Learning How to Correct a Knowledge Base from the Edit History

Thomas Tanon, Camille Bourgaux, Fabian Suchanek

To cite this version:

Thomas Tanon, Camille Bourgaux, Fabian Suchanek. Learning How to Correct a Knowledge Base from the Edit History. World Wide Web Conference, May 2019, San Francisco, United States. 10.1145/3308558.331358. hal-02066041

HAL Id: hal-02066041

https://hal-imt.archives-ouvertes.fr/hal-02066041

Submitted on 13 Mar 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License
Learning How to Correct a Knowledge Base from the Edit History

Thomas Pellissier Tanon
Télécom ParisTech
ttanon@enst.fr

Camille Bourgaux
DI ENS, CNRS, ENS, PSL Univ. & Inria
camille.bourgaux@ens.fr

Fabian Suchanek
Télécom ParisTech
suchanek@enst.fr

ABSTRACT
The curation of a knowledge base is a crucial but costly task. In this work, we propose to take advantage of the edit history of the knowledge base in order to learn how to correct constraint violations. Our method is based on rule mining, and uses the edits that solved some violations in the past to infer how to solve similar violations in the present. The experimental evaluation of our method on Wikidata shows significant improvements over baselines.

KEYWORDS
knowledge base; history; data cleaning; rule mining; Wikidata

ACM Reference Format:

1 INTRODUCTION
Knowledge bases (KBs) play a key role in many applications and lay at the core of the Semantic Web. They contain entities (such as persons, cities, etc.) and statements about them (such as relationships between persons, places of residence, etc.). In this work, we focus on RDFS-style KBs, such as Wikidata [38], YAGO [35] or DBpedia [7]. The data quality of a KB is crucial for its usability. However, it is usually very costly to check the correctness of the data, because KBs can be huge (Wikidata, e.g., contains about 50 millions entities). Moreover, KBs are often built using methods that are error-prone. For instance YAGO and DBpedia are automatically extracted from Wikipedia, Wikidata, for its part, is a collaborative KB that anyone can edit, with more than 18,000 active contributors.

A way of avoiding or at least detecting some of the flaws in the data is to impose constraints on the KB. Such constraints can enforce that some information must be present (e.g., ensuring that every human being has a birth date), or that some statements may not occur (e.g., ensuring that a person is not also a city). Constraints are related to, but different from, ontological rules: A constraint imposes a certain condition, whereas an ontological rule infers certain statements. For example, consider a KB that contains the statement "Spinoza is a human being" without knowing any birth date for Spinoza, and consider a rule "All human beings have a birth date". If the rule is taken as an ontological rule, then it would just infer that Spinoza has some birth date. If the rule is taken as a constraint, in contrast, the KB would be considered incorrect. Constraints are thus similar in spirit to database integrity constraints. In practice, constraints often have exceptions. Therefore, it is useful to allow data that does not respect them (in Wikidata, e.g., constraint violations are simply flagged). Nonetheless, by design, most of the constraint violations are not exceptions but actual errors and proposing to repair them is a good starting point when it comes to improving KB quality.

In this paper, we aim at learning how to repair constraint violations. Our goal is to help a KB editor by suggesting how to clean the data locally (providing a solution to a particular constraint violation) or globally (providing rules that can be automatically applied to all constraint violations of a given form once validated by the editor). To do that, we take advantage of the edit history of the KB. We use it to mine correction rules that express how different kinds of constraint violations are usually solved. To the best of our knowledge, this is the first work that builds on past users’ corrections in the history to infer possible new ones. We validate our framework experimentally on Wikidata, for which the whole edit history of more than 700 millions edits is available. Our experiments show substantial improvements over baselines. More concretely, our contributions are as follows:

• a formal definition of the problem of correction rule mining,
• a dataset of more than 67M past corrections for ten different kinds of Wikidata constraints (13k constraints in total),
• a correction rule mining algorithm, together with an implementation for Wikidata, CorHist,\textsuperscript{2}
• a suggestion tool for users to correct data based on our mined correction rules,
• an experimental evaluation based both on the prediction of the corrections in the history and on user validation of the suggested local corrections.

2 RELATED WORK
We start with a brief discussion of works relevant to our problem along three axes: constraints for KBs, KB cleaning, and rule learning.

Constraints. Constraints have long been used in databases and KBs to express rules that the data should follow. Databases typically operate under the closed world assumption, where missing facts are considered to be false. This allows for “completeness” constraints such as tuple generating dependencies. KBs, in contrast, operate under the open world assumption, where missing

\textsuperscript{1}available at https://doi.org/10.6084/m9.figshare.7712720
\textsuperscript{2}available at https://github.com/Tpt/corhist
\textsuperscript{3}available at https://tools.wmflabs.org/wikidata-game/distributed/#game=43

This paper is published under the Creative Commons Attribution 4.0 International (CC-BY 4.0) license. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution.

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313584
facts are not necessarily false. They thus classically have only “correctness” constraints, such as disjointness or functionality axioms (corresponding to special cases of denial constraints and equality generating dependencies in databases).

To express also completeness constraints, several works propose to use description logics, with varying semantics [28, 37]. Another possibility is to use queries that should or should not hold as constraints (see e.g., [24] for methods for writing constraint queries in SPARQL). Other approaches define constraint languages to specify conditions for RDF graph [9] validation, such as SHACL [22] or ShEx [8]. It has been argued in [30] that description logics under the closed world assumption are also suitable for constraint checking in RDF, which can then be implemented with SPARQL queries. In our work, we follow a similar path, using description logic axioms as constraints for RDFs KBs, because it corresponds best to what we observe in current real-world KBs.

Contrary to the above works, we do not aim at expressing constraints, but at repairing their violations. The correction rules we learn for this purpose are similar in spirit to active integrity constraints [11], which specify for each constraint a set of possible repair actions. This type of constraints has recently been applied to description logics KBs as well [32]. Conditioned active integrity constraints add conditions for choosing among the possible actions, and we propose, in a similar spirit, to take into account the context of the constraint violation to correct it. Different from the existing work [11, 32], our goal is to mine correction rules automatically from the edit history of the KB.

**Knowledge base cleaning.** Several recent approaches have dealt with the interactive cleaning of KBs. The proposed methods detect when a constraint is violated, compute the responsible facts, and with the interactive cleaning of KBs. The proposed methods detect we observe in current real-world KBs.

**Rule learning.** Mining logical rules by finding correlations in a dataset is a well-established research topic. In particular, learning patterns in the data can be used for completing KBs [14, 36]. An algorithm for learning conjunctive patterns from a KB enriched with a set of rules is described in [20]. Methods similar to association rule mining have also been used for induction of new ontological rules from a KB [33]. A more recent trend is to use embedding-based models for KB completion. A comparison between these models and usual rule learning approaches is reported in [27] and significant recent works in this area include [19, 39, 40].

In this paper, we use a vanilla rule mining algorithm inspired by [14]. Our contribution is not the rule mining per se, but the application of rule mining to the edit history of a KB in order to mine correction rules. This avenue has, to the best of our knowledge, never been investigated.

### 3 PRELIMINARIES

In this work, we use description logics (DL) [4] as KB language and as constraint language, because they are the foundation of the Semantic Web standard OWL [16].

**Syntax.** We assume a set $NC$ of concept names (unary predicates, also called classes), a set $Nc$ of role names (binary predicates, also called properties), and a set $Nt$ of individuals (also called constants).

An ABox (dataset) is a set of concept or role assertions of the form $A(a)$ or $R(a, b)$, where $A \in NC, R \in Nc, a, b \in Nt$. A TBox (ontology) is a set of axioms whose form depends on the DL $L$ in question, and expresses relationships between concepts and roles (e.g., concept or role hierarchies, role domains and ranges...). A knowledge base (KB) $K = T \cup A$ is the union of an ABox $A$ and a TBox $T$.

In this work, we assume that $T$ is a flat DL TBox [23], i.e., that $L$ differs from the standard RDF Schema (RDFS) [17] only by allowing inverse roles in role inclusions. More precisely, $T$ can contain concept inclusions of the form $A1 \sqsubseteq A2$ (subclass), $\exists P \sqsubseteq A$ (domain or range), and role inclusions $P1 \sqsubseteq P2$ (subproperty), where $A1 \in NC$ and $P1, P2 \in Nc$.

A KB can also be written as a set of RDF triples $(s, p, o)$ where $s$ is the subject, $p$ is the property, and $o$ the object, using special properties to translate concept membership and relationships between concepts and roles [29]. A concept assertion $A(a)$ is written as $(a, rdf:type, A)$, and a role assertion $R(a, b)$ as $(a, R, b)$. Flat DL TBox axioms can also be represented by single triples. For example, $A1 \sqsubseteq A2$ is written as $(\forall a, \text{rdfs:subClassOf}, A2)$ and $\exists R \sqsubseteq A$ is written as $(\forall a, R, \text{rdfs:range}, A)$.

**Semantics.** We recall the standard semantics of DL KBs. An interpretation has the form $I = (\Delta^I, \cdot^I)$, where $\Delta^I$ is a non-empty set and $\cdot^I$ is a function that injectively maps each $a \in Nt$ to $a^I \in \Delta^I$ (unique name assumption), $\top \in \Delta^I$, each $A \in NC$ to $A^I \subseteq \Delta^I$, and each $R \in NR$ to $R^I \subseteq \Delta^I \times \Delta^I$. The function $\cdot^I$ is straightforwardly extended to general concepts and roles, e.g. $(\neg B)^I = \Delta^I \setminus B^I$, $(R^\sqsubseteq)^I = \{(c, d) \mid (c, d) \in R^I\}$, $\{a_1, \ldots, a_n\}^I = \{a_1^I, \ldots, a_n^I\}$, $(\exists P \cdot B)^I = \{c \mid \exists d : (c, d) \in P^I \land d \in B^I\}$, $(B1 \sqcap B2)^I = B1^I \cap B2^I$, $(B1 \sqcup B2)^I = B1^I \cup B2^I$. An interpretation $I$ satisfies an inclusion $G \sqsubseteq H$, if $G^I \subseteq H^I$; it satisfies an axiom (func $P$) if $P^I$ is transitive; and it satisfies an axiom (trans $P$) if $P^I$ is reflexive and total; and it satisfies $A(a)$ (resp. $R(a, b)$), if $a^I \in A^I$ (resp. $(a^I, b^I) \in R^I$). We write $I \models \alpha$ if $I$ satisfies the DL axiom $\alpha$.

An interpretation $I$ is a model of $K = T \cup A$ if $I$ satisfies all axioms in $K$. A KB $K$ entails a DL axiom $\alpha$ if $I \models \alpha$ for every model $I$ of $K$.

**Queries.** A conjunctive query (CQ) takes the form $q(\bar{x}) = \exists \bar{y} \psi(\bar{x}, \bar{y})$, where $\psi$ is a conjunction of atoms of the form $A(t)$ or $R(t, t')$ or of equalities $t = t'$, where $t, t'$ are individual names or variables from
We denote by a union of CQs (BCQ). A BCQ $q$ is satisfied by an interpretation $I$, written $I \models q$, if there is a homomorphism $\pi$ mapping the variables and individual names of $q$ into $\Delta$ such that: $\pi(a) = a^I$ for every $a \in N_I$, $\pi(t) = a^I$ for every concept atom $A(t)$ in $\psi$, $\pi(t(t')) = R^I$ for every role atom $R(t(t'))$ in $\psi$, and $\pi(t) = \pi(t')$ for every $t = t'$ in $\psi$. We also consider as BCQs the queries true and false which are respectively always and never satisfied by an interpretation. A BCQ $q$ is entailed from $\mathcal{K}$, written $\mathcal{K} \models q$, iff $q$ is satisfied by every model of $\mathcal{K}$. A tuple of constants $\bar{a}$ is a (certain) answer to a CQ $q(\bar{x})$ if $\mathcal{K} \models q(\bar{a})$ where $q(\bar{a})$ is the BCQ obtained by replacing the variables from $\bar{x}$ by the constants $\bar{a}$. We denote by answers($q(\bar{x})$, $\mathcal{K}$) the set of answers of $q(\bar{x})$ over $\mathcal{K}$. A union of CQs (UCQ) is a disjunction of CQs and has as answers the union of the answers of the CQs it contains.

**Canonical model.** It is well-known that a flat QL KB $\mathcal{K}$ has a canonical model $I_{\mathcal{K}}$ such that for every BCQ $q$, $\mathcal{K} \models q$ iff $I_{\mathcal{K}} \models q$. The domain of $I_{\mathcal{K}}$ is the set of individual names that occur in $\mathcal{K}$ and $A^{I_{\mathcal{K}}} = \{a | A \models b(a), T \subseteq B \subseteq A\}$ for every $A \in N_{\mathcal{A}}$ and $R^{I_{\mathcal{K}}} = \{(a, b) | A \models (p(a, b), T \subseteq P \subseteq R)\}$ for every $R \in N_{\mathcal{B}}$ [23].

### 4 CONSTRAINTS

This section defines the constraints that can be imposed on a KB, and relates the problem of checking that a KB complies with these constraints to QA answering over this KB.

**Defining constraints.** In this work, we consider two types of constraints: consistency constraints (which express that some statements are contradictory), and completeness constraints (which impose that certain statements should hold in the KB as soon as some others do). While violations of consistency constraints can only be solved by removing statements, those of completeness constraints can also be solved by adding statements.

#### Definition 1 (Constraint): Constraints are built from complex concepts and roles defined by the following grammar rules:

$P ::= R \mid R^*$

$B ::= \top \mid A \mid B \lor B \mid B \land B \mid \exists P \cdot \{a_1, \ldots, a_n\} \mid \exists P \cdot B$

where $R \in N_{\mathcal{R}}$, $A \in N_{\mathcal{C}}$, $a_1, \ldots, a_n \in N_I$.

- A consistency constraint is a concept inclusion of the form $B_1 \subseteq \neg B_2$ or of the form $B \subseteq \{a_1, \ldots, a_n\}$, a role inclusion of the form $P_1 \subseteq \neg P_2$ or a functionality axiom of the form $(\text{fun} P)$.

- A completeness constraint is a concept inclusion of the form $B_1 \subseteq B_2$, a role inclusion of the form $P_1 \subseteq P_2$, or a transitivity axiom of the form $(\text{trans} P)$.

This definition of constraints covers the “constraining” versions of the majority of the most popular DL axioms used on the Web of Data according to the ranking done by [15].

We assume without loss of generality that concepts of the form $B_1 \lor B_2$ or $\{a_1, \ldots, a_n\}$ with $n > 1$ appear only on the right side of inclusions, and not at all in negative inclusions of the form $B_1 \subseteq \neg B_2$. For example, we assume that $\exists P \cdot (B_1 \lor B_2) \subseteq C$ is rewritten as $\exists P \cdot B_1 \subseteq C$, $\exists P \cdot B_2 \subseteq C$, and $B \subseteq \neg \exists P \cdot \{a_1, \ldots, a_n\}$ is rewritten as $B \subseteq \neg \exists P \cdot \{a_1\}, \ldots, B \subseteq \neg \exists P \cdot \{a_n\}$. As usual, we abbreviate $\exists P \cdot \top$ as $\exists P$.

**Example 1:** As a running example, we consider the following KB $\mathcal{K} = \mathcal{T} \cup \mathcal{A}$ and set of constraints $C$ inspired from Wikidata. Our TBox $\mathcal{T}$ expresses that human beings and deities are persons. Our ABox $\mathcal{A}$ provides information on several individuals. Our constraints $C$ state that there are three possible genders (consistency constraint), that those who have a mother or are a mother must be persons or animals, that a mother must have gender female, and that if a has mother $b$, then $b$ must have child (completeness constraints).

$\mathcal{T} = \{ \text{Human} \subseteq \text{Person}, \text{Deity} \subseteq \text{Person} \}$

$\mathcal{A} = \{ \text{Deity(Zeus)}, \text{Deity(Rhea)}, \text{hasGender(Zeus, masculine)}, \text{hasGender(Rhea, female)}, \text{hasMother(Zeus, Rhea)}, \text{hasChild(Rhea, Zeus)}, \text{Human(Spinoza)}, \text{hasMother(Spinoza, Marques)} \}$

$C = \{ \Gamma_0 = \exists \text{hasGender} \subseteq \{\text{male, female, nonbinary}\}, \Gamma_1 = \exists \text{hasMother} \subseteq \text{Person} \cup \text{Animal}, \Gamma_2 = \exists \text{hasMother} \subseteq \text{Person} \cup \text{Animal}, \Gamma_3 = \exists \text{hasMother} \subseteq \exists \text{hasGender} \cdot \{\text{female}\}, \Gamma_4 = \exists \text{hasMother} \subseteq \exists \text{hasChild} \}$

We say that a KB $\mathcal{K}$ satisfies a constraint $\Gamma \in C$ if $I_{\mathcal{K}} \models \Gamma$, where $I_{\mathcal{K}}$ is the canonical model of $\mathcal{K}$. Otherwise, $\mathcal{K}$ violates $\Gamma$.

**Example 2:** In our running example, the KB $\mathcal{K}$ satisfies $\Gamma_1$ since $\exists \text{hasMother}^{I_{\mathcal{K}}} = \{\text{Zeus, Spinoza}\}$ and $I_{\mathcal{K}} \models \exists \text{Person}(\text{Spinoza})$. However, it violates $\Gamma_2$ because $I_{\mathcal{K}} \not\models \exists \text{Person}(\text{Marques}) \lor \text{Animal}(\text{Marques})$ while $\text{Marques} \in \exists \text{hasMother}^{I_{\mathcal{K}}}$. It violates $\Gamma_3$ and $\Gamma_4$ for similar reasons. Finally, $(\exists \text{hasGender}(\text{Zeus, masculine}), \Gamma_0)$ has no model because of the unique name assumption (which enforces that the interpretation of masculine differs from those of male, female and nonbinary). Hence, $I_{\mathcal{K}}$ cannot be a model of $\Gamma_0$. Thus, $\mathcal{K}$ violates $\Gamma_0$.

Note the semantic difference between the constraints and the axioms of the TBox: The axiom (Human (\text{Person} in the TBox makes every human an answer to the query asking for persons. In contrast, if we had put the axiom in the set of constraints, it would have required all human beings in the KB to be explicitly marked as persons. As another example, consider the axiom (func hasBirthdate) (which says that everyone can have at most one birth date). If this axiom appears in the TBox, it renders the KB inconsistent whenever a person is given two distinct dates of birth. This has severe consequences on the reasoning capabilities, since everything is entailed from an inconsistent KB. If this axiom is in the set of constraints, in contrast, then distinct dates of birth lead only to the violation of the constraint. This gives us relevant information without having any impact on the usability of the KB.

**Checking constraints.** We show that our setting allows us to check constraint satisfaction via QA answering. For this purpose, we use a function $\pi$, which maps each constraint $\Gamma \in C$ to a rule of the form $\exists \mathcal{P}(\bar{x}, \bar{y}) \rightarrow \exists \mathcal{P'}(\bar{x}, \bar{z})$. This function is defined recursively as shown in Table 1. The left side of the rule is called the body and its right side the head.

**Example 3:** In our running example, we obtain the following rules:

$\Gamma_0(\bar{x}) : \exists \text{hasGender}(y, x) \rightarrow x \equiv \text{male} \lor x \equiv \text{female} \land x \equiv \text{nonbinary}$

$\Gamma_1(\bar{x}) : \exists \text{hasMother}(x, y) \rightarrow \exists \text{Person}(x) \lor \exists \text{Animal}(x)$

$\Gamma_2(\bar{x}) : \exists \text{hasMother}(y, x) \rightarrow \exists \text{Person}(x) \lor \exists \text{Animal}(x)$

$\Gamma_3(\bar{x}) : \exists \text{hasMother}(y, x) \rightarrow \exists \text{hasGender}(x, z) \land z \equiv \text{female}$

$\Gamma_4(\bar{x}, y) : \exists \text{hasMother}(y, x) \rightarrow \exists \text{hasChild}(y, x)$
Table 1: Translation of DL axioms into rules. Variables that appear in the right column and not in the left one are fresh.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi(T, x)$</td>
<td>true</td>
</tr>
<tr>
<td>$\pi(A, x)$</td>
<td>$A(x)$</td>
</tr>
<tr>
<td>$\pi({a_1, \ldots, a_n}, x)$</td>
<td>$x = a_1 \lor \cdots \lor x = a_n$</td>
</tr>
<tr>
<td>$\pi(R, x, y)$</td>
<td>$R(x, y)$</td>
</tr>
<tr>
<td>$\pi(R^-, x, y)$</td>
<td>$R(x, y)$</td>
</tr>
<tr>
<td>$\pi(B_1 \cap B_2, x)$</td>
<td>$\pi(B_1, x) \land \pi(B_2, x)$</td>
</tr>
<tr>
<td>$\pi(B_1 \cup B_2, x)$</td>
<td>$\pi(B_1, x) \lor \pi(B_2, x)$</td>
</tr>
<tr>
<td>$\pi(\exists P - \cdot B, x)$</td>
<td>$\exists y (\pi(P, x, y) \land \pi(B, x))$</td>
</tr>
<tr>
<td>$\pi(B \subseteq C)$</td>
<td>$\pi(B, x) \rightarrow \pi(C, x)$</td>
</tr>
<tr>
<td>$\pi(P \subseteq \top)$</td>
<td>$\pi(P, x, y) \rightarrow \pi(\top, x, y)$</td>
</tr>
<tr>
<td>$\pi(B \subseteq C)$</td>
<td>$\pi(B, x) \land \pi(C, x) \rightarrow false$</td>
</tr>
<tr>
<td>$\pi(P \subseteq \bot)$</td>
<td>$\pi(P, x, y) \land \pi(\bot, x, y) \rightarrow false$</td>
</tr>
<tr>
<td>$\pi(func P)$</td>
<td>$\pi(P, x, y) \land \pi(P, x, z) \rightarrow y = z$</td>
</tr>
<tr>
<td>$\pi(trans P)$</td>
<td>$\pi(P, x, y) \land \pi(P, y, z) \rightarrow \pi(P, x, z)$</td>
</tr>
</tbody>
</table>

The following proposition shows that this transformation is sound and that the rule body and head can be rewritten as CQ and UCQ.

**Proposition 1:** For every constraint $\Gamma \subseteq C$, $\pi(\Gamma)$ can be rewritten as a rule $\Gamma(x) \rightarrow h(x)$ where $h(x)$ is a CQ. $h(x)$ is a UCQ and for every flat QL KB $\mathcal{K}$, $\Gamma$ satisfies $\Gamma$ iff answers($h(x), \mathcal{K}$) $\subseteq$ answers($h(x), \mathcal{K}$).

**Proof.** By our assumptions on the form of the concepts that occur in the left side of the inclusions or in the right side of a negative inclusion, $b(x) := \exists y \varphi(x, y)$ is a CQ. It is easy to show by structural induction that for every concept $B$ (resp. role $P$), $\pi(B, x)$ (resp. $\pi(P, x, y)$) can be written as a UCQ $q(x)$ (resp. $q(x, y)$) and that answers($q(x), \mathcal{K}$) $= B^K$ (resp. answers($q(x, y), \mathcal{K}$) $= P^K$). If $\Gamma$ is a completeness constraint of the form $B_1 \subseteq B_2$ (resp. $P_1 \subseteq P_2$), or a consistency constraint of the form $B \subseteq \{a_1, \ldots, a_n\}$, the result follows immediately since $\Gamma$ satisfies $\Gamma$ iff $\Delta_{B_K} \models \Gamma$. If $\Gamma$ is a consistency constraint of the form $B_1 \subseteq \lnot B_2$ (resp. $P_1 \subseteq \lnot P_2$), answers($h(x), \mathcal{K}$) $= B^K \cap B_2^K$ (resp. answers($h(x), \mathcal{K}$) $= P^K \cap P_2^K$) and is empty iff $\Gamma$ is satisfied. Since in this case $h(x) := false$, answers($h(x), \mathcal{K}$) $= \emptyset$, and the desired relation holds.

If $\Gamma$ is of the form (func $P$), since the answers of $h(x, y, z) := y = z$ over $\mathcal{K}$ are all possible tuples of the form $(a, b, b)$, $P^{FK}$ is functional iff answers($h(x, y, z), \mathcal{K}$) $\subseteq$ answers($h(x, y, z), \mathcal{K}$) and $\mathcal{K}$, and the comparison becomes $\mathcal{K}$ is a minimal subset of $\mathcal{K}$ such that $\mathcal{K} \cap \mathcal{K} = \emptyset$ and $\mathcal{K}$ is a minimal subset of $\mathcal{K}$ such that $\mathcal{K} \cap \mathcal{K} = \emptyset$ and $\mathcal{K}$ is a minimal subset of $\mathcal{K}$ such that $\mathcal{K} \cap \mathcal{K} = \emptyset$. $\Box$

**Constraint violations.** A constraint instance $\Gamma(\bar{a})$ of a constraint $\Gamma(\bar{x})$ is obtained by replacing the variables $\bar{x}$ by the individual names $\bar{a}$ in $\Gamma(\bar{x})$. This notion allows us to define constraint violations.

**Definition 2 (Constraint violation):** A violation of a constraint $\Gamma(\bar{x})$ in $\mathcal{K}$ is a minimal subset $\mathcal{V} \subseteq \mathcal{K}$ such that there exists $\bar{a}$ such that $\mathcal{V}$ violates $\Gamma(\bar{a})$ and $\mathcal{K}$ violates $\Gamma(\bar{a})$. We denote by Violations($\mathcal{K}, \Gamma(\bar{x})$) the set of violations of $\Gamma(\bar{x})$.

In this definition, the requirement that $\mathcal{K}$ violates $\Gamma(\bar{a})$ may seem superfluous. Yet, if $\Gamma$ is a completeness constraint, it may be the case that some $\mathcal{V} \subseteq \mathcal{K}$ violates $\Gamma(\bar{a})$, while $\mathcal{K}$ satisfies it.

**Example 4:** In our running example, it is easy to see that the subset $\mathcal{V}_0 = \{\text{hasGender(Zeus, masculine)}\}$ is a violation of $\Gamma_0$. Consider now $\mathcal{V} = \{\text{hasMother(Spinosa, Marques)}\}$. $\mathcal{V}$ is a violation of $\Gamma_2$, $\Gamma_3$ and $\Gamma_4$. Indeed, it violates $\Gamma_2$ (Marques), $\Gamma_3$ (Marques), and $\Gamma_4$ (Spinosa, Marques) and $\mathcal{K}$ does not satisfy any of these constraint instances. However, even if $\mathcal{V}$ violates $\Gamma_1$ (Spinosa), $\mathcal{V}$ is not a violation of $\Gamma_1$ because $\{\text{Human(Spinosa), Human \subseteq Person}\} \subseteq \mathcal{K}$ satisfies the head of $\Gamma_1$ (Spinosa).

If a constraint instance $\Gamma(\bar{a})$ is violated, $\bar{a} \in \text{answers}(h(\bar{x}), \mathcal{K})$ and $\bar{a} \notin \text{answers}(h(\bar{x}), \mathcal{K})$, so its violations are the minimal subsets of $\mathcal{K}$ responsible for $\bar{a} \in \text{answers}(h(\bar{x}), \mathcal{K})$. The next proposition relates constraint violations and justifications. A justification (also known as an explanation, axiom pinpointing, or MinAs) for the entailment of a BCQ is a minimal subset of the KB that entails the BCQ [21, 34].

**Proposition 2:** If $\mathcal{K}$ violates $\Gamma(\bar{a})$ : $\langle b(\bar{a}) \rightarrow b(h(\bar{x})) \rangle$, a subset $\mathcal{V} \subseteq \mathcal{K}$ is a violation of $\Gamma(\bar{a})$ iff $\mathcal{V}$ is a justification of $\mathcal{K} \models b(\bar{a})$.

**5 CORRECTIONS**

We now turn to correcting constraint violations.

**Solutions.** We will make use of atomic modifications of the KB to define solutions to constraint violations.

**Definition 3 (Atomic modification):** An atomic modification of a KB $\mathcal{K}$ is a pair $m = (M^+, M^-)$ of two sets of assertions or $\mathcal{L}$-axioms that takes one of the following forms:

Addition: $m = \{(s, p, o), \delta\}$, where $(s, p, o) \notin \mathcal{K}$

Deletion: $m = (\emptyset, \{(s, p, o), \delta\})$, where $(s, p, o) \in \mathcal{K}$

Replacement: $m = \{(s, p, o), \{(s', p', o')\}\}$, where $(s, p, o) \notin \mathcal{K}$, $(s', p', o') \in \mathcal{K}$, and $(s, p, o)$ differs from $(s', p', o')$ in exactly one component.

Thus, an atomic modification consists of two sets $M^+$ and $M^-$, each of which is either the empty set or a singleton set. $M^+$ will be added to the KB, and $M^-$ will be removed from the KB. Since the sets contain at most one triple, we slightly abuse the notation and identify the singletons with their elements (e.g., we will denote the addition of $(s, p, o)$ simply by $(s, p, o, \delta)$). A replacement is equivalent to a sequence of a deletion and an addition. We chose to keep it as an atomic modification because it corresponds to common knowledge base curation tasks, such as correcting an erroneous object for a given subject and predicate, or fixing a predicate misuse. Atomic modifications can be used to solve a constraint violation, as follows:

**Definition 4 (Solution):** A solution to a violation $\mathcal{V}$ of a constraint instance $\Gamma(\bar{a})$ in $\mathcal{K}$ is an atomic modification $(M^+, M^-)$ such that there exists $\mathcal{K}' \subseteq \mathcal{K}$ such that $\mathcal{V} \cup \mathcal{K}' \cup M^+ \setminus M^-$ satisfies $\Gamma(\bar{a})$. We call $(M^+, M^-)$ a solution to $\mathcal{V}$ for $\Gamma(\bar{a})$ in $\mathcal{K}$.

Note that $\Gamma(\bar{a})$ can still be violated in $(\mathcal{K} \cup M^+) \setminus M^-$ if $\mathcal{K}$ contains other violations of $\Gamma(\bar{a})$ for which $(M^+, M^-)$ is not a solution. For example, if $\Gamma(\bar{a}) : \exists \mathcal{R}(x, a) \land A(a) \rightarrow false$, and $\mathcal{K} = \{\mathcal{R}(a, b), \mathcal{R}(a, c), A(a)\}$, the deletion of $\mathcal{R}(a, b)$ is a solution to the violation $\{\mathcal{R}(a, b), A(a)\}$, but $\{\mathcal{R}(a, c), A(a)\}$ still violates $\Gamma(a)$. Note also that every constraint violation has at least one solution, which consists of the deletion of any of its elements. Solutions may also be additions or replacements, as in the following example:

**Example 5:** In our running example, the deletion $(\emptyset, \text{hasGender(Zeus, masculine)})$ and the replacement $(\text{hasGender(Zeus, male)},$
We thus consider only those past corrections that would have been 
violated if the constraint A ⊆ B. If, in the meantime, the inclusion 
C ⊆ B has been removed, we do not want to learn from this past 
correction. The following definition formalizes these requirements.

DEFINITION 7 (Relevant Past Correction): A relevant past correction 
(M⁺, M⁻) to a violation V of a constraint instance Γ(δ) in 𝊦 is a past 
correction such that (i) M⁺ ∪ M⁻ contains only assertions, and 
(ii) (V ∨ A₁) ∪ Tₚ contains a violation V' of Γ(δ) such that 
(M⁺, M⁻) is also a solution to V' in 𝐴₁ ∪ Tₚ.

We will now see how we can use the relevant past corrections to 
mine correction rules.

6 FROM HISTORY TO CORRECTION RULES

In this section, we propose an approach based on rule mining to 
learn correction rules for building solutions to constraint violations.

6.1 Extraction of the Relevant Past Corrections

Algorithm 1 constructs the set of relevant past corrections from the 
KB history. It consists of three main steps. First, it constructs 
patterns to spot KB modifications that could be part of a relevant past 
correction. Then it uses these patterns to extract atomic modifications 
that solved some violation in the past. Finally, the relevant past 
corrections are obtained by pruning those that have been reversed.

Algorithm 1 Construction of PCDataset

Input: set of constraints 𝐶, current TBox 𝑇ₚ, history (Kᵢ)₀≤𝑖≤p
Output: set of relevant past corrections PCDataset

// Construct correction seed patterns
for all 𝑥 ∈ 𝐶 such that Γ(δ) : b(𝑥) → h(𝑥) do
Patterns(Γ) := {(l, A(δ))[A(δ) ∈ b(𝑥)](b(𝑥) ∈ rewrite(b(𝑥), 𝑇ₚ))} // Construct extract past corrections
for 0 ≤ 𝑖 ≤ p − 1 do
if (M⁺ᵢ, M⁻ᵢ) such that 𝐾ᵢ₊₁ = (𝐾ᵢ ⊆ 𝑀⁺ᵢ) \ M⁻ᵢ matches 
some pattern in Patterns(Γ) then
PCDataset := (PCDataset \ Violations(Kᵢ₊₁, Γ(δ))) \ Violations(Kᵢ₊₁, Γ(δ))

// Remove reversed past corrections
for (M⁺ᵢ, M⁻ᵢ), Γ(δ), ∀ 𝑖 ∈ PCDataset do
if M⁺ᵢ ⊆ 𝐾ᵢ or M⁻ᵢ ⊆ 𝐾ᵢ then
PCDataset := (PCDataset \ (M⁺ᵢ, M⁻ᵢ), Γ(δ), ∀ 𝑖) // Remove non-reversed past corrections

Let us explain our algorithm with our running example. Consider 
the constraint I₀(x) : ∃y hasGender(y, x) → x = male ∨ x = female ∨ x = nonbinary. Assume that 
(Zeus, hasGender, masculine) was added between 𝐾₃ and 𝐾₂, but 
then replaced by (Zeus, hasGender, male) between 𝐾₁₀₀ and 𝐾₁₀⁰. 
The first goal of the algorithm is to find out that the removal of 
(Zeus, hasGender, masculine) between 𝐾₁₀₀ and 𝐾₁₀⁰ (as part of the replacement) 
may be part of a relevant past correction. We 
call this deletion a correction seed. Formally, a correction seed is a 
deletion (∅, M⁻) or an addition (M⁺, ∅) such that (i) there exists 
0 ≤ 𝑖 ≤ p − 1 such that 𝐾ᵢ₊₁ = (𝐾ᵢ ⊆ M⁺ᵢ) \ M⁻ᵢ with M⁻ᵢ = M⁻.
We collect the patterns for the constraint \( \Gamma \) which is not in \( \mathcal{D} \) (Correction rule): dataset and the KB history. Relevant past corrections (the PCDataset, exemplified in Table 2). The previous algorithm has given us a list of \( \{\langle \text{masculine} \rangle, \langle \text{Zeus} \rangle \} \) addition, which is why such constraints have only deletion patterns. Note that it is not possible to solve a consistency constraint with an addition pattern, and each atom that occurs in the rewriting of the head of a constraint corresponds to a deletion pattern, and each atom that occurs in the rewriting of the body of a constraint corresponds to a deletion pattern. We store this information as a tuple in the Constraints(\( \Gamma \)) instance to be relevant for the current TBox, computing the correction seed patterns can be done via query rewriting of the CQs in the body of \( \gamma \) and the head of \( \gamma \) of the constraint w.r.t. \( \mathcal{T} \). Indeed, if \( \mathcal{T} \) is a flat QL TBox, any CQ \( q(\gamma) \) can be rewritten w.r.t. \( \mathcal{T} \) into a UCQ \( q(\hat{\gamma}) \) such that for every ABox \( \mathcal{A} \), answering \( q(\hat{\gamma}) \) over \( \mathcal{T} \cup \mathcal{A} \) amounts to answering \( q(\gamma) \) over \( \mathcal{A} \) [23]. Each atom that occurs in the rewriting of the body of a constraint corresponds to a deletion pattern, and each atom that occurs in the rewriting of the head of a completeness constraint corresponds to an addition pattern. We collect the patterns for the constraint \( \Gamma \) in the set Patterns(\( \Gamma \)). Note that it is not possible to solve a consistency constraint with an addition, which is why such constraints have only deletion patterns.

The second step of the algorithm verifies, for each correction seed, whether it solved some constraint violation in the past – i.e., whether \( \mathcal{K}_1 \) contains some violations of some constraint instances that are not in \( \mathcal{K}_{i+1} \). If so, the modification between \( \mathcal{K}_1 \) and \( \mathcal{K}_{i+1} \) is a solution that solved these violations in \( \mathcal{K}_1 \). In the example we would have found the violation \( \{\langle \text{masculine} \rangle, \langle \text{Zeus} \rangle \} \) of \( \mathcal{I}_0(\text{masculine}) \) in \( \mathcal{K}_{100} \) which is not in \( \mathcal{K}_{101} \). So we would have extracted that \( \{\langle \text{Zeus} \rangle, \text{hasGender, masculine} \} \) is a solution that solved the violation \( \{\langle \text{masculine} \rangle, \langle \text{Zeus} \rangle \} \) of \( \mathcal{I}_0(\text{masculine}) \) in \( \mathcal{K}_{100} \). We store this information as a tuple in the Relevant past corrections dataset (the PCDataset), as shown in Table 2. Finding the constraint instances violated in \( \mathcal{K}_0 \) or \( \mathcal{K}_{i+1} \) is done via CQ answering (Proposition 1), and computing their violations amounts to computing BCJ justifications (Proposition 2).

The final step of the algorithm removes corrections that have been reversed. The result is thus the set of relevant past corrections.

### 6.2 Correction Rule Mining

**Correction rules.** The previous algorithm has given us a list of relevant past corrections (the PCDataset, exemplified in Table 2). We now present our approach to mine correction rules from this dataset and the KB history.

**Definition 8 (Correction rule):** A correction rule is of the form 
\[
\text{r := } [\Gamma(\bar{x})] : E(\bar{x}, \bar{y}, \bar{z}) \rightarrow (M^+(\bar{x}, \bar{y}), M^-(\bar{x}, \bar{y})), \text{ where}
\]
- \( \Gamma(\bar{x}) \) is a constraint that can be partially instantiated, i.e., some of its variables have been replaced by constants,
- \((M^+(\bar{x}, \bar{y}), M^-(\bar{x}, \bar{y}))\) is a pair of sets at most one triple,
- \( E(\bar{x}, \bar{y}, \bar{z}) \) is a set of atoms called the context of the violation such that \( M^-(\bar{x}, \bar{y}) \subseteq \bar{E}(\bar{x}, \bar{y}, \bar{z}) \),

and both \((M^+(\bar{x}, \bar{y}), M^-(\bar{x}, \bar{y}))\) and \( E(\bar{x}, \bar{y}, \bar{z}) \) are built from \( N_R \cup N_T \cup \{ \text{rdf:type} \} \cup N_1 \cup \mathcal{R} \cup \mathcal{Z} \).

A correction rule can be applied to a KB \( \mathcal{K} \) when there exist tuples of constants \( \bar{a}, \bar{b} \) such that \( \mathcal{K} \) violates \( \Gamma(\bar{a}) \) (recall that this can be decided via CQ answering by Proposition 1) and \( \mathcal{K} \models \exists \bar{a}E(\bar{a}, \bar{b}, \bar{z}) \). The result of the rule application is then \((M^+(\bar{a}, \bar{b}), M^-(\bar{a}, \bar{b}))\).

Note that while the variables from \( \bar{E}(\bar{x}, \bar{y}, \bar{z}) \) can be rewritten w.r.t. \( \mathcal{K} \), if so, the modification \( r \) of the partially instantiated \( \bar{E}(\bar{x}, \bar{y}, \bar{z}) \) could express how to solve a violation of \( \Gamma(\bar{a}) \) that appeared in \( \mathcal{K} \). In particular, these \( \bar{a} \) that do not appear in \( \Gamma(\bar{x}) \) or in the head of \( r \) can be existentially quantified, those that occur in the head of \( r \) have to be free: they have to be mapped to individuals occurring in the KB in order to construct the result.

**Example 6:** In our running example, we would like to learn the following correction rules:

\[
\begin{align*}
\text{r}_1 & := [\mathcal{I}_0(\text{masculine})] : \{\text{hasGender}(y, \text{masculine})\} \\
& \quad \rightarrow (\text{hasGender}(y, \text{male}), \text{hasGender}(y, \text{masculine})) \\
\text{r}_2 & := [\mathcal{I}_2(x)] : \{\text{hasMother}(y, x), \text{Human}(y)\} \\
& \quad \rightarrow (\text{Human}(x), 0)
\end{align*}
\]

The context of the second rule says that if \( x \) is the mother of a human, then \( x \) must also be a human. The rule obtained by replacing Human by Animal would express how to solve a violation of \( \mathcal{I}_2 \) in the context where \( y \) is an animal.

**Mining correction rules.** We mine correction rules with Algorithm 2. This algorithm is an adaptation of the algorithm in [13, 14] to our context, where we learn rules not from a KB but from the PCDataset and the KB history. We first adapt the definitions of the confidence and support from [13, 14] to our case. The support of the body of a correction rule \( r \) for a constraint \( \Gamma \) is the number of violations of \( \Gamma \) stored in the PCDataset that could have been corrected by applying \( r \). Such violations are associated with an instance \( \Gamma(\bar{a}) \) of the partially instantiated \( \Gamma(\bar{x}) \) that appears in \( r \) and with an index \( i \) such that \( \mathcal{K}_i \models \exists \bar{a}E(\bar{a}, \bar{b}, \bar{z}) \) for some \( \bar{b} \). These two conditions imply that \( r \) could be applied to the KB \( \mathcal{K}_i \). Moreover, we need to check that the result of applying \( r \) to \( \mathcal{K}_i \) actually gives a solution to \( \mathcal{V} \). For example, consider the case where the PCDataset contains both \( \langle \emptyset, R(a, b) \rangle, \langle \Gamma(a), [R(a, b), A(a), i] \rangle \) and \( \langle \emptyset, R(a, c) \rangle, \langle \Gamma(a), [R(a, c), A(a), j] \rangle \) for \( \Gamma(a) : \exists x R(a, x) \land A(a) \rightarrow false \). Both violations count for the support of the body of \( \Gamma(a) \) : \( R(a, x) \rightarrow (\emptyset, R(a, x)) \) but only the second one counts for the support of the body of \( \Gamma(a) : R(a, c) \rightarrow (\emptyset, R(a, c)) \), even in the case where \( \mathcal{K}_i \models R(a, c) \). Formally, \( \text{sup}_{\text{body}}(r) = |BSup| \) where

\[
BSup = \{ \mathcal{V} | \exists \bar{a} \Gamma(\bar{a}), \mathcal{V}, i \in \text{PCDataset}, \exists \bar{b} \mathcal{K}_i \models \exists \bar{a}E(\bar{a}, \bar{b}, \bar{z}) \}
\]
and the result of the application of \( r \) to \( \mathcal{K}_i \) is a solution to \( \mathcal{V} \).
The support of the rule \( r \) measures when the past correction is exactly the result of the application of the rule in the cases where it could be applied. Formally, \( \text{sup}_{\text{rule}}(r) = |\text{RSup}| \), where

\[
\text{RSup} = \{ V \mid ((M^+(\bar{a}, \bar{b}), M^-(\bar{a}, \bar{b})), \Gamma(\bar{a}), V, i) \in \text{PCDataset}, \text{and } K_i \models \exists \exists E(\bar{a}, \bar{b}, \bar{z}) \}.
\]

Finally, the confidence of a correction rule \( r \) is \( \text{conf}(r) = \frac{\text{sup}_{\text{rule}}(r)}{\text{sup}_{\text{rule}}(\theta)} \).

**Algorithm 2 Correction rule mining**

**Input:** PCDataset, \((K_i)_{0 \leq i < p}, \text{minsup}, \text{minconf}, \theta\)

**Output:** correction rules

// Generate basic rules
BasicR := ∅

for all \(((M^+(\bar{a}, \bar{b}), M^-(\bar{a}, \bar{b})), \Gamma(\bar{a}), V, i) \in \text{PCDataset} \)
\( r_0 := [\Gamma(\bar{a})] : M^-(\bar{a}, \bar{b}) \rightarrow (M^+(\bar{a}, \bar{b}), M^-(\bar{a}, \bar{b})) \)

BasicR := BasicR ∪ \{ \{ σ(r_0) \mid C \subseteq \bar{a} \cup \bar{b}, σ : C \rightarrow \text{Var}, \text{sup}_{\text{rule}}(σ(r_0)) \geq \text{minsup}, \text{conf}(σ(r_0)) \geq \text{minconf} \} \}

// Refine the context part of the rules

\( q := [], q.\text{enqueueAll}(\text{BasicR}) \)

while \( q \) is not empty do
\( r := q.\text{dequeue()} \)

Output \( r \)

for all operators \( \text{op} \) do

for all \( r' \in \text{op}(r) \) do

if \( \text{sup}_{\text{rule}}(r') \geq \text{minsup} \) and \( \text{conf}(r') \geq \text{conf}(r) + \theta \) then
\( q.\text{enqueue}(r') \)

Applying correction rules. When all rules have been mined, they are sorted by decreasing confidence, breaking ties by help of the support (as it is done in [26] to build classifiers from rules). This set of rules then forms a program that can be used to fix constraint violations as follows. Given a violation \( V \) of a constraint \( \Gamma \) in \( K \), choose the first rule \( r \) in the program that is relevant for \( \Gamma \) (i.e., that contains \([\Gamma(\bar{x})] \) where \( \Gamma(\bar{x}) \) is a partially instantiated version of \( \Gamma \)). Then check whether \( r \) can be applied to \( V \). The correction is the result of the rule application.

**Example 7:** Assume we mined the rules \( r_1 \) and \( r_2 \) of the preceding example with confidence 0.9 and 0.8 respectively, and another rule \( r_3 := [\Gamma_0(\bar{x})] : (\text{hasGender}(\bar{x}, y)) \rightarrow (\emptyset, \text{hasGender}(\bar{x}, y)) \) with confidence 0.5. The correction program is \( (r_1, r_2, r_3) \). To correct a violation of \( \Gamma_0 \), i.e. a wrong value for the \text{hasGender} property, the program first checks whether \( r_1 \) is applicable. If so, it replaces masculine by male. Otherwise, it falls back to \( r_3 \) and removes the wrong value. To correct a violation of \( \Gamma_2 \), it ignores \( r_1 \) that is not related to \( \Gamma_2 \) and either applies \( r_2 \) if the context matches or does nothing.

### 7 EXPERIMENTS ON WIKIDATA

This section describes CorHist, which implements the framework introduced for Wikidata, and presents its experimental evaluation.

#### 7.1 Wikidata

Wikidata is a generalist collaborative knowledge base. The project started in 2012, and as of July 2018, it has collected more than 500M statements about 50M entities. The data about each entity is stored in a versioned JSON blob, and there are more than 700M revisions. Wikidata encodes facts not in plain RDF triples but in a reified representation, in which each main triple can be annotated with qualifiers and provenance information [38].

Wikidata knows the property \text{instanceOf} which is similar to rdf:type. It does not have a formally defined TBox, but knows properties such as \text{subClassOf}, \text{subPropertyOf}, and \text{inverseOf}. However, only the property \text{subClassOf} is used to flag the constraint violations. Therefore, we use only this property in our TBox, which thus contains simple concept inclusions.

We consider the set \( C \) of constraints built from ten types of Wikidata property constraints (see Table 3). They are the top Wikidata property constraints that can be expressed in DL, covering the majority of the most used constraints, as well as 71% of Wikidata property constraints. The remaining constraints are mainly about string format validation with regular expressions (52% of the remaining constraints) and about qualifiers (31% of them).

#### 7.2 Dataset Construction

We stored the RDF version [10, 18] of the Wikidata edit history in an RDF quad store. We used named graphs for the global state of Wikidata after each revision, and for the triple additions and deletions. Our dataset stores 390M annotated triples about 49M items extracted from the July 1st, 2018 full database dump.

We extracted the relevant past corrections as explained in Section 6.1. Wikidata revisions do not correspond exactly to atomic modifications in our sense. For example, Wikidata bots are able to change multiple unrelated facts about the same entity at the same
Table 3: Wikidata property constraints. $R$ is the property for which the constraint is given. A constraint has several lines when it uses a property whose set of values may be specified or not. $\#\text{constr.}$ is the total number of constraints of the given type in Wikidata. $\#\text{triples}$ is the sum for all these constraints of the numbers of triples with the property $R$ on which they apply. $\#\text{violations}$ is the number of violations for this constraint in Wikidata on July 1st, 2018. $\#\text{past cor.}$ is the number of past corrections we extracted from Wikidata history. t.o. indicates that we were not able to extract all past corrections because of timeout so that we sample them (we then indicate the number of corrections we extracted).

<table>
<thead>
<tr>
<th>Name in Wikidata</th>
<th>DL form</th>
<th>Rule form</th>
<th>$#\text{constr.}$</th>
<th>$#\text{triples}$</th>
<th>$#\text{violations}$</th>
<th>$#\text{past cor.}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 4</td>
<td>$\exists R \subseteq A_1 \sqcup \cdots \sqcup A_n$</td>
<td>$\forall y \exists x (y, x) \rightarrow A_1(x) \vee \cdots \vee A_n(x)$</td>
<td>2575</td>
<td>249M</td>
<td>3465k</td>
<td>t.o.(&gt;16M)</td>
</tr>
<tr>
<td>Value type 4</td>
<td>$\exists R' \subseteq A_1 \sqcup \cdots \sqcup A_n$</td>
<td>$\forall y \exists x (y, x) \rightarrow A_1(x) \vee \cdots \vee A_n(x)$</td>
<td>696</td>
<td>67M</td>
<td>3062k</td>
<td>t.o.(&gt;19M)</td>
</tr>
<tr>
<td>One-of</td>
<td>$\exists R \subseteq {a_1, \ldots, a_n}$</td>
<td>$\forall y \exists x (y, x) \rightarrow x = a_1 \vee \cdots \vee x = a_n$</td>
<td>104</td>
<td>3.6M</td>
<td>4k</td>
<td>14k</td>
</tr>
<tr>
<td>Item requires</td>
<td>$\exists R \subseteq \exists R', {a_1, \ldots, a_n}$</td>
<td>$\forall y \exists x (y, x) \rightarrow R'(x, a_1) \vee \cdots \vee R'(x, a_n)$</td>
<td>3102</td>
<td>255M</td>
<td>3710k</td>
<td>t.o.(&gt;15M)</td>
</tr>
<tr>
<td>Statement</td>
<td>$\exists R \subseteq \exists R'$</td>
<td>$\forall y \exists x (y, x) \rightarrow \exists z R'(x, z)$</td>
<td>243</td>
<td>85M</td>
<td>1345k</td>
<td>t.o.(&gt;6M)</td>
</tr>
<tr>
<td>Value requires</td>
<td>$\exists R' \subseteq R \subseteq \exists R'$</td>
<td>$\forall y \exists x (y, x) \rightarrow R'(x, a_1) \vee \cdots \vee R'(x, a_n)$</td>
<td>243</td>
<td>85M</td>
<td>1345k</td>
<td>t.o.(&gt;6M)</td>
</tr>
<tr>
<td>Conflict with</td>
<td>$\exists R \subseteq \neg R' \cdot {a_1, \ldots, a_n}$</td>
<td>$\forall y \exists x (y, x) \land (R'(x, a_1) \vee \cdots \vee R'(x, a_n)) \rightarrow false$</td>
<td>601</td>
<td>449M</td>
<td>142k</td>
<td>465k</td>
</tr>
<tr>
<td>Inverse/Symmetric</td>
<td>$R \subseteq R^-$</td>
<td>$R(x, y) \rightarrow R'(y, x)$</td>
<td>146</td>
<td>6M</td>
<td>409k</td>
<td>2989k</td>
</tr>
<tr>
<td>Single value</td>
<td>(func $R$)</td>
<td>$R(x, y) \land R(z, z) \rightarrow y = z$</td>
<td>2772</td>
<td>85M</td>
<td>334k</td>
<td>389k</td>
</tr>
<tr>
<td>Distinct values</td>
<td>(func $R^-$)</td>
<td>$R(y, x) \land R(z, x) \rightarrow y = z$</td>
<td>2728</td>
<td>56M</td>
<td>189k</td>
<td>7432k</td>
</tr>
</tbody>
</table>

For each constraint. In practice, it affects only the most frequent 0.9% of Type, 2% of Value type, 0.5% of Item requires statement, and 3% of Value requires statement constraints.

7.3 Mining Rules

The output of our method is a set of correction rules that form a program (Section 6.2). To evaluate such a program, we apply it to each of the constraint violations stored in the PCDataset, using the associated stage of the KB to evaluate the part of the context which is not the deletion part of the correction. Then we check whether the correction we compute is exactly the same as the one associated to the constraint violation in the PCDataset. The precision $p$ of the program is given by the fraction of the corrections computed by the program that are actually the same as those that have been applied. The recall $r$ of the program is the fraction of the constraint violations stored in PCDataset for which the program gives some correction. The F1 score $f_1$ is $f_1 = 2\frac{pr}{p+r}$.

CorHist mines rules as explained in Section 6.2. In order to decrease the computation time, we only allow one atom $p(x, o)$ in $E(x, y, z) \setminus Body(x, y)$, where $Body(x, y)$ corresponds to the part of the context that matches part of the constraint body, such that $s$ is a variable of $x \cup y$ and $o$ is a fresh variable or a constant.

Rules were mined per constraint. For each constraint, we split the set of extracted past corrections into a 70% training set, a 10% cross-validation set, and a 20% test set. The training set is used to mine the rules, the cross-validation set is used to determine the confidence threshold that maximizes the F1 score of the obtained program, and the test set is used to evaluate the final program.

Table 4 gives examples of rules mined by CorHist. Several of these rules show the crucial importance of the instantiation of the constraint and/or of the context to be able to choose the correction. For instance, the rule for the Single value constraint uses the fact that an entity involved in a property “member of sport team” is

---

1 The Wikidata constraint Type can be qualified to modify its meaning. We ignore these cases, which are marginal: they concern less than 6% of the Type constraints. The same goes analogously for Value type.
2 Inverse and Symmetric are two distinct kinds of constraints in Wikidata but we treat them together since Symmetric is actually a special case of Inverse.
probably a human being, and thus that if it has several values for the functional property “sex or gender” and one of them is a value reserved for non-human organisms in Wikidata, this value is possibly wrong. In the same vein, the rule for the Item requires statement constraint recognizes that an entity has a heritage designation that is specific to Sweden (“monument in Formminnesregistret”), to conclude that its country is Sweden. The rules also propose fixes to misused predicates: in Wikidata, the property “manner of death” is intended for the general circumstances of a person’s death (such as “accident”), while the property “cause of death” is intended to give more precise causes (such as “traffic accident”).

Table 4: Example of mined rules.

<table>
<thead>
<tr>
<th>Constr. type</th>
<th>Constraint ( \Gamma )</th>
<th>Correction rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>( \exists \text{filmplID} \sqsubseteq \exists \text{country} )</td>
<td>( [\Gamma(s)] : \text{sexOrGender}(s, \text{maleOrg}) \land \text{sportsTeam}(s, v) \to (\emptyset, \text{sexOrGender}(s, \text{maleOrg})) )</td>
</tr>
<tr>
<td>Value type</td>
<td>( \exists \text{foundInTax} \sqsubseteq \exists \text{Taxon} )</td>
<td>( [\Gamma(\text{human})] : \text{foundInTax}(s, \text{human}) \land \text{hasPart}(s, v) \to (\emptyset, \text{foundInTax}(s, \text{human})) )</td>
</tr>
<tr>
<td>One-of</td>
<td>( \exists \text{mannerDeath} \subseteq { \ldots } )</td>
<td>( [\Gamma(\text{trafficAcc})] : \text{mannerDeath}(s, \text{trafficAcc}) \to (\text{causeDeath}(s, \text{trafficAcc}), \text{mannerDeath}(s, \text{trafficAcc})) )</td>
</tr>
<tr>
<td>Item req. stm.</td>
<td>( \exists \text{presidence} \sqsubseteq \exists \text{Country} )</td>
<td>( [\Gamma(s)] : \text{heritageStatus}(s, \text{monumentInFormminnesregistret}) \to (\text{country}(s, \text{Sweden}), \emptyset) )</td>
</tr>
<tr>
<td>Val. req. stm.</td>
<td>( \exists \text{inflpID} \sqsubseteq \neg \exists \text{filmplID} )</td>
<td>( [\Gamma(s)] : \text{filmplID}(s, o) \to (\emptyset, \text{filmplID}(s, o)) )</td>
</tr>
<tr>
<td>Inv./Sym.</td>
<td>( \exists \text{geneticAssoc} \sqsubseteq \exists \text{geneticAssoc}^{-} )</td>
<td>( [\Gamma(s)] : \text{sexOrGender}(s, \text{maleOrg}) \land \text{sportsTeam}(s, v) \to (\emptyset, \text{sexOrGender}(s, \text{maleOrg})) )</td>
</tr>
<tr>
<td>Single val.</td>
<td>( \text{func sexOrGender} )</td>
<td>( [\Gamma(s)] : \text{sexOrGender}(s, \text{maleOrg}) \land \text{sportsTeam}(s, v) \to (\emptyset, \text{sexOrGender}(s, \text{maleOrg})) )</td>
</tr>
<tr>
<td>Distinct val.</td>
<td>( \text{func ncbiLocusTag} )</td>
<td>( [\Gamma(s)] : \text{ncbiLocusTag}(o, s) \land \text{molecularFunction}(o, v) \to (\emptyset, \text{ncbiLocusTag}(o, s)) )</td>
</tr>
</tbody>
</table>

Table 5: Evaluation of the correction rules mined by CorHist with a minimal support of 10, a minimal confidence between 0.3 and 1 and a regularization threshold of 0.05, and comparison with the baselines. Best precision and F1 scores in bold.

| Constraint type | Micro average | | Macro average | |
|-----------------|--------------|-----------------|-----------------|
|                 | Prec. | Rec. | F1 | Prec. | Rec. | F1 |
| Type \(^6\)     | add     | 0.35 | 0.33 | 0.34 | 0.29 | 0.55 | 0.38 |
|                 | delete   | 0.04 | 1   | 0.07 | 0.13 | 1   | 0.23 |
|                 | CorHist  | 0.84 | 0.70 | 0.76 | 0.91 | 0.39 | 0.55 |
| Value type \(^6\) | add | 0.09 | 0.21 | 0.13 | 0.33 | 0.58 | 0.42 |
|                 | delete   | 0.01 | 0.02 | 0.16 | 0.16 | 1   | 0.27 |
|                 | CorHist  | 0.81 | 0.61 | 0.70 | 0.89 | 0.51 | 0.65 |
| One-of          | delete   | 0.26 | 1   | 0.42 | 0.35 | 1   | 0.52 |
|                 | CorHist  | 0.77 | 0.83 | 0.80 | 0.95 | 0.37 | 0.53 |
| Item requires statement \(^6\) | delete | 0.017 | 1 | 0.033 | 0.10 | 1 | 0.19 |
|                 | CorHist  | 0.94 | 0.35 | 0.51 | 0.95 | 0.52 | 0.47 |
| Value requires statement \(^6\) | add | 0.44 | 20e-6 | 41e-6 | 0.33 | 0.12 | 0.18 |
|                 | delete   | 0.041 | 1 | 0.079 | 0.082 | 1 | 0.15 |
|                 | CorHist  | 0.91 | 0.53 | 0.67 | 0.94 | 0.37 | 0.53 |
| Conflict with   | delete   | 0.39 | 1 | 0.56 | 0.39 | 1 | 0.56 |
|                 | CorHist  | 0.92 | 0.55 | 0.69 | 0.91 | 0.41 | 0.57 |
| Inverse/Sym.    | add     | 0.91 | 1 | 0.95 | 0.77 | 1 | 0.86 |
|                 | delete   | 0.072 | 1 | 0.12 | 0.14 | 1 | 0.24 |
|                 | CorHist  | 0.92 | 1 | 0.96 | 0.90 | 0.84 | 0.87 |
| Single value    | delete   | 0.34 | 1 | 0.51 | 0.42 | 1 | 0.59 |
|                 | CorHist  | 0.95 | 0.093 | 0.17 | 0.96 | 0.078 | 0.14 |
| Distinct values | delete   | 0.036 | 1 | 0.070 | 0.42 | 1 | 0.59 |
|                 | CorHist  | 0.99 | 0.020 | 0.039 | 0.93 | 0.12 | 0.21 |

\(^6\)Computed from a sample of the set of relevant past corrections. One Type constraint, six Value type constraints and one Val. req. stmt constraint were omitted due to time-out.

\(^7\)The actual value is greater than 0.995 and rounded to 1 for consistency.
We have introduced the problem of learning how to fix constraint violations from a KB history. We have also presented a method based on rule mining to this end. Our experimental evaluation on Wikidata shows significant improvement over baselines. Our tool is live on Wikidata and has already allowed users to correct more than 23k constraint violations. While our evaluation focused on Wikidata for which the whole edit history was available, we believe that our method can be applied in other settings, for example using edits done during the partial cleaning of an automatically extracted KB.

For future work, it would be interesting to evaluate the impact of parameters such as the size of the context part of the correction rule in terms of rule quality. We also plan to extend the learning dataset with external knowledge (such as other KBs), or with information extracted from other sources (for instance from Wikipedia). We believe that this will allow finding even more precise correction rules, thus making KBs ever more precise and more useful.

ACKNOWLEDGMENTS

Partially supported by the grant ANR-16-CE23-0007-01 (“DICOS”).