The VLDB Journal (2018) 27:497–519
https://doi.org/10.1007/s00778-018-0506-9

REGULAR PAPER

Distilling relations using knowledge bases

Shuang Hao[1] · Nan Tang[2] · Guoliang Li[1] · Jian Li[1] · Jianhua Feng[1]

Received: 27 October 2017 / Revised: 13 February 2018 / Accepted: 3 May 2018 / Published online: 17 May 2018

Abstract

Given a relational table, we study the problem of detecting and repairing erroneous data, as well as marking correct data, using well curated knowledge bases (KBs). We propose detective rules (DRs), a new type of data cleaning rules that can make actionable decisions on relational data, by building connections between a relation and a KB. The main invention is that a DR simultaneously models two opposite semantics of an attribute belonging to a relation using types and relationships in a KB: The *positive semantics* explains how its value should be linked to other attribute values in a correct tuple, and the *negative semantics* indicate how a wrong attribute value is connected to other correct attribute values within the same tuple. Naturally, a DR can mark correct values in a tuple if it matches the *positive semantics*. Meanwhile, a DR can detect/repair an error if it matches the *negative semantics*. We study fundamental problems associated with DRs, e.g., rule consistency and rule implication. We present efficient algorithms to apply DRs to clean a relation, based on rule order selection and inverted indexes. Moreover, we discuss approaches on how to generate DRs from examples. Extensive experiments, using both real-world and synthetic datasets, verify the effectiveness and efficiency of applying DRs in practice.

Keywords Data cleaning · Knowledge base · Detective rule · Rule generation

1 Introduction

Nowadays, many enterprises are data driven. However, extracting actual business value from data is hard, where dirty data are at the core of bad decisions. The classical situation, garbage in and garbage out, is applicable to any data analytical task, including: database queries, data mining pipelines, and machine learning training processes. It is known that real-world data are dirty, up to 30% of an organization's data could be dirty; and it is very expensive, which costs the US economy $3 trillion+ in 2016 as reported by Harvard Business Review.

There have been many studies in cleaning data using integrity constraints (ICs) [7,9,10,49,51]. ICs are good at capturing errors. However, the serious drawback of ICs is that they cannot precisely tell which value is wrong. Take functional dependencies (FDs) for example. Consider an FD country → capital over the relation $R(\mathsf{country}, \mathsf{capital})$, and two tuples $t(\textit{China, Beijing})$ and $t'(\textit{China, Shanghai})$. The FD can identify the existence of errors in t and t', but cannot tell which value is wrong. All FD-based repairing algorithms use some heuristics to guess the wrong value, $t[\mathsf{country}]$, $t[\mathsf{capital}]$, $t'[\mathsf{country}]$, or $t'[\mathsf{capital}]$, and then repair it.

In contrast, rule-based data repairing explicitly tells how to repair an error. For instance, fixing rules [55] can specify that for each tuple, if its country is *China* and its capital is *Shanghai*, then *Shanghai* is wrong and should be changed to *Beijing*. Other rule-based approaches such as editing rules [23] and Sherlock rules [31] use tabular master data, to collect evidence from external reliable data sources.

Arguably, both IC- and rule-based methods can make mistakes when repairing data. However, IC-based tools are more like *black-boxes*, while rule-based methods are *white-*

✉ Guoliang Li
liguoliang@tsinghua.edu.cn

Shuang Hao
haos13@mails.tsinghua.edu.cn

Nan Tang
ntang@hbku.edu.qa

Jian Li
lijian83@tsinghua.edu.cn

Jianhua Feng
fengjh@tsinghua.edu.cn

[1] Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

[2] Qatar Computing Research Institute, HBKU, Doha, Qatar

boxes. When some mistakes made by the tools are identified, the latter is more interpretable about what happened. Not surprisedly, in industries, rule-based repairing methods are widely adopted, e.g., ETL rules, but IC-based tools are rarely employed.

Currently, we are witnessing an increased availability of well curated KBs such as Yago [30] and DBpedia [41]. Also, large companies maintain their own KBs, e.g., Warlmart [16], Google and Microsoft. In order to take advantage of these KBs, we extend prior rule-based cleaning methodologies [23,31] to clean relations by collecting evidence from KBs. The core of our proposal is a new type of data cleaning rules that build the connections between relations and KBs. They are rule-based, such that the actions of how to clean the data are baked in the rules, which rely on neither other heuristic solutions as those for ICs [7,9,10], nor the domain experts [11,43].

Motivating examples Perhaps the best way of understanding our proposal is by examples.

Example 1 Consider a database D of Nobel laureate records, specified by the following schema:

Nobel (Name, DOB, Country, Prize, Institution, City)

where a Nobel tuple specifies a Nobel laureate in Chemistry, identified by Name, together with its DOB, Country, Prize, Institution and City of the institute. Table 1 shows four tuples $t_1 - t_4$. All errors are highlighted, and their correct values are given between brackets. For instance, consider t_1 about *Avram Hershko*, a Hungarian-born Israeli biochemist. The value t_1[City] = *Karcag* is an error, which is the city he was born in, whose correct value should be *Haifa* where he works at.

Next we discuss, based on the available evidence from KBs, how to make judgment on the correctness of a relation.

Example 2 Consider an excerpt of a KB Yago [30], as depicted in Fig. 1. Here, a solid rectangle represents an *entity*, e.g., o_1 to o_7 , a dotted rectangle indicates a *class*, e.g., country and city, a shaded rectangle is a *literal*, e.g., o_8, a labeled and directed edge shows the *relationship* between entities or the *property* from an entity to a literal, and a dotted edge associates an entity with a class.

Consider tuple t_1 in Table 1 and the sample KB in Fig. 1, both about *Avram Hershko*. It is easy to see that most values of t_1 appear in Fig. 1. Based on different bindings of relationships, we can have the following three actions for t_1.

(i) Proof positive Based on the evidence in Fig. 1, we may judge that t_1[Name, DOB, Country, Institution] are correct.

(ii) Proof negative If we know that, t_1[City] should be the city that he works in, and we find from Fig. 1 the following evidence:

Table 1 Database D: Nobel laureates in Chemistry

	Name	DOB	Country	Prize	Institution	City
t_1	Avram Hershko	1937-12-31	Israel	**Albert Lasker Award for Medicine (Nobel Prize in Chemistry)**	Israel Institute of Technology	**Karcag (Haifa)**
t_2	Marie Curie	1867-11-07	France	Nobel Prize in Chemistry	**Paster Institute (Pasteur Institute)**	Paris
t_3	Roald Hoffmann	1937-07-18	**Ukraine (United States)**	**National Medal of Science (Nobel Prize in Chemistry)**	**Columbia University (Cornell University)**	Ithaca
t_4	Melvin Calvin	1911-04-08	United States	Nobel Prize in Chemistry	**University of Minnesota (UC Berkeley)**	**St. Paul (Berkeley)**

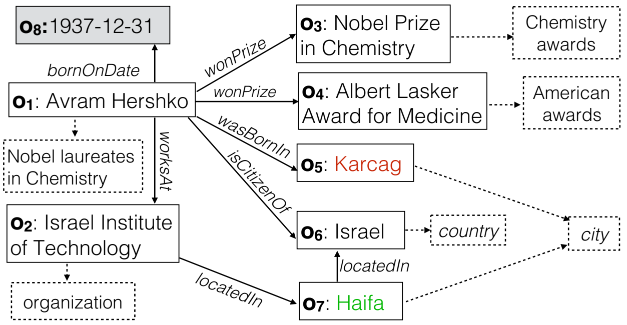

Fig. 1 Excerpt of laureates knowledge bases

(a) *Karcag* is the city he was born in;
(b) *Haifa* is the city he works in, via the links *Avram Hershko* worksAt *Israel Institute of Technology* that in turn locatedIn *Haifa*; and
(c) *Karcag* is different from *Haifa*, we can judge that t_1[City] is wrong.
The way to identity the error in t_1[Prize] is similar, if we know this column should be Chemistry awards rather than American awards.

(iii) Correction Following (*ii*), we can draw the value *Haifa* from the KB to update t_1[City] from *Karcag* to *Haifa*.

Challenges Example 2 shows that we can make different judgments, based on various evidence from KB. Nevertheless, effectively employing reliable KBs faces several challenges.

(i) *Semantic connections between relations and* KBs In order to collect evidence from KBs and judge on the relations at hand, it requires to build graphical (semantic) connections between tables and KBs.
(ii) *Ambiguity of repairing* A typical ambiguity raised by IC-based approaches is that they cannot tell precisely which attribute value is to be repaired. Hence, we need to explicitly specify which attribute is wrong and how to repair, a departure from traditional ICs that only detect errors.
(iii) *Efficiency and scalability* Repairing a relation using multiple rules by collecting evidence from a large KB (a graph) is a costly task, which requires efficient and scalable solutions to execute in practice.

Contributions Our main contribution is to propose a new type of rules to deterministically tell how to repair relations using trusted KBs. We summarize our contributions below.

(1) We formally define detective rule (DR), to address the above challenges (i) and (ii). A DR simultaneously models two opposite semantics of an attribute belonging to a relation using types and relationships in a KB (Sect. 3): The *positive semantics* explains how its value should be linked to other attribute values in a correct tuple, and the *negative semantics* indicate how a wrong attribute value is connected to other correct attribute values within the same tuple. Naturally, a DR can mark correct values in a tuple if it matches the *positive semantics*. Meanwhile, a DR can detect/repair an error if it matches the *negative semantics*.
(2) We study several fundamental problems of using DRs (Sect. 4). Specifically, given a set Σ of DRs, we determine whether these rules are consistent. We prove that this problem is coNP-complete. We also study the problem of whether another DR is implied by Σ, which is also proved to be a coNP-complete problem.
(3) We devise efficient algorithms for cleaning a relation, given a set of consistent DRs, by smartly selecting the right order to apply DRs and by using various indexes such as rule indexes and signature-based indexes (Sect. 5). This is to cope with challenge (iii).
(4) We discuss the problem of generating detective rules from examples (Sect. 6).
(5) We experimentally verify the effectiveness and efficiency of the proposed algorithms (Sect. 7). We find that algorithms with DRs can repair and mark data with high accuracy. In addition, they scale well with the number of DRs.

Organization Section 2 discusses related work. Section 3 defines DRs. Section 4 studies its fundamental problems. Section 5 presents efficient algorithms for applying consistent DRs. Section 6 discusses how to generate DRs from examples. Section 7 presents experimental findings. Finally, Sect. 8 concludes this paper.

2 Related work

This work extends our conference version [26] by including: (1) a comprehensive analysis of fundamental problems associated with detective rules, including termination, consistency, determinism and implication (Sect. 4); (2) an algorithm of multiple-version repairs that a tuple has multiple versions to be repaired (Sect. 5.3); (3) a method on how to generate DRs (Sect. 6); and (4) a new empirical evaluation of rule generation techniques (Sect. 7 Exp-3). None of the detailed proofs of (1) was presented in [26]. We extend Corollary 2 in [26] and prove that the consistency problem is PTIME without assuming $|R|$ is a constant in this work. Besides, we study the implication problem w.r.t. DRs in this article, which was not discussed in [26]. The algorithm of (2) about how to support multiple-version repairing was not given in [26]. We modify the framework of (3) in the conference article, and the method for rule refinement was not addressed, which is important for employing DRs. The experimental study of (4) was not given in [26].

Constraint-based data cleaning IC-based heuristic data cleaning methods have been widely studied [7,12,19] for the problem introduced in [4]: Repairing is to find another database that is consistent and minimally differs from the original database. They compute a consistent database by using different cost functions for value updates and various heuristics to guide repairing. However, the consistency may not be an ideal objective, since the ground truth database is consistent, but not vice versa. That is, a consistent database may not be correct, which indicates that the optimal solution of the above problem does not ensure correctness. In contrast, rule-based data cleaning is typically more conservative and reliable, since it does not use heuristics. DRs only mark data as correct or wrong, and repair errors when the evidence is sufficient.

Rule-based data cleaning Different methods exist in the literature regarding rule-based data cleaning: editing rules [23], fixing rules [55] and methods to discover fixing rules by interacting with users [27], and Sherlock rules [31]. Editing rules [23] use relational master data and interact with users for trusted repairing. Fixing rules [55] encode constant values in the rules for automated cleaning. Closer to this work is Sherlock rules [31] that automatically annotate and repair data. Along the same line with them, DRs are the research effort to leverage KBs for data cleaning. The new challenges of using KBs are remarked earlier in Sect. 1.

Table understanding using KBs Table understanding, including identifying column types and the relationship between columns using KBs, has been addressed by several techniques such as those in [14,34,40,45,50,59,60]. In fact, these techniques are friends, instead competitors, of DRs. We will show how they can help to discover DRs (Sect. 6).

KB powered data cleaning KATARA [11] is a KB and crowd powered data cleaning system that identifies correct and incorrect data. The table patterns [11] introduced by KATARA are a way of explaining table semantics in a holistic way. However, (1) KATARA cannot detect errors automatically: Whenever a mismatch happens between a tuple and a KB w.r.t. a table pattern, KATARA will ask the crowd workers to identify that such a mismatch is caused by an error in the tuple, or incompleteness of the KB; and (2) KATARA cannot repair errors automatically: When an error, such as (China, Shanghai) for relation (country, capital), is identified by users, KATARA cannot tell which value is wrong. One main difference between DRs and KATARA is that DRs can precisely tell which attribute of a tuple is wrong.

User guided data cleaning Several approaches [23,27,28,43,57] have been proposed to involve experts as first-class citizen. Involving users is certainly valid and useful for specific applications. DRs are our attempt to relieve users from the tedious and iterative data cleaning process. There are some recent studies that utilize crowdsourcing to clean and integrate the data [8,35,36,39,54].

Machine learning-based data cleaning There are also works that repair data using machine learning-based methods [44,56,57]; or more generally, running probabilistic inference to derive the true values [5,42,46].

We differ from machine-based approaches in that rule-based approaches are declarative. As discussed and demonstrated in [47,48], interpretable rules are often preferred by end users, since they are interpretable, easy to debug, maintain, and easy to inject domain knowledge. Not surprisedly, rule-based cleaning methods have been deployed in industries for decades, such as ETL rules (see [29] for a survey), but heuristic methods are rarely deployed in practice.

Data cleaning systems There are more general-purpose data cleaning systems that can be used to encode any data quality rules (please see a recent experimental paper [1] that uses real-world datasets to evaluate different error detection methods) and data cleaning algorithms, such as NADEEF [13] and BigDansing [33]. In fact, our detective rules can be easily implemented and incorporated in these systems.

3 Detective rules

We first introduce notations for knowledge bases (KBs) (Sect. 3.1). We then present the basic concepts of building connections between relations and KBs (Sect. 3.2). We close this section by defining detective rules (Sect. 3.3) and describe the semantics of applying multiple detective rules (Sect. 3.4).

3.1 Knowledge bases

We consider KBs as RDF-based data, defined using Resource Description Framework Schema (RDFS).

Classes, instances, literals, relationships, and properties

- A *class* represents the concept of a set of entities, e.g., country.
- An *instance* represents an entity, e.g., *Israel*, which belongs to a class, e.g., type (*Israel*) = country.
- A *literal* is a string, a date, or a number. For example, the birth date of *Avram Hershko* is *1937-12-31*, which is a literal.
- A *relationship* is a binary predicate that represents a connection between two instances.

For instance, isCitizenOf(*Avram Hershko, Israel*) indicates that *Avram Hershko* is a citizen of *Israel*, where isCitizenOf is a relationship defined in a KB. An instance can have some properties, e.g., bornOnDate. A *property* is a binary predicate that connects an instance and a literal.

Let **I** be a set of instances, **L** a set of literals, **C** a set of classes, **R** a set of relationships, and **P** a set of properties.

RDF graphs An *RDF* dataset is a set of triples $\{(s, r, o)\}$, where s is an instance in **I**, r is a relationship in **R** or a property in **P**, o is an object in $\mathbf{I} \cup \mathbf{L}$ (i.e., either an instance or a literal). We model the set of triples $\{(s, r, o)\}$ as a directed graph. Each vertex v is either an s or an o from the given triples. Each directed edge $e : (s, o)$ corresponds to a triple (s, r, o), with r as the edge label denoted by $\mathsf{rel}(e) = r$.

Please refer to Fig. 1 as a sample RDF graph, which describes an excerpt of Yago describing *Avram Hershko*.

Remark In practice, KBs are not 100% accurate. There have been several efforts that aim at improving the quality and coverage of KBs [16–18]. Nevertheless, KBs are usually more reliable than the dirty relational data at hand. Hence, in this work, when applying KBs, we assume that they are correct, i.e., finding and correcting KB errors is out of the scope of this work. Moreover, the *incompleteness* of KBs makes the case happen that an entity that is in KBs but cannot be matched. Although we can use approximate graph matching methods [3] to improve the recall, but will decrease the precision correspondingly. Which matching strategy to use, exact or approximate, is up to the end users to decide.

3.2 Schema- and instance-level matching graphs

Given a table D of schema R, and a KB K, next we discuss how to build connections between them, a necessary step to collect evidence from K for D. Generally speaking, we need schema-level matching graphs to explain the schema of D using K, and instance-level matching graphs to find values in K that correspond to tuples in D.

Schema-level matching graphs A *schema-level matching graph* is a graph $\mathbf{G^S}(\mathbf{V^S}, \mathbf{E^S})$, where:

1. each vertex $u \in \mathbf{V^S}$ specifies a *match* between a column in D and a type in K. It contains three labels:
   (a) $\mathsf{col}(u)$: the corresponding column in D;
   (b) $\mathsf{type}(u)$: a type in K - either a class or a literal;
   (c) $\mathsf{sim}(u)$: a similarity-based matching operation.
2. for two different nodes $u, v \in \mathbf{V^S}$, $\mathsf{col}(u) \neq \mathsf{col}(v)$.
3. each directed edge $e : (u, v) \in \mathbf{E^S}$ has one label $\mathsf{rel}(e)$, which is a *relationship* or *property* in K, indicating how $\mathsf{col}(u)$ and $\mathsf{col}(v)$ are semantically linked.

An important issue is to define the matching operation $\mathsf{sim}(u)$ (the above 1(c)) between a column in a relation and a class in a KB, which will be used later to decide whether two values match. We can utilize similarity metrics [32], e.g., Jaccard, Cosine or edit distance. For example, if string equality "=" is used, a cell value in column $\mathsf{col}(u)$ and an instance in K with class $\mathsf{type}(u)$ refer to the same entity if they have identical value. If "ED, 2" is used, a cell value in column $\mathsf{col}(u)$ and an instance in K with class $\mathsf{type}(u)$ refer to the same entity if their edit distance[1] is within 2. Without loss of generality, we take string equality and edit distance as examples.

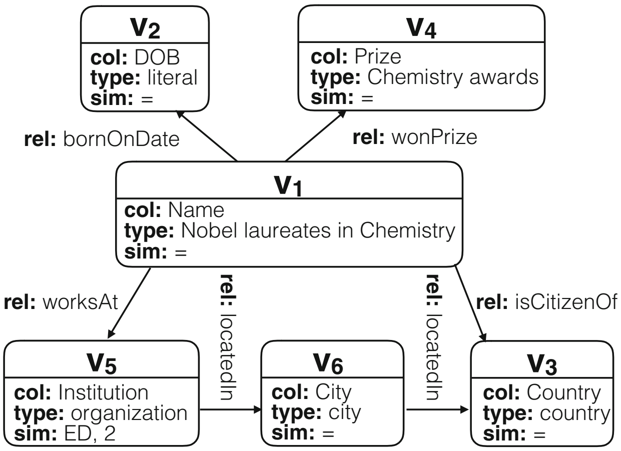

Fig. 2 A sample schema-level matching graph

Example 3 A sample schema-level matching graph for the Nobel table in Table 1 is given in Fig. 2. Node v_1 shows that column Name corresponds to the class Nobel laureates in Chemistry in the KB. To match a value in column Name and a value in the KB with type Nobel laureates in Chemistry, string equality "=" is used. "ED, 2" is used in node v_5 to tolerate the errors between t[Institution] and an instance of organization in KB. The directed edge (v_1, v_2) labeled bornOnDate explains the relationship between columns Name and DOB.

By default, we assume that a schema-level matching graph is connected. Naturally, any induced subgraph of a schema-level matching graph is also a schema-level matching graph. In other words, a schema-level matching graph is not necessarily a global understanding of the table (see Fig. 2). In contrast, it is a local interpretation about how partial attributes of a table are semantically linked, e.g., Fig. 3a. Please refer to Fig. 2 for the corresponding node labels.

Construction The schema-level matching graph is essentially the table semantics interpreted by a KB, which has been widely studied. A basic idea is to connect a column to a type by matching the cell-value set of the column to the entity set

[1] Edit distance of two instances is the minimum number of edit transformations from one to the other, where the edit operations include insertion, deletion and substitution. For example ED(Chemistry, Chamstry) = 2.

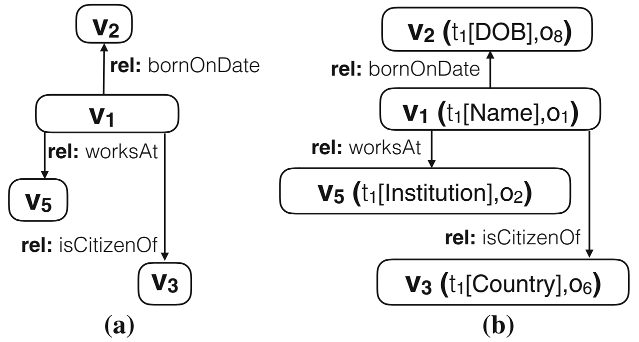

Fig. 3 A sample schema/instance-level matching graph. **a** Schema-level and **b** instance-level

of the type, and we can use existing tools [11,14,40] to build the schema-level matching graph.

Instance-level matching graph An *instance-level matching graph*, denoted by $\mathbf{G^I}(\mathbf{V^I}, \mathbf{E^I})$, is an instantiation of a schema-level matching graph w.r.t. one tuple t in the relational table and some instances from the KB. More formally speaking:

1. For each node $u_i \in \mathbf{V^I}$, there is an instance x_i from the KB such that the types match, i.e., $\mathsf{type}(u_i) = \mathsf{type}(x_i)$; and the values match, i.e., $t[\mathsf{col}(u_i)]$ and x_i are similar based on the similarity function $\mathsf{sim}(u_i)$.
2. For each edge $(u_i, u_j) \in \mathbf{E^I}$, the correspondingly matched KB instances x_i, x_j satisfy $\mathsf{rel}(u_i, u_j) = \mathsf{rel}(x_i, x_j)$, i.e., the two instances x_i and x_j in the KB have the relationship required by the schema-level matching graph.

Example 4 Consider the small schema-level matching graph shown in Fig. 3a. One instance-level matching graph for tuple t_1 in Table 1 is given in Fig. 3b, where the types and relationships of the nodes o_1, o_2, o_6, o_8 can be verified from the KB in Fig. 1.

Limitations Indeed, *matching operation* is the core of data cleaning, since one always needs to link different real-world entities. Historically, many matching operators have been studied, e.g., matching dependencies [22] for two tables, keys for graphs [20] defined on one graph, and Swoosh [6] for matching two generic objects for entity resolution. When there is a match, one common usage is to say something is correct. The main limitation for detecting errors is that, when there is a mismatch, it cannot tell that something is wrong.

Opportunities If we define some matching operations to capture negative semantics, intuitively, the errors can be detected. This observation reveals the opportunity to define new methods to match (i.e., detect) data errors. For instance, in Sect. 1 Example 2 Case (ii), if we know the followings:

1. the column City in the table is where he works in;
2. the current value t_1[City] is *Karcag*;
3. he works in *Haifa*, derived from the KB; and
4. the two values *Karcag* and *Haifa* are different, we may decide that *Karcag* is an error.

3.3 Detective rules

The broad intuition of our proposal is that, for a column, if we can simultaneously capture both the *positive semantics* of what correct values should look like, and the *negative semantics* of what wrong values commonly behave, we can detect and repair errors.

Consider column City in Table 1. We can discover one schema-level matching graph to capture the semantics *lives at*. Similarly, we can find another one to capture the semantics *born in*. If the user enforces City to have the *lives at* semantics, and we find that the value in the table maps to the *born in* semantics, we know how to repair. Note that some semantics can be captured by a directed edge, e.g., wasBornIn in Fig. 1 for the *born in* semantics, while some other semantics needs to be captured by more than one edge, e.g., putting worksAt and locatedIn in Fig. 1 together for the *lives at* semantics.

Let $\mathbf{G^S_1}(\mathbf{V^S_1}, \mathbf{E^S_1})$ and $\mathbf{G^S_2}(\mathbf{V^S_2}, \mathbf{E^S_2})$ be two schema-level matching graphs that exist a node $p \in \mathbf{V^S_1}$ and a node $n \in \mathbf{V^S_2}$ such that (i) $\mathsf{col}(p) = \mathsf{col}(n)$ and (ii) the subgraphs $\mathbf{G^S_1} \setminus \{p\}$ and $\mathbf{G^S_2} \setminus \{n\}$ are isomorphic, where $\mathbf{G^S_1} \setminus \{p\}$ (resp. $\mathbf{G^S_2} \setminus \{n\}$) is a subgraph of $\mathbf{G^S_1}$ (resp. $\mathbf{G^S_2}$) by removing node p (resp. n) and associated edges. Obviously, both graphs are defined over the same set of columns in the relation: $\mathbf{G^S_1}$ is to capture their positive semantics and $\mathbf{G^S_2}$ is for the negative semantics of values in column $\mathsf{col}(n)$.

Detective rules A *detective rule* is a graph $\mathbf{G}(\mathbf{V}, \mathbf{E})$ that merges the above two graphs $\mathbf{G^S_1}$ and $\mathbf{G^S_2}$. Let $\mathbf{V_e} = \mathbf{G^S_1} \setminus \{p\} = \mathbf{G^S_2} \setminus \{n\}$. The node set in DR is $\mathbf{V} = \mathbf{V_e} \cup \{p, n\}$. The edges $\mathbf{E}$ are all the edges carried over from the above two graphs. Note that $\mathsf{col}(p) = \mathsf{col}(n)$. We call p the *positive node*, n the *negative node*, and $\mathbf{V_e}$ the evidence nodes.

Semantics Let $\mathsf{col}(\mathbf{V_e})$ be the columns corresponding to the evidence nodes $\mathbf{V_e}$ of a DR, and $\mathsf{col}(p) = \mathsf{col}(n)$. Let $|\mathbf{V_e}|$ be the cardinality of the set $\mathbf{V_e}$. Consider a tuple t over relation R and a KB K:

(1) Proof positive If there is an instance-level matching graph between t and $|\mathbf{V_e}| + 1$ instances in K w.r.t. the nodes $\mathbf{V_e} \cup \{p\}$, i.e., the positive semantics is captured, we say that the attribute values of $t[\mathsf{col}(\mathbf{V_e}) \cup \mathsf{col}(p)]$ are correct.

(2) Proof negative If there is an instance-level matching graph between t and $|\mathbf{V_e}|+1$ instances in K w.r.t. the nodes $\mathbf{V_e} \cup \{n\}$, we say that the attribute values of $t[\mathsf{col}(\mathbf{V_e})]$ are correct, but the value $t[\mathsf{col}(n)]$ is potentially wrong, i.e., the negative semantics is captured. In addition, if we can find another instance x in K such that if we replace $t[\mathsf{col}(n)]$ by x, we

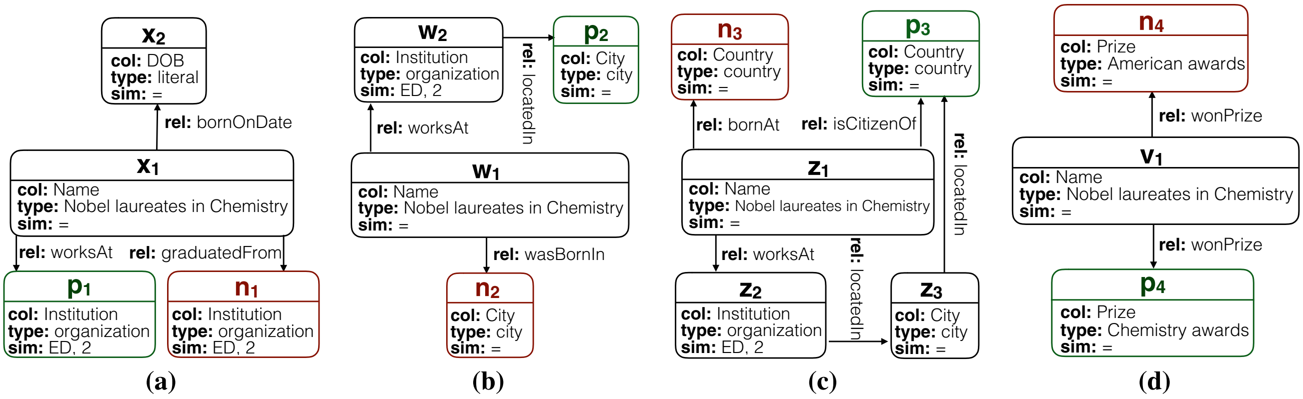

Fig. 4 Sample detective rules. **a** Rule φ_1, **b** rule φ_2, **c** rule φ_3 and **d** rule φ_4

can find the case of proof positive as in (1). At this point, we confirm that $t[\mathsf{col}(n)]$ is wrong.
(3) Correction Following the above case (2), we know the correct value for $t[\mathsf{col}(n)]$ is the new instance x from K.

Intuitively, a DR specifies how to judge whether a set of attribute values of a tuple is correct (the above (1)), how to find a wrong attribute value of the tuple (the above (2)), and how to repair the identified error (the above (3)). Besides, note that each node u in DR has a label $\mathsf{sim}(u)$. If a cell value in $\mathsf{col}(u)$ can match an instance with $\mathsf{type}(u)$ in K based on $\mathsf{sim}(u)$ but not equal, DR will modify them to identical value, e.g., t_2[Institution] from *Paster Institute* to *Pasteur Institute*.

Example 5 Figure 4 shows four DRs. We discuss their semantics w.r.t. t_1 in Table 1 and KB in Fig. 1.
(1) Proof positive Consider rule φ_1. We can find that o_1 in Fig. 1 matches t_1[Name] w.r.t. node x_1 in φ_1; o_8 in Fig. 1 matches t_1[DOB] w.r.t. node x_2 in φ_1; and o_2 in Fig. 1 matches t_1[Institution] w.r.t. node p_1 in φ_1. Moreover, the relationship from o_1 to o_8 is bornOnDate and from o_1 to o_2 is worksAt. That is, both value constraints and structural constraints enforced by φ_1 are satisfied. Consequently, we can conclude that t_1[Name, DOB, Institution] are correct.
(2) Proof negative Consider rule φ_2. We can find that o_1 in Fig. 1 matches t_1[Name] w.r.t. node w_1 in φ_2; o_2 in Fig. 1 matches t_1[Institution] w.r.t. node w_2 in φ_2; o_5 in Fig. 1 matches t_1[City] w.r.t. node n_2 in φ_2. Moreover, it can find a node o_7, with value *Haifa*, in Fig. 1 that satisfies the constraints imposed on p_2 in φ_2. Combined with the other edge relationships from o_1 to o_2, o_2 to o_7, and o_1 to o_5, we can confirm that $t_1[\text{City}] = \textit{Karcag}$ is an error.
(3) Correction Following case (2), we know t_1[City] should be repaired to *Haifa*.

Other rules will be discussed later in this paper.

Remark There might exist multiple repairs (i.e., multi-version ground truth) for one error, and each makes sense, e.g., one country may have multiple capitals and one person may have different nationalities. For the simplicity of the discussion, we assume that there is only one repair for this moment, i.e., the corresponding relationship in the KB is functional. We will present algorithms to handle multiple repairs in latter sections. Also, we allow only one negative node n in the rule, which is also to simplify our discussion. It is straightforward to extend from one negative node (i.e., one relationship) to a negative path (i.e., a sequence of nodes) in order to identify an error.

Rule applicability Apparently, DRs can handle semantic error well where the value is replaced with a different one from a semantically related domain. Also, one can define domain specific similarity functions in DRs to capture whether two labels match or not, which may possibly capture typos and other types of errors, depending on the power of the similarity functions. Note, however, that when a similarity function is used, typically, the recall will increase but the precision will decrease.

3.4 Applying multiple detective rules

We will start by discussing how to apply one DR, followed by using multiples DRs. For simplicity, at the moment, we assume that each DR will return a single repair.
Applying one rule Consider one DR $\varphi : \mathbf{G}(\mathbf{V}, \mathbf{E})$, a tuple t, and a KB K. Applying φ to t has only two cases: (1) *Proof positive:* the attribute values $t[\mathsf{col}(\mathbf{V_e} \cup \{p\})]$ are correct; (2) *Proof negative and correction:* the attribute values $t[\mathsf{col}(\mathbf{V_e})]$ are correct, the attribute value $t[\mathsf{col}(n)]$ is wrong, we will update $t[\mathsf{col}(n)]$ using an instance x drawn from the KB K.

We use the symbol "+" to mark a value as positive, which is confirmed either from the above (1), the evidence attributes $\mathbf{V_e}$ from the above (2), or a wrong value $t[\mathsf{col}(n)]$ as in the above (2) but has been corrected and thus is marked as positive. The other attributes that are not marked as positive are those whose correctness is *unknown*.
Marked tuples A tuple is a *marked tuple* if at least one of its attribute values has been marked as positive.

Example 6 We discuss how to apply rule φ_2 to tuple t_1 by following Example 5 cases (2) & (3). t_1[Name] and t_1[Institution] are identified to be correct, and t_1[City] is wrong. After changing t_1[City] to *Haifa*, it will mark t_1 as t_1'(*Avram Hershko*$^+$, *1937-12-31, Israel, Albert Lasher Award for Medicine, Israel Institute of Technology*$^+$, *Haifa*$^+$).

Applying multiple rules When applying multiple DRs, we need to make sure that the values that have been marked as positive cannot be changed any more by other DRs. Consider a *marked tuple t* of relation R that the attributes X have been marked as positive. We say that a DR φ is *applicable* to t, if (i) it will not change $t[X]$; and (ii) it can mark some values in $t[R \setminus X]$ as positive, with (i.e., proof negative and correction) or without (i.e., proof positive) value updates.

Fixpoint A *fixpoint* of applying a set of DRs on a tuple t is the state that no more rules are further applicable.

Example 7 After applying rule φ_2 to tuple t_1 as in Example 6, rule φ_3 is applicable. Applying rule φ_3 will not change any value, but will mark the tuple as t_1''(*Avram Hershko*$^+$, *1937-12-31, Israel*$^+$, *Albert Lasher Award for Medicine, Israel Institute of Technology*$^+$, *Haifa*$^+$). Rule φ_4 will repair t_1''[Prize] and rule φ_1 marks t_1''[DOB] as positive. At the end, tuple t_1 is modified to t_1'''(*Avram Hershko*$^+$, *1937-12-31*$^+$, *Israel*$^+$, *Nobel Prize in Chemistry*$^+$, *Israel Institute of Technology*$^+$, *Haifa*$^+$). It is a fixpoint, since no more rule can be further applied.

Multiple-version repairs Although not desired, the case that there are multiple-version repairs for one error by using one DR does happen. Note that this is different from IC-based repair to guess which value is wrong in, e.g., (*Netherlands, Rotterdam*) for {country, capital}. Instead, in our case, we know that *Rotterdam* is not the capital of *Netherlands*, and we find two repairs, *Amsterdam* and *Den Hagg*, and both are correct. In reality, when the user picks DRs (Sect. 6), they will pick the ones that semantically, the repair is approximately functional, e.g., the capital of a country or a nationality of a person, not a city of a country or a hobby of a person.

When multiple-version repairs happen for applying one DR to a tuple t, instead of having one marked tuple t', we generate multiple marked tuples T. These tuples T mark exactly the same set of attributes as positive, and these tuples are different only on one attribute w.r.t. the negative node in the given DR. This can be easily propagated to apply multiple DRs.

4 Fundamental problems

In this section, we study the fundamental problems associated with detective rules and establish their complexity.

4.1 Termination

One natural question for rule-based data cleaning processes is the termination problem that determines whether a rule-based process will stop.

Termination problem Given a tuple t over relation R, a KB K and a set Σ of DRs, the $termination$ problem is to determine whether any repairing process leads to a fixpoint.

Note that when applying any rule to t, we have that the number of marked positive attributes will strictly increase. That is, up to $|R|$ DRs can be applied to any tuple and the termination is naturally assured.

4.2 Consistency

Another important problem of applying any data cleaning rules is to make sure that it makes sense to put these rules together.

Consistency problem Let Σ be a set of DRs and K a KB. Σ is said to be *consistent* w.r.t. K, if given any tuple t, all the possible repairs via Σ and K terminate in the same fixpoint(s), i.e., the repair is unique.

Theorem 1 *The consistency problem for detective rules is* coNP*-complete, even when the knowledge base K is given.*

Proof We first show it is in coNP and then show it is coNP-hard.

Upper bound The coNP upper bound is verified by providing an NP algorithm. The algorithm returns 'Yes' iff Σ is not consistent. The NP algorithm works as follows:

(1) Guess a tuple t that draws values from adom, where adom is the set of all instances in K w.r.t. Σ plus an additional distinct constant not in adom;
(2) Guess two sequences of rules and check whether t has the same fixpoint for both sequences. If the answer is no, return 'Yes';

Both step (1) and step (2) run in nondeterministic PTIME, the consistency problem is in coNP.

Lower bound For the lower bound, we show that the problem is coNP-hard by a reduction from the 3SAT problem.

An instance of the 3SAT problem is a well-formed Boolean formula $\tau = C_1 \wedge \ldots \wedge C_n$, where all the variables in τ are $x_1, \ldots, x_p$, the clause C_i is of the form $l_{i1} \vee l_{i2} \vee l_{i3}$, and each literal l_{ij} is either x_k or $\overline{x}_k$ for $k \in [1, p]$. The 3SAT problem is then to determine whether there is a truth assignment such that τ is satisfied. Given an instance τ of the 3SAT problem, we define an instance of the consistency problem for detective rules such that Σ is not consistent iff τ is satisfiable.

Definition of Relation R and Tuples in R. The relation R has $n + 2$ attributes, $(X, C_1, \ldots, C_n, B)$, where each attribute

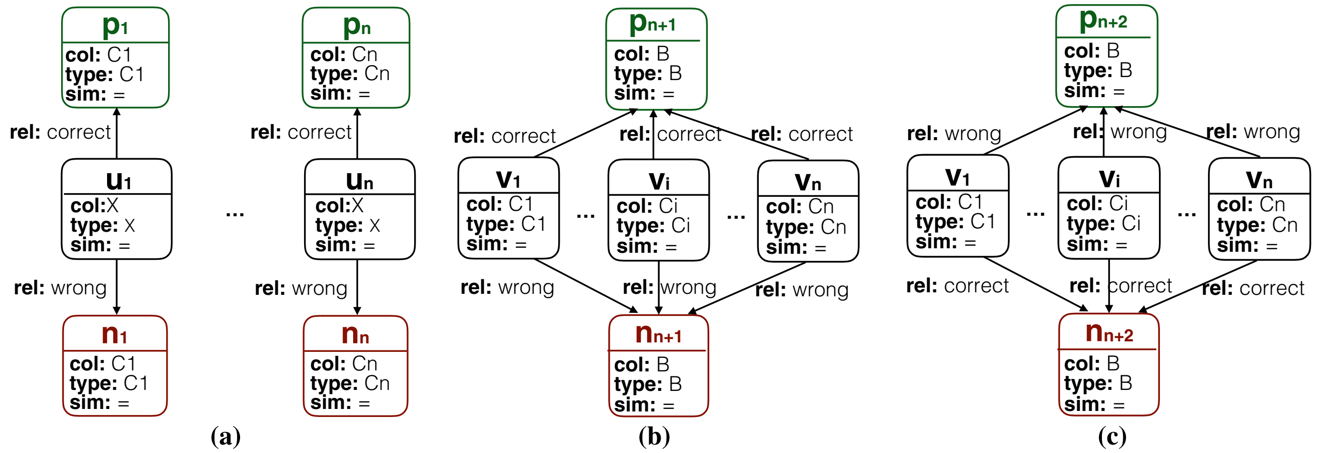

Fig. 5 The set Σ of DRs in 3SAT. **a** Rule φ_1 to φ_n, **b** rule φ_{n+1} and **c** Rule φ_{n+2}

value of X is p-sized corresponding to the assignment of variables $x_1, \ldots, x_p$, C_i corresponds to clauses, and B is used to check whether there exists conflict. There are 2^p tuples. X contains all the possible combination of truth assignments for variables $x_1, \ldots, x_p$, and each cell value in the last $n+1$ attributes is false.

Definition of Classes in The Knowledge Base. There are $n+2$ classes in the knowledge base: $X, C_1, \ldots, C_n, B$, where X corresponds to variables, C_i corresponds to clauses and B is used to check conflicts.

Definition of Instances in The Knowledge Base. For class X, its instance is a p-sized entity (true/false, true/false, ..., true/false) that corresponds to a combination of variables $x_1, x_2, \ldots, x_p$. If a tuple can make a clause true, there exists such an instance; not otherwise. For each class $C_1, \ldots, C_n, B$, there exist two instances true and false.

Definition of Relationships in The Knowledge Base. There exist relationships between instances of X and instances of C_i. If an instance of X makes the clause C_i true, we add a relationship between the instance of X and the true instance of C_i, and the relationship label is correct; we also add a relationship between the instance of X and the false instance of C_i, and the relationship label is wrong.

There also exist relationships between instances of C_i and B. There exists a relationship between the true instance in C_i and the true/false instance in B, and the relationship label is correct/wrong.

Definition of Rules in The Knowledge Base. We define $n+2$ rules. The first n rules want to correct C_i based on X, where column X matches type X. More specifically, consider $C_i = (l_{i1} \vee l_{i2} \vee l_{i3})$, where l_{ij} is either x_k or $\overline{x}_k$. The corresponding rule φ_i dictates that if the value of l_{ij} $(j = 1, 2, 3)$ satisfies C_i, the rule should correct the value of C_i to true. The $n+1$-th rule φ_{n+1} aims to correct attribute B by matching $C_1, C_2, \ldots, C_n$. If C_i = true for all $i = 1, \ldots, n$, we should correct B to value true. The $n+2$-th rule φ_{n+2} aims to correct attribute B based by matching $C_1, C_2, \ldots, C_n$. If C_i = true for all $i = 1, \ldots, n$, we should correct B to value false. The rules are defined in Fig. 5.

3SAT vs Our problem. Assume that τ is satisfiable. That is, there exists a true assignment ν for each variable x_i such that τ = true. Consider the tuple $t = \{\nu(x_1), \ldots, \nu(x_p)$, false, ..., false, false}. Applying φ_1 to φ_n can update all values in $C_1, \ldots, C_n$ to true. Then, we have that B can have two different values. Therefore, $\varphi_{n+1}, \varphi_{n+2}$ contradict to each other and Σ is not consistent. Conversely, if τ is not satisfied, $\varphi_{n+1}, \varphi_{n+2}$ will not be applied. It is easy to see that φ_1 to φ_n are consistent, because each rule only modifies the attributes of $C_i (i \in [1, n])$, which has no impact on the other rules. Thus, Σ is consistent.

Putting the above together, the consistency problem is coNP-complete. □

The above consistency problem is hard, since one has to guess all the tuples that the given rules are applicable in an arbitrary order. Oftentimes, in practice, we only care about whether the rules are consistent w.r.t. a specific dataset D. Fortunately, the problem of checking the consistency when D is present becomes PTIME. To prove this, let us pause and present the following proposition.

Proposition 1 *The set Σ of DRs is consistent, iff any two DRs φ_i and φ_j in Σ are consistent.*

Proof We prove it by contradiction.

$\boxed{\Rightarrow}$ Suppose that the detective rules are pairwise consistent, but when putting together, they are inconsistent. In other words, they may lead to (at least) two different fixpoints.

We construct two repair processes that end up with two different final tuples t^*, t'^* as follows:

$$\rho : t \xrightarrow{\varphi_1} t_1 \ldots \xrightarrow{\varphi_i} t_i \ldots t_{m-1} \xrightarrow{\varphi_m} t^*$$
$$\rho' : t \xrightarrow{\varphi'_1} t'_1 \ldots \xrightarrow{\varphi'_j} t'_j \ldots t_{n-1} \xrightarrow{\varphi_n} t'^*$$

Consider the longest equivalent prefixes of ρ and ρ' such that $t_{i-1} \xrightarrow{\varphi} t_i$ is the ith step in ρ, $t'_{j-1} \xrightarrow{\varphi'} t'_j$ is the jth step in ρ', $t_{i-1} = t'_{j-1}$ and $t_{i+k} \neq t'_{j+l}(0 \leqslant k \leqslant m-i, 0 \leqslant l \leqslant n-j)$. We can prove that φ and φ' must be inconsistent. Let us consider two cases: (i) $\mathsf{col}(p) \notin \mathsf{col}(\mathbf{V}'_\mathbf{e} \cup \{n'\})$ and $\mathsf{col}(p') \notin \mathsf{col}(\mathbf{V}_\mathbf{e} \cup \{n\})$; (ii) $\mathsf{col}(p) \in \mathsf{col}(\mathbf{V}'_\mathbf{e} \cup \{n'\})$ or $\mathsf{col}(p') \in \mathsf{col}(\mathbf{V}_\mathbf{e} \cup \{n\})$.

For case (i), we know that rule φ will update $\mathsf{col}(p)$ of tuple t_{i-1}. Note that it is impossible that φ only marks $\mathsf{col}(p)$ as positive without changing any value, because in this situation t_i and t_{j-1} will be the longest equivalent prefixes. Since $\mathsf{col}(p) \notin \mathsf{col}(\mathbf{V}'_\mathbf{e} \cup \{n'\})$, rule φ' can be applied to t_i. In the same way, rule φ can be applied to t'_j. The related output will then be tuples t_{i+1}, t'_{j+1}, and by construction, $t_{i+1} = t'_{j+1}$. However, we have assumed that t_{i-1} and t'_{j-1} are the longest equivalent prefixes, which contradicts to this case.

For case (ii), without loss of generality, we assume that $\mathsf{col}(p) \in \mathsf{col}(\mathbf{V}'_\mathbf{e} \cup \{n'\})$. Since φ will update $\mathsf{col}(p)$ of t_{i-1} and $\mathsf{col}(p) \in \mathsf{col}(\mathbf{V}'_\mathbf{e} \cup \{n'\})$, rule φ' can not be applied to t_i. Rule φ also can not be applied to t'_j. Then, we have that $t_{i-1} = t'_{j-1}$ but $t_i \neq t'_j$. That is to say, applying φ and φ' to the same tuple in different order will lead to different results. φ and φ' are inconsistent which is against the assumption.

Putting all contradicting cases together, it suffices to see that we were wrong to assume that Σ is inconsistent.

$\boxed{\Leftarrow}$ Suppose that Σ is consistent, but there exist φ and φ' in Σ that are inconsistent.

For any tuple t that leads to different fixes by φ and φ', t' and t'' are the fixpoints we first apply φ and φ', respectively. Then we can construct the following two fixes ρ and ρ' on t by using the rules in Σ:

$$\rho : t \xrightarrow{\varphi,\varphi'} t' \xrightarrow{\Sigma-\{\varphi,\varphi'\}} t^*$$
$$\rho' : t \xrightarrow{\varphi',\varphi} t'' \xrightarrow{\Sigma-\{\varphi',\varphi\}} t'^*$$

Since φ and φ' are inconsistent, there must exist an attribute A that $t'[A] \neq t''[A]$. Without loss of generality, we assumed that $t[A] \neq t'[A]$ and $t'[A]$ is annotated as positive by applying φ in ρ. If $t''[A]$ is also annotated as positive, then $t^*[A] \neq t'^*[A]$. Σ is inconsistent. If $t''[A]$ remains free ($t[A] = t''[A]$), which means φ is not applied in ρ', there exists an evidence attribute $E \in \mathsf{col}(\mathbf{V}_\mathbf{e})$ that $t''[E] \neq t[E] = t'[E]$. So $t^*[E] \neq t'^*[E]$ and Σ is inconsistent. In summary, ρ and ρ' must yield two different fixpoints. This is contradict that Σ is consistent. □

Corollary 1 *Given an instance D over relation R and a* KB *K, the consistency problem for a set Σ of DRs w.r.t. D and K is PTIME.*

Proof Given each tuple $t \in D$ over relation R, a set Σ of DRs, two DRs $\varphi, \varphi' \in \Sigma$ and a KB K, checking whether φ and φ' is consistent for t is PTIME. Since there are $O(|\Sigma|^2)$ pairs of DRs in Σ, checking whether Σ is consistent for t is PTIME. Also, the number of tuples is $|D|$. Naturally, we can check whether Σ is consistent for D in PTIME. □

The above result brings us to the bright side that practically it is feasible to make sure that a set of DRs is consistent for the dataset at hand. In our experiments, when a set of rules are selected, we run them on random sample tuples to check whether they always compute the same results. If not, we will ask users to double check the selected rules. In the following of the paper, we build our discussion using consistent DRs.

4.3 Determinism

Determinism problem The *determinism* problem is to decide whether all terminating cleaning processes end up with the same repair.

Based on the definition of consistency, we can get that, if a set Σ of DRs is consistent w.r.t. a KB K, for any tuple t of D, applying Σ to t will terminate, and the repair is deterministic.

4.4 Implication

Given a set Σ of consistent DRs, and another DR φ that is not in Σ, we say that φ is implied by Σ, denoted by $\Sigma \models \varphi$, if

(1) $\Sigma \cup \{\varphi\}$ is consistent;
(2) for any tuple $t \in D$ and KB K, applying Σ or $\Sigma \cup \{\varphi\}$ on t agrees on the same fixpoint(s).

Condition (1) says that Σ and φ must agree with each other. Condition (2) ensures that for any tuple t, the outputs of applying Σ or $\Sigma \cup \{\varphi\}$ are the same, which indicates that φ is redundant.

Implication problem The *implication* problem is to decide that, given a set Σ of consistent DRs and another rule φ, whether Σ implies φ for any tuple and knowledge base.

The implication analysis can help us find and remove redundant rules from Σ. However, the implication problem is coNP-complete.

Theorem 2 *The implication problem for detective rules is* coNP*-complete, even when the knowledge base K is given.*

Proof We first prove that the problem is in coNP and then show it is coNP-hard.

Upper bound The coNP upper bound is verified by providing an NP algorithm for its complement problem. The algorithm returns 'Yes' iff $\Sigma \not\models \varphi$. The NP algorithm works as follows:

(1) Guess a tuple t that draws values from adom, where adom is the set of all instances in K w.r.t. $\Sigma \cup \{\varphi\}$ plus an additional distinct constant not in adom;
(2) Check whether $\Sigma \cup \{\varphi\}$ is consistent. If the answer is no, return 'Yes' (i.e., $\Sigma \not\models \varphi$);
(3) Check whether t has the same fixpoint by Σ and $\Sigma \cup \{\varphi\}$. If the answer is no, return 'Yes';

Step (1)(2) and step (3) run in nondeterministic PTIME. Hence the implication problem is in coNP.
Lower bound For the lower bound, we show that the problem is coNP-hard by a reduction from the 3SAT problem.

Given an instance τ of the 3SAT problem, we define an instance of the implication problem for detective rules such that $\Sigma \not\models \varphi$ iff τ is satisfiable. The techniques employed in Theorem 1 can be applied to construct the instance with few modifications. Let again $R = \{X, C_1, \ldots, C_n, B\}$. X contains all the possible combination of truth assignments for variables $x_1, \ldots, x_p$, and each cell value in the last $n+1$ attributes is false. We also utilize the knowledge base defined in Theorem 1 to repair R. That is, the definitions of classes, instances and relationships in KB are all the same. Let Σ contain φ_1 to φ_n which are shown in Fig. 5a. We now demonstrate that τ is satisfiable if and only if Σ does not imply φ_{n+1} which is in Fig. 5b (i.e., $\Sigma \not\models \varphi_{n+1}$).

Obviously, Σ is consistent, since the application of one rule has no impact on the other rules. Assume that τ is satisfiable. That is, there exists a true assignment ν for each variable x_i such that $\tau = \text{true}$. Consider the tuple $t = \{\nu(x_1), \ldots, \nu(x_p), \text{false}, \ldots, \text{false}, \text{false}\}$. Applying Σ to t can modify all values in $C_1, \ldots, C_n$ to true. It is noteworthy that $\Sigma \cup \{\varphi_{n+1}\}$ is also consistent but applying φ_{n+1} to t will further update B to true. It means that there exists a tuple t such that t has two distinct fixpoints by Σ and $\Sigma \cup \{\varphi_{n+1}\}$. Thus, $\Sigma \not\models \varphi_{n+1}$.

Suppose that τ is not satisfiable. We next prove $\Sigma \models \varphi_{n+1}$. When τ is not satisfiable, B can not be updated by φ_{n+1}. In other words, for any tuple t, $\Sigma \cup \{\varphi_{n+1}\}$ is consistent and t has the same fixpoint by Σ and $\Sigma \cup \{\varphi_{n+1}\}$. Thus, $\Sigma \models \varphi_{n+1}$, which indicates that φ_{n+1} is redundant for Σ.

In conclusion, the implication problem is coNP-complete. □

Although in its general case, checking implication is coNP-complete, in the special case which both D and K are available, we are able to checking the implication in PTIME.

Corollary 2 *Given an instance D over relation R and a* KB *K, the implication problem for a set Σ of consistent DRs and another DR $\varphi \notin \Sigma$ w.r.t. D and K is PTIME.*

Algorithm 1: Basic Repair Algorithm

```
   Input: a tuple $t$, a set $\Sigma$ of consistent DRs.
   Output: a repaired tuple $t$.
1  POS $\leftarrow \emptyset$;
2  while $\exists\varphi : \mathbf{G}(\mathbf{V_e} \cup \{p, n\}, \mathbf{E}) \in \Sigma$ that is applicable to $t$ do
3      if $\exists$KB instances match $\mathbf{V_e} \cup \{p\}$ with $t[col(\mathbf{V_e} \cup \{p\})]$ then
4          POS $\leftarrow$ POS $\cup$ col$(\mathbf{V_e} \cup \{p\})$;
5      else if $\exists$KB instances $I \cup \{x_n\}$ that match $\mathbf{V_e} \cup \{n\}$ with
       $t[col(\mathbf{V_e} \cup \{n\})]$ and $\exists$ a KB instance $x_p$ such that $I \cup \{x_p\}$
       match $\mathbf{V_e} \cup \{p\}$ and $x_p \neq x_n$ then
6          $t[\text{col}(n)] \leftarrow x_p$;
7          POS $\leftarrow$ POS $\cup$ col$(\mathbf{V_e} \cup \{p\})$;
8      $\Sigma \leftarrow \Sigma \setminus \{\varphi\}$;
9  return $t$;
```

Proof From Corollary 1, we know that checking the consistency of $\Sigma \cup \{\varphi\}$ is PTIME when both D and K are given. Thus, we just focus on the complexity of checking whether Σ and $\Sigma \cup \{\varphi\}$ bring to the same final relation D'. From Sects. 4.1 and 4.3, we already know that every repair over consistent rules is terminated and deterministic. Given a tuple $t \in D$, it needs $O(|\Sigma|)$ times to find an applicable rule, and applying one rule is PTIME. The above process iterates $O(|R|)$ times until no rule can be applied. As a result, checking whether t has the same fixpoint by Σ and $\Sigma \cup \{\varphi\}$ is PTIME. Also, the number of tuples is $|D|$. Thus, we can determine whether φ is implied by Σ for D in PTIME. □

5 Detective in action

Given a set of consistent DRs, we first present a basic repair algorithm (Sect. 5.1), followed by an optimized algorithm to speed-up the repairing process (Sect. 5.2). Finally, we extend our methods to support multiple repairs (Sect. 5.3).

5.1 Basic repairing

When a set Σ of DRs is consistent, for any tuple t, applying Σ to t will get a unique final result, which is also known as the Church-Rosser property [2]. Hence, the basic solution is a chase-based process to iteratively pick a rule that can be applied until a fixpoint is reached, i.e., no rule can be applied. *Algorithm 1* It uses a set to keep track of the attributes marked to be positive in t, initialized as empty (line 1). It picks one rule that is applicable to t in each iteration until no DR can be applied (lines 2-8). In each iteration, if there exist KB instances that match the nodes $\mathbf{V_e} \cup \{p\}$ with $t[\text{col}(\mathbf{V_e} \cup \{p\})]$, the attributes $\text{col}(\mathbf{V_e} \cup \{p\})$ are marked as positive (line 3-4). Otherwise, (i) if there exist KB instances $I \cup \{x_n\}$, such that they match nodes $\mathbf{V_e} \cup \{n\}$ with $t[\text{col}(\mathbf{V_e} \cup \{n\})]$; and (ii) if there also exist KB instances $I \cup \{x_p\}$ that if we update $t[\text{col}(n)]$ to x_p as t', $I \cup \{x_p\}$ match nodes $\mathbf{V_e} \cup \{p\}$ with

$t'[\mathsf{col}(\mathbf{V_e} \cup \{p\})]$, and (iii) if $x_p \neq x_n$, it will repair this error by the value x_p and mark it as positive (lines 5-7). Afterward, the rule will be removed from Σ since each rule can be applied only once (line 8). Finally, a repaired tuple is returned (line 9).

Complexity The loop (lines 2-8) iterates at most $|R|$ times. In each iteration, it at most checks $|\Sigma|$ unused rules to find an applicable one. Within each loop, the worse case of checking each rule node $u \in \mathbf{V}$ is $O(|C||X|)$ where $|C|$ is the number of instances belonging to $\mathsf{type}(u)$ and $O(|X|)$ is the complexity of calculating the similarity between $t[\mathsf{col}(u)]$ and a KB instance. Checking whether $t[\mathsf{col}(u)]$ and $t[\mathsf{col}(v)]$ have the relationship $\mathsf{rel}(e)$ for each edge $e : (u, v) \in \mathbf{E}$ or drawing the correct value from KB needs $O(1)$ by utilizing a hash table. Thus, the algorithm runs in $O(|\Sigma||R| \times (|C||X||\mathbf{V}| + |\mathbf{E}|))$ time, where $|\mathbf{V}|$ is the number of nodes and $|\mathbf{E}|$ is the number of edges in the rule.

Example 8 Consider tuple t_3 in Table 1 and four DRs in Fig. 4. Suppose that the rules will be checked in the order $\langle \varphi_1, \varphi_2, \varphi_3, \varphi_4 \rangle$.

For rule φ_1, since t_3 can match nodes x_1, x_2, n_1 and edges (x_1, x_2),(x_1, n_1), $t_3[\mathsf{Institution}]$ will be modified to *Cornell University* and we have t_3'(*Roald Hoffmann*$^+$, *1937-07-18*$^+$, *Ukraine, National Medal of Science, Cornell University*$^+$, *Ithaca*). φ_1 will be removed from Σ, and we next check φ_2. As $t_3'[\mathsf{Institution}]$ is located in $t_3'[\mathsf{City}]$ actually, we just mark them as positive. Then, φ_2 will be removed from Σ. For rule φ_3, since t_3' can match nodes z_1, z_2, z_3, n_3 and edges (z_1, z_2),(z_2, z_3),(z_1, n_3), $t_3'[\mathsf{Country}]$ will be modified to *United States*. φ_3 will be removed from Σ and φ_4 repair $t_3'[\mathsf{Prize}]$ to *Nobel Prize in Chemistry*. In the end, we have $\Sigma = \emptyset$ and t_3''(*Roald Hoffmann*$^+$, *1937-07-18*$^+$, *United States*$^+$, *Nobel Prize in Chemistry*$^+$, *Cornell University*$^+$, *Ithaca*$^+$).

In the above example, rule φ_3 cannot be applied until we find and apply φ_1 actually. If we check the rules in order $\langle \varphi_4, \varphi_3, \varphi_2, \varphi_1 \rangle$, φ_3 will be checked many times in this situation until it can be applied, which means that a more efficient repair algorithm is needed.

5.2 Fast repairing

We improve the above algorithm from three aspects.

(1) Rule order selection Note that in Algorithm 1, when picking a rule to apply after the tuple has been changed, in the worst case, we need to scan all rules, even if some rules have been checked before. Naturally, we want to avoid checking rules repeatedly in each iteration.

The observation is that applying a rule φ will affect another rule φ' only if φ changes some tuple value that φ' needs to check. More concretely, consider two DRs $\varphi : \mathbf{G}(\mathbf{V}, \mathbf{E})$ where $\mathbf{V} = \mathbf{V_e} \cup \{p, n\}$, and $\varphi' : \mathbf{G}'(\mathbf{V}', \mathbf{E}')$ where $\mathbf{V}' = \mathbf{V_e}' \cup \{p', n'\}$. If $\mathsf{col}(n) \in \mathsf{col}(\mathbf{V_e}')$, i.e., the first rule will change some value of the tuple that can be used as the evidence for the second rule, then φ should be applied before φ'.

Rule graph Based on the above observation, we build a *rule graph* $\mathbf{G^r}(\mathbf{V^r}, \mathbf{E^r})$ for a set Σ of DRs. Each vertex $v_r \in \mathbf{V^r}$ corresponds to a rule in Σ. There is an edge from rule φ to φ' if $\mathsf{col}(p) \in \mathsf{col}(\mathbf{V_e}')$. Note that a cycle may exist, i.e., there might also have an edge from φ' to φ if $\mathsf{col}(p') \in \mathsf{col}(\mathbf{V_e})$.

When repairing a tuple, we follow the topological ordering of the rule graph to check the availability of rules. Note that, if a cycle exists in the rule graph, it is hard to ensure that the rules in the cycle can be checked only once. We first treat the cycle as a single node $\widetilde{v}$ to get the global order of Σ. When checking node $\widetilde{v}$, we first find a rule φ in this cycle that can be applied. Then the edges pointing to φ can be removed.

Example 9 Consider the rules in Fig. 4. There are two connected components $\{\varphi_1, \varphi_2, \varphi_3\}$ and $\{\varphi_4\}$. The first three rules should be checked in the order $\langle \varphi_1, \varphi_2, \varphi_3 \rangle$, since φ_1 may change attribute $\mathsf{Institution}$ that belongs to the evidence nodes of φ_2, and in turn φ_2 may change attribute City that is in the evidence nodes of φ_3. Checking φ_4 is irrelevant of the other rules, and thus it only needs to be checked once.

(2) Efficient instance matching The node in DR provides a similarity function to map values between schema R and KB K. If it is "=" (the cell $t[\mathsf{col}(u)]$ must be equal to an instance with $\mathsf{type}(u)$ in KB), we can just find all instances with $\mathsf{type}(u)$ and use a hash table to check whether $t[\mathsf{col}(u)]$ matches one of them. Otherwise, it is time-consuming to calculate the similarity between $t[\mathsf{col}(u)]$ and each instance. To improve the similarity-based matching, we use a signature-based framework [15,24,37,38,52,53,58]. For each $\mathsf{type}(u)$, we generate signatures for each instance in KB belonging to $\mathsf{type}(u)$. If a cell value in a column $\mathsf{col}(u)$ can match an instance (the similarity is larger than a threshold), they must share a common signature. In other words, for each cell value, we only need to find the instances that share common signatures with the cell. To this end, we build a signature-based inverted index. For each signature, we maintain an inverted list of instances that contain the signature. Given a cell value, the instances on the inverted list of signatures which also belong to the cell value are similarity-based matching candidates. In this way, we do not need to enumerate every instance.

(3) Sharing computations on common nodes between different rules Note that a node can be used in multiple rules and it is expensive to check the node for every rule. To address this issue, we want to check each node only once. Furthermore, we want to build indexes to quickly check that, after a tuple has been updated, which rules are possibly affected. After a rule φ is applied, it marks attributes $\mathsf{col}(\mathbf{V_e} \cup \{p\})$ as positive. The attributes that are marked as positive cannot be changed by any other rules. Hence, after a tuple is updated

Algorithm 2: Fast Repair Algorithm

```
   Input: a tuple $t$, a set $\Sigma$ of consistent DRs, inverted lists $\mathcal{I}$.
   Output: a repaired tuple $t$.
1  sort $\Sigma$ in topological ordering;
2  for each $\varphi \in \Sigma$ do
3      for each vertex or edge $\alpha \in \varphi : \mathbf{G(V,E)}$ do
4          if $t$ matches $\alpha$ then
5              for each $(\varphi', \alpha')$ in $\mathcal{I}(\alpha)$ do
6                  mark the vertex or edge $\alpha' \in \varphi'$ as already
                   checked;
7          else
8              for each $(\varphi', \alpha')$ in $\mathcal{I}(\alpha)$ do
9                  $\Sigma \leftarrow \Sigma \setminus \{\varphi'\}$;
10     if $\varphi$ is applicable to $t$ then
11         update or mark $t$ by $\varphi$;
12         if $t[\mathsf{col}(p)]$ is marked as positive then
13             delete rules in $\Sigma$ that also update $\mathsf{col}(p)$;
14             for each $\alpha \in \{p\} \cup \{e|$edge $e$ connected $p\}$ do
15                 for each $(\varphi', \alpha')$ in $\mathcal{I}(\alpha)$ do
16                     mark the vertex or edge $\alpha' \in \varphi'$ as checked;
17     else
18         $\Sigma \leftarrow \Sigma \setminus \{\varphi\}$;
19 return $t$;
```

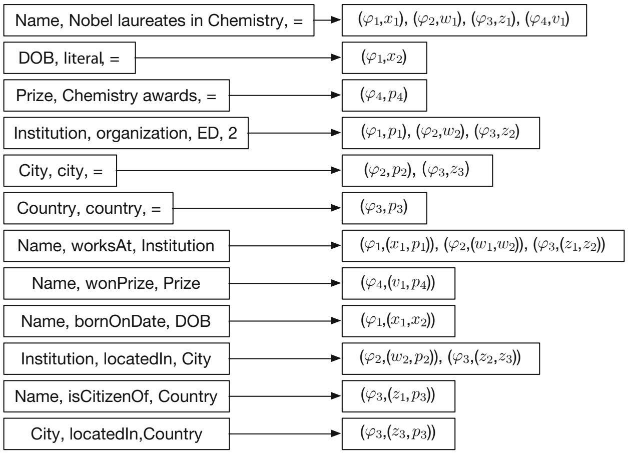
Name, Nobel laureates in Chemistry, =
(φ_1,x_1), (φ_2,w_1), (φ_3,z_1), (φ_4,v_1)
DOB, literal, =
(φ_1,x_2)
Prize, Chemistry awards, =
(φ_4,p_4)
Institution, organization, ED, 2
(φ_1,p_1), (φ_2,w_2), (φ_3,z_2)
City, city, =
(φ_2,p_2), (φ_3,z_3)
Country, country, =
(φ_3,p_3)
Name, worksAt, Institution
(φ_1,(x_1,p_1)), (φ_2,(w_1,w_2)), (φ_3,(z_1,z_2))
Name, wonPrize, Prize
(φ_4,(v_1,p_4))
Name, bornOnDate, DOB
(φ_1,(x_1,x_2))
Institution, locatedIn, City
(φ_2,(w_2,p_2)), (φ_3,(z_2,z_3))
Name, isCitizenOf, Country
(φ_3,(z_1,p_3))
City, locatedIn,Country
(φ_3,(z_3,p_3))

Fig. 6 Rule indexes

by φ, all the rules φ' satisfying $\mathsf{col}(p') \in \mathsf{col}(\mathbf{V_e} \cup \{p\})$ can be safely removed. We utilize inverted lists to track these useless rules and in the meantime, to avoid repeated calculations that are shared between different rules. Similarly we can avoid checking the same relationship in different rules multiple times.

To achieve this, we propose a novel inverted list that can be used interchangeably for both nodes and relationships.

Inverted lists Each *inverted list* is a mapping from a key to a set Ψ of values. Each key is (i) a match between a column in R and a class in KB with similarity function $\approx_u$ or (ii) a relationship/property in KB that describes the relationship between two columns. Each value in Ψ is a pair (φ, α) where $\varphi \in \Sigma$ and α is either a vertex or an edge in $\mathbf{G} \setminus \{n\}$. Each pair in Ψ satisfies that the vertex (or edge) α must contain the node (or relationship) in the key. The inverted lists w.r.t. rules in Fig. 4 are shown in Fig. 6.

We are now ready to present the fast repair algorithm.

Algorithm 2 It first sorts the rules in Σ in topological ordering (line 1) and then checks the rules in turn (lines 2-18). For each rule, every vertex and edge in a DR $\varphi : \mathbf{G}(\mathbf{V}, \mathbf{E})$ will be visited. If it has not been checked, we detect whether $t[\mathsf{col}(u)]$ belongs to $\mathsf{type}(u)$ for vertex u or whether $t[\mathsf{col}(u)]$, $t[\mathsf{col}(v)]$ have relationship $\mathsf{rel}(e)$ for edge e (lines 3-9). If so, we mark this vertex or edge in other rule φ' as already being checked using $\mathcal{I}$ (lines 4-6). Otherwise, we delete φ' from Σ (lines 8-9). We apply rule φ to tuple t if it is applicable (lines 10-16). Otherwise, rule φ is deleted (line 18). If $t[\mathsf{col}(p)]$ is marked as positive, we delete all rules which also update $\mathsf{col}(p)$ from Σ (line 13). Meanwhile, for each rule φ' that also contains p (as evidence node) or the edge connected to p, it also should be marked as already being checked (lines 14-16).

Complexity The complexity of sorting rules is $O(|\Sigma| + |\mathbf{E^r}|)$ where $|\mathbf{E^r}|$ is the number of edges in the rule graph. Note that we only need to sort once and apply to all tuples. It is obvious that the outer loop (lines 2-17) runs at most $|\Sigma|$ times. For each DR $\varphi : \mathbf{G}(\mathbf{V}, \mathbf{E})$, the worst case of checking each vertex and edge needs $O(|C||X||\mathbf{V}| + |\mathbf{E}|)$ as stated above even utilizing the similarity indexes. Thus, the algorithm requires $O(|\Sigma| \times (|C||X||\mathbf{V}| + |\mathbf{E}|))$.

Example 10 Consider tuple t_3 in Table 1, four DRs in Fig. 4, and the invert lists in Fig. 6. The rules will be checked in the order $\langle \varphi_4, \varphi_1, \varphi_2, \varphi_3 \rangle$. Let $\mathsf{checked}_\varphi$ denotes the set storing the vertexes and edges in φ that have been checked to be matched with tuple t.

For rule φ_4, $t_3[\mathsf{Name}]$ can match node v_1. We maintain $\mathsf{checked}_{\varphi_4} = \{v_1\}$. Besides, by utilizing the inverted lists, we have $\mathsf{checked}_{\varphi_1} = \{x_1\}$, $\mathsf{checked}_{\varphi_2} = \{w_1\}$ and $\mathsf{checked}_{\varphi_3} = \{z_1\}$. The negative node n_4 can also be matched, thus $t_3[\mathsf{Prize}]$ will be updated to *Nobel Prize in Chemistry*. Tuple t_3 becomes t'_3(*Roald Hoffmann*$^+$, *1937-07-18, Ukraine, Nobel Prize in Chemistry*$^+$, *Columbia University, Ithaca*).

Then for rule φ_1, since t'_3 can match nodes x_1, x_2, n_1 and edges (x_1, x_2),(x_1, n_1), $t'_3[\mathsf{Institution}]$ will be modified to *Cornell University* and we have t''_3(*Roald Hoffmann*$^+$, *1937-07-18*$^+$, *Ukraine, Nobel Prize in Chemistry*$^+$, *Cornell University*$^+$, *Ithaca*). We can expand $\mathsf{checked}_{\varphi_2} = \{w_1, w_2, (w_1, w_2)\}$ and $\mathsf{checked}_{\varphi_3} = \{z_1, z_2, (z_1, z_2)\}$ by utilizing the inverted lists.

When considering rule φ_2, we only need to check the negative and positive nodes based on $\mathsf{checked}_{\varphi_2}$. As

Algorithm 3: Multiple Repair Algorithm

```
   Input: a tuple $t$, consistent $\Sigma$ in topological ordering, inverted
          lists $\mathcal{I}$.
   Output: a set $S$ of repaired tuples.
1  if $\Sigma$ is empty then $S \leftarrow S \cup \{t\}$; return
2  $\varphi \leftarrow$ the first DR in $\Sigma$;
3  for each vertex or edge $\alpha \in \varphi : \mathbf{G(V,E)}$ do
4      if $t$ matches $\alpha$ then
5          mark the vertexes or edges as already checked;
6      else
7          $\Sigma \leftarrow$ delete the rules in $\Sigma$ that contains $\alpha$;
8  if $\varphi$ is applicable to $t$ then
9      $T \leftarrow$ update or mark $t$ by $\varphi$;
10     if $t[\mathsf{col}(p)]$ is marked as positive then
11         $\Sigma' \leftarrow$ delete rules in $\Sigma$ that update $\mathsf{col}(p)$;
12         for each $\alpha \in \{p\} \cup \{e|edge\ e\ connected\ p\}$ do
13             mark the vertexes or edges as checked;
14     for each $t' \in T$ do
15         MultipleRepairAlg($t'$, $\Sigma'$, $\mathcal{I}$);
16 else
17     $\Sigma' \leftarrow \Sigma \setminus \{\varphi\}$;
18     MultipleRepairAlg($t$, $\Sigma'$, $\mathcal{I}$);
19 put the deleted rules back to $\Sigma$;
20 delete the marks on vertexes or edges of DRs;
```

t_3''[Institution] is actually located in t_3''[City]. We have t_3'''(*Roald Hoffmann*$^+$, *1937-07-18*$^+$, *Ukraine, Nobel Prize in Chemistry*$^+$, *Cornell University*$^+$, *Ithaca*$^+$). Meanwhile, $\mathsf{checked}_{\varphi_3} = \{z_1, z_2, z_3, (z_1, z_2), (z_2, z_3)\}$

Based on $\mathsf{checked}_{\varphi_3}$, we only need to examine the negative and positive nodes. Since tuple t_3''' matches node n_3 and the relationship (z_1, n_3), t_3'''[Country] will be updated to *United States* by applying φ_3. Tuple t_3''''(*Roald Hoffmann*$^+$, *1937-07-18*$^+$, *United States*$^+$, *Nobel Prize in Chemistry*$^+$, *Cornell University*$^+$, *Ithaca*$^+$) is a fixpoint.

5.3 Multiple-version repairs

We can extend our methods to support multiple-version repairs. Different from single-version repair, there might exist multiple ways to modify an error. Thus, instead of having only one updated tuple, we need to keep multiple repaired results based on backtracking techniques. The property of consistency still holds, which means that, all the possible orders of repairs via Σ terminate in the same fixpoints.

Algorithm 3 At first, we need to check whether Σ is empty, which means that whether we have scanned all DRs. If $\Sigma = \emptyset$, t will be added into S and this procedure can be returned (line 1). If Σ is not empty, we take the first DR φ out of Σ (line 2) and check whether φ can be applied to tuple t (lines 3-7). Every vertex and edge in $\varphi : \mathbf{G}(\mathbf{V}, \mathbf{E})$ will be visited, which is the same as our fast repair algorithm. We apply rule φ to t if it is applicable (lines 8-15). Otherwise, we delete φ from Σ (line 17) and call this multiple repair algorithm again to go over the remaining DRs (line 18). Applying rule φ to t may generate multiple-version repairs, and we use a set T to keep all intermediate results (line 9). If $t[\mathsf{col}(p)]$ is marked as positive, we delete all rules which also update $\mathsf{col}(p)$ from Σ (line 11). Meanwhile, for other rules that also contains p (as evidence node) or have edges connected to p, they also should be marked as already being checked (lines 12-13). Then, for each tuple $t' \in T$, this multiple repair algorithm will be executed again for further repair (lines 14-15). In the end, we put the deleted rules back to Σ and delete the marks on DRs (lines 19-20), so that the generation of one version repair will not affect another.

Complexity Let $|S|$ denote the number of possible repairs. The fast repair algorithm for multiple repairs runs in $O(|S||\Sigma| \times (|C||X||\mathbf{V}| + |\mathbf{E}|))$ times.

Example 11 Consider tuple t_4 in Table 1, four DRs in Fig. 4. The rules will be checked in the order $\langle \varphi_4, \varphi_1, \varphi_2, \varphi_3 \rangle$.

Rule φ_4 will not repair any value but mark t_4[Name] and t_4[Prize] as positive. As for rule φ_1, tuple t_4 matches nodes x_1, x_2, n_1 and edges (x_1, x_2), (x_1, n_1), so φ_1 can be applied. From the KB, we know that *Melvin Calvin* worked in two institutions *{UC Berkeley, University of Manchester}*. Thus, t_4 can be repaired in two ways: t_4'(*Melvin Calvin*$^+$, *1911-04-08*$^+$, *United States, Nobel Prize in Chemistry*$^+$, *UC Berkeley*$^+$, *St. Paul*) and t_4''(*Melvin Calvin*$^+$, *1911-04-08*$^+$, *United States, Nobel Prize in Chemistry*$^+$, *University of Manchester*$^+$, *St. Paul*). At this time, $T = \{t_4', t_4''\}$ and $\Sigma' = \{\varphi_2, \varphi_3\}$.

As for t_4', rule φ_2 will update t_4'[City] to *Berkeley* from *St. Paul* and φ_3 marks t_4''[Country] as positive. All four rules have been checked ($\Sigma = \emptyset$), and t_4'''(*Melvin Calvin*$^+$, *1911-04-08*$^+$, *United States*$^+$, *Nobel Prize in Chemistry*$^+$, *UC Berkeley*$^+$, *Berkeley*$^+$) is a fixpoint. Thus, we get one result $S = \{t_4'''\}$.

Then t_4'' will be taken out from T for further repair and rule φ_2, φ_3 need to be checked again. Rule φ_2 will update t_4''[City] to *Manchester* from *St. Paul* and φ_3 marks t_4''[Country] as positive. t_4''''(*Melvin Calvin*$^+$, *1911-04-08*$^+$, *United States*$^+$, *Nobel Prize in Chemistry*$^+$, *University of Manchester*$^+$, *Manchester*$^+$) is another fixpoint. Finally, we have two valid repair $S = \{t''' S_4, t_4''''\}$.

6 Rule generation from examples

Our next goal is to study methods for generating detective rules. Actually, it is rather hard to generate high-quality DRs because (1) the knowledge bases are rather large and there are many ways to build connections between a table and a KB; (2) it is hard to decide which attributes act as evidence nodes and positive/negative node. Usually, the user has to be involved to identify the validity of DRs before they could be applied. Our aim is to make the user's life easier, by automatically computing a set of DRs to be verified. We first describe how to generate candidate DRs from positive/negative examples (Sect. 6.1). We then discuss the refinement of detective rules (Sect. 6.2).

6.1 Generating candidate DRs by examples

We propose to *generate rules by examples*. Let D be a table of relation R and K a KB. We only discuss how to generate rules for one attribute $A \in R$, and the rules for the other attributes can be generated similarly. Let P be a set of positive tuple examples, i.e., all values are correct. Let N be a set of negative examples, where only A-attribute values (i.e., values in attribute A) are wrong.

Candidate DR generation algorithm We describe our algorithm below.

S1 [Schema-level matching graphs for P.] We use an existing solution [11] to compute a set $\mathcal{G}^+$ of schema-level matching graphs using the KB K, for the positive examples P. These correspond to the positive semantics of the table.

In a nutshell, given a set of correct tuples, the algorithm will map tuple values to KB instances to find their types and relationships. For instance, given two tuples with correct values as *t(China, Beijing)* and *t′(Japan, Tokyo)*, the algorithm can find out that the first (resp. second) column has class country (resp. city) and their relationship is country hasCaptial city.

Then, we can ask experts to select the best schema-level matching graph $\mathbf{G}^+$ from $\mathcal{G}^+$, which is the best understanding of the semantics of R.

S2 [Schema-level matching graphs for N.] We also compute a set $\mathcal{G}^-$ of schema-level matching graphs for the negative examples N. As for N, the schema-level matching subgraph of attributes $R \setminus A$ must be the same as $\mathbf{G}^+$. What we need to do is just decide the negative semantics of attribute A.

Given a negative example $t \in N$, we can map $t[A]$ to several instances in the KB K whose class can then be extracted, and the relationship between $t[A]$ and $t[A']$ ($A' \in R \setminus A$) from K can also be retrieved. Since not all combinations of $\mathsf{type}(t[A])$ and $\mathsf{rel}(t[A], t[A'])$ make sense in practice, we need a scoring function to rank them for human validation. For instance, given a tuple with wrong value as t(*China*, *Shanghai*). We have known that the first column has class country from the above step. Meanwhile, we find that the second column has two classes city and film in K, and their relationship is locatedIn. Obviously, city is more suitable for the relationship locatedIn, so city as a candidate class of attribute A should get higher score than film. Now, it is time to introduce our scoring model.

Note that we have confirmed the corresponding class of $t[A']$ in the KB K, i.e., $\mathcal{P}(\mathsf{type}(t[A'])\text{=}c') = 1$. The score $\mathcal{P}(\mathsf{rel}(t[A], t[A']) = r)$ is:

$$\begin{aligned}
&\mathcal{P}\big(\mathsf{rel}(t[A], t[A']) = r|\mathsf{type}(t[A']) = c'\big)\\
&\quad= \frac{\mathcal{P}\big(\mathsf{rel}(t[A], t[A']) = r \cap \mathsf{type}(t[A']) = c'\big)}{\mathcal{P}(\mathsf{type}(t[A']) = c')}\\
&\quad= \frac{|Ins(r) \cap Ins(c')|}{|Ins(c')|}
\end{aligned}$$

where $Ins(c')$ is the set of instances in K which belong to class c', and $Ins(r)$ is the set of instances that appear in the subject (resp. object) of relationship r in K if $t[A']$ is the subject (resp. object) of r.

The score of $\mathsf{type}(A)$ is:

$$\begin{aligned}
&\mathcal{P}(\mathsf{type}(t[A]) = c|\mathsf{rel}(t[A], t[A']) = r)\\
&\quad= \frac{\mathcal{P}(\mathsf{rel}(t[A], t[A']) = r \cap \mathsf{type}(t[A]) = c)}{\mathcal{P}(\mathsf{rel}(t[A], t[A']) = r)}\\
&\quad= \frac{|Ins(c) \cap Ins(r)|}{|Ins(c) \times \mathcal{P}(\mathsf{rel}(t[A], t[A']) = r)|}
\end{aligned}$$

Each relationship between A and $A' \in R \setminus A$ will be bounded with the most compatible type of attribute A. Let $\mathbf{R}_c$ denotes the set of relationships bounded with class c, then $\mathbf{R}_c$ will be used to construct the schema-level matching graph $\mathbf{G}^-$ with c. The score of $\mathbf{G}^-$ is the score of type c plus the sum score of $\mathbf{R}_c$:

$$\begin{aligned}
\mathcal{P}(\mathbf{G}^-) &= \sum_{r \in \mathbf{R}_c} \mathcal{P}(\mathsf{type}(t[A]) = c|\mathsf{rel}(t[A], t[A']) = r)\\
&\quad+ \sum_{r \in \mathbf{R}_c} \mathcal{P}(\mathsf{rel}(t[A], t[A']) = r)
\end{aligned}$$

Moreover, when different negative examples have the same schema-level matching graph $\mathbf{G}^-$, we will add up as the score of $\mathbf{G}^-$.

S3 [Candidate DR Generation.] For each graph $\mathbf{G}^- \in \mathcal{G}^-$, $\mathbf{G}^+$ and $\mathbf{G}^-$ have only one different node, that is, $p \in \mathbf{G}^+$ and $n \in \mathbf{G}^-$ are different and the two graphs $\mathbf{G}^+ \setminus \{p\}$ and $\mathbf{G}^- \setminus \{n\}$ are isomorphic. We merge $\mathbf{G}^+$ and $\mathbf{G}^-$ as one DR, where p (resp. n) becomes the positive (resp. negative) node of the generated DR, and the score of this DR is the same with the score of $\mathbf{G}^-$, e.g., one can merge the outputs from **S1** and **S2**, the positive semantics country hasCaptial city and the negative semantics city locatedIn country, to generate one DR.

The above process will generate a set of candidate DRs. Indeed, the number is not large so the user can manually pick. Our claim is that, compared with asking the user to do the eyeballing exercise to write DRs manually, the above algorithm is simple but useful in practice.

Example 12 Suppose that we have known the correct values of the tuples in Table 1, then these repaired tuples t_1 to t_4 can

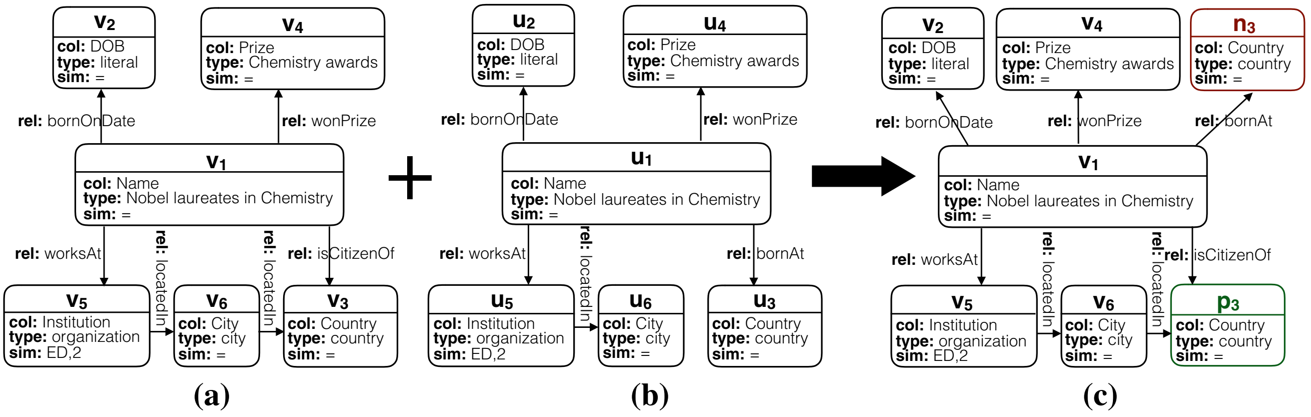

Fig. 7 Rule generation from example. **a** G$^+$, **b** G$^-$ and **c** candidate DR

be used as positive examples P. And the best schema-level matching graph $\mathbf{G}^+$ for P is shown in Fig. 7a. Tuple t_3 can generate one negative example t'_3(*Roald Hoffmann, 1937-07-18, Ukraine, Nobel Prize in Chemistry, Cornell University, Ithaca*). From KB, we find out that the relationship between *Roald Hoffmann* and *Ukraine* is bornAt, but *Ukraine* has 8 classes such as country, member states of United Nations and Ukrainian_speaking countries. country is more suitable for relationship bornAt with the instances belong to Nobel laureates in Chemistry. Thus, the schema-level matching graph $\mathbf{G}^-$ for t'_3 which has the highest score is shown in Fig. 7b. $\mathbf{G}^+$ and $\mathbf{G}^-$ can be merged to be one candidate DR which is shown in Fig. 7c.

6.2 Detective rule refinement

The candidate DRs generated following Sect. 6.1 can be applied to modify the tuple of which only one attribute is wrong, since other attributes are all utilized in evidence nodes to detect the error. When there are more than one incorrect attributes in a tuple, no candidate DRs can be applied. Hence, what we should do next is to refine candidate DRs, i.e., pick out the indispensable evidence nodes and remove unnecessary ones from candidate DRs.

Intuitively, all evidence nodes in DR φ which connect directly with the positive node and negative node may ought to be reserved to detect error and guide repair, as well as the relationships between them. On the contrary, other evidence nodes in DR which have no relationship with positive and negative node are not so important for repair and can be removed. Let V_e^p, V_e^n denote the sets of these evidence nodes that connect with positive node and negative node, respectively, and $\hat{\mathbf{V}}_e = V_e^p \cup V_e^n$. Then, we focus on the following two questions.

(1) Whether all evidence nodes in $\hat{\mathbf{V}}_e$ *should be reserved?* If the positive/negative node only connects with one evidence node ($|V_e^p| = 1/|V_e^n| = 1$), there is no doubt that this node is essential. If the positive/negative node links with more than one evidence nodes, we need to make sure that V_e^p and V_e^n are *minimal* which means that no subset of V_e^p and V_e^n can determine how to repair the error exactly. Therefore, in each step, we try to remove one evidence node u from V_e^p and check whether in KB the positive value determined by $V_e^p \setminus \{u\}$ is always equal to the value determined by V_e^p. If so, u can be removed from V_e^p and we continue to check whether $V_e^p \setminus \{u\}$ is a minimal set. This strategy can also be applied to V_e^n.

(2) Whether all evidence nodes in $\mathbf{V_e} \setminus \hat{\mathbf{V}}_e$ *can be removed?* In general, the detective rule which has $\hat{\mathbf{V}}_e$ as evidence nodes can be applied to repair the tuple. However, if errors exist in $\mathsf{col}(\hat{\mathbf{V}}_e)$, this DR may inject error into $\mathsf{col}(p)$ and meanwhile mark $\mathsf{col}(\hat{\mathbf{V}}_e \cup \{p\})$ as positive with the result that other DRs also cannot revise the mistakes. For example, if we apply the refined DR which is in Fig. 8b to tuple t_4 in Table 1, this rule will mark t_4[Name], t_4[City] and t_4[Country] as positive. Then, t_4[City] cannot be modified to *Berkeley* ever. Therefore, we need to make sure that a DR cannot be applied when $\mathsf{col}(\hat{\mathbf{V}}_e)$ have errorous. To this end, suppose that there is another DR φ' that $\mathsf{col}(p') \in \mathsf{col}(V_e^p)$, we can merge the nodes in $V_e^{p'}$ to DR φ as evidences. Then, rule φ can be applied only after $\mathsf{col}(p')$ satisfy the positive semantic.

Example 13 Consider the candidate DR in Fig. 8a, $V_e^{p_3} = \{v_1, v_6\}$ and $V_e^{n_3} = \{v_1\}$. Since $|V_e^{n_3}| = 1$, node v_1 is indispensable. We only need to make sure that $V_e^{p_3}$ is minimal. We find that in KB the country where one works is not always his mother country and vice versa. That is to say, only v_1 or only v_6 can not decide how to repair Country. Thus, $V_e^{p_3}$ has been minimal, and the refined DR is shown in Fig. 8b.

Then, let us consider rule φ_2 in Fig. 4 that $\mathsf{col}(p_2) \in \mathsf{col}(V_e^{p_3})$. Since $V_e^{p_2} = \{w_2\}$, we merge w_2 to the DR in Fig. 8b and complement the relationships about w_2. The final detective rule is shown in Fig. 8c.

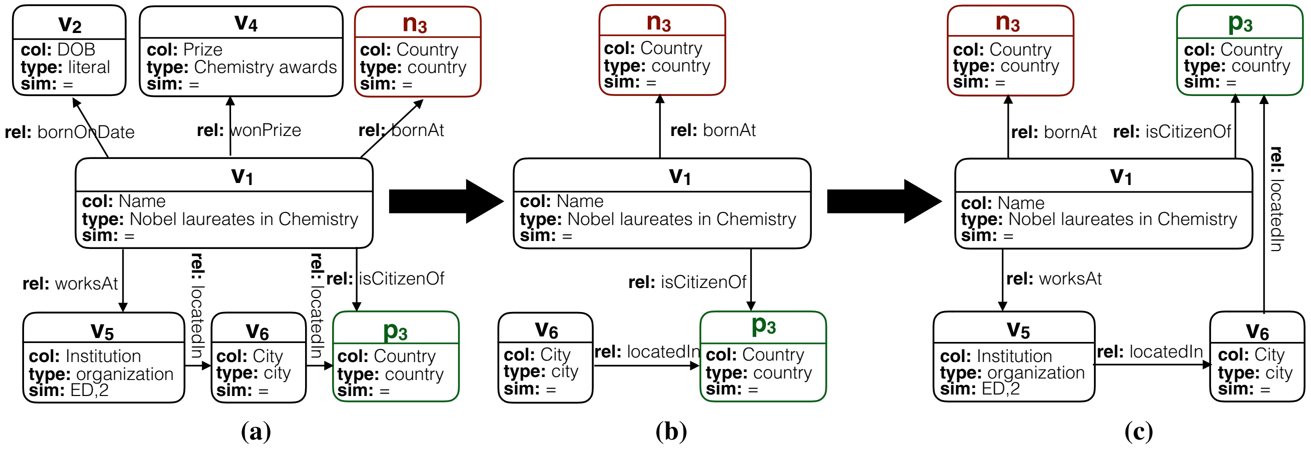

Fig. 8 Rule refinement. **a** candidate DR, **b** DR with $\hat{V}_e$ and **c** DR

7 Experimental study

7.1 Experimental setup

Datasets We used two sets of small Web tables, a real-world dataset and a synthetic dataset. Note that we used the datasets that are covered by general purpose KBs.

(1) WWT. This dataset contains 37 Web tables[2], with the average number of tuples 44. Each table has 3 attributes on average.
(2) WEX. We chose 20 Web tables from original WEX dataset[3], which have lots of attributes covered by KBs. Each table has 184 tuples on average, with the average number of attributes 6.
(3) Nobel. It contains 1069 tuples about Nobel laureates, obtained by joining two tables from Wikipedia: List of Nobel laureates by country[4] and List of countries by Nobel laureates per capita[5]. We tested this case to see how our approach performs for personal information, an important topic considered in many applications.
(4) UIS. It is a synthetic dataset generated by the UIS Database Generator[6]. We generated 100K tuples.

Knowledge bases We used Yago [30] and DBpedia [41] for our experiments. It is known that both Yago and DBpedia share general information of generating a structured ontology. However, the difference is that Yago focuses more on the taxonomic structure, e.g., richer type/relationship hierarchies. This indeed makes the experiment more interesting to see how taxonomic structure plays the role for mapping the information between relations and KBs. The number of aligned classes and relations of testing datasets are given in Table 2. We retrieved the classes and relationships by issuing SPARQL queries to a KB.

Table 2 Datasets (aligned classes and relations)

	Yago		DBPedia	
	#-class	#-relationship	#-class	#-relationship
WWT	42	30	51	30
WEX	67	45	78	39
Nobel	5	4	5	4
UIS	5	5	5	4

Noise We did not inject noise to Web tables because they are dirty originally. Noises injected to Nobel and UIS have two types: (i) typos; (ii) semantic errors: The value is replaced with a different one from a semantically related attribute. Errors were produced by adding noises with a certain rate e%, i.e., the percentage of dirty cells over all data cells.

Detective rules The DRs were generated as described in Sect. 6, verified by experts. For WWT and WEX, we totally generated 50 DRs and 45 DRs, respectively. For Nobel and UIS, we generated 5 DRs for each table.

Algorithms We implemented the following algorithms: (i) bRepair: the basic repair algorithm (Sect. 5.1); (ii) fRepair: the fast repair algorithm (Sect. 5.2); (iii) rule generation algorithm (Sect. 6) to generate candidateDRs and refinedDRs. For comparison, we have implemented KATARA [11], which is also a KB-based data cleaning system. We also compared with two IC-based repairing algorithms: Llunatic [25] and constant CFDs [21].

Measuring quality We used precision, recall and F-measure to evaluate the repairing quality: precision is the ratio of

[2] https://www.cse.iitb.ac.in/~sunita/wwt/

[3] http://wiki.freebase.com/wiki/WEX

[4] https://en.wikipedia.org/wiki/List_of_Nobel_laureates_by_country

[5] https://en.wikipedia.org/wiki/List_of_countries_by_Nobel_laureates_per_capita

[6] http://sherlock.ics.uci.edu/data.html

correctly repaired attribute values to the number of all the repaired attributes; and recall is the ratio of correctly repaired attribute values to the number of all erroneous values; and F-measure is the harmonic mean of precision and recall. We manually repaired Web tables and regarded them as ground truth. There are few cases that multi-version repairs appear in our experiments. In this case, if one of the repairs matches the ground truth value, we treat it as a correct repair. Besides, knowledge bases cannot cover the whole tables. For the other tables, we mainly evaluated the tuples whose value in *key* attribute (e.g., Name w.r.t. Nobel or State w.r.t. UIS) have corresponding entities in KBs.

Experimental environment All methods were written in Java, and all tests were conducted on a PC with a 2.40GHz Intel CPU and 64GB RAM.

7.2 Experimental results

We tested DRs from four aspects. Exp-1: The comparison with other KB-based data cleaning methods. Exp-2: The comparison with IC-based cleaning on tables. Exp-3: Effectiveness of detective rule generation. Exp-4: Efficiency and scalability of our solutions.

Exp-1: comparison with KB powered data cleaning We compared with KATARA [11], the most recent data cleaning system that is powered by KBs and experts in crowdsourcing. Although KATARA can mark data as correct, it cannot automatically detect or repair errors. In fact, KATARA relies on experts to manually detect and repair errors.

In order to have a fair comparison by removing the expert sourcing factor, we revised KATARA by simulating expert behavior as follows. When there was a full match of a tuple and the KB under the table pattern defined by KATARA, the whole tuple was marked as correct. When there was a partial match, we revised KATARA by marking the minimally unmatched attributes as wrong. For repairing, since KATARA also computes candidate repairs, as presented in [11], we picked the one from all candidates that minimizes the repair cost.

(A) Data repair We first compared with KATARA about data repairing accuracy, using the datasets reported in Table 2. For Nobel and UIS, the error rate was 10%.

Precision Table 3 shows the results of applying DRs and using KATARA for data repairing. DRs were carefully designed to ensure trusted repair. Hence, not surprisedly, the precision was always 1. This is always true if the DRs are correct. KATARA, on the other hand, relies on experts to make decisions. Once experts are absent, KATARA itself cannot decide how to repair, which result in relatively low precision as reported in the table.

Recall As shown in Table 3, for Web tables, DRs had lower recall than KATARA. It is because some of Web tables have few number of attributes. This is not enough to support the modifications of DRs. For example, considering the schema (Author, Book), when t[Author] and t[Book] do not satisfy the relationship wrote, it is hard to judge that which attribute is wrong. So our methods would not repair this kind of tables, in a conservative way. Meanwhile, since DRs depend on some attributes as evidences to ensure trusted repair, if those attribute values in a tuple are not completely covered by KBs, DRs would still not make any action. It is noteworthy that DRs can achieve similar recall in WEX with KATARA. This is because more evidences that covered by KBs can be used to indicate which attribute is wrong. For Nobel and UIS, the recall of our algorithms were higher than KATARA. The reason is that KATARA does not support fuzzy matching. In order to find proper modifications, at least one attribute must be correct. Meanwhile, semantic errors would mislead KATARA to repair a tuple.

F-measure Our method had higher F-measure than KATARA, because our method had higher precision and sim-

Table 3 Data annotation and repair accuracy

	Precision	Recall	F-measure	#-POS
WWT				
DRs				
Yago	1	0.38	0.55	1469
DBpedia	1	0.43	0.60	1326
KATARA				
Yago	0.73	0.40	0.52	864
DBpedia	0.78	0.46	0.58	752
WEX				
DRs				
Yago	1	0.49	0.66	4857
DBpedia	1	0.42	0.59	3924
KATARA				
Yago	0.52	0.50	0.51	2972
DBpedia	0.54	0.42	0.47	2105
Nobel				
DRs				
Yago	1	0.70	0.82	1543
DBpedia	1	0.54	0.70	715
KATARA				
Yago	0.74	0.68	0.71	396
DBpedia	0.64	0.49	0.56	189
UIS				
DRs				
Yago	1	0.73	0.84	77001
DBpedia	1	0.63	0.77	57703
KATARA				
Yago	0.67	0.77	0.72	35084
DBpedia	0.63	0.57	0.60	25152

ilar recall. Taken the explanations above, it is easy to see that DRs had comparable F-measure with KATARA for WWT, and better F-measure in WEX, Nobel and UIS, as shown in Table 3.

(B) Data annotation KATARA can also mark correct data, when a given tuple can find a full match in the given KB, relative to the table pattern they used. Note that in their paper, KATARA can mark wrong data. However, each wrong value has to be manually verified by experts. For comparison, given a tuple and a KB, if a partial match is found by KATARA, one way is to heuristically mark the matched part as correct and the unmatched part as wrong. This will cause both false positives and false negatives. Hence, in order to have a relatively fair comparison, we favor KATARA by only checking the full matches that they mark as correct.

Table 3 gives the results of both DRs and KATARA in marking data, listed in the last column #-POS. The results show that, even by ignoring the ability of data repairing, DRs can automatically mark much more positive data than KATARA. These information is extremely important for both heuristic and probabilistic methods, since the main reason they make false positives and false negatives is that they cannot precisely guess which data are correct or wrong.

Exp-2: comparison with IC-based repair In this group of experimental study, we compared with IC-based repairing algorithms. Llunatic [25] involves different kinds of ICs and different strategies to select preferred values. For Llunatic, we used FDs and chose its frequency cost-manager. *Metric* 0.5 was used to measure the repair quality (for each cell repaired to a variable, it was counted as a partially correct change). For constant CFDs, they were generated from ground truth. We simulated the user behavior by repairing the right hand side of a tuple t based on a constant CFD, if the left side values of t were the same as the values in the given constant CFD. In this case, constant CFDs will make mistakes if the tuple's left hand side values are wrong.

Also, since there is not much redundancy in the Web tables that IC rely on to find errors (or violations), we tested using only Nobel and UIS datasets. We first evaluated the accuracy of repair algorithm over different error rates. We then varied the percentage of error types in Nobel and UIS to get better insight into the strong and weak points of DRs, compared with other IC-based approaches.

(A) Varying error rate For Nobel and UIS, we studied the accuracy of our repair algorithm by varying the error rate from 4 to 20%, and reported the precision, recall and F-measure in Fig. 9. The rates of different error types, i.e., typos and semantic errors, were equal, i.e., 50–50.

We can see that our methods had stable performance when error rates increased. However, the precision and recall of Llunatic moderately decreased, since when more errors were injected, it became harder to detect errors and link relevant

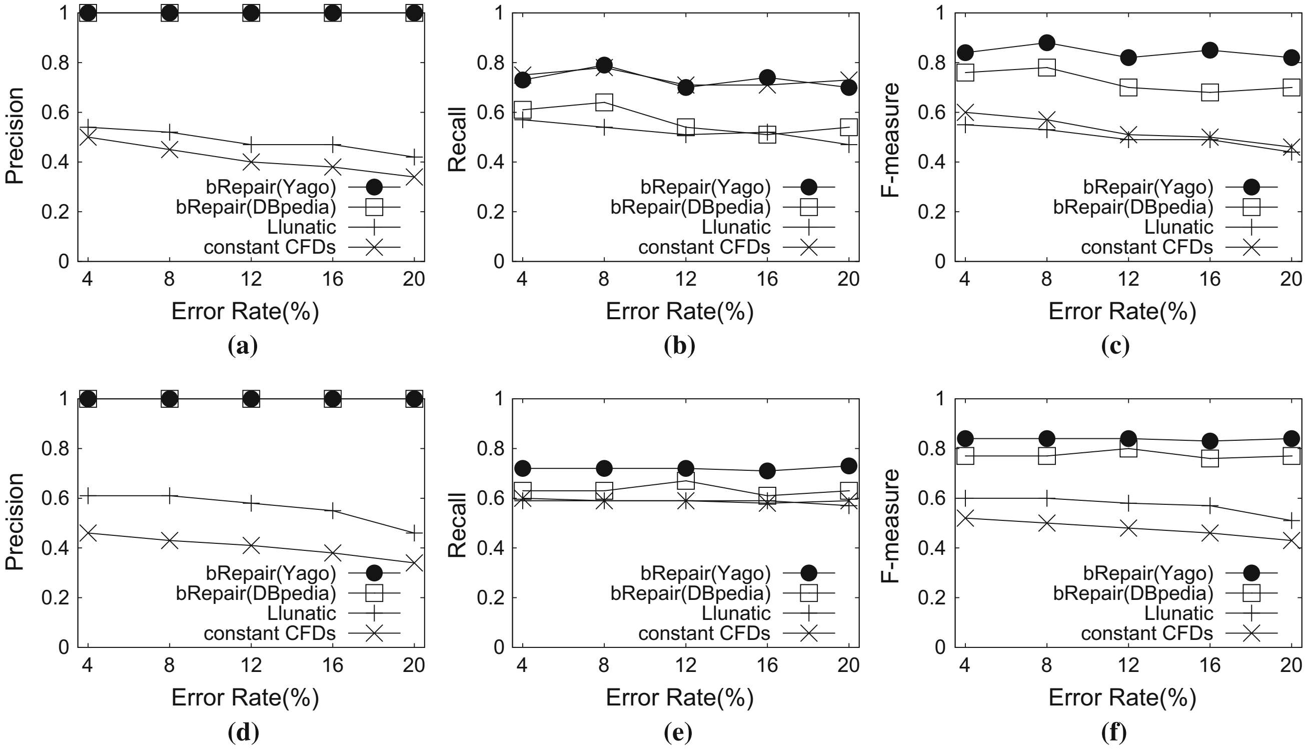

Fig. 9 Effectiveness (varying error rate). **a** Precision (Nobel), **b** recall (Nobel), **c** F-measure (Nobel), **d** precision (UIS), **e** recall (UIS) and **f** F-measure (UIS)

tuples for heuristic repair algorithm. The precision and recall of constant CFDs also decreased because there were more chances that errors happened on the left hand side of constant CFDs.

From Fig. 9b, we see that our algorithms did not have higher recall. This is because: (i) KBs cannot cover all attribute values in Nobel and UIS, e.g., some City can find corresponding resource with property locatedIn to repair the attribute State but some cannot. Thus, some errors cannot be detected; (ii) to ensure the precision, we would not repair errors when the evidence was not sufficient; and (iii) if semantic errors were injected into the evidence nodes of DRs, we cannot detect and repair them. On the contrary, the other two methods would repair some potentially erroneous heuristically, which may increase their recall.

(B) Varying typo rate We fixed the error rate at 10% and varied the percentages of typos from 0 to 100% (semantic errors from 100 to 0% corresponding) to evaluate the ability of capture typos and the semantic error. The experimental results are shown in Fig. 10.

Figure 10 shows that Llunatic and our methods behaved better with typos than with semantic error. The reason is that they all chose to repair an error to the most similar candidate, which for typos are more likely to be correct value. On the contrary, if the semantic errors were added to the evidence nodes of DRs or left hand of FDs, none of us can detect that errors. Meanwhile, more *lluns* (unknown defined in Llunatic) were introduced to make the table consistent. Constant CFDs do not support fuzzy matching. Thus, it is hard to say which type of errors it can detect better. These errors can be detected only when they are injected to the right hand of constraints.

Exp-3: effectiveness of detective rule generation We also evaluated the effectiveness of detective rule generation described in Sect. 6, including candidate rule generation and rule refinement. For Nobel and UIS, we generated 5 candidate DRs and 5 refined DRs for each table. We varied the error rate from 10 to 50%, and reported the precision, recall and F-measure in Fig. 11. The rates of different error types, i.e., typos and semantic errors, were equal, i.e., 50–50.

From Fig. 11, we can see that candidate DRs can achieve good precision and recall already. However, along with the increase in error rate, the recall of candidate DRs decreased. On the contrary, refined DRs can have more stable recall without losing any precision, and certainly higher F-measure. The reason is that, to repair an error in one attribute, candidate DRs need all other attributes as evidences to detect the error. As error rate increased, it was more likely that a tuple had more than one incorrect attributes. Candidate DRs cannot repair these errors in this case. Refined DRs only contain essential evidence nodes, so they can repair more errors. Furthermore, KBs cannot cover all attributes of a tuple sometimes. In other words, candidate DRs was not easy to be

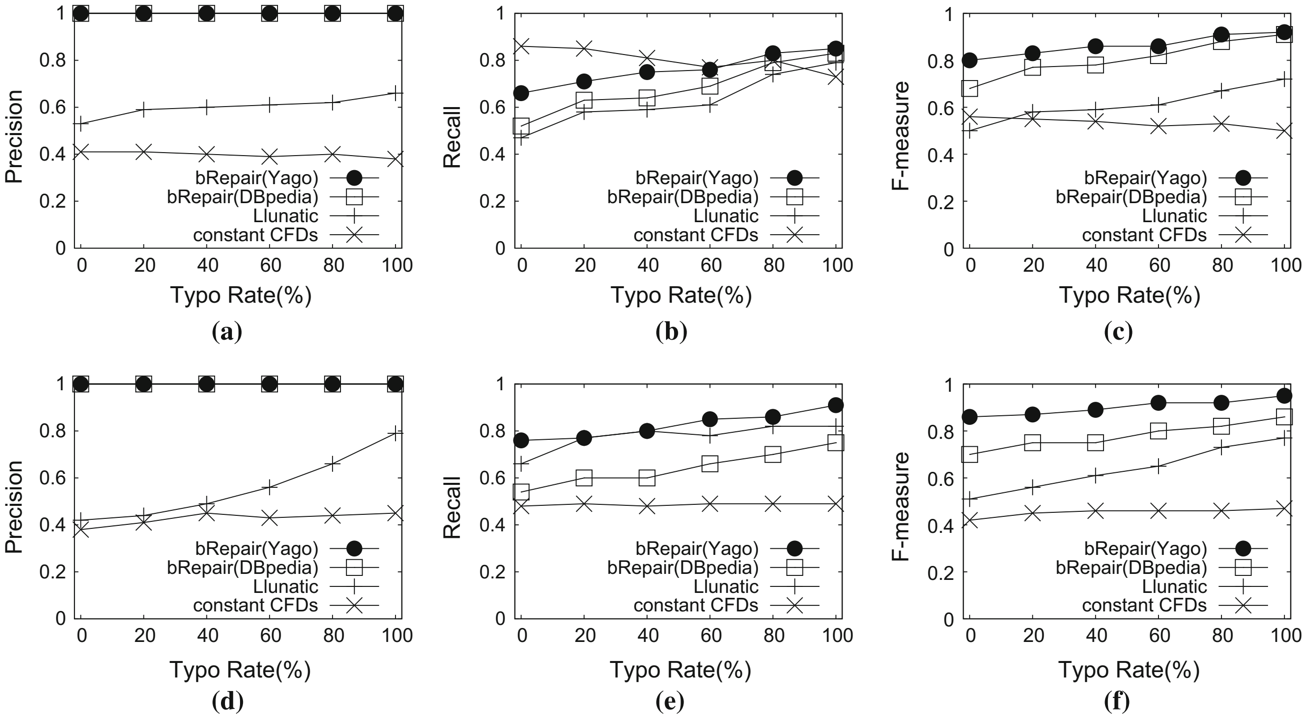

Fig. 10 Effectiveness (varying typo rate). **a** Precision (Nobel), **b** recall (Nobel), **c** F-measure (Nobel), **d** precision (UIS), **e** recall (UIS) and **f** F-measure (UIS)

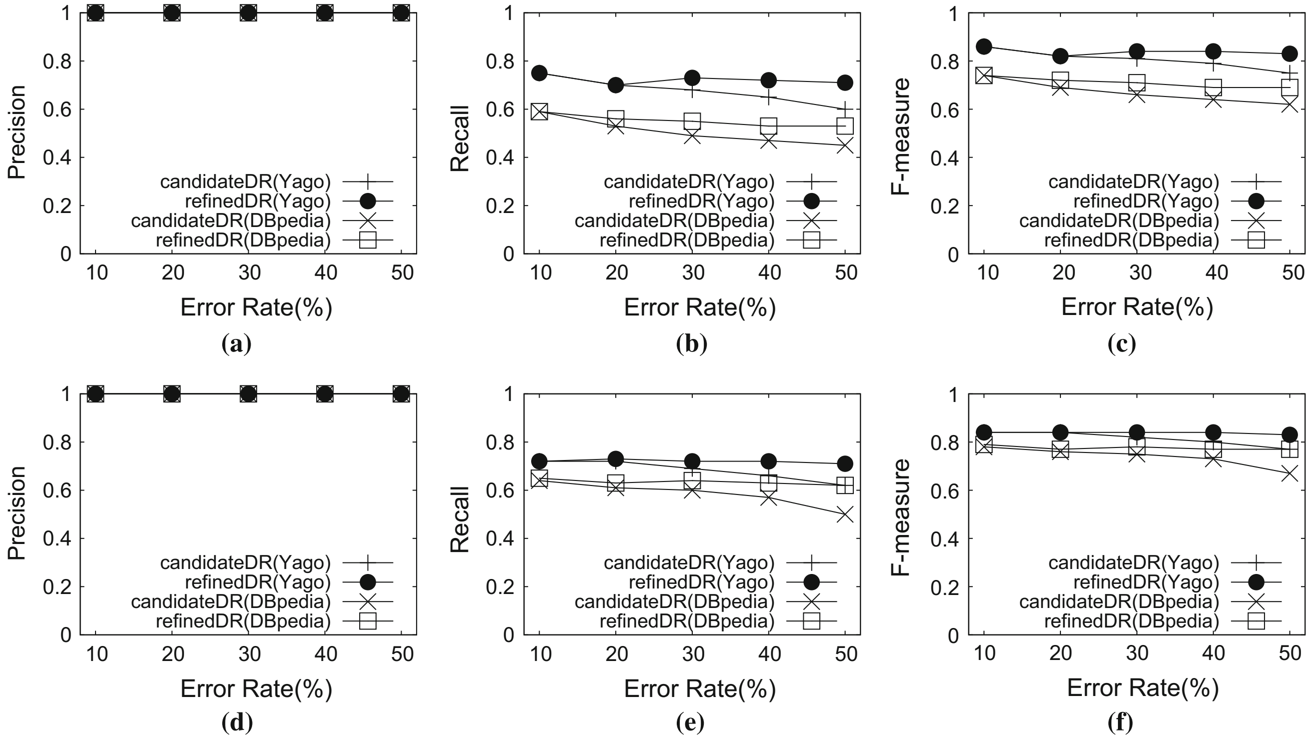

Fig. 11 Effectiveness of detective rule generation. **a** Precision (Nobel), **b** recall (Nobel), **c** F-measure (Nobel), **d** precision (UIS), **e** Recall (UIS) and **f** F-measure (UIS)

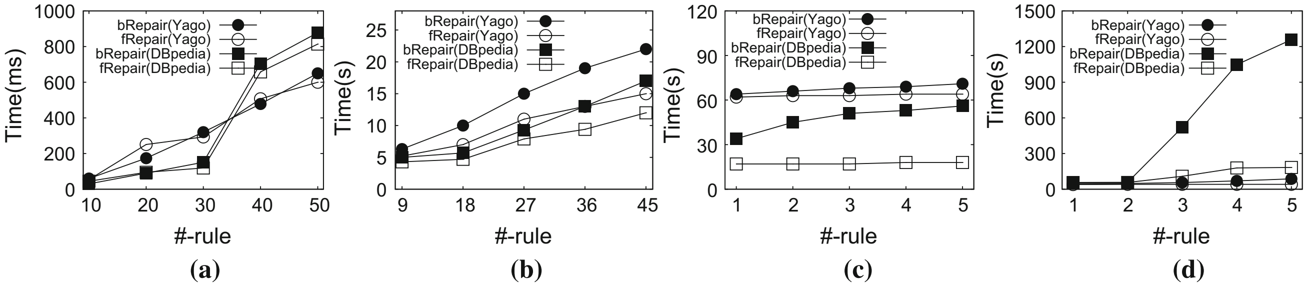

Fig. 12 Efficiency (varying #-rule). **a** Time (WWT), **b** time (WEX), **c** time (Nobel) and **d** time (UIS)

satisfied. That is why refined DRs was better than candidate DRs even when error rate was low.

Exp-4: efficiency study We evaluated the efficiency of our repair algorithms using WWT, WEX, Nobel and UIS. We first varied the number of DRs to measure the performance of bRepair and fRepair. Then we studied the scalability of the algorithms utilizing the UIS Database Generator.

(A) Varying #-Rule We varied the number of rules from 10 to 50 by a step of 10 for WWT, from 9 to 45 by a step of 9 for WEX and varied from 1 to 5 by a step of 1 for Nobel and UIS. The execution time were reported in Fig. 12. The error rate of Nobel and UIS was fixed at 10%, and we generated 20K tuples for UIS. To better represent the impact of indexes, we did not sum the time of reading and handling KBs.

There is no doubt that with the growing size of ruleset, fRepair was more efficient than bRepair. For example, when there were 5 DRs to repair UIS utilizing DBpedia, bRepair ran 1323s, while fRepair only ran 217s. For WWT, fRepair was not so faster than bRepair. It is because the extra cost of sorting rules and keeping inverted lists became unnegligible when only a few of DRs were used for repairing a few tuples.

(B) Varying #-Tuple In this part of experiment, we evaluated the scalability of our methods and compared with the other three repair algorithms: KATARA, Llunatic and constant CFDs. We utilized the UIS Database Generator and varied the number of tuples from 20 to 100 K by a step of 20 K, fixing the error rate at 10%. The experimental result was reported in Fig. 13. Note that the time of reading and handling KBs was included in this part of experiments.

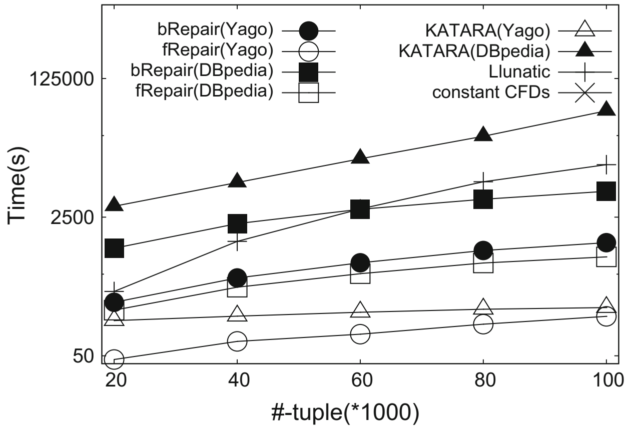

Fig. 13 Efficiency (varying #-tuple)

Figure 13 indicates that the impact of indexes became more and more remarkable with the growing the data size. For example, when there were 100K tuples to repair, the bRepair algorithm utilizing Yago ran 1216s, while fRepair only ran 152s. The fRepair algorithm always ran faster than Llunatic, and the time cost of Llunatic increased faster along with the number of tuples grew. The reason is that Llunatic needed to consider multiple tuples to detect violations and holistically consider multiple violations to decide a repair strategy. Our methods also ran faster than KATARA especially for DBpedia, because KATARA needed to list all instance graphs and find the most similar one for each tuple. Note that constant CFDs use only instances, thus it can repair 100K tuples within 1s.

Summary of experimental findings We find the followings. (1) DRs are effective in using KBs for cleaning relations. Without experts being involved, DRs are more accurate than the state-of-the-art data cleaning system KATARA that also uses KBs (Exp-1). (2) DRs are more effective than IC-based data cleaning (Exp-2). Note that we did not compared with other rule-based algorithms that are also ensured correctness if the rules are correct. The only reason is that existing rule-based methods do not rely on KBs but on expert knowledge or master tables. Though similar, we focus on how to design rules for trusted cleaning using KBs, which cannot be achieved by existing rule-based systems. (3) We can generate valid DRs from positive examples and negative examples, and the reduction strategy is effective (Exp-3). (4) It is efficient and scalable to apply DRs (Exp-4), since repairing one tuple is irrelevant to any other tuple, which is thus naturally parallelizable.

8 Conclusion

In this paper, we have proposed detective rules. Given a relation and a KB, DRs tell us that which tuple values are correct, which tuple values are erroneous, and how to repair them if there is enough evidence in the KB, in a deterministic fashion. We have studied fundamental problems associated with DRs, such as consistency analysis. We have proposed efficient data repairing algorithms including multiple optimization strategies and data structures. We have discussed how to generate DRs. Finally, we have experimentally verified the effectiveness and efficiency of DRs.

Acknowledgements This work was supported by the 973 Program of China (2015CB358700), NSF of China (61632016, 61472198, 61521002, 61661166012), and TAL education.

References

1. Abedjan, Z., Chu, X., Deng, D., Fernandez, R.C., Ilyas, I.F., Ouzzani, M., Papotti, P., Stonebraker, M., Tang, N.: Detecting data errors: where are we and what needs to be done? PVLDB **9**(12), 993–1004 (2016)
2. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Boston (1995)
3. Anchuri, P., Zaki, M.J., Barkol, O., Golan, S., Shamy, M.: Approximate graph mining with label costs. In: KDD, pp. 518–526 (2013)
4. Arenas, M., Bertossi, L.E., Chomicki, J.: Consistent query answers in inconsistent databases. In: SIGMOD, pp. 68–79. ACM (1999)
5. Bach, S.H., Broecheler, M., Huang, B., Getoor, L.: Hinge-loss markov random fields and probabilistic soft logic. *CoRR*, arXiv:1505.04406 (2015)
6. Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Whang, S.E., Widom, J.: Swoosh: a generic approach to entity resolution. VLDB J. **18**(1), 255–276 (2009)
7. Bohannon, P., Fan, W., Flaster, M., Rastogi, R.: A cost-based model and effective heuristic for repairing constraints by value modification. In: SIGMOD (2005)
8. Chai, C., Li, G., Li, J., Deng, D., Feng, J.: Cost-effective crowdsourced entity resolution: a partial-order approach. In: SIGMOD, pp. 969–984 (2016)
9. Chiang, F., Miller, R.J.: A unified model for data and constraint repair. In: ICDE (2011)
10. Chu, X., Ilyas, I.F., Papotti, P.: Holistic data cleaning: putting violations into context. In: ICDE (2013)
11. Chu, X., Morcos, J., Ilyas, I.F., Ouzzani, M., Papotti, P., Tang, N., Ye, Y.: KATARA: a data cleaning system powered by knowledge bases and crowdsourcing. In: SIGMOD (2015)
12. Cong, G., Fan, W., Geerts, F., Jia, X., Ma, S.: Improving data quality: consistency and accuracy. In: VLDB (2007)
13. Dallachiesa, M., Ebaid, A., Eldawy, A., Elmagarmid, A.K., Ilyas, I.F., Ouzzani, M., Tang, N.: NADEEF: a commodity data cleaning system. In: SIGMOD (2013)
14. Deng, D., Jiang, Y., Li, G., Li, J., Yu, C.: Scalable column concept determination for web tables using large knowledge bases. PVLDB **6**(13), 1606–1617 (2013)
15. Deng, D., Li, G., Wen, H., Feng, J.: An efficient partition based method for exact set similarity joins. PVLDB **9**(4), 360–371 (2015)
16. Deshpande, O., Lamba, D.S., Tourn, M., Das, S., Subramaniam, S., Rajaraman, A., Harinarayan, V., Doan, A.: Building, maintaining, and using knowledge bases: a report from the trenches. In: SIGMOD Conference (2013)
17. Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., Zhang, W.: Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In: SIGKDD (2014)

18. Dong, X.L., Gabrilovich, E., Heitz, G., Horn, W., Murphy, K., Sun, S., Zhang, W.: From data fusion to knowledge fusion. PVLDB **7**(10), 881–892 (2014)
19. Fan, W.: Dependencies revisited for improving data quality. In: PODS (2008)
20. Fan, W., Fan, Z., Tian, C., Dong, X.L.: Keys for graphs. PVLDB **8**(12), 1590–1601 (2015)
21. Fan, W., Geerts, F., Jia, X., Kementsietsidis, A.: Conditional functional dependencies for capturing data inconsistencies. TODS **33**(2), 6 (2008)
22. Fan, W., Jia, X., Li, J., Ma, S.: Reasoning about record matching rules. PVLDB **2**(1), 407–418 (2009)
23. Fan, W., Li, J., Ma, S., Tang, N., Yu, W.: Towards certain fixes with editing rules and master data. VLDB J. **21**(2), 213–238 (2012)
24. Feng, J., Wang, J., Li, G.: Trie-join: a trie-based method for efficient string similarity joins. VLDB J. **21**(4), 437–461 (2012)
25. Geerts, F., Mecca, G., Papotti, P., Santoro, D.: The LLUNATIC data-cleaning framework. PVLDB **6**(9), 625–636 (2013)
26. Hao, S., Tang, N., Li, G., Li, J.: Cleaning relations using knowledge bases. In: ICDE (2017)
27. He, J., Veltri, E., Santoro, D., Li, G., Mecca, G., Papotti, P., Tang, N.: Interactive and deterministic data cleaning. In: SIGMOD (2016)
28. Heer, J., Hellerstein, J.M., Kandel, S.: Predictive interaction for data transformation. In: CIDR (2015)
29. Herzog, T.N., Scheuren, F.J., Winkler, W.E.: Data Quality and Record Linkage Techniques. Springer, Berlin (2009)
30. Hoffart, J., Suchanek, F.M., Berberich, K., Weikum, G.: YAGO2: A spatially and temporally enhanced knowledge base from wikipedia. Artif. Intell. **194**, 28–61 (2013)
31. Interlandi, M., Tang, N.: Proof positive and negative in data cleaning. In: ICDE (2015)
32. Jiang, Y., Li, G., Feng, J., Li, W.: String similarity joins: an experimental evaluation. PVLDB **7**(8), 625–636 (2014)
33. Khayyat, Z., Ilyas, I.F., Jindal, A., Madden, S., Ouzzani, M., Papotti, P., Quiané-Ruiz, J.-A., Tang, N., Yin, S.: Bigdansing: a system for big data cleansing. In: SIGMOD (2015)
34. Li, G.: A human-machine method for web table understanding. In: WAIM, pp. 179–189 (2013)
35. Li, G.: Human-in-the-loop data integration. PVLDB **10**(12), 2006–2017 (2017)
36. Li, G., Chai, C., Fan, J., Weng, X., Li, J., Zheng, Y., Li, Y., Yu, X., Zhang, X., Yuan, H.: CDB: optimizing queries with crowd-based selections and joins. In: SIGMOD, pp. 1463–1478 (2017)
37. Li, G., Deng, D., Wang, J., Feng, J.: PASS-JOIN: a partition-based method for similarity joins. PVLDB **5**(3), 253–264 (2011)
38. Li, G., Ooi, B.C., Feng, J., Wang, J., Zhou, L.: EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In: SIGMOD, pp. 903–914 (2008)
39. Li, G., Wang, J., Zheng, Y., Franklin, M.J.: Crowdsourced data management: a survey. IEEE Trans. Knowl. Data Eng. **28**(9), 2296–2319 (2016)
40. Limaye, G., Sarawagi, S., Chakrabarti, S.: Annotating and searching web tables using entities, types and relationships. PVLDB **3**(12), 1338–1347 (2010)
41. Morsey, M., Lehmann, J., Auer, S., Ngomo, A.N.: Dbpedia SPARQL benchmark—performance assessment with real queries on real data. In: ISWC (2011)
42. Niu, F., Ré, C., Doan, A., Shavlik, J.W.: Tuffy: Scaling up statistical inference in markov logic networks using an RDBMS. PVLDB **4**(6), 373–384 (2011)
43. Raman, V., Hellerstein, J.M.: Potter's wheel: an interactive data cleaning system. In: VLDB (2001)
44. Rekatsinas, T., Chu, X., Ilyas, I.F., Ré, C.: Holoclean Holistic data repairs with probabilistic inference. PVLDB **10**(11), 1190–1201 (2017)
45. Shang, Z., Liu, Y., Li, G., Feng, J.: K-join: knowledge-aware similarity join. IEEE Trans. Knowl. Data Eng. **28**(12), 3293–3308 (2016)
46. Shin, J., Wu, S., Wang, F., Sa, C.D., Zhang, C., Ré, C.: Incremental knowledge base construction using deepdive. PVLDB **8**(11), 1310–1321 (2015)
47. Singh, R., Meduri, V., Elmagarmid, A.K., Madden, S., Papotti, P., Quiané-Ruiz, J., Solar-Lezama, A., Tang, N.: Generating concise entity matching rules. In: PVLDB (2017)
48. Singh, R., Meduri, V., Elmagarmid, A.K., Madden, S., Papotti, P., Quiané-Ruiz, J., Solar-Lezama, A., Tang, N.: Synthesizing entity matching rules by examples. In: SIGMOD demo (2017)
49. Song, S., Cheng, H., Yu, J.X., Chen, L.: Repairing vertex labels under neighborhood constraints. PVLDB **7**(11), 987–998 (2014)
50. Venetis, P., Halevy, A.Y., Madhavan, J., Pasca, M., Shen, W., Wu, F., Miao, G., Wu, C.: Recovering semantics of tables on the web. PVLDB **4**(9), 528–538 (2011)
51. Volkovs, M., Chiang, F., Szlichta, J., Miller, R.J.: Continuous data cleaning. In: ICDE (2014)
52. Wang, J., Li, G., Feng, J.: Trie-join: efficient trie-based string similarity joins with edit-distance constraints. PVLDB **3**(1), 1219–1230 (2010)
53. Wang, J., Li, G., Feng, J.: Fast-join: an efficient method for fuzzy token matching based string similarity join. In: Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11–16, 2011, Hannover, Germany, pp. 458–469 (2011)
54. Wang, J., Li, G., Kraska, T., Franklin, M.J., Feng, J.: Leveraging transitive relations for crowdsourced joins. In: SIGMOD, pp. 229–240 (2013)
55. Wang, J., Tang, N.: Towards dependable data repairing with fixing rules. In: SIGMOD (2014)
56. Yakout, M., Berti-Equille, L., Elmagarmid, A.K.: Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In: SIGMOD (2013)
57. Yakout, M., Elmagarmid, A.K., Neville, J., Ouzzani, M., Ilyas, I.F.: Guided data repair. PVLDB **4**(5), 279–289 (2011)
58. Yu, M., Wang, J., Li, G., Zhang, Y., Deng, D., Feng, J.: A unified framework for string similarity search with edit-distance constraint. VLDB J. **26**(2), 249–274 (2017)
59. Zhuang, Y., Li, G., Feng, Z.Z.J.: Hike: a hybrid human-machine method for entity alignment in large-scale knowledge bases. In: CIKM (2017)
60. Zhuang, Y., Li, G., Zhong, Z., Feng, J.: PBA: partition and blocking based alignment for large knowledge bases. In: DASFAA, pp. 415–431 (2016)