Answers may or may not have integrity in databases with unsatisfiable integrity theories. As opposed to that, each answer is consistent by definition of CQA if integrity is unsatisfiable. That is because each answer is vacuously *true* in each repair if there is no repair. For instance, let $D = \{r(a),\ s(b,b)\}$ and $IC = \{\exists\, s(x,x);\ \leftarrow s(x,y)\}$, which is clearly unsatisfiable. Rewriting $\exists\, s(x,x)$ in denial form yields $IC' = \{\leftarrow \sim q;\ \leftarrow s(x,y)\}$ and $D' = D \cup \{q \leftarrow s(x,x)\}$. The answer {x=a} to the query $\leftarrow r(x)$ in D' has integrity, while the answer {x=b} to $\leftarrow s(x,x)$ does not. Yet, both answers are consistent by the definition of CQA.

CQA does not consider that the integrity theory, rather than the database could be in need of a repair, while AHI is impartial wrt both possibilities. For example, let $D = \{q(1,2,3,1), q(2,3,2,4), q(2,1,2,3)\}$, $IC = \{\leftarrow q(x,y,x,z)\}$ and $\leftarrow q(2,x,y,z)$ be a query. Clearly, none of the two answers given to the query has integrity in (D, IC), since their respective cause coincides with one of the two causes $(\{q(2,3,2,4)\}, \emptyset)$, $(\{q(2,1,2,3)\}, \emptyset)$ of the violation of IC in D. The answer to the same query according to CQA is *no*, since the only minimal repair of D is to delete the two tuples given as answers according to AHI. However, it might well be that IC, rather than D is in need of a repair (which can of course only be determined if the particular 'real-world' meanings of q and IC are taken into account). For instance, a reasonable repair of IC could be $IC' = \{\leftarrow q(x,y,z,y)\}$. For any change of IC, all answers given to queries in D according to AHI remain the same, while answers given by CQA may change completely, such as they do in the preceding example.

Causes for AHI in definite and in hierarchical databases can be computed by SLD, resp., SLDNF derivations and only-if halves of predicate completions, i.e., by conventional query answering procedures and, for hierarchical databases, a well-known logic programming concept. As opposed to that, each of the four known approaches to compute CQA [4] appears to be more complicated or more unusual: One uses techniques from semantic query optimization [3]. Two others compute CQA from compact representations of repairs, either by conflict graphs or by extended disjunctive logic programs. Finally, repairs can be computed explicitly in order to decide whether an answer is *true* in each repair.

5.2 Repairing, Belief Revision, Knowledge Assimilation, Abduction

AHI and CQA tolerate inconsistency, since extant integrity violations may persist after the query has been answered. A less tolerant way to query a database D that is inconsistent with its integrity theory IC is to actually repair D, i.e., to minimally update D such that the resulting database satisfies IC, and then query that database. In the context of knowledge bases, repairing is often called 'belief revision' [25]. However, the minimality of repairs suffers from the ambivalence already mentioned for CQA. Moreover, repairs usually are not unique, i.e. an answer given with regard to a particular repair may not be a correct answer with regard to some other repair. AHI does not suffer from such ambiguities.

A field closely related to repairing and belief revision is knowledge assimilation (abbr. KA) [8]. For a database D, a query R (sometimes called 'update request') and an integrity theory IC that is satisfied in D, KA is asked to compute minimal updates such that the updated database satisfies IC and R is *true* in it. View updating is a well-known special case of KA. Again, such updates are not unique in general, and their minimality is ambivalent.

Abduction is a technique to compute KA [16]. Several abductive resolution procedures, e.g., as described in [17, 24, 14, 13, 22], are able to compute answers in a given database D, together with a set H of hypothesized facts, such that $D \cup H$ is consistent with regard to a given integrity theory. The hypothesized facts of abductively computed answers can also be interpreted as partial repairs of extant integrity violations. (For a definition and discussion of partial repairs, see [10].) We conjecture that the answers computed by the cited procedures in the partially repaired databases have integrity. Moreover, a striking similarity of AHI and the abductive (C)IFF procedures in [14, 22] is that each of them makes explicit use of only-if halves of predicate completions.

5.3 Other related work

In [15], negative database consequences such as those that can be inferred from iff-completions are made explicit for inconsistency-tolerant reasoning (and, in particular, query answering) with explicit representations of the database theory, including its integrity constraints. The formalizations in [15] have to sacrifice several classical inference rules that are cornerstones of the logic framework assumed in this paper, such as modus ponens and reductio ad absurdum.

Lots of other work is going on in terms of inconsistency-tolerant and paraconsistent reasoning, which all is somehow related to CQA, AHI or avoiding the inference of arbitrary answers according to the *ex contradictione quodlibet* principle. For instance, inconsistency tolerance is discussed broadly in [2]. Another example is inconsistency-tolerant integrity checking (abbr. ITIC), which has been related to procuring the improvement of answer consistency in databases with extant integrity violations in [10]. As outlined in [7, 6], a new approach to inconsistency-tolerant integrity checking is possible, based on the concept of causes as described in 3.3. As already indicted in 5.1, causes can also be used to measure inconsistency, similar to quantifying inconsistency by counting the number of violated instances of constraints, as done in [11].

There is a millenia-old tradition of philosophical treatment of causality. More recently, it has come to bear on informatics and AI, mostly in the form of probabilistic or counterfactual reasoning, as witnessed by a growing number of congresses about computing and philosophy. Those treatments of causality mostly go beyond the framework of causes as in this paper, which is confined by the comparatively simple theory of databases, the semantics of which is given by nothing but the completions of predicates.

6 Conclusion

This paper generalizes the concept of answers that have integrity (AHI) as presented in [6]. AHI distinguishes useful from doubtful answers: an answer is doubtful, i.e., does not have integrity, iff its causes overlap with the causes of integrity violation in the given state.

The contribution of this paper is twofold. Firstly, the definition of causes in [6] has been extended to negative answers, by taking the only-if halves of completed predicate definitions into account. That extension permits to apply the simple conceptual idea of answers that have integrity also to queries, constraints and databases involving datalog negation. Secondly, the computation of positive answers that have integrity by SLD, for definite queries in definite databases, has been generalized by suitable extensions of SLDNF to positive and negative answers for first-order queries in hierarchical databases.

In upcoming work, we intend to investigate the conjecture that it may suffice to check overlaps of terminal literals in causes for deciding if an answer has integrity or not. In [7], we have shown that this conjecture holds for positive answers to queries and constraints in definite databases.

Also, ways to avoid redundant multiple selection of literals in goal nodes is an important issue to be explored, in order to limit the cost of cause-of-failure computations. For instance, consider $D = \{q(1), r(1), s(2)\}$. Four of the six finitely failed SLDNF trees in the cause computation of $D \cup \{\leftarrow q(x), r(x), s(x)\}$ yield the same cause of the answer *no* to the given query, i.e., three of them are redundant. This redundancy can be avoided without sacrificing completeness by, e.g., never selecting any ground literal that is *true* in D, in any cause-of-failure computation.

Moreover, we intend to look into possibilities of computing AHI by other procedures for query evaluation, such as abductive logic programming [12, 22] or answer set computing [20].

Moreover, it should be interesting to look into explanation bases and causes for a fresh take on explanations in expert systems [19].

References

1. M. Arenas, L. Bertossi, J. Chomicki. Consistent query answers in inconsistent databases. *Proc. 18th PODS*, 68-79. ACM Press, 1999.

2. L. Bertossi, A. Hunter, T. Schaub. *Inconsistency Tolerance*. Springer LNCS 3300, 2005.
3. S. Chakravarthy, J. Grant, J. Minker. Logic-based approach to semantic query optimization. *ACM TODS* 15(2):162-207, 1990.
4. J. Chomicki. Consistent Query Answering: Five Easy Pieces. *Proc. 11th ICDT*, 1-17. Springer LNCS 4353, 2007.
5. K. Clark. Negation as Failure. In H. Gallaire, J. Minker (eds): *Logic and Data Bases*, 293-322. Plenum Press, 1978.
6. H. Decker. Toward a Uniform Cause-based Approach to Inconsistency-tolerant Database Semantics. In R. Meersman, T. Dillon and P. Herrero (Eds.), *On the Move to Meaningful Internet Systems: OTM 2010*, Part II, 983-998. Springer LNCS 6427, 2010.
7. H. Decker. Basic causes for the Inconsistency Tolerance of Query Answering and Integrity Checking. *Proc. 21st DEXA Workshop*, 318-322. IEEE CPS, 2010.
8. H. Decker. Some Notes on Knowledge Assimilation in Deductive Databases. In *Transactions and Change in Logic Databases*, 249-286. Springer LNCS 1472, 1998.
9. H. Decker. On Explanations in Deductive Databases. *Proc. 3rd Workshop on Foundations of Models and Languages for Data and Objects*, Inst. f. Informatik, Tech. Univ. Clausthal, Informatik-Bericht 91/3, 173-185, 1991.
10. H. Decker, D. Martinenghi. Inconsistency-tolerant Integrity Checking. *Transactions of Knowledge and Data Engineering* 23(2):218-234, 2011.
11. H. Decker, D. Martinenghi. Modeling, Measuring and Monitoring the Quality of Information. *Proc. ER Workshop QOIS*, 212-221. Springer LNCS 5833, 2009.
12. M. Denecker, A. Kakas. Abduction in Logic Programming. In A. Kakas, F. Sadri (Eds.): *Computational Logic*, 402-436. Springer LNAI 2407, 2002.
13. P. Dung, R. Kowalski, F. Toni. Dialectic proof procedures for assumption-based admissible argumentation. *Artif. Intell.* 170(2):114-159, 2006.
14. T. Fung, R. Kowalski. The IFF proof procedure for abductive logic programming. *J. Logic Programming* 33(2):151-165, 1997.
15. T. Hinrichs, J. Kao, M. Genesereth. Inconsistency-Tolerant Reasoning with Classical Logic and Large Databases. *Proc. 8thSARA*, 105-112. AAAI Publications, 2009.
16. A. Kakas, R. Kowalski, F. Toni. The role of abduction in logic programming. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, *vol. 5*, 235-324. Oxford Univ. Press, 1998.
17. A. Kakas, P. Mancarella. Database Updates through Abduction. *Proc. 16th VLDB*, 650-661. Morgan Kaufmann, 1990.
18. R. Kowalski. Predicate Logic as a Programming Language. *Proc. 6th IFIP*, 569-574. North-Holland, 1974.
19. J. Liebowitz (ed). *Handbook of Applied Expert Systems*. CRC Press, 1998.
20. V. Lifschitz. What is Answer Set Computing? *Proc. 23rd AAAI*, 1594-1597, 2008.
21. J. Lloyd. *Foundations of Logic Programming (2nd edition)*. Springer, 1987.
22. P. Mancarella, G. Terreni, F. Sadri, F. Toni, U. Endriss. The CIFF proof procedure for abductive logic programming with constraints: Theory, implementation and experiments. *TPLP* 9(6):691-750, 2009.
23. R. Reiter. On closed world data bases. In H. Gallaire, J. Minker (eds): *Logic and Data Bases*, 119-140. Plenum Press, 1978.
24. F. Toni, R. Kowalski. Reduction of abductive logic programs to normal logic programs. *Proc. 12th ICLP*, 367-381. MIT Press, 1995.
25. M.-A. Williams. Applications of Belief Revision. In *Transactions and Change in Logic Databases*, 287-316. Springer LNCS 1472, 1998.