HAL open science

Learning How to Correct a Knowledge Base from the Edit History

Thomas Pellissier Tanon, Camille Bourgaux, Fabian M. Suchanek

To cite this version:

Thomas Pellissier Tanon, Camille Bourgaux, Fabian M. Suchanek. Learning How to Correct a Knowledge Base from the Edit History. World Wide Web Conference, May 2019, San Francisco, United States. 10.1145/3308558.3313584 . hal-02066041

HAL Id: hal-02066041

https://imt.hal.science/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:
Thomas Pellissier Tanon, Camille Bourgaux, and Fabian Suchanek. 2019. Learning How to Correct a Knowledge Base from the Edit History. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3308558.3313584

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., imposing 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 order 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),[1]
- a correction rule mining algorithm, together with an implementation for Wikidata, CorHist,[2]
- a suggestion tool for users to correct data based on our mined correction rules,[3]
- 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

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.
WWW '19, May 13–17, 2019, San Francisco, CA, USA
© 2019 IW3C2 (International World Wide Web Conference Committee), published under Creative Commons CC-BY 4.0 License.
ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313584

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

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 logic 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 then interact with the user to find out how to update the KB. The goal is then to minimize the number of questions the user has to answer. This is done in various ways, which include taking into account the dependencies among the facts to check or the interaction between several constraints violations to define heuristics to choose the best question to ask the user [2, 3, 5, 6].

Other approaches to improve the quality of a KB rely on statistics, clustering, or structural aspects of the KBs. The work of [31] uses statistics to add missing types to the KB, and to detect wrong statements. The work of [25] exploits the observation that cycles in the KB often contain wrong "IsA" relations. Again other approaches [1] use crowdsourcing to detect Linked Data quality issues. We refer the reader to Section 7.2 of [1] for a recent overview of approaches for data quality assessment.

Our method also exploits KB constraints. However, it differs from the above in that it *learns the corrections automatically* from the edit history. It thus taps a source of knowledge that has so far not been exploited.

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 $\mathsf{N_C}$ of *concept names* (unary predicates, also called classes), a set $\mathsf{N_R}$ of *role names* (binary predicates, also called properties), and a set $\mathsf{N_I}$ 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 $A \in \mathsf{N_C}$, $R \in \mathsf{N_R}$, $a, b \in \mathsf{N_I}$. A *TBox* (ontology) is a set of axioms whose form depends on the DL $\mathcal{L}$ in question, and expresses relationships between concepts and roles (e.g., concept or role hierarchies, role domains and ranges...). A *knowledge base* (KB) $\mathcal{K} = \mathcal{T} \cup \mathcal{A}$ is the union of an ABox $\mathcal{A}$ and a TBox $\mathcal{T}$.

In this work, we assume that $\mathcal{T}$ is a *flat QL* TBox [23], i.e., that $\mathcal{L}$ differs from the standard *RDF Schema (RDFS)* [17] only by allowing inverse roles in role inclusions. More precisely, $\mathcal{T}$ can contain concept inclusions of the form $A_1 \sqsubseteq A_2$ (subclass), $\exists P \sqsubseteq A$ (domain or range), and role inclusions $P_1 \sqsubseteq P_2$ (subproperty), where $A_{(i)} \in \mathsf{N_C}$ and $P_{(i)} := R \mid R^-$ with $R \in \mathsf{N_R}$.

A KB can also be written as a set of *RDF triples* $\langle s, p, o\rangle$ 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 $\langle a, \texttt{rdf:type}, A\rangle$, and a role assertion $R(a, b)$ as $\langle a, R, b\rangle$. Flat QL TBox axioms can also be represented by single triples. For example, $A_1 \sqsubseteq A_2$ is written as $\langle A_1, \texttt{rdfs:subClassOf}, A_2\rangle$ and $\exists R^- \sqsubseteq A$ is written as $\langle R, \texttt{rdfs:range}, A\rangle$.

Semantics. We recall the standard semantics of DL KBs. An *interpretation* has the form $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta^\mathcal{I}$ is a non-empty set and $\cdot^\mathcal{I}$ is a function that injectively maps each $a \in \mathsf{N_I}$ to $a^\mathcal{I} \in \Delta^\mathcal{I}$ (unique name assumption), $\top$ to $\Delta^\mathcal{I}$, each $A \in \mathsf{N_C}$ to $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$, and each $R \in \mathsf{N_R}$ to $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$. The function $\cdot^\mathcal{I}$ is straightforwardly extended to general concepts and roles, e.g. $(\neg B)^\mathcal{I} = \Delta^\mathcal{I} \setminus B^\mathcal{I}$, $(R^-)^\mathcal{I} = \{(c, d) \mid (d, c) \in R^\mathcal{I}\}$, $\{a_1, \ldots, a_n\} = \{a_1^\mathcal{I}, \ldots, a_n^\mathcal{I}\}$, $(\exists P \cdot B)^\mathcal{I} = \{c \mid \exists d : (c, d) \in P^\mathcal{I}, d \in B^\mathcal{I}\}$, $(B_1 \sqcap B_2)^\mathcal{I} = B_1^\mathcal{I} \cap B_2^\mathcal{I}$, $(B_1 \sqcup B_2)^\mathcal{I} = B_1^\mathcal{I} \cup B_2^\mathcal{I}$. An interpretation $\mathcal{I}$ *satisfies* an inclusion $G \sqsubseteq H$, if $G^\mathcal{I} \subseteq H^\mathcal{I}$; it satisfies an axiom (func P) if $P^\mathcal{I}$ is functional; it satisfies an axiom (trans P) if $P^\mathcal{I}$ is transitive; and it satisfies $A(a)$ (resp. $R(a, b)$), if $a^\mathcal{I} \in A^\mathcal{I}$ (resp. $(a^\mathcal{I}, b^\mathcal{I}) \in R^\mathcal{I}$). We write $\mathcal{I} \models \alpha$ if $\mathcal{I}$ satisfies the DL axiom α.

An interpretation $\mathcal{I}$ is a *model* of $\mathcal{K} = \mathcal{T} \cup \mathcal{A}$ if $\mathcal{I}$ satisfies all axioms in $\mathcal{K}$. A KB is *consistent* if it has a model. A KB $\mathcal{K}$ *entails* a DL axiom α if $\mathcal{I} \models \alpha$ for every model $\mathcal{I}$ of $\mathcal{K}$.

Queries. A *conjunctive query* (CQ) takes the form $q(\vec{x}) = \exists \vec{y} \psi(\vec{x}, \vec{y})$, where ψ 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

$\vec{x} \cup \vec{y}$. If $\vec{x} = \emptyset$, q is a Boolean CQ (BCQ). A BCQ q is *satisfied* by an interpretation $\mathcal{I}$, written $\mathcal{I} \models q$, if there is a homomorphism π mapping the variables and individual names of q into $\Delta^{\mathcal{I}}$ such that: $\pi(a) = a^{\mathcal{I}}$ for every $a \in \mathsf{N_I}$, $\pi(t) \in A^{\mathcal{I}}$ for every concept atom $A(t)$ in ψ, $(\pi(t), \pi(t')) \in R^{\mathcal{I}}$ for every role atom $R(t, t')$ in ψ, and $\pi(t) = \pi(t')$ for every $t = t'$ in ψ. 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 $\vec{a}$ is a (certain) *answer* to a CQ $q(\vec{x})$ if $\mathcal{K} \models q(\vec{a})$ where $q(\vec{a})$ is the BCQ obtained by replacing the variables from $\vec{x}$ by the constants $\vec{a}$. We denote by answers$(q(\vec{x}), \mathcal{K})$ the set of answers of $q(\vec{x})$ 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* $\mathcal{I}_{\mathcal{K}}$ such that for every BCQ q, $\mathcal{K} \models q$ iff $\mathcal{I}_{\mathcal{K}} \models q$. The domain of $\mathcal{I}_{\mathcal{K}}$ is the set of individual names that occur in $\mathcal{K}$ and $A^{\mathcal{I}_{\mathcal{K}}} = \{a \mid \mathcal{A} \models B(a), \mathcal{T} \models B \sqsubseteq A\}$ for every $A \in \mathsf{N_C}$ and $R^{\mathcal{I}_{\mathcal{K}}} = \{(a, b) \mid \mathcal{A} \models P(a, b), \mathcal{T} \models P \sqsubseteq R\}$ for every $R \in \mathsf{N_R}$ [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 CQ 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:*

$P := R \mid R^-$

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

where $R \in \mathsf{N_R}$, $A \in \mathsf{N_C}$, $a_1, \ldots, a_n \in \mathsf{N_I}$.

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

A completeness constraint *is a concept inclusion of the form* $B_1 \sqsubseteq B_2$, *a role inclusion of the form* $P_1 \sqsubseteq P_2$, *or a transitivity axiom of the form* (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 $B_1 \sqcup B_2$ 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 $B_1 \sqsubseteq \neg B_2$. For example, we assume that $\exists P \cdot (B_1 \sqcup B_2) \sqsubseteq C$ is rewritten as $\exists P \cdot B_1 \sqsubseteq C, \exists P \cdot B_2 \sqsubseteq C$, and $B \sqsubseteq \neg \exists P \cdot \{a_1, \ldots, a_n\}$ is rewritten as $B \sqsubseteq \neg \exists P \cdot \{a_1\}, \ldots, B \sqsubseteq \neg \exists P \cdot \{a_n\}$. 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 a (completeness constraints).*

$\mathcal{T} = \{$ Human $\sqsubseteq$ Person, Deity $\sqsubseteq$ Person $\}$

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

$C = \{ \Gamma_0 = \exists \mathsf{hasGender}^- \sqsubseteq \{male, female, nonbinary\},$

$\Gamma_1 = \exists \mathsf{hasMother} \sqsubseteq \mathsf{Person} \sqcup \mathsf{Animal},$

$\Gamma_2 = \exists \mathsf{hasMother}^- \sqsubseteq \mathsf{Person} \sqcup \mathsf{Animal},$

$\Gamma_3 = \exists \mathsf{hasMother}^- \sqsubseteq \exists \mathsf{hasGender} \cdot \{female\},$

$\Gamma_4 = \mathsf{hasMother} \sqsubseteq \mathsf{hasChild}^- \}$ ◁

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

Example 2: *In our running example, the KB* $\mathcal{K}$ *satisfies* Γ_1 *since* $\exists \mathsf{hasMother}^{\mathcal{I}_{\mathcal{K}}} = \{Zeus, Spinoza\}$ *and* $\mathcal{I}_{\mathcal{K}} \models \mathsf{Person}(Zeus)$ *and* $\mathcal{I}_{\mathcal{K}} \models \mathsf{Person}(Spinoza)$. *However, it violates* Γ_2 *because* $\mathcal{I}_{\mathcal{K}} \not\models \mathsf{Person}(Marques) \vee \mathsf{Animal}(Marques)$ *while Marques* $\in \exists \mathsf{hasMother}^{-\mathcal{I}_{\mathcal{K}}}$. *It violates* Γ_3 *and* Γ_4 *for similar reasons. Finally,* $\{\mathsf{hasGender}(Zeus, masculine), \Gamma_0\}$ *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,* $\mathcal{I}_{\mathcal{K}}$ *cannot be a model of* Γ_0. *Thus,* $\mathcal{K}$ *violates* Γ_0. ◁

Note the semantic difference between the constraints and the axioms of the TBox: The axiom (Human $\sqsubseteq$ 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 CQ answering. For this purpose, we use a function π, which maps each constraint $\Gamma \in C$ to a rule of the form $\exists \vec{y} \varphi(\vec{x}, \vec{y}) \rightarrow \exists \vec{z} \varphi'(\vec{x}, \vec{z})$. 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:*

$\Gamma_0(x) : \exists y \mathsf{hasGender}(y, x) \rightarrow x = male \vee x = female \vee x = nonbinary$

$\Gamma_1(x) : \exists y \mathsf{hasMother}(x, y) \rightarrow \mathsf{Person}(x) \vee \mathsf{Animal}(x)$

$\Gamma_2(x) : \exists y \mathsf{hasMother}(y, x) \rightarrow \mathsf{Person}(x) \vee \mathsf{Animal}(x)$

$\Gamma_3(x) : \exists y \mathsf{hasMother}(y, x) \rightarrow \exists z (\mathsf{hasGender}(x, z) \wedge z = female)$

$\Gamma_4(x, y) : \mathsf{hasMother}(x, y) \rightarrow \mathsf{hasChild}(y, x)$ ◁

Table 1: Translation of DL axioms into rules. Variables that appear in the right column and not in the left one are fresh.

$\pi(\top, x)$	=	true
$\pi(A, x)$	=	$A(x)$
$\pi(\{a_1, \ldots, a_n\}, x)$	=	$x = a_1 \vee \cdots \vee x = a_n$
$\pi(R, x, y)$	=	$R(x, y)$
$\pi(R^-, x, y)$	=	$\pi(R, y, x)$
$\pi(B_1 \sqcap B_2, x)$	=	$\pi(B_1, x) \wedge \pi(B_2, x)$
$\pi(B_1 \sqcup B_2, x)$	=	$\pi(B_1, x) \vee \pi(B_2, x)$
$\pi(\exists P \cdot B, x)$	=	$\exists y(\pi(P, x, y) \wedge \pi(B, y))$
$\pi(B \sqsubseteq C)$	=	$\pi(B, x) \rightarrow \pi(C, x)$
$\pi(P \sqsubseteq Q)$	=	$\pi(P, x, y) \rightarrow \pi(Q, x, y)$
$\pi(B \sqsubseteq \neg C)$	=	$\pi(B, x) \wedge \pi(C, x) \rightarrow$ false
$\pi(P \sqsubseteq \neg Q)$	=	$\pi(P, x, y) \wedge \pi(Q, x, y) \rightarrow$ false
$\pi(\text{func } P)$	=	$\pi(P, x, y) \wedge \pi(P, x, z) \rightarrow y = z$
$\pi(\text{trans } P)$	=	$\pi(P, x, y) \wedge \pi(P, y, z) \rightarrow \pi(P, x, z)$

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

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, $b(\vec{x}) := \exists \vec{y} \varphi(\vec{x}, \vec{y})$ 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 $\text{answers}(q(x), \mathcal{K}) = B^{\mathcal{I}_\mathcal{K}}$ (resp. $\text{answers}(q(x, y), \mathcal{K}) = P^{\mathcal{I}_\mathcal{K}}$). If Γ is a completeness constraint of the form $B_1 \sqsubseteq B_2$ (resp. $P_1 \sqsubseteq P_2$), or a consistency constraint of the form $B \sqsubseteq \{a_1, \ldots, a_n\}$, the result follows immediately since $\mathcal{K}$ satisfies Γ iff $\mathcal{I}_\mathcal{K} \models \Gamma$. If Γ is a consistency constraint of the form $B_1 \sqsubseteq \neg B_2$ (resp. $P_1 \sqsubseteq \neg P_2$), $\text{answers}(b(\vec{x}), \mathcal{K}) = B_1^{\mathcal{I}_\mathcal{K}} \cap B_2^{\mathcal{I}_\mathcal{K}}$ (resp. $\text{answers}(b(\vec{x}), \mathcal{K}) = P_1^{\mathcal{I}_\mathcal{K}} \cap P_2^{\mathcal{I}_\mathcal{K}}$) and is empty iff Γ is satisfied. Since in this case $h(\vec{x}) :=$ false, $\text{answers}(h(\vec{x}), \mathcal{K}) = \emptyset$, and the desired relation holds. If Γ 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), $P^{\mathcal{I}_\mathcal{K}}$ is functional iff $\text{answers}(b(x, y, z), \mathcal{K}) \subseteq \text{answers}(h(x, y, z), \mathcal{K})$. Finally, if Γ is of the form (trans P), $b(x, y, z) := \pi(P, x, y) \wedge \pi(P, y, z)$ and $h(x, y, z) := \pi(P, x, z)$, and it is easy to see that $P^{\mathcal{I}_\mathcal{K}}$ is transitive iff $\text{answers}(b(x, y, z), \mathcal{K}) \subseteq \text{answers}(h(x, y, z), \mathcal{K})$. □

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

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

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

Example 4: *In our running example, it is easy to see that the subset $\mathcal{V}_0 = \{\text{hasGender}(\textit{Zeus}, \textit{masculine})\}$ is a violation of Γ_0. Consider now $\mathcal{V} = \{\text{hasMother}(\textit{Spinoza}, \textit{Marques})\}$. $\mathcal{V}$ is a violation of Γ_2, Γ_3 and Γ_4. Indeed, it violates $\Gamma_2(\textit{Marques})$, $\Gamma_3(\textit{Marques})$, and $\Gamma_4(\textit{Spinoza}, \textit{Marques})$ and $\mathcal{K}$ does not satisfy any of these constraint instances. However, even if $\mathcal{V}$ violates $\Gamma_1(\textit{Spinoza})$, $\mathcal{V}$ is not a violation of Γ_1 because $\{\text{Human}(\textit{Spinoza}), \text{Human} \sqsubseteq \text{Person}\} \subseteq \mathcal{K}$ satisfies the head of $\Gamma_1(\textit{Spinoza})$.* ◁

If a constraint instance $\Gamma(\vec{a})$ is violated, $\vec{a} \in \text{answers}(b(\vec{x}), \mathcal{K})$ and $\vec{a} \notin \text{answers}(h(\vec{x}), \mathcal{K})$, so its violations are the minimal subsets of $\mathcal{K}$ responsible for $\vec{a} \in \text{answers}(b(\vec{x}), \mathcal{K})$. 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 $\Gamma(\vec{a}) : b(\vec{a}) \rightarrow h(\vec{a})$, a subset $\mathcal{V} \subseteq \mathcal{K}$ is a violation of $\Gamma(\vec{a})$ iff $\mathcal{V}$ is a justification of $\mathcal{K} \models b(\vec{a})$.*

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 $m = (\mathcal{M}^+, \mathcal{M}^-)$ of two sets of assertions or $\mathcal{L}$-axioms that takes one of the following forms:*
Addition: $m = (\{\langle s, p, o\rangle\}, \emptyset)$, *where $\langle s, p, o\rangle \notin \mathcal{K}$*
Deletion: $m = (\emptyset, \{\langle s, p, o\rangle\})$, *where $\langle s, p, o\rangle \in \mathcal{K}$*
Replacement: $m = (\{\langle s, p, o\rangle\}, \{\langle s', p', o'\rangle\})$, *where $\langle s, p, o\rangle \notin \mathcal{K}$, $\langle s', p', o'\rangle \in \mathcal{K}$, and $\langle s, p, o\rangle$ differs from $\langle s', p', o'\rangle$ in exactly one component.*

Thus, an atomic modification consists of two sets $\mathcal{M}^+$ and $\mathcal{M}^-$, each of which is either the empty set or a singleton set. $\mathcal{M}^+$ will be added to the KB, and $\mathcal{M}^-$ 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 $\langle s, p, o\rangle$ simply by $(\langle s, p, o\rangle, \emptyset)$). 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 $\Gamma(\vec{a})$ in $\mathcal{K}$ is an atomic modification $(\mathcal{M}^+, \mathcal{M}^-)$ such that there exists $\mathcal{K}' \subseteq \mathcal{K}$ such that $(\mathcal{V} \cup \mathcal{K}' \cup \mathcal{M}^+) \setminus \mathcal{M}^-$ satisfies $\Gamma(\vec{a})$. We call $(\mathcal{M}^+, \mathcal{M}^-)$ a solution to $\mathcal{V}$ for $\Gamma(\vec{a})$ in $\mathcal{K}$.*

Note that $\Gamma(\vec{a})$ can still be violated in $(\mathcal{K} \cup \mathcal{M}^+) \setminus \mathcal{M}^-$ if $\mathcal{K}$ contains other violations of $\Gamma(\vec{a})$ for which $(\mathcal{M}^+, \mathcal{M}^-)$ is not a solution. For example, if $\Gamma(a) : \exists x R(a, x) \wedge A(a) \rightarrow$ false, and $\mathcal{K} = \{R(a, b), R(a, c), A(a)\}$, the deletion of $R(a, b)$ is a solution to the violation $\{R(a, b), A(a)\}$, but $\{R(a, c), A(a)\}$ 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* $(\emptyset, \text{hasGender}(\textit{Zeus}, \textit{masculine}))$ *and the replacement* $(\text{hasGender}(\textit{Zeus}, \textit{male}),$

hasGender(*Zeus*, *masculine*)) *are two possible solutions to* $\mathcal{V}_0$ *for* Γ_0(*masculine*).

The deletion ($\emptyset$, hasMother(*Spinoza*, *Marques*)) *is a solution to* $\mathcal{V}$ *for the three constraint instances* Γ_2(*Marques*), Γ_3(*Marques*) *and* Γ_4(*Spinoza*, *Marques*). *The additions* (Human(*Marques*), $\emptyset$), (hasGender(*Marques*, *female*), $\emptyset$) *and* (hasChild(*Marques*, *Spinoza*), $\emptyset$) *are solutions to* $\mathcal{V}$ *for respectively* Γ_2(*Marques*), Γ_3(*Marques*) *and* Γ_4(*Spinoza*, *Marques*). ◁

Good solutions. Our goal is to find "good" solutions to constraint violations, i.e., solutions that make the KB as close to the real world as possible. The basic requirement for a "good" solution is that it deletes only erroneous facts, and that it adds only true facts. We also prefer replacements over deletions as long as they fulfill this condition. For instance, in our running example, the replacement (hasGender(*Zeus*, *male*), hasGender(*Zeus*, *masculine*)) is better than the deletion ($\emptyset$, hasGender(*Zeus*, *masculine*)), because it corrects erroneous information instead of simply erasing it.

In some cases, there may be no "good" solution that consists of a single atomic modification. Consider for example a completeness constraint of the form $A \sqsubseteq \exists P \cdot B$ violated by $\{A(a)\}$. If $A(a)$ is true, we should actually add both $P(a, b)$ and $B(b)$ for some b. We choose to define solutions as atomic modifications nevertheless to simplify the problem by reducing the size of possible solutions. This is a limitation of our approach since we will not be able to learn solutions that are not atomic. However, we will still be able to learn to add $B(b)$ to solve the aforementioned constraint violation in the case where $P(a, b)$ is already present.

The main difficulty in finding good solutions to constraint violations is that we do not have access to an oracle that knows the validity of all facts. This is the problem that all KB cleaning approaches face (cf. Section 2). Our idea is to exploit the history of the KB modifications to learn how to correct constraint violations.

Definition 5 (Edit history): *The* edit history *of a KB is a sequence of KBs* $(\mathcal{K}_i)_{0 \leq i \leq p} = (\mathcal{T}_i \cup \mathcal{A}_i)_{0 \leq i \leq p}$ *such that* $\mathcal{K}_{i+1} = (\mathcal{K}_i \cup \mathcal{M}_i^+) \setminus \mathcal{M}_i^-$, *where* $(\mathcal{M}_i^+, \mathcal{M}_i^-)$ *is an atomic modification.*

The edit history allows us to pinpoint how constraint violations have been corrected in the past. In order to avoid learning from vandalism or mistakes, we consider only those corrections that have not been reversed:

Definition 6 (Past correction): *A* past correction *is a solution* $(\mathcal{M}^+, \mathcal{M}^-)$ *to a violation* $\mathcal{V}$ *of a constraint instance* $\Gamma(\vec{a})$ *in* $\mathcal{K}_i$ *such that there exist* $\mathcal{B}$ *and* $\mathcal{D}$ *such that the current KB* $\mathcal{K}_p = (\mathcal{K}_i \cup \mathcal{B}) \setminus \mathcal{D}$ *with* $\mathcal{M}^+ \subseteq \mathcal{B}$, $\mathcal{M}^- \subseteq \mathcal{D}$, $\mathcal{M}^+ \cap \mathcal{D} = \emptyset$, $\mathcal{M}^- \cap \mathcal{B} = \emptyset$.

Intuitively, $(\mathcal{B}, \mathcal{D})$ corresponds to the sequence of additions and deletions that leads from $\mathcal{K}_i$ to the current state of the KB $\mathcal{K}_p$, that contains the solution, and that does not "undo" it.

Relevant past corrections. During the history of a KB, users can change not just the assertions of the KB, but also the TBox. However, the TBox is typically much smaller and more stable than the ABox. Therefore, the edit history of the TBox is not a rich ground for correction rule mining. Moreover, we are interested in learning solutions that correct constraint violations in the *current* KB $\mathcal{K}_p$. We thus consider only those past corrections that would have been corrections also under the current TBox. For example, assume that the TBox contained $C \sqsubseteq B$. Assume that $C(a)$ was added to correct a violation of the constraint $A \sqsubseteq B$. If, in the meantime, the inclusion $C \sqsubseteq B$ 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 $(\mathcal{M}^+, \mathcal{M}^-)$ *to a violation* $\mathcal{V}$ *of a constraint instance* $\Gamma(\vec{a})$ *in* $\mathcal{K}_i$ *is a past correction such that (i)* $\mathcal{M}^+ \cup \mathcal{M}^-$ *contains only assertions, and (ii)* $(\mathcal{V} \cap \mathcal{A}_i) \cup \mathcal{T}_p$ *contains a violation* $\mathcal{V}'$ *of* $\Gamma(\vec{a})$ *such that* $(\mathcal{M}^+, \mathcal{M}^-)$ *is also a solution to* $\mathcal{V}'$ *in* $\mathcal{A}_i \cup \mathcal{T}_p$.

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 $\mathcal{C}$, current TBox $\mathcal{T}_p$, history $(\mathcal{K}_i)_{0 \leq i \leq p}$
Output: set of relevant past corrections PCDataset

```
// Construct correction seed patterns
for all Γ ∈ C such that Γ(x⃗) : b(x⃗) → h(x⃗) do
    Patterns(Γ) := {(_, A(x⃗)) | A(x⃗)∈b′(x⃗), b′(x⃗)∈rewrite(b(x⃗), T_p)}
    if Γ is a completeness constraint then
        Patterns(Γ) ∪= ({(A(x⃗), _) | A(x⃗) ∈ h′(x⃗),
                                h′(x⃗) ∈ rewrite(h(x⃗), T_p)})
// Extract past corrections
for 0 ≤ i ≤ p − 1 do
    if (M_i^+, M_i^−) such that K_{i+1} = (K_i ∪ M_i^+) \ M_i^− matches
    some pattern in Patterns(Γ) then
        PCDataset ∪= {⟨(M_i^+, M_i^−), Γ(a⃗), V, i⟩ |
                V ∈ Violations(K_i, Γ(a⃗)) \ Violations(K_{i+1}, Γ(a⃗))}
// Remove reversed past corrections
for ⟨(M_i^+, M_i^−), Γ(a⃗), V, i⟩ ∈ PCDataset do
    if M_i^+ ⊈ K_p or M_i^− ∩ K_p ≠ ∅ then
        PCDataset \= {⟨(M_i^+, M_i^−), Γ(a⃗), V, i⟩}
```

Let us explain our algorithm with our running example. Consider the constraint $\Gamma_0(x) : \exists y \text{hasGender}(y, x) \rightarrow x = male \vee x = female \vee x = nonbinary$. Assume that $\langle Zeus, \text{hasGender}, masculine \rangle$ was added between $\mathcal{K}_1$ and $\mathcal{K}_2$, but then replaced by $\langle Zeus, \text{hasGender}, male \rangle$ between $\mathcal{K}_{100}$ and $\mathcal{K}_{101}$.

The first goal of the algorithm is to find out that the removal of $\langle Zeus, \text{hasGender}, masculine \rangle$ between $\mathcal{K}_{100}$ and $\mathcal{K}_{101}$ (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 $(\emptyset, \mathcal{M}^-)$ or an addition $(\mathcal{M}^+, \emptyset)$ such that (i) there exists $0 \leq i \leq p - 1$ such that $\mathcal{K}_{i+1} = (\mathcal{K}_i \cup \mathcal{M}_i^+) \setminus \mathcal{M}_i^-$ with $\mathcal{M}_i^- = \mathcal{M}^-$

Table 2: Dataset PCDataset **of relevant past corrections extracted for our running example.**

Relevant past correction	Constraint instance	Violation	KB index
$(\{\langle Zeus, \mathsf{hasGender}, male\rangle\}, \{\langle Zeus, \mathsf{hasGender}, masculine\rangle\})$	$\Gamma_0(masculine)$	$\{\langle Zeus, \mathsf{hasGender}, masculine\rangle\}$	100

(resp. $\mathcal{M}_i^+ = \mathcal{M}^+$) and (ii) there exists a KB $\mathcal{T}_p \cup \mathcal{D}$, where $\mathcal{D}$ is a set of assertions, such that $\mathcal{T}_p \cup \mathcal{D}$ contains a violation $\mathcal{V}$ of some constraint instance and $(\emptyset, \mathcal{M}^-)$ (resp. $(\mathcal{M}^+, \emptyset)$) is a solution to $\mathcal{V}$. Looking for correction seeds instead of computing the constraint violations for all constraints on all KB versions has the advantage of significantly reducing the search space.

To find such correction seeds efficiently, the first step of the algorithm precomputes for each constraint a set of atomic modification patterns that the possible correction seeds would match. In the example there would be only one pattern: the deletion pattern $(_, \langle ?, \mathsf{hasGender}, ?\rangle)$, where _ can be anything so that it matches both the deletion of $\langle ?, \mathsf{hasGender}, ?\rangle$ and its replacements. Since we only consider past corrections that involve assertions, and want them to be relevant for the current TBox, computing the correction seed patterns can be done via query rewriting of the CQs in the body $b(\vec{x})$ and the head $h(\vec{x})$ of the constraint w.r.t. $\mathcal{T}_p$. Indeed, if $\mathcal{T}$ is a flat QL TBox, any CQ $q(\vec{x})$ can be rewritten w.r.t. $\mathcal{T}$ into a UCQ $q'(\vec{x})$ such that for every ABox $\mathcal{A}$, answering $q(\vec{x})$ over $\mathcal{T} \cup \mathcal{A}$ amounts to answering $q'(\vec{x})$ 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 Γ in the set Patterns(Γ). 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 $\mathcal{K}_i$ contains some violations of some constraint instances that are not in $\mathcal{K}_{i+1}$. If so, the modification between $\mathcal{K}_i$ and $\mathcal{K}_{i+1}$ is a solution that solved these violations in $\mathcal{K}_i$. In the example we would have found the violation $\{\langle Zeus, \mathsf{hasGender}, masculine\rangle\}$ of $\Gamma_0(masculine)$ in $\mathcal{K}_{100}$, which is not in $\mathcal{K}_{101}$. So we would have extracted that $(\langle Zeus, \mathsf{hasGender}, male\rangle, \langle Zeus, \mathsf{hasGender}, masculine\rangle)$ is a solution that solved the violation $\{\langle Zeus, \mathsf{hasGender}, masculine\rangle\}$ of $\Gamma_0(masculine)$ 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 $\mathcal{K}_i$ or $\mathcal{K}_{i+1}$ is done via CQ answering (Proposition 1), and computing their violations amounts to computing BCQ 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*

$$r := [\Gamma(\vec{x})] : \mathcal{E}(\vec{x}, \vec{y}, \vec{z}) \rightarrow (\mathcal{M}^+(\vec{x}, \vec{y}), \mathcal{M}^-(\vec{x}, \vec{y})), \text{ where}$$

- $\Gamma(\vec{x})$ *is a constraint that can be partially instantiated, i.e., some of its variables have been replaced by constants,*
- $(\mathcal{M}^+(\vec{x}, \vec{y}), \mathcal{M}^-(\vec{x}, \vec{y}))$ *is a pair of sets of at most one triple,*
- $\mathcal{E}(\vec{x}, \vec{y}, \vec{z})$ *is a set of atoms called the* context *of the violation such that* $\mathcal{M}^-(\vec{x}, \vec{y}) \subseteq \mathcal{E}(\vec{x}, \vec{y}, \vec{z})$,

and both $(\mathcal{M}^+(\vec{x}, \vec{y}), \mathcal{M}^-(\vec{x}, \vec{y}))$ *and* $\mathcal{E}(\vec{x}, \vec{y}, \vec{z})$ *are built from* $\mathsf{N_C} \cup \mathsf{N_R} \cup \{\mathtt{rdf:type}\} \cup \mathsf{N_I} \cup \vec{x} \cup \vec{y} \cup \vec{z}$.

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

Note that while the variables from $\mathcal{E}(\vec{x}, \vec{y}, \vec{z})$ that do not appear in $\Gamma(\vec{x})$ 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:*

$$r_1 := [\Gamma_0(masculine)] : \{\mathsf{hasGender}(y, masculine)\} \rightarrow (\mathsf{hasGender}(y, male), \mathsf{hasGender}(y, masculine))$$

$$r_2 := [\Gamma_2(x)] : \{\mathsf{hasMother}(y, x), \mathsf{Human}(y)\} \rightarrow (\mathsf{Human}(x), \emptyset)$$

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* Γ_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 Γ is the number of violations of Γ stored in the PCDataset that could have been corrected by applying r. Such violations are associated with an instance $\Gamma(\vec{a})$ of the partially instantiated $\Gamma(\vec{x})$ that appears in r and with an index i such that $\mathcal{K}_i \models \exists \vec{z} \mathcal{E}(\vec{a}, \vec{b}, \vec{z})$ for some $\vec{b}$. 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 PCDataset contains both $\langle(\emptyset, R(a, b))), \Gamma(a), \{R(a, b), A(a)\}, i\rangle$ and $\langle(\emptyset, R(a, c))), \Gamma(a), \{R(a, c), A(a)\}, j\rangle$ for $\Gamma(a) : \exists x R(a, x) \wedge A(a) \rightarrow$ 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, $sup_{bod}(r) = |BSup|$ where

$$BSup = \{\mathcal{V} \mid \langle_, \Gamma(\vec{a}), \mathcal{V}, i\rangle \in \text{PCDataset}, \exists \vec{b}\, \mathcal{K}_i \models \exists \vec{z} \mathcal{E}(\vec{a}, \vec{b}, \vec{z}) \text{ and the result of the application of } r \text{ to } \mathcal{K}_i \text{ is a solution to } \mathcal{V}\}.$$

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, $sup_{rule}(r) = |RSup|$, where

$$RSup = \{\mathcal{V} \mid \langle(\mathcal{M}^+(\vec{a},\vec{b}), \mathcal{M}^-(\vec{a},\vec{b})), \Gamma(\vec{a}), \mathcal{V}, i\rangle \in \mathsf{PCDataset}, \text{ and } \mathcal{K}_i \models \exists\vec{z}\mathcal{E}(\vec{a},\vec{b},\vec{z})\}.$$

Finally, the *confidence of a correction rule r* is $conf(r) = \frac{sup_{rule}(r)}{sup_{bod}(r)}$.

Algorithm 2 Correction rule mining

Input: PCDataset, $(\mathcal{K}_i)_{0\leq i\leq p}$, *minsup*, *minconf*, θ
Output: correction rules

```
// Generate basic rules
BasicR := ∅
for all ⟨(M+(a⃗,b⃗), M−(a⃗,b⃗)), Γ(a⃗), V, i⟩ ∈ PCDataset do
    r0 := [Γ(a⃗)] : M−(a⃗,b⃗) → (M+(a⃗,b⃗), M−(a⃗,b⃗))
    BasicR ∪= {σ(r0) | C ⊆ a⃗ ∪ b⃗, σ : C ↣ Var,
                sup_rule(σ(r0)) ≥ minsup, conf(σ(r0)) ≥ minconf}
// Refine the context part of the rules
q := [], q.enqueueAll(BasicR)
while q is not empty do
    r := q.dequeue()
    Output r
    for all operators op do
        for all r′ ∈ op(r) do
            if sup_rule(r′) ≥ minsup and conf(r′) ≥ conf(r) + θ then
                q.enqueue(r′)
```

Algorithm 2 shows our mining algorithm. It takes as input the PCDataset computed by Algorithm 1, the KB history, a minimum support threshold, a minimum confidence threshold, and a regularization threshold θ. These thresholds are chosen empirically (see Section 7.3). The algorithm produces correction rules (Definition 8). For this purpose, it first generates a trivial rule r_0 for each entry of the PCDataset. This rule has as context simply the deletion part of the constraint past correction. This trivial rule is then transformed into several more general rules, which we call *basic rules*, each of which is obtained from r_0 by replacing some of the constants by variables. Formally, the algorithm uses all partial substitutions σ from constants to distinct fresh variables. It retains only those basic rules that meet the minimum support and confidence thresholds.

In the second step, the algorithm incrementally refines each rule by building up its context part $\mathcal{E}(\vec{x},\vec{y},\vec{z})$. This works similarly to the mining algorithm of [14]: Each refinement step adds one atom built from the KB concept and role names and the variables and constants that appear in the rule, plus at most one fresh variable. For this purpose, the algorithm uses the operators defined in [14]. If the resulting rule meets the minimum support threshold, and improves the confidence by at least θ, the rule is retained.

Note that Algorithm 2 outputs only rules that would have been returned by the algorithm of [13, 14] if evaluated with our confidence function. It prunes more rules because of the use of the θ and the *minconf* thresholds that are also used to do an early pruning of the rules during the context construction. Algorithm 2 can be easily parallelized by running it independently on each constraint and/or having multiple workers working on the same queue.

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 $\mathcal{V}$ of a constraint Γ in $\mathcal{K}$, choose the first rule r in the program that is relevant for Γ (i.e., that contains $[\Gamma(\vec{x})]$ where $\Gamma(\vec{x})$ is a partially instantiated version of Γ). Then check whether r can be applied to $\mathcal{V}$. 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 $r_3 := [\Gamma_0(x)] : \{\mathsf{hasGender}(x,y)\} \to (\emptyset, \mathsf{hasGender}(x,y))$ with confidence 0.5. The correction program is (r_1, r_2, r_3). To correct a violation of Γ_0, i.e. a wrong value for the* 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 Γ_2, it ignores r_1 that is not related to Γ_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 $\langle s, p, o\rangle$ triple can be annotated with qualifiers and provenance information [38].

Wikidata knows the property instanceOf which is similar to `rdf:type`. It does not have a formally defined TBox, but knows properties such as subClassOf, subPropertyOf, and inverseOf. However, only the property 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. ♯constr. is the total number of constraints of the given type in Wikidata. ♯triples is the sum for all these constraints of the numbers of triples with the property R on which they apply. ♯violations is the number of violations for this constraint in Wikidata on July 1st, 2018. ♯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).

Name in Wikidata	DL form	Rule form	♯constr.	♯triples	♯violations	♯past cor.
Type [4]	$\exists R \sqsubseteq A_1 \sqcup \cdots \sqcup A_n$	$\exists y R(x,y) \rightarrow A_1(x) \vee \cdots \vee A_n(x)$	2575	249M	3465k	t.o.(>16M)
Value type [4]	$\exists R^- \sqsubseteq A_1 \sqcup \cdots \sqcup A_n$	$\exists y R(y,x) \rightarrow A_1(x) \vee \cdots \vee A_n(x)$	696	67M	3062k	t.o.(>19M)
One-of	$\exists R^- \sqsubseteq \{a_1, \ldots, a_n\}$	$\exists y R(y,x) \rightarrow x = a_1 \vee \cdots \vee x = a_n$	104	3.6M	4k	14k
Item requires statement	$\exists R \sqsubseteq \exists R' \cdot \{a_1, \ldots, a_n\}$ $\exists R \sqsubseteq \exists R'$	$\exists y R(x,y) \rightarrow R'(x,a_1) \vee \cdots \vee R'(x,a_n)$ $\exists y R(x,y) \rightarrow \exists z R'(x,z)$	3102	255M	3710k	t.o.(>15M)
Value requires statement	$\exists R^- \sqsubseteq \exists R' \cdot \{a_1, \ldots, a_n\}$ $\exists R^- \sqsubseteq \exists R'$	$\exists y R(y,x) \rightarrow R'(x,a_1) \vee \cdots \vee R'(x,a_n)$ $\exists y R(y,x) \rightarrow \exists z R'(x,z)$	243	85M	1345k	t.o.(>6M)
Conflict with	$\exists R \sqsubseteq \neg\exists R' \cdot \{a_1, \ldots, a_n\}$ $\exists R \sqsubseteq \neg\exists R'$	$\exists y R(x,y) \wedge (R'(x,a_1) \vee \cdots \vee R'(x,a_n)) \rightarrow \text{false}$ $\exists yz R(x,y) \wedge R'(x,z) \rightarrow \text{false}$	601	449M	142k	465k
Inverse/Symmetric[5]	$R \sqsubseteq R'^-$	$R(x,y) \rightarrow R'(y,x)$	146	6M	409k	2989k
Single value	(func R)	$R(x,y) \wedge R(x,z) \rightarrow y = z$	2772	85M	334k	389k
Distinct values	(func R^-)	$R(y,x) \wedge R(z,x) \rightarrow y = z$	2728	56M	189k	7432k

time. Wikidata users also sometimes prefer to delete a statement then add another one with the same property instead of directly modifying the value, in order to clear the existing qualifiers and references. Therefore, we artificially created a replacement modification for every deletion with a neighboring addition by the same user, which shares at least two components of the triple (analogously for additions). For example, if the correction seed is the deletion of ⟨*Zeus*, hasGender, *masculine*⟩, and if this revision or a neighboring one adds ⟨*Zeus*, hasGender, *male*⟩, then we consider this a replacement. However, if the same revision added the triple ⟨*Zeus*, hasMother, *Rhea*⟩, then we would not consider this a replacement, because it does not share two components with the first one.

Since the TBox consists of simple concept inclusions and the constraint bodies contain only roles, the deletion patterns for correction seeds correspond directly to the atoms of the constraint body. In the same vein, only atoms in the head of the *Type* or *Value type* constraints need to be rewritten. To find the constraint violations solved by a correction seed, we make use of the fact that the correction seed allows us to know the constraint instance $\Gamma(\vec{a})$, and we look for matches of the constraint instance body.

To speed up the execution for the four constraint types which have the highest numbers of past corrections, *Type*, *Value type*, *Item requires statement* and *Value requires statement*, we did not extract all the past corrections but sample them as follows. We compute only the relevant past corrections that where applied between $\mathcal{K}_i$ and $\mathcal{K}_{i+1}$ where i is a multiple of $s := max(1, N/10^6)$ with N the number of triples with the property R of the constraint at hand. This sampling allows us to get a sufficient ground for rule mining 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* is $F_1 = 2\frac{p \cdot r}{p+r}$.

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

[4] 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*.

[5] *Inverse* and *Symetric* are two distinct kinds of constraints in Wikidata but we treat them together since *Symetric* is actually a special case of *Inverse*.

Table 4: Example of mined rules.

Constr. type	Constraint Γ	Correction rule
Type	$\exists$isAListOf $\sqsubseteq$ List	$[\Gamma(s)]$: isAListOf$(s, o) \wedge$ WikiDisambiguationPage$(s) \rightarrow (\emptyset,$ isAListOf$(s, o))$
Value type	$\exists$foundInTax$^- \sqsubseteq$ Taxon	$[\Gamma(\mathit{human})]$: foundInTax$(s, \mathit{human}) \wedge$ hasPart$(s, v) \rightarrow$ (foundInTax$(s, \mathit{homoSapiens})$, foundInTax$(s, \mathit{human})$)
One-of	$\exists$mannerDeath$^- \sqsubseteq \{\ldots\}$	$[\Gamma(\mathit{trafficAcc})]$: mannerDeath$(s, \mathit{trafficAcc}) \rightarrow$ (causeDeath$(s, \mathit{trafficAcc})$, mannerDeath$(s, \mathit{trafficAcc})$)
Item req. stm.	$\exists$heritageStatus $\sqsubseteq \exists$country	$[\Gamma(s)]$: heritageStatus$(s, \mathit{monumentInFornminnesregistret}) \rightarrow$ (country$(s, \mathit{Sweden}), \emptyset)$
Val. req. stm.	$\exists$residence$^- \sqsubseteq \exists$country	$[\Gamma(s)]$: diplomaticRelation$(s, v) \rightarrow$ (country$(s, s), \emptyset)$
Conflict	$\exists$filmplID $\sqsubseteq \neg\exists$filmplFilmID	$[\Gamma(s)]$: filmplID$(s, o) \rightarrow (\emptyset,$ filmplID$(s, o))$
Inv./Sym.	geneticAssoc $\sqsubseteq$ geneticAssoc$^-$	$[\Gamma(s, o)] : \rightarrow$ (geneticAssoc$(o, s), \emptyset)$
Single val.	(func sexOrGender)	$[\Gamma(s)]$: sexOrGender$(s, \mathit{maleOrg}) \wedge$ sportsTeam$(s, v) \rightarrow (\emptyset,$ sexOrGender$(s, \mathit{maleOrg}))$
Distinct val.	(func ncbiLocusTag$^-$)	$[\Gamma(s)]$: ncbiLocusTag$(o, s) \wedge$ molecularFunction$(o, v) \rightarrow (\emptyset,$ ncbiLocusTag$(o, s))$

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 probably 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 Fornminnesregistret"), 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").

7.4 Evaluation against the Test Set

Table 5 presents the results of the evaluation of the mined programs against the test set. We computed both the micro and macro average of the precision, recall and F1 score per kind of constraint. The micro average aggregates over the whole set of relevant past corrections for the given kind of constraint, whereas the macro average computes the scores for each constraint of this given kind, and then computes the average. Both numbers are important: The micro average gives more weight to correction rules that fix many violations. It thus measures the overall impact of the correction rules on the dataset. However, if few rules had a large impact, then it would be easier to formulate these rules by hand. Our method, in contrast, can also find rules that by themselves solve less violations, but together contribute a large mass of corrections. To illustrate this, we also report the macro average: It measures the average performance across different constraints.

We compare our approach with two baselines: The first one, called "delete", is the most basic one and uses the fact that all Wikidata constraint bodies contain an atom of the form $R(x, y)$ and the TBox contains only concept inclusions, so that all constraint violations contain an assertion that matches $R(x, y)$. The "delete" baseline simply deletes this assertion. For the completeness constraints we define an additional baseline, "add", which tries to add a new triple to solve the constraint violation. For *Inverse* and *Symmetric* constraints this baseline adds the missing reverse edge and performs very well. For *Item requires statement*, *Value requires statement*, *Type* and *Value type*, it adds the missing triple only if it is possible to know the expected value from the constraint rule.

[6]Computed from a sample of the set of relevant past corrections. One *Type* constraint, six *Value type* constraints and one *Val. req. stm* constraint were omitted due to time-out.
[7]The actual value is greater than 0.995 and rounded to 1 for consistency.

Table 5: Evaluation of the correction rules mined by CorHist with a minimal support of 10, a minimal confidence between 0.5 and 1 and a regularization threshold of 0.05, and comparison with the baselines. Best precision and F1 scores in bold.

		Micro average			Macro average		
Constraint type		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	1	0.02	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]	add	**0.99**	0.14	0.25	0.80	0.27	0.40
	delete	0.017	1	0.033	0.10	1	0.19
	CorHist	0.94	0.35	**0.51**	**0.95**	0.32	**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[7]	**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

For example, for *Type* constraints of the form $\exists y R(x, y) \rightarrow A(x)$ it applies the correction $(A(x), \emptyset)$. *Value type* constraints are handled in the same way. However, this baseline is not able to figure out what is the relevant addition correction for a constraint of the form $\exists y R(x, y) \rightarrow A_1(x) \vee A_2(x)$ because there is no way to know a priori if A_1 or A_2 should be added. For *Item requires statement* constraints of the form $\exists y R(x, y) \rightarrow R'(x, a)$, the "add" baseline applies $(R'(x, a), \emptyset)$ (similarly for *Value requires statement* constraints). However, it cannot find a correction if there are multiple a_i.

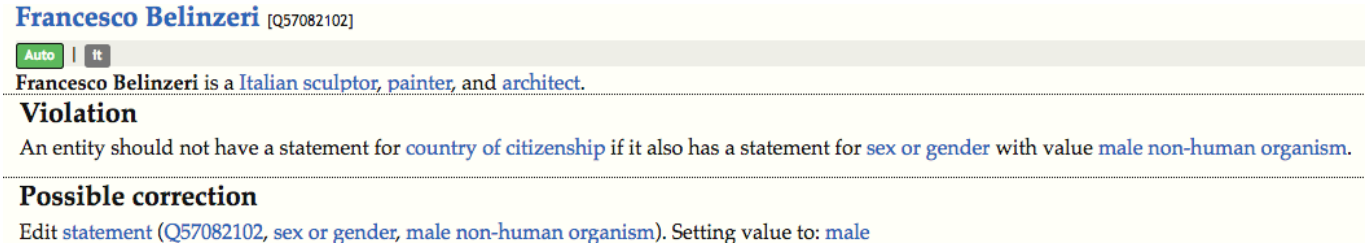

Figure 1: Example of a replacement correction suggested by CorHist for a violation of the constraint $\exists \mathsf{countryOfCitizenship} \sqsubseteq \neg\exists \mathsf{sexOrGender} \cdot \{maleOrg, femaleOrg\}$**.**

Table 6: Human evaluation of the suggested corrections.

Constraint type	Suggested	"Apply"	"Wrong"	Approval
Type	9908	252	312	0.45
Value type	2374	195	208	0.48
One-of	239	14	47	0.23
Item requires stm.	41	8	32	0.2
Value requires stm.	1024	790	178	0.82
Conflict with	3254	1717	203	0.89
Inverse/Symmetric	28138	20247	1720	0.92
Single value	3264	41	71	0.37
Distinct values	921	8	23	0.26

As shown in Table 5, the precision of our approach significantly outperforms the two baselines – often by a very high margin. Regarding the recall, we manage to keep a reasonable, and sometimes even good, recall (see best F1 scores in Table 5), except for *Single* and *Distinct value*. The very low recall obtained for these two kinds of constraints is easily explainable because they are mostly used on predicates that link Wikidata to other databases (91% of the *Single* and 95% of the *Distinct* constraints), and we cannot get meaningful information about the target database to mine corrections.

7.5 User Evaluation

To see whether our corrections are accepted by the community, we designed a user study. We created a tool that suggests our corrections to Wikidata users for validation (available at https://tools.wmflabs.org/wikidata-game/distributed/#game=43). The user can choose a constraint type, and the tool then suggests corrections for random violations of constraints of this type (Figure 1). The violations for which corrections are suggested are provided by `query.wikidata.org`, which limits their number for performance reasons. For each proposed correction, the user has to choose between three options: apply the proposed correction to Wikidata, tag it as wrong, or get another correction to review. We ran the experiment for 3 months and 47 Wikidata users participated.

Table 6 presents the results. The number of corrections reviewed is highly unbalanced between the kinds of constraints, mainly because a few users evaluate a lot of suggestions, and have a predilection for some kinds of constraints. It is thus difficult to draw conclusions for those kinds of constraints for which very few corrections have been evaluated. However, we can still make some interesting observations. In particular, the proposed corrections marked as wrong give us insights about possible weaknesses of our approach.

For the constraints which got a significant number of evaluations, our approach seems to perform well for *Inverse* and *Symmetric*, *Conflict with* and *Value requires statement* constraints, with approval rates above 80%. The other approval rates are lower. This is partly due to biases in the data. For example, when a gender is missing, our approach proposes the value "male" by default, because of the over-representation of men in Wikidata. Another issue is the quality of the constraints, which in Wikidata are sometimes questionable or difficult to understand (e.g., an incomplete set of possible types or values for completeness or *One-of* constraints).

However, even lower approval scores do not mean that our approach would be useless: Psychological research [12] shows that people find it much easier to choose from given options than to come up with an answer by themselves. The actual time needed to come up with an answer may vary, but if it takes just 3 times longer to come up with an answer than to accept or reject our proposed correction, then achieving a precision of 40% is already useful: If we have a precision of 40%, and if a free-form answer takes time t, then the expected answer time with our tool is $40\% \times \frac{1}{3} \times t + 60\% \times \frac{4}{3} \times t < t$.

8 CONCLUSION AND FUTURE WORK

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").

REFERENCES

[1] Maribel Acosta, Amrapali Zaveri, Elena Simperl, Dimitris Kontokostas, Fabian Flöck, and Jens Lehmann. 2018. Detecting Linked Data quality issues via crowdsourcing: A DBpedia study. *Semantic Web* 9, 3 (2018), 303–335. https://doi.org/10.3233/SW-160239

[2] Abdallah Arioua and Angela Bonifati. 2018. User-guided Repairing of Inconsistent Knowledge Bases. In *Proceedings of the 21th International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018*. 133–144. https://doi.org/10.5441/002/edbt.2018.13

[3] Ahmad Assadi, Tova Milo, and Slava Novgorodov. 2018. Cleaning Data with Constraints and Experts. In *Proceedings of the 21st International Workshop on the Web and Databases, Houston, TX, USA, June 10, 2018*. 1:1–1:6. https://doi.org/10.1145/3201463.3201464

[4] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider (Eds.). 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

[5] Moria Bergman, Tova Milo, Slava Novgorodov, and Wang-Chiew Tan. 2015. QOCO: A Query Oriented Data Cleaning System with Oracles. *PVLDB* 8, 12 (2015), 1900–1903. https://doi.org/10.14778/2824032.2824096

[6] Meghyn Bienvenu, Camille Bourgaux, and François Goasdoué. 2016. Query-Driven Repairing of Inconsistent DL-Lite Knowledge Bases. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*. 957–964.

[7] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. 2009. DBpedia - A crystallization point for the Web of Data. *Journal of Web Semantics* 7, 3 (2009), 154–165. https://doi.org/10.1016/j.websem.2009.07.002

[8] Iovka Boneva, José Emilio Labra Gayo, and Eric G. Prud'hommeaux. 2017. Semantics and Validation of Shapes Schemas for RDF. In *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I*. 104–120. https://doi.org/10.1007/978-3-319-68288-4_7

[9] Richard Cyganiak, David Wood, and Markus Lanthaler. 2014. RDF 1.1 Concepts and Abstract Syntax. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

[10] Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez, and Denny Vrandečić. 2014. Introducing Wikidata to the Linked Data Web. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*. 50–65. https://doi.org/10.1007/978-3-319-11964-9_4

[11] Sergio Flesca, Sergio Greco, and Ester Zumpano. 2004. Active integrity constraints. In *Proceedings of the 6th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, 24-26 August 2004, Verona, Italy*. 98–107. https://doi.org/10.1145/1013963.1013977

[12] Steven C Funk and K Laurie Dickson. 2011. Multiple-choice and short-answer exam performance in a college classroom. *Teaching of Psychology* 38, 4 (2011), 273–277.

[13] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. 2015. Fast rule mining in ontological knowledge bases with AMIE+. *VLDB J.* 24, 6 (2015), 707–730. https://doi.org/10.1007/s00778-015-0394-1

[14] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. 2013. AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*. 413–422. https://doi.org/10.1145/2488388.2488425

[15] Birte Glimm, Aidan Hogan, Markus Krötzsch, and Axel Polleres. 2012. OWL: Yet to arrive on the Web of Data?. In *WWW2012 Workshop on Linked Data on the Web, Lyon, France, 16 April, 2012*.

[16] Bernardo Cuenca Grau, Boris Motik, Zhe Wu, Ian Horrocks, Achille Fokoue, and Carsten Lutz. 2009. OWL 2 Web Ontology Language Profiles. https://www.w3.org/TR/owl2-profiles/

[17] Ramanathan Guha and Dan Brickley. 2014. RDF Schema 1.1. http://www.w3.org/TR/2014/REC-rdf-schema-20140225/

[18] Daniel Hernández, Aidan Hogan, and Markus Krötzsch. 2015. Reifying RDF: What Works Well With Wikidata?. In *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems co-located with 14th International Semantic Web Conference (ISWC 2015), Bethlehem, PA, USA, October 11, 2015*. 32–47.

[19] Vinh Thinh Ho, Daria Stepanova, Mohamed H. Gad-Elrab, Evgeny Kharlamov, and Gerhard Weikum. 2018. Rule Learning from Knowledge Graphs Guided by Embedding Models. In *The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part I*. 72–90. https://doi.org/10.1007/978-3-030-00671-6_5

[20] Joanna Józefowska, Agnieszka Lawrynowicz, and Tomasz Lukaszewski. 2010. The role of semantics in mining frequent patterns from knowledge bases in description logics with rules. *TPLP* 10, 3 (2010), 251–289. https://doi.org/10.1017/S1471068410000098

[21] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. 2007. Finding All Justifications of OWL DL Entailments. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*. 267–280. https://doi.org/10.1007/978-3-540-76298-0_20

[22] Holger Knublauch and Dimitris Kontokostas. 2017. Shapes Constraint Language (SHACL). https://www.w3.org/TR/shacl/

[23] Roman Kontchakov and Michael Zakharyaschev. 2014. An Introduction to Description Logics and Query Rewriting. In *Reasoning Web. Reasoning on the Web in the Big Data Era - 10th International Summer School 2014, Athens, Greece, September 8-13, 2014. Proceedings*. 195–244. https://doi.org/10.1007/978-3-319-10587-1_5

[24] Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. 2014. Test-driven evaluation of linked data quality. In *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*. 747–758. https://doi.org/10.1145/2566486.2568002

[25] Jiaqing Liang, Yanghua Xiao, Yi Zhang, Seung-won Hwang, and Haixun Wang. 2017. Graph-Based Wrong IsA Relation Detection in a Large-Scale Lexical Taxonomy. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*. 1178–1184.

[26] Bing Liu, Wynne Hsu, and Yiming Ma. 1998. Integrating Classification and Association Rule Mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98), New York City, New York, USA, August 27-31, 1998*. 80–86.

[27] Christian Meilicke, Manuel Fink, Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, and Heiner Stuckenschmidt. 2018. Fine-Grained Evaluation of Rule- and Embedding-Based Systems for Knowledge Graph Completion. In *The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part I*. 3–20. https://doi.org/10.1007/978-3-030-00671-6_1

[28] Boris Motik, Ian Horrocks, and Ulrike Sattler. 2009. Bridging the gap between OWL and relational databases. *J. Web Sem.* 7, 2 (2009), 74–89. https://doi.org/10.1016/j.websem.2009.02.001

[29] Boris Motik and Peter Patel-Schneider. 2009. OWL 2 Web Ontology Language Mapping to RDF Graphs. https://www.w3.org/TR/owl-mapping-to-rdf/

[30] Peter F. Patel-Schneider. 2015. Using Description Logics for RDF Constraint Checking and Closed-World Recognition. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. 247–253.

[31] Heiko Paulheim and Christian Bizer. 2014. Improving the Quality of Linked Data Using Statistical Distributions. *Int. J. Semantic Web Inf. Syst.* 10, 2 (2014), 63–86. https://doi.org/10.4018/ijswis.2014040104

[32] Christos Rantsoudis, Guillaume Feuillade, and Andreas Herzig. 2017. Repairing ABoxes through Active Integrity Constraints. In *Proceedings of the 30th International Workshop on Description Logics, Montpellier, France, July 18-21, 2017*.

[33] Viachaslau Sazonau, Uli Sattler, and Gavin Brown. 2015. General Terminology Induction in OWL. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I*. 533–550. https://doi.org/10.1007/978-3-319-25007-6_31

[34] Stefan Schlobach and Ronald Cornet. 2003. Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*. 355–362.

[35] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*. 697–706. https://doi.org/10.1145/1242572.1242667

[36] Thomas Pellissier Tanon, Daria Stepanova, Simon Razniewski, Paramita Mirza, and Gerhard Weikum. 2017. Completeness-Aware Rule Learning from Knowledge Graphs. In *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I*. 507–525. https://doi.org/10.1007/978-3-319-68288-4_30

[37] Jiao Tao, Evren Sirin, Jie Bao, and Deborah L. McGuinness. 2010. Integrity Constraints in OWL. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*.

[38] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85. https://doi.org/10.1145/2629489

[39] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2014. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. *CoRR* abs/1412.6575 (2014). arXiv:1412.6575 http://arxiv.org/abs/1412.6575

[40] Fan Yang, Zhilin Yang, and William W. Cohen. 2017. Differentiable Learning of Logical Rules for Knowledge Base Reasoning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 2316–2325.