The VLDB Journal (2012) 21:213–238
DOI 10.1007/s00778-011-0253-7

SPECIAL ISSUE PAPER

Towards certain fixes with editing rules and master data

Wenfei Fan · Jianzhong Li · Shuai Ma · Nan Tang · Wenyuan Yu

Received: 14 February 2011 / Revised: 18 August 2011 / Accepted: 25 August 2011 / Published online: 30 October 2011

Abstract A variety of integrity constraints have been studied for data cleaning. While these constraints can detect the presence of errors, they fall short of guiding us to correct the errors. Indeed, data repairing based on these constraints may not find *certain fixes* that are guaranteed correct, and worse still, may even introduce new errors when attempting to repair the data. We propose a method for finding certain fixes, based on master data, a notion of *certain regions*, and a class of *editing rules*. A certain region is a set of attributes that are assured correct by the users. Given a certain region and master data, editing rules tell us what attributes to fix and how to update them. We show how the method can be used in data monitoring and enrichment. We also develop techniques for reasoning about editing rules, to decide whether they lead to a unique fix and whether they are able to fix all the attributes in a tuple, *relative to* master data and a certain region. Furthermore, we present a framework and an algorithm to find certain fixes, by interacting with the users to ensure that one of the certain regions is correct. We experimentally verify the effectiveness and scalability of the algorithm.

Keywords Certain fix · Editing rule · Master data · Data cleaning · Data quality

Electronic supplementary material The online version of this article (doi:10.1007/s00778-011-0253-7) contains supplementary material, which is available to authorized users.

W. Fan · N. Tang · W. Yu
University of Edinburgh, Edinburgh, UK
e-mail: wenfei@inf.ed.ac.uk

N. Tang
e-mail: ntang@inf.ed.ac.uk

W. Yu
e-mail: wenyuan.yu@ed.ac.uk

W. Fan · J. Li (✉)
Harbin Institute of Technology, Harbin, China
e-mail: lijzh@hit.edu.cn

S. Ma (✉)
Beihang University, Beijing, China
e-mail: shuai.ma@ed.ac.uk; mashuai@act.buaa.edu.cn

1 Introduction

Real-life data is often dirty: 1–5% of business data contains errors [36]. Dirty data costs US companies alone 600 billion dollars each year [15]. These highlight the need for data cleaning, to catch and fix errors in the data. Indeed, the market for data cleaning tools is growing at 17% annually, way above the 7% average forecast for the other IT sectors [24].

An important functionality expected from a data cleaning tool is *data monitoring* [9,37]: when a tuple t is created (either entered manually or generated automatically by some process), it is to find errors in t and correct the errors. That is, we want to ensure that t is clean before it is used, to prevent errors introduced by adding t. As noted by [37], it is far less costly to correct t at the point of data entry than fixing it afterward.

A variety of integrity constraints have been studied for data cleaning, from traditional constraints (*e.g.,* functional and inclusion dependencies [6,13,40]) to their extensions (*e.g.,* conditional functional and inclusion dependencies [8,19,26]). These constraints help us determine whether data is dirty or not, *i.e.,* whether errors are present in the data. However, they fall short of telling us which attributes of t are erroneous and moreover, how to correct the errors.

Example 1 Consider an input tuple t_1 given in Fig. 1a. It specifies a supplier in the UK in terms of name (FN, LN), phone number (area code AC and phone phn) and type,

	FN	LN	AC	phn	type	str	city	zip	item
t_1:	Bob	Brady	020	079172485	2	501 Elm St.	Edi	EH7 4AH	CD
t_2:	Robert	Brady	131	6884563	1	null	Ldn	null	CD
t_3:	Robert	Brady	020	6884563	1	null	null	EH7 4AH	DVD
t_4:	Mary	Burn	029	9978543	1	null	Cad	null	BOOK

(a)

	FN	LN	AC	Hphn	Mphn	str	city	zip	DOB	gender
s_1:	Robert	Brady	131	6884563	079172485	51 Elm Row	Edi	EH7 4AH	11/11/55	M
s_2:	Mark	Smith	020	6884563	075568485	20 Baker St.	Ldn	NW1 6XE	25/12/67	M

(b)

Fig. 1 **a** Example input tuples t_1, t_2, t_3 and t_4, **b** example master relation D_m

address (street str, city, zip code) and items supplied. Here phn is either home phone or mobile phone, indicated by type (1 or 2, respectively).

It is known that in the UK, if AC is 020, city should be Ldn, and when AC is 131, city must be Edi. These can be expressed as conditional functional dependencies (CFDs [19]). The CFDs find that tuple t_1 is *inconsistent*: t_1[AC] = 020 but t_1[city] = Edi. In other words, either t_1[AC] or t_1[city] is incorrect, or both. However, they do not tell us which of the two attributes is wrong and to what value it should be changed. □

Several heuristic methods have been studied for repairing data based on constraints [2,6,14,23,28,30]. For the reasons mentioned above, however, these methods do not guarantee to find correct fixes in data monitoring; worse still, they may introduce new errors when trying to repair the data. For instance, the tuple s_1 of Fig. 1b indicates corrections to t_1. Nevertheless, the prior methods may opt to change t_1[city] to Ldn; this does not fix the erroneous t_1[AC] and worse, messes up the correct attribute t_1[city].

This highlights the quest for effective methods to find *certain fixes* that are guaranteed correct [25,28]. The need for this is especially evident in monitoring *critical* data, in which a minor error may have disastrous consequences [28]. To this end we need *editing rules* that tell us how to fix errors, *i.e.,* which attributes are wrong and what values they should take. In contrast, constraints only detect the presence of errors.

This is possible given the recent development of master data management (MDM [31]). An enterprise nowadays typically maintains *master data* (*a.k.a. reference data*), a single repository of high-quality data that provides various applications with a synchronized, consistent view of its core business entities. MDM systems are being developed by IBM, SAP, Microsoft and Oracle. In particular, master data have been explored to provide *a data entry solution* in the Service Oriented Architecture (SOA) at IBM [37], for data monitoring.

Example 2 A master relation D_m is shown in Fig. 1. Each tuple in D_m specifies a person in the UK in terms of the name (FN, LN), home phone (Hphn), mobile phone (Mphn), address, date of birth (DOB) and gender. An example editing rule eR_1 is:

- for an input tuple t, if there exists a master tuple s in D_m with s[zip] = t[zip], then t should be updated by t[AC, str, city] := s[AC, str, city], provided that t[zip] is *certain*, *i.e.,* it is assured correct by the users.

This rule makes *corrections* to attributes t[AC], t[str] and t[city], by taking values from the master tuple s_1.

Another editing rule eR_2 is:

- if t[type] = 2 (indicating mobile phone) and if there is a master tuple s with s[Mphn] = t[phn], then t[FN, LN] := s[FN, LN], as long as t[phn, type] is certain.

This *standardizes* t_1[FN] by changing Bob to Robert.

As another example, consider tuple t_2 in Fig. 1a, in which t_2[str, zip] are missing, and t_2[AC] and t_2[city] are inconsistent. Consider an editing rule eR_3:

- if t[type] = 1 (indicating home phone) and if there exists a master tuple s in D_m such that s[AC, phn] = t[AC, Hphn], then t[str, city, zip] := s[str, city, zip], provided that t[type, AC, phn] is *certain*.

This helps us fix t_2[city] and *enrich* t_2[str, zip] by taking the corresponding values from the master tuple s_1. □

Contributions. We propose a method for data monitoring, by capitalizing on editing rules and master data.

(1) We introduce a class of editing rules defined in terms of data patterns and updates (Sect. 2). Given an input tuple t that matches a pattern, editing rules tell us what attributes of t should be updated and what values from master data should be assigned to them. In contrast to constraints, editing rules have a *dynamic* semantics, and are *relative to* master data. All the rules in Example 2 can be written as editing rules, but they are not expressible as traditional constraints.

(2) We identify and study fundamental problems for reasoning about editing rules (Sects. 3 and 4). The analyses are relative to a *region* (Z, T_c), where Z is a set of attributes and T_c is a pattern tableau. One problem is to decide whether a set Σ of editing rules guarantees to find a *unique* (deterministic [25,28]) fix for input tuples t's that match a pattern in T_c. The other problems concern whether Σ is able to fix all the attributes of such tuples. Intuitively, as long as $t[Z]$ is assured correct, we want to ensure that editing rules can find a certain fix for t. We show that these problems are coNP-complete, NP-complete or #P-complete, but we identify special cases that are in polynomial time (PTIME).

(3) We present an interactive framework and an algorithm to find certain fixes (Sect. 5). A set of certain regions are first recommended to the users, derived from a set Σ of editing rules and master data D_m available, by using an algorithm of [20]. For an input tuple t, the users may only ensure that $t[X]$ is correct, for a set X of attributes of t. If $t[X]$ matches any of the certain regions, the rules guarantee to find t a certain fix. Otherwise we deduce what other attributes Y of t are implied correct by $t[X]$ and the rules, and moreover, suggest a *minimal* set S of attributes such that as long as $t[S]$ is assured correct, $Y \cup S$ covers a certain region and hence, a certain fix to the entire t is warranted. The interactive process proceeds until the users are guided to reach a certain region. We show that it is NP-complete to find a minimum S. Nonetheless, we develop an efficient heuristic algorithm to find a set of suggestions and introduce effective optimization techniques. These yield a practical data entry solution to clean data.

(4) We experimentally verify the effectiveness and scalability of the algorithm, using real-life hospital data and DBLP (Sect. 6). We find that the algorithm effectively provides suggestions, such that most input tuples are fixed with two or three rounds of interactions only. We also show that it scales well with the size of master data, and moreover, that the optimization techniques effectively reduce the latency during interactions.

Related work. This work extends [20] by including (1) a comprehensive analysis of the fundamental problems in connection with certain fixes (Sect. 4); (2) an interactive framework and algorithm for finding certain fixes (Sect. 5), and (3) its experimental study (Sect. 6). Neither (2) nor (3) was studied in [20]. All the proofs and some of the results of (1) were not presented in [20]. Due to the space constraint we opt to cover these new results by leaving out the deduction algorithms for certain regions and their experimental study of [20].

A variety of constraints have been studied for data cleaning, such as FDs [40], FDs and inclusion dependencies (INDs) [6], CFDs [14,19], conditional inclusion dependencies (CINDs) [8], matching dependencies (MDs) [18], and extensions of CFDs and CINDs [7,11] (see *e.g.,* [17] for a survey). (a) These constraints help us determine whether data are dirty or not, but they do not tell us which attributes are erroneous or how to fix the errors, as illustrated earlier. (b) The static analyses of those constraints have been focusing on the satisfiability and implication problems [7,8,11,18,19], along the same lines as traditional FDs and INDs [1]. Editing rules differ from those constraints in the following: (a) they are defined in terms of updates, and (b) their reasoning is relative to master data and is based on its dynamic semantics, a departure from our familiar terrain of dependency analysis. The rules aim to fix errors, rather than to detect the presence of errors only.

Editing rules are also quite different from edits studied for census data repairing [23,25,28]. Edits (a) are conditions defined on single records of a single relation and (b) are not capable of locating and fixing errors.

Closer to editing rules are MDs [18]. In contrast to editing rules (a) MDs are for record matching (see *e.g.,* [16] for a survey), not for data repairing. (b) They only specify what attributes should be identified, but do not tell us how to update them. (c) MDs neither carry data patterns, nor consider master data; and hence, their analysis is far less challenging. Indeed, the static analyses are in PTIME for MDs [18], but in contrast, the analyses are intractable for editing rules.

There has also been work on rules for active databases (see [39] for a survey). Those rules are far more general than editing rules, specifying events, conditions, and actions. Indeed, even the termination problem for those rules is undecidable, as opposed to the coNP upper bounds for editing rules. Results on those rules do not carry over to editing rules.

Prior work on constraint-based data cleaning has mostly focused on two topics introduced in [2]: *repairing* is to find another consistent database that minimally differs from the original database [2,6,8,10,13,19,23,25,28,30,34,41]; and *consistent query answering* is to find an answer to a given query in every possible repair of the original database (*e.g.,* [2,40]). Although the need for finding certain fixes has long been recognized [25,28], prior methods do not guarantee that fixes are correct, *i.e.,* new errors may be introduced while fixing existing ones in the repairing process. Moreover, master data is not considered in those methods. We shall evaluate the effectiveness of our approach compared with the repairing algorithm of [14] (Sect. 6).

This work studies data monitoring, which is advocated in [9,10,22,37], as opposed to prior data repairing methods [2,6,8,13,19,23,25,28,30,40] that aim to generate another database as a candidate repair of the original data. As noted by [37], it is far less costly to correct t at the point of entry than fixing it afterward. A method for matching input tuples with master data was presented in [9], without repairing the tuples.

Another line of work on data cleaning has focused on record matching [5,18,23,27], to identify records that refer to the same real-world object (see [16] for a survey). This work involves record matching between input tuples and master tuples. There has also been a host of work on more general data cleaning and ETL tools (see [4] for a survey), which are essentially orthogonal, but complementary, to data repairing and this work.

There have also been efforts to interleave merging and matching operations [5,21,27,32]: [27] clusters data rather than repair data, and [5,27] only merge/fuse tuples when matches are found. Those merge operations are far more restrictive than value modifications considered in this work and data repairing. While [21] conducts both repairing and matching using CFDs and MDs, these operations cannot assure the correctness of the repaired data. Indeed, the prior work neither guarantees certain fixes, nor considers master data.

Our data monitoring framework leverages user feedback, similar to [10,34,41]. Potter's Wheel [34] supports interactive data transformations, based on iterative user feedback on example data. USHER [10] cleans data by asking users online about erroneous values, identified by a probabilistic method. GDR [41] develops a CFD-based repairing approach by soliciting user feedback on the updates that are likely to improve data quality. Our approach asks users to assure the correctness of a small number of attributes for an input tuple, to find a certain fix. While all these methods interact with users, they differ from each other in what feedback is requested and how the feedback is used.

Editing rules can be extracted from business rules. They can also be automatically discovered from sample data along the same lines as mining constraints for data cleaning, *e.g.,* [12,26] for CFDs and [38] for MDs.

Organization. Section 2 defines editing rules. Section 3 presents certain fixes. Section 4 studies fundamental problems in connection with certain fixes. An interactive framework for data monitoring is introduced in Sect. 5. The experimental study is presented in Sect. 6, followed by conclusions in Sect. 7.

2 Editing rules

We study editing rules for data monitoring. Given a master relation D_m and an input tuple t, we want to fix errors in t using editing rules and data values in D_m.

We specify input tuples t with a relation schema R and use $A \in R$ to denote that A is an attribute of R. The master relation D_m is an instance of a relation schema R_m, often distinct from R. As remarked earlier, D_m can be assumed consistent and complete [31].

Editing rules. An *editing rule* (eR) φ defined on (R, R_m) is a pair $((X, X_m) \rightarrow (B, B_m), t_p[X_p])$, where

- X and X_m are two lists of distinct attributes in schemas R and R_m, respectively, with the same length, *i.e.,* $|X| = |X_m|$;
- B is an attribute such that $B \in R \setminus X$, and attribute $B_m \in R_m$; and
- t_p is a pattern tuple over a set of distinct attributes X_p in R such that for each $A \in X_p$, $t_p[A]$ is one of _, a or $\bar{a}$. Here a is a constant drawn from the domain of A, and _ is an unnamed variable.

Intuitively, a and $\bar{a}$ specify Boolean conditions $x = a$ and $x \neq a$ for a value x, respectively, and _ is a wildcard that imposes no conditions. More specifically, we say that a tuple t of R *matches* pattern tuple t_p, denoted by $t[X_p] \approx t_p[X_p]$, if for each attribute $A \in X_p$ (1) $t[A] = a$ if $t_p[A]$ is a (2) $t[A] \neq a$ if $t_p[A]$ is $\bar{a}$, and (3) $t[A]$ is any value from the domain of A if $t_p[A]$ is _.

Example 3 Consider the supplier schema R and master relation schema R_m shown in Fig. 1. The rules eR_1, eR_2 and eR_3 described in Example 2(b) can be expressed as the following editing rules φ_1– φ_4 defined on (R, R_m).

φ_1: $((\mathsf{zip}, \mathsf{zip}) \rightarrow (B_1, B_1), t_{p1} = ())$;
φ_2: $((\mathsf{phn}, \mathsf{Mphn}) \rightarrow (B_2, B_2), t_{p2}[\mathsf{type}] = (2))$;
φ_3: $((\mathsf{[AC, phn]}, \mathsf{[AC, Hphn]}) \rightarrow (B_3, B_3), t_{p3}[\mathsf{type, AC}] = (1, \overline{0800}))$;
φ_4: $((\mathsf{AC}, \mathsf{AC}) \rightarrow (\mathsf{city}, \mathsf{city}), t_{p4}[\mathsf{AC}] = (\overline{0800}))$.

Here eR_1 is expressed as three editing rules of the form φ_1, for B_1 ranging over {AC, str, city}. In φ_1, both X and X_m consist of zip, and B and B_m are B_1. Its pattern tuple t_{p1} poses no constraints. Similarly, eR_2 is expressed as two editing rules of the form φ_2, in which B_2 is either FN or LN. The pattern tuple $t_{p2}[\mathsf{type}] = (2)$, requiring that phn is mobile phone. The rule eR_3 is written as φ_3 for B_3 ranging over {str, city, zip}, where $t_{p3}[\mathsf{type, AC}]$ requires that type = 1 (home phone) yet AC $\neq$ 0800 (toll free, non-geographic). The eR φ_4 states that for a tuple t, if $t[\mathsf{AC}] \neq 0800$ and $t[\mathsf{AC}]$ is correct, we can update $t[\mathsf{city}]$ using the master data. □

Semantics. We next give the semantics of editing rules.

We say that an eR φ and a master tuple $t_m \in D_m$ *apply* to an R tuple t, which results in a tuple t', denoted by $t \rightarrow_{(\varphi, t_m)} t'$, if (1) $t[X_p] \approx t_p[X_p]$ (2) $t[X] = t_m[X_m]$, and (3) t' is obtained by the update $t[B] := t_m[B_m]$. We shall simply say that (φ, t_m) apply to t.

That is, if t matches t_p and if $t[X]$ agrees with $t_m[X_m]$, then we assign $t_m[B_m]$ to $t[B]$. Intuitively, if $t[X, X_p]$ is assured correct (referred to as *validated*), we can safely *enrich* $t[B]$ with master data $t_m[B_m]$ as long as (1) $t[X]$ and $t_m[X_m]$ are identified, and (2) $t[X_p]$ matches the pattern in φ.

This yields a new tuple t' such that $t'[B] = t_m[B_m]$ and $t'[R \setminus \{B\}] = t[R \setminus \{B\}]$.

We write $t \rightarrow_{(\varphi, t_m)} t$ if φ and t_m do *not* apply to t, *i.e.,* t is unchanged by φ if either $t[X_p] \not\approx t_p[X_p]$ or $t[X] \neq t_m[X_m]$.

Example 4 As shown in Example 2, we can correct t_1 by applying the eR φ_1 and master tuple s_1 to t_1. As a result, $t_1[\mathsf{AC}, \mathsf{str}]$ is changed from (020, 501 Elm St.) to (131, 51 Elm Row). Furthermore, we can standardize $t_1[\mathrm{FN}]$ by applying φ_2 and s_1 to t_1, such that $t_1[\mathrm{FN}]$ is changed from Bob to Robert.

The eR φ_3 and master tuple s_1 can be applied to t_2, to correct $t_2[\mathsf{city}]$ and enrich $t_2[\mathsf{str}, \mathsf{zip}]$. □

Notations. We shall use the following notations.

(1) Given an eR $\varphi = ((X, X_m) \rightarrow (B, B_m), t_p[X_p])$, we denote (a) $\mathrm{LHS}(\varphi) = X, \mathrm{RHS}(\varphi) = B$; (b) $\mathrm{LHS}_m(\varphi) = X_m, \mathrm{RHS}_m(\varphi) = B_m$; and (c) $\mathrm{LHS}_p(\varphi) = X_p$.

(2) Given a set Σ of eRs, we denote $\cup_{\varphi \in \Sigma} \mathrm{LHS}(\varphi)$ by $\mathrm{LHS}(\Sigma)$; similarly for $\mathrm{RHS}(\Sigma)$, $\mathrm{LHS}_m(\Sigma)$ and $\mathrm{RHS}_m(\Sigma)$. Here abusing the notions for sets, we use $X \cup Y$, $X \cap Y$ and $X \setminus Y$ to denote the *union*, *intersection* and *difference* of two lists X and Y of attributes, respectively.

(3) An eR $\varphi = ((X, X_m) \rightarrow (B, B_m), t_p[X_p])$ is said to be in *the normal form* if $t_p[X_p]$ does not contain wildcard _. Every eR φ can be normalized to an eR φ' by removing all such attributes A from $t_p[X_p]$ that $t_p[A] =$ _. From the semantics of eRs one can readily verify that φ and φ' are *equivalent*: for any input tuple t, master tuple t_m, and tuple t', $t \rightarrow_{(\varphi, t_m)} t'$ iff $t \rightarrow_{(\varphi', t_m)} t'$.

Remarks. (1) As remarked earlier, editing rules are quite different from CFDs [19]. A CFD $\psi = (X \rightarrow Y, t_p)$ is defined on a single relation R, where $X \rightarrow Y$ is a standard FD and t_p is a pattern tuple on X and Y. It requires that for any tuples t_1, t_2 of R, if t_1 and t_2 match t_p, then $X \rightarrow Y$ is enforced on t_1 and t_2. When $t_p[Y]$ consists of constants only, it is referred to as a constant CFD. It has a *static* semantics: t_1 and t_2 either satisfy or violate ψ, but they are not updated. As shown in Example 1, when t_1 and t_2 violate φ, one cannot tell which of $t_1[X]$, $t_1[Y]$ or $t_2[Y]$ is erroneous, and hence, cannot simply apply φ to find a certain fix. The problem remains even when φ is a constant CFD, which can be violated by a single tuple. In contrast, an eR φ specifies an action: applying φ and a master tuple t_m to t yields an updated t'. It is defined in terms of master data. As will be seen shortly, this yields a certain fix when φ and t_m are applied to a region that is validated.

(2) MDs of [18] also have a dynamic semantics. An MD ϕ is of the form $((X, X'), (Y, Y'), \mathsf{OP})$, where X, Y and X', Y' are lists of attributes in schemas R, R', respectively, and OP is a list of similarity operators. For an R_1 tuple t_1 and an R_2 tuple t_2, ϕ states that if $t_1[X]$ and $t_2[X']$ match *w.r.t.* the operators in OP, then $t_1[Y]$ and $t_2[Y']$ are identified as the same object. As remarked in Sect. 1, eRs differ from MDs in several aspects.

Neither CFDs nor MDs are expressible as eRs, and *vice versa*, because of their different semantics.

(3) To simplify the discussion we consider a single master relation D_m. Nonetheless, the results of this work readily carry over to multiple master relations. Indeed, given master schemas $R_{m_1}, \ldots, R_{m_k}$, there exists a single master schema R_m such that each instance D_m of R_m characterizes an instance of $(D_{m_1}, \ldots, D_{m_k})$ of those schemas. Here R_m has a special attribute id such that $\sigma_{\mathsf{id}=i}(R_m)$ yields D_{m_i} for $i \in [1, k]$.

We summarize notations of this section in Table 1.

Table 1 Summary of notations of Sect. 2

R	Input relation schema
R_m	Master relation schema
Σ	A set of eRs on (R, R_m)
D_m	Master data on R_m
$\bar{a}$	Boolean condition $x \neq a$ for a value x
$t \approx t_c$	An input tuple t *matches* a pattern tuple t_c
$t \rightarrow_{(\varphi, t_m)} t'$	Applying eR φ and a master tuple t_m to an input tuple t, yielding t'

3 Certain fixes and certain regions

Consider a master relation D_m of schema R_m, and a set Σ of editing rules defined on (R, R_m). Given a tuple t of R, we want to find a "certain fix" t' of t by using Σ and D_m. That is (1) no matter how eRs of Σ and master tuples in D_m are applied, Σ and D_m yield a unique t' by updating t; and (2) all the attributes of t' are ensured correct (validated).

To formalize the notion of certain fixes, we first introduce a notion of regions. When applying an eR φ and a master tuple t_m to t, we update t with values in t_m. To ensure that the changes make sense, some attributes of t have to be validated. In addition, we are not able to update t if either it does not match the pattern tuple of φ or it cannot find a master tuple t_m in D_m that carries the information needed for correcting t.

Example 5 Consider the master data D_m of Fig. 1b and a set Σ_0 consisting of $\varphi_1, \varphi_2, \varphi_3$ and φ_4 of Example 3. Both (φ_1, s_1) and (φ_3, s_2) apply to tuple t_3 of Fig. 1a. However, they suggest to update $t_3[\mathsf{city}]$ with distinct values Edi and Lnd. The conflict arises because $t_3[\mathsf{AC}]$ and $t_3[\mathsf{zip}]$ are inconsistent. Hence to fix t_3, we need to assure that one of $t_3[\mathsf{AC}]$ and $t_3[\mathsf{zip}]$ is correct.

Now consider tuple t_4 of Fig. 1a. Since no eRs in Σ_0 and master tuples in D_m can be applied to t_4, we cannot tell

whether t_4 is correct. This is because Σ_0 and D_m do not cover all the cases of input tuples. □

This motivates us to introduce the following notion.

Regions. A *region* is a pair (Z, T_c), where Z is a list of distinct attributes in R, T_c is a *pattern tableau* consisting of a set of pattern tuples with attributes in Z, and each pattern tuple is defined as its counterparts in eRs.

We say that a tuple t is *marked* by (Z, T_c) if there exists $t_c \in T_c$ such that $t \approx t_c$.

Intuitively, a region (Z, T_c) specifies what input tuples can be corrected with certain fixes by a set Σ of eRs and master data. As will be seen shortly (1) it tells us that to correctly fix errors in a tuple t, $t[Z]$ should be assured correct, and moreover, t is marked such that there exist an eR and a master tuple that can be applied to t. (2) There exist no two eRs in Σ such that both of them can be applied to t, but they lead to inconsistent updates. In other words, T_c imposes constraints *stronger than* those specified by pattern tuples in eRs, to prevent the abnormal cases illustrated in Example 5.

Consider an eR $\varphi = ((X, X_m) \to (B, B_m), t_p[X_p])$, a master tuple t_m and a region (Z, T_c). When we apply φ and t_m to a tuple t marked by (Z, T_c), we require that $X \subseteq Z$, $X_p \subseteq Z$, $B \notin Z$. That is, it is justified to apply φ and t_m to t for those t marked by (Z, T_c) if $t[X, X_p]$ is correct. As $t[Z]$ is validated, we make $t[B]$ "protected", *i.e.,* unchanged, by enforcing $B \notin Z$. We denote this as $t \to_{((Z,T_c),\varphi,t_m)} t'$, where $t \to_{(\varphi,t_m)} t'$.

Example 6 Referring to Example 5, a region defined on R is $(Z_{\mathsf{AH}}, T_{\mathsf{AH}}) = ((\mathsf{AC}, \mathsf{phn}, \mathsf{type}), \{(\overline{0800}, \_, 1)\})$. Note that tuple t_3 of Fig. 1a is marked by $(Z_{\mathsf{AH}}, T_{\mathsf{AH}})$. Hence, if $t_3[\mathsf{AC}, \mathsf{phn}, \mathsf{type}]$ is validated, then (φ_3, s_2) can be applied to t_3, yielding $t_3 \to_{((Z_{\mathsf{AH}},T_{\mathsf{AH}}),\varphi_3,s_2)} t'_3$, where $t'_3[\mathsf{str}, \mathsf{city}, \mathsf{zip}] := s_2[\mathsf{str}, \mathsf{city}, \mathsf{zip}]$, and t'_3 and t_3 agree on all the other attributes of R. □

Note that if $t \to_{((Z,T_c),\varphi,t_m)} t'$, then $t'[B]$ is validated as a logical consequence of the application of φ and t_m, since $t[Z]$ is validated. That is, $t'[B]$ is assured correct when applying rules to t' in the process for fixing t (see below). Hence we can extend (Z, T_c) by including B in Z and by expanding each t_c in T_c such that $t_c[B] = \_$. We denote the extended region as $\mathsf{ext}(Z, T_c, \varphi)$.

Example 7 Consider the region $(Z_{\mathsf{AH}}, T_{\mathsf{AH}})$ in Example 6. Then $\mathsf{ext}(Z_{\mathsf{AH}}, T_{\mathsf{AH}}, \varphi_3)$ is (Z', T'), where Z' consists of attributes AC, phn, type, str, city and zip, and T' has a single pattern tuple $t'_c = (\overline{0800}, \_, 1, \_, \_, \_)$. □

Fixes. We say that a tuple t' is a *fix* of t by (Σ, D_m) *w.r.t.* (Z, T_c), denoted by $t \to^*_{((Z,T_c),\Sigma,D_m)} t'$, if there exists a finite sequence $t_0 = t, t_1, \ldots, t_k = t'$ of tuples of R such that for each $i \in [1, k]$, there exist $\varphi_i \in \Sigma$ and $t_{m_i} \in D_m$ such that

(1) $t_{i-1} \to_{((Z_{i-1},T_{i-1}),\varphi_i,t_{m_i})} t_i$, where $(Z_0, T_0) = (Z, T_c)$ and $(Z_i, T_i) = \mathsf{ext}(Z_{i-1}, T_{i-1}, \varphi_i)$; and
(2) for all $\varphi \in \Sigma$ and $t_m \in D_m$, $t' \to_{((Z_k,T_k),\varphi,t_m)} t'$.

These conditions ensure that (1) each step of the process is justified; and (2) t' is a fixpoint and cannot be further updated. Note that $t_{i-1} \to_{((Z_{i-1},T_{i-1}),\varphi_i,t_{m_i})} t_i$ assures that $t_i[Z] = t_0[Z] = t[Z]$, *i.e., t*$[Z]$ is assumed correct and hence, remains unchanged in the process.

Unique fixes. We say that an R tuple t has a *unique fix* by (Σ, D_m) *w.r.t.* (Z, T_c) if there exists a unique t' such that $t \to^*_{((Z,T_c),\Sigma,D_m)} t'$.

When there exists a unique fix t' of t with a finite sequence $t_0 = t, t_1, \ldots, t_k = t'$ of tuples of R, we refer to Z_k as the set of attributes of t *covered by* (Z, T_c, Σ, D_m).

Certain fixes. We say that an R tuple t has a *certain fix* by (Σ, D_m) *w.r.t.* (Z, T_c) if (1) t has a unique fix and (2) the set of attributes covered by (Z, T_c, Σ, D_m) includes *all* the attributes in R.

A notion of deterministic fixes was addressed in [25,28]. It refers to unique fixes, *i.e.,* (1) above, without requiring (2). Further, it is not defined relative to (Z, T_c).

Intuitively, a unique fix t' becomes a certain fix when the set of attributes covered by (Z, T_c, Σ, D_m) includes *all* the attributes in R. We can find a certain fix for a tuple t of R marked by a region (Z, T_c) if (a) $t[Z]$ is assured correct (b) there is a unique fix t'; and (c) all the remaining values of $t'[R \setminus Z]$ are correctly fixed.

Example 8 By the set Σ_0 of eRs of Example 5 and the master data D_m of Fig. 1b, tuple t_3 of Fig. 1a has a unique fix *w.r.t.* $(Z_{\mathsf{AH}}, T_{\mathsf{AH}})$, namely, t'_3 given in Example 6. However, as observed in Example 5, if we extend the region by adding zip, denoted by $(Z_{\mathsf{AHZ}}, T_{\mathsf{AH}})$, then t_3 no longer has a unique fix by (Σ_0, D_m) *w.r.t.* $(Z_{\mathsf{AHZ}}, T_{\mathsf{AH}})$.

As another example, consider a region $(Z_{\mathsf{zm}}, T_{\mathsf{zm}})$, where $Z_{\mathsf{zm}} = (\mathsf{zip}, \mathsf{phn}, \mathsf{type})$, and T_{zm} has a single tuple $(\_, \_, 2)$. As shown in Example 4, tuple t_1 of Fig. 1a has a unique fix by Σ_0 and D_m *w.r.t.* $(Z_{\mathsf{zm}}, T_{\mathsf{zm}})$, by correctly applying (φ_1, s_1) and (φ_2, s_2). It is *not* a certain fix, since the set of attributes covered by $(Z_{\mathsf{zm}}, T_{\mathsf{zm}}, \Sigma_0, D_m)$ does not include item. Indeed, the master data D_m of Fig. 1b has no information about item, and hence, does not help here. To find a certain fix, one has to extend Z_{zm} by adding item. In other words, its correctness has to be assured by the users. □

Certain regions. We next introduce the last notion of this section. We say that a region (Z, T_c) is a *certain region* for (Σ, D_m) if for all tuples t of R that are marked by (Z, T_c), t has a certain fix by (Σ, D_m) *w.r.t.* (Z, T_c).

We are naturally interested in certain regions since they warrant absolute corrections, which are assured either by the

Table 2 Summary of notations of Sect. 3

(Z, T_c)	A *region* with a list Z of distinct attributes and a pattern tableau T_c
$t \rightarrow_{((Z,T_c),\varphi,t_m)} t'$	*Applying* eR φ and master tuple t_m to input tuple t *w.r.t.* (Z, T_c), yielding t'
$t \rightarrow^*_{((Z,T_c),\Sigma,D_m)} t'$	Tuple t' is a *fix* of input tuple t by (Σ, D_m) *w.r.t.* (Z, T_c)
Attributes *covered*	All those attributes in t' that are validated by $t \rightarrow^*_{((Z,T_c),\Sigma,D_m)} t'$

users (the attributes Z) or by master data (the remaining attributes $R \backslash Z$).

Example 9 As shown in Example 8, $(Z_{\mathsf{zm}}, T_{\mathsf{zm}})$ is not a certain region. One can verify that a certain region for (Σ_0, D_m) is $(Z_{\mathsf{zmi}}, T_{\mathsf{zmi}})$, where Z_{zmi} extends Z_{zm} by including item, and T_{zmi} consists of patterns of the form $(z, p, 2, \_)$ for z, p ranging over $s[\mathsf{zip}, \mathsf{Mphn}]$ for all master tuples s in D_m. For those tuples marked by the region, certain fixes are warranted.

Another certain region for (Σ_0, D_m) is $(Z_{\mathsf{L}}, T_{\mathsf{L}})$, where $Z_{\mathsf{L}} = (\mathrm{FN}, \mathrm{LN}, \mathsf{AC}, \mathsf{phn}, \mathsf{type}, \mathsf{item})$, T_{L} consists of pattern tuples of the form $(f, l, a, h, 1, \_)$, and (f, l, a, h) is $s[\mathrm{FN}, \mathrm{LN}, \mathsf{AC}, \mathsf{Hphn}]$ for all $s \in D_m$. □

We summarize notations in Table 2.

4 Static analyses of fundamental problems

Given a set Σ of eRs and a master relation D_m, we want to make sure that they can *correctly* fix *all* errors in those input tuples marked by a region (Z, T_c). This motivates us to study fundamental problems associated with certain fixes by (Σ, D_m) and (Z, T_c), and establish their complexity and approximation bounds.

4.1 Reasoning about editing rules

We start with the problems for reasoning about editing rules when regions are provided. Given (Σ, D_m) and a region (Z, T_c), we want to know (a) whether (Σ, D_m) and (Z, T_c) have any conflicts when put together (referred to as the consistency problem), and (b) whether (Z, T_c) makes a certain region for (Σ, D_m) (known as the coverage problem). We show that these problems are intractable, but identify PTIME special cases.

The consistency problem. We say that (Σ, D_m) is *consistent relative to* (Z, T_c) if for each input R tuple t marked by (Z, T_c), t has a unique fix by (Σ, D_m) *w.r.t.* (Z, T_c). Intuitively, this says that Σ and D_m do not have conflicts *w.r.t.*(Z, T_c), as illustrated below.

Example 10 There exist (Σ, D_m) and (Z, T_c) that are inconsistent. Indeed, (Σ_0, D_m) described in Example 5 is not consistent relative to region $(Z_{\mathsf{AHZ}}, T_{\mathsf{AHZ}})$ of Example 8, since eRs in Σ_0 suggest distinct values to update $t_3[\mathsf{city}]$ for tuple t_3 of Fig. 1a, *i.e.,* conflicts arise, as shown in Example 5. Hence t_3 does not have a unique fix by (Σ_0, D_m) *w.r.t.* $(Z_{\mathsf{AHZ}}, T_{\mathsf{AHZ}})$. □

The *consistency problem* for editing rules is to determine, given any (Z, T_c) and (Σ, D_m), whether (Σ, D_m) is consistent relative to (Z, T_c).

The problem is obviously important, but is nontrivial. It is known that for constraints defined with pattern tuples, the presence of attributes with a finite domain makes their static analysis hard [8,19]. For instance, when it comes to the problem for deciding whether a set of CFDs can be satisfied by a nonempty database, the problem is NP-complete if attributes in the CFDs may have a finite domain, but it becomes tractable when all the attributes in the CFDs have an infinite domain [19]. In contrast, below we show that the consistency problem for editing rules is intractable even when all the attributes involved have an infinite domain.

Theorem 1 *The consistency problem for editing rules is* coNP*-complete, even when data and master relations have infinite-domain attributes only.*

Proof We first show that the complement of the problem is in NP. We then show the problem is coNP-hard, even when only infinite-domain attributes are involved.

(I) We show that the problem is in coNP by providing an NP algorithm for its complement. Given (Σ, D_m) and (Z, T_c), the algorithm returns 'yes' iff (Σ, D_m) is *not* consistent relative to (Z, T_c). Let dom be the set of all constants appearing in D_m or Σ, and an additional distinct constant that is not in dom (if there exists one).

The NP algorithm works as follows:

(a) guess a pattern tuple t_c in T_c, and an R tuple t such that for each R attribute A, $t[A]$ is a constant in dom;
(b) if $t \approx t_c$, then check whether (Σ, D_m) is consistent relative to region $(Z, \{t[Z]\})$; and
(c) if the answer is 'no', the algorithm returns 'yes'. Otherwise reject the guess and repeat the process.

Obviously the algorithm returns 'yes' iff there exists a tuple t marked by (Z, T_c) and it serves as a witness of the inconsistency of (Σ, D_m). The algorithm returns "no" when there exists no such a witness tuple.

As will be shown by Theorem 4 below, step (b) is in PTIME. From this it follows that the algorithm is in NP. Hence the consistency problem is in coNP.

(II) We next show that the problem is coNP-hard, by reduction from the 3SAT problem to its complement. It is known that the 3SAT problem is NP-complete (cf. [33]).

An instance ϕ of 3SAT is of the form $C_1 \wedge \cdots \wedge C_n$, where the variables in ϕ are $x_1, \ldots, x_m$, each clause C_j ($j \in [1, n]$) is of the form $y_{j_1} \vee y_{j_2} \vee y_{j_3}$, and moreover, for $i \in [1, 3]$, y_{j_i} is either $x_{p_{ji}}$ or $\overline{x_{p_{ji}}}$ for $p_{ji} \in [1, m]$. Here we use $x_{p_{ji}}$ to denote the occurrence of a variable in the literal i of clause C_j. The 3SAT problem is to determine whether ϕ is satisfiable.

Given a 3SAT instance ϕ, we construct an instance of the consistency problem consisting of: (a) two schemas R and R_m (b) a master instance D_m of R_m (c) a pattern tableau T_c consisting of a single pattern tuple t_c for a list Z of distinct attributes of R, and (d) a set Σ of eRs. We show that (Σ, D_m) is consistent relative to region (Z, T_c) iff the instance ϕ is *not* satisfiable.

(1) We first construct the consistency instance.

(a) The two schemas are $R(A, X_1, \ldots, X_m, C_1, \ldots, C_n, V, B)$ and $R_m(Y_0, Y_1, A, V, B)$, respectively, where all attributes have an (infinite) integer domain.

Intuitively, for each R tuple t, $t[X_1 \ldots X_m]$, $t[C_1 \ldots C_n]$ and $t[V]$ specify a truth assignment ξ for the variables $x_1, \ldots, x_m$ of ϕ, the truth values of the clauses $C_1, \ldots, C_n$, and the truth value of ϕ under ξ, respectively. Attributes A and B will be used to match patterns of eRs in Σ, and to demonstrate conflicts, respectively.

(b) The master relation D_m consists of three master tuples t_{m_1}, t_{m_2} and t_{m_3}, given as follows.

	Y_0	Y_1	A	V	B
t_{m_1}:	0	1	1	1	1
t_{m_2}:	0	1	1	1	0
t_{m_3}:	0	1	1	0	1

As will seen shortly (i) data values 0 and 1 correspond to Boolean truth values true and false, respectively; (ii) for R tuples t such that $t[V] = 1$, there are two distinct fixes since $t_{m_1}[V] = t_{m_2}[V] = 1$, but $t_{m_1}[B] \neq t_{m_2}[V]$; and (iii) for R tuples t such that $t[V] = 0$, there is only one possible fix.

(c) The list of attributes $Z = (A, X_1, \ldots, X_m)$ and the pattern tuple $t_c[Z] = (1, \_, \ldots, \_)$.

(d) The set Σ is the union of $n + 2$ sets of eRs: $\Sigma_1 \cup \ldots \cup \Sigma_n \cup \Sigma_{C,V} \cup \Sigma_{V,B}$.

- For each $j \in [1, n]$, Σ_j defines eight eRs for clause C_j of ϕ. Each eR $\varphi_{(j,\langle b_1 b_2 b_3\rangle)}$ is of the form $((A, A) \rightarrow (C_j, Y_j), t_{(p_j,\langle b_1 b_2 b_3\rangle)}[X_{p_{j1}} X_{p_{j2}} X_{p_{j3}}] = (b_1, b_2, b_3))$, where (1) for each $i \in [1, 3]$, $b_i \in \{0, 1\}$, and (2) $Y_j = Y_0$ if (b_1, b_2, b_3) makes C_j false by letting $\xi(x_{p_{ji}}) = b_i$, and $Y_j = Y_1$ otherwise.
  Intuitively, we enumerate all eight distinct truth assignments for each clause C_j ($j \in [1, n]$), and construct an eR to assign the corresponding truth value of C_j for each truth assignment.
- The set $\Sigma_{C,V} = \{\varphi_0, \ldots, \varphi_n\}$ consists of $n + 1$ eRs, where (1) for $j \in [1, n]$, $\varphi_j = ((A, A) \rightarrow (V, Y_0), t_{p_j}[C_j] = (0))$, and (2) $\varphi_0 = ((A, A) \rightarrow (V, Y_1), t_{p_0}[C_1 \ldots C_n] = (1, \ldots, 1))$.
  Intuitively, these eRs define the relationships between the truth values of ϕ and the clauses $C_1, \ldots, C_n$. If there exists a clause C_j with truth value 0, then the truth value of ϕ is 0; and if all clauses have a truth value 1, then the truth value of ϕ is 1.
- The set $\Sigma_{V,B}$ consists of a single eR $\varphi_{V,B} = ((V, V) \rightarrow (B, B), ())$, *i.e.,* with an empty pattern tuple. Intuitively, this eR says that for an R tuple t, (1) if $t[V] = 0$, there exists a unique fix t' of t such that $t'[B] = 1$; and (2) if $t[V] = 1$, there exist two fixes t'_1 and t'_2 of t such that $t'_1[B] = 1$ and $t'_1[B] = 0$.

Observe that D_m has a fixed size and Σ consists of $9n + 2$ eRs. Thus the reduction above is in PTIME.

(2) We next show that (Σ, D_m) is consistent relative to (Z, T_c) iff the 3SAT instance ϕ is not satisfiable.

Assume first that (Σ, D_m) is consistent relative to (Z, T_c). We prove that ϕ is not satisfiable by contradiction. If ϕ is satisfiable, then there exists a satisfying truth assignment ξ of the variables $x_1, \ldots, x_m$. Let t be an R tuple such that $t[A, X_1, \ldots, X_m] = (1, \xi(x_1), \ldots, \xi(x_m))$ and $t[C_1, \ldots, C_n, V, B]$ be any (partial) tuple.

Observe the following. (a) By applying the eRs in $\Sigma_1 \cup \ldots \cup \Sigma_n$ and the master tuple t_{m_1} (or one of t_{m_2} and t_{m_3}) in D_m to tuple t, we have a fix t_1 of t such that $t_1[A, X_1, \ldots, X_m] = t[A, X_1, \ldots, X_m]$ and $t_1[C_1, \ldots, C_n] = (1, \ldots, 1)$. (b) By applying the eR φ_{n+1} in $\Sigma_{C,V}$ and the master tuple t_{m_1} (or t_{m_2}) in D_m to tuple t_1, we have a fix t_2 of t_1 such that $t_2[A, X_1, \ldots, X_m, C_1, \ldots, C_n] = t_1[A, X_1, \ldots, X_m, C_1, \ldots, C_n]$ and $t_2[V] = 1$. (c) Finally, by applying the single eR in $\Sigma_{V,B}$ and the master tuple t_{m_1} in D_m to tuple t_2, we have a fix $t_{3,1}$ of t_2 such that $t_{3,1}[A, X_1, \ldots, X_m, C_1, \ldots, C_n, V] = t_2[A, X_1, \ldots, X_m, C_1, \ldots, C_n, V]$ and $t_{3,1}[B] = 1$. In contrast, by applying the eR in $\Sigma_{V,B}$ and t_{m_2} to t_2, we have another distinct fix $t_{3,2}$ of t_2 such that $t_{3,2}[A, X_1, \ldots, X_m, C_1, \ldots, C_n, V] = t_2[A, X_1, \ldots, X_m, C_1, \ldots C_n, V]$ and $t_{3,2}[B] = 0$. That is, (Σ, D_m) is not consistent relative to (Z, T_c), which contradicts our assumption.

Conversely, assume that ϕ is not satisfiable. We show that (Σ, D_m) is consistent relative to (Z, T_c). Let t be an R tuple such that $t[A, X_1, \ldots, X_m]$ is assured correct. It suffices to consider the following two cases.

Case (a). There exists $i \in [1, m]$ such that $t[X_i] \notin \{0, 1\}$. Then there must exist Σ_j ($1 \leq j \leq n$) such that no eRs in Σ_j and master tuples in D_m can be applied to the tuple t.

In particular, the eR φ_0 in $\Sigma_{C,V}$ may not be applied to t since the region (Z, T_c) cannot be expanded to include all attributes $C_1, \ldots, C_n$. Hence it is easy to verify that (Σ, D_m) is consistent relative to (Z, T_c).

Case (b). For each $i \in [1, m]$, $t[X_i] \in \{0, 1\}$. Since ϕ is not satisfiable, the eR φ_0 in $\Sigma_{C,V}$ and any master tuple in D_m cannot be applied to the tuple t. Thus again (Σ, D_m) is consistent relative to (Z, T_c).

Taken together, (I) and (II) show that the consistency problem for editing rules is coNP-complete. Moreover, the reduction of (II) uses infinite-domain attributes only, and hence, the coNP lower bound remains intact when all the attributes of input tuples and master tuples have an infinite domain. □

Theorem 1 tells us that the consistency analysis of eRs is more intricate than its CFD counterpart, which is in PTIME when all attributes involved have an infinite domain. It is also much harder than MDs, since any set of MDs is consistent [18]. Nevertheless, it is still decidable, as opposed to the undecidability for reasoning about rules for active databases [39].

The coverage problem. The *coverage problem* is to decide, given any (Z, T_c) and (Σ, D_m), whether (Z, T_c) is a certain region for (Σ, D_m). That is, whether (Σ, D_m) is able to fix errors in all the attributes of input tuples that are marked by (Z, T_c).

The coverage problem is, however, also intractable.

Theorem 2 *The coverage problem is* coNP*-complete, even for input tuples and master relations that have infinite-domain attributes only.*

The proof is similar to the proof of Theorem 1: the coNP upper bound is verified by providing a similar coNP algorithm, and the coNP-hardness is shown by reduction from the 3SAT problem to its complement. As opposed to its counterpart of Theorem 1, the reduction uses negations in the pattern tuples of eRs. We defer the proof to the appendix due to the space constraint.

Remark. Like the consistency and the coverage problems we have seen earlier, for all the problems to be studied in the rest of the section, their complexity remains the same in the presence of finite-domain attributes and in their absence. Hence in the sequel, we shall simply refer to their complexity bounds without remarking the absence of finite-domain attributes.

Special cases. To better understand these problems, we further investigate the following five special cases.

(1) *Fixed* Σ. In this setting, the set Σ of eRs is fixed. Indeed, editing rules are often predefined in practice.

(2) *Fixed* D_m. In this case, the master data D_m is fixed. In real-life, master data is changed less frequently than (input) data relations.

(3) *Positive* T_c. This case assumes no pattern tuples in T_c contain $\bar{a}$, *i.e.,* in the absence of negations.

(4) *Concrete* T_c. This case requires that no pattern tuples in T_c contain wildcard '_' or $\bar{a}$, *i.e.,* they contain a's only. Note that a concrete T_c must be a positive T_c.

(5) *Direct fixes*. We consider in this setting that (a) for all eRs $\varphi = ((X, X_m) \to (B, B_m), t_p[X_p])$ in Σ, $X_p \subseteq X$, *i.e.,* the pattern attributes X_p are also required to find a match in D_m, and (b) each step of a fixing process employs (Z, T_c) without extending (Z, T_c), *i.e.,* $t_{i-1} \to_{((Z,T_c),\varphi_i,t_{m_i})} t_i$.

Among these, cases (1) and (2) assume that Σ and D_m are fixed, respectively; (3) and (4) restrict the form of patterns in T_c; and case (5) restricts the form of eRs and adopts a simpler semantics for fixing input tuples.

One might think that fixed master data or positive patterns would simplify the analysis of eRs. Unfortunately, these do not help. Observe that in the lower-bound proofs of Theorems 1 and 2 (a) the master relation D_m is fixed, *i.e.,* it is independent of 3SAT instances, and (b) the tableau T_c consists of wildcard and constants only. From these the next corollary follows.

Corollary 3 *The consistency problem and the coverage problem remain* coNP*-complete even for* (1) *fixed master data* D_m *and* (2) *a positive tableau* T_c.

In contrast, special cases (1) and (4) indeed make our lives easier, as verified below.

Theorem 4 *The consistency problem and the coverage problem are in* PTIME *for either* (1) *a fixed set* Σ *of eRs or* (2) *a concrete pattern tableau* T_c.

Proof We consider a set Σ of eRs on schemas (R, R_m), a master relation D_m of R_m, and a region (Z, T_c), where there is a single tuple $t_c \in T_c$ only. When there are multiple tuples in T_c, we can test them one by one by using the PTIME algorithms for a single-tuple T_c.

Below we first show that if the consistency problem or the coverage problem is in PTIME for a concrete T_c, the problem is in PTIME for a fixed Σ. We then show that both problems are in PTIME for a concrete T_c.

(I) We first show that when Σ is fixed, we can construct a concrete T_c' from T_c such that (1) the size of T_c' is polynomially bounded by the size of (Σ, D_m) (2) (D_m, Σ) is consistent relative to (Z, T_c) iff it is consistent relative to (Z, T_c'), and (3) (Z, T_c) is a certain region for (Σ, D_m) iff (Z, T_c') is a certain region for (Σ, D_m).

The tableau T_c' is constructed as follows.

- Let Z_Σ be the set of R attributes that appear in Σ, $Z' = Z \cap Z_\Sigma$, and let dom be the active domain of Σ and D_m as defined in the proof of Theorem 1.

- The tableau $T_c' = \{ t_c' \mid t_c'[Z'] = t[Z']$ for all possible R tuples t such that $t \approx t_c[Z']$, $t[B] \in \mathsf{dom}$ for each attribute $B \in Z'$, and $t_c'[Z \setminus Z'] \approx t_c[Z \setminus Z']$ consisting of constants drawn from dom only$\}$. That is, all _ and $\bar{c}$ in t_c are instantiated to all possible values in dom, and hence all pattern tuples in T_c' contain concrete data values only.

Observe that T_c' contains at most $O(|\mathsf{dom}|^{|\Sigma|})$ pattern tuples since the number of possible R tuples is bounded by $O(|\mathsf{dom}|^{|\Sigma|})$. Hence the size of T_c' is a polynomial of the size of (Σ, D_m) when Σ is fixed.

We next show that (D_m, Σ) is consistent relative to (Z, T_c) iff it is consistent relative to (Z, T_c').

Consider an R tuple t such that for each attribute $B \in Z$, $t[B] \in \mathsf{dom}$. If $t \not\approx t_c[Z]$, then it is easy to see that t has a unique fix, but does not have a certain fix. If $t \approx t_c[Z]$, we can verify that t has a unique fix by (Σ, D_m) *w.r.t.* (Z, T_c) iff it has a unique fix by (Σ, D_m) *w.r.t.* (Z, T_c'). From these the statement follows.

Similarly, we show that (Z, T_c) is a certain region for (D_m, Σ) iff (Z, T_c') is a certain region for (D_m, Σ).

Putting these together, we have shown that if the consistency or coverage problem is in PTIME for a concrete T_c, the problem is in PTIME for a fixed Σ.

(II) We next show that the consistency problem for a concrete T_c is in PTIME, by giving a PTIME algorithm that takes (Σ, D_m) and (Z, T_c) as input, and returns 'yes' iff (Σ, D_m) is consistent relative to (Z, T_c). Recall that it suffices to consider single-tuple $T_c = \{t_c\}$.

Below we first present the algorithm, and then show that the algorithm runs in PTIME. Finally, we verify the correctness of the algorithm.

(1) We first present the algorithm.

(a) Let t be an R tuple such that $t[Z] = t_c$ and $t[R \setminus Z] = (\_, \ldots, \_)$, and let $Z_b = Z$. As will be seen shortly, any R tuple t with $t[Z] = t_c$ is allowed. We simply use wildcards to denote that t is any R tuple marked by t_c. Here Z_b is used to store the initial Z, and remains unchanged in the entire process.

(b) Let a set $\mathsf{dep}(A) = \emptyset$ for each attribute $A \in R$. Here $\mathsf{dep}(A)$ is used to remember $\text{LHS}(\varphi)$ for all $\varphi \in \Sigma$ such that $t[A]$ is updated and validated by making use of φ and some mater tuple t_m in D_m.

(c) Let $S = \{(\varphi_1, t_{m_1}), \ldots, (\varphi_k, t_{m_k})\}$ be the set of all rule-tuple pairs such that for each $i \in [1, k]$, (1) $t_{m_i} \in D_m$ and $\varphi_i \in \Sigma$, (2) $(\text{LHS}(\varphi_i) \cup \text{LHS}_p(\varphi_i)) \subseteq Z$, but $\text{RHS}(\varphi_i) \notin Z$; (3) $t[\text{LHS}(\varphi_i)] = t_{m_i}[\text{LHS}_m(\varphi_i)]$; and (4) tuple t matches the pattern tuple in φ_i.

(d) The algorithm returns 'yes' if the set S is empty, *i.e.,* the tuple t reaches a fix point. Otherwise it continues.

(e) If there exist $i, j \in [1, k]$ such that $\text{RHS}(\varphi_i) = \text{RHS}(\varphi_j)$ and $t_{m_i}[\text{RHS}_m(\varphi_i)] \neq t_{m_j}[\text{RHS}_m(\varphi_j)]$, then the algorithm returns 'no', and it continues otherwise.

(f) For each $i \in [1, k]$, let $\mathsf{dep}(\text{RHS}(\varphi_i)) := \mathsf{dep}(\text{RHS}(\varphi_i)) \cup \{\text{LHS}(\varphi_i)\}$, $t[\text{RHS}(\varphi_i)] := t_{m_i}[\text{RHS}_m(\varphi_i)]$; and expand $Z := Z \cup \{\text{RHS}(\varphi_1), \ldots, \text{RHS}(\varphi_k)\}$.

(g) If there exist an eR φ in Σ and a master tuple t_m in D_m such that (1) φ and t_m can be applied to tuple t; (2) $\text{LHS}(\varphi) \subseteq Z$, $\text{RHS}(\varphi) \in Z \setminus Z_b$; and (3) $t_m[\text{RHS}_m(\varphi)] \neq t[\text{RHS}(\varphi)]$, then the algorithm does the following:

- If for each attribute $A \in \text{LHS}(\varphi)$, there exists an $X \in \mathsf{dep}(A)$ with $\text{RHS}(\varphi) \notin X$, then it returns 'no'.

(h) The algorithm repeats the process from step (c).

(2) To see that the algorithm is in PTIME, observe the following: (i) each time Z is expanded by at least one more attribute; (ii) there are $|\Sigma| \times |D_m|$ rule-tuple pairs, and once such a pair is applied to the tuple t at step (c) or (g), it will not be considered again; and (iii) all steps alone can be done in PTIME. Putting these together, the algorithm indeed runs in PTIME.

(3) We next verify the correctness of the algorithm.

Assume first that the algorithm returns 'no'. Then we show that (Σ, D_m) is not consistent relative to (Z, T_c). Note that the algorithm returns 'no' at steps (e) and (g) only. In both cases, it is obvious that (Σ, D_m) is not consistent relative to (Z, T_c).

Conversely, assume that the algorithm returns 'yes'. Then we prove that (Σ, D_m) is consistent relative to (Z, T_c) by contradiction. Assume that (Σ, D_m) is not consistent relative to (Z, T_c). Then there exist two distinct fixes t' and s' for an R tuple t such that (a) $t' \neq s'$, (b) $t_0 = t \to^*_{((Z,T_c),\Sigma,D_m)} t_k = t'$, and (c) $s_0 = t \to^*_{((Z,T_c),\Sigma,D_m)} s_h = s'$. That is, there exist two finite sequences $L_1 = [t_0 = t, t_1, \ldots, t_k = t']$ and $L_2 = [s_0 = t, s_1, \ldots, s_h = s']$ such that $k, h \leq |R|$, and for each $i \in [1, k]$, $j \in [1, h]$, there exist $\varphi_i, \varphi_j \in \Sigma$ and $t_{m_i}, t_{m_j} \in D_m$ that satisfy the following:

- $t_i[Z] = t_0[Z] = t[Z]$ and $s_j[Z] = s_0[Z] = t[Z]$;
- $t_{i-1} \to_{((Z_{i-1},T_{i-1}),\varphi_i,t_{m_i})} t_i$, where $(Z_0, T_0) = (Z, T_c)$ and $(Z_i, T_i) = \mathsf{ext}(Z_{i-1}, T_{i-1}, \varphi_i)$;
- $s_{j-1} \to_{((Z_{j-1},T_{j-1}),\varphi_j,t_{m_j})} s_j$, where $(Z_0, T_0) = (Z, T_c)$ and $(Z_j, T_j) = \mathsf{ext}(Z_{j-1}, T_{j-1}, \varphi_j)$; and
- for all $\varphi \in \Sigma$ and $t_m \in D_m$, $t' \to_{((Z_k,T_k),\varphi,t_m)} t'$ and $s' \to_{((Z_h,T_h),\varphi,t_m)} s'$.

To see that these lead to a contradiction, we first define a partition $\{P_1, \ldots, P_g\}$ of the rule-tuple pairs (φ_i, t_{m_i}) $(i \in [1, k])$ and (φ_j, t_{m_j}) $(j \in [1, h])$ involved in the two sequences L_1 and L_2. Along the same lines as step (c) of

the algorithm, those pairs that are *applicable* to t at the same time are processed together, and form a distinct partition P_l ($1 \leq l \leq g$). Note that for any $l_1 \neq l_2 \in [1, g]$, $P_{l_1} \cap P_{l_2} = \emptyset$.

To see the contradiction, in L_1 and L_2, let B be the first R attribute such that (i) $t_i[B] \neq s_j[B]$ ($i \in [1, k]$, $j \in [1, h]$) (ii) $t_{i-1} \rightarrow_{((Z_{i-1}, T_{i-1}), \varphi_i, t_{m_i})} t_i$, and (iii) $s_{j-1} \rightarrow_{((Z_{j-1}, T_{j-1}), \varphi_j, t_{m_j})} s_j$. Assume that $(\varphi_i, t_{m_i}) \in P_{l_1}$ and $(\varphi_j, t_{m_j}) \in P_{l_2}$ ($l_1, l_2 \in [1, g]$). There are three cases to consider: (i) $l_1 = l_2$ (ii) $l_1 < l_2$, and (iii) $l_1 > l_2$.

(i) If $l_1 = l_2$, the conflict can be detected at step (e) of the algorithm when it updates the attribute B. Thus the algorithm would have returned 'no'.

(ii) If $l_1 < l_2$, the conflict can be found at step (g) when the algorithm updates the attribute B using (s_j, t_{m_j}). Thus the algorithm would also have returned 'no'.

(iii) If $l_1 > l_2$, the algorithm would have returned 'no' as well, by the same analysis as (b).

That is, all three cases contradict our assumption.

Putting all these together, we have shown that the algorithm correctly determines whether (Σ, D_m) is consistent relative to (Z, T_c), in PTIME.

(III) Finally, we show that the coverage problem for a concrete T_c is also in PTIME. Indeed, the PTIME algorithm given above can be adapted and applied here, while it only returns 'yes' at step (d) if both the set S is empty and if the tuple t consists of constants only. □

Furthermore, special case (5) identified above also simplifies the consistency and coverage analyses.

Theorem 5 *The consistency problem and the coverage problem are in* PTIME *when direct fixes are considered.*

Proof Consider a set Σ of eRs defined on schemas (R, R_m), a master relation D_m of R_m and a region (Z, T_c). As in the proof of Theorem 4, we assume *w.l.o.g.* that there is a single tuple $t_c \in T_c$ only.

Below we first show that the consistency problem for direct fixes is in PTIME, based on which we then show that the same result holds for the coverage problem.

(I) We first show how to check the consistency for direct fixes via a set of SQL queries, which yields a PTIME algorithm for the problem.

Let Σ_Z be the set of eRs φ in Σ such that $\text{LHS}(\varphi) \subseteq Z$, but $\text{RHS}(\varphi) \notin Z$. For any two eRs φ_1 and φ_2 in Σ_Z with $\text{RHS}(\varphi_1) = \text{RHS}(\varphi_2)$, we define an SQL query Q_{φ_1, φ_2} such that (Σ, D_m) is consistent relative to (Z, T_c) iff all the queries return an *empty* set.

We first define an SQL query Q_φ for an eR $\varphi = ((X, X_m) \rightarrow (B, B_m), t_p[X_p])$ in Σ_Z, as follows.

Q_φ: **select distinct** (X_m, B_m) **as** (X, B)
from $\mathsf{R_m}$
where $\mathsf{R_m}.\mathsf{X_{p_m}} \approx t_p[X_p]$ **and** $\mathsf{R_m}.\mathsf{X_m} \approx t_c[X]$,

where $X_{p_m} \subseteq X_m$ is the list of attributes corresponding to X_p. Recall that $X_p \subseteq X$ and $X \subseteq Z$ for direct fixes. Here $\mathsf{R_m}.\mathsf{X_{p_m}} \approx t_p[X_p]$ is a disjunction of $(t_p[A] = \_)$**or** $(t_p[A] = c$ & $\mathsf{R_m}.\mathsf{A_m} = t_p[A])$**or** $(t_p[A] = \bar{c}$ & $\mathsf{R_m}.\mathsf{A_m} \langle\rangle t_p[A]))$ for each attribute $A \in X_p$, where $A_m \in X_m$ is the attribute corresponding to A. It is similar for $\mathsf{R_m}.\mathsf{X_m} \approx t_c[X]$. Intuitively, Q_φ returns (partial) master tuples that both match the pattern tuple t_p of φ and the pattern tuple t_c in T_c. We also use $Q_\varphi(X, B)$ and $Q_\varphi(X)$ to denote the projected results of Q_φ on $X \cup \{B\}$ and X, respectively.

We also define SQL query Q_{φ_1, φ_2}. Assume *w.l.o.g.* that $\varphi_1 = ((X_1 X, X_{m_1} X_m) \rightarrow (B, B_{m_1}), t_{p_1}[X_{p_1}])$ and $\varphi_2 = ((X_2 X, X_{m_2} X'_m) \rightarrow (B, B_{m_2}), t_{p_2}[X_{p_2}])$ such that $X_1 \cap X_2 = \emptyset$ and $|X| = |X_m| = |X'_m|$. Note that here X may be empty.

Q_{φ_1, φ_2}: **select** $R_1.X_1, R_1.X, R_2.X_2$
from $Q_{\varphi_1}(X_1 X, B)$ **as** R_1, $Q_{\varphi_2}(X_2 X, B)$ **as** R_2
where $R_1.X = R_2.X$ **and** $R_1.B \neq R_2.B$.

Intuitively, Q_{φ_1, φ_2} returns those (partial) master tuples that may introduce conflicts when fixing R tuples.

When the semantics of direct fixes is considered, (Σ, D_m) is consistent relative to $(Z, \{t_c\})$ if for all eRs φ_1 and φ_2 in Σ_Z, the query Q_{φ_1, φ_2} returns an empty result. Note that (1) Q_φ can be evaluated by scanning the master relation D_m once, and hence it can be done in $O(|\varphi||D_m|)$ time, where $|\varphi|$ is the size of φ and $|D_m|$ is the number of master tuples in D_m, respectively; and (2) Q_{φ_1, φ_2} can be evaluated in $O(|\varphi_1||\varphi_2||D_m|^2)$ time. Hence, the consistency problem is in $O(|\Sigma|^2|D_m|^2)$ time (PTIME) for direct fixes, where $|\Sigma|$ is the size of Σ.

(II) For the coverage problem, observe that (Z, T_c) is a certain region for (Σ, D_m) iff

(1) (Σ, D_m) is consistent relative to (Z, T_c); and

(2) for each $B \in R \setminus Z$, there exists an eR $\varphi = ((X, X_m) \rightarrow (B, B_m), t_p[X_p])$ in Σ such that (a) $X \subseteq Z$ (b) $t_c[X]$ consists of constants only (c) $t_p[X_p] \approx t_c[X_p]$, and (d) there is a master tuple $t_m \in D_m$ with $t_m[X_m] = t_c[X]$.

Conditions (1) and (2) can be checked in $O(|\Sigma|^2|D_m|^2)$ time and $O(|R||\Sigma||D_m|)$ time, respectively. Hence the coverage problem for direct fixes is in $O(|\Sigma|^2|D_m|^2)$ time (PTIME) since $|R|$ is bounded by $|\Sigma|$. □

4.2 The complexity of computing certain regions

We next study three fundamental problems in connection with computing certain regions, when regions are either partially given or not given at all.

To derive a certain region (Z, T_c) from (Σ, D_m), one wants to know whether a given list Z of attributes could make a certain region by finding a *nonempty* T_c.

The *Z-validating* problem is to decide, given (Σ, D_m) and a list Z of distinct attributes, whether there exists a

non-empty pattern tableau T_c such that (Z, T_c) is a certain region for (Σ, D_m).

Another question is to determine, if Z can make a certain region by finding a nonempty T_c, how large T_c is. Let (Z, T_c) be a certain region for (Σ, D_m). For any pattern tuple $t_c \in T_c$, we require the following:

(1) $t_c[A] = \_$ for all attributes A not appearing in Σ;

(2) $t_c[A]$ is replaced with v (resp. $\bar{v}$) if $t_c[A] = c$ (resp. $\bar{c}$) and c is a constant *not* appearing in Σ or D_m. Here v is a variable denoting any constant not in Σ or D_m.

Note that these requirements do not lose generality. It is easy to verify for any certain region (Z, T_c), we can find an equivalent one (with no more pattern tuples) satisfying the two conditions. Moreover, these allow us to deal with only a finite number of pattern tuples, and to focus on the essential properties of the problems.

The *Z-counting* problem is to count, given (Σ, D_m) and a list Z of distinct attributes, the number of distinct pattern tuples that can be found from (Σ, D_m) to build a tableau T_c such that (Z, T_c) is a certain region.

Both problems are beyond reach in practice, as shown below. In particular, the Z-counting problem is as hard as finding the number of truth assignments that satisfy a given 3SAT instance [33].

Theorem 6 *The Z-validating problem is* NP*-complete.*

Proof We first show that the problem is in NP. We then show that the problem is NP-hard.

(I) We show that the problem is in NP, by providing an NP algorithm that, given (Σ, D_m) and a list Z of distinct attributes as input, returns 'yes' iff there exists a non-empty pattern tableau T_c such that (Z, T_c) is a certain region for (Σ, D_m). Let dom be the active domain dom of Σ and D_m as given in the proof of Theorem 1.

The NP algorithm works as follows.

(a) Guess a tuple t_c such that for each attribute $A \in Z$, $t_c[A] \in$ dom, *i.e.,* t_c consists of constants only.

(b) If $(Z, \{t_c\})$ is a certain region for (Σ, D_m), then the algorithm returns 'yes'.

By Theorem 4, step (b) can be done in PTIME. Hence the algorithm is in NP.

The correctness of the algorithm follows from the observation below, which can be readily verified. Given Z, there exists a non-empty pattern tableau T_c such that (Z, T_c) is a certain region for (Σ, D_m) iff there exists a pattern tuple t_c consisting of values from dom such that $(Z, \{t_c\})$ is a certain region for (Σ, D_m).

(II) We next show that the problem is NP-hard, by reduction from 3SAT. Given an instance ϕ of the 3SAT problem as described in the proof of Theorem 1, we construct an instance of the Z-validating problem consisting of: (a) two relational schemas R and R_m, (b) a master relation D_m of R_m, (c) a set Σ of eRs, and (d) a list Z of distinct attributes of R. We show that there exists a non-empty pattern tableau T_c that yields a certain region (Z, T_c) for (Σ, D_m) iff ϕ is satisfiable.

(1) We first define the Z-validating instance.

(a) The two schemas are $R(X_1, \ldots, X_m, C_1, \ldots, C_n, V)$ and $R_m(B_1, B_2, B_3, C, V_1, V_0)$, respectively, in which all the attributes have an integer domain.

Intuitively, for each R tuple t, $t[X_1 \ldots X_m]$, $t[C_1 \ldots C_n]$ and $t[V]$ specify a truth assignment ξ for the variables $x_1, \ldots, x_m$ of ϕ, the truth values of the clauses $C_1, \ldots, C_n$, and the truth value of ϕ under ξ, respectively.

(b) The master relation D_m consists of eight tuples:

	B_1	B_2	B_3	C	V_1	V_0
t_{m_0}:	0	0	0	1	1	0
t_{m_1}:	0	0	1	1	1	0
t_{m_2}:	0	1	0	1	1	0
t_{m_3}:	0	1	1	1	1	0
t_{m_4}:	1	0	0	1	1	0
t_{m_5}:	1	0	1	1	1	0
t_{m_6}:	1	1	0	1	1	0
t_{m_7}:	1	1	1	1	1	0

Here (1) $t_{m_0}[C, V_1, V_0] = \cdots = t_{m_7}[C, V_1, V_0] = (1, 1, 0)$, and (2) $t_{m_0}[B_1, B_2, B_3]$, …, and $t_{m_7}[B_1, B_2, B_3]$ together enumerate the eight truth assignments of a three-variable clause, ranging from (0, 0, 0) to (1, 1, 1).

(c) The set Σ consists of $3n$ eRs.

We encode each clause $C_j = y_{j_1} \vee y_{j_2} \vee y_{j_3}$ ($j \in [1, n]$) of ϕ with three eRs: $\varphi_{j,1}$, $\varphi_{j,2}$ and $\varphi_{j,3}$, where

- $\varphi_{j,1} = ((X_{p_{j1}} X_{p_{j2}} X_{p_{j3}}, B_1 B_2 B_3) \to (C_j, C), ())$,
- $\varphi_{j,2} = ((X_{p_{j1}} X_{p_{j2}} X_{p_{j3}}, B_1 B_2 B_3) \to (V, V_1), ())$,
- $\varphi_{j,3} = ((X_{p_{j1}} X_{p_{j2}} X_{p_{j3}}, B_1 B_2 B_3) \to (V, V_0), t_{p_j}[X_{p_{j1}} X_{p_{j2}} X_{p_{j3}}])$ such that $t_{p_j}[X_{p_{j1}} X_{p_{j2}} X_{p_{j3}}]$ is the only truth assignment that makes clause C_j false.

(d) We define the attribute list Z as $X_1, \ldots, X_m$.

(2) We next show that there exists a non-empty tableau T_c such that (Z, T_c) is a certain region for (Σ, D_m) iff the 3SAT instance ϕ is satisfiable.

Assume first that there exists a non-empty T_c such that (Z, T_c) is a certain region for (Σ, D_m). We show that the 3SAT instance ϕ is satisfiable. Observe that for any pattern tuple $t_c \in T_c$, $t_c[X_i] \in \{0, 1\}$ for $i \in [1, m]$ since $(Z, \{t_c\})$ is a certain region for (Σ, D_m). We prove that $t_c[X_1, \ldots, X_m]$ is a satisfying truth assignment of ϕ by contradiction. If $t_c[X_1, \ldots, X_m]$ does not satisfy ϕ, then there exists a clause $C_j (1 \le j \le n)$ such that $t_c[X_{p_{j1}}, X_{p_{j2}}, X_{p_{j3}}]$ makes C_j false. Then eRs $\varphi_{j,2}$ and $\varphi_{j,3}$ are both applicable to any R tuples t if $t \approx t_c$, which updates $t[V]$ to 1 by $\varphi_{j,2}$, but $t[V]$ to

0 by $\varphi_{j,3}$. That is, $(Z, \{t_c\})$ is not a certain region for (Σ, D_m), which contradicts the assumption above.

Conversely, assume that the instance ϕ is satisfiable. We show that there exists a pattern tuple t_c such that $(Z, \{t_c\})$ is a certain region for (Σ, D_m). Let $t_c[X_1, \ldots, X_m]$ be a satisfying truth assignment of ϕ, which exists since ϕ is satisfiable. One can easily verify that $(Z, \{t_c\})$ is a certain region for (Σ, D_m). □

A closer look at the reduction in the proof of Theorem 6 tells us the following. In particular, in contrast to Theorem 5, the Z-validating problem remains intractable even when direct fixes are considered.

Corollary 7 *The Z-validating problem remains* NP*-complete even when we consider* (1) *fixed master data D_m,* (2) *a positive pattern tableau T_c,* (3) *a concrete pattern tableau T_c, or* (4) *direct fixes.*

However, when fixing Σ, the Z-validating problem becomes much simpler, as shown below.

Proposition 8 *The Z-validating problem is in* PTIME *given a fixed set Σ of eRs.*

Proof We show that the problem is in PTIME by providing a PTIME algorithm that takes (Σ, D_m) and a list Z of distinct attributes as input, returns 'yes' iff there exists a non-empty pattern tableau T_c such that (Z, T_c) is a certain region for (Σ, D_m). Let dom be the active domain dom as given in the proof of Theorem 1.

The algorithm works as follows. Let t_c be a pattern tuple over attributes Z such that $t_c[A] = \_$ for all attributes $A \in Z$ not appearing in Σ, and $t_c[B]$ is either c or $\bar{c}$, where c is a value drawn from dom, for all the other attributes B. For all such pattern tuples t_c, the algorithm checks whether $(Z, \{t_c\})$ is a certain region for (Σ, D_m), and if so, it returns 'yes'.

To see that the algorithm runs in PTIME, observe the following: (1) the number of pattern tuples t_c is bounded by $O(|\mathsf{dom}|^{|\Sigma|})$, a polynomial of Σ and D_m when Σ is fixed; and (2) by Theorem 4, it is in PTIME to check whether $(Z, \{t_c\})$ is a certain region.

For the correctness, observe that given Z, there exists a non-empty pattern tableau T_c such that (Z, T_c) is a certain region for (Σ, D_m) iff there exists a pattern tuple t_c inspected by the algorithm, such that $(Z, \{t_c\})$ is a certain region for (Σ, D_m). □

We next investigate the Z-counting problem.

Theorem 9 *The Z-counting problem is* #P*-complete.*

Proof The Z-counting problem is obviously in #P since it is the counting version of the Z-validating problem, which is NP-complete as shown by Theorem 6.

We next show that the Z-counting problem is #P-hard by a *parsimonious reduction* [3] from the #3SAT problem, which is #P-complete [3,33]. Given a 3SAT formula, the #3SAT problem counts the number of satisfying truth assignments, *i.e.,* the problem is the counting version of the 3SAT problem. Recall that there exists a parsimonious reduction from counting problems $\#A$ to $\#B$ if there is a polynomial time reduction f such that for all instances x and its solution y of A, $|\{y \mid (x, y) \in A\}| = |\{z \mid (f(x), z) \in B\}|$ [3].

We show that the reduction given in the proof of Theorem 6 is already parsimonious. As argued there, for all pattern tuples t_c such that $(Z, \{t_c\})$ is a certain region for (Σ, D_m), $t_c[A_1 \ldots A_m]$ is a satisfying truth assignment for the 3SAT formula, and vice versa. This implies that the number of satisfying truth assignments for the 3SAT formula is exactly equal to the number of pattern tuples t_c in a pattern tableau T_c such that (Z, T_c) is a certain region for (Σ, D_m). □

From Theorems 6, 9 and Corollary 7 it follows:

Corollary 10 *The Z-counting problem remains* #P*-complete even when we consider* (1) *fixed master data D_m,* (2) *a positive pattern tableau T_c,* (3) *a concrete pattern tableau T_c, or* (4) *direct fixes.*

When only a fixed set Σ of eRs is considered, the Z-counting problem becomes easier. This is consistent with Proposition 8.

Proposition 11 *The Z-counting problem is in* PTIME *given a fixed set Σ of eRs.*

Proof We show this by giving a PTIME algorithm that counts the number of pattern tuples t_c such that $(Z, \{t_c\})$ is a certain region for (Σ, D_m).

The algorithm is a revision of the PTIME algorithm given in the proof of Proposition 8, by simply adding a counter that keeps track of the number of pattern tuples t_c such that $(Z, \{t_c\})$ is a certain region for (Σ, D_m). The revised algorithm runs in PTIME, and its correctness can be verified along the same lines as its counterpart given in the proof of Proposition 8. □

Certain regions with minimum Z. One would naturally want a certain region (Z, T_c) with a "small" Z, such that the users only need to assure the correctness of a small number of attributes in input tuples.

The *Z-minimum* problem is to decide, given (Σ, D_m) and a positive integer K, whether there exists a list Z of distinct attributes such that (a) $|Z| \leq K$ and (b) there exists a *non-empty* pattern tableau T_c such that (Z, T_c) is a certain region for (Σ, D_m).

This problem is also intractable, as shown below.

Theorem 12 *The Z-minimum problem is* NP*-complete.*

Proof We first show that the problem is in NP. We then show that the problem is NP-hard.

(I) We show that the problem is in NP by giving an NP algorithm. Given (Σ, D_m) and a positive integer K, the algorithm returns 'yes' iff there exists a list Z of distinct attributes such that (a) $|Z| \leq K$, and (b) there exists a non-empty pattern tableau T_c such that (Z, T_c) is a certain region for (Σ, D_m)

The algorithm works as follows.

(a) Guess a list Z of at most K distinct R attributes, and guess a pattern tuple t_c with attributes in Z as described in the proof of Theorem 6.

(b) If $(Z, \{t_c\})$ is a certain region for (Σ, D_m), then the algorithm returns 'yes'.

One can easily verify that the algorithm is correct. In addition, the algorithm is in NP since by Theorem 4, step (b) can be done in PTIME.

(II) We next show the NP-hardness of the Z-minimum problem by reduction from the *set covering* (SC) problem, which is known to be NP-complete (cf. [33]).

Given a finite set $U = \{x_1, \ldots, x_n\}$ of elements, a collection $S = \{C_1, \ldots, C_h\}$ of subsets of U, and a positive integer $K \leq h$, the SC problem asks whether there exists a cover for S of size K or less, *i.e.,* a subset $S' \subseteq S$ such that $|S'| \leq K$, and every element of U belongs to at least one member of S'.

Given an instance of the SC problem, we construct an instance of the Z-minimum problem consisting of (a) two relational schemas R and R_m, and (b) a master relation D_m of R_m. We show that there exists a list Z of distinct attributes such that (a) $|Z| \leq K$ and (b) there exists a non-empty pattern tableau T_c such that (Z, T_c) is a certain region for (Σ, D_m) iff there is a cover of size K or less for the instance of the SC problem.

(1) We first define the Z-minimum instance as follows.

(a) We use $R(C_1, \ldots, C_h, X_{1,1}, \ldots, X_{1,h+1}, \ldots, X_{n,1}, \ldots, X_{n,h+1})$ as the schema of input data, and $R_m(B_1, B_2)$ as the master data schema, in which all attributes have an integer domain.

Intuitively, attribute C_j ($j \in [1, h]$) is to encode the member C_j in S, and the $h+1$ attributes $X_{i,1}, \ldots, X_{i,h+1}$ ($i \in [1, n]$) together denote the element x_i in U.

(b) The master relation D_m consists of a single master tuple $t_m = (1, 1)$.

(c) The set Σ consists of $(h+1)\sum_{j=1}^{h} |C_j| + h$ eRs.

We encode each member $C_j = \{x_{j_1}, \ldots, x_{j_{|C_j|}}\}$ ($j \in [1, h]$) of S with $(h+1)|C_j| + 1$ eRs, where

- for each $x_{j_i} \in C_j$, $\varphi_{j,i,1} = ((C_j, B_1) \rightarrow (X_{j_i}, B_2), ())$, where X_{j_i} ranges over $\{X_{j_i,1}, \ldots, X_{j_i,h+1}\}$, and
- $\varphi_{j,2} = ((X_{j_1,1} \ldots X_{j_1,h+1} \ldots X_{j_{|C_j|},1} \ldots X_{j_{|C_j|},h+1}, B_1 \ldots B_1) \rightarrow (C_j, B_2), ())$.

Intuitively, when identifying a list Z of h attributes or less, these eRs ensure that the attributes are taken from $\{C_1, \ldots, C_h\}$ only.

(2) We now verify the correctness of the reduction.

Assume first that there exists a list Z of distinct attributes such that $|Z| \leq K$ and there exists a non-empty pattern tableau T_c such that (Z, T_c) is a certain region for (Σ, D_m). Then we construct a cover of size K or less for the instance of the SC problem. Let $Z' = Z \cap \{C_1, \ldots, C_h\}$ and S' be the set of subsets in S of the SC instance denoted by Z'. We prove that S' is a cover for the SC instance by showing that for each $x_i \in U$, there exists a $C_j \in S'$ such that $x_i \in C_j$. Indeed, if there exists no such C_j, the set Σ of eRs requires us to include all the $h+1$ attributes X_{i_l} ($l \in [1, h+1]$) in Z. This, however, would have made $|Z| > h$, which contradicts the assumption that $|Z| \leq K \leq h$.

Conversely, assume that there is a cover S' of size K or less for the instance of the SC problem. We show that there exists a list Z of distinct attributes of size K or less and a non-empty pattern tableau T_c on Z such that (Z, T_c) is a certain region for (Σ, D_m). Let Z be the list of distinct attributes denoted by the cover S', and $t_c = (1, \ldots, 1)$ be a pattern tuple. Then $|Z| \leq K$, and $(Z, \{t_c\})$ is a certain region for (Σ, D_m).

Parts (I) and (II) together show that the Z-minimum problem is NP-complete. □

Observe that the reduction in the proof of Theorem 12 utilizes a fixed master relation D_m and a concrete tableau T_c. Hence we have the following.

Corollary 13 *The Z-minimum problem remains* NP*-complete even when we consider* (1) *fixed master data* D_m*,* (2) *a positive pattern tableau* T_c*, or* (3) *a concrete pattern tableau* T_c*.*

When direct fixes are considered, the Z-minimum problem remains intractable, as opposed to Theorem 5.

Theorem 14 *The Z-minimum problem remains* NP*-complete even when direct fixes are considered.*

The problem is in NP by Theorem 12. It is verified NP-hard by reduction from the SC problem along the same lines as the proof of Theorem 12. We defer the proof to the appendix due to the space constraint.

Having seen Propositions 8 and 11, it is not surprising to find that the Z-minimum problem becomes tractable for a fixed set Σ of eRs, as shown below.

Proposition 15 *The Z-minimum problem is in* PTIME *given a fixed set* Σ *of eRs.*

Proof Consider a fixed set Σ of eRs defined on schemas (R, R_m), and a master relation D_m of R_m. We provide a PTIME algorithm that, given K and (Σ, D_m), checks whether there exists a list Z of no more than K distinct attributes and a non-empty pattern tableau T_c, such that (Z, T_c) is a certain region for (Σ, D_m).

Let Z_Σ and $\overline{Z_\Sigma}$ be two sets of R attributes that appear in Σ and do not appear in Σ, respectively. It is easy to verify that it suffices to consider Z with $\overline{Z_\Sigma} \subseteq Z$, since for any list Z of distinct R attributes, if $\overline{Z_\Sigma} \not\subseteq Z$, there exists *no* non-empty tableau T_c such that (Z, T_c) is a certain region for (Σ, D_m).

Let S be the collection of all lists of distinct attributes in Z_Σ, *i.e.,* for each $Z' \in S$, $Z' \subseteq Z_\Sigma$. When Σ is fixed, both S and Z_Σ have a fixed size.

Based on these we give the algorithm as follows. For each $Z' \in S$, we check whether there exists a non-empty tableau T_c such that (Z, T_c) is a certain region for (Σ, D_m), where $Z = \overline{Z_\Sigma} \cup Z'$. The algorithm is in PTIME. Indeed, by Proposition 8, when Σ is fixed, the checking can be done in PTIME. Moreover, as argued above, the cardinality of S is fixed. Hence the Z-minimum problem is in PTIME if Σ is fixed. □

Approximation hardness. Worse still, there exist no approximate algorithms for the (optimization version) Z-minimum problem with a reasonable bound. To show the approximation bound, we adopt L-reductions [33].

Let Π_1 and Π_2 be two minimization problems. An L-reduction from Π_1 to Π_2 is a quadruple (f, g, α, β), where f and g are two PTIME computable functions, and α and β are two constants, such that

- for any instance I_1 of Π_1, $I_2 = f(I_1)$ is an instance of Π_2 such that $\mathsf{opt}_2(I_2) \leq \alpha \cdot \mathsf{opt}_1(I_1)$, where opt_1 (resp. opt_2) is the objective of an optimal solution to I_1 (resp. I_2), and
- for any solution s_2 to I_2, $s_1 = g(s_2)$ is a solution to I_1 such that $\mathsf{obj}_1(s_1) \leq \beta \cdot \mathsf{obj}_2(s_2)$, where $\mathsf{obj}_1()$ (resp. $\mathsf{obj}_2()$) is a function measuring the objective of a solution to I_1 (resp. I_2).

We say an algorithm $\mathcal{A}$ for a minimization problem has performance guarantee ϵ ($\epsilon \geq 1$) if for any instance I, $\mathsf{obj}(\mathcal{A}(I)) \leq \epsilon \cdot \mathsf{opt}(I)$.

L-reductions retain approximation bounds [33].

Proposition 16 *If (f, g, α, β) is an* L-reduction *from problems Π_1 to Π_2, and there is a* PTIME *algorithm for Π_2 with performance guarantee ϵ, then there is a* PTIME *algorithm for Π_1 with performance guarantee $\alpha\beta\epsilon$* [33].

Leveraging Proposition 16, we next show the approximation-hardness of the Z-minimum problem.

Theorem 17 *Unless* NP =P, *the Z-minimum problem cannot be approximated within a factor of $c \log n$ in* PTIME *for a constant c.*

Proof It is known that the set covering (SC) problem cannot be approximated within a factor of $c \log n$ in PTIME for a constant c unless NP =P [35]. Hence it suffices to show that there exists an L-reduction from the SC problem to the Z-minimum problem.

We next construct such an L-reduction (f, g, α, β).

(1) Let f be the PTIME reduction given in the proof of Theorem 12. Given an SC instance I_1 as input, $I_2 = f(I_1)$ is a Z-minimum instance. It was shown there that for the instance I_1, there is a cover of size K or less iff for the instance I_2, there exist a $|Z| \leq K$ and a non-empty pattern tableau T_c such that (Z, T_c) is a certain region for (Σ, D_m). That is, the optimal objective $\mathsf{opt}_2(I_2)$ is equal to the optimal objective $\mathsf{opt}_1(I_1)$.

(2) We next define the function g. Let Z be a solution for the Z-minimum instance I_2, *i.e.,* a list of distinct attributes such that $|Z| \leq K$ and there exists a non-empty pattern tableau T_c such that (Z, T_c) is a certain region for (Σ, D_m). The function g constructs a cover for the SC instance I_1, as follows: Let $Z' = Z \cap \{C_1, \ldots, C_h\}$ and S' be the set of subsets in S of the SC instance I_1 denoted by Z'.

The function g is obviously computable in PTIME, and it was shown in the proof of Theorem 12 that S' is a cover for I_1 with size K or less. Hence given a solution s_2 for I_2, $s_1 = g(s_2)$ is a solution for I_1 such that $\mathsf{obj}_1(s_1) \leq \mathsf{obj}_2(s_2)$.

(3) Let $\alpha = \beta = 1$. Then we have $\mathsf{opt}_2(I_2) \leq \alpha \cdot \mathsf{opt}_1(I_1)$ and $\mathsf{obj}_1(s_1) \leq \beta \cdot \mathsf{obj}_2(s_2)$.

This completes the construction of the L-reduction. Thus by the approximation-hardness of SC [35] and Proposition 16, the Z-minimum problem cannot be approximated within $c \log n$ in PTIME unless NP = P. □

From Theorem 17 and Corollary 13, the result below immediately follows.

Corollary 18 *Unless* NP =P, *the Z-minimum problem cannot be approximated within a factor of $c \log n$ in* PTIME *for a constant c even when we consider* (1) *a fixed master relation D_m,* (2) *a positive pattern tableau T_c, or* (3) *a concrete pattern tableau T_c.*

Direct fixes do not make our lives easier when approximation is concerned either, similar to Theorem 14.

Theorem 19 *Unless* NP = P, *the Z-minimum problem cannot be approximated within a factor of $c \log n$ in* PTIME *for a constant c for direct fixes.*

The proof is similar to the proof of Theorem 17. It is verified by a L-reduction (f, g, α, β) from the SC problem.

As opposed to its counterpart of Theorem 17, here f is a PTIME function defined in terms of the PTIME reduction given in the proof of Theorem 14. We defer the proof to the appendix for the lack of space.

Theorems 17, 19 and Corollary 18 tell us that to find certain regions, it is necessary to develop heuristic algorithms. Such algorithms are provided in [20].

Summary. The complexity results are summarized in Table 3. Observe the following.

(1) The complexity bounds of all these problems remain unchanged in the presence of finite-domain attributes and in the absence of such attributes, as opposed to the analyses of CFDs [19], CINDs [8] and MDs [18].

(2) For a fixed set Σ of eRs, all the problems become PTIME computable, *i.e.,* fixed eRs simplify the analyses.

(3) For fixed master data D_m or a positive tableau T_c, all the problems remain intractable. That is, these special cases do not make our lives easier.

(4) When we consider direct fixes or a concrete tableau T_c, the consistency problem and the coverage problem become tractable, while the other problems remain intractable. That is, these special cases simplify the analyses, but only to an extent. Due to the space constraint, we encourage the interested reader to consult [20] for algorithms and experimental results based on direct fixes, which illustrate the practical impact of direct fixes.

5 An interactive framework for certain fixes

We next present a framework to find certain fixes for tuples at the point of data entry, by making use of editing rules and master data, and by interacting with users.

As depicted in Fig. 2, the framework is provided with a master relation D_m of schema R_m and a set Σ of eRs defined on (R, R_m). It takes a tuple t of schema R as input, and warrants to find a certain fix for t.

The algorithm underlying the framework, referred to as CertainFix, is shown in Fig. 3. The algorithm interacts with users and finds a certain fix for t as follows.

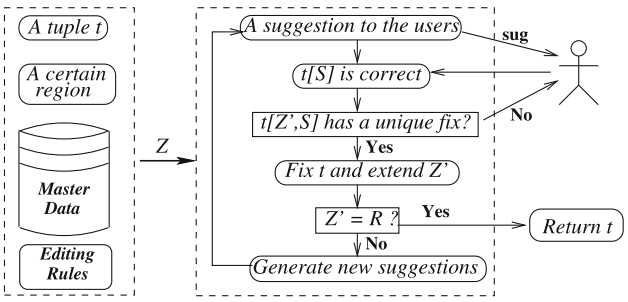

Fig. 2 Framework overview

Input: A tuple t, a certain region (Z, T_c), a set Σ of eRs, and a master relation D_m.
Output: A fixed tuple t.

1. sug := Z; /* Z is the initial suggestion */
2. Z' := $\emptyset$; flag := true;
3. **while** flag **do**
4. recommend sug to the users;
5. input S, where $t[S]$ is assured correct and $S \cap Z' = \emptyset$;
6. **if** $t[Z' \cup S]$ does not yield a unique fix **then** request new input from the users (back to line 4);
7. (t, Z') := TransFix $(t, Z' \cup S, D_m, \mathbf{G})$;
8. **if** $Z' = R$ **then** flag := false;
9. sug:= Suggest (t, Z', Σ, D_m);
10. **return** t.

Fig. 3 Algorithm CertainFix

(1) *Initialization* (lines 1–2). It first picks a precomputed certain region (Z, T_c), and recommends Z as the first suggestion to the users (line 1). For an input tuple t, if $t[Z]$ is assured correct and if $t[Z]$ matches a pattern tuple in T_c, then a certain fix can be found for t. It also uses a set Z' to keep track of the attributes of t that are already fixed, which is initially empty (line 2).

As shown by Theorems 12 and 17, it is intractable and approximation-hard to find a certain region with a minimum set Z of attributes. Nevertheless, an efficient heuristic algorithm is provided by [20], which is able to derive a set of certain regions from Σ and D_m based on a quality metric. Algorithm CertainFix picks the precomputed region (Z, T_c) with the highest quality. The region is computed once and is repeatedly used as long as Σ and D_m are unchanged.

(2) *Generating correct fixes* (lines 3–7). In each round of interaction with users, a set sug of attributes is recommended to the users as a suggestion (line 4), initially Z. The users get back with a set S of attributes that are asserted correct (line 5), where S may *not* necessarily be the same as sug. The algorithm validates $t[S]$ by checking whether $t[Z' \cup S]$ leads to a unique fix, *i.e.,* whether $t[S]$ is indeed correct. If $t[S]$ is invalid, the users are requested to revise the set S of attributes assured correct (line 6). If $t[Z' \cup S]$ yields a unique fix, procedure TransFix is invoked to find the fix, which extends Z' by including the newly corrected attributes (line 7). it finds the unique fix by invoking a procedure TransFix.

(3) *Generating new suggestions* (lines 8–9). If at this point, Z' covers all the attributes of R, the entire tuple t is validated and the fixed t is returned (lines 8, 10). Otherwise it computes a new suggestion from Σ and D_m via procedure Suggest (line 9), which is recommended to the users in the next round of interaction.

This process proceeds until a certain fix is found for t. All the attributes of t are corrected or validated, by using the users' input, the eRs and the master data.

The framework aims to guarantee the following. (a) *The correctness*. Each correcting step is justified by using the eRs and the master data. (b) *Minimizing user efforts*. It requires

Table 3 Summary of complexity results

Problems	General setting/infinite-domain attributes only					
	General	Fixed Σ	Fixed D_m	Positive T_c	Concrete T_c	Direct fixes
Consistency	coNP-complete (Theorem 1)	PTIME (Theorem 4)	coNP-complete (Corollary 3)	coNP-complete (Corollary 3)	PTIME (Theorem 4)	PTIME (Theorem 5)
Coverage	coNP-complete (Theorem 2)	PTIME (Theorem 4)	coNP-complete (Corollary 3)	coNP-complete (Corollary 3)	PTIME (Theorem 4)	PTIME (Theorem 5)
Z-validating	NP-complete (Theorem 6)	PTIME (Proposition 8)	NP-complete (Corollary 7)	NP-complete (Corollary 7)	NP-complete (Corollary 7)	NP-complete (Corollary 7)
Z-counting	#P-complete (Theorem 9)	PTIME (Proposition 11)	#P-complete (Corollary 10)	#P-complete (Corollary 10)	#P-complete (Corollary 10)	#P-complete (Corollary 10)
Z-minimum	NP-complete (Theorem 12)	PTIME (Proposition 15)	NP-complete (Corollary 13)	NP-complete (Corollary 13)	NP-complete (Corollary 13)	NP-complete (Theorem 14)
	non-approx* (Theorem 17)	PTIME (Proposition 15)	non-approx (Corollary 18)	non-approx (Corollary 18)	non-approx (Corollary 18)	non-approx (Theorem 19)

non-approx: cannot be approximated within $c \log n$ in PTIME for a constant c, unless P = NP

the users to validate a minimal number of attributes, while automatically deducing other attributes that are entailed correct. (c) *Minimal delays*. It improves the response time by reducing the latency for generating new suggestions at each interactive step.

Note that the users are not necessarily domain experts, as long as they can assure the correctness of certain attributes of input tuples that are required to match eRs and master tuples. In practice, different people may be responsible for entering and interpreting different attributes. Hence distinct attributes are often inspected and validated by different people.

In the rest of the section we present the details of the procedures and optimization techniques employed by CertainFix. Note that it is in PTIME to check whether $t[Z' \cup S]$ leads to a unique fix. Indeed, the PTIME algorithm given in the proof of Theorem 4 suffices to do the checking when $t[Z' \cup S]$ is treated as a pattern tuple, which consists of constants only and is hence concrete. Therefore, below we focus on TransFix and Suggest.

5.1 TransFix: Generating correct fixes

We first present procedure TransFix. It takes as input a tuple t, a master relation D_m, a set Σ of eRs, a set Z' of attributes such that $t[Z']$ has been validated. It finds a unique fix for t and extends Z' by including those newly validated attributes. While not all of the attributes of t may be validated, the procedure ensures that the attributes updated are correct.

Procedure TransFix represents Σ as a dependency graph **G**, which tells us the order of applying eRs.

Dependency graph. The *dependency graph* **G** of a set Σ of eRs is a directed graph (V, E). Each node $v \in V$ denotes an eR $\varphi_v = ((X_v, X_{m_v}) \to (B_v, B_{m_v}), t_{p_v}[X_{p_v}])$. There exists an edge $(u, v) \in E$ from node u to v if $B_u \cap (X_v \cup X_{p_v}) \neq \emptyset$. Intuitively, (u, v) indicates that whether φ_v can be applied to t depends on the outcome of applying φ_u to t. Hence φ_u is applied before φ_v.

The dependency graph of Σ remains unchanged as long as Σ is not changed. Hence it is computed once, and is used to repair all input tuples until Σ is updated.

Example 11 The set Σ_0 of eRs given in Example 3 consists of 9 eRs, fully expressed as follows:

φ_1: ((zip, zip) → (AC, AC), t_{p1} = ());
φ_2: ((zip, zip) → (str, str), t_{p2} = ());
φ_3: ((zip, zip) → (city, city), t_{p3} = ());
φ_4: ((phn, Mphn) → (FN, FN), t_{p4}[type] = (2));
φ_5: ((phn, Mphn) → (LN, LN), t_{p5}[type] = (2));
φ_6: (([AC, phn], [AC, Hphn])→(str, str), t_{p6}[type, AC]=(1,$\overline{0800}$));
φ_7: (([AC, phn], [AC, Hphn])→(city, city), t_{p7}[type, AC]=(1,$\overline{0800}$));
φ_8: (([AC, phn], [AC, Hphn])→(zip, zip), t_{p8}[type, AC]=(1,$\overline{0800}$));
φ_9: ((AC, AC) → (city, city), t_{p9}[AC] = ($\overline{0800}$)).

The dependency graph of Σ_0 is depicted in Fig. 4. Note that, for instance, there is an edge from φ_1 to φ_6 since the RHS of φ_1 (*i.e.,* {AC}) is the subset of LHS of φ_6 (*i.e.,* {AC, phn}); similarly for the other edges. □

Procedure. Procedure TransFix is given in Fig. 5. It validates attributes of t as follows. It first marks all the nodes in the dependency graph as *unusable* (line 1). It then collects those nodes (eRs) whose LHS and pattern attributes are validated, puts them in a set vset (line 2), and marks them as *usable* (line 3). Intuitively, for the eR φ_v represented by a usable v, the attributes in $t[X_v \cup X_{p_v}]$ have already been validated, and hence, φ_v can be possibly applied to t. The procedure uses

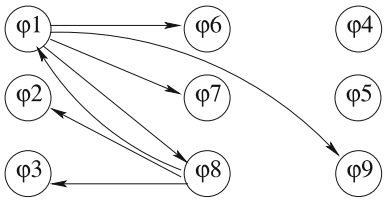

Fig. 4 An example dependency graph

Input: A tuple t, a set Z' of attributes, a master relation D_m, and a dependency graph $\mathbf{G}(V, E)$.
Output: A (partially) fixed tuple t and validated attributes Z'.
/* node u: $\varphi_u = ((X_u, X_{m_u}) \to (B_u, B_{m_u}), t_{p_u}[X_{p_u}])$ */;
1. mark u.usable := false for each $u \in V$;
2. vset := { $u \mid u \in V$ **and** $(X_u \cup X_{p_u}) \subseteq Z'$ };
3. mark u.usable := true for each $u \in$ vset;
4. uset := { $u \mid u \in V$ **and** $(X_u \cup X_{p_u}) \not\subseteq Z'$ **and** $(X_u \cup X_{p_u}) \cap Z' \neq \emptyset$ };
5. **while** vset $\neq \emptyset$ **do**
6. v := an eR picked from vset; vset := vset $\setminus \{v\}$;
7. **if** $\exists t_m \in D_m$, (t_m, φ_v) applies to t **and** $B_v \notin Z'$ **then**
8. $t[B_v] := t_m[B_{m_v}]$; $Z' := Z' \cup \{B_v\}$;
9. **for each** edge $(v, u) \in E$ **do**
10. **if** $u \in$ uset **and** $(X_u \cup X_{p_u}) \subseteq Z'$ **then**
11. vset := vset $\cup \{u\}$; uset := uset $\setminus \{u\}$;
12. u.usable := true;
13. **else if** $u \notin$ uset **and** u.usable = false **then**
14. **if** $\{B_v\} = (X_u \cup X_{p_u})$ **then** vset := vset $\cup \{u\}$;
15. **else if** $B_v \in (X_u \cup X_{p_u})$ **then** uset := uset $\cup \{u\}$;
16. **return** (t, Z');

Fig. 5 Procedure TransFix

another set uset to maintain those eRs that are not yet usable but may become usable later on (line 4).

The procedure iteratively makes use of eRs in vset to fix attributes of t, and upgrades eRs from uset to vset (lines 5–15). In each iteration, a node v is randomly picked and removed from vset (line 6). If a master tuple t_m can be found such that (t_m, φ_v) applies to t, and moreover, if for the RHS attribute B_v of φ_v, $t[B_v]$ is not yet validated (line 7), then $t[B_v]$ is fixed using φ_v and t_m, and B_v is included in Z' (line 8).

The procedure then inspects each edge (v, u) emanating from v, to examine whether φ_u becomes usable (lines 9–15). If u is in the candidate set uset, and moreover, if $\text{RHS}(\varphi_u)$ and $\text{RHS}_p(\varphi_u)$ are included in the extended Z' (line 10), then u is added to vset, removed from uset (line 11), and is marked usable (line 12). Otherwise, if u is in neither vset nor uset (line 13), node u is added to vset if $X_u \cup X_{p_u}$ is a singleton set containing B_v (line 14), or to uset if $X_u \cup X_{p_u}$ contains other attributes besides B_v (line 15). Finally, the tuple t is returned along with the extended Z' (line 16).

Example 12 Consider tuple t_1 and the master data D_m of Fig. 1, and the set Σ_0 of eRs given in Example 11. Assume that Z consists of zip only. Given D_m, Z and the dependency graph $\mathbf{G}$ of Fig. 4, we show how procedure TransFix fixes attributes of t_1. As indicated in the table below, in iteration 0, uset is empty, while φ_1 is in vset since its $X \cup X_p \subseteq Z'$; similarly for φ_2 and φ_3.

iteration	Z'	vset	uset
0	zip	$\varphi_1, \varphi_2, \varphi_3$	$\emptyset$
1	zip, AC	$\varphi_2, \varphi_3, \varphi_9$	$\varphi_6, \varphi_7, \varphi_8$
2	zip, AC, str	φ_3, φ_9	$\varphi_6, \varphi_7, \varphi_8$
3	zip, AC, str, city	φ_9	$\varphi_6, \varphi_7, \varphi_8$
4	zip, AC, str, city	$\emptyset$	$\varphi_6, \varphi_7, \varphi_8$

In iteration 1, TransFix picks and removes φ_1 from vset. It finds that φ_1 and master tuple s_1 (in Fig. 1) can be applied to t_1. Hence it normalizes $t_1[\text{AC}] := s_1[\text{AC}] = 131$, and expands Z' by including AC. It adds φ_9 to vset since $X \cup X_p$ of φ_9, *i.e.,* {AC}, is validated. Moreover, φ_6–φ_8 are added to uset, since while AC is validated, attributes phn and type are not yet. In iteration 2 (resp. 3), φ_2 (resp. φ_3) is selected from vset, and str (resp. city) is fixed by matching s_1. Here t_1 is updated by $t_1[\text{str}] := s_1[\text{str}] = 51$ Elm Row.

In iteration 4, φ_9 is selected and removed from vset. No change is incurred to t since city is already validated. TransFix terminates since vset is now empty. □

Correctness. Observe the following. (1) Each eR is used at most once. When a node is removed from vset, it will not be put back. Since the size of vset is at most the number $\text{card}(\Sigma)$ of eRs in Σ, the while loop (lines 5–15) iterates at most $\text{card}(\Sigma)$ times. (2) When applying (t_m, φ) to t, $t[X \cup X_p]$ have already been validated; thus $t[B]$ is ensured correct. (3) All the eRs that are possibly usable are examined. Hence, when TransFix terminates, no more attributes of t could be fixed given Z.

Complexity. Let $\mathbf{G}(V, E)$ be the dependency graph of Σ. Note that $|V| = \text{card}(\Sigma)$. The initialization of TransFix runs in $O(|\Sigma|)$ time (lines 1–4), by employing a hash table. As argued above, at most $|V|$ iterations of the outer loop (lines 6–15) are executed, since each iteration consumes at least one eR in Σ. The inner loop (lines 10–15) is run at most $|V|$ times for each outer iteration (*i.e.,* checking all eRs in Σ). In addition, observe the following: (a) checking containment and intersection of two attribute sets $(X_u \cup X_{p_u})$ and Z' is in $O(|X_u \cup X_{p_u}|)$ time if we use a hash table; and (b) it takes constant time to check whether there exists a master tuple that is applicable to t with an eR, by using a hash table that stores $t_m[X_m]$ as a key for $t_m \in D_m$. Putting these together, each outer iteration is in $O(|\Sigma|)$ time, and hence, TransFix is in $O(|V||\Sigma|)$ time, which is at most $O(|\Sigma|^2)$. In practice, $|\Sigma|$ is typically small.

5.2 Suggest: Generating new suggestions

To present procedure Suggest, we first define *suggestions* and state the problem of finding suggestions.

Suggestions. Consider a tuple t, where $t[Z]$ has been validated. A *suggestion* for t *w.r.t.* $t[Z]$ is a set S of attributes such that there exists a certain region $(Z \cup S, \{t_c\})$, where t_c is a pattern and $t[Z]$ satisfies $t_c[Z]$.

That is, if the users additionally assert that $t[S]$ is correct and $t[Z \cup S]$ matches some certain region, then a certain fix is warranted for t.

Example 13 Recall from Example 12 that $t_1[Z]$ is fixed by using Σ_0 and D_m, where $Z = \{\mathsf{zip}, \mathsf{AC}, \mathsf{str}, \mathsf{city}\}$. Let $S = \{\mathsf{phn}, \mathsf{type}, \mathsf{item}\}$. One can verify that S is a suggestion for t_1 *w.r.t.* $t_1[Z]$. Indeed, $(Z \cup S, \{t_c\})$ is a certain region for (Σ_0, D_m), where $t_c =$
($\underbrace{\text{EH7 4AH, 131, 51 Elm Row, Edi}}_{Z}$, $\underbrace{079172485, 2, \_}_{S}$). □

The users would naturally want a suggestion as "small" as possible, so that they need to make minimal efforts to ensure some attributes of t to be correct. This motivates us to study the following problem.

The *S-minimum problem* is to decide, given (Σ, D_m), a set $t[Z]$ of attributes that has been validated, and a positive integer K, whether there exists a non-empty set S of attributes such that (a) $Z \cap S = \emptyset$ (b) $|S| \leq K$ and (c) S is a suggestion for t *w.r.t.* $t[Z]$.

Observe that the Z-minimum problem (Sect. 4) is a special case of the S-minimum problem when no attribute is fixed initially (*i.e.,* $Z = \emptyset$). From this and Theorems 12 and 17 it follows that the S-minimum problem is NP-complete and approximation-hard.

These complexity bounds suggest that we develop heuristic algorithms to compute suggestions, along the same lines as computing certain regions, as discussed in [20]. When computing Z-minimum certain regions, all eRs need to be considered [20]. When it comes to suggestions, in contrast, attributes $t[Z]$ are already validated, which can be used to reduce the search space of eRs by refining some eRs and leaving the others out.

To do this we use the following notations. For an eR $\varphi = ((X, X_m) \to (B, B_m), t_p[X_p])$ and a list X_i of attributes in X, we use $\lambda_\varphi(X_i)$ to denote the corresponding attributes in X_m. For instance, when $(X_i, X_{m_i}) = (ABC, A_m B_m C_m)$, $\lambda_\varphi(AC) = A_m C_m$. We also write $\varphi^+ = ((X, X_m) \to (B, B_m), t_p^+[X_p^+])$, where $X_p \subseteq X_p^+$, *i.e.,* φ^+ differs from φ only in the pattern.

Consider a set Σ of eRs, a master relation D_m, an input tuple t, and attributes Z such that $t[Z]$ is fixed using TransFix. For an eR φ in Σ (1) if there exists no tuple $t_m \in D_m$ such that (φ, t_m) applies to t, then φ cannot be used to fix t; otherwise (2) we may extend the pattern of φ and refine its values with $t[Z]$, which yields φ^+. Hence we introduce the following notion.

The set of *applicable rules* for $t[Z]$ *w.r.t.* Σ, denoted as $\Sigma_{t[Z]}$, consists of eRs φ^+ defined as follows. For each φ in Σ, φ^+ is derived from φ if (a) $B \notin Z$; (b) $t_p[X_p \cap Z] \approx t[X_p \cap Z]$; and (c) there exists a master tuple $t_m \in D_m$, where $t_m[\lambda_\varphi(X_p \cap X)] \approx t_p[X_p \cap X]$ and $t_m[\lambda_\varphi(X \cap Z)] = t[X \cap Z]$. Here in φ^+ (i) $X_p^+ = X_p \cup (X \cap Z)$ and (ii) $t_p^+[X_p^+ \cap Z] = t[X_p^+ \cap Z]$.

Intuitively, φ^+ can be derived from φ if φ does not change the validated attributes (*i.e.,* (a) above), matches them (*i.e.,* (b)), and moreover, if there exists some master tuple that can be applied to t with φ (*i.e.,* (c)). The refined rule φ^+ extends the pattern attributes of φ with Z (*i.e.,* (i) above), and enriches its pattern values using the specific values of $t[Z]$ (*i.e.,* (ii)).

Example 14 For $t_1[\mathsf{zip}, \mathsf{AC}, \mathsf{str}, \mathsf{city}]$ validated in Example 12, applicable rules in $\Sigma_{t_1[\mathsf{zip},\mathsf{AC},\mathsf{str},\mathsf{city}]}$ include:

φ_4: ((phn, Mphn) → (FN, FN), t_{p4}[type] = (2));
φ_5: ((phn, Mphn) → (LN, LN), t_{p5}[type] = (2));
φ_6^+:(([AC, phn], [AC, Hphn]) → (str, str), t_{p6}[type, AC]=(1,131));
φ_7^+:(([AC, phn], [AC, Hphn])→(city, city), t_{p7}[type, AC]=(1,131));
φ_8^+:(([AC, phn], [AC, Hphn]) → (zip, zip), t_{p8}[type, AC]=(1,131));

Here φ_4 and φ_5 are taken from Σ_0, while φ_6^+ is derived from φ_6 by refining t_{p6}[AC] (from $\overline{0800}$ to 131), when t_1[AC] is known to be 131; similarly for φ_7^+ and φ_8^+. □

We show below that it suffices to consider $\Sigma_{t[Z]}$.

Proposition 20 *When $t[Z]$ is assured correct, S is a suggestion for t iff there exists a pattern tuple t_c such that $(Z \cup S, \{t_c\})$ is a certain region for $(\Sigma_{t[Z]}, D_m)$.*

Proof Assume that there exists t_c such that $(Z \cup S, \{t_c\})$ is a certain region for $(\Sigma_{t[Z]}, D_m)$. We show that S is a suggestion by constructing a pattern tuple t_c' such that $(Z \cup S, \{t_c'\})$ is a certain region for (Σ, D_m). Consider t_c', where $t_c'[Z] = t[Z]$ and $t_c'[S] = t_c[S]$. One can easily verify the following. (1) $(Z \cup S, \{t_c'\})$ is a certain region for $(\Sigma_{t[Z]}, D_m)$; (2) the set of attributes covered by $(Z \cup S, \{t_c'\}, \Sigma, D_m)$ is the same as the set covered by $(Z \cup S, \{t_c'\}, \Sigma_{t[Z]}, D_m)$; and (3) $(\Sigma_{t[Z]}, D_m)$ is consistent *w.r.t.* $(Z \cup S, \{t_c'\})$ iff (Σ, D_m) is consistent *w.r.t.* $(Z \cup S, \{t_c'\})$. From these it follows that $(Z \cup S, \{t_c'\})$ is also a certain region for (Σ, D_m).

Conversely, assume that S is a suggestion. Then there exists a certain region $(Z \cup S, \{t_c\})$ for (Σ, D_m). We define a pattern tuple t_c', where $t_c'[Z] = t[Z]$ and $t_c'[S] = t_c[S]$. One can show that $(Z \cup S, \{t_c'\})$ is a certain region for $(\Sigma_{t[Z]}, D_m)$. Indeed, this can be verified along the same lines as the argument given above. □

Procedure Suggest. Leveraging Proposition 20, we outline procedure Suggest in Fig. 6. It takes Σ, D_m, Z and t as input, and finds a suggestion as follows. It first derives applicable rules $\Sigma_{t[Z]}$ from Σ and $t[Z]$ (line 1). It then computes a certain region for $(\Sigma_{t[Z]}, D_m)$ (line 2), by employing the

Input: Tuple t, attributes Z, eRs Σ, and master data D_m.
Output: A set sug of attributes as suggestion.
1. derive $\Sigma_{t[Z]}$ using t, Z and Σ;
2. compute a certain region (Z', T_c) using $\Sigma_{t[Z]}$ and D_m;
3. return sug $:= Z' \setminus Z$;

Fig. 6 Procedure Suggest

algorithm provided in [20]. Finally, it constructs and returns a new suggestion (line 3).

Correctness and complexity. The correctness of Suggest follows from the definition of suggestions and Proposition 20. For its complexity, observe the following. (1) The set $\Sigma_{t[Z]}$ can be derived from Σ and $t[Z]$ in $O(|\Sigma| + |t|)$ time, by employing the indices developed for Procedure TransFix. Indeed, the conditions for applicable rules can be checked in constant time. (2) The algorithm of [20] computes a certain region in $O(|\Sigma_{t[Z]}|^2 |D_m| \log(|D_m|))$ time, where $|\Sigma_{t[Z]}| \leq |\Sigma|$. Hence Suggest is in $O(|\Sigma|^2 |D_m| \log(|D_m|))$ time.

Optimization. It is quite costly to compute a certain region in each round of user interactions. This motivates us to develop an optimization strategy, which aims to minimize unnecessary recomputation by reusing certain regions computed earlier. In a nutshell, when processing a stream of input tuples of schema R, we maintain certain regions generated for them. When a new input tuple t arrives, we check whether some region computed previously remains a certain region for fixing t. If so, we simply reuse the region, without computing a new one starting from scratch. We compute new suggestions only when necessary. As will be verified by our experimental study, this reduces the cost significantly, since it is far less costly to check whether a region is certain than computing new certain regions [20].

We maintain previously computed certain regions by using a *binary decision diagram* (BDD) [29]. A BDD is a directed acyclic graph $\mathbf{G}_b = (V_b, E_b)$. Each node u in V_b represents either a condition or a call for Suggest, and it has at most two outgoing edges. The root of $\mathbf{G}_b$ is denoted as *start*. Each edge (u, v) is labeled with a pair (bval, act), where bval is either Boolean value true or false; and act is an action, which provides a suggestion if bval is true, and generates new suggestions otherwise.

Example 15 Consider the evolution of a BDD depicted in Fig. 7. When no tuples have been processed, the BDD is shown in Fig. 7a. Here the set Z of attributes taken from the precomputed certain region is treated as the first suggestion, as described in procedure TransFix. For the first input tuple t_1, if $t_1[Z]$ does not match any certain region, a new suggestion needs to be computed; hence the call for procedure Suggest.

Assume that t_1 is fixed with two suggestions S_1 and S_2. Then BDD is expanded, as shown in Fig. 7b. Consider a newly arrived tuple t_2. If $t_2[Z]$ does not satisfy any certain region, TransFix expands the set Z' of validated attributes. We check whether S_1 is a suggestion *w.r.t.* $t_2[Z']$. If so, the true branch is followed and S_1 is recommended to the users; otherwise Suggest is invoked to generate a new suggestion. Similarly, S_2 is checked. If t_2 still cannot be fixed with S_2, Suggest is invoked for both the true and the false branches to produce a new suggestion. The new suggestion is added to the BDD. After more tuples are fixed, the BDD may evolve to Fig. 7c, which collects those certain regions generated when processing these tuples. As shown above, these regions are reused when processing new tuples. □

Capitalizing on BDD, we present an optimized Suggest, denoted as Suggest$^+$, which is outlined in Fig. 8. It takes t, Z, Σ, D_m, a BDD G_b and a node u on G_b as input, and finds a suggestion as follows.

Suggest$^+$ traverses G_b top-down starting from its root, *i.e.,* the input u is initialized at *start* node. At each round of interaction, a node u of G_b is visited, at which it checks whether a precomputed suggestion associated with u remains a suggestion for t. If not, it checks other previously computed regions via a false branch (lines 1–2). Otherwise, it recommends the same suggestion to the users, and moves

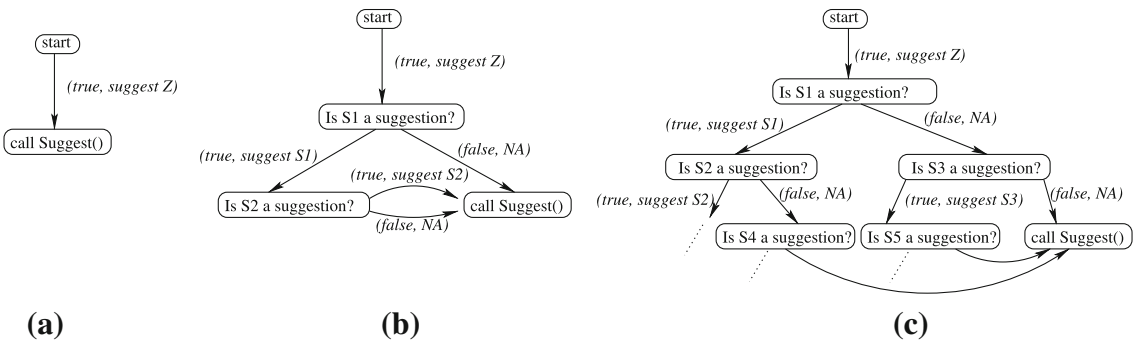

Fig. 7 A sample BDD, **a** initial state, **b** after a tuple t_1 is fixed, **c** after several tuples are fixed

Input: Tuple t, attributes Z, eRs Σ, master data D_m, a BDD $G_b(V_b, E_b)$, and a node u in V_b.
Output: A set sug of attributes as suggestion.
1. **while** u is not a suggestion **and** u has a *false* branch **do**
2. $u := v$ where the edge (u, v) is a *false* branch;
3. **if** u is a suggestion **then**
4. get sug from u; $u := v$ where (u, v) is a *true* branch;
5. **else** sug := Suggest(t, Z, Σ, D_m); maintain G_b with sug;
6. return sug;

Fig. 8 Procedure Suggest$^+$

to the child of u via a true branch (lines 3–4). In the next round of interaction, if needed, checking resumes at node u. Suggest is invoked to compute new suggestions when no known regions can be reused, and G_b is also maintained (line 5). Finally, a suggestion is returned (line 6).

It implements a strategy to decide what suggestions are maintained by a BDD (line 5), to strike a balance between checking a set of suggestions and recomputing a certain region. It also compresses BDD to reduce the space cost. We omit the details for space limit.

We revise CertainFix by using Suggest$^+$ instead of Suggest, and refer to it as CertainFix$^+$.

6 Experimental study

We next present an experimental study, using real-life data. Two sets of experiments were conducted, to verify (1) the effectiveness of our method in terms of the quality of suggestions generated, measured by the number of attributes that are correctly fixed in a round of user interactions; and (2) the efficiency and scalability of our algorithm for finding fixes and suggestions.

For the effectiveness study, we compared with the following: (a) GRegion that greedily finds a certain region. It chooses attributes according to one rule: at each stage, choose an attribute which may fix the largest number of uncovered attributes; and (b) IncRep, the algorithm in [14] for data repairing; given a dirty database D and a set of constraints, it is a heuristic method to make D consistent, *i.e.,* finds a repair D' that satisfies the constraints and "minimally" differs from D. It adopts a metric to minimize (1) the distance between the original values and the new values of changed attributes and (2) the weights of the attributes modified.

Experimental data. Real-life datasets were employed to examine the applicability of our method in practice.

(1) HOSP *(Hospital Compare)* is publicly available from U.S. Department of Health & Human Services[1]. We used three tables: HOSP, HOSP_MSR_XWLK, and STATE_MSR_AVG, which record the hospital information, the score of measurement of each hospital and the average score of each hospital measurement, respectively. We created a big table by joining the three tables with *natural join*, among which we chose 19 attributes for the schema of both the master relation R_m and the relation R: zip, ST (state), phn, mCode (measure code), measure name, sAvg (StateAvg), hName (hospital name), hospital type, hospital owner, provider number, city, emergency service, condition, Score, sample, id, address1, address2, address3.

We designed 21 eRs for the HOSP data, with five representative ones as follows:

φ_1 : ((zip, zip) → (ST, ST), t_{p1}[zip] = ($\overline{\text{nil}}$));
φ_2 : ((phn, phn) → (zip, zip), t_{p2}[phn] = ($\overline{\text{nil}}$));
φ_3 : (((mCode, ST), (mCode, ST)) → (sAvg,sAvg), t_{p3} = ());
φ_4 : (((id, mCode), (id, mCode)) → (Score,Score), t_{p4} = ());
φ_5 : ((id, id) → (hName,hName), t_{p5} = ()).

(2) DBLP is from the DBLP Bibliography[2]. We first transformed the XML data into relations. We then created a big table by joining the *inproceedings* data (conference papers) with the *proceedings* data (conferences) on the crossref attribute (a foreign key). Besides, we also included the homepage info (hp) for authors, which was joined by the homepage entries in the DBLP data.

From the big table, we chose 12 attributes to specify the schema of both the master relation R_m and the data relation R, including ptitle (paper title), a1 (the first author), a2 (the second author), hp1 (the homepage of a1), hp2 (the homepage of a2), btitle (book title), publisher, isbn, crossref, year, type, and pages.

We designed 16 eRs for the DBLP data, shown below.

ϕ_1 : ((a1, a1) → (hp1, hp1), t_{p1}[a1] = ($\overline{\text{nil}}$));
ϕ_2 : ((a2, a1) → (hp2, hp1), t_{p2}[a2] = ($\overline{\text{nil}}$));
ϕ_3 : ((a2, a2) → (hp2, hp2), t_{p3}[a2] = ($\overline{\text{nil}}$));
ϕ_4 : ((a1, a2) → (hp1,hp2), t_{p4}[a1] = ($\overline{\text{nil}}$));
ϕ_5 : (((type, btitle, year), (type, btitle, year)) → (A, A), t_{p5}[type] = ('inproceeding'));
ϕ_6 : (((type, crossref), (type, crossref) → (B, B), t_{p6}[type] = ('inproceeding'));
ϕ_7 : (((type, a1, a2, title, pages), (type, a1, a2, title, pages)) → (C, C), t_{p7}[type] = ('inproceeding')).

where the attributes A, B and C range over the sets {isbn, publisher, crossref}, {btitle, year, isbn, publisher} and {isbn, publisher, year, btitle, crossref}, respectively.

Observe that in eRs ϕ_2 and ϕ_4, the attributes are mapped to different attributes. That is, even when the master relation R_m and the relation R share the same schema, some eRs still could not be syntactically expressed as CFDs, not to mention their semantics.

A dirty data generator was developed. Given a clean dataset (HOSP or DBLP), it generated dirty data controlled by three parameters: (a) duplicate rate $d\%$, which is the probability that an input tuple matches a tuple in master data D_m, indicating the relevance and completeness of D_m; (b) noise rate

[1] http://www.hospitalcompare.hhs.gov/.

[2] http://www.informatik.uni-trier.de/~ley/db/.

$n\%$, which is the percentage of erroneous attributes in input tuples; and (c) the cardinality $|D_m|$ of master dataset D_m.

User interactions. User feedback was simulated by providing the correct values of the given suggestions.

Implementation. All algorithms were implemented in C++. The experiments were run on a machine with an Intel(R) Core(TM)2 Duo P8700 (2.53 GHz) CPU and 4 GB of memory. Each experiment was repeated 5 times and the average is reported here.

Experimental results. We next present our findings.

Exp-1: Effectiveness. The tests were conducted by varying $d\%$, $|D_m|$ and $n\%$, The default values for $d\%$, $|D_m|$ and $n\%$ were 30%, $10K$ and 20%, respectively. When all these parameters were fixed, we generated $10K$ tuples for this set of experiments, but allowed the dataset to scale to $10M$ tuples in the scalability study.

This set of experiments includes (1) the effectiveness of certain regions generated by our algorithm compared with GRegion; (2) the initial suggestion selection; (3) the effectiveness of suggestions in terms of the number of interaction rounds needed; (4) the impact of duplicate rate $d\%$; (5) the impact of master data size $|D_m|$; (6) the impact of noise rate $n\%$; and (7) the effectiveness of our method compared with IncRep.

The studies were quantified at both the tuple level and the attribute level. Since we assure that each fixed tuple is correct, we have a 100% precision. Hence the first measure we used is *recall*, defined as follows:

$$\text{recall}_t = \#\text{-corrected tuples} \,/\, \#\text{-erroneous tuples}$$
$$\text{recall}_a = \#\text{-corrected attributes} \,/\, \#\text{-erroneous attributes}$$

The number of corrected attributes does *not* include those fixed by the users.

To compare with IncRep, we also used F-measure [3] to combine *recall* and *precision*, since the precision of repairs produced by IncRep is not 100%. Precision and F-measure are given as follows:

$$\text{precision}_a = \#\text{-corrected attributes} \,/\, \#\text{-changed attributes}$$
$$\text{F-measure} = 2 \cdot (\text{recall}_a \cdot \text{precision}_a)/(\text{recall}_a + \text{precision}_a)$$

(1) *The effectiveness of certain regions.* The table below shows the number of attributes in the certain region found by our method CompCRegion [20] and its counterpart found by GRegion. It shows that the certain region computed by CompCRegion has far less attributes than its counterpart by GRegion, which thus minimizes user efforts, as expected. Indeed, CompCRegion found the best certain region (*i.e.,* with the least number of attributes) for both datasets as a suggestion.

[3] http://www.en.wikipedia.org/wiki/F-measure.

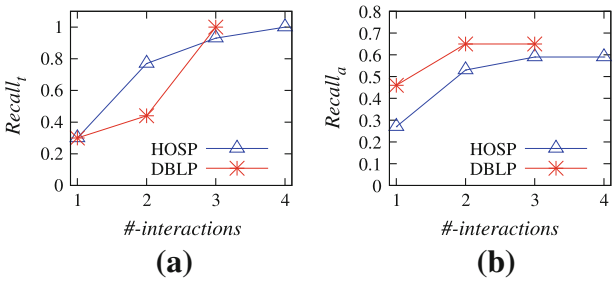

Fig. 9 Recall values *w.r.t.* the number of interactions **a** tuple-level recalls, **b** attribute-level recalls

Dataset	CompCRegion	GRegion
HOSP	2	4
DBLP	5	9

(2) *The initial suggestion selection.* We evaluated the impact of initial suggestions by using the certain region with the highest quality (denoted by CRHQ) vs. the one with the median quality (CRMQ). As shown in the table below, when CRHQ is used as the initial suggestion, CertainFix yields higher F-measure values than its CRMQ counterpart. That is, CRHQ allows CertainFix to automatically fix more attributes than CRMQ.

Dataset	F-measure	
	CRHQ	CRMQ
HOSP	0.74	0.70
DBLP	0.79	0.69

(3) *The effectiveness of suggestions.* Fixing the three parameters, we studied recall *w.r.t.* user interactions.

Figure 9a shows the tuple-level recalls. The x-axis indicates the number of interactions and the y-axis represents recall values. It tells us that few rounds of interactions are required to fix the entire set of attributes of an input tuple, *e.g.,* at most 4 (resp. 3) rounds for HOSP (resp. DBLP). Most tuples could be correctly fixed within few interactions, *e.g.,* 93% (resp. 100%) of tuples are fixed in the third round for HOSP (resp. DBLP).

Figure 9b reports the attribute-level recalls, to complement Fig. 9a. Among the errors fixed, some were automatically corrected by our algorithm, while the others by user feedback during the interactions. As remarked earlier, the errors fixed by the users were *not* counted in our recall values. Hence recall_a is typically below 100%. As shown in Fig. 9b, our method could fix at least 50% of the errors within 2 rounds of interactions, although the errors were distributed across all attributes, and moreover, only a portion of the errors were fixable by the given Σ and D_m given that the duplicate rate $d\%$ is only 30%. One can see that the recall value at the 4th (resp. 3rd) round of interaction for HOSP (resp. DBLP) is unchanged, indicating that the users corrected the attributes that are irrelevant to Σ and D_m. As will be seen later, when $d\%$ is increased, the attribute-level recall gets higher.

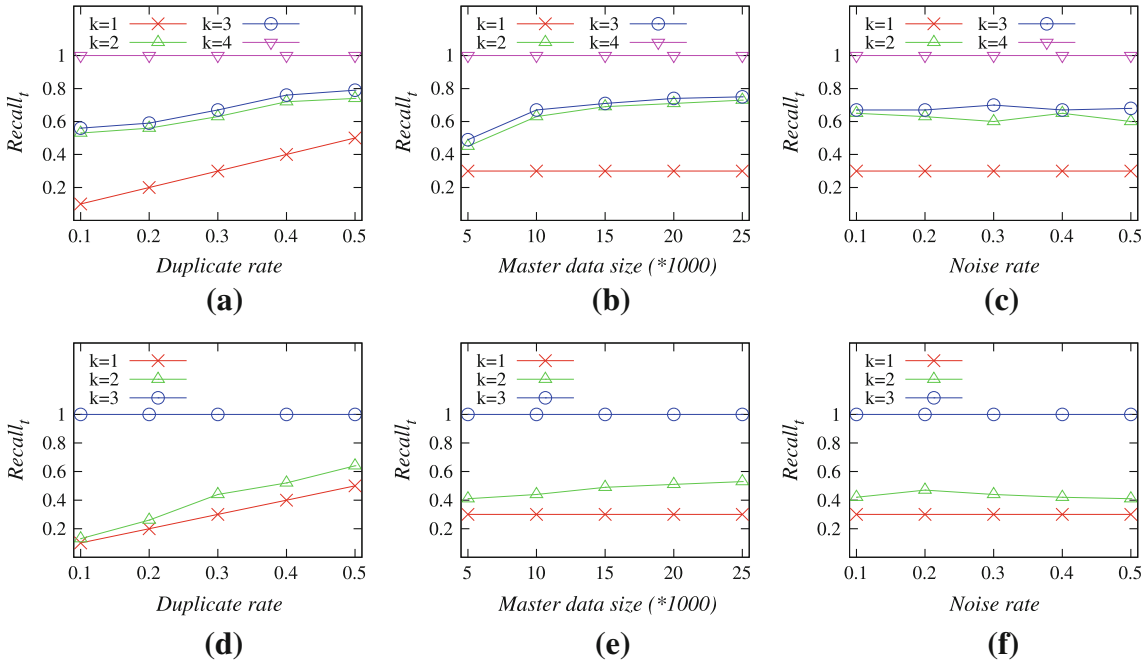

Fig. 10 Tuple-level fixes when varying one of $d\%$, $|D_m|$ and $n\%$. **a** varying $d\%$ for HOSP, **b** varying $|D_m|$ for HOSP, **c** varying $n\%$ for HOSP, **d** varying $d\%$ for DBLP, **e** varying $|D_m|$ for DBLP, **f** varying $n\%$ for DBLP

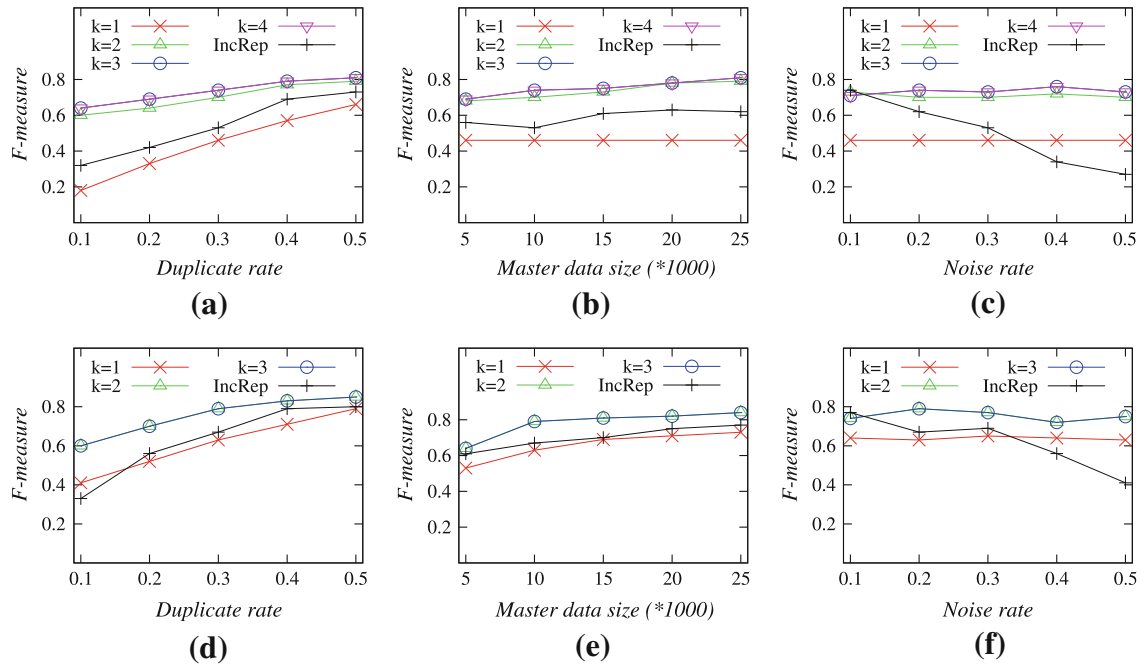

Fig. 11 Attribute-level fixes when varying one of $d\%$, $|D_m|$ and $n\%$. **a** varying $d\%$ for HOSP, **b** varying $|D_m|$ for HOSP, **c**HOSP attribute-level *w.r.t.* $n\%$, **d** varying $d\%$ for DBLP, **e** varying $|D_m|$ for DBLP, **f**DBLP attribute-level *w.r.t.* $n\%$

These experimental results verify that our method is able to provide effective suggestions, such that all errors could be fixed within few rounds of user interactions, by using eRs and master data, even when the master data is not very relevant (when $d\% = 30\%$).

(4) *Impact of* $d\%$. Fixing $|D_m| = 10K$ and $n\% = 20\%$, we varied duplicate rate $d\%$ from 10 to 50%. Figures 10a and d (resp. Figs. 11a, d) report the tuple-level recalls (resp. F-measure) after k rounds of interactions for HOSP and DBLP, respectively.

Figures 10a and d show that the larger $d\%$ is, the higher the recall is, as expected, since a larger $d\%$ means a higher probability that an input tuple matches some master tuple such that its errors can be fixed. A closer examination reveals that

early interactions are more sensitive to $d\%$, *e.g.,* when $k = 1$, the percentage of fixed tuples increases from 0.1 to 0.5, when $d\%$ varies from 10 to 50%. In later interactions, *e.g.,* the last round when $k = 4$, the users have to ensure the correctness of those attributes that cannot be fixed by eRs and D_m. Hence recall_t remains unchanged there.

Figures 11a and d further verify this observation: most attributes are fixed by our method in early interactions, while those fixed in later rounds are by the users' feedback. Moreover, the gap between the first two rounds of interactions (when $k = 1$ and $k = 2$) shows that the suggestions generated are effective.

The results tell us that our method is sensitive to duplicate rate $d\%$: the higher $d\%$ is, the more errors could be automatically fixed, in early interactions.

(5) *Impact of* $|D_m|$. Fixing $d\% = 30\%$ and $n\% = 20\%$, we varied $|D_m|$ from $5K$ to $25K$. The tuple-level recalls (resp. F-measure values) are reported in Figs. 10b and e (resp. Figs. 11b, e) after k rounds of interactions for HOSP and DBLP, respectively.

Figures 10b and e show that in the first round of interactions, *i.e.,* $k = 1$, recall_t is insensitive to $|D_m|$. Indeed, whether a certain fix exists or not in the first interaction is determined by the duplicate rate $d\%$, rather than $|D_m|$. As shown in both figures, the recall_t is 0.3 when $k = 1$, exactly the same as $d\%$. However, when interacting with the users, the recall values increase for larger D_m. This verifies that TransFix is effective, which identifies eRs and master data to fix errors.

Figures 11b and e show that more attributes can be fixed by increasing $|D_m|$, *i.e.,* F-measure gets higher, even when the recall_t is unchanged (*e.g.,* $k = 1$), *i.e.,* when not the entire tuple could be fixed. These results also confirm the observations above about the sensitivity of later rounds of interactions to $|D_m|$.

These results tell us that the amount of master data is important to generating effective suggestions. The more the master data, the higher possibility that eRs could find master tuples to fix attributes, as expected.

(6) *Impact of* $n\%$. Fixing $d\% = 30\%$ and $|D_m| = 10K$, we varied the noise rate $n\%$ from 0.1 to 0.5. Figures 10c and f (resp. Figs. 11c, f) show the tuple-level recalls (resp. F-measure) after k rounds of interactions for HOSP and DBLP, respectively.

The results show that our method is sensitive to $n\%$ at *neither* the tuple level *nor* the attribute level. At the tuple level (Figs. 10c, f), recall_t is the ratio of the number of corrected tuples to the number of erroneous tuples. For a set of attributes asserted by the users, the attributes fixed by our algorithm remain the same for all input tuples, irrelevant to what attributes are originally erroneous. At the attribute-level (Figs. 11c, f), since the precision of our algorithm is 100%, F-measure is determined by the recall values. As recall_t is insensitive to $n\%$, so is F-measure.

(7) *Comparison with* IncRep. To favor IncRep, we fixed $k = 1$, since IncRep does not interact with the users. Since IncRep measures recall at the attribute level only [14], we focus on F-measure. Figures 11a and d (resp. Figs. 11b, e) show the F-measure values when varying $d\%$ (resp. $|D_m|$) while fixing the other two parameters. The results tell us that IncRep has slightly higher F-measure values than our method. This is because IncRep attempts to repair the entire tuple, while our method only corrects those attributes when the fixes are certain in the first round of interaction, and defers the repairing of the other attributes to later rounds upon the availability of user feedback.

Figures 11c and f show that when the noise rate $n\%$ is increased, the F-measure values of IncRep get substantially lower, and are worse than ours. This is because IncRep introduces more errors when the noise rate is higher. Our method, in contrast, ensures that each fix is correct, and hence is *insensitive* to $n\%$.

Exp-2: Efficiency and scalability. This set of experiments evaluated the efficiency of our method by varying the size of D_m (resp. a set D of input tuples) in Figs. 12a and b for HOSP (resp. Figs. 12b, d for DBLP). We report the average elapsed time for each round of interaction, *i.e.,* the time spent on fixing tuples in D and for generating a suggestion. Here CertainFix and CertainFix$^+$ denote the algorithm that does not use BDD and employs BDD, respectively.

Figures 12a and b show that our method takes no more than a second to fix attributes of a tuple and to come up with a suggestion. Further, the optimization strategy by using BDD is effective: it substantially reduces the response time. Moreover, both CertainFix and CertainFix$^+$ scale well with master data.

As shown in Figures 12c and d, CertainFix is insensitive to $|D|$, since each input tuple is processed independently. For CertainFix$^+$, when $|D|$ is very small (*e.g.,* 10), BDD does not help us find suggestions, and the elapsed time of CertainFix$^+$ is similar to the time of CertainFix; when $|D|$ increases from 10 to 100, the response time is significantly reduced since more suggestions could be found with BDD; when $|D| > 100$, BDD can provide effective suggestions such that the average elapsed time remains unchanged, around 0.1 s.

Summary. The experimental results show the followings. (1) The initial suggestions computed by our method are more effective than those found by greedy approaches. (2) Our method is effective: it mostly takes less than four rounds of user interactions to find a certain fix for an input tuple. (3) The number of interactions highly depends on the relevance of an input tuple to the master data, *i.e.,* $d\%$, and $|D_m|$ to a lesser extent. (4) Our method is insensitive to the error

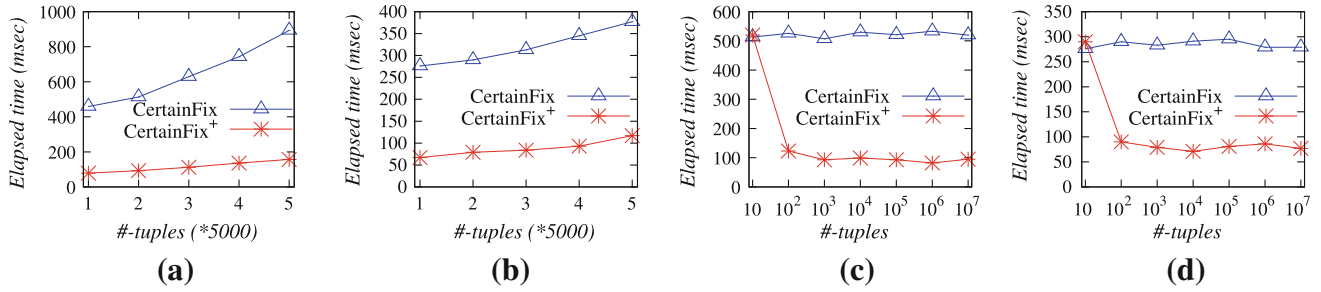

Fig. 12 Efficiency and scalability. **a** varying $|D_m|$ for HOSP, **b** Varying $|D_m|$ for DBLP, **c** varying $|D|$ for HOSP, **d** varying $|D|$ for DBLP

rate $n\%$. It outperforms the repairing method of [14] when the error rate is high, even with two or three rounds of interactions. (5) Our algorithm scales well with the size of D_m. (6) The optimization strategy with BDD is effective in finding suggestions with low latency.

It should be remarked that data monitoring incurs extra overhead of fixing input tuples for the database engine. Nevertheless, as pointed out by [37], it is far less costly to correct a tuple at the point of data entry than fixing it afterward. The need for this is particularly evident when it comes to critical data. In addition, as verified by our experimental results, the extra cost is rather small since effective suggestions (Exp-1 (1–3)) and certain fixes (Exp-2) can be generated efficiently, below 0.2 s in average with CertainFix$^+$ (Fig. 12).

7 Conclusion

We have proposed editing rules that, in contrast to constraints used in data cleaning, are able to find certain fixes for input tuples by leveraging master data. We have identified fundamental problems for deciding certain fixes and certain regions, and established their complexity bounds. We have also developed a framework to compute certain fixes at the point of data entry, by interacting with users, along with its underlying algorithm and optimization techniques. Our experimental results with real-life data have verified the effectiveness, efficiency and scalability of our method. These yield a promising method for data monitoring.

This work is just a first step toward repairing data with correctness guarantees. One topic for future work is to efficiently find certain fixes for data in a database, *i.e.,* certain fixes in data repairing rather than monitoring. Another topic is to develop data repairing and monitoring methods with correctness guarantees in the absence of high-quality master data. Finally, effective algorithms have to be in place for discovering editing rules from sample inputs and master data, along the same lines as discovering other data quality rules [12,26].

Acknowledgments Fan is supported in part by the RSE-NSFC Joint Project Scheme and an IBM scalable data analytics for a smarter planet innovation award. Fan and Li are also supported in part by the National Basic Research Program of China (973 Program) 2012CB316200. Shuai is supported in part by NGFR 973 2011CB302602 and NSFC grants 90818028 and 60903149.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Boston (1995)
2. Arenas, M., Bertossi, L.E., Chomicki, J.: Consistent query answers in inconsistent databases. TPLP **3**(4–5), 393–424 (2003)
3. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press, Cambridge (2009)
4. Batini, C., Scannapieco, M.: Data Quality: Concepts, Methodologies and Techniques. Springer, Berlin (2006)
5. Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., WhangS.E. Widom, J.: Swoosh: a generic approach to entity resolution. VLDB J. **18**(1), 255–276 (2009)
6. Bohannon, P., Fan, W., Flaster, M., Rastogi, R.: A cost-based model and effective heuristic for repairing constraints by value modification. In: Proceedings of the ACM SIGMOD (2005)
7. Bravo, L., Fan, W., Geerts, F., Ma, S.: Increasing the expressivity of conditional functional dependencies without extra complexity. In: Proceedings of the International Conference on Data Engineering (ICDE) (2008)
8. Bravo, L., Fan, W., Ma, S.: Extending dependencies with conditions. In: Proceedings of Very Large Data Bases (VLDB) (2007)
9. Chaudhuri, S., Ganjam, K., Ganti, V., Motwani, R.: Robust and efficient fuzzy match for online data cleaning. In: Proceedings of the ACM SIGMOD (2003)
10. Chen, K., Chen, H., Conway, N., Hellerstein, J.M., Parikh, T.S.: Usher: improving data quality with dynamic forms. In: Proceedings of the International Conference on Data Engineering (ICDE) (2010)
11. Chen, W., Fan, W., Ma, S.: Analyses and validation of conditional dependencies with built-in predicates. In: Proceedings of Database and Expert Systems Applications (2009)
12. Chiang, F., Miller, R.: Discovering data quality rules. PVLDB 1(1) (2008)
13. Chomicki, J., Marcinkowski, J.: Minimal-change integrity maintenance using tuple deletions. Inf. Comput. **197**(1–2), 90–121 (2005)
14. Cong, G., Fan, W., Geerts, F., Jia, X., Ma, S.: Improving data quality: Consistency and accuracy. In: Proceedings of Very Large Data Bases (VLDB) (2007)
15. Eckerson, W.W.: Data quality and the bottom line: achieving business success through a commitment to high quality data. The Data Warehousing Institute (2002)
16. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: a survey. TKDE **19**(1), 1–16 (2007)

17. Fan, W.: Dependencies revisited for improving data quality. In: PODS (2008)
18. Fan, W., Gao, H., Jia, X., Li, J., Ma, S.: Dynamic constraints for record matching. VLDB J. (To appear)
19. Fan, W., Geerts, F., Jia, X., Kementsietsidis, A.: Conditional functional dependencies for capturing data inconsistencies. TODS **33**(2) (2008)
20. Fan, W., Li, J., Ma, S., Tang, N., Yu, W.: Towards certain fixes with editing rules and master data. PVLDB **3**(1), (2010)
21. Fan, W., Li, J., Ma, S., Tang, N., Yu, W.: Interaction between record matching and data repairing. In: Proceedings of the ACM SIGMOD (2011)
22. Faruquie, T., et al.: Data cleansing as a transient service. In: Proceedings of the International Conference on Data Engineering (ICDE) (2010)
23. Fellegi, I., Holt, D.: A systematic approach to automatic edit and imputation. J. Am. Stat. Assoc. **71**(353), 17–35 (1976)
24. Gartner.: Forecast: data quality tools, Worldwide, 2006–2011. Technical report, Gartner (2007)
25. Giles, P.: A model for generalized edit and imputation of survey data. Can. J. Stat. **16**, 57–73 (1988)
26. Golab, L., Karloff, H.J., Korn, F., Srivastava, D., Yu, B.: On generating near-optimal tableaux for conditional functional dependencies. PVLDB **1**(1) (2008)
27. Guo, S., Dong, X., Srivastava, D., Zajac, R.: Record linkage with uniqueness constraints and erroneous values. PVLDB **3**(1), (2010)
28. Herzog, T.N., Scheuren, F.J., Winkler, W.E.: Data Quality and Record Linkage Techniques. Springer, Berlin (2009)
29. Knuth, D.E.: The Art of Computer Programming Volume 4, Fascicle 1: Bitwise tricks & techniques; Binary Decision Diagrams. Addison-Wesley Professional, Boston (2009)
30. Kolahi, S., Lakshmanan, L.: On approximating optimum repairs for functional dependency violations. In: Proceedings of International Conference on Database Theory (ICDT) (2009)
31. Loshin, D.: Master Data Management. Knowledge Integrity, Inc., California (2009)
32. Naumann, F., Bilke, A., Bleiholder, J., Weis, M.: Data fusion in three steps: resolving schema, tuple, and value inconsistencies. IEEE Data Eng. Bull. **29**(2), 21–31 (2006)
33. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley, Boston (1994)
34. Raman, V., Hellerstein, J.M.: Potter's Wheel: An interactive data cleaning system. In: Proceedings of Very Large Data Bases (VLDB) (2001)
35. Raz, R., Safra, S.: A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In: Proceedings of Symposium on the Theory of Computing (STOC) (1997)
36. Redman, T.: The impact of poor data quality on the typical enterprise. Commun. ACM **41**(2), 79–82 (1998)
37. Sauter, G., Mathews, B., Ostic, E.: Information service patterns, part 3: Data cleansing pattern. IBM (2007)
38. Song, S., Chen, L.: Discovering matching dependencies. In: Proceedings of the 10th International Conference on Information and Knowledge Management (CIKM) (2009)
39. Widom, J., Ceri, S.: Active database systems: triggers and rules for advanced database processing. Morgan Kaufmann, California (1996)
40. Wijsen, J.: Database repairing using updates. TODS **30**(3), 722–768 (2005)
41. Yakout, M., Elmagarmid, A.K., Neville, J., Ouzzani, M., Ilyas, I.F.: Guided data repair. PVLDB **4**(1), (2011)