Consistent query answers on numerical databases under aggregate constraints

Sergio Flesca, Filippo Furfaro, and Francesco Parisi

DEIS - Università della Calabria
Via Bucci - 87036 Rende (CS) ITALY
Fax: +39 0984 494713
{flesca, furfaro, parisi}@si.deis.unical.it

Abstract. The problem of extracting consistent information from relational databases violating integrity constraints on numerical data is addressed. In particular, aggregate constraints defined as linear inequalities on aggregate-sum queries on input data are considered. The notion of repair as consistent set of updates at attribute-value level is exploited, and the characterization of several complexity issues related to repairing data and computing consistent query answers is provided.

1 Introduction

Research has deeply investigated several issues related to the use of integrity constraints on relational databases. In this context, a great deal of attention has been devoted to the problem of extracting reliable information from databases containing pieces of information inconsistent w.r.t. some integrity constraints. All previous works in this area deal with "classical" forms of constraint (such as keys, foreign keys, functional dependencies), and propose different strategies for updating inconsistent data reasonably, in order to make it consistent by means of minimal changes. Indeed these kinds of constraint often do not suffice to manage data consistency, as they cannot be used to define algebraic relations between stored values. In fact, this issue frequently occurs in several scenarios, such as scientific databases, statistical databases, and data warehouses, where numerical values of tuples are derivable by aggregating values stored in other tuples.

In this work we focus our attention on databases where stored data violates a set of *aggregate constraints*, i.e. integrity constraints defined on aggregate values extracted from the database. These constraints are defined on numerical attributes (such as sales prices, costs, etc.) which represent measure values and are not intrinsically involved in other forms of constraints.

Example 1. Table 1 represents a two-years *cash budget* for a firm, that is a summary of cash flows (receipts, disbursements, and cash balances) over the specified periods. Values '*det*', '*aggr*' and '*drv*' in column *Type* stand for *detail*, *aggregate* and *derived*, respectively. In particular, an item of the table is *aggregate* if it is obtained by aggregating items of type *detail* of the same section, whereas a *derived* item is an item whose value can be computed using the values of other items of any type and belonging to any section.

A cash budget must satisfy these integrity constraints:

Year	***Section***	***Subsection***	***Type***	***Value***
2003	Receipts	beginning cash	drv	20
2003	Receipts	cash sales	det	100
2003	Receipts	receivables	det	120
2003	Receipts	total cash receipts	aggr	250
2003	Disbursements	payment of accounts	det	120
2003	Disbursements	capital expenditure	det	0
2003	Disbursements	long-term financing	det	40
2003	Disbursements	total disbursements	aggr	160
2003	Balance	net cash inflow	drv	60
2003	Balance	ending cash balance	drv	80
2004	Receipts	beginning cash	drv	80
2004	Receipts	cash sales	det	100
2004	Receipts	receivables	det	100
2004	Receipts	total cash receipts	aggr	200
2004	Disbursements	payment of accounts	det	130
2004	Disbursements	capital expenditure	det	40
2004	Disbursements	long-term financing	det	20
2004	Disbursements	total disbursements	aggr	190
2004	Balance	net cash inflow	drv	10
2004	Balance	ending cash balance	drv	90

Table 1. A cash budget

1. for each section and year, the sum of the values of all *detail* items must be equal to the value of the *aggregate* item of the same section and year;
2. for each year, the net cash inflow must be equal to the difference between total cash receipts and total disbursements;
3. for each year, the ending cash balance must be equal to the sum of the beginning cash and the net cash balance.

Table 1 was acquired by means of an OCR tool from two paper documents, reporting the cash budget for 2003 and 2004. The original paper document was consistent, but some symbol recognition errors occurred during the digitizing phase, as constraints 1) and 2) are not satisfied on the acquired data for year 2003, that is:

i) in section *Receipts*, the aggregate value of *total cash receipts* is not equal to the sum of detail values of the same section.
ii) the value of *net cash inflow* is not to equal the difference between *total cash receipts* and *total disbursements*.

In order to exploit the digital version of the cash budget, a fundamental issue is to define a reasonable strategy for locating OCR errors, and then “repairing” the acquired data to extract reliable information.

Most of well-known techniques for repairing data violating either key constraints or functional dependencies accomplish this task by performing deletions and insertions of tuples. Indeed this approach is not suitable for contexts analogous to that of Example 1, that is of data acquired by OCR tools from paper documents. For instance, repairing Table 1 by either adding or removing rows means hypothesizing that the OCR tool either jumped a row or "invented" it when acquiring the source paper document, which is rather unrealistic. The same issue arises in other scenarios dealing with numerical data representing pieces of information acquired automatically, such as sensor networks. In a sensor network with error-free communication channels, no reading generated by sensors can be lost, thus repairing the database by adding new readings (as well as removing collected ones) is of no sense. In this kind of scenario, the most natural approach to data repairing is updating directly the numerical data: this means working at attribute-level, rather than at tuple-level. For instance, in the case of Example 1, we can reasonably assume that inconsistencies of digitized data are due to symbol recognition errors, and thus trying to re-construct actual data values is well founded. Likewise, in the case of sensor readings violating aggregate constraints, we can hypothesize that inconsistency is due to some trouble occurred at a sensor while generating some reading, thus repairing data by modifying readings instead of deleting (or inserting) them is justified.

1.1 Related Work

First theoretical approaches to the problem of dealing with incomplete and inconsistent information date back to 80s, but these works mainly focus on issues related to the semantics of incompleteness [12]. The problem of extracting reliable information from inconsistent data was first addressed in [4], where an extension of relational algebra (namely *flexible algebra*) was proposed to evaluate queries on data inconsistent w.r.t. key constraints (i.e. tuples having the same values for key attributes, but conflicting values for other attributes). The first proof-theoretic notion of *consistent query answer* was introduced in [6], expressing the idea that tuples involved in an integrity violation should not be considered in the evaluation of consistent query answering. In [1] a different notion of consistent answer was introduced, based on the notion of *repair*: a repair of an inconsistent database D is a database D' satisfying the given integrity constraints and which is minimally different from D. Thus, the consistent answer of a query q posed on D is the answer which is in every result of q on each repair D'. In particular, in [1] the authors show that, for restricted classes of queries and constraints, consistent answers can be evaluated without computing repairs, but by looking only at the specified constraints and rewriting the original query q into a query q' such that the answer of q' on D is equal to the consistent answer of q on D. Based on the notions of repair and consistent query answer introduced in [1], several works investigated more expressive classes of queries and constraints. In [2] extended disjunctive logic programs with exceptions were used for the computation of repairs, and in [3] the evaluation of aggregate queries on inconsistent data was investigated. A further generalization was proposed in [11], where the authors defined a technique based on the rewriting of constraints into extended disjunctive rules with two different forms of negation (negation as failure and classical negation). This technique was shown to be sound and complete for universally quantified constraints.

All the above-cited approaches assume that tuple insertions and deletions are the basic primitives for repairing inconsistent data. More recently, in [9] a repairing strategy using only tuple deletions was proposed, and in [17] repairs also consisting of update operations were considered. The latter is the first approach performing repairs at the attribute-value level, but is not well-suited in our context, as it works only in the case that constraints consist of full dependencies.

The first work investigating aggregate constraints on numerical data is [16], where the consistency problem of very general forms of aggregation is considered, but no issue related to data-repairing is investigated. In [5] the problem of repairing databases by fixing numerical data at attribute level is investigated. The authors show that deciding the existence of a repair under both denial constraints (where built-in comparison predicates are allowed) and a non-linear form of multi-attribute aggregate constraints is undecidable. Then they disregard aggregate constraints and focus on the problem of repairing data violating denial constraints, where no form of aggregation is allowed in the adopted constraints.

1.2 Main Contribution

We investigate the problem of repairing and extracting reliable information from data violating a given set of aggregate constraints. These constraints consist of linear inequalities on aggregate-sum queries issued on measure values stored in the database. This syntactic form enables meaningful constraints to be expressed, such as those of Example 1 as well as other forms which often occur in practice.

We consider database repairs consisting of “reasonable” sets of value-update operations aiming at re-constructing the correct measure values of inconsistent data. We adopt two different criteria for determining whether a set of update operations repairing data can be considered “reasonable” or not: *set*-minimal semantics and *card*-minimal semantics. Both these semantics aim at preserving the information represented in the source data as much as possible. They correspond to different repairing strategies which turn out to be well-suited for different application scenarios.

We provide the complexity characterization of three fundamental problems: i) *repairability* (is there at least one repair for the given database w.r.t. the specified constraints?); ii) *repair checking* (given a set of update operations, is it a “reasonable” repair?); iii) *consistent query answer* (is a given boolean query true in every “reasonable” repair?).

2 Preliminaries

We assume classical notions of database scheme, relational scheme, and relations. In the following we will also use a logical formalism to represent relational databases, and relational schemes will be represented by means of sorted predicates of the form $R(A_1 : \Delta_1, \ldots, A_n : \Delta_n)$, where $A_1, \ldots, A_n$ are attribute names and $\Delta_1, \ldots, \Delta_n$ are the corresponding domains. Each Δ_i can be either $\mathbb{Z}$ (infinite domain of integers), $\mathbb{R}$ (reals), or $\mathbb{S}$ (strings). Domains $\mathbb{R}$ and $\mathbb{Z}$ will be said to be *numerical domains*, and attributes defined over $\mathbb{R}$ or $\mathbb{Z}$ will be said to be *numerical attributes*. Given a ground atom t denoting a tuple, the value of attribute A of t will be denoted as $t[A]$.

Given a database scheme $\mathcal{D}$, we will denote as $\mathcal{M}_{\mathcal{D}}$ (namely, *Measure attributes*) the set of numerical attributes representing measure data. That is, $\mathcal{M}_{\mathcal{D}}$ specifies the set of attributes representing measure values, such as weights, lengths, prices, etc. For instance, in Example 1, $\mathcal{M}_{\mathcal{D}}$ consists of the only attribute *Value*.

Given two sets M, M', $M \triangle M'$ denotes their symmetric difference $(M \cup M') \setminus (M \cap M')$.

2.1 Aggregate constraints

Given a relational scheme $R(A_1 : \Delta_1, \ldots, A_n : \Delta_n)$, an *attribute expression* on R is defined recursively as follows:

- a numerical constant is an attribute expression;
- each A_i (with $i \in [1..n]$) is an attribute expression;
- $e_1 \psi e_2$ is an attribute expression on R, if e_1, e_2 are attribute expressions on R and ψ is an arithmetic operator in $\{+, -\}$;
- $c \times (e)$ is an attribute expressions on R, if e is an attribute expression on R and c a numerical constant.

Let R be a relational scheme, e an attribute expression on R, and C a boolean formula on constants and attributes of R. An *aggregation function* on R is a function $\chi : (\Delta_1 \times \cdots \times \Delta_k) \rightarrow \mathbb{R}$, where $\Delta_1, \ldots, \Delta_k$ are the relational domains of some attributes $A_1, \ldots, A_k$ of R. $\chi(x_1, \ldots, x_k)$ is defined as follows:

$$\chi(x_1, \ldots, x_k) = \begin{array}{ll} \texttt{SELECT} & \texttt{sum(e)} \\ \texttt{FROM} & \texttt{R} \\ \texttt{WHERE} & \alpha(\texttt{x}_1, \ldots, \texttt{x}_\texttt{k}) \end{array}$$

where $\alpha(x_1, \ldots, x_k) = C \wedge \ (A_1 = x_1) \ \wedge \cdots \wedge \ (A_k = x_k)$.

Example 2. The following aggregation functions are defined on the relational scheme *CashBudget(Year, Section, Subsection, Type, Value)* of Example 1:

$$\chi_1(x, y, z) = \begin{array}{ll} \texttt{SELECT} & \texttt{sum(Value)} \\ \texttt{FROM} & \texttt{CashBudget} \\ \texttt{WHERE} & \texttt{Section} = \texttt{x} \\ \multicolumn{2}{l}{\quad \texttt{AND Year} = \texttt{y AND Type} = \texttt{z}} \end{array} \qquad \chi_2(x, y) = \begin{array}{ll} \texttt{SELECT} & \texttt{sum(Value)} \\ \texttt{FROM} & \texttt{CashBudget} \\ \texttt{WHERE} & \texttt{Year} = \texttt{x} \\ \multicolumn{2}{l}{\quad \texttt{AND Subsection} = \texttt{y}} \end{array}$$

Function χ_1 returns the sum of *Value* of all the tuples having *Section* x, *Year* y and *Type* z. For instance, $\chi_1(\text{'Receipts', '2003', 'det'})$ returns $100 + 120 = 220$, whereas $\chi_1(\text{'Disbursements', '2003', 'aggr'})$ returns 160. Function χ_2 returns the sum of *Value* of all the tuples where *Year=x* and *Subsection=y*. In our running example, as the pair *Year*, *Subsection* uniquely identifies tuples of *CashBudget*, the sum returned by χ_2 coincides with a single value. For instance, $\chi_2(\text{'2003', 'cash sales'})$ returns 100, whereas $\chi_2(\text{'2004', 'net cash inflow'})$ returns 10.

Definition 1 (Aggregate constraint). *Given a database scheme $\mathcal{D}$, an aggregate constraint on $\mathcal{D}$ is an expression of the form:*

$$\forall x_1, \ldots, x_k \ \left(\phi(x_1, \ldots, x_k) \implies \sum_{i=1}^{n} c_i \cdot \chi_i(X_i) \leq K \right) \tag{1}$$

where:

1. $c_1, \ldots, c_n, K$ *are constants;*
2. $\phi(x_1, \ldots, x_k)$ *is a conjunction of atoms containing the variables* $x_1, \ldots, x_k$*;*
3. *each* $\chi_i(X_i)$ *is an aggregation function, where* X_i *is a list of variables and constants, and variables appearing in* X_i *are a subset of* $\{x_1, \ldots, x_k\}$.

Given a database D and a set of aggregate constraints $\mathcal{AC}$, we will use the notation $D \models \mathcal{AC}$ [resp. $D \not\models \mathcal{AC}$] to say that D is consistent [resp. inconsistent] w.r.t. $\mathcal{AC}$. Observe that aggregate constraints enable equalities to be expressed as well, since an equality can be viewed as a pair of inequalities. For the sake of brevity, in the following equalities will be written explicitly.

Example 3. Constraint 1 defined in Example 1 can be expressed as follows:
$\forall\, x, y, s, t, v \quad CashBudget(y, x, s, t, v) \implies \chi_1(x, y, \text{‘det’}) - \chi_1(x, y, \text{‘aggr’}) = 0$

For the sake of simplicity, in the following we will use a shorter notation for denoting aggregate constraints, where universal quantification is implied and variables in ϕ which do not occur in any aggregation function are replaced with the symbol ‘_’. For instance, the constraint of Example 3 can be written as follows:
$CashBudget(y, x, \_, \_, \_) \implies \chi_1(x, y, \text{‘}det\text{’}) - \chi_1(x, y, \text{‘}aggr\text{’}) = 0$

Example 4. Constraints 2 and 3 defined in Example 1 can be expressed as follows:
Constraint 2: $CashBudget(x, \_, \_, \_, \_) \implies$
$\chi_2(x, \text{‘net cash inflow’}) - (\chi_2(x, \text{‘total cash receipts’}) - \chi_2(x, \text{‘total disbursements’})) = 0$
Constraint 3: $CashBudget(x, \_, \_, \_, \_) \implies$
$\chi_2(x, \text{‘ending cash balance’}) - (\chi_2(x, \text{‘beginning cash’}) + \chi_2(x, \text{‘net cash balance’})) = 0$
Consider the database scheme consisting of relation *CashBudget* and relation *Sales(Product, Year, Income)*, containing pieces of information on annual product sales. The following aggregate constraint says that, for each year, the value of *cash sales* in *CashBudget* must be equal to the total incomes obtained from relation *Sales*:
CashBudget $(x, \_, \_, \_, \_) \wedge$ Sales$(_, x, _) \implies \chi_2(x, \text{‘cash sales’}) - \chi_3(x) = 0$
where $\chi_3(x)$ is the aggregation function returning the total income due to products sales in year x:
$\chi_3(x) =$
```
SELECT sum(Income)
FROM   Sales
WHERE  Year = x
```

2.2 Updates

Updates at attribute-level will be used in the following as the basic primitives for repairing data violating aggregate constraints. Given a relational scheme R in the database scheme $\mathcal{D}$, let $\mathcal{M}_R = \{A_1, \ldots, A_k\}$ be the subset of $\mathcal{M}_\mathcal{D}$ containing all the attributes in R belonging to $\mathcal{M}_\mathcal{D}$.

Definition 2 (Atomic update). *Let* $t = R(v_1, \ldots, v_n)$ *be a tuple on the relational scheme* $R(A_1 : \Delta_1, \ldots, A_n : \Delta_n)$. *An* atomic update *on* t *is a triplet* $< t, A_i, v'_i >$, *where* $A_i \in \mathcal{M}_R$ *and* v'_i *is a value in* Δ_i *and* $v'_i \neq v_i$.

Update $u = < t, A_i, v'_i >$ replaces $t[A_i]$ with v'_i, thus yielding the tuple $u(t) = R(v_1, \ldots, v_{i-1}, v'_i, v_{i+1}, \ldots, v_n)$.

Observe that atomic updates work on the set $\mathcal{M}_R$ of measure attributes, as our framework is based on the assumption that data inconsistency is due to errors in the acquisition phase (as in the case of digitization of paper documents) or in the measurement phase (as in the case of sensor readings). Therefore our approach will only consider repairs aiming at re-constructing the correct measures.

Example 5. Update $u = < t, \mathit{Value}, 130 >$ issued on tuple $t = \text{CashBudget}(2003, \text{Receipts}, \text{cash sales}, \text{det}, 100)$ returns $u(t) = \text{CashBudget}(2003$, Receipts, cash sales, det, 130$)$.

Given an update u, we denote the attribute updated by u as $\lambda(u)$. That is, if $u = < t,\ A_i,\ v >$ then $\lambda(u) =< t, A_i >$.

Definition 3 (Consistent database update). *Let D be a database and $U = \{u_1, \ldots, u_n\}$ be a set of atomic updates on tuples of D. The set U is said to be a* consistent database update *iff $\forall\ j, k \in [1..n]$ if $j \neq k$ then $\lambda(u_j) \neq \lambda(u_k)$.*

Informally, a set of atomic updates U is a consistent database update iff for each pair of updates $u_1, u_2 \in U$, u_1 and u_2 do not work on the same tuples, or they change different attributes of the same tuple.

The set of pairs $< \mathit{tuple}, \mathit{attribute} >$ updated by a consistent database update U will be denoted as $\lambda(U) = \cup_{u_i \in U} \lambda(u_i)$.

Given a database D and a consistent database update U, the result of performing U on D consists in the new database $U(D)$ obtained by performing all atomic updates in U.

3 Repairing inconsistent databases

Definition 4 (Repair). *Let $\mathcal{D}$ be a database scheme, $\mathcal{AC}$ a set of aggregate constraints on $\mathcal{D}$, and D an instance of $\mathcal{D}$ such that $D \not\models \mathcal{AC}$. A* repair *$\rho$ for D is a consistent database update such that $\rho(D) \models \mathcal{AC}$.*

Example 6. A repair ρ for *CashBudget* w.r.t. constraints 1), 2) and 3) consists in decreasing attribute *Value* in the tuple $t =$ CashBudget(2003, Receipts, total cash receipts, aggr, 250) down to 220; that is, $\rho = \{ < t, \mathit{Value}, 220 > \}$.

We now characterize the complexity of the repair-existence problem. All the complexity results in the paper refer to data-complexity, that is the size of the constraints is assumed to be bounded by a constant.

The following lemma is a preliminary result which states that potential repairs for an inconsistent database can be found among set of updates whose size is polynomially bounded by the size of the original database.

Lemma 1. *Let $\mathcal{D}$ be a database scheme, $\mathcal{AC}$ a set of aggregate constraints on $\mathcal{D}$, and D an instance of $\mathcal{D}$ such that $D \not\models \mathcal{AC}$. If there is a repair ρ for D w.r.t. $\mathcal{AC}$, then there is a repair ρ' for D such that $\lambda(\rho') \subseteq \lambda(\rho)$ and ρ' has polynomial size w.r.t. D.*

Proof. (sketch) W.l.o.g. we assume that the attribute expression e_{χ_i} occurring in each aggregate function χ_i in $\mathcal{AC}$ is either an attribute or a constant. Let ρ be a repair for D, and $\mathcal{AC}^*$ be the set of inequalities obtained as follows:

1. a variable $x_{t,A}$ is associated to each pair $< t, A > \in \lambda(\rho)$;
2. for every constraint in $\mathcal{AC}$ of the form (1) and for every ground substitution θ of $x_1, \ldots, x_k$ s.t. $\phi(\theta x_1, \ldots, \theta x_k)$ is true, the following inequalities are added to $\mathcal{AC}^*$:

a. $\sum_{i=1}^{n} c_i \cdot \sum_{<t,e_{\chi_i}> \in \lambda(\rho) \wedge t \models \alpha_i(\theta x_1,\ldots,\theta x_k)} x_{t,e_{\chi_i}} \leq K'$, where K' is K minus the contribution to the left-hand side of the constraint due to values which have not been changed by ρ, i.e. $K' = K - \sum_{i=1}^{n} c_i \cdot \sum_{<t,e_{\chi_i}> \notin \lambda(\rho) \wedge t \models \alpha_i(\theta x_1,\ldots,\theta x_k)} e_{\chi_i}$.

b. for each tuple t such that $t \models \alpha_i(\theta x_1, \ldots, \theta x_k)$, let α'_i be the disjunctive normal form of α_i and let β be a disjunct in α'_i such that $t \models \beta(\theta x_1, \ldots, \theta x_k)$. For each conjunct γ in β of the form $w_1 \diamond w_2$, where $\diamond$ is a comparison operator, and either w_1 or w_2 is an attribute A such that $< t, A > \in \lambda(\rho)$, the constraint $v_1 \diamond v_2$ is added to $\mathcal{AC}^*$, where, for $\jmath \in \{1, 2\}$ $^{1)}$ if w_j is constant, $v_j = w_j$; $^{2)}$ if $w_j = A$ and $< t, A > \in \lambda(\rho)$, $v_j = x_{t,A}$; $^{3)}$ if $w_j = A$ and $< t, A > \notin \lambda(\rho)$, $v_j = t[A]$.

Obviously $\mathcal{AC}^*$ has one solution, which corresponds to assigning to each variable x_{t,A_i} the value assigned by ρ to attribute A_i of tuple t. Moreover, the number of variables and equations, and the size of constants in $\mathcal{AC}^*$ are polynomially bounded by the size of D. Therefore there is a solution $\overline{X}$ to $\mathcal{AC}^*$ whose size is polynomially bounded by the size of D, since $\mathcal{AC}^*$ is a PLI problem with at least one solution [14]. $\overline{X}$ defines a repair ρ' for D such that $\lambda(\rho') \subseteq \lambda(\rho)$ and ρ' has polynomial size w.r.t. D. □

Theorem 1 (Repair existence). *Let $\mathcal{D}$ be a database scheme, $\mathcal{AC}$ a set of aggregate constraints on $\mathcal{D}$, and D an instance of $\mathcal{D}$ such that $D \not\models$ AC. The problem of deciding whether there is a repair for D is NP-complete.*

Proof. Membership. A polynomial size witness for deciding the existence of a repair is a database update U on D: testing whether U is a repair for D means verifying $U(D) \models \mathcal{AC}$, which can be accomplished in polynomial time w.r.t. the size of D and U. If a repair exists for D, then Lemma 1 guarantees that a polynomial size repair for D exists too.
Hardness. We show a reduction from CIRCUIT SAT to our problem. Without loss of generality, we consider a boolean circuit C using only NOR gates. The inputs of C will be denoted as $x_1, \ldots, x_n$. The boolean circuit C can be represented by means of the database scheme:

$gate(\underline{IDGate}, norVal, orVal)$,
$gateInput(\underline{IDGate, IDIngoing}, Val)$,
$input(\underline{IDInput}, Val)$.

Therein:

1. each gate in C corresponds to a tuple in $gate$ (attributes *norVal* and *orVal* represent the output of the corresponding NOR gate and its negation, respectively);
2. inputs of C correspond to tuples of *input*: attribute *Val* in a tuple of *input* represents the truth assignment to the input $x_{IDInput}$;
3. each tuple in *gateInput* represents an input of the gate identified by *IDGate*. In particular, *IDIngoing* refers to either a gate identifier or an input identifier; attribute *Val* is a copy of the truth value of the specified ingoing gate or input.

We consider the database instance D where the relations defined above are populated as follows. For each input x_i in C we insert the tuple $input(id(x_i), -1)$ into D, and for each gate g in C we insert the tuple $gate(id(g), -1, -1)$, where function $id(x)$ assigns a unique identifier to its argument (we assume that gate identifiers are distinct from input identifiers, and that the output gate of C is assigned the identifier 0). Moreover, for each edge in C going from g' to the gate g (where g' is either a gate or an input of C), the tuple $gateInput(id(g), id(g'), -1)$ is inserted into D. Assume that $\mathcal{M}_{gate} = \{norVal, orVal\}$, $\mathcal{M}_{gateInput} = \{Val\}$, $\mathcal{M}_{input} = \{Val\}$. In the following, we will define aggregate constraints to force measure attributes of all tuples to be assigned either 1 or 0, representing the truth value *true* and *false*, respectively. The initial assignment (where every measure attribute is set to -1) means that the truth values of inputs and gate outputs is undefined.

Consider the following aggregation functions:

$NORVal(X) =$ `SELECT Sum(norVal) FROM gate WHERE (IDGate = X)`

$ORVal(X) =$ `SELECT Sum(orVal) FROM gate WHERE (IDGate = X)`

$IngoingVal(X, Y) =$ `SELECT Sum(Val) FROM gateInput WHERE (IDGate = X) AND (IDIngoing = Y)`

$IngoingSum(X) =$ `SELECT Sum(Val) FROM gateInput WHERE (IDGate = X)`

$InputVal(X) =$ `SELECT Sum(Val) FROM Input WHERE (IDInput = X)`

$ValidInput(\,) =$ `SELECT Sum(1) FROM input WHERE (Val ≠ 0) AND (Val ≠ 1)`

$ValidGate(\,) =$ `SELECT Sum(1) FROM gate WHERE (orVal ≠ 0 AND orVal ≠ 1) OR (norVal ≠ 0 AND norVal ≠ 1)`

Therein: $NORVal(X)$ and $ORVal(X)$ return the truth value of the gate X and its opposite, respectively; $IngoingVal(X, Y)$ returns, for the gate with identifier X, the truth value of the ingoing gate or input having identifier Y; $IngoingSum(X)$ returns the sum of the truth values of the inputs of the gate X; $InputVal(X)$ returns the truth assignment of the input X; $ValidInput(\,)$ returns 0 iff there is no tuple in relation $input$ where attribute Val is neither 0 nor 1, otherwise it returns a number greater than 0; likewise, $ValidGate(\,)$ returns 0 iff there is no tuple in relation $gate$ where attributes $norVal$ or $orVal$ are neither 0 nor 1 (otherwise it returns a number greater than 0).

Consider the following aggregate constraints on $\mathcal{D}$:

1. $ValidInput(\,) + ValidGate(\,) = 0$, which entails that only 0 and 1 can be assigned either to attributes $orVal$ and $norVal$ in relation $gate$, and to attribute Val in relation $input$;
2. $gate(X, \_, \_) \Rightarrow ORVal(X) + NORVal(X) = 1$, which says that for each tuple representing a NOR gate, the value of $orVal$ must be complementary to $norVal$;

3. $gate(X, \_, \_) \Rightarrow ORVal(X) - IngoingSum(X) \leq 0$, which says that for each tuple representing a NOR gate, the value of $orVal$ cannot be greater than the sum of truth assignments of its inputs (i.e. if all inputs are 0, $orVal$ must be 0 too);
4. $gateInput(X, Y, \_) \Rightarrow IngoingVal(X, Y) - ORVal(X) \leq 0$, which implies that, for each gate g, attribute $orVal$ must be 1 if at least one input of g has value 1;
5. $gateInput(X, Y, \_) \Rightarrow IngoingVal(X, Y) - NORVal(Y) - InputVal(Y) = 0$, which imposes that the attribute Val in each tuple of $gateInput$ is the same as the truth value of either the ingoing gate or the ingoing input.

Observe that D does not satisfy these constraints, but every repair of D corresponds to a valid truth assignment of C.

Let $\mathcal{AC}$ be the set of aggregate constraints consisting of constraints 1-5 defined above plus constraint $NORVal(0) = 1$ (which imposes that the truth value of the output gate must be *true*). Therefore, deciding whether there is a truth assignment which evaluates C to $true$ is equivalent to asking whether if there is a repair ρ for D w.r.t. $\mathcal{AC}$. $\square$

Remark. Theorem 1 states that the repair existence problem is decidable. This result, together with the practical usefulness of the considered class of constraints, makes the complexity analysis of finding consistent answers on inconsistent data interesting. Basically decidability results from the linear nature of the considered constraints. If products between two attributes were allowed as attribute expressions, the repair-existence problem would be undecidable (this can be proved straightforwardly, since this form of non-linear constraints is more expressive than those introduced in [5], where the corresponding repair-existence problem was shown to be undecidable). However, observe that occurrences of products of the form $A_i \times A_j$ in attribute expressions can lead to undecidability only if both A_i and A_j are measure attribute. Otherwise, this case is equivalent to products of the form $c \times A$, which can be expressed in our form of aggregate constraints.

3.1 Minimal repairs

Theorem 1 deals with the problem of deciding whether a database D violating a set of aggregate constraints $\mathcal{AC}$ can be repaired. If this is the case, different repairs can be performed on D yielding a new database consistent w.r.t. $\mathcal{AC}$, although not all of them can be considered "reasonable". For instance, if a repair exists for D changing only one value in one tuple of D, any repair updating all values in all tuples of D can be reasonably disregarded. To evaluate whether a repair should be considered "relevant" or not, we introduce two different ordering criteria on repairs, corresponding to the comparison operators '$\leq_{set}$' and '$\leq_{card}$'. The former compares two repairs by evaluating whether one of the two performs a subset of the updates of the other. That is, given two repairs ρ_1, ρ_2, we say that ρ_1 precedes ρ_2 ($\rho_1 \leq_{set} \rho_2$) iff $\lambda(\rho_1) \subseteq \lambda(\rho_2)$. The latter ordering criterion states that a repair ρ_1 is preferred w.r.t. a repair ρ_2 ($\rho_1 \leq_{card} \rho_2$) iff $|\lambda(\rho_1)| \leq |\lambda(\rho_2)|$, that is if the number of changes issued by ρ_1 is less than ρ_2.

Observe that $\rho_1 <_{set} \rho_2$ implies $\rho_1 <_{card} \rho_2$, but the vice versa does not hold, as it can be the case that repair ρ_1 changes a set of values $\lambda(\rho_1)$ which is not subset of $\lambda(\rho_2)$, but having cardinality less than $\lambda(\rho_2)$.

Example 7. Another repair for *CashBudget* is $\rho' = \{\langle t_1, \mathit{Value}, 130\rangle, \langle t_2, \mathit{Value}, 70\rangle, \langle t_3, \mathit{Value}, 190\rangle\}$, where $t_1 =$ CashBudget(2003, Receipts, cash sales, det, 100), $t_2 =$ CashBudget(2003, Disbursements, long-term financing, det, 40), and $t_3 =$ CashBudget (2003, Disbursements, total disbursements, aggr, 160).
Observe that $\rho <_{\text{card}} \rho'$, but not $\rho <_{set} \rho'$ (where ρ is the repair defined in Example 6).

Definition 5 (Minimal repairs). *Let $\mathcal{D}$ be a database scheme, $\mathcal{AC}$ a set of aggregate constraints on $\mathcal{D}$, and D an instance of $\mathcal{D}$. A repair ρ for D w.r.t. $\mathcal{AC}$ is a* set-*minimal repair [resp.* card-*minimal repair] iff there is no repair ρ' for D w.r.t. $\mathcal{AC}$ such that $\rho' <_{set} \rho$ [resp. $\rho' <_{card} \rho$].*

Example 8. Repair ρ of Example 6 is minimal under both the *set*-minimal and the *card*-minimal semantics, whereas ρ' defined in Example 7 is minimal only under the *set*-minimal semantics.
Consider the repair ρ'' consisting of the following updates: $\rho'' = \{\langle t_1, \mathit{Value}, 110\rangle, \langle t_2, \mathit{Value}, 110\rangle, \langle t_3, \mathit{Value}, 220\rangle\}$ where: $t_1 =$ CashBudget(2003, Receipts, cash sales, det, 100), $t_2 =$ CashBudget(2003, Receipts, receivables, det, 120), $t_3 =$ CashBudget(2003, Receipts, total cash receipts, aggr, 250).
The strategy adopted by ρ'' can be reasonably disregarded, since the only atomic update on tuple t_3 suffices to make D consistent. In fact, ρ'' is not minimal neither under the *set*-minimal semantics (as $\lambda(\rho) \subset \lambda(\rho'')$ and thus $\rho <_{\text{set}} \rho''$) nor under the *card*-minimal one.

Given a database D which is not consistent w.r.t. a set of aggregate constraints $\mathcal{AC}$, different *set*-minimal repairs (resp. *card*-minimal repairs) can exist on D. In our running example, repair ρ of Example 6 is the unique *card*-minimal repair, and both ρ and ρ' are *set*-minimal repairs (where ρ' is the repair defined in Example 7). The set of *set*-minimal repairs and the set of *card*-minimal repairs will be denoted, respectively, as ρ_M^{set} and ρ_M^{card}.

Theorem 2 (Minimal-repair checking). *Let $\mathcal{D}$ be a database scheme, $\mathcal{AC}$ a set of aggregate constraints on $\mathcal{D}$, and D be an instance of $\mathcal{D}$ such that $D \not\models \mathcal{AC}$. Given a repair ρ for D w.r.t. $\mathcal{AC}$, deciding whether ρ is minimal (under both the* card-*minimality and* set-*minimality semantics) is coNP-complete.*

Proof. (Membership) A polynomial size witness for the complement of the problem of deciding whether $\rho \in \rho_M^{set}$ [resp. $\rho \in \rho_M^{card}$] is a repair ρ' such that $\rho' <_{set} \rho$ [resp. $\rho' <_{card} \rho$]. From Lemma 1 we have that ρ' can be found among repairs having polynomial size w.r.t. D.
(Hardness) We show a reduction of MINIMAL MODEL CHECKING (MMC) [7] to our problem. Consider an instance $\langle f, M\rangle$ of MMC, where f is a propositional formula and M a model for f. Formula f can be translated into an equivalent boolean circuit C using only NOR gates, and C can be represented as shown in the hardness proof of Theorem 1. Therefore, we consider the same database scheme $\mathcal{D}$ and the same set of aggregate constraints $\mathcal{AC}$ on $\mathcal{D}$ as those in the proof of Theorem 1. Let D be the instance of $\mathcal{D}$ constructed as follows. For each input x_i in C we insert the tuple $input(id(x_i), 0)$ into D. Then, as for the construction in the hardness proof of Theorem 1, for each gate g in C we insert the tuple $gate(id(g), -1, -1)$ into D, and for each

edge in C going from g' to the gate g (where g' is either a gate or an input of C), the tuple $gateInput(id(g), id(g'), -1)$ is inserted into D.

Observe that any repair for D must update all measure attributes in D with value -1. Therefore, given two repairs ρ', ρ'', it holds that for each $< t, A > \in\ (\lambda(\rho') \triangle \lambda(\rho''))$, t is a tuple of $input$ and $A = Val$.

Obviously, a repair ρ for D exists, consisting of the following updates: 1) attribute Val is assigned 1 in every tuple of $input$ corresponding to an atom in f which is true in m; 2) attributes $norVal$, $orVal$ in $gate$ and Val in $gateInput$ are updated accordingly to updates described above. Basically, such a constructed repair ρ corresponds to M (we say that a repair corresponds to a model if it assigns 1 to attribute Val in the tuples of $input$ corresponding to the atoms which are true in the model, 0 otherwise).

If M is not a minimal model for f, then there exists a model M' such that $M' \subset M$ (i.e. atoms which are true in M' are a proper subset of atoms which are true in M). Then, the repair ρ' corresponding to M' satisfies $\rho' <_{set} \rho$. Vice versa, if there exists a repair ρ' such that $\rho' <_{set} \rho$, then the model M' corresponding to ρ' is a proper subset of M, thus M is not minimal. This proves that M is a minimal model for f iff ρ is a minimal repair (under *set*-minimal semantics) for D w.r.t $\mathcal{AC}$.

Proving hardness under *card*-minimal semantics can be accomplished as follows. First, a formula f_M is constructed from f by replacing, for each atom $a \notin M$, each occurrence of a in f with the contradiction $(a \wedge \neg a)$. Then, an instance D of $\mathcal{D}$ is constructed corresponding to formula f_M with the same value assignments as before (attribute Val in all the tuples of $input$ are set to 0, and all the other measure attributes are set to -1).

M is a model for both f and f_M, and it is minimal for f iff it is minimum for f_M. In fact, if M is minimal for f there is no subset M' of M which is a model of f. Then, assume that a model M'' for f_M exists, such that $|M''| < |M|$. Then, also $M''' = M'' \cap M$ is a model for f_M, implying that M''' is a model for f, which is a contradiction (as $M''' \subset M$). On the other hand, if M is minimum for f_M then M must be minimal for f. Otherwise, there would exist a model M' for f s.t. $M' \subset M$. However M' is also a model for f_M, which is a contradiction, as $|M'| < |M|$.

Let ρ be the repair of D w.r.t. $\mathcal{AC}$ corresponding to M. If M is not minimum, then there exists M' (with $|M'| < |M|$) which is a model for f_M. Therefore the repair ρ' corresponding to M' satisfies $\rho' <_{card} \rho$. Vice versa, if a repair ρ' for D w.r.t. $\mathcal{AC}$ exists such that $\rho' <_{card} \rho$, then the model M' corresponding to ρ' is such that $|M'| < |M|$, thus M is not minimum for f_M. This proves that M is a minimal model for f iff there is no repair ρ' for D w.r.t. $\mathcal{AC}$ such that $\rho' <_{card} \rho$. $\square$

Set-minimality vs card-minimality

Basically, both the *set*-minimal and the *card*-minimal semantics aim at considering "reasonable" repairs which preserve the content of the input database as much as possible. To the best of our knowledge the notion of repair minimality based on the number of performed updates has not been used in the context of relational data violating "non-numerical" constraints (such as keys, foreign keys, and functional dependencies). In this context, most of the proposed approaches consider repairs consisting of deletions and insertions of tuples, and preferred repairs are those consisting of minimal sets of insert/delete operations. In fact, the *set*-minimal semantics is more natural than the *card*-minimal one when no hypothesis can be reasonably formulated to "guess" how data inconsistency occurred, which is the case of previous works on database-repairing. As it will be clear in the following, in the general case, the adoption of the *card*-minimal semantics could make reasonable sets of delete/insert operations to be not considered as candidate repairs, even if they correspond to error configurations which cannot be excluded.

For instance, consider a relational scheme *Department(Name, Area, Employers, Category)* where the following functional dependencies are defined: $FD_1 : Area \rightarrow Employers$ (i.e. departments having the same area must have the same number of employers) and $FD_2 : Employers \rightarrow Category$ (i.e. departments with the same number of employers must be of the same category). Consider the following relation:

Department	***Area***	***Employers***	***Category***	
D_1	100	24	A	$\longrightarrow t_1$
D_2	100	30	B	$\longrightarrow t_2$
D_3	100	30	B	$\longrightarrow t_3$

Relation above does not satisfy FD_1, as the three departments occupy the same area but do not have the same number of employers. Suppose we are using a repairing strategy based on deletions and insertions of tuples. Different repairs can be adopted. For instance, if we suppose that the inconsistency arises as tuple t_1 contains wrong information, *Department* can be repaired by only deleting t_1. Otherwise, if we assume that t_1 is correct, a possible repair consists of deleting t_2 and t_3. If the *card*-minimal semantics is adopted, the latter strategy will be disregarded, as it performs two deletions, whereas the former deletes only one tuple. On the contrary, if the *set*-minimal semantics is adopted, both the two strategies define minimal repairs (as the sets of tuples deleted by each of these strategies are not subsets of one another). In fact, if we do not know how the error occurred, there is no reason to assume that the error configuration corresponding to the second repairing strategy is not possible. Indeed, inconsistency could be due to integrating data coming from different sources, where some sources are not up-to-date. However, there is no good reason to assume that the source which contains the smallest number of tuples is the one that is up to date. See [13] for a survey on inconsistency due to data integration.

Likewise, the *card*-minimal semantics could disregard reasonable repairs also in the case that a repairing strategy based on updating values instead of deleting/inserting whole tuples is adopted[1]. For instance, if we suppose that the inconsistency arises as the value of attribute *Area* is wrong for either t_1 or both t_2 and t_3, *Department* can be repaired by replacing the *Area* value for either t_1 or both t_2 and t_3 with a value different from 100. Otherwise, if we assume that the *Area* values for all the tuples are correct, *Department* can be repaired w.r.t. FD_1 by making the *Employers* value of t_1 equal to that of t_2 and t_3. Indeed this update yields a relation which does not satisfy FD_2 (as $t_1[\textit{Category}] \neq t_2[\textit{Category}]$) so that another value update is necessary in order to make it consistent. Under the *card*-minimal semantics the latter strategy is disregarded, as it performs more than one value update, whereas the former changes only the *Area* value of one tuple. On the contrary, under the *set*-minimal semantics both the two strategies

[1] Value updates cannot be necessarily simulated as a sequence deletion/insertion, as this might not be minimal under set inclusion.

define minimal repairs (as the sets of updates issued by each of these strategies are not subsets of one another). As for the case explained above, disregarding the second repairing strategy is arbitrary, if we do not know how the error occurred.

Our framework addresses scenarios where also *card*-minimal semantics can be reasonable. For instance, if we assume that integrity violations are generated while acquiring data by means of an automatic or semi-automatic system (e.g. an OCR digitizing a paper document, a sensor monitoring atmospheric conditions, etc.), focusing on error configurations which can be repaired with the minimum number of updates is well founded. Indeed this corresponds to the case that the acquiring system made the minimum number of errors (e.g. bad symbol-recognition for an OCR, sensor troubles, etc.), which can be considered the most probable event.

In this work we discuss the existence of repairs, and their computation under both *card*-minimal and *set*-minimal semantics. The latter has to be preferred when no warranty is given on the accuracy of acquiring tools, and, more generally, when no hypothesis can be formulated on the cause of errors.

3.2 Consistent query answers

In this section we address the problem of extracting reliable information from data violating a given set of aggregate constraints. We consider boolean queries checking whether a given tuple belongs to a database, and adopt the widely-used notion of consistent query answer introduced in [1].

Definition 6 (Query). *A* query *over a database scheme $\mathcal{D}$ is a ground atom of the form $R(v_1, \ldots, v_n)$, where $R(A_1, \ldots, A_n)$ is a relational scheme in $\mathcal{D}$.*

Definition 7 (Consistent query answer). *Let $\mathcal{D}$ be a database scheme, D be an instance of $\mathcal{D}$, $\mathcal{AC}$ be a set of aggregate constraints on $\mathcal{D}$ and q be a query over $\mathcal{D}$. The* consistent query answer *of q on D under the* set-*minimal semantics [resp.* card-*minimal semantics] is true iff $q \in \rho(D)$ for each $\rho \in \rho_M^{set}$ [resp. for each $\rho \in \rho_M^{card}$].*

The consistent query answers of a query q issued on the database D under the *set*-minimal and *card*-minimal semantics will be denoted as $q^{set}(D)$ and $q^{card}(D)$, respectively.

Theorem 3 (Consistent query answer under *set*-minimal semantics). *Let $\mathcal{D}$ be a database scheme, D be an instance of $\mathcal{D}$, $\mathcal{AC}$ be a set of aggregate constraints on $\mathcal{D}$ and q be a query over D. Deciding whether $q^{set}(D) = true$ is Π_2^p-complete.*

Proof. See appendix. □

Theorem 4 (Consistent query answer under *card*-minimal semantics). *Let $\mathcal{D}$ be a database scheme, D be an instance of $\mathcal{D}$, $\mathcal{AC}$ be a set of aggregate constraints on $\mathcal{D}$ and q be a query over D. Deciding whether $q^{card}(D) = true$ is $\Delta_2^p[log\ n]$-complete.*

Proof. See appendix. □

Conclusions and Future Work

We have addressed the problem of repairing and extracting reliable information from numerical databases violating aggregate constraints, thus filling a gap in previous works dealing with inconsistent data, where only traditional forms of constraints (such as keys, foreign keys, etc.) were considered. In fact, aggregate constraints frequently occur in many real-life scenarios where guaranteeing the consistency of numerical data is mandatory. In particular, we have considered aggregate constraints defined as sets of linear inequalities on aggregate-sum queries on input data. For this class of constraints we have characterized the complexity of several issues related to the computation of consistent query answers.

Future work will be devoted to the identification of decidable cases when more expressive forms of constraint are adopted, that allow products between attribute values (as explained in the paper, enabling non-linear forms of aggregate expressions makes the repair-existence problem undecidable in the general case). Moreover the design of efficient algorithms for computing consistent answers will be addressed.

References

1. Arenas, M., Bertossi, L. E., Chomicki, J., Consistent Query Answers in Inconsistent Databases, *Proc. PODS*, 68–79, 1999.
2. Arenas, M., Bertossi, L. E., Chomicki, J., Specifying and Querying Database Repairs using Logic Programs with Exceptions *Proc. FQAS*, 27–41,2000.
3. Arenas, M., Bertossi, L. E., Chomicki, J., He, X., Raghavan, V., Spinrad, J., Scalar aggregation in inconsistent databases, *TCS*, Vol. 3(296), 405-434, 2003.
4. Agarwal, S., Keller, A. M., Wiederhold, G., Saraswat, K., Flexible Relation: An Approach for Integrating Data from Multiple, Possibly Inconsistent Databases, *Proc. ICDE*, 495–504, 1995.
5. Bertossi, L., Bravo, L., Franconi, E., Lopatenko, A., Fixing Numerical Attributes Under Integrity Constraints, Computing Research Repository, paper cs.DB/0503032.
6. Bry, F., Query Answering in Information Systems with Integrity Constraints, *IFIP IICIS*, 113–130, 1997.
7. Cadoli, M., Donini, F. M., Liberatore, P., Schaerf, M., Feasibility and unfeasibility of off-line processing. *Proc. ISTCS*, 100–109, 1996.
8. Chomicki, J., Marcinkowski, J., Staworko, S., Computing consistent query answers using conflict hypergraphs, *Proc. CIKM*, 417–426, 2004.
9. Chomicki, J., Marcinkowski, J., Minimal-Change Integrity Maintenance Using Tuple Deletions, *Information and Computation*, 90–121, 2005.
10. Eiter, T., Gottlob, G., On the Complexity of Propositional Knowledge Base Revision, Updates, and Counterfactual, *Artificial Intelligence*, 227–270, 1992.
11. Greco, G., Greco, S., Zumpano, E., A Logical Framework for Querying and Repairing Inconsistent Databases, *TKDE*, Vol. 15(6), 1389–1408, 2003.
12. Imielinski, T., Lipski, W., Incomplete Information in Relational Databases, *JACM*, Vol. 31(4), 761–791, 1984.
13. Lenzerini, M., Data Integration: A Theoretical Perspective, *Proc. PODS*, 233-246, 2002.
14. Papadimitriou, C. H., On the complexity of integer programming, *JACM*, Vol. 28(4), 765–768, 1981.
15. Papadimitriou, C. H., Computational Complexity, Addison-Wesley, 1994.
16. Ross, K. A., Srivastava, D., Stuckey, P. J., Sudarshan, S., Foundations of Aggregation Constraints, *TCS*, Vol. 193(1-2), 149–179, 1998.
17. Wijsen, J., Condensed Representation of Database Repairs for Consistent Query Answering, *Proc. ICDT*, 378–393, 2003.

Appendix: Proofs of theorems

Theorem 3. *Let $\mathcal{D}$ be a database scheme, D be an instance of $\mathcal{D}$, $\mathcal{AC}$ be a set of aggregate constraints on $\mathcal{D}$ and q be a query over D. Deciding whether $q^{set}(D) = true$ is Π_2^p-complete.*

Proof. (Membership) Membership in Π_2^p can be proved by reasoning as for Theorem 1, by exploiting a result similar to that of Lemma 1 (it can be proved that if there is a repair ρ s.t. $q(\rho(D))$ is true, then there is a repair ρ' having polynomial size w.r.t. q and D s.t. $\lambda(\rho') \subseteq \lambda(\rho)$).
(Hardness) Hardness can be proved by showing a reduction from the following implication problem in the context of propositional logic over a finite domain V, which was shown to be Π_2^p-complete in [10]: "*given an atomic knowledge base $T = \{a_1, \dots, a_n\}$, where $a_1, \dots, a_n$ are atoms of V, an atom $Q \in T$ and a formula p on V, decide whether Q is derivable from every model in $T \circ_S p$*", where $T \circ_S p$ is the updated (or revised) knowledge base according to the Satoh's revision operator.

Informally, Satoh's revision operator $\circ_S$ selects the models of p that are "closest" to models of T: closest models are those whose symmetric difference with models of T is minimal under set-inclusion semantics. In order to define formally the semantics of $\circ_S$ we first introduce some preliminaries. Let $Mod(p)$ be the set of models of a formula p. Let $\triangle^{min}(T,p) = min_{\subseteq}(\{M \triangle M' : M \in Mod(p),\ M' \in Mod(T)\})$, that is the family of $\subseteq$-minimal sets obtained as symmetric difference between models of p and T. The semantics of Satoh's operator (i.e. the set of models of the knowledge base T revised according to the formula p) is defined as follows:
$Mod(T \circ_S p) = \{\ M\ \in\ Mod(p)\ : \exists M' \in Mod(T)\ s.t.\ M \triangle M' \in \triangle^{min}(T,p)\}$.

In the following the set of atoms occurring in p will be denoted as $V(p)$. Π_2^p-completeness of the implication problem was shown to hold also if $V(p)\ \subseteq\ T$ [10]: we consider this case in our proof. Observe that the definition of $\circ_S$ entails that for each $M \in \triangle^{min}(T,p)$ it holds that $M \subseteq T \cap V(p)$, thus M is a subset of T.

We now consider an instance $< T, p, Q >$ of implication problem, where T is the atomic knowledge base $\{a_1, \dots, a_n\}$, p is a propositional formula (with $V(p) \subseteq T$), and Q is an atom in T.

Let C_p be a boolean circuit equivalent to p. We consider the database scheme $\mathcal{D}$ introduced in the hardness proof of Theorem 1. Moreover, we consider an instance D which is the translation of C_p obtained in the same way as Theorem 1, except that:

- relation $input$ must contain not only the tuples corresponding to the inputs of C_p (i.e. the atoms in $V(p)$), but also the tuples corresponding to the atoms of $T \setminus V(p)$;
- for each tuple inserted in relation $input$, attribute Val is set to 1, which means assigning *true* to all the atoms of T.

Recall that measure attributes in the tuples of relations $gate$ and $gateInput$ are set to -1 (corresponding to an undefined truth value).

Let $\mathcal{AC}$ be the same set of constraints used in the proof of Theorem 1. As explained in the hardness proof of Theorem 1, $\mathcal{AC}$ defines the semantics of C_p and requires that C_p is true. Note that every repair ρ for D w.r.t. $\mathcal{AC}$ must update all measure attributes that initially are set to -1 in D. Therefore, given two repairs ρ and ρ', they differ only on the set of atomic updates performed on relation $input$.

Obviously, every *set*-minimal repair of ρ for D w.r.t. $\mathcal{AC}$ corresponds to a model M in $Mod(T \circ_S p)$, and vice versa. In fact, given a *set*-minimal repair ρ for D w.r.t. $\mathcal{AC}$, a model M for $T \circ_S p$ can be obtained from the repaired database considering only the tuples in relation $input$ where attribute Val is equal to 1 after applying ρ. Observe that the set of atoms M corresponding to ρ is a model $T \circ_S p$, otherwise there would exist $M' \subset M$ with $M' \in Mod(T \circ_S p)$, and the repair ρ' corresponding to M' would satisfy $\rho' <_{set} \rho$, thus contradicting the minimality of ρ. Likewise, it is easy to see that any model in $Mod(T \circ_S p)$ corresponds to a minimal repair for D w.r.t. $\mathcal{AC}$.

Finally consider the query $q = input(id(Q), 1)$. The above considerations suffice to prove that Q is derivable from every model in $Mod(T \circ_S p)$ iff $input(id(Q), 1)$ is true in $\rho(D)$ for every *set*-minimal repair ρ for D w.r.t. $\mathcal{AC}$, that is the consistent answer of $input(id(Q), 1)$ on D w.r.t. $\mathcal{AC}$ is true. $\square$

Theorem 4. *Let $\mathcal{D}$ be a database scheme, D be an instance of $\mathcal{D}$, $\mathcal{AC}$ be a set of aggregate constraints on $\mathcal{D}$ and q be a query over D. Deciding whether $q^{card}(D) = true$ is $\Delta_2^p[log\ n]$-complete.*

Proof. (*Membership*) Membership in $\Delta_2^p[log\ n]$ derives from the fact that repairs on D can be partitioned into the two sets T and F consisting of all repairs ρ_i s.t. $q(\rho_i(D)) =$ *true* and, respectively, $q(\rho_i(D)) =$ *false*. Let *MinSize*$(T) = min_{\rho \in T}(|\lambda(\rho)|)$, and $MinSize(F) = min_{\rho \in F}(|\lambda(\rho)|)$. It can be shown that $q^{card}(D) =$ *true* iff *MinSize*$(T) <$ *MinSize*(F). Both *MinSize*(T) and *MinSize*(F) can be evaluated by a logarithmic number of NP-oracle invocations.
(*Hardness*). Hardness can be proved by showing a reduction from the following implication problem in the context of propositional logic over a finite domain V: "*given an atomic knowledge base T on V, a formula Q on T and a formula p on V, decide whether Q is derivable from every model in $T \circ_D p$*", where $T \circ_D p$ is the updated (or revised) knowledge base according to the Dalal's revision operator. $\Delta_2^p[log\ n]$-completeness of this problem was shown in [10].

The semantics of Dalal's revision operator is as follows. The models of $T \circ_D p$ are the models of p whose symmetric difference with models of T has minimum cardinality w.r.t. all other models of p. More formally, let $|\triangle^{min}(T,p)| = min\{\ |M \triangle M'|\ : M \in Mod(p),\ M' \in Mod(T)\}$, that is the minimum number of atoms in which models of T and p diverge. Then models of $T \circ_D p$ are given by:
$Mod(T \circ_D p) = \{M \in Mod(p) : \exists M' \in Mod(T)\ s.t.\ |M \triangle M'| \in |\triangle|^{min}(T,p)\}$.

Consider an instance $< V, T, p, Q >$ of the implication problem, where V is the finite domain of atoms, T an atomic knowledge base on V, p a formula on V, and Q a formula on T. Let $V(p)$ and $V(Q)$ denote the set of atoms of V occurring in p and Q, respectively. Sets T, $V(p)$ and $V(Q)$ can be partitioned into A, B, C, D, E, as shown in Fig. 1(a).

Let C_p and C_Q be two boolean circuits equivalent to p and Q, respectively. C_p and C_Q are reported in Fig. 1(b), with their inputs. In this figure, atoms belonging to T, $V(p)$ and $V(Q)$ are represented as circles, and the two circuits are represented by means of triangles. In particular, inputs of C_Q are the atoms $b_1, \dots, b_n$ of B and the atoms $c_1, \dots, c_r$ of C, whereas inputs of C_Q are the atoms $c_1, \dots, c_r$ of C, the atoms $d_1, \dots, d_s$ of D, and the atoms $e_1, \dots, e_t$ of D. That is, the atoms of C are inputs of both C_p and C_Q.

These circuits can be represented as an instance of the database scheme $\mathcal{D}$ introduced in the hardness proof of Theorem 1. In particular, we consider an instance D of

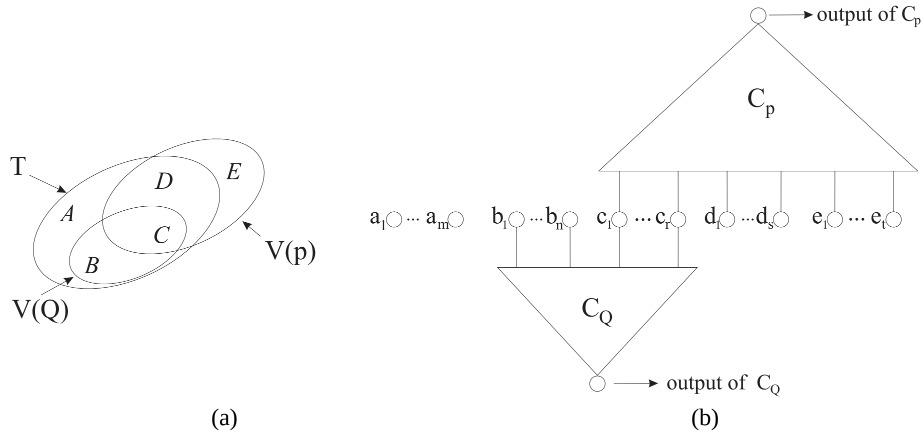

Fig. 1. (a) The partitioning of T, $V(p)$, $V(Q)$; (b) Circuits

$\mathcal{D}$ which is the translation of C_p and C_Q obtained in the same way as Theorem 1, except that:

- relation $input$ contains a tuple for each atom in $A \cup B \cup C \cup D \cup E$;
- for each tuple inserted in relation $input$, attribute Val is set to 1 if it refers to an atom in T, -1 otherwise. This means assigning ***true*** to all the atoms of T, and an undefined truth value to atoms in E.

Recall that measure attributes in the tuples of relations $gate$ and $gateInput$ are set to -1.

We consider the set of aggregate constraints $\mathcal{AC}$ consisting of constraints 1-5 introduced in the hardness proof of Theorem 1, plus the aggregate constraint $NORVal(id(o_p)) = 1$, where $id(o_p)$ is the identifier of the output gate of C_p. As explained in the hardness proof of Theorem 1, $\mathcal{AC}$ defines the semantics of C_p and C_Q and requires that C_p is true.

Note that every repair ρ for D w.r.t. $\mathcal{AC}$ must update all value attributes that initially are assigned -1 in D. Therefore, given two repairs ρ and ρ' for D w.r.t. $\mathcal{AC}$, they differ only on the number of atomic updates performed on the tuples of $input$ where Val was set to 1 in D.

Obviously, every *card*-minimal repair of ρ for D w.r.t. $\mathcal{AC}$ corresponds to a model M in $Mod(T \circ_D p)$, and vice versa (this can be proven straightforwardly, analogously to the proof of Theorem 3, where the correspondence between *set*-minimal repairs for D and models of $T \circ_S p$ has been shown).

Finally consider the query $q = input(id(o_Q), 1)$, where o_Q denotes the the output gate of C_Q. The above-mentioned considerations suffice to prove that Q is derivable from every model in $Mod(T \circ_D p)$ iff $input(id(o_Q), 1)$ is true in $\rho(D)$ for every *card*-minimal repair ρ for D w.r.t. $\mathcal{AC}$, that is the consistent answer of $input(id(o_Q), 1)$ on D w.r.t. $\mathcal{AC}$ is true. □