EPiC Series in Computing
Volume 72, 2020, Pages 92–106
GCAI 2020. 6th Global Conference on Artificial Intelligence (GCAI 2020)

Guided Inductive Logic Programming: Cleaning Knowledge Bases with Iterative User Feedback

Yan Wu[1,2], Jinchuan Chen[3], Plarent Haxhidauti[2], Vinu E. Venugopal[2], and Martin Theobald[2]

[1] College of Computer and Information Science, Southwest University, China
[2] Department of Computer Science & Communications, University of Luxembourg, Luxembourg
[3] School of Information, Renmin University of China, China

Abstract

Domain-oriented knowledge bases (KBs) such as DBpedia and YAGO are largely constructed by applying a set of predefined extraction rules to the semi-structured contents of Wikipedia articles. Although both of these large-scale KBs achieve very high average precision values (above 95% for YAGO3), subtle mistakes in a few of the underlying extraction rules may still impose a substantial amount of *systematic extraction mistakes* for specific relations. For example, by applying the same regular expressions to extract person names of both Asian and Western nationality, YAGO erroneously swaps most of the family and given names of Asian person entities. For traditional rule-learning approaches based on Inductive Logic Programming (ILP), it is very difficult to detect these systematic extraction mistakes, since they usually occur only in a relatively small subdomain of the relations' arguments. In this paper, we thus propose a *guided* form of ILP, coined "GILP", that iteratively asks for small amounts of user feedback over a given KB to learn a set of data-cleaning rules that (1) best match the feedback and (2) also generalize to a larger portion of facts in the KB. We propose both algorithms and respective metrics to automatically assess the quality of the learned rules with respect to the user feedback.

1 Introduction

There are a number of recent approaches that specifically tackle the problem of learning *consistency constraints* from a given KB (or, respectively, from a fixed training subset of the KB) for data-cleaning purposes (see [21] for a recent overview). The kinds of constraints considered for data cleaning traditionally comprise functional dependencies, conditional functional dependencies [6], equality-generating dependencies [5], denial constraints and more general Horn clauses [9, 22]. The common rationale behind these learning approaches is that *"unusual implies incorrect"*. That is, constraints are established if they are followed by most data items and thus receive a high confidence according to the traditional rule-mining metrics [23]. Items violating the constraints will thus be regarded as *"unusual"* and subsequently be marked as incorrect by the data-cleaning framework. For example, if we observe that 95% of all unemployment rates are inside the range of 0 to 100 percent, we might indeed correctly conclude that *"unemployment rates should be between 0 and 100"* and remove all data items outside this range.

However, *"usual"* does not necessarily guarantee for the correctness of a data item, either. If an extraction rule is overly general or contains a logical mistake, which is quite common in practice, many incorrect tuples will be extracted from just a single such rule. For example, in YAGO3, about 44% of all unemployment rates are actually incorrect (i.e., less than 0 or larger than 100), stating, for example, that *"the unemployment rate of Italy is 2014"*. These mistakes seem to be produced by an extraction strategy which erroneously interprets years as unemployment rates. As another example, the family and given names of most Asian people have mostly been swapped in YAGO3. These mistakes are due to an extraction rule which extracts the first token of a person's name as his/her given name and conversely assigns the last token as the family name. Clearly, this extraction strategy is most likely wrong for people from Asian countries like China, Japan, Korea, etc. Hence, the idea of *"unusual implies incorrect"* cannot be applied within this particular subdomain of person entities. Unfortunately, it is very difficult to detect such kinds of systematic mistakes from the given KB only. Thus, one solution is to resort to asking for explicit user feedback, as it is shown in the following (running) example.

Example 1. *Suppose a user is browsing for people's family names and posts the feedback*[1] *depicted in Table 1. A general conclusion we could draw from the above feedback is that "all*

Fact	Feedback
hasFamilyName(Yao_Ming, Ming)	×
hasFamilyName(Li_Na, Na)	×
hasFamilyName(Lionel_Mercy, Mercy)	✓
hasFamilyName(Yi_Jianlian, Jianlian)	×

Table 1: Initial user feedback $\mathcal{F}_0$

family names are incorrect". This looks quite reasonable with respect to the initial feedback because it is consistent with most of the user comments, except for the third one. However, generalizing and applying this rule to the entire KB would obviously be rather keen at this point. A better strategy would be to first carefully generalize *and then immediately again* specialize *the rule [17] into multiple descendent rules before applying these descendent rules to clean the KB. For example, descendent rules such as "the family names of all Chinese people are incorrect", "the family names of all Japanese people are incorrect", and so on, would be much more reasonable conclusions to draw. On the other hand, the descendent rule "the family names of all American people are correct" could be derived from the positive feedback about the family name of Lionel Mercy. The respective conditions "Chinese people", "Korean people" and "American people" can be automatically extracted from the KB by joining the facts about the family names of the given person entities with the respective facts about their nationalities. To*

Fact	Feedback
hasFamilyName(Liu_Xiang, Xiang)	×
hasFamilyName(Deng_Yaping, Yaping)	×
hasFamilyName(Jimmy_Hendrix, Hendrix)	✓
hasFamilyName(Janis_Joplin, Joplin)	✓

Table 2: First iteration of user feedback $\mathcal{F}_1$

verify and apply our refined conclusions to a larger set of entities captured by the KB, we need more feedback. Thus, we choose a small amount of related facts from the KB and ask the user to comment on them. The new feedback for our running example is listed in Table 2. Generally, we may need to iteratively *learn rules and pull user feedback until a set of sufficiently qualified descendent rules is found.*

[1]User feedback is assumed to consist of binary *true*/*false* assessments of facts contained in the KB.

Our problem setting falls within the area of Inductive Logic Programming (ILP) [16], which investigates the inductive construction of first-order formulas from a given training set. However, the induction process in our problem will be *guided* by iterative user feedback, such that the training set is not fixed and the termination condition for the learning algorithm is not given a-priori. Hence, we refer to this new problem as *Guided Inductive Logic Programming* (GILP). The goal of GILP is to learn a targeted set of consistency constraints from much less training data (i.e., user feedback in our case) than what is required by traditional ILP techniques. Figure 1 illustrates the workflow of our system. First, a user browses the KB and sends an

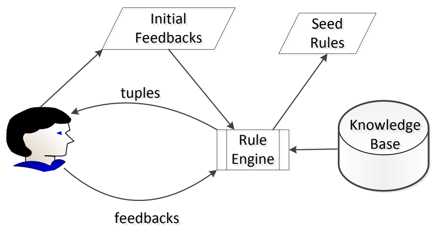

Figure 1: Iterative GILP workflow

initial, small amount of binary feedback assessments. Then, the rule engine derives a number of generic seed rules from this initial feedback by using the assessed facts as anchors. Next, the rule engine aims to refine the seed rules based on the KB. During the refinement process, the rule engine may ask the user to comment on a few additional facts in order to verify the candidate rules and thereby also learn new rules from the additional feedback. Finally, a set of qualified rules will be produced.

There are several challenges to be addressed by GILP. First of all, the number of candidate rules may grow exponentially with the number of refinement conditions attached to an initial seed rule, which limits the number of possible refinement conditions to a very small amount. Notice that GILP is a form of interactive learning, and we cannot expect the user to spend too much time on waiting between the feedback iterations. Furthermore, since each rule should be verified by pulling feedback from users, the learning process will become too laborious if large volumes of additional facts need to be assessed. Another major challenge arises due to the fact that the training set of GILP is not fixed but will expand during the learning phase. All recent works in the context of rule learning, such as [9, 18, 22], take an entire knowledge base as the training set (thus following the assumption that *"unusual implies incorrect"*) and inductively learn rules until a fixpoint condition is reached. In GILP, we have to rely on a possibly small but growing amount of user feedback to both induce and verify the candidate rules, and hence we also aim to discover systematic extraction mistakes that could not be uncovered by traditional ILP approaches. **Contributions.** We summarize the novel contributions of our work as follows.

- GILP allows users to interactively browse a KB and thereby detect systematic extraction mistakes that usually only cover a small subdomain of a predicate's arguments.
- GILP can learn targeted constraints very quickly, with just a few iterations and small amounts of user feedback at each iteration.
- GILP can learn consistency constraints that cannot be learned with standard ILP approaches that take an entire KB as their basis for learning constraints. This typically results in a higher accuracy (both in terms of precision and relative recall) for the cleaned KB than what existing ILP techniques can achieve.

2 Background & Preliminaries

In this section, we formally define our data model and introduce the basic notation that is used through the rest of the paper. Our goal is to iteratively learn consistency constraints, represented as Horn clauses, over a relational representation of a KB. We also provide a unifying approach to represent relations and constraints as first-order literals and first-order rules that capture both positive and negative feedback, respectively.

Relations. We represent a *relation schema* R by a logical predicate of arity $k \geq 1$. For a fixed *universe of constants* $\tilde{U}$, a *relation instance* $\mathbf{R}$ is a finite subset $\mathbf{R} \subseteq \tilde{U}^k$. We call the elements of $\mathbf{R}$ *tuples*, and we write $R(\bar{t})$ to refer to a tuple in $\mathbf{R}$, where $\bar{t}$ is a vector consisting of constants in $\tilde{U}$. Moreover, $R(\bar{X})$ refers to an *atomic formula* (or just "*atom*", for short) over the relation schema R, where $\bar{X}$ is a vector that may consist of both constants and variables. We write $\mathit{Var}(\bar{X})$ to denote the set of variables in $\bar{X}$. By writing $\bar{t} \restriction_X$, we denote the *restriction* (i.e., the "projection") of a tuple $\bar{t}$ onto a subset of variables $X \subseteq \mathit{Var}(\bar{X})$ (see the Appendix A for an example).

Feedback. Formally, we consider a *knowledge base* $\mathcal{K}$ as a set of tuples. We write $\mathcal{K}(R, \bar{t})$ to denote that tuple $R(\bar{t}) \in \mathcal{K}$. The *feedback* $\mathcal{F}$ for a tuple $R(\bar{t}) \in \mathcal{K}$ indicates that $R(\bar{t})$ is either *correct* or *incorrect*. Thus, the two sets $\mathcal{F}^+ \subseteq \mathcal{K}$ and $\mathcal{F}^- \subseteq \mathcal{K}$ contain all tuples that received either positive or negative feedback, respectively, where we assume that $\mathcal{F}^+ \cap \mathcal{F}^- = \emptyset$ and $\mathcal{F}^+ \cup \mathcal{F}^- = \mathcal{F}$ holds. Moreover, we write $\mathcal{T}^+ \subseteq \mathcal{K}$ and $\mathcal{T}^- \subseteq \mathcal{K}$ to denote the sets of facts, which are found to be correct or incorrect by the consistency constraints we learn over $\mathcal{K}$. In analogy to the feedback, we assume that $\mathcal{T}^+ \cap \mathcal{T}^- = \emptyset$ and $\mathcal{T}^+ \cup \mathcal{T}^- = \mathcal{T}$ holds. The embeddings of these sets within each other are illustrated in Figure 2. The key goal of our work is to learn constraints based on a limited amount of user feedback $\mathcal{F}^+$ and $\mathcal{F}^-$, while expanding $\mathcal{T}^+$ and $\mathcal{T}^-$ over the set of tuples in $\mathcal{K}$ as much as possible.

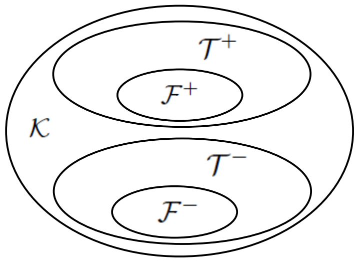

Figure 2: Embeddings of set predicates $\mathcal{F}^+$, $\mathcal{F}^-$, $\mathcal{T}^+$, $\mathcal{T}^-$ into the knowledge base $\mathcal{K}$

Set Predicates. To simplify our notation of constraints, we will focus on *Horn clauses* for the following steps (see Appendix A for a detailed definition of various classes of constraints commonly used in data cleaning). Specifically, we denote $\mathcal{K}$, $\mathcal{F}^+$, $\mathcal{F}^-$, $\mathcal{T}^+$ and $\mathcal{T}^-$ as *set predicates*, and we introduce a new binary predicate of the form $\mathcal{T}^+(R, \bar{t})$ which will serve as a short-hand notation for $\mathcal{T}^+(\bar{t}) \wedge R(\bar{t})$ (and using a similar construction for the remaining four set predicates).

Definition 1. *An* atomic formula *(or* "atom"*, for short) in our GILP framework either is of the form*

- $\mathcal{P}(R, \bar{X})$*, where* $\mathcal{P}$ *is one of the* set predicates $\mathcal{K}$*,* $\mathcal{F}^+$*,* $\mathcal{F}^-$*,* $\mathcal{T}^+$ *or* $\mathcal{T}^-$*, and* $\bar{X}$ *is a vector consisting of variables and constants;*

- *or* $(u\,\theta\,v)$*, where* θ *is a* comparison operator *(including "=", ">", "<", etc.), and* u*,* v *are either variables or constants.*

That is, a relational atom in this notation is a set predicate which has a given relation R and a vector $\bar{X}$ consisting both of variables and constants as arguments. For example, $\mathcal{T}^+(R,\bar{X})$ is an atom denoting that all constants that may become bound to the variables in $Var(\bar{X})$ are inferred to be correct by a rule, while $\mathcal{T}^-(R,\bar{X})$ denotes the set of facts that are inferred to be incorrect.

GILP Rules. The final step to define our reasoning framework is to specify which kinds of rules we allow to be learned by GILP.

Definition 2. *A* GILP rule*, denoted by* ϕ*, is a Horn clause of the form*

$$\forall x_1 \ldots \forall x_k \ \left(\varphi(\bar{X}) \rightarrow \psi(\bar{X})\right)$$

where $\varphi(\bar{X})$ *is a conjunction of atoms,* $\psi(\bar{X})$ *is a single atom (see Definition 1) and* $x_1,\ldots,x_k$ *are the variables in* $Var(\bar{X})$*.*

By using the above form of Horn clauses over set predicates, we unify all classes of constraints commonly considered for data cleaning (see Appendix A) into a single notation. Moreover, by wrapping the relational predicates into five set predicates, we are able to distinguish which of the rule atoms address facts in the given KB, which ones are derived from either positive or negative feedback facts, and which inferred facts should be considered to be either correct or incorrect. Specifically, since feedback is either positive or negative in our framework, we focus on the following two kinds of GILP rules.

Exclusive Rules. An *exclusive rule* is of the form

$$\forall x_1 \ldots \forall x_k \ \left(\varphi(\bar{X}) \rightarrow \mathcal{T}^-(R,\bar{X})\right)$$

where $x_1,\ldots,x_k$ are the variables in $Var(\bar{X})$.

Inclusive Rules. An *inclusive rule* is of the form

$$\forall x_1 \ldots \forall x_k \ \left(\varphi(\bar{X}) \rightarrow \mathcal{T}^+(R,\bar{X})\right)$$

where $x_1,\ldots,x_k$ are the variables in $Var(\bar{X})$.

Example 2. *The following inclusive rule specifies that "*The given names of all people with US nationality are correct.*"*

$$\mathcal{K}(hasGivenName, \langle x_1, y_1\rangle) \wedge \mathcal{K}(hasNationality, \langle x_1, USA\rangle) \rightarrow \mathcal{T}^+(hasGivenName, \langle x_1, y_1\rangle)$$

*Conversely, the following exclusive rule specifies that "*The given names of all people with Chinese nationality are incorrect.*"*

$$\mathcal{K}(hasGivenName, \langle x_1, y_1\rangle) \wedge \mathcal{K}(hasNationality, \langle x_1, China\rangle) \rightarrow \mathcal{T}^-(hasGivenName, \langle x_1, y_1\rangle)$$

Language Bias. Our GILP framework adopts the following *language bias* which is common in (relational) ILP settings. This ensures that the grounding procedure of the rules terminates, and it additionally helps to prune the search space of rules accepted by our GILP algorithm.

- We only allow *safe rules*, i.e., the relation R and all variables in the head of a rule ϕ must also appear in the body of ϕ. Moreover, all variables occurring as an argument of a comparison operator in ϕ must also occur as an argument of a set predicate in the body of ϕ.

- We focus on mining *connected rules*. We say that two atoms are connected iff they share a variable. A rule ϕ is connected iff every pair of atoms in the body of ϕ is transitively connected by one or more shared variables.
- We only allow *non-repeating rules*, i.e., a same atom must not occur repeatedly in each ϕ.
- Finally, to both avoid overfitting and to limit the runtime of the induction step, the rules learned by our system must not be too long, i.e., the number of atoms in the body should not exceed a predefined *maximum length* L.

3 GILP Algorithm

In this section, we outline our GILP algorithm that implements the iterative learning of rules and pulling of user feedback as it is depicted by the workflow of Figure 1. Specifically, we illustrate how to initialize our framework by generating seed rules Φ_0 from an initial set of (both positive and negative) user feedback $\mathcal{F}_0$, and we define our refinement operators to derive the overall set of candidate rules from the seed rules. Finally, we propose our basic quality measures for either accepting or rejecting these candidate rules.

3.1 Basic Algorithm

Algorithm 1 illustrates our basic GILP algorithm. We generate the set of *seed rules* Φ_0 (see Section 3.2) from the *initial user feedback* $\mathcal{F}_0$ (Line 1). The sets of *accepted rules* Φ_{acc} and *predicted facts* $\mathcal{T}$ (which will be based on all rules in Φ_{acc}) are initially set to be empty (Lines 2 & 3). Similarly, we initialize the iteratively merged sets of overall rules Φ and feedback facts $\mathcal{F}$ to Φ_0 and $\mathcal{F}_0$, respectively (Lines 4 & 5). At each GILP iteration, the set of accepted rules Φ_{acc} is selected from the current Φ based on the union of all feedback facts $\mathcal{F}$ we collected so far (Line 7). Similarly, $\mathcal{T}$ is expanded by the facts predicted by Φ_{acc} at the current iteration (Line 8). Next, we expand the set of all rules Φ by using any general ILP subroutine [9, 8] which supports the refinement operations for the kinds of constraints we wish to learn (see Section 3.3) based on the current set of feedback facts $\mathcal{F}$ over the knowledge base $\mathcal{K}$ (Line 9). Finally, pullFeedback($\mathcal{T}$) collects the next round of user feedback as a randomly chosen subset of the currently predicted facts $\mathcal{T}$ (Line 10). We terminate the GILP iterations either when no rules in Φ could initially be generated, or when $\mathcal{T}$ remains unchanged among two iterations.

```
1  $\Phi_0$ := generateSeedRules($\mathcal{F}_0$);
2  $\Phi_{acc}$ := $\emptyset$;
3  $\mathcal{T}$ := $\emptyset$;
4  $\Phi$ := $\Phi_0$;
5  $\mathcal{F}$ := $\mathcal{F}_0$;
6  while $\Phi \neq \emptyset$ or $\mathcal{T}$ changed do
7      $\Phi_{acc}$ := all rules in $\Phi$ which are accepted based on $\mathcal{F}$;
8      $\mathcal{T}$ := facts predicted by $\Phi_{acc}$;
9      $\Phi$ := $\Phi \cup$ ILP($\mathcal{F}$, $\mathcal{K}$);
10     $\mathcal{F}$ := $\mathcal{F} \cup$ pullFeedback($\mathcal{T}$);
11 return $\Phi_{acc}$;
```

Algorithm 1: GILP($\mathcal{F}_0, \mathcal{K}$)

3.2 Generating Seed Rules

Ground Seed Rules. Given the initial user feedback $\mathcal{F}_0$, we first generate a set of *ground seed rules*, one for each tuple $R(\bar{t})$ in $\mathcal{F}_0$ that received either positive or negative feedback.

- $\mathcal{F}^+(R, \bar{t}) \rightarrow \mathcal{T}^+(R, \bar{t})$: a tuple with positive feedback must be correct.
- $\mathcal{F}^-(R, \bar{t}) \rightarrow \mathcal{T}^-(R, \bar{t})$: a tuple with negative feedback must be incorrect.

Notice that in the beginning of our learning algorithm, both $\mathcal{T}^+$ and $\mathcal{T}^-$ are empty. The ground seed rules thus first of all copy the sets of tuples with either positive or negative feedback into $\mathcal{T}^+$ and $\mathcal{T}^-$, respectively. **First-Order Seed Rules.** Next, to also induce a set of *first-order seed rules* from the initial user feedback, we generalize each tuple $R(\bar{t})$ into a first-order atom $R(\bar{X})$ by replacing all constants in $\bar{t}$ with a distinct variable in $\bar{X}$ according to the relation schema R. These first-order seed rules will be the basis for our further GILP steps. The generation of seed rules from the initial user feedback is summarized in Algorithm 2. According to the language bias of our GILP framework (see Section 2), each first-order rule we learn must be both safe, connected and non-repeating. The seed rules generated by Algorithm 2 are the most specific ones (i.e., ground seed rules) and the most general ones (i.e., first-order seed rules) satisfying this language bias, respectively.

```
1  $\Phi_0 := \emptyset$;
2  for each tuple $R(\bar{t}) \in \mathcal{F}_0^+$ do
3      $\phi : \mathcal{F}^+(R, \bar{t}) \rightarrow \mathcal{T}^+(R, \bar{t})$;
4      insert $\phi$ into $\Phi_0$;
5  for each tuple $R(\bar{t}) \in \mathcal{F}_0^-$ do
6      $\phi : \mathcal{F}^-(R, \bar{t}) \rightarrow \mathcal{T}^-(R, \bar{t})$;
7      insert $\phi$ into $\Phi_0$;
8  for each relation schema $R$ such that $\exists \bar{t}\ \mathcal{F}_0^+(R, \bar{t})$ do
9      $\phi : \mathcal{F}^+(R, \bar{X}) \rightarrow \mathcal{T}^+(R, \bar{X})$;
10     insert $\phi$ into $\Phi_0$;
11 for each relation schema $R$ such that $\exists \bar{t}\ \mathcal{F}_0^-(R, \bar{t})$ do
12     $\phi : \mathcal{F}^-(R, \bar{X}) \rightarrow \mathcal{T}^-(R, \bar{X})$;
13     insert $\phi$ into $\Phi_0$;
14 return $\Phi_0$;
```

Algorithm 2: generateSeedRules($\mathcal{F}_0$)

3.3 Refinement Operators

Before we actually populate the sets $\mathcal{T}^+$ and $\mathcal{T}^-$ with facts inferred from rules in Φ, we aim to *refine* the first-order seed rules in Φ_0 to a set of candidate rules Φ that capture the user feedback as precisely as possible, while remaining general enough to also cover a large part of the underlying KB. To do so, our inductive learning approach implements a top-down search, in which a first-order seed rule $\phi \in \Phi_0$ is iteratively specialized by appending a set of atoms (see Definition 1) to the body of ϕ. In analogy to [9, 8], the following two refinement operators help to prune the search space but are also adapted to our GILP setting by starting from the particular type of seed rules and the language bias we consider in our framework.

Definition 3. *Beginning with the set of seed rules in Φ_0, we iteratively apply the following* refinement operators *to each first-order seed rule $\phi \in \Phi_0$.*

- **Add Relational Atom.** *This operator adds a new atom into ϕ, which is of the form of $R(\bar{X})$, where R is a relational predicate and $\bar{X}$ is a vector consisting of both variables and constants, such that at least one variable in $\bar{X}$ already appears in another relational atom (including the head) of ϕ.*

- **Add Comparison Atom.** *This operator adds a new atom into ϕ, which is of the form of $x\,\theta\,y$, where θ is a comparison operator. Here, either $x, y \in \tilde{V}$ are both variables that must already appear in a set predicate of ϕ, or $x \in \tilde{V}$ is a variable that must already appear in another atom of ϕ and $y \in \tilde{U}$ is a constant.*

Rules with repeating atoms are pruned immediately. We add every refined rule that is both safe and connected to the set of candidate rules Φ. Moreover, we prune the search space by dropping all candidate rules that are either not safe or not connected when we reach a maximum length of L. Only relational atoms in a rule's body are counted for the length of the rule. **Comparison Atoms with Constants.** There are a number of classical ways for constructing atoms with comparisons of variables to constants (or so-called "features" in the context of rule learning). In this paper, we adopt the one in [10]. For a numerical attribute X, such as "*Age*", we first sort all distinct values of X and then compute an intermediate value (e.g., the arithmetic mean) between each pair of successive values. We then create intervals (a, b) based on every pair of intermediate values a and b (with $a < b$). Next, we enumerate all of these intervals and add two comparison atoms $(x > a)$ and $(x < b)$ for each such range. For a categorical attribute X, such as "*Nationality*", we generate a comparison atom using the comparison operator "=", thus using $(x = a)$, for each distinct value a of X.

3.4 Quality Measures

3.4.1 Feedback-Based Coverage and Accuracy

Recall that we infer whether tuple $R(\bar{t})$ is correct or incorrect, i.e., belongs to $\mathcal{T}^+$ or $\mathcal{T}^-$, by a rule ϕ, based on whether there exist a set of tuples in $\mathcal{K}$ which can be used to ground the body of ϕ and thus imply that $R(\bar{t})$ should be inside $\mathcal{T}^+$ or $\mathcal{T}^-$, respectively. Next, we formally define an *indicator function*, $I_R(\bar{t}, \phi)$, one for each relation R, to indicate whether the correctness of $R(\bar{t})$ can be deduced by a rule ϕ of the form $\varphi(\bar{X}) \rightarrow \mathcal{T}^+(R, \bar{t})$ or $\varphi(\bar{X}) \rightarrow \mathcal{T}^-(R, \bar{t})$.

$$I_R(\bar{t}, \phi) = \begin{cases} 1 & \text{if } \exists z_1, \ldots, z_m \left(\varphi(\bar{X}) \rightarrow \mathcal{T}^+(R, \bar{t})\right) \\ -1 & \text{if } \exists z_1, \ldots, z_m \left(\varphi(\bar{X}) \rightarrow \mathcal{T}^-(R, \bar{t})\right) \\ 0 & \text{otherwise} \end{cases}$$

Here, $z_1, \ldots, z_m$ are variables in $Var(\bar{X})$. Thus, the assignments of constants to $z_1, \ldots, z_m$ imply whether $R(\bar{t}) \in \mathcal{T}^+$ or $R(\bar{t}) \in \mathcal{T}^-$, or whether $R(\bar{t})$ is in none of the two sets.

However, the actual *accuracy* of the predictions generated by a rule ϕ still needs to be verified by the user feedback $\mathcal{F} = \mathcal{F}^+ \cup \mathcal{F}^-$. There are four cases of when a tuple may be assigned to $\mathcal{T}^+$ or $\mathcal{T}^-$ with respect to $\mathcal{F}^+$ and $\mathcal{F}^-$ by a rule ϕ. These four cases of *true/false positives/negatives* very naturally translate to our feedback-based GILP framework as follows.

$$\begin{aligned} TP(\phi, \mathcal{F}) &= \{R(\bar{t}) \,|\, R(\bar{t}) \in \mathcal{F}^+ \wedge I_R(\bar{t}, \phi) = 1\} && \textit{“true positives”} \\ TN(\phi, \mathcal{F}) &= \{R(\bar{t}) \,|\, R(\bar{t}) \in \mathcal{F}^- \wedge I_R(\bar{t}, \phi) = -1\} && \textit{“true negatives”} \\ FP(\phi, \mathcal{F}) &= \{R(\bar{t}) \,|\, R(\bar{t}) \in \mathcal{F}^- \wedge I_R(\bar{t}, \phi) = 1\} && \textit{“false positives”} \\ FN(\phi, \mathcal{F}) &= \{R(\bar{t}) \,|\, R(\bar{t}) \in \mathcal{F}^+ \wedge I_R(\bar{t}, \phi) = -1\} && \textit{“false negatives”} \end{aligned}$$

Based on these four cases, we define the *coverage*, denoted $Cov(\phi, \mathcal{F})$, of a rule ϕ with respect to the feedback $\mathcal{F}$ as the union of the above four sets.

$$Cov(\phi, \mathcal{F}) = \{R(\bar{t}) \,|\, R(\bar{t}) \in \mathcal{F} \wedge I_R(\bar{t}, \phi) \neq 0\}$$

Similarly, we define the *accuracy*, denoted $Acc(\phi, \mathcal{F})$, of a rule ϕ with respect to the feedback $\mathcal{F}$ as the following ratio.

$$Acc(\phi, \mathcal{F}) = \frac{|TP(\phi, \mathcal{F})| + |TN(\phi, \mathcal{F})|}{|Cov(\phi, \mathcal{F})|}$$

Example 3. *Consider the following exclusive seed rule which rather boldly claims that "*The given names of all people are incorrect.*"*

$$\mathcal{K}(hasFamilyName, \langle x_1, y_1 \rangle) \rightarrow \mathcal{T}^{-}(hasFamilyName, \langle x_1, y_1 \rangle)$$

Based on the initial user feedback $\mathcal{F}_0$ depicted in Table 1, the coverage $Cov(\phi, \mathcal{F}_0)$ of ϕ is 4, while its accuracy $Acc(\phi, \mathcal{F}_0)$ is 3/4. Conversely, the coverage $Cov(\phi, \mathcal{F}_0 \cup \mathcal{F}_1)$ and accuracy $Acc(\phi, \mathcal{F}_0 \cup \mathcal{F}_1)$ of ϕ with respect to both the initial and first iteration of feedbacks shown in Tables 1 and 2 are 8 and 5/8, respectively.

3.4.2 Confidence Intervals for Precision

As can be seen from the above example, a major challenge in evaluating a rule's overall precision with respect to $\mathcal{K}$ is given by the lack of a fixed training set. Notice that, unlike common ILP settings, we have to rely on a small fraction of facts in $\mathcal{F}$ to be labeled as correct/incorrect, while the vast majority of facts in $\mathcal{K}$ will remain unlabeled. Moreover, the feedback set $\mathcal{F}$ will change as we pull more comments, and so does a rule's precision. Ideally, we would like to obtain comments for all tuples inside $\mathcal{K}$, in the following denoted as $\mathbb{F}$, to compute the precision of a rule with respect to $\mathcal{K}$, which is obviously infeasible.

According to the law of large numbers, however, we can *estimate* the range of $Prec(\phi, \mathbb{F})$ with high confidence. Suppose we have obtained a set of comments $\mathcal{F}$, then according to a Wilson confidence interval [20] with a confidence level of γ, the value of $Prec(\phi, \mathbb{F})$ is located within the following range.

$$Prec(\phi, \mathbb{F}) \in_{\gamma} \left[\frac{1}{1 + z^2/n} \left(\hat{p} + \frac{z^2}{2n} \pm z \sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4 * n^2}} \right) \right]$$

Here, $\hat{p}$ is an estimator for the parameter of a binomial distribution from which we repeatedly draw n random facts. That is, $\hat{p} = (|TP(\phi, \mathcal{F})| / (|TP(\phi, \mathcal{F})| + |FP(\phi, \mathcal{F})|)$, n is $|Cov(\phi, \mathcal{F})|$, and z is the $1 - \frac{1}{2}\gamma$ quantile of a standard-normal distribution that is used to approximate the binomial distribution (e.g., z is 1.96 for $\gamma = 0.95$).

4 Experimental Study

4.1 General Setup

We utilize the current dump of the core relations of YAGO3 [15] as our underlying KB, thus containing about 120 million facts with an estimated average precision of above 0.95 across all relations. As basic ILP learner, we use AMIE+ which is tailored to induce Horn clauses from a given set of training facts. Both YAGO3 and AMIE+ are openly available[2]. Since AMIE+ cannot distinguish between positive and negative examples, we call AMIE+ on two sets of expanded facts obtained from both the positive and negative feedback at each iteration, respectively. These expanded sets of facts are obtained by a depth-first search (of depth L) over the KB,

[2] https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/

which is started from both the subjects and objects contained in the feedback facts at each iteration. We remark that AMIE+ can learn facts with constants, such as *hasNationality*(x, USA) which resolves to an equality condition of the form *hasNationality*$(x, y) \wedge (y = USA)$, but it cannot learn rules with range conditions, such as *hasUnemployment*$(x, y) \wedge (y < 0.1)$, over numerical attributes, which is a basic limitation of AMIE+. Both our GILP framework as well as AMIE+ are implemented in Java 1.8.0. The experiments are run on a notebook with an Intel i7 CPU @1.80 GHz and 16 GB memory.

4.2 Tuning Parameters & Robustness

A strength of our GILP framework is that it relies on very few tuning parameters only. These are summarized in Table 3 which also lists their possible ranges of values, of which the bold ones are used as default values for the following steps. Additional AMIE+ parameters, such as the *minimum head coverage* (HC_{min}) and the *minimum standard confidence* (STD_{min}) are both kept at their default value of 0.01. The head target relation is fixed to *hasGivenName* for all of the following runs. Due to space constraints, we here only report the impact of the

Parameter	**Considered range of values**
Size of initial and further feedback sets	10, 20, 30, **40**, 50
Maximum rule length (and depth of expansion sets) L	2, **3**, 4, 5
Accuracy threshold τ to either accept or reject candidate rules	0.6, 0.7, 0.8, **0.9**

Table 3: Parameters and Possible Values.

initial size of the feedback set $|\mathcal{F}_0|$ (containing an equal amount of positive and negative facts for the head target relation) on the number of rules learned by the GILP algorithm. The results are depicted in Figure 3. It can be seen that the more initial feedback we use, the higher the number of rules we learn for each iteration. The largest difference can be observed from using 40 to 50 initial feedback facts, while the difference from using 30 to 40 initial feedback facts is much smaller. We therefore conclude that our GILP framework is fairly robust with respect to the above parameters and resort to fixing the amount of initial feedback facts to 40 for the effectiveness experiments. Figure 4 depict the number of positive and negative facts predicted

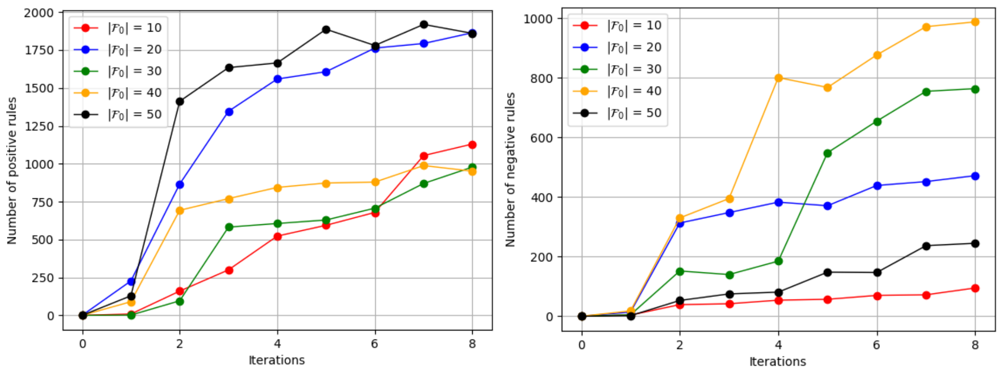

Figure 3: Rules learned for various $|\mathcal{F}_0|$ over multiple GILP iterations

for various $|\mathcal{F}_0|$ over multiple GILP iterations. We see that the number of facts predicted for both the positive and negative cases quickly increases for the first 3 iterations but then also saturates completely during the following two iterations. From this, we conclude that we indeed reached a fixpoint which serves as our stopping condition for the GILP algorithm.

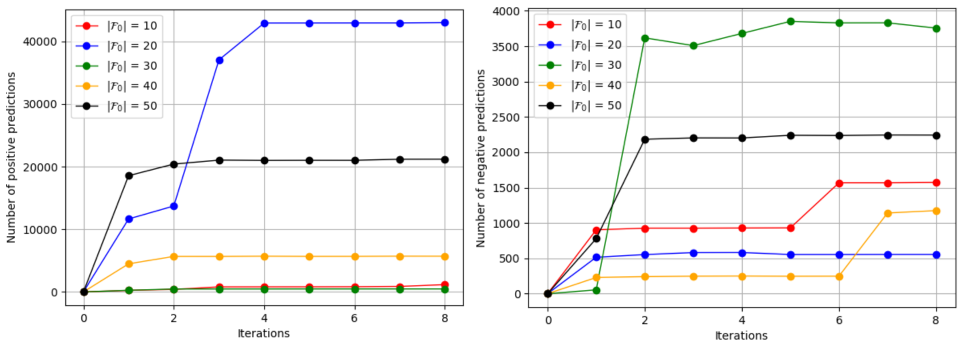

Figure 4: Facts predicted for various $|\mathcal{F}_0|$ over multiple GILP iterations

Positive rules
$hasGivenName(x, y) \wedge rdftype(x, wikicat\_American\_people) \rightarrow hasGivenName(x, y)$
$hasGivenName(x, y) \wedge rdftype(x, wikicat\_British\_people) \rightarrow hasGivenName(x, y)$
$hasGivenName(x, y) \wedge wasBornIn(x, England) \wedge isCitizenOf(x, England) \rightarrow hasGivenName(x, y)$
$hasGivenName(x, y) \wedge isPoliticianOf(x, England) \wedge isCitizenOf(x, England) \rightarrow hasGivenName(x, y)$
$hasGivenName(x, y) \wedge wasBornIn(x, United\_States) \wedge diedIn(x, United\_States) \rightarrow hasGivenName(x, y)$
Negative rules
$hasGivenName(x, y) \wedge rdftype(x, wikicat\_Chinese\_people) \rightarrow hasGivenName(x, y)$
$hasGivenName(x, y) \wedge rdftype(x, wikicat\_Vietnamese\_people) \rightarrow hasGivenName(x, y)$
$hasGivenName(x, y) \wedge rdftype(x, wikicat\_Singaporean\_people) \rightarrow hasGivenName(x, y)$
$hasGivenName(x, y) \wedge wasBornIn(x, China) \wedge isCitizenOf(x, China) \rightarrow hasGivenName(x, y)$
$hasGivenName(x, y) \wedge wasBornIn(x, China) \wedge diedIn(x, China) \rightarrow hasGivenName(x, y)$
$hasGivenName(x, y) \wedge isAffiliatedTo(x, Communist\_Party\_of\_China) \rightarrow hasGivenName(x, y)$
$hasGivenName(x, y) \wedge isCitizenOf(x, China) \wedge hasFamilyName(x, y) \rightarrow hasGivenName(x, y)$

Table 4: Example rules learned by GILP

4.3 Effectiveness

Table 4 lists some of the rules learned by GILP. Since the accuracy threshold τ is set as 0.9, as estimated over the user feedback at each iteration, the achieved precision of the learned rules is very high (see also Figure 5). GILP is able to learn many meaningful rules and thereby perform many correct, both positive and negative, predictions. Note that due to the usage of the set predicates $\mathcal{K}$ in the body of the rules and either $\mathcal{T}^+$ (positive predictions) or $\mathcal{T}^-$ (negative predictions) in the head of the rules, the actual target predicate *hasGivenName* may occur in both the body and the head of the rules. Positive rules thus reinforce existing facts in the KB, while negative rules indicate which facts should be cleaned from the KB.In Figure 5, we provide a detailed analysis of the Wilson intervals (at $\gamma = 0.95$) we obtain as an estimate for the precision of the learned rules over $\mathcal{T}^+$ (positive predictions) and $\mathcal{T}^-$ (negative predictions) at each iteration (using $|\mathcal{F}_0| = 40$). To do so, we randomly sample and manually assess $n = 100$ facts from $\mathcal{T}^+$ and $\mathcal{T}^-$, respectively, at each iteration and remember their assessments also for the next iterations, thereby collecting $n = 200, 300, ...$ facts for the second, third and following iterations. We see that the intervals not only become tighter due to the increasing amount of assessments, but they also indicate a strongly increased precision of the predictions over the following iterations due to the increased amount of feedback we collect at each iteration. After three to eight iterations, the means of the precision intervals converge to about 0.91 for $\mathcal{T}^+$ and even to 0.94 for $\mathcal{T}^-$, which confirms that our iterative GILP approach is particularly suitable for pruning negative facts.

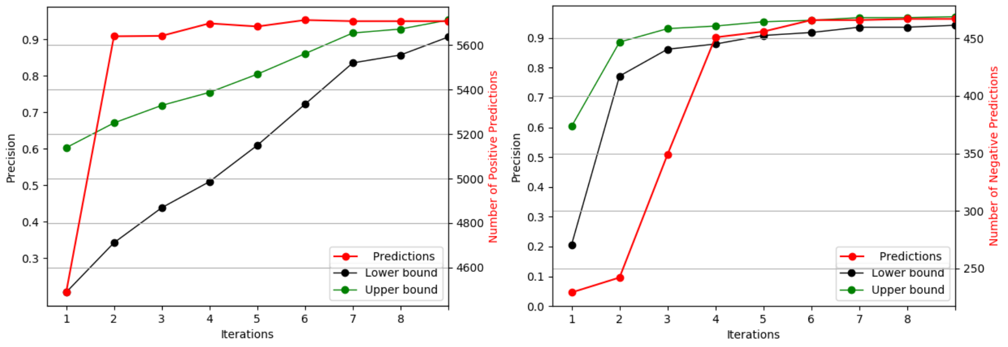

Figure 5: Precision vs. predictions $|\mathcal{T}^-|/|\mathcal{T}^+|$ for $|\mathcal{F}_0| = 40$ over multiple GILP iterations

5 Related Works

Inductive logic programming (ILP) considers the task of learning rules, represented as logic programs, from a set of examples [4, 16]. Given a training set consisting of both positive and negative examples, an ILP system aims to derive a set of rules which should be both complete and consistent, thereby covering all positive examples and none of the negative ones [18]. ILP is widely applied in many applications, such as event recognition [13], data cleaning [22], classification [18], and many others. In this paper, we extend ILP to handle a non-fixed training set, while the learning process is guided by user feedback in an iterative and interactive manner. Data quality is a classical problem in the data-management community [6, 12, 14]. A number of recent works have focused on mining constraints or logical rules from large data sets [1, 9, 8, 22, 2, 7, 19]. They adopt different learning approaches, such as ILP [22, 19], association rule mining [9, 8, 7], and depth-first search [1, 2]. The target of this paper is to interactively learn rules in order to clean large KB based on user feedback, while existing works utilize KBs with static training sets only. Several recent works also tackle the issue of crowdsourcing part of the data cleaning tasks. The KATARA system [3] cleans a relational database by aligning it to a KB. During the cleaning process, the system will collect annotations from crowdsourcing in order to validate the mappings between the relational representation and the KB. In [11], the data cleaning process is bootstrapped from user input. The system will generate a set of possible SQL updates according to the input, and then further interact with the user to validate the generated SQL queries. Compared with these works, our solution requires users to justify the correctness of facts, instead of rules, which is more natural in most settings. Furthermore, our target is not to repair data, but to identify which tuples are correct and/or incorrect in a unified framework.

6 Conclusions

This paper studies the problem of learning rules from both user feedback and a background knowledge bases. The obtained rules help to reveal systematic extraction mistakes that occurred during the construction of the knowledge base in a targeted manner. We highlight the challenges of this problem, especially the non-fixed training set, and propose a design of an iterative ILP algorithm that is able to detect these systematic mistakes. Moreover, we propose appropriate metrics to evaluate the quality of candidate rules. The robustness and effectiveness of our proposed approach are verified by a series of experiments over the YAGO3 knowledge base. In the future, we will study other kinds of user feedbacks, including conflicting feedback facts or predictions made by both positive and negative rules.

7 Acknowledgments

This work is funded by the University of Luxembourg and the China Scholarship Council (CSC).

A Classes of Consistency Constraints

Functional Dependencies. A *functional dependency* (FD) φ over $R(\bar{X})$ is generally defined as $X \rightarrow Y$, where both $X \subseteq \mathit{Var}(\bar{X})$ and $Y \subseteq \mathit{Var}(\bar{X})$ are variables denoting attributes over a same relation schema R. We say that an instance $\mathbf{R}$ of R *satisfies* φ, denoted as $\varphi \vDash \mathbf{R}$, iff for any two tuples $R(\bar{t}_1)$, $R(\bar{t}_2)$ in $\mathbf{R}$, with $\bar{t}_1 \restriction_X = \bar{t}_2 \restriction_X$, it holds that $\bar{t}_1 \restriction_Y = \bar{t}_2 \restriction_Y$. For example, assume R represents the *bornIn* relation with attributes A, B, thus mapping person entities A to their birthplaces B, and we are given the FD $A \rightarrow B$. Then, for any instance $\mathbf{R}$ and pair of tuples $R(\bar{t}_1)$, $R(\bar{t}_2) \in \mathbf{R}$ with $\bar{t}_1 \restriction_{\{A\}} = \bar{t}_2 \restriction_{\{A\}}$, it must also hold that $\bar{t}_1 \restriction_{\{B\}} = \bar{t}_2 \restriction_{\{B\}}$. In other words, every person A occurring in $\mathbf{R}$ must not have two or more different birthplaces.

An FD $\varphi : X \rightarrow Y$ can equivalently be *normalized* into a set of individual FDs, each of the form $\varphi_i : X \rightarrow y_i$, such that the right-hand-side argument of each φ_i consists of just a single variable $y_i \in Y$. Moreover, in first-order notation, each such φ_i can then equivalently be represented as a logical implication of the form

$$\forall x_1 \ldots \forall x_k \forall y_i \ \big(\varphi(\bar{X}) \rightarrow (x_i = y_i)\big) \tag{1}$$

where

- $\varphi(\bar{X})$ is a conjunction of two atomic formulas over relation R, both having variables in $\mathit{Var}(\bar{X})$;
- every variable in $\mathit{Var}(\bar{X})$ appears in $\varphi(\bar{X})$;
- and x_i and y_i are two distinct variables in $\mathit{Var}(\bar{X})$.

Example 4. *An example for an FD, which is represented as a logical implication, is the following constraint that implies a unique birth place for each person in the KB.*

$\forall x_1 \forall x_2 \forall y_2 \ (\ \mathit{bornIn}(x_1, x_2) \wedge \mathit{bornIn}(x_1, y_2) \rightarrow (x_2 = y_2)\ )$

We remark that our above notation for FDs also captures *conditional functional dependencies* (CFDs), since we explicitly allow each $\varphi_i(\bar{X})$ to be defined over a combination of constants and variables in $\bar{X}$. An atom $R(\bar{X})$ in $\varphi_i(\bar{X})$, consisting of both constants and variables, can thus be seen as a template for all tuples we aim to capture by a CFD.

Equality-Generating Dependencies. In the more general case, a dependency that implies the equivalence of a pair of variables may also span multiple relations $R_1, \ldots, R_m$, which we then refer to as an *equality-generating dependency* (EGD) [5]. An EGD can again be represented as a logical implication of the form

$$\forall x_1 \ldots \forall x_k \forall y_i \ \big(\varphi(\bar{X}) \rightarrow (x_i = y_i)\big) \tag{2}$$

where

- $\varphi(\bar{X})$ is a conjunction of atomic formulas over relations $R_1, \ldots, R_m$, all having variables in $\mathit{Var}(\bar{X})$;
- every variable in $\mathit{Var}(\bar{X})$ appears in $\varphi(\bar{X})$;
- and x_i and y_i are two distinct variables in $\mathit{Var}(\bar{X})$.

EGDs thus are a generalization of FDs.

Denial Constraints. As expressive as FDs and EGDs are, they still cannot capture all real-life consistency constraints we may wish to learn from and apply to a given KB. Thus, an even more general class of consistency constraints than the class of EGDs are *denial constraints* (DCs). Formally, a DC is a logical formula of the form

$$\forall x_1 \ldots \forall x_k \ \left(\neg\varphi(\bar{X}) \right) \tag{3}$$

where

- $\varphi(\bar{X})$ is a conjunction of atoms, all having variables in $\mathit{Var}(\bar{X})$;
- each atom refers either to a *relational predicate* of the form $R(\bar{X})$, or to an *arithmetic predicate* $(x_i \, \theta \, x_j)$ over a pair of variables x_i, x_j (with θ including comparison operators such as "=", "≠", ">", "<", etc.);
- every variable in $\mathit{Var}(\bar{X})$ appears in $\varphi(\bar{X})$.

We remark that denial constraints can equivalently be represented as so-called *goal clauses*, i.e., as Horn clauses with empty head literals.

Example 5.

$\forall x_1 \, \forall x_2 \, \forall y_2 \, \neg \, (\, \mathit{bornIn}(x_1, x_2) \wedge \mathit{bornIn}(x_1, y_2) \wedge (x_2 \neq y_2) \,)$

Horn Clauses. A *Horn clause* (HC), finally, is a logical implication of the form

$$\forall x_1 \ldots \forall x_k \ \left(\varphi(\bar{X}) \rightarrow \psi(\bar{X}) \right) \tag{4}$$

where $\varphi(\bar{X})$ is a conjunction of atomic formulas; $\psi(\bar{X})$ is an atomic formula; and $x_1, \ldots, x_k$ are variables in $\mathit{Var}(\bar{X})$.

Example 6.

$\forall x_1 \ \ \mathit{bornIn}(x_1, \mathit{NewYork}) \rightarrow \mathit{hasNationality}(x_1, \mathit{USA})$

Lemma 1. *In terms of expressiveness, it holds that:*

$$\mathit{FD} \subset \mathit{EGD} \subset \mathit{DC} \subset \mathit{HC}$$

Proof. We only show that $\mathit{EGD} \subset \mathit{DC}$ holds, since the other subsumptions directly follow from our previous constructions. Note that the EGD defined in Equation 2 can equivalently be rewritten as

$$\forall x_1 \ldots \forall x_k \, \forall y_i \, \neg \left(R_1(\bar{X}_1) \wedge \cdots \wedge R_m(\bar{X}_m) \wedge (x_i \neq y_i) \right)$$

where $Var(\bar{X}_i) \subseteq Var(\bar{X})$ (for $i = 1, \ldots, m$) and $x_i, y_i \in Var(\bar{X})$. That is, each relational atom R_i in Equation 3 also corresponds to a relational atom in Equation 2. The above constraint thus conforms to a DC. □

References

[1] Z. Abedjan, P. Schulze, and F. Naumann. Dfd: Efficient functional dependency discovery. In *CIKM*, pages 949–958, 2014.

[2] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013.

[3] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. KATARA: A Data Cleaning System Powered by Knowledge Bases and Crowdsourcing. In *SIGMOD*, pages 1247–1261, 2015.

[4] F. Esposito, G. Semeraro, N. Fanizzi, and S. Ferilli. Multistrategy theory revision: Induction and abduction in inthelex. *Machine Learning*, 38(1-2):133–156, 2000.

[5] R. Fagin. Equality-generating dependencies. In L. Liu and T. M. Özsu, editors, *Encyclopedia of Database Systems*, pages 1009–1010. Springer, 2009.

[6] W. Fan and F. Geerts. *Foundations of Data Quality Management*. Morgan and Claypool Publishers, 2012.

[7] W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering conditional functional dependencies. *IEEE Trans. on Knowl. and Data Eng.*, 23(5):683–698, May 2011.

[8] L. Galárraga, C. Teflioudi, K. Hose, and F. M. Suchanek. Fast rule mining in ontological knowledge bases with AMIE+. *VLDB-J.*, 24(6):707–730, 2015.

[9] L. A. Galárraga, C. Teflioudi, K. Hose, and F. Suchanek. AMIE: Association Rule Mining Under Incomplete Evidence in Ontological Knowledge Bases. In *WWW*, pages 413–422, 2013.

[10] D. Gamberger and N. Lavrač. Expert-guided subgroup discovery: Methodology and application. *J. Artif. Intell. Res.*, 17:501–527, 2002.

[11] J. He, E. Veltri, D. Santoro, G. Li, G. Mecca, P. Papotti, and N. Tang. Interactive and deterministic data cleaning: A tossed stone raises a thousand ripples. In *SIGMOD*, pages 893–907, 2016.

[12] I. F. Ilyas and X. Chu. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends in Databases*, 5(4):281–393, 2012.

[13] N. Katzouris, A. Artikis, and G. Paliouras. Incremental learning of event definitions with inductive logic programming. *Machine Learning*, 100(2):555–585, 2015.

[14] Z. Khayyat, I. F. Ilyas, A. Jindal, S. Madden, M. Ouzzani, P. Papotti, J.-A. Quiané-Ruiz, N. Tang, and S. Yin. Bigdansing: A system for big data cleansing. In *SIGMOD*, pages 1215–1230, 2015.

[15] F. Mahdisoltani, J. Biega, and F. M. Suchanek. YAGO3: A knowledge base from multilingual wikipedias. In *CIDR Online Proceedings*, 2015.

[16] S. Muggleton and L. D. Raedt. Inductive Logic Programming: Theory and Methods. *J. Log. Program.*, 19(20):629–679, 1994.

[17] S.-H. Nienhuys-Cheng and R. de Wolf, editors. *Foundations of Inductive Logic Programming*. Springer, 1997.

[18] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

[19] S. Schoenmackers, O. Etzioni, D. S. Weld, and J. Davis. Learning first-order horn clauses from web text. In *EMNLP*, pages 1088–1098, 2010.

[20] E. B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

[21] J. N. Yan, O. Schulte, J. Wang, and R. Cheng. Detecting data errors with statistical constraints. *CoRR*, abs/1902.09711, 2019.

[22] Q. Zeng, J. M. Patel, and D. Page. QuickFOIL: Scalable Inductive Logic Programming. *PVLDB*, 8(3):197–208, 2014.

[23] L. Zhang, W. Wang, and Y. Zhang. Privacy preserving association rule mining: Taxonomy, techniques, and metrics. *IEEE Access*, 7:45032–45047, 2019.