Data Curation with Ontology Functional Dependencies

by

Alexander Keller

A thesis submitted in partial fulfillment
of the requirements for the degree of

Masters of Science

in

Computer Science

University of Ontario Institute of Technology

Supervisor: Dr. Jaroslaw Szlichta Ph.D.

April 2017

Abstract

Poor data quality has become a pervasive issue due to the increasing complexity and size of modern datasets. Constraint based data cleaning techniques rely on integrity constraints as a benchmark to identify and correct errors. While functional dependencies have traditionally been used in existing data cleaning solutions to model syntactic equivalence, they are not able to model broader relationships (e.g., is-a) defined by an ontology. In this work, we take a first step towards extending the set of data quality constraints by defining, discovering, and cleaning *Ontology Functional Dependencies*. We lay out their theoretical foundations, including a set of sound and complete axioms, and a linear inference procedure. We develop efficient algorithms for data verification over ontology FDs. We then develop effective algorithms that discover a complete, minimal set of ontology FDs, and a set of optimizations that efficiently prune the search space. We finally develop cost minimal cleaning algorithms to repair a dataset in violation of a set of constraints, and an extension to discern ontology problems affecting data repair prediction. Our experimental evaluation using real data shows the scalability and accuracy of our algorithms. We show that ontology FDs are a useful data quality rule to capture domain attribute relationships, and can significantly reduce the number of false positive errors in data cleaning techniques that rely on traditional FDs.

Acknowledgments

I would never have been able to finish my dissertation without the guidance of my committee members, help from friends, and support from my family and girlfriend.

It is with immense gratitude that I acknowledge the support and help of my advisor, Dr. Jaroslaw Szlichta, without whom much of this research would not have been undertaken. In addition, I would like to thank Dr. Fei Chiang, and Sridevi Baskaran for the opportunity to work and collaborate on research together.

I would like to thank my laboratory friends Wesley Taylor, Adele Hendrick, Michael Stergianis, Garrett Hayes, Brent McRae, Titus Okathe, Chris Bonk, and anyone else who I may not have been as close with but whom I enjoyed time in conversation with.

Finally, I would like to thank my girlfriend, Sabrina Skoczylas. She is that little spirit that kept me going from time to time.

Contents

List of Figures

List of Tables

Abbreviations

CFD Conditional Functional Dependency

FD Functional Dependency

GFD Graph Functional Dependency

GIS Geographic Information System

GPS Global Positioning System

LHS left-hand-side

MFD Metric Functional Dependency

NSAID Nonsteroidal Anti-inflammatory Drug

OD Ordered Dependency

OLAP Online Analytical Processing

OWL Web Ontology Language

RHS right-hand-side

RUD Roll-Up Dependency

TFD Temporal Functional Dependency

Co-Authorship

Work in Chapters 2 and 4 come from a publication by Alexander Keller and Jaroslaw Szlichta in their paper published in Proceedings of the 10th Alberto Mendelzon International Workshop on Foundations of Data Management [25].

Additionally, Section 4.2 was created in collaboration between Sridevi Baskaran, Fei Chiang, Alexander Keller, and Jaroslaw Szlichta. The contributions of Alexander Keller are discussed in this work. Alexander's work acted as theoretical foundations in the work produced and published. The collaboration went on to complete an implementation and scalability study for joint publication [5].

Chapter 1

Introduction

Organizations are finding it hard to extract value from their data because of poor quality [14]. Big data means exponentially more inconsistent, duplicate, and missing data. A Gartner Research study reports that by 2017, 33% of the largest global companies will experience a data quality crisis due to their inability to trust and govern their enterprise information [24]. With the interest in data analytics at an all-time high, data quality has become a critical issue in research and practice. Integrity constraints are commonly used to characterize and ensure data quality [10, 12, 18, 27, 39, 40]. Databases with data quality problems are often referred to as unclean/dirty databases. The process of improving this quality is called data cleaning. We define a relation to be clean if all data within adheres to all defined integrity constraints.

As introduction, we cover the Functional Dependency (FD). FDs allow an equality relationship to be enforced among two sets of information [8,35]. The main use of FDs is to define group identity among a set of attributes. An FD states that if two tuples agree on the subsequent attributes, then they also must agree on the consequent attributes. An example of this would be F: [Postal Code] $\rightarrow$ [City, State, Country]. Here, when any two or more tuples agree on Postal Code, they must then also agree on their City, State, and Country values. If they do not, they are said to be in violation.

Another integrity constraint is the Metric Functional Dependency (MFD) [27]. MFDs are like regular FDs in that they relate one set of attributes to another. However, MFDs do so within a metric based deviation among all *right-hand-side (RHS)* consequent values whom agree on the *left-hand-side (LHS)* subsequent attributes. One example use case is when dealing with two integrated relations of geographic coordinates such as in Geographic Information System (GIS) data. Precision from the Global Positioning System (GPS) is not as ideal as we expect, measurement errors are common when reporting the location of the same point on the earth, and arbitrary precision means reported values are quite likely to be non-identical among as little as two sources. Traditional functional dependencies are unable to handle such near equalities. Metric functional dependencies are designed specifically to handle these. In our work, we build upon the idea of handling similar, though non-identical data.

1.1 Motivation

To motivate this work, we examine current needs in the medical data domain. The medical data community has, at present, rigorously produced taxonomies for many of their core disciplines (medicine, symptoms, deceases, parasites, etc.). Additionally, the medical community deals with massive amounts of research data in attempting to understand the human body. Although we focus on medical trial data, likely many other areas exist that would benefit from ontology FD.

Example 1.1.1 *Table 1.1 shows a sample of clinical trial records containing patient country codes, country, symptoms, diagnosis, and the prescribed medication. Consider three FDs: F_1: [Country Code] $\rightarrow$ [Country], F_2: [Symptom, Diagnosis] $\rightarrow$ [Medicine], and F_3: [Diagnosis] $\rightarrow$ [Symptom].*

The tuples (t_1, t_5, t_6) do not satisfy F_1 as "United States", "America", and "USA" are not syntactically (string) equivalent (the same is true for (t_2, t_4, t_7)). However,

we know that the "United States" is synonymous with "America" and "USA", and (t_1, t_5, t_6) all refer to the same country. Similarly, "Bharat" in t_4 is also synonymous with "India" as it is the country's original Sanskrit name.

For F_2, (t_1, t_2, t_3) and (t_4, t_5) do not satisfy the dependency as the consequent RHS values all refer to different medications. However, upon closer inspection, with domain knowledge from a medication ontology (Figure 1.2), we see that the values participate in an inheritance relationship. Both "ibuprofen" and "naproxen" both have an is-a *relation to Nonsteroidal Anti-inflammatory Drugs (NSAIDs), and so does "tylenol" as a synonym of "acetaminophen".*

For F_3, (t_4, t_5, t_6) does not satisfy the defined dependency as the consequent values contain multiple symptoms for a disease. Again, we can consult a disease symptom ontology (Figure 1.3) to find the values participate in a component relationship. Here, both tinnitus and nausea are symptoms of a migraine.

	Country Code	Country	Symptom	Diagnosis	Medicine
t_1	US	United States	joint pain	osteoarthritis	ibuprofen
t_2	IN	India	joint pain	osteoarthritis	NSAID
t_3	CA	Canada	joint pain	osteoarthritis	naproxen
t_4	IN	Bharat	nausea	migraine	acetaminophen
t_5	US	America	nausea	migraine	tylenol
t_6	US	USA	tinnitus	migraine	tylenol
t_7	IN	India	chest pain	hypertension	morphine

Table 1.1: Medical Trials Relation

The above example demonstrates that real data often contains domain specific relationships that go beyond simple syntactic equivalence. It also highlights three common relationships that occur frequently between two values b and c: (1) b and c can be *synonyms*; (2) b *is-a* c denoting *inheritance*; and (3) b *part-of* c denoting *composite membership*. These relationships are often defined within domain specific ontologies that can be leveraged during the data cleaning process to identify and enforce domain specific data quality rules. Unfortunately, traditional FDs are unable

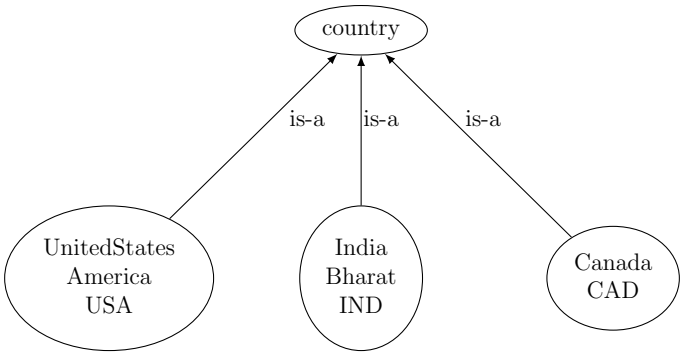

Figure 1.1: Location Ontology

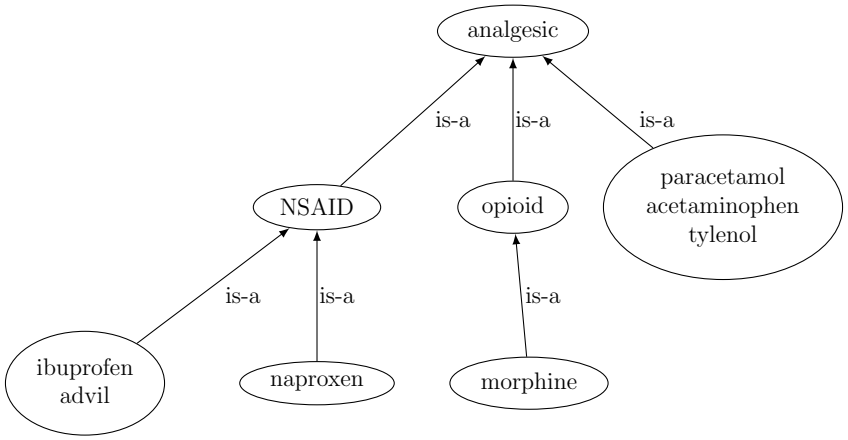

Figure 1.2: Medicine Ontology

to capture these relationships, and existing data cleaning approaches flag tuples containing synonymous, inheritance, or component values as erroneous. This leads to an increased number of "errors" and a larger search space of data repairs to consider.

Existing data cleaning approaches have traditionally considered data quality rules with equality based attribute relationships, such as FDs, Conditional Functional Dependencys (CFDs), and denial constraints [12,13,40]. These data quality rules do not capture the broader semantics modeled in ontologies containing relationships such as synonym, is-a, part-of, and type-of. Existing work in the semantic web community have defined domain constraints over ontologies for the purposes of validating domain values and data completeness [31]. Similarly, recent work in graph databases

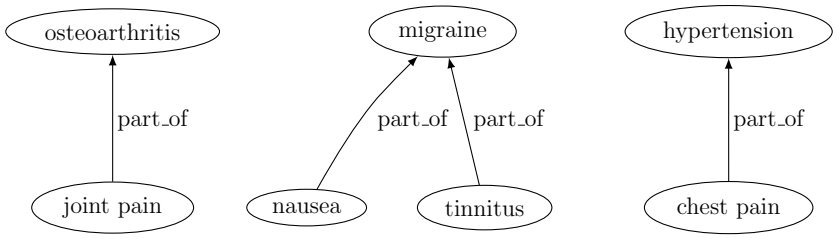

Figure 1.3: Symptoms Ontology

have considered FDs and keys for graphs [15, 17]. Our work is in similar spirit, but addresses gaps that were not previously considered. Namely, we consider attribute relationships that go beyond equality (i.e., synonym, inheritance, and components). We also consider the notion of *senses* that state how a dependency should be interpreted, since multiple interpretations are possible for a given ontology; these interpretations are not considered in existing techniques.

1.2 Contributions

In this paper, we take a first step to address this problem by defining a new class of dependencies called *Ontology Functional Dependencies* that capture relationships defined in an ontology. We focus on the synonym, is-a (inheritance), and part-of (component) relationships between two attribute values. We make the following contributions:

1. In Chapter 2, we define a new class of dependencies called ontology FDs based on the synonym, inheritance, and component relationships of ontologies/taxonomies. In contrast to existing work, our dependencies include attribute relationships that go beyond equality, and consider the notion of *senses* that provide the interpretations under which the dependencies are evaluated.

2. In Chapter 3, we introduce a set of axioms (inference rules) for ontology FDs, and prove these are sound and complete. While the inference complexity of

other FD extensions is co-NP-complete, we show that the inference problem for ontology FDs remains linear. Our inference procedure can be used to reason about the consistency and correctness of data design.

3. In Section 4.1, we develop a set of algorithms for efficient data verification under ontology FDs that run in polynomial time in the number of tuples. We go on to discuss optimizations in these algorithms that aid implementation performance. Efficient data verification is critical for our discovery and cleaning algorithms.

4. In Section 4.2, we use our inference rules to propose optimizations that enable ontology FD discovery algorithms to avoid redundant computations. We present a set of optimizations to prune the search space and improve the algorithm running time, without sacrificing correctness. We prove that our discovery algorithm produces a *complete* and *minimal* set of ontology FDs. We also introduce approximate ontology FDs and show they are a useful data quality rule to capture domain relationships, and can significantly reduce the number of false positives in data cleaning techniques that rely on traditional FDs.

5. In Chapter 5, we develop a set of algorithms that search for changes to a relation in order to realign it with defined constraints. Since the cardinality minimal repair problem for ontology FDs is NP-hard, we develop a greedy cost minimal algorithm to clean the data. We focus on cleaning RHS attributes based on LHS equivalence classes, allowing our approach to scale in co/parallel processing implementations. We also develop an algorithm for detecting when an ontology behind a set of ontology FDs has become stale and should be updated.

6. In Chapter 6, we evaluate the performance and effectiveness of our data verification and cleaning techniques using a real medical trials relation containing 1 million records. Our experiments demonstrate that our algorithms scale well with comparable cleaning precision to those of other dependency constraints.

Chapter 2

Background

2.1 Definitions

Ontological functional dependencies extend FDs by using ontologies to define the function over which the dependency operates. We assume an ontology contains a set of classes (concepts) $\mathbf{C} \in \mathbf{O}$. There are inheritance relations *is-a* between classes that are partial order, where one class $\mathbf{D}$ is a subclass of another class $\mathbf{C}$ ($\mathbf{C}$ is a superclass of $\mathbf{D}$). A partial order is reflexive, transitive, and antisymmetric. For a generalization (subsumption) relation a hyponym (subclass) has an *is-a* relationship with its hyponym. There are also component relations *part-of* among classes. Component sets $\mathscr{C}$ share one or more classes such that $\mathscr{C} = \{\mathbf{C}, \mathbf{D}, \dots\}$. Classes may appear in many sets and sets may contain many classes.

We assume the relation contains, as attribute values, string representations of classes b called string terms. Terms are defined as $\texttt{synonyms}(\mathbf{C}) = \{\mathrm{b}_1, \dots, \mathrm{b}_n\}$. A class with multiple synonyms, i.e., $|\texttt{synonyms}(\mathbf{C})| > 1$, contains alternative string representations for the class (synonyms). Similarly, each term can appear in multiple classes (multiple meanings/senses).

We assume the $\texttt{synonyms}(\mathbf{C})$ reverse predicate $\texttt{classes}(\mathrm{b})$ returns a set of all

classes represented by string term b. Any class with a single name, i.e. $|\texttt{synonyms}(\mathbf{C})| = 1$ possesses the property $\texttt{classes}(\texttt{synonyms}(\mathbf{C})) = \mathbf{C}$.

Symbol(s) Used	Denotational Meaning
$\mathbf{R}$	Relation schema
$\mathbf{r}$	Specific relation (table)
$\mathscr{M}, \mathscr{N}$	Set of dependencies
F	Single dependency
$\mathscr{V}, \mathscr{W}, \mathscr{X}, \mathscr{Y}, \mathscr{Z}$	Set of attributes
A, B	Single attribute
$\mathfrak{s}, \mathfrak{t}$	Set of tuples
s, t, u, v	Single tuple
b, c, d, e	Single string term
$\mathbf{O}$	Ontology
$\mathscr{C}, \mathscr{D}, \mathscr{X}$	Set of ontology classes
$\mathbf{C}, \mathbf{D}, \mathbf{E}, \mathbf{F}$	Single ontology class
$\mathbf{C}_n, F_n, t_n, \mathrm{b}_n$	The nth element of a set of such elements
`subroutine`	Function/subroutine name

Table 2.1: Notation and Examples

Definition 2.1.1 *Let* $t[\mathrm{A}] = \mathrm{b}$ *denote the value of attribute* A *in tuple* t *as being equal to the string term* b.

Definition 2.1.2 *Let* $g_s[\mathscr{X}] = \{t | t \in \mathbf{r},\ \textit{and}\ t[\mathscr{X}] = s\}$.

That is, let $g_s[\mathscr{X}]$ project the set of tuples from $\mathbf{r}$ into subsets such that each tuple set over the attributes of $\mathscr{X}$ are equal to the values defined in s. When s is variably defined, this returns all sets, groups by the values within $\mathscr{X}$. These groupings are equivalence classes.

Definition 2.1.3 *A relation* $\mathbf{r}$ *satisfies a* synonym FD $\mathscr{X} \overset{s}{\mapsto} \mathscr{Y}$, *if for each attribute* $\mathrm{A} \in \mathscr{Y}$, *for each* $s \in \Pi_{\mathscr{X}}(\mathbf{r})$, *there exists a class* $\mathbf{C}$, *such that* $\Pi_A(g_s[\mathscr{X}]) \subseteq \texttt{synonyms}(\mathbf{C})$.

Note that if all classes have a single string representation, i.e., $(\forall(C) \in \mathbf{O}\ |\texttt{synonyms}(\mathbf{C})| = 1)$, then a synonym FD is an FD.

Example 2.1.1 *We define a* synonym FD *on Table 1.1 [Country Code]* $\overset{s}{\mapsto}$ *[Country] using the ontology from Figure 1.1. For tuples* (t_1, t_5, t_6)*, grouping on the equivalence class "US" requires "United States", "America", and "USA" to all appear as synonyms of some class. Examining Figure 1.1 we see this is in fact the case. Therefore, this equivalence class is valid. The same holds for equivalence classes* (t_2, t_4, t_7) *and* (t_3)*.*

We assume $\texttt{ancestors}(\mathbf{C})$ returns the path set of recursive ancestor classes from the class $\mathbf{C}$. That is:

Definition 2.1.4 $\texttt{ancestors}(\mathbf{C}) = \{\mathbf{C}|\mathbf{C}$ is-a $\mathbf{C}_n|\mathbf{C}_n$ is-a $\mathbf{C}_{n-1}, \ldots, \mathbf{C}_1$ is-a $\mathbf{C}\}$.

We use the Lowest Common Ancestor function $\texttt{LCA}(\{\mathbf{C}, \ldots, \mathbf{C}_n\}, \{\mathbf{D}, \ldots, \mathbf{D}_n\}, \ldots)$ to return the distance to the lowest common ancestor among sets (paths) of ancestor classes. For example:

Example 2.1.2 $\texttt{LCA}(\{\mathbf{D}, \mathbf{C}, \mathbf{F}\}, \{\mathbf{E}, \mathbf{C}, \mathbf{F}\}) = 1$ *because* $\mathbf{D}$ is-a $\mathbf{C}$ *is one traversal and* $\mathbf{E}$ is-a $\mathbf{C}$ *is one traversal. All paths have found a common ancestor and the longest path is 1, therefore the returned distance is 1. If no common ancestor is found, this distance is infinite.*

Definition 2.1.5 *A relation* $\mathbf{r}$ *satisfies a* generalization FD $\mathfrak{X} \overset{g}{\mapsto} \mathcal{Y}$*, if for each attribute* $\mathrm{A} \in \mathcal{Y}$*, for each* $s \in \Pi_{\mathfrak{X}}(\mathbf{r})$ *there exists a class* $\mathbf{C}$*, such that* $\texttt{LCA}(\texttt{ancestors}(\texttt{classes}(t_1)), \ldots, \texttt{ancestors}(\texttt{classes}(t_n))) \leq \theta$*.*

We consider a restricted version of generalization FDs that limits the inheritance to whole number paths of maximum length θ, denoted as $\mathfrak{X} \overset{g}{\mapsto}_{\theta} \mathcal{Y}$. A length of $\theta = 0$ generalization FD is equivalent to a synonym FD. That is $\mathfrak{X} \overset{g}{\mapsto}_0 \mathcal{Y} \equiv \mathfrak{X} \overset{s}{\mapsto} \mathfrak{X}$.

Example 2.1.3 *In this paper we define a* generalization FD *on Table 1.1 [Symptom, Diagnosis]* $\overset{g}{\mapsto}_1$ *[Medicine] using the ontology from Figure 1.2. Here we allow one level*

of generalization between the values of Medicine. For tuples (t_1, t_2, t_3), grouping on the equivalence class "joint pain, osteoarthritis" requires "ibuprofen", "NSAID", and "naproxen" to share a common ancestor within 1 level of generalization. Examining Figure 1.2 we see this is in fact the case. More formally we see that

$$\texttt{LCA}(\texttt{ancestors}(\texttt{classes}(t_1)), \texttt{ancestors}(\texttt{classes}(t_2)), \texttt{ancestors}(\texttt{classes}(t_3)))$$

$$\texttt{LCA}(\{\textit{ibu.}, \textit{NSAID}, \textit{ana.}\}, \{\textit{NSAID}, \textit{ana.}\}, \{\textit{nap.}, \textit{NSAID}, \textit{ana.}\})$$

This returns 1 as the longest distance to "NSAID" (the lowest common ancestor) is 1 (ibuprofen is-a *NSAID). 1 is $\leq \theta$ and therefore the equivalence class is valid. The same holds for equivalence classes (t_4, t_5), (t_6), and (t_7).*

We assume $\texttt{components}(\mathscr{C})$ returns the set of all classes having a *part-of* relation into the composite $\mathscr{C}$. We assume the inverse $\texttt{composes}(\mathbf{C})$ returns the set of all component sets a class is a part of.

Definition 2.1.6 *A relation* $\mathbf{r}$ *satisfies a* component FD $\mathfrak{X} \overset{c}{\mapsto} \mathcal{Y}$*, if for each attribute* $\mathrm{A} \in \mathcal{Y}$*, for each* $s \in \Pi_{\mathfrak{X}}(\mathbf{r})$ *there exists a composite* $\mathscr{C}$*, such that* $\forall \mathrm{b} \in \Pi_{\mathrm{A}}(g_s[\mathfrak{X}])$*,* $\{\texttt{classes}(\mathrm{b}) \cap \texttt{components}(\mathscr{C})\} \neq \emptyset$*.*

Example 2.1.4 *In this paper we define a* component FD *on Table 1.1 [Diagnosis]* $\overset{c}{\mapsto}$ *[Symptom] using the ontology from Figure 1.3. For tuples (t_4, t_5, t_6), grouping on the equivalence class "migrane" requires "nausea" and "tinnitus" to both be components in a set. Examining Figure 1.3 we see this is in fact the case. Therefore, this equivalence class is valid. The same holds for equivalence classes (t_1, t_2, t_3) and (t_7).*

This definition means that not all classes $\mathbf{C}$ contained as *part-of* the component set $\mathscr{C}$ need appear in the equivalence class $\Pi_{\mathrm{A}}(g_{\mathrm{b}}[\mathfrak{X}])$ for the component FD integrity constraint to hold over a relation. This allows tuples to be added atomically without falsifying the constraint.

Chapter 3

Theoretical Framework

3.1 Axiomatization

We present an *axiomatization* for ontology FDs, analogous to Armstrong's axiomatization for FDs [3]. This provides a formal framework for reasoning about ontology FDs. The axioms provide insight into how ontology FDs behave—and patterns for how dependencies logically follow from others—that are not easily evident reasoning from first principles. A sound and complete axiomatization is the first necessary step to designing an efficient *inference procedure*.

1. *Identity*
   $\mathcal{X} \mapsto \mathcal{X}$.

2. *Decomposition*
   If $\mathcal{X} \mapsto \mathcal{Y}$,
   and $\mathcal{Z} \subseteq \mathcal{Y}$,
   then $\mathcal{X} \mapsto \mathcal{Z}$.

3. *Composition*
   If $\mathcal{X} \mapsto \mathcal{Y}$,
   and $\mathcal{Z} \mapsto \mathcal{W}$,
   then $\mathcal{XZ} \mapsto \mathcal{YW}$.

Figure 3.1: Axiomatization for Ontology FDs

The axioms (inference rules) for ontology FDs are presented in Figure 3.1. One of the axioms, i.e., Identity, generates *trivial* dependencies, which are always true. We introduce additional inference rules, which follow from axiom in Figure 3.1, as they will be used throughout in the reminder of the section, in particular to prove that ontology FD axioms are complete.

Lemma 3.1.1 (*Reflexivity*) *If* $\mathcal{Y} \subseteq \mathcal{X}$, *then* $\mathcal{X} \mapsto \mathcal{Y}$.

Proof 3.1.1 $\mathcal{X} \mapsto \mathcal{X}$ *holds by Identity axiom. Therefore, it can be inferred by the Decomposition inference rule that* $\mathcal{X} \mapsto \mathcal{Y}$ *holds.*

Union inference rule shows what can be inferred from two or more dependencies which have the same sets on the left side.

Lemma 3.1.2 (*Union*) *If* $\mathcal{X} \mapsto \mathcal{Y}$ *and* $\mathcal{X} \mapsto \mathcal{Z}$, *then* $\mathcal{X} \mapsto \mathcal{Y}\mathcal{Z}$.

Proof 3.1.2 *We are given* $\mathcal{X} \mapsto \mathcal{Y}$ *and* $\mathcal{X} \mapsto \mathcal{Z}$. *Hence, the Composition axiom can be used to infer* $\mathcal{X} \mapsto \mathcal{Y}\mathcal{Z}$.

Next, we define the *closure* of a set of attributes $\mathcal{X}$ over a set of ontology FDs $\mathscr{M}$. We use the notation $\mathscr{M} \vdash$ to state that $\mathcal{X} \mapsto \mathcal{Y}$ is provable with axioms from $\mathscr{M}$.

Definition 3.1.1 (*Closure*) *The closure of* $\mathcal{X}$, *denoted as* $\mathcal{X}^+$, *with respect to the set of ontology FDs* $\mathscr{M}$ *is defined as* $\mathcal{X}^+ = \{\mathrm{A} | \mathscr{M} \vdash \mathcal{X} \mapsto \mathrm{A}\}$.

The important information about closure $\mathcal{X}^+$ is that it can be used to determine whether an ontology FD follows from $\mathscr{M}$ by axioms. The following lemma shows how.

Lemma 3.1.3 $\mathscr{M} \vdash \mathcal{X} \mapsto \mathcal{Y}$ *iff* $\mathcal{Y} \subseteq \mathcal{X}^+$.

Proof 3.1.3 *Let* $\mathcal{Y} = \{\mathrm{A}_1, \ldots, \mathrm{A}_n\}$. *Assume* $\mathcal{Y} \subseteq \mathcal{X}^+$. *By definition of* $\mathcal{X}^+$, $\mathcal{X} \mapsto \mathrm{A}_i$, *for all* $i \in \{1, \ldots, n\}$. *Therefore, by Union inference rule,* $\mathcal{X} \mapsto \mathcal{Y}$ *follows. The other direction, suppose* $\mathcal{X} \mapsto \mathcal{Y}$ *follows from the axioms. For each* $i \in \{1, \ldots, n\}$, $\mathcal{X} \mapsto \mathrm{A}_i$ *follows by the Decomposition axiom. Therefore,* $\mathcal{Y} \subseteq \mathcal{X}^+$.

We denote logical operator $\models$ of the form $\mathscr{M} \models F$ to imply the assertion that a set of axioms ($\mathscr{M}$) logically implies the dependency F. i.e., in every circumstance in which $\mathscr{M}$ is true, F is true. We also denote the logical operator $\vdash$ of the form $\mathscr{M} \vdash F$ to imply the assertion that the dependency F follows from $\mathscr{M}$ as a consequence of a set of inferences, dependent on the set of axioms $\mathscr{M}$. Soundness states $\mathscr{M} \vdash F$ implies $\mathscr{M} \models F$ while completeness states $\mathscr{M} \models F$ implies $\mathscr{M} \vdash F$.

Theorem 3.1.1 *Ontology FD axioms in Figure 3.1 are sound and complete.*

Proof 3.1.4 *First we prove that the axioms are sound. That is, if* $\mathscr{M} \vdash \mathfrak{X} \mapsto \mathscr{Y}$, *then* $\mathscr{M} \models \mathfrak{X} \mapsto \mathscr{Y}$. *The Identity axiom is clearly sound. We cannot have a relation with tuples that agree on* $\mathfrak{X}$ *yet are not in synonym, generalization or component relationship, respectively. To prove Decomposition, suppose we have a relation that satisfies* $\mathfrak{X} \mapsto \mathscr{Y}$ *and* $\mathfrak{Z} \subseteq \mathscr{Y}$. *Therefore, for all tuples that agree on* $\mathfrak{X}$, *they are in synonym, generalization or component relationship on all attributes in* $\mathscr{Y}$ *and hence, also on* $\mathfrak{Z}$. *Therefore,* $\mathfrak{X} \mapsto \mathfrak{Z}$. *The soundness of Composition is an extension of the argument given previously.*

Below we present the completeness proof, that is, if $\mathscr{M} \models \mathfrak{X} \mapsto \mathscr{Y}$, *then* $\mathscr{M} \vdash \mathfrak{X} \mapsto \mathscr{Y}$. *Without a loss of generality, we consider a table* $\mathfrak{t}$ *with three tuples shown in Table 3.1. We divide the attributes of a relation* $\mathfrak{t}$ *into three subsets:* $\mathfrak{X}$, *the set consisting of attributes in the closure* $\mathfrak{X}^+$ *minus attributes in* $\mathfrak{X}$ *and all remaining attributes. Assume that the values* b, b′ *and* b″ *are not equal (*$\mathrm{b} \neq \mathrm{b}'$, $\mathrm{b} \neq \mathrm{b}''$ *and* $\mathrm{b}' \neq \mathrm{b}''$*), however, they are in synonym, generalization or component relationship, respectively. Also,* b, c *and* d *are not in synonym, generalization and component relationship, and hence, they are also not equal.*

We first show that all dependencies in the set of ontology FDs $\mathscr{M}$ *are satisfied by a table* $\mathfrak{t}$ *(*$\mathfrak{t} \models F$*). Since ontology FD axioms are sound, ontology FDs inferred from* $\mathscr{M}$ *are true. Assume* $\mathscr{V} \mapsto \mathfrak{Z}$ *is in* $\mathscr{M}$, *however, it is not satisfied by a relation* $\mathfrak{t}$. *Therefore,* $\mathscr{V} \subseteq \mathfrak{X}$ *because otherwise tuples of* $\mathfrak{t}$ *disagree on some attribute of* $\mathscr{V}$

$\mathfrak{X}^+$		
$\mathfrak{X}$	$\mathfrak{X}^+ \setminus \mathfrak{X}$	Other attributes
b...b	b...b	b...b
b...b	b′...b′	c...c
b...b	b″...b″	d...d

Table 3.1: Table Template for Ontology FDs.

since b, b′ *and* b″ *as well as* b, c, d *are not equal, and consequently an ontology FD* $\mathcal{V} \mapsto \mathfrak{Z}$ *would not be violated. Moreover,* $\mathfrak{Z}$ *cannot be a subset of* $\mathfrak{X}^+$ *(*$\mathfrak{Z} \nsubseteq \mathfrak{X}^+$*), or else* $\mathcal{V} \mapsto \mathfrak{Z}$*, would be satisfied by a table* $\mathfrak{t}$*. Let* A *be an attribute of* $\mathfrak{Z}$ *not in* $\mathfrak{X}^+$*. Since,* $\mathcal{V} \subseteq \mathfrak{X}$*,* $\mathfrak{X} \mapsto \mathcal{V}$ *by Reflexivity. Also a dependency* $\mathcal{V} \mapsto \mathfrak{Z}$ *is in* $\mathcal{M}$*, hence, by Decomposition,* $\mathcal{V} \mapsto$ A*. By Composition* $\mathfrak{X}\mathcal{V} \mapsto \mathcal{V}$A *can be inferred, therefore,* $\mathfrak{X} \mapsto \mathcal{V}$A *as* $\mathcal{V} \subseteq \mathfrak{X}$*. However, then Decomposition rule tells us that* $\mathfrak{X} \mapsto$ A*, which would mean by the definition of the closure that* A *is in* $\mathfrak{X}^+$*, which we assumed not to be the case. Contradiction. An ontology FD* $\mathcal{V} \mapsto \mathfrak{Z}$ *which is in* $\mathcal{M}$ *is satisfied by* $\mathfrak{t}$*.*

Our remaining proof obligation is to show that any ontology FD not inferable from set of ontology FDs $\mathcal{M}$ *with ontology FD axioms (*$\mathcal{M} \nvdash \mathfrak{X} \mapsto \mathcal{Y}$*) is not true (*$\mathcal{M} \nvDash \mathfrak{X} \mapsto \mathcal{Y}$*). Suppose it is satisfied (*$\mathcal{M} \vDash \mathfrak{X} \mapsto \mathcal{Y}$*). By Reflexivity* $\mathfrak{X} \mapsto \mathfrak{X}$*, therefore, by Lemma 3.1.3* $\mathfrak{X} \subseteq X^+$*. Since* $\mathfrak{X} \subseteq \mathfrak{X}^+$ *it follows by the construction of table* $\mathfrak{t}$ *that* $\mathcal{Y} \subseteq \mathfrak{X}^+$*. Else tuples of table* $\mathfrak{t}$ *agree on* $\mathfrak{X}$ *but are not in synonym, generalization or component relationship, respectively, on some attribute* A $\in \mathcal{Y}$*. Then, from Lemma 3.1.3 it can be inferred that* $\mathfrak{X} \mapsto \mathcal{Y}$*. Contradiction. Thus, whenever* $\mathfrak{X} \mapsto \mathcal{Y}$ *does not follow from* $\mathcal{M}$ *by ontology FDs axioms,* $\mathcal{M}$ *does not logically imply* $\mathfrak{X} \mapsto \mathcal{Y}$*. That is the axiom system is complete over ontology FDs, which ends the proof of Theorem 3.1.1.*

It is interesting to note some axioms that hold for FDs do not hold for ontology FDs, including Transitivity: if $\mathfrak{X} \mapsto \mathcal{Y}$ and $\mathcal{Y} \mapsto \mathfrak{Z}$, then $\mathfrak{X} \mapsto \mathfrak{Z}$.

Example 3.1.1 *Consider the relation with three tuples in Table 3.2. The synonym*

FD [Country] $\overset{s}{\mapsto}$ [Country Code] holds since CAD and CA are synonyms. In addition, [Country Code] $\overset{s}{\mapsto}$ [Symptom] holds as CAD and CA are not equal, That is, CAD $\neq$ CA. However, the transitive synonym FD: [Country] $\overset{s}{\mapsto}$ [Symptom] does not hold as congestion is not a synonym to both fever and pyrexia.

Patient ID	Country	Country Code	Symptom
10	Canada	CAD	Fever
11	Canada	CA	Congestion
12	Canada	CAD	Pyrexia

Table 3.2: Lack of Transitivity

3.2 Inference System

A goal in any dependency theory is to develop algorithms for inference problem. The inference procedure can be used to reason about the consistency and correctness of data design. We present an inference procedure for the inference problem for ontology FDs. Computing the closure for ontology FDs can be done efficiently. It takes time proportional to the length of all the dependencies in $\mathscr{M}$, written out.

Algorithm 3.1 Inference Procedure for Ontology FDs

Input: A set of ontology FDs $\mathscr{M}$, and a set of attributes $\mathfrak{X}$
Output: The closure of $\mathfrak{X}$ with respect to $\mathscr{M}$

1: $F_{unused} \leftarrow \mathscr{M}$
2: $n \leftarrow 0$
3: $\mathfrak{X}^n \leftarrow \mathfrak{X}$
4: **loop**
5: **if** $\exists \mathscr{V} \mapsto \mathfrak{Z} \in F_{unused}$ and $\mathscr{V} \subseteq \mathfrak{X}$ **then**
6: $\mathfrak{X}^{n+1} \leftarrow \mathfrak{X}^n \cup \mathfrak{Z}$
7: $F_{unused} \leftarrow F_{unused} \setminus \{\mathscr{V} \mapsto \mathfrak{Z}\}$
8: $n \leftarrow n + 1$
9: **else**
10: **return** $\mathfrak{X}^n$
11: **end if**
12: **end loop**

Theorem 3.2.1 *Algorithm 3.1 correctly computes closure* $\mathfrak{X}^+$.

Proof 3.2.1 *First we show by induction on* n *that if* $\mathfrak{Z}$ *is placed in* $\mathfrak{X}^n$ *in Algorithm 3.1, then* $\mathfrak{Z}$ *is in* $\mathfrak{X}^+$.

Basis*:* $n = 0$. *By Identity axiom* $\mathfrak{X} \mapsto \mathfrak{X}$.

Induction*:* $n > 0$. *Assume that* $\mathfrak{X}^{n-1}$ *consists only of attributes in* $\mathfrak{X}^+$. *Suppose* $\mathfrak{Z}$ *is placed in* $\mathfrak{X}^n$ *because* $\mathcal{V} \mapsto \mathfrak{Z}$, *and* $\mathcal{V} \subseteq \mathfrak{X}$. *By Reflexivity* $\mathfrak{X} \mapsto \mathcal{V}$, *therefore, by Composition and Decomposition,* $\mathfrak{X} \mapsto \mathfrak{Z}$. *Thus,* $\mathfrak{Z}$ *is in* $\mathfrak{X}^+$.

Now we prove the opposite, if $\mathfrak{Z}$ *is in* $\mathfrak{X}^+$, *then* $\mathfrak{Z}$ *is in the set returned by Algorithm 3.1. Suppose* $\mathfrak{Z}$ *is in* $\mathfrak{X}^+$ *but* $\mathfrak{Z}$ *is not in the set returned by Algorithm 3.1.*

Consider table $\mathfrak{t}$ *similar to that in Table 3.1. Table* $\mathfrak{t}$ *has three tuples that agree on attributes in* $\mathfrak{X}$, *are in synonym, generalization or component relationship, respectively, but not equal on* $\{\mathfrak{X}^n \setminus \mathfrak{X}\}$, *and are not in synonym, generalization or component relationship, respectively, on all other attributes (hence, also no equal).*

We claim that $\mathfrak{t}$ *satisfies* $\mathscr{M}$. *If not, let* $\mathcal{V} \mapsto \mathcal{W}$ *be a dependency in* $\mathscr{M}$ *that is violated by* $\mathfrak{t}$. *Then* $\mathcal{V} \subseteq \mathfrak{X}$ *and* $\mathcal{W}$ *cannot be a subset of* $\mathfrak{X}^n$, *if the violation happens. Similar argument was used in the proof of Theorem 3.1.1. Thus, by Algorithm 3.1, Lines 5–8 there exists* $\mathfrak{X}^{n+1}$, *which is a contradiction.*

Example 3.2.1 *Let* $\mathscr{M}$ *be the set of generalization FDs from our running example in Table 1.1: [Country]* $\overset{g}{\mapsto}$ *[Country Code] and [Country, Disease]* $\overset{s}{\mapsto}$ *[Medicine]. Note that [Country]* $\overset{g}{\mapsto}$ *[Country Code] holds since [Country]* $\overset{s}{\mapsto}$ *[Country Code] and generalization FDs subsume synonym FDs. Therefore, the closure [Country, Disease]*$^+$ *computed with our inference procedure (Algorithm 3.2.1) is [Country, Disease, Medicine].*

For a given set of ontology FDs $\mathscr{M}$, we can find an equivalent set with a number of useful properties. A minimal set of ontology FDs is a set with single attributes in the consequence that contain no redundant attributes in the antecedent and that

contain no redundant dependencies. We assumed that the input ontology FDs for our repair algorithm are minimal. To achieve this, we can apply the inference procedure described above to compute a minimal cover of a set of ontology FDs.

Definition 3.2.1 (*Minimal Cover*) *A set $\mathscr{M}$ of ontology FDs is minimal if*

1. $\forall \mathfrak{X} \mapsto \mathcal{Y} \in \mathscr{M}$, $\mathcal{Y}$ *contains a single attribute*
2. *for no* $\mathfrak{X} \mapsto \mathcal{Y} \in \mathscr{M}$ *is* $\mathscr{M} \setminus \{\mathfrak{X} \mapsto \mathrm{A}\}$ *equivalent to* $\mathscr{M}$
3. *for no* $\mathfrak{X} \mapsto \mathrm{A}$ *and proper subset* $\mathfrak{Z}$ *of* $\mathfrak{X}$ *is* $\mathscr{M} \setminus \{\mathfrak{X} \mapsto \mathrm{A}\} \cup \{\mathfrak{Z} \mapsto \mathrm{A}\}$ *equivalent to* $\mathscr{M}$

If $\mathscr{M}$ is minimal and $\mathscr{M}$ is equivalent to a set of ontology FDs $\mathscr{N}$, then we say $\mathscr{M}$ is a minimal cover of $\mathscr{N}$.

Theorem 3.2.2 *Every set of ontology FDs $\mathscr{M}$ has a minimal cover.*

Proof 3.2.2 *By the Union and Decomposition inference rules, it is possible to have $\mathscr{M}$ with only a single attribute in the RHS. We can achieve two other conditions by repeatedly deleting an attribute and then repeatedly removing a dependency. We can test whether an attribute* B *from* $\mathfrak{X}$ *is redundant for the ontology FD* $\mathfrak{X} \mapsto \mathrm{A}$ *by checking if* A *is in* $\{\mathfrak{X} \setminus \mathrm{B}\}^+$. *We can test whether* $\mathfrak{X} \mapsto \mathrm{A}$ *is redundant by computing closure* $\mathfrak{X}^+$ *with respect to* $\mathscr{M} \setminus \{\mathfrak{X} \mapsto \mathrm{A}\}$. *Therefore, we eventually reach a set of ontology FDs which is equivalent to $\mathscr{M}$ and satisfies conditions 1, 2 and 3.*

Example 3.2.2 *Let the set of ontology FDs* $\mathscr{M} = F_1 : \{[\textit{Country}] \overset{g}{\mapsto} [\textit{Country Code}]\}$, $F_2 : \{[\textit{Country, Disease}] \overset{g}{\mapsto} [\textit{Medicine}]\}$, $F_3 : \{[\textit{Country, Disease}] \overset{g}{\mapsto} [\textit{Medicine, Country Code}]\}$. *Therefore, the set of ontology FDs $\mathscr{M}$ is not a minimal cover as F_3 follows from F_1 and F_2 by the Composition inference rule.*

Chapter 4

Data Verification

4.1 Data Verification

Data verification checks whether one or more defined dependencies hold over a relation. To aid parallel and distributed computing, all of these algorithms can be run in parallel when dealing with multiple constraints simultaneously.

In data verification, we present three algorithms, one for each of the defined ontology FDs. Traditional FDs $\mathfrak{X} \rightarrow \mathcal{Y}$ hold over a relation instance, if for each grouping $\mathfrak{z} \in \Pi_{\mathfrak{X}}(\mathbf{r})$, we can verify that $|t| = 1$, where $t = \Pi_{\mathrm{A}}(g_{\mathfrak{z}}[\mathfrak{X}])$. More complex algorithms are required for ontology FDs. The choice of ontological relation (synonym, generalization, or component) directly impacts the complexity and efficiency of the verification algorithms as is discussed in each description below. By using the decomposition axiom (Figure 3.1) we can verify any RHS attributes $\mathcal{Y}$ by testing each $\mathrm{A} \in \mathcal{Y}$ against $\mathfrak{X}$ independently. This simplifies our algorithms and means an algorithm to test $\mathfrak{X} \rightarrow \mathrm{A}$ can be used to test $\mathfrak{X} \rightarrow \mathcal{Y}$ through decomposition.

The synonym FD verification Algorithm 4.1 iterates over all unique tuples composed under the attributes in $\mathfrak{X}$ denoted as $\mathfrak{z}$. This allows us to group on each distinct tuple from $\mathfrak{X}$ in the second loop. The second loop uses the $\Pi_{\mathrm{A}}(g_{\mathfrak{z}}[\mathfrak{X}])$ operation to

Algorithm 4.1 Verify Synonym FD

Input: Relation $\mathbf{r}$, set of attributes $\mathfrak{X}$ and an attribute A
Output:true if dependency $\mathfrak{X} \overset{s}{\mapsto}$ A holds, otherwise false

```
1: for all ȝ ∈ Π_𝔛(r) do
2:    for all t ∈ Π_A(g_ȝ[𝔛]) do
3:       t = {t_1, ..., t_n}
4:       if classes(t_1) ∩ ··· ∩ classes(t_n) = ∅ then
5:          return false
6:       end if
7:    end for
8: end for
9: return true
```

get the string terms of A grouped into subsets where all $\mathfrak{X}$ attributes are equal. This leaves the RHS (A) grouped by the LHS in our relation. We next project A from the relation, grouped for evaluation. Defining t just affirms this set is composed of a group of tuples denoted $\{t_1, \ldots, t_n\}$. That is, t comprises all unique tuples of A under one equivalence class of $\mathfrak{X}$. This process is shared by all following verification algorithms.

The next step is testing the tuples' string terms to determine if they all appear within a single class. To do this, the intersection of all classes of all string terms of the equivalence class is computed. If at least one class contains all string terms in the equivalence class, then the dependency is met. To test this efficiently, we prove the dependency holds by contradiction. In practice we do so by testing for an empty set following an iterative intersection, thus the function returns false as soon as we have proven just one term falsifies the dependency for it's equivalence class.

In the worst case, this algorithm's complexity requirement is $\mathcal{O}(n^2)$. We assume that access to the ontology is indexed (as a map) and can be achieved within a constant factor, i.e., $\mathcal{O}(1)$.

To verify a generalization FD we implement Algorithm 4.2. This algorithm works by computing the distance to the lowest common ancestor of all ancestral paths, of all classes, of all string terms. That is, the path from each class returned from

Algorithm 4.2 Verify Generalization FD

Input: Relation $\mathbf{r}$, set of attributes $\mathfrak{X}$, an attribute A, and maximum distance θ
Output:true if dependency $\mathfrak{X} \overset{g}{\mapsto}_{\theta}$ A holds, otherwise false

```
1: for all ϑ ∈ Π_X(r) do
2:    for all t ∈ Π_A(g_ϑ[𝔛]) do
3:       t = {t_1, ..., t_n}
4:       if LCA(ancestors(classes(t_1)), ..., ancestors(classes(t_n))) < θ then
5:          return false
6:       end if
7:    end for
8: end for
9: return true
```

`classes`() to it's highest root node is returned as a series. Each of these sets of paths are passed as order sets to the `LCA`() function that computes the longest distance to the first common node. Algorithm 4.2 requires a complexity of $\mathcal{O}(n^3)$ to accommodate ancestral queries. This complexity is only worst case and in practice more linear complexity is seen (see experimental results in Section 6.2).

Algorithm 4.3 Verify Component FD

Input: Relation $\mathbf{r}$, set of attributes $\mathfrak{X}$ and an attribute A
Output:true if dependency $\mathfrak{X} \overset{c}{\mapsto}$ A holds, otherwise false

```
1: for all ϑ ∈ Π_X r) do
2:    for all t ∈ Π_A(g_ϑ[𝔛]) do
3:       t = {t_1, ..., t_n}
4:       if composes(classes(t_1)) ∩ ··· ∩ composes(classes(t_n)) = ∅ then
5:          return false
6:       end if
7:    end for
8: end for
9: return true
```

Verification of component FDs uses a similar method to the synonym FD verification function (Algorithm 4.3). The primary difference is the function `composes`(t) which takes as input a set of classes and returns a set of component sets. As with synonym FD verification, the worst case complexity is $\mathcal{O}(n^2)$. We again assume the ontology is accessible via constant lookup from string terms to both classes and from classes to component sets.

4.2 Dependency Discovery

Dependency discovery examines a relation's attributes to find dependencies that are valid for given data. Discovery is useful when relations exist prior to constraint application and a complete knowledge of applicable constraints is not known. Even in such cases where most constraints are known, it is possible that some constraints hold which are non-obvious, and may therefore be found and considered for enforcement.

Given inference rules for ontology FDs, we present our algorithm, named *FASTOFD* (Algorithm 4.4), which efficiently discovers a complete and minimal set of ontology FDs over a relational instance. An ontology FD, $\mathcal{X} \rightarrow$ A is *trivial* if A $\in \mathcal{X}$ by Reflexivity. An ontology FD $\mathcal{X} \rightarrow$ A is *minimal* if it is non-trivial and there is no set of attributes $\mathcal{Y} \subset \mathcal{X}$ such that $\mathcal{Y} \rightarrow$ A holds in a table by Augmentation.

Algorithm 4.4 FASTOFD

Input: Relation **r** over schema **R**
Output: Minimal set of ontology FDs $\mathscr{M}$, such that $\mathbf{r} \models \mathscr{M}$

1: $\mathscr{L}_1 = \emptyset$
2: $\mathscr{C}^+(\emptyset) = \mathbf{R}$
3: $n = 1$
4: $\mathscr{L}_1 = \{\text{A} | \text{A} \in \mathbf{R}\}$
5: **while** $\mathscr{L}_n \neq \emptyset$ **do**
6: `computeOFDs`$(\mathscr{L}_n)$
7: $\mathscr{L}_{n+1}$ = `calculateNextLevel`$(\mathscr{L}_n)$
8: $n = n + 1$
9: **end while**
10: **return** $\mathscr{M}$

FASTOFD traverses a *lattice* of all possible sets of attributes in a level-wise manner (Figure 4.1). In level $\mathscr{L}_n$, our algorithm generates candidate ontology FDs with n attributes using `computeOFDs`$(\mathscr{L}_n)$. *FASTOFD* starts the search from singleton sets of attributes and works its way to larger attribute sets through the set-containment lattice, level by level. When the algorithm processes an attribute set $\mathcal{X}$, it verifies candidate ontology FDs of the form $(\mathcal{X} \setminus \text{A}) \rightarrow$ A, where A $\in \mathcal{X}$. This guarantees that only non-trivial ontology FDs are considered. For each candidate, we

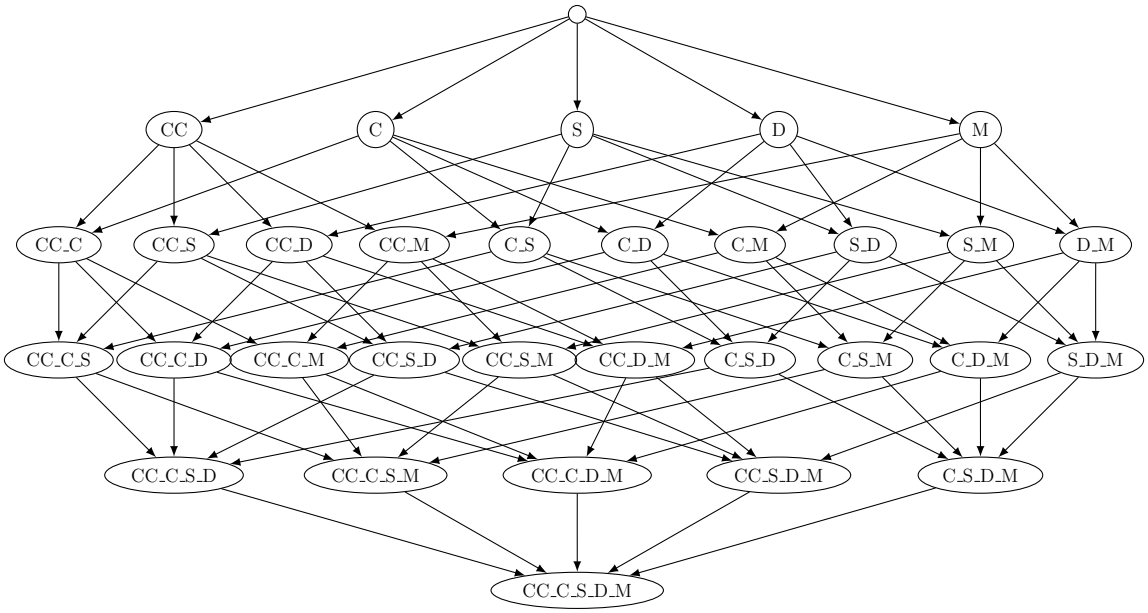

Figure 4.1: Sample Discovery Lattice

invoke `verifySynonymFD`() (Algorithm 4.1), and `verifyGeneralizationFD`() (Algorithm 4.2) to verify whether a synonym or inheritance FD is found.

The small-to-large search strategy of the discovery algorithm guarantees that only ontology FDs that are minimal are added to the output set of ontology FDs $\mathscr{M}$, and is used to prune the search space effectively. The ontology FD candidates generated in a given level are checked for *minimality* based on the previous levels and are added to a *valid* set of ontology FDs $\mathscr{M}$ if applicable. The algorithm `calculateNextLevel`($\mathscr{L}_n$) forms the next level from the current level.

Next, we explain, in turn, each of the algorithms that are called in the main loop of *FASTOFD*.

4.2.1 Finding Minimal OFDs

FASTOFD traverses the lattice until all complete and minimal ontology FDs are found. We deal with ontology FDs of the form $\mathcal{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$, where $\mathrm{A} \in \mathcal{X}$. To check if such an ontology FD is minimal, we need to know if $\mathcal{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$ is valid for $\mathcal{Y} \subset \mathcal{X}$.

If $\mathcal{Y} \setminus \mathrm{A} \rightarrow \mathrm{A}$, then by Augmentation $\mathcal{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$ holds. An ontology FD $\mathcal{X} \rightarrow \mathrm{A}$ holds for any relational instance by Reflexivity, therefore, considering only $\mathcal{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$ guarantees that only non-trivial ontology FDs are taken into account.

We maintain information about minimal ontology FDs, in the form of $\mathcal{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$, in the *candidate* set $\mathscr{C}^+(\mathcal{X})$. If $\mathrm{A} \in \mathscr{C}^+(\mathcal{X})$ for a given set $\mathcal{X}$, then A has not been found to depend on any proper subset of $\mathcal{X}$. Therefore, to find minimal ontology FDs, it suffices to verify ontology FDs $\mathcal{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$, where $\mathrm{A} \in \mathcal{X}$ and $\mathrm{A} \in \mathscr{C}^+(\mathcal{X} \setminus \mathrm{B})$ for all $\mathrm{B} \in \mathcal{X}$.

Example 4.2.1 *Assume that* $\mathrm{B} \rightarrow \mathrm{A}$ *and that we consider the set* $\mathcal{X} = \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$. *As* $\mathrm{B} \rightarrow \mathrm{A}$ *holds,* $\mathrm{A} \notin \mathscr{C}^+(\mathcal{X} \setminus \mathrm{C})$. *Hence, the ontology FD* $\{\mathrm{B}, \mathrm{C}\} \rightarrow \mathrm{A}$ *is not minimal.*

Hence, we define the candidate set $\mathscr{C}^+(\mathcal{X})$, formally as follows.

Definition 4.2.1 $\mathscr{C}^+(\mathcal{X}) = \{\mathrm{A} \in \mathbf{R} | \forall_{\mathrm{A} \in \mathcal{X}} \mathcal{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$ *does not hold*$\}$.

4.2.2 Computing Levels

Algorithm 4.5 explains `calculateNextLevel`($\mathcal{L}_n$), which computes $\mathcal{L}_{n+1}$ from $\mathcal{L}_n$. It uses the subroutine `singleAttrDifferBlocks`($\mathcal{L}_n$) that partitions $\mathcal{L}_n$ into blocks (Line 2). Two sets belong to the same block if they have a common subset $\mathcal{Y}$ of length $n-1$ and differ in only one attribute, A and B, respectively. Therefore, the blocks are not difficult to calculate as sets $\mathcal{Y}\mathrm{A}$ and $\mathcal{Y}\mathrm{B}$ can be preserved as sorted sets of attributes. Other usual use cases of Apriori [1] such as *TANE* [22] and *FASTOD* [38] use a similar approach.

Some of our techniques are similar to *TANE* [22] for FD discovery and *FASTOD* [38] for Ordered Dependencys (ODs) discovery since ontology FDs subsume FDs and ODs subsume FDs. However, *FASTOFD* differs in many details from *TANE* and *FASTOD*, e.g., optimizations, removing the nodes from the lattice and the key pruning rules. *FASTOFD* includes ontology FD-specific rules. For instance, for FDs

if $\{\mathrm{B}, \mathrm{C}\} \rightarrow \mathrm{A}$ and $\mathrm{B} \rightarrow \mathrm{C}$, then $\mathrm{B} \rightarrow \mathrm{A}$ holds, hence, $\{\mathrm{B}, \mathrm{C}\} \rightarrow \mathrm{A}$ is considered non-minimal. However, this rule does not hold for ontology FDs, therefore, our definition of candidate set $\mathscr{C}^+(\mathfrak{X})$ differs from *TANE*.

Algorithm 4.5 calculateNextLevel($\mathfrak{L}_n$)

1: $\mathfrak{L}_{n+1} = \emptyset$
2: **for all** $\{\mathcal{Y}\mathrm{B}, \mathcal{Y}\mathrm{C}\} \in$ `singleAttrDiffBlocks`($\mathfrak{L}_n$) **do**
3: $\quad \mathfrak{X} = \mathcal{Y} \cup \{\mathrm{B}, \mathrm{C}\}$
4: $\quad$ Add $\mathfrak{X}$ to $\mathfrak{L}_{n+1}$
5: **end for**
6: **return** $\mathfrak{L}_{n+1}$

The level $\mathfrak{L}_{n+1}$ contains those sets of attributes of size $n+1$ which have their subsets of size n in $\mathfrak{L}_n$.

4.2.3 Computing Dependencies and Completeness

Algorithm 4.6, *computeOFDs*($\mathfrak{L}_l$), adds minimal ontology FDs from level $\mathfrak{L}_n$ to $\mathscr{M}$, in the form of $\mathfrak{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$, where $\mathrm{A} \in \mathfrak{X}$. The following lemma shows that we can use the candidate set $\mathscr{C}^+(\mathfrak{X})$ to test whether $\mathfrak{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$ is minimal.

Lemma 4.2.1 *An ontology FD* $\mathfrak{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$*, where* $\mathrm{A} \in \mathfrak{X}$*, is minimal iff* $\forall_{\mathrm{B} \in \mathfrak{X}} \mathrm{A} \in \mathscr{C}^+(\mathfrak{X} \setminus \mathrm{B})$.

Proof 4.2.1 *Assume first that the dependency* $\mathfrak{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$ *is not minimal. Therefore, there exists* $\mathrm{B} \in \mathfrak{X}$ *for which* $\mathfrak{X} \setminus \{\mathrm{A}, \mathrm{B}\} \rightarrow \mathrm{A}$ *holds. Then,* $\mathrm{A} \notin \mathscr{C}^+(\mathfrak{X} \setminus \mathrm{B})$.

To prove the other direction assume that there exists $\mathrm{B} \in \mathfrak{X}$*, such that* $\mathrm{A} \notin \mathscr{C}^+(\mathfrak{X} \setminus \mathrm{B})$*. Therefore,* $\mathfrak{X} \setminus \{\mathrm{A}, \mathrm{B}\} \rightarrow \mathrm{A}$ *holds, where* $\mathrm{A} \neq \mathrm{B}$*. Hence, by Reflexivity the dependency* $\mathfrak{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$ *is not minimal.*

By Lemma 4.2.1, the steps in Lines 2, 5, 6 and 7 guarantee that the algorithm adds to $\mathscr{M}$ only the minimal ontology FDs of the form $\mathfrak{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$, where $\mathfrak{X} \in \mathfrak{L}_n$ and $\mathrm{A} \in \mathfrak{X}$. In Line 6, to verify whether $\mathfrak{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$ is a synonym or inheritance

Algorithm 4.6 computeOFDs($\mathfrak{L}_l$)

1: **for all** $\mathfrak{X} \in \mathfrak{L}_l$ **do**
2: $\quad \mathscr{C}^+(\mathfrak{X}) = \bigcap_{\mathrm{A} \in \mathfrak{X}} \mathscr{C}_c^+(\mathfrak{X} \setminus \mathrm{A})$
3: **end for**
4: **for all** $\mathfrak{X} \in \mathfrak{L}_l$ **do**
5: $\quad$ **for all** $\mathrm{A} \in \mathfrak{X} \cap \mathscr{C}^+(\mathfrak{X})$ **do**
6: $\quad\quad$ **if** $\mathfrak{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$ **then**
7: $\quad\quad\quad$ Add $\mathfrak{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$ to $\mathscr{M}$
8: $\quad\quad\quad$ Remove A from $\mathscr{C}^+(\mathfrak{X})$
9: $\quad\quad$ **end if**
10: $\quad$ **end for**
11: **end for**

FD, we invoke `verifySynonymFD`() (Algorithm 4.1), and `verifyGeneralizationFD`() (Algorithm 4.2), respectively.

Lemma 4.2.2 *$\mathscr{C}^+(\mathcal{Y})$ be correctly computed $\forall \mathcal{Y} \in \mathfrak{L}_{n-1}$. `computeOFDs`($\mathfrak{L}_n$) calculates correctly $\mathscr{C}^+(\mathfrak{X})$, $\forall \mathfrak{X} \in \mathfrak{L}_n$.*

Proof 4.2.2 *An attribute* A *is in $\mathscr{C}^+(\mathfrak{X})$ after the execution of `computeOFDs`($\mathfrak{L}_n$) unless it is excluded from $\mathscr{C}^+(\mathfrak{X})$ on Line 2 or 8. First we show that if* A *is excluded from $\mathscr{C}^+(\mathfrak{X})$ by `computeOFDs`($\mathfrak{L}_l$), then $\mathrm{A} \notin \mathscr{C}^+(\mathfrak{X})$ by the definition of $\mathscr{C}^+(\mathfrak{X})$.*

- *If* A *is excluded from $\mathscr{C}^+(\mathfrak{X})$ on Line 2, there exists $\mathrm{B} \in \mathfrak{X}$ with $\mathrm{A} \notin \mathscr{C}^+(\mathfrak{X} \setminus \mathrm{B})$. Therefore, $\mathfrak{X} \setminus \{\mathrm{A}, \mathrm{B}\} \rightarrow \mathrm{A}$ holds, where $\mathrm{A} \neq \mathrm{B}$. Hence, $\mathrm{A} \notin \mathscr{C}^+(\mathfrak{X})$ by the definition of $\mathscr{C}^+(\mathfrak{X})$.*

- *If* A *is excluded on Line 8, then $\mathrm{A} \in \mathfrak{X}$ and $\mathfrak{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$ holds. Hence, $\mathrm{A} \notin \mathscr{C}^+(\mathfrak{X})$ by the definition of $\mathscr{C}^+(\mathfrak{X})$.*

Next, we show the other direction, that if $\mathrm{A} \notin \mathscr{C}^+(\mathfrak{X})$ by the definition of $\mathscr{C}^+(\mathfrak{X})$, then A *is excluded from $\mathscr{C}^+(\mathfrak{X})$ by the algorithm `computeOFDs`($\mathfrak{L}_l$). Assume $\mathrm{A} \notin \mathscr{C}^+(\mathfrak{X})$ by the definition of $\mathscr{C}^+(\mathfrak{X})$. Therefore, there exists $\mathrm{B} \in \mathfrak{X}$, such that $\mathfrak{X} \setminus \{\mathrm{A}, \mathrm{B}\} \rightarrow \mathrm{A}$ holds. We have the following two cases.*

- A = B. *Thus,* $\mathfrak{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$ *holds and* A *is removed on Line 8, if* $\mathfrak{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$ *is minimal; and on Line 2 otherwise.*

- A ≠ B. *Hence,* $\mathrm{A} \notin \mathscr{C}^+(\mathfrak{X} \setminus \mathrm{B})$ *and* A *is removed on Line 2.*

This ends the proof of correctness of computing the candidate set $\mathscr{C}^+(\mathfrak{X})$, $\forall \mathfrak{X} \in \mathscr{L}_n$.

Next, we show that the ontology FD discovery algorithm produces a complete, minimal set of ontology FD.

Theorem 4.2.1 *The FASTOFD algorithm computes a complete, minimal set of ontology FDs* $\mathscr{M}$.

Proof 4.2.3 *The algorithm* `computeOFDs`$(\mathscr{L}_n)$ *adds to set of ontology FDs* $\mathscr{M}$ *only the minimal ontology FDs. The steps in Lines 2, 5, 6 and 7 guarantee that the algorithm adds to* $\mathscr{M}$ *only the minimal ontology FDs of the form* $\mathfrak{X} \setminus \mathrm{A} \rightarrow \mathrm{A}$, *where* $\mathfrak{X} \in \mathscr{L}_n$ *and* $\mathrm{A} \in \mathfrak{X}$ *by Lemma 4.2.1. It follows by induction that* `computeOFDs`$(\mathscr{L}_n)$ *calculates correctly* $\mathscr{C}^+(\mathfrak{X})$ *for all levels n of the lattice since Lemma 4.2.2 holds. Therefore, the FASTOFD algorithm computes a complete set of minimal ontology FDs* $\mathscr{M}$.

4.2.4 Complexity Analysis

The algorithm complexity depends on the number of candidates in the lattice. The worst case complexity is *exponential* in the number of attributes as there are 2^n nodes. However, the complexity is polynomial in the number of tuples. These results are in line with previous FD [22], inclusion dependency [34], and order dependency [38] discovery algorithms.

Since the solution space for minimal ontology FDs is exponential, a polynomial time algorithm in the number of attributes cannot exist. The same conclusions have been reached for the discovery of traditional FDs and inclusion dependencies [22].

However, the algorithm for the discovery of ODs in Langer and Naumann [28] has a *factorial* worst-case time complexity. (ODs subsume FDs.) This is because ODs are defined over lists of attributes, hence, the search space is represented as a lattice of attribute permutations, which results in a factorial number of OD candidates, as there are $\lfloor |\mathbf{R}|! \times e \rfloor$ nodes. However, recently in Szlichta et al. [38] the authors translate ODs into an equivalent set-based canonical form that allows to efficiently discover ODs by traversing a set-containment lattice with exponential worst-case time complexity in the number of attributes and linear in the number of tuples.

For ontology FDs, the ontological relationship (synonym, inheritance, or component) influences the complexity of the verification task. We assume values in the ontology are indexed, and can be accessed within a constant factor. To verify whether a synonym FD holds over $\mathbf{r}$, for each $x \in \Pi_X(\mathbf{r})$, we check whether the intersection of the canonical classes over the consequent values is non-empty. This leads to a worst case time complexity that is quadratic in the number of tuples. A similar argument (checking the least common ancestor applies for an inheritance FD, leading to a worst case complexity that is cubic in the number of tuples.

4.2.5 Approximate ontology FDs

Up to now, we focus on the discovery of ontology FDs that hold over the entire relational instance $\mathbf{r}$.

In practice, some applications do not require such a strict notion of satisfaction, and ontology FDs may not hold exactly over the entire relation due to errors in the data. In such cases, approximate ontology FDs, which hold over a subset of $\mathbf{r}$ are useful. Similar to previous work on approximate FD discovery, we define a minimum support level, τ, that defines the minimum number of tuples that must satisfy an ontology FD $\mathscr{M}$. We define the problem of approximate ontology FD discovery as follows: given a relational instance $\mathbf{r}$, and a minimum support threshold $\tau, 0 \leq \tau \leq 1$,

find all minimal ontology FDs $\mathscr{M}$ such that $s(\mathscr{M}) \geq \tau$ where $s(\mathscr{M}) = max\{|\mathbf{r}'| \mid \mathbf{r}' \subseteq \mathbf{r}, \mathbf{r}' \models \mathscr{M}\}$.

The main modification to discover approximate ontology FDs is in the verification step of checking whether a candidate is a synonym or inheritance FD. The candidate generation and optimization steps remain the same. This requires first identifying the tuples participating in a synonymous or inheritance relationship, and checking whether the number of satisfying tuples is greater than or equal to τ. For synonyms, we check for the maximum overlap among a set of values under a common sense (Line 4, Algorithm 4.1), and check whether the number of satisfying tuples satisfies our minimum support level τ. Similarly, we look for the maximal number of satisfying tuples under a least common ancestor for an inheritance FD.

Chapter 5

Data Cleaning

Data cleaning is the process of taking a set of constraints and a relation in violation of those constraints, then changing the relation by non-trivial means to realign the relation with the constraints. Non-trivial because the simplest modification deletes all records that produce violations (often undesirable behavior) [8]. Given a set of ontology FD constraints and a relation, we present our novel algorithms for computing a cost optimal set of changes to minimize the number of changes while realigning the relation with the constraints. We approach the problem in a method that facilitates parallelization and also distribution of the solution among multiple nodes.

Formally, given a set of ontology FDs $\mathscr{M}$ over a relation $\mathbf{r}$, such that each $F \in \mathscr{M}$ takes the form $\mathscr{X} \mapsto \mathscr{Y}$ we propose to solve where the verification of $\mathscr{M}$ over $\mathbf{r}$ does not hold. The issue to overcome is that of overlapping dependencies. That is for some $F_1 : \mathscr{X} \mapsto \mathscr{Y} \in \mathscr{M}$ and $F_2 : \mathscr{W} \mapsto \mathscr{Z} \in \mathscr{M}$, $\mathscr{W} \cap \mathscr{X} \cap \mathscr{Y} \cap \mathscr{Z} \neq \emptyset$.

The cardinality minimality repair (minimizing the total number of changes) for traditional FDs is NP-hard [7, 8] and ontology FDs subsume traditional FDs, hence, cardinality minimal repair for ontology FDs is NP-hard, too. Since the problem is NP-hard, we develop greedy cost optimal algorithms that are effective in practice to overcome the NP-hard nature of the problem.

5.1 Gating

Gating allows us to independently clean sets of ontology FDs that are disjoint in their attributes. Doing this allows simultaneous data cleaning of these subsets. In Algorithm 5.1 we start by defining a set of sets of attributes $\mathfrak{X}$ and an iterator i. For each ontology FD we first initialize an empty set in $\mathfrak{X}$. We then proceed through each attribute in all ontology FD. On Line 6 we return the sets from $\mathfrak{X}$ containing the current attribute A from the current ontology FD. If an attribute A is not yet in any set of attributes $\mathfrak{X}$ within $\mathfrak{X}$ we add attribute A to the set of attributes for this FD i.e. $\mathfrak{X}_i$. If however attribute A has already been assigned to a previous set in $\mathfrak{X}$, we merge that set into the current set and remove the old set from the set of attribute sets $\mathfrak{X}$.

Algorithm 5.1 Gate Ontology FDs

Input: Set of ontology FDs $\mathscr{M}$
Output: Sets of attributes $\{\mathfrak{X}_1, \ldots, \mathfrak{X}_n\}$

1: $i = 1$
2: $\mathfrak{X} = \emptyset$
3: **for all** $F \in \mathscr{M}$ **do**
4: $\quad \mathfrak{X}_i = \emptyset$
5: $\quad$ **for all** A $\in F$ **do**
6: $\quad\quad \mathscr{Y} = \mathfrak{X}_n |$A $\in \mathfrak{X}_n$ and $\mathfrak{X}_n \in \mathfrak{X}$, else $\emptyset$
7: $\quad\quad$ **if** $\mathscr{Y} = \emptyset$ **then**
8: $\quad\quad\quad \mathfrak{X}_i = \mathfrak{X}_i \cup \{A\}$
9: $\quad\quad$ **else if** $\mathfrak{X}_i \neq \mathscr{Y}$ **then**
10: $\quad\quad\quad \mathfrak{X}_i = \mathfrak{X}_i \cup \mathscr{Y}$
11: $\quad\quad\quad \mathfrak{X} = \mathfrak{X} \setminus \mathscr{Y}$
12: $\quad\quad$ **end if**
13: $\quad$ **end for**
14: $\quad i = i + 1$
15: **end for**
16: **return** $\mathfrak{X}$

Example 5.1.1 *For example, our ontology FDs from Table 1.1 were* F_1 : *[Country Code]* $\overset{s}{\mapsto}$ *[Country],* F_2 : *[Symptom, Diagnosis]* $\overset{g}{\mapsto}_1$ *[Medicine], and* F_3 : *[Diagnosis]* $\overset{c}{\mapsto}$ *[Symptom]. We begin with* F_1. $\mathfrak{X} = \{\mathfrak{X}_1 = \{\}\}$. *To start, we see if Country*

Code is in $\mathfrak{X}$. It is not in any set within $\mathfrak{X}$, so we add it to $\mathfrak{X}_1$. $\mathfrak{X} = \{\mathfrak{X}_1 = \{$Country Code$\}\}$. We next check for Country in $\mathfrak{X}$, which is not in any set within $\mathfrak{X}$, so we also add it to $\mathfrak{X}_1$. Therefor, $\mathfrak{X} = \{\mathfrak{X}_1 = \{$Country Code, Country$\}\}$. We have processed all attributes in F_1 so we begin the next ontology FD. To start, we create a new subset in $\mathfrak{X}$. $\mathfrak{X} = \{\mathfrak{X}_1 = \{$Country Code, Country$\}, \mathfrak{X}_2 = \{\}\}$. We check if Symptom is in $\mathfrak{X}$, it is not in any set, so we add it to $\mathfrak{X}_2$. $\mathfrak{X} = \{\mathfrak{X}_1 = \{$Country Code, Country$\}, \mathfrak{X}_2 = \{$Symptom$\}\}$. We next lookup Diagnosis in $\mathfrak{X}$, not in any set, so we add it to $\mathfrak{X}_2$. $\mathfrak{X} = \{\mathfrak{X}_1 = \{$Country Code, Country$\}, \mathfrak{X}_2 = \{$Symptom, Diagnosis$\}\}$. We check if Medicine is in $\mathfrak{X}$, it is also not in any set, so we add it to $\mathfrak{X}_2$. $\mathfrak{X} = \{\mathfrak{X}_1 = \{$Country Code, Country$\}, \mathfrak{X}_2 = \{$Symptom, Diagnosis, Medicine$\}\}$. We start the last dependency, thus we create a new subset in $\mathfrak{X}$. $\mathfrak{X} = \{\mathfrak{X}_1 = \{$Country Code, Country$\}, \mathfrak{X}_2 = \{$Symptom, Diagnosis, Medicine$, \mathfrak{X}_3 = \{\}\}$. Now we check if Medicine is in $\mathfrak{X}$, it is! It is in $\mathfrak{X}_2$, in this case we add all of $\mathfrak{X}_2$ it to $\mathfrak{X}_3$ and delete $\mathfrak{X}_2$. Now, $\mathfrak{X} = \{\mathfrak{X}_1 = \{$Country Code, Country$\}, \mathfrak{X}_3 = \{$Symptom, Diagnosis, Medicine$\}\}$. Finally, we lookup Diagnosis in $\mathfrak{X}$, it is in $\mathfrak{X}_3$ but we are currently assigning new attributes to $\mathfrak{X}_3$ so we do nothing. This is the end of all attributes of all dependencies, therefore we return $\{\{$Country Code, Country$\}$, $\{$Symptom, Diagnosis, Medicine$\}\}$. With this, each set of attributes can be cleaned independently.

5.2 Data Cleaning

We begin by cleaning the relation under a single ontology FD and later generalize to clean a set of dependencies. We focus our approach to cleaning the RHS. This approach allows dramatically more concurrency, similar to the one used by W. Fan et al., in [8]. As each LHS equivalence class is independent in write access, no lock is needed over the whole attribute set.

First, we identifying when an equivalence class is in violation of an ontology FD constraint. We re-purpose `verifySynonymFD()` (Algorithm 4.1), `verifyComponentFD()` (Algorithm 4.3), and `verifyGeneralizationFD()` (Algorithm 4.2) to compute this. We replace the return false call with instead a dispatch to the corresponding function below (synonyms to Algorithm 5.2, components to Algorithm 5.3, and generalizations to Algorithm 5.4) providing the RHS as input.

Algorithm 5.2 Clean Synonym FD Equivalence Class

Input: Invalid equivalence class RHS $\mathfrak{z}$
Output: Cleaned equivalence class RHS $\mathfrak{z}$

```
1: v = 0
2: D = ∅
3: dMap[] = {0, ..., 0}
4: for all t ∈ 𝔷 do
5:    for all C ∈ classes(t) do
6:       dMap[C] = dMap[C] + 1
7:       if dMap[C] > v then
8:          D = C
9:          v = dMap[C]
10:      end if
11:   end for
12: end for
13: s = u1 ∈ synonyms(D)
14: for all t ∈ 𝔷 do
15:   if D ∉ classes(t) then
16:      t = s
17:   end if
18: end for
19: return 𝔷
```

The synonym FD cleaning algorithm initializes three variables for counting support: $dMap[]$, v, and $\mathbf{D}$. $dMap[]$ is a map (associative array) which is used to count the number of times each class is seen, v is used to track the count of the current popular candidate, and $\mathbf{D}$ is used to store the current popular candidate. $\mathbf{D}$ and v trade a small piece of memory to prevent a scan of the counting map after the instances have been tallied.

Next, the algorithm iterates over every tuple, for each tuple looking up the set

of classes to which it belongs. It increments the count of each class and if a count surpasses the current popular class it is promoted. After counting, the canonical name of the popular class is assigned to s. We assume the first synonym of the popular class is in fact the preferred term of that class.

A possible second map may improve accuracy. If a simultaneous count is made to track the most popular string term, then at Line 13 the algorithm may test to see that the popular string term is a synonym of the popular class $\mathbf{D}$. If this is the case, the most popular string term may be used instead of the canonical name of the class. This increases the runtime and memory consumption of the algorithm and may or may not aid accuracy. In the worst case, the string term is not a synonym of the popular class. In this case, the canonical name must be used instead, negating all this extra work.

Moving forward, the equivalence class is again scanned for tuples whose string term in `classes`() does not return the popular class. When such a tuple is found, it is replaced with the chosen string term. This process is similar in all following verification algorithms.

The component FD cleaning algorithm begins similar to Algorithm 5.2. Instead of a popular class however, a popular component set $\mathfrak{D}$ is found instead. When selecting the string term, the canonical name of the primary component is selected in Algorithm 5.3. Similar to Algorithm 5.2, counting the popular string term may or may not improve accuracy depending on the dataset and ontology. However, when testing the popular string term it must be a member of any class in the component set to be valid.

In Algorithm 5.4 for cleaning generalization FDs we again identify the most popular class, however, we do so from the set of all ancestors of a path length not longer than θ. Notice Line 6, here we pull each class from the ancestors set one at a time starting at the class itself and stopping when the path traversed is at most $\theta + 1$

Algorithm 5.3 Clean Component FD Equivalence Class

Input: Invalid equivalence class RHS $\mathfrak{z}$
Output: Cleaned equivalence class RHS $\mathfrak{z}$

1: $v = 0$
2: $\mathfrak{D} = \varnothing$
3: $dMap[] = \{0, \ldots, 0\}$
4: **for all** $t \in \mathfrak{z}$ **do**
5: **for all** $\mathbf{C} \in \texttt{classes}(t)$ **do**
6: **for all** $\mathfrak{C} \in \texttt{composes}(\mathbf{C})$ **do**
7: $dMap[\mathfrak{C}] = dMap[\mathfrak{C}] + 1$
8: **if** $dMap[\mathfrak{C}] > v$ **then**
9: $\mathfrak{D} = \mathfrak{C}$
10: $v = dMap[\mathfrak{C}]$
11: **end if**
12: **end for**
13: **end for**
14: **end for**
15: $s = u_1 \in \texttt{synonyms}(\mathbf{C}_1 \in \mathfrak{D})$
16: **for all** $t \in \mathfrak{z}$ **do**
17: **for all** $\mathbf{C} \in \texttt{classes}(t)$ **do**
18: **if** $\texttt{composes}(\mathbf{C}) \notin \mathfrak{D}$ **then**
19: $t = s$
20: **end if**
21: **end for**
22: **end for**
23: **return** $\mathfrak{z}$

Algorithm 5.4 Clean Generalization FD Equivalence Class

Input: Invalid equivalence class RHS $\mathfrak{z}$

Output: Cleaned equivalence class RHS $\mathfrak{z}$

1: $v = 0$
2: $\mathbf{D} = \varnothing$
3: $dMap[] = \{0, \ldots, 0\}$
4: **for all** $t \in \mathfrak{z}$ **do**
5: **for all** $\mathbf{C} \in \texttt{classes}(t)$ **do**
6: **for all** $\{\mathbf{C}_1, \ldots, \mathbf{C}_{\theta+1}\} \in \texttt{ancestors}(\mathbf{C})$ **do**
7: $dMap[\mathbf{C}] = dMap[\mathbf{C}] + 1$
8: **if** $dMap[\mathbf{C}] > v$ **then**
9: $\mathbf{D} = \mathbf{C}$
10: $v = dMap[\mathbf{C}]$
11: **end if**
12: **end for**
13: **end for**
14: **end for**
15: $s = u_1 \in \texttt{synonyms}(\mathbf{D})$
16: **for all** $t \in \mathfrak{z}$ **do**
17: $\delta = \texttt{LCA}(\texttt{ancestors}(\texttt{classes}(t)), \{\mathbf{C}\})$
18: **if** $\theta < \delta < \infty$ **then**
19: $s = u_1 \in \texttt{synonyms}(\mathbf{E}_{\delta-\theta+1} \in \texttt{ancestors}(\texttt{classes}(t)))$
20: **else if** $\theta < \delta$ **then**
21: $t = s$
22: **end if**
23: **end for**
24: **return** $\mathfrak{z}$

elements. This search attempts to identify the most popular ancestor within the required threshold. Again, counting the string terms is a possible accuracy boosting method.

The next big difference is in replacing the string terms. Here we use `LCA`() to identify when the violating tuple is a distant descendant of the popular ancestor (distant being over a greater distance than θ). This helps fix when violations are the result of over specificity in a string term. When we find that a term is too specific, it makes the most sense to generalize the term until it is within threshold of the popular ancestor. In the case that the violating term is not a descendant, baring a count of popular string terms, the only sensible alternative is to use the canonical name of the generalized ancestor.

	Country Code	Country	Symptom	Diagnosis	Medicine
t_1	US	United States	joint pain	osteoarthritis	ibuprofen
t_2	IN	India	joint pain	osteoarthritis	NSAID
t_3	CA	Canada	joint pain	osteoarthritis	naproxen
t_4	IN	Bharat	nausea	migraine	acetaminophen
t_5	US	Amirica	nausea	migraine	morphine
t_6	US	USA	tinnitus	migraine	tylenol
t_7	IN	India	chest pain	hypertension	morphine

Table 5.1: Dirty Medical Trials Relation

Example 5.2.1 *To demonstrate these, we will use a modified version of Table 1.1, that is Table 5.1. We already know by Example 5.1.1, that we can evaluate* F_1 : *[Country Code]* $\overset{s}{\mapsto}$ *[Country] independently. As such, we begin by verifying* F_1 *given the ontology in Figure 1.1. We first run verification over the relation. Grouping equivalence classes gives us: (United States, Amirica, USA), (India, Bharat), and (Canada).*

First, we will evaluate "CA", which has a RHS of (Canada). As there is only one value on the RHS, we do not need to clean further since this verifies as a pure FD. Next

we will evaluate (United States, Amirica, and USA). For synonyms, `classes`(*United States*) ∩ `classes`(*Amirica*) ∩ `classes`(*USA*) *must not be an empty set. As we find however,* `classes`(*Amirica*) *returns an empty set. Any intersection with the empty set is also the empty set. Therefore, the equivalence class is falsified. Here we dispatch cleaning Algorithm 5.2. We begin counting class support.* `classes`(*United States*) *returns one class from our ontology. We increment the count of it by one and move on.* `classes`(*Amirica*) *returns nothing so we skip it and move on.* `classes`(*USA*) *also returns one class, that class being the same as returned by* `classes`(*United States*)*. The count of this class is now two. As a result, that class is chosen as the popular class and the canonical name "United States" is selected from that class. Next we scan the equivalence class' RHS for terms that do not have the popular class as a class of theirs. In this case the only term is "Amirica". This term is replaced with "United States" and the tuple values becomes: (United States, United States, USA).*

We will also cover another dirty equivalence class in the dependency F_2 : *[Symptom, Diagnosis]* $\overset{g}{\mapsto}_1$ *[Medicine]. During verification we notice that the equivalence class "nausea, migraine" has a RHS of (acetaminophen, morphine) which falsifies the dependency given the ontology in Figure 1.2. We will now run through this as an example of cleaning a generalization FD. We begin by collecting all ancestral paths for the classes of the string terms. This gives us* {*paracetamol, analgesic*} *for the term "acetaminophen" (note the terms in brackets are the canonical names of the classes actually returned), and* {*morphine, opioid, analgesic*} *for the term "morphine". We next count each class from each path up to maximum length of 1 (that is the* θ *distance defined for this generalization FD). For "acetaminophen" this means the class of paracetamol and analgesic get counted once. For "morphine" this means the classes of morphine and opioid both get counted once. Notice we do not count analgesic for "morphine" as it is not within the path length. Here the most popular class was paracetamol as it was the first class to reach the count of 1. As such, the canonical*

name "paracetamol" is selected. Now we scan the tuples one at a time. First, "acetaminophen" does not need to change as its `LCA()` *distance to the class paracetamol is 0 (being they are synonyms of the same drug). Next "morphine" is tested. Its* `LCA()` *distance is infinite, therefore the term is replaced with "paracetamol" and the algorithm finishes.*

5.2.1 Dependency Ordering

We now extend our methodology to cases of overlapping dependencies. When a relation is inconsistent over more than one constraint, and these constraints overlap, our algorithms must choose an order for processing the constraints within the same gating bucket. Similarly as in other work with data cleaning over traditional FDs [10], we consider two criteria for ordering constraints. First, we consider the degree of inconsistency of each dependency as a whole. An $\mathfrak{X}$ value is inconsistent if it fails to validate. The degree of inconsistency for $\mathfrak{X}$ is the number of tuples in $\mathbf{r}$ that fail to validate under the dependency $F : \mathfrak{X} \mapsto \mathrm{A}$.

Definition 5.2.1 *The degree of inconsistency (ic_F) of F over $\mathbf{r}$ is:*

$$ic_F = \frac{\Sigma_{s \in \Pi_{\mathfrak{X}}(\mathbf{r})}(\texttt{verify}(\mathbf{r}, \mathfrak{X}, \mathrm{A}))}{|\Pi_{\mathfrak{X}\mathscr{A}}(\mathbf{r})|}$$

If the relation $\mathbf{r}$ is consistent $ic_F = 0$. As ic_F approaches 1, the relation becomes more inconsistent w.r.t. F. The second criteria we consider are the potential conflicts F shares with other inconsistent constraints F', defined based on the number of attributes they have in common ($|F \cap F'|$).

Definition 5.2.2 *The conflict score of an ontology FD F composed of attributes $\mathfrak{X}\mathrm{A}$ from all attributes in the bucket $\mathfrak{L}$ is:*

$$cf_F = \frac{\Sigma_{F' \in \mathfrak{L}} \frac{|F \cap F'|}{\max(|F|,|F'|)}}{|\mathfrak{L}|}$$

The conflict score cf_F scales from 0 (no overlap) to 1 indicating the overlap of all attributes among all dependencies in the gated bucket. Of course zero is impossible due to gating applied from Section 5.1. Constraints with high cf_F values have the greatest potential for repair conflicts with other rules. Since both our evaluation scores are normalized, we average the two values to get a combined score O_F :

$$O_F = \frac{ic_F + cf_F}{2}$$

We evaluate multiple constraints in decreasing O_F order, since large values indicate rules with the highest degree of inconsistency and the greatest potential of repair conflicts with other constraints.

Example 5.2.2 *Here we will discuss ordering dependencies that contain {Symptom, Diagnosis, Medicine}. This would be* F_2 : *[Symptom, Diagnosis]* $\overset{g}{\mapsto}_1$ *[Medicine] and* F_3 : *[Diagnosis]* $\overset{c}{\mapsto}$ *[Symptom].*

We first calculate ic_F *for each. In* F_2*, we have 4 equivalence classes: "joint pain, osteoarthritis", "nausea, migraine", "tinnitus, migraine", and "chest pain, hypertension". Of these, 1 is broken. This broken equivalence class contains 2 tuples. Therefore* $ic_{F_2} = \frac{2}{7} \approx 0.285714$*. In* F_3*, we have 3 equivalence classes: "osteoarthritis", "migraine", and "hypertension". None of which are dirty. Therefore* $ic_{F_3} = \frac{0}{7} = 0$*.*

Next we must calculate cf_F*. First,* $|\{Symptom, Diagnosis, Medicine\} \cap \{Diagnosis, Symptom\}| = |\{Diagnosis, Symptom\}| = 2$*. Using this for* F_2*, we calculate* $\frac{2}{\max(|F_2|,|F_2'|)}$*. There are 3 attributes in* F_2 *which leaves 0 unused, the maximum of 3 and 0 is 3. So the value becomes* $cf_{F_2} = \frac{2}{3} \approx 0.666667$*. For* F_3 *we do similarly, except there are 2 attributes in* F_2 *which leaves 1 unused, the maximum of 2 and 1 is 2. Therefore* $cf_{F_2} = \frac{2}{2} = 1$*. This leads to us computing:*

$$O_{F_2} = \frac{ic_{F_2} + cf_{F_2}}{2} = \frac{\frac{2}{7} + \frac{2}{3}}{2} \approx 0.4761905$$

$$O_{F_3} = \frac{ic_{F_3} + cf_{F_3}}{2} = \frac{\frac{0}{7} + \frac{2}{2}}{2} = 0.5$$

Therefore we first clean F_2 then we clean F_3.

5.3 Ontology vs Data Cleaning

In dynamic environments, the needs of an organization will change. This change over time leads to ontologies also becoming dirty or outdated. As an example, imagine a new drug is approved for use in patients with a certain disease. If the ontology is not properly updated to reflect this, use of this new drug will be flagged as a violation in a relation. If cleaning is to be trusted, identifying when the real world has changed is crucial to making good data correction suggestions. We also noticed in our research that ontologies themselves suffer from data quality issues. To address both of these problems, we present Algorithm 5.5 in order to scan a dependency to test if the ontology is missing likely valid data from the relation.

Algorithm 5.5 Compute Ontology Validity

Input: Relation **r**, set of attributes $\mathfrak{X}$, an attribute A, and damage deviation σ
Output:true if ontology is at fault, otherwise false

1: $sMap[] = \{0, \ldots, 0\}$
2: **for all** $\mathfrak{z} \in \Pi_{\mathrm{X}}(\mathbf{r})$ **do**
3: **for all** $t \in \Pi_{\mathrm{A}}(g_{\mathfrak{z}}[\mathfrak{X}])$ **do**
4: **for all** $s \in t$ **do**
5: **if** $|\texttt{classes}(s)| = 0$ **then**
6: $sMap[s] = sMap[s] + 1$
7: **end if**
8: **end for**
9: **end for**
10: **end for**
11: **if** $\frac{\Sigma\{sMap[s] | sMap[s] > 1\}}{|t|} \geq \sigma$ **then**
12: **return true**
13: **else**
14: **return true**
15: **end if**

Our algorithm searches values of the RHS attribute to test what string terms fail

to validate as a result of not being in the ontology. In our algorithm, a string must be seen at least twice in the relation to be counted. This search gathers support for terms that, as a result of appearing often, should likely be included in the ontology. This likelihood is controlled by the user supplied threshold value σ.

Example 5.3.1 *Let us examine Table 5.1 again, but this time we will assume the class containing "nausea" from the ontology in Figure 1.3 is missing. We scan* F_3 : *[Diagnosis]* $\overset{c}{\mapsto}$ *[Symptom] to see if we have evidence of a dirty ontology. We begin by grouping by equivalence classes of which this dependency has 3: "osteoarthritis", "migraine", and "hypertension". Next we count the number of classes for each string term under the equivalency class. The RHS of "osteoarthritis" is (joint pain, joint pain, joint pain). They each have more than 0 classes so nothing needs to be done. Now we examine "migraine". "migraine" has a RHS consisting of (nausea, nausea, tinnitus). In this case, we count each term's classes.* $|\texttt{classes}(nausea)| = 0$ *therefore we increment the count on the term. Again, we find "nausea" and increment the count. Finally,* $|\texttt{classes}(tinnitus)| = 1$ *and we are done. Now we test to see if this equivalence class has ontology problems. We begin by discarding all counts of 1 from the map. We have none so we move on. Next we calculate* $\frac{\Sigma\{2\}}{|3|}$. $\frac{2}{3} \geq 0.66$ *therefore this equivalence class is likely broken because of the ontology (which it is). Finally "hypertension" can be tested, but (chest pain) is in the ontology so we're done.*

Chapter 6

Experiments

We present an experimental evaluation of our techniques. Our evaluation focuses on three areas. In Section 6.2, we evaluate the performance of our data verification algorithms. In Section 6.3, we evaluate the performance and precision of our cleaning algorithms. We note the difference in the performance between concurrent and serial cleaning obtained from our algorithms. Finally, in Section 6.4 we evaluate the performance, scalability, and other influencing factors in our ontology versus data repair algorithm.

6.1 Setup

Our experiments were run on an Intel Xeon CPU E5-2630 v3, 2.40GHz with 8GB of memory. Each experiment was run three times (unless otherwise noted) and the average time is reported. The algorithms were implemented in the Go programming language. Our ontology is made partly from extracting the ontology directly from the relation, and partly from a real world ontology [26]. Our dataset contains 1,000,000 tuples from the Linked Clinical Trials (LinkedCT.org) database [20]. The LinkedCT.org project provides an open interface for international clinical trials data. The XML version of this data was transformed into open linked data [30].

6.2 Constraint Verification

For data verification, a manually cleaned version of the database was tested. When testing, the algorithms were run 10 times and the average time is reported. Graph 6.1 shows the results of our experiments. Our technique shows a linear scale-up in the number of tuples. For our chosen datasets, we found there was a large number of smaller equivalence classes, which lead to the decreased verification time that in turn dominated the overall running time.

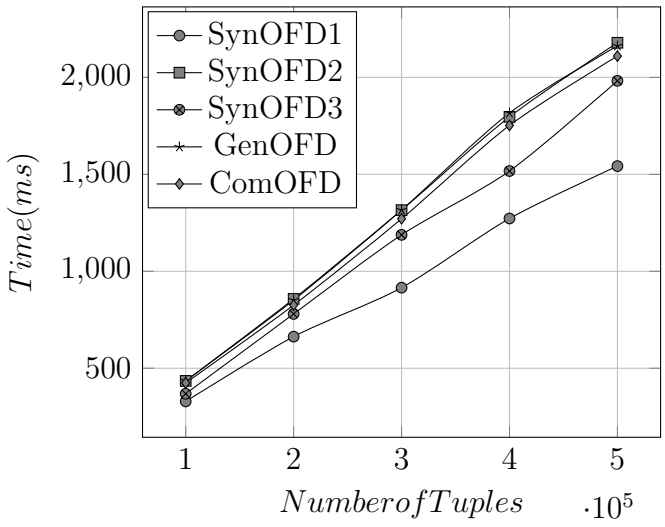

Figure 6.1: Ontology FD Verification Complexity

6.3 Data Cleaning

When testing data cleaning, we started with a clean copy of the relation then wrote a script to randomly introduce errors into the relation as a percentage of the total number of tuples in the relation. These mutations were performed by either changing a letter to simulate a mistyped string or by replacing the value with a term selected at random from the ontology. The latter methodology attempts to simulate a user making a wrong selection during input such as from a drop down menu. Both operations

were given equal probability.

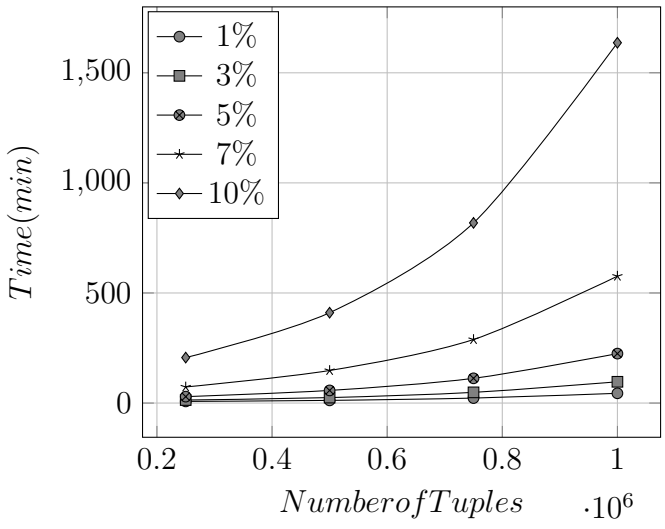

Figure 6.2: Serial Ontology FD Cleaning

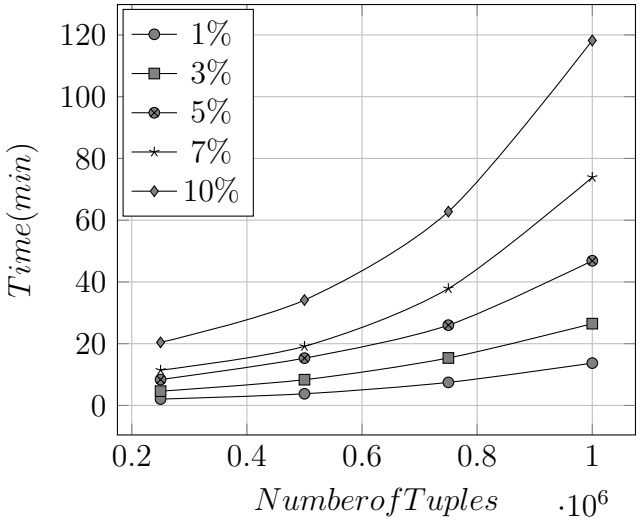

Figure 6.3: Concurrent Ontology FD Cleaning Complexity

When evaluating the accuracy in Graph 6.4, 100 damaged tuples were randomly selected and their values before and after cleaning are presented to a user. The user provided their best judgment to determine if the values were analogous enough to be considered a successful cleaning result. This was done because the canonical name of a class and the most generalized form of a taxonomy are not always the best fit

for a data repair. Direct equality comparison does not work because of the semantic equivalence introduced by ontology FDs.

Graphs 6.2 and 6.3 compare the running scalability and time between serial and concurrent execution modes. Both show exponential complexity in the number of tuples. We note that concurrent execution reduces the overall runtime by more than one order of magnitude for the worst case complexity (10% error rate over 1 million tuples). We see runtime improvements in all cases. The disparity between 1% and 10% error rates appears to scale in the number of equivalence classes broken.

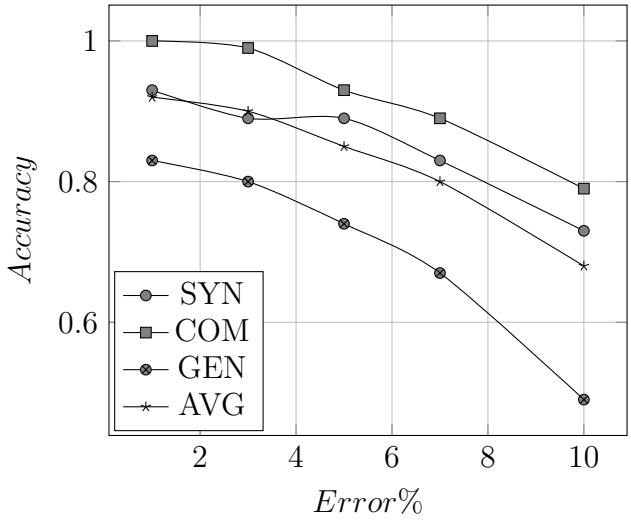

Figure 6.4: Concurrent Ontology FD Cleaning Precession

We move to test the precision of our cleaning algorithms in Graph 6.4. We note that precision decays as the percentage of errors in the dataset increases. It can be seen that generalizations performed worse that synonyms or components. This was caused by most terms being replaced with their most generalized form. Additionally, 100% accuracy is achieved with component FDs at 1% error rates. Although component repairs did suggest components that were not the same as the original, the components themselves given the equivalence class they were under still made the repairs accurate enough to be correct.

6.4 Ontology Versus Data Cleaning

Testing ontology versus data cleaning ran similar to data cleaning. The same random mutation script was used to introduce errors into tuples. We then added a subroutine that would randomly drop 10% percent of the classes from our ontology. This introduced a number of errors that would break many equivalence classes. During mutation, we noted which equivalence classes' RHS contained a string term from a deleted class. During testing, if the algorithm identified an equivalence class was as broken because of the ontology and in fact one of the string term's classes was missing, we awarded it a victory.

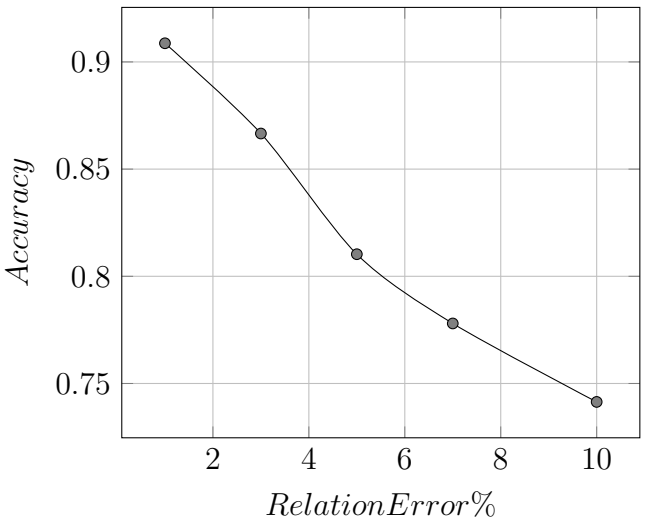

Figure 6.5: Ontology Versus Relation Precision by Damage

In Graph 6.5, we chart the damage to the relation against the accuracy achieved. As expected, the accuracy drops as the dataset becomes less reliable. This makes sense as ontology damage usually results in few to no terms on the RHS being found in any class within the ontology. With more damage, it is less likely that many terms will agree amongst themselves, yet not appear in the ontology.

Graphs 6.6 and 6.7 chart the scalability of our algorithm over the number of input tuples and ontology damage respectively. Both see linear growth despite our

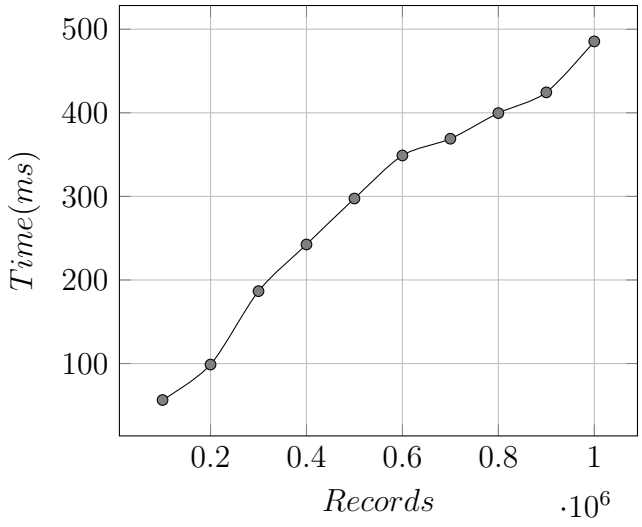

Figure 6.6: Ontology Versus Relation Scalability by Input Size

algorithm's worst case complexity being quadratic. This is because the scalability is actually determined by the number of classes from terms on the RHS of each equivalence class, summed over all equivalence classes. Therefore we see linear growth when we add more equivalence classes, and when the number of unique terms grows as is the case when they contain erroneous data.

Initially we ran the algorithm with a threshold of 66%. We experimented with this value and the results are charted in Graph 6.8. We see that our choice was fairly good, however, we notice our algorithm could have performed better given a higher accuracy such as 70%. We also notice that the accuracy is most harmful when set too low as our accuracy is only 5.38% better than a threshold of 0.9.

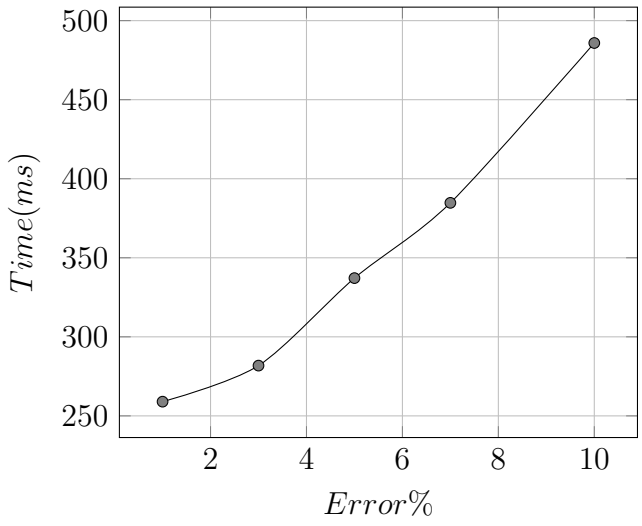

Figure 6.7: Ontology Versus Relation Scalability by Damage

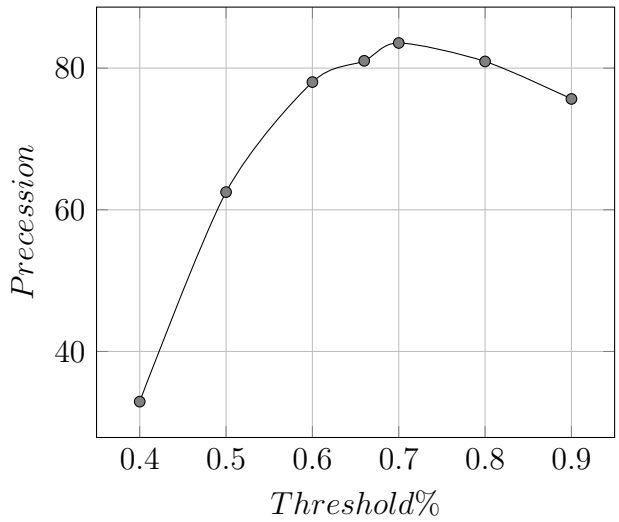

Figure 6.8: Ontology Versus Relation Precision by Threshold

Chapter 7

Related Work

Our work finds similarities to dependency discovery in databases and to defining integrity constraints over ontologies and graphs. We first discuss the relationship between ontology FDs and two types of dependencies used in data cleaning (FDs, and metric FDs), and then discuss related work.

Synonym FDs subsume FDs, since we can create a database where all values have a single string representation, i.e., for all classes $\mathbf{C}$, $|\texttt{synonyms}(\mathbf{C})| = 1$. If we set $\theta = 0$, then an inheritance FD becomes a synonym FD, thus, inheritance FDs subsume traditional FDs. Metric FDs are defined when two tuples agree on $\mathcal{X}$, then the $\mathcal{Y}$ values must have similar values w.r.t. some metric distance [27,35]. Ontology FDs, however, are defined over the values in equivalence classes in $\Pi_{\mathcal{X}\mathcal{Y}}$, and there must exist a common class across these values.

id	X	Y	Classes for Y
t_1	b	c	$\{\mathbf{C}, \mathbf{D}\}$
t_2	b	d	$\{\mathbf{D}, \mathbf{E}, \mathbf{F}\}$
t_3	b	e	$\{\mathbf{C}, \mathbf{E}\}$

Table 7.1: Defining Ontology FDs.

Consider Table 7.1, where synonym FD $\mathcal{X} \overset{s}{\mapsto} \mathcal{Y}$ is falsified. For each $\mathcal{Y}$ value, the

defined classes are given in the last column. Although all pairs of $t[\mathcal{Y}]$ values share a common class (i.e., $\{c, d\} : \mathbf{D}, \{c, e\} : \mathbf{C}, \{d, e\} : \mathbf{E}$), the intersection of the classes (for each $\Pi_{\mathfrak{X}}$ value) is empty. Furthermore, ontological similarity is not a metric distance since it does not satisfy the identity of indiscernibles (e.g., for synonyms). Thus, ontology FDs are not a subclass of metric FDs (and vice versa).

Dependency discovery involves mining for all dependencies that hold over a data instance. This includes discovery of functional dependencies (FDs) [22, 29, 32, 33, 44], conditional functional dependencies (CFDs) [9, 16, 19], inclusion dependencies [34], order dependencies [38], matching dependencies [37], and denial constraints [11]. In previous FD discovery algorithms, both TANE [22] and DepMiner [29] search the attribute lattice in a level-wise manner for a minimal FD cover. In CFD discovery algorithms, a similar lattice traversal is used to identify a subset of tuples that functionally hold over a relational instance [9, 16]. In our work, we generalize the lattice based level-wise search strategy for discovering synonym and inheritance FDs.

Previous work have extended classical FDs to consider attribute domains that contain a partial order, and to support time-related dependencies in temporal databases [23, 41–43]. Wijsen et al., propose Roll-Up Dependencys (RUDs) that generalize FDs for attribute domains containing concept hierarchies that are commonly found in data mining and Online Analytical Processing (OLAP) applications [43]. For example, a [Time] attribute contains values that can be organized into a partial order, measured in days, weeks, months, etc. RUDs capture roll-up semantics from one or more attributes that have been aggregated at the finer levels. The set of possible generalizations for an attribute set in a candidate RUD is modelled as a lattice. Similar to our approach, the RUD discovery algorithm traverses the lattice in a levelwise top-down manner. However, our inheritance FDs, in particular, capture a containment semantics (similar to the is-a semantics) that is not modelled by RUDs.

Similar to traditional FDs, Jensen et al., propose Temporal Functional Dependen-

cys (TFDs) that hold over a snapshot of a temporal database, called a timeslice [23]. Extensions of TFDs include constraining tuples across multiple timeslices [41], and generalizing the time model to include objects and classes, whereby an attribute value is no longer necessarily atomic, but may refer to an object of another class [42]. This object referencing implicitly provides referential integrity, but does not consider the synonym nor inheritance relationships we investigate in our work. We focus on identifying synonym and inheritance dependencies containing atomic data types at a given timeslice. In the future, it will be interesting to consider how our dependencies can be extended to include object classes, and their variation over time.

Ontologies are used to model concepts, entities, and relationships for a given domain. Existing techniques have proposed FDs over RUD triples based on the co-occurrence of values. However, the defined FDs do not consider structural requirements to specify which entities should carry the values [2, 21]. Motif et al., define integrity constraints using the Web Ontology Language (OWL). OWL ontologies are often incomplete, whereas many databases in practice are complete [31]. They propose an extension of OWL with integrity constraints to validate completeness in the ontology by defining inclusion dependencies and domain constraints to check for missing values and valid domain values within an ontology. The proposed constraints do not model functional dependencies (as proposed in our work) since the focus is data completeness. Furthermore, these existing techniques do not consider the notion of senses to distinguish similar terms under an interpretation. For example, the term "jaguar" is synonymous with "Mercedes" and "tiger". However, an application would have an interpretation (sense) for "jaguar" either as vehicle or animal.

Fan et al., define keys for graphs based on patterns that specify topological constraints and value bindings to perform entity matching [15]. Keys contain variables that are bound to constant values satisfying node and value equality. The authors focus on the definition of keys (not their discovery), and present three sub-graph

entity matching algorithms that utilize keys. In subsequent work, Fan et al., propose functional dependencies for Graph Functional Dependencys (GFDs) since FDs cannot be expressed via keys [17]. GFDs contain topological constraints to identify the entities participating in the dependency and value bindings (similar to conditional FDs) that specify dependencies among the attribute values. GFDs model is-a relationships (e.g., y is-a x) by assuming this inheritance relationship is known in advance, and then enforce the requirement that for any property A of x must also be true for y, i.e., $x.A = y.A$. In our work, we focus on the *discovery* of is-a relationships where the antecedent attribute values determine inheritance relationships between consequent attribute values. For example, in Table 1.1, tuples t_5 and t_6 indicate that "nausea" and "migraine" symptoms can be treated with medication "tylenol", which is synonymous with "acetaminophen". While our work is similar in spirit, we identify attribute relationships that go beyond equality (i.e., synonyms and inheritance). In contrast to keys, our discovered dependencies are value based (no variables are present). In our work, we consider the notion of senses that states how a dependency should be interpreted, since multiple interpretations are possible for a given ontology; these interpretations are not considered in existing techniques. Lastly, we study the axiomatization and inference of synonym and inheritance relationships in ontology FDs, which were not studied in previous work.

Work by Fan et al. [10] investigates cost minimal data repairs to traditional FDs and a set of constraints simultaneously. Beskales et al. [7] investigates cardinality-set-minimal data repairs, that balance the requirements of minimal data changes (cardinality) and necessary changes (set minimality). While our work is similar in spirit, our model was developed to permit notable performance improvements as a result of gating and equivalence class co-processing. We also apply our work to the domain of ontology FDs, a class of dependencies not considered by these works. Bertossi et al. lay a complexity analysis from which we derived our analysis [6].

Chapter 8

Conclusions

In this work we proposed a new class of dependencies, *Ontology Functional Dependencies* that captures domain relationships found in ontologies. We focused on the synonym, inheritance, and component relationships between attribute values. We proposed a set of data verification, discovery, and cleaning algorithms that work together to offer utility for these dependencies. We produced a set of axioms and an inference system, useful in reasoning about and proving often non-intuitive properties of our dependencies.

Data verification allows us to test a relation given a set of ontology FD for conformity. We showed that our data verification algorithms accurately discerns validity, for synonyms and components in worst case quadratic time, and generalization in worst case cubic time. Our experiments show real world scale is often linear to super-linear depending on variables like average equivalence class size and taxonomy generalization depth.

Dependency discovery allows us to search a relation for ontology FD that hold over it. We introduce the notion of approximate ontology FDs to allow us to identify when ontology FDs might hold despite minor inconsistencies in the relation. We show that our discovery approach achieves linear-time inference in the number of tuples

and exponential time in the number of attributes. It also ensures that the discovered set of ontology FDs remains minimal.

Data cleaning allows us to take a relation and set of constraints and modify the relation to conform to the constraints. We focus on RHS repairs to overcome the loss of transitivity ontology FDs have compared to traditional FDs. Our approach seeks cost minimal changes to the relation to produce useful results. We show our cleaning algorithm scale quadratically and verify this experimentally.

Finally, we showed that ontology FDs are a useful data quality rule to capture domain relationships and can significantly reduce the number of false positive errors in data cleaning solutions that rely on traditional FDs.

8.1 Future Work

Naturally, there is more work to be done. We intend to consider extensions to other relationships such as *type-of* and the use of ontologies to discover other types of data quality rules such as conditional FDs and denial constraints. Our work will attempt to adapt the framework proposed by Szlichta, et al. [40] utilizing machine learning to consider data versus constraint versus ontology repairs for ontology FDs. We intend to study extensions to mutation algorithms used in our experiments to better simulate real world data integrity problems.

We intend to study the application of Ontology FDs to aid query optimization in a similar fashion as traditional FDs, for instance, to compute group-by statement on the fly similar to other work [36].

Bibliography

[1] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, A. I. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996, pp. 307–328.

[2] Akhtar, W., Cortés-Calabuig, A., and Paredaens, J. Constraints in RDF. In *Semantics in Data and Knowledge Bases - 4th International Workshops, SDKB 2010, Bordeaux, France, July 5, 2010, Revised Selected Papers* (2010), K. Schewe and B. Thalheim, Eds., vol. 6834 of *Lecture Notes in Computer Science*, Springer, pp. 23–39.

[3] Armstrong, W. W. Dependency structures of data base relationships. In *IFIP congress* (1974), vol. 74, Geneva, Switzerland, pp. 580–583.

[4] Astley, R. Construction of information retrieval systems. `http://bit.ly/IqT6zt`, 2009. Accessed: 2015-09-16.

[5] Baskaran, S., Keller, A., Chiang, F., and Szlichta, J. Efficient discovery of ontology functional dependencies. *CoRR abs/1611.02737* (2016).

[6] Bertossi, L. E., Bravo, L., Franconi, E., and Lopatenko, A. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Inf. Syst. 33*, 4-5 (2008), 407–434.

[7] Beskales, G., Ilyas, I. F., and Golab, L. Sampling the repairs of functional dependency violations under hard constraints. *PVLDB 3*, 1 (2010), 197–207.

[8] Bohannon, P., Flaster, M., Fan, W., and Rastogi, R. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005* (2005), F. Özcan, Ed., ACM, pp. 143–154.

[9] Chiang, F., and Miller, R. J. Discovering data quality rules. *PVLDB 1*, 1 (2008), 1166–1177.

[10] Chiang, F., and Miller, R. J. A unified model for data and constraint repair. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany* (2011), S. Abiteboul, K. Böhm, C. Koch, and K. Tan, Eds., IEEE Computer Society, pp. 446–457.

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

[12] Chu, X., Ilyas, I. F., and Papotti, P. Holistic data cleaning: Putting violations into context. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013* (2013), C. S. Jensen, C. M. Jermaine, and X. Zhou, Eds., IEEE Computer Society, pp. 458–469.

[13] Cong, G., Fan, W., Geerts, F., Jia, X., and Ma, S. Improving data quality: Consistency and accuracy. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007* (2007), C. Koch, J. Gehrke, M. N. Garofalakis, D. Srivastava, K. Aberer, A. Deshpande, D. Florescu, C. Y. Chan, V. Ganti, C. Kanne, W. Klas, and E. J. Neuhold, Eds., ACM, pp. 315–326.

[14] Eckerson, W. W. Data quality and the bottom line. *TDWI Report, The Data Warehouse Institute* (2002).

[15] Fan, W., Fan, Z., Tian, C., and Dong, X. L. Keys for graphs. *PVLDB 8*, 12 (2015), 1590–1601.

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

[17] Fan, W., Wu, Y., and Xu, J. Functional dependencies for graphs. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016* (2016), F. Özcan, G. Koutrika, and S. Madden, Eds., ACM, pp. 1843–1857.

[18] Golab, L., Karloff, H. J., Korn, F., Saha, A., and Srivastava, D. Sequential dependencies. *PVLDB 2*, 1 (2009), 574–585.

[19] Golab, L., Karloff, H. J., Korn, F., Srivastava, D., and Yu, B. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB 1*, 1 (2008), 376–390.

[20] Hassanzadeh, O., Yeganeh, S. H., and Miller, R. J. Linking semistructured data on the web. In Marian and Vassalos [30].

[21] Hellings, J., Gyssens, M., Paredaens, J., and Wu, Y. Implication and axiomatization of functional and constant constraints. *Ann. Math. Artif. Intell. 76*, 3-4 (2016), 251–279.

[22] Huhtala, Y., Kärkkäinen, J., Porkka, P., and Toivonen, H. Efficient discovery of functional and approximate dependencies using partitions. In *Proceedings of the Fourteenth International Conference on Data Engineering, Or-*

lando, Florida, USA, February 23-27, 1998 (1998), S. D. Urban and E. Bertino, Eds., IEEE Computer Society, pp. 392–401.

[23] Jensen, C. S., Snodgrass, R. T., and Soo, M. D. Extending existing dependency theory to temporal databases. *IEEE Trans. Knowl. Data Eng. 8*, 4 (1996), 563–582.

[24] Judah, S., and Friedman, T. Twelve ways to improve your data quality. Tech. rep., Gartner Research Report, 2014.

[25] Keller, A., and Szlichta, J. Ontology functional dependencies. In *Proceedings of the 10th Alberto Mendelzon International Workshop on Foundations of Data Management, Panama City, Panama, May 8-10, 2016* (2016), R. Pichler and A. S. da Silva, Eds., vol. 1644 of *CEUR Workshop Proceedings*, CEUR-WS.org.

[26] Kibbe, W. A., Arze, C., Felix, V., Mitraka, E., Bolton, E., Fu, G., Mungall, C. J., Binder, J. X., Malone, J., Vasant, D., Parkinson, H. E., and Schriml, L. M. Disease ontology 2015 update: an expanded and updated database of human diseases for linking biomedical knowledge through disease data. *Nucleic Acids Research 43*, Database-Issue (2015), 1071–1078.

[27] Koudas, N., Saha, A., Srivastava, D., and Venkatasubramanian, S. Metric functional dependencies. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China* (2009), Y. E. Ioannidis, D. L. Lee, and R. T. Ng, Eds., IEEE Computer Society, pp. 1275–1278.

[28] Langer, P., and Naumann, F. Efficient order dependency detection. *VLDB J. 25*, 2 (2016), 223–241.

[29] Lopes, S., Petit, J., and Lakhal, L. Efficient discovery of functional dependencies and armstrong relations. In *Advances in Database Technology - EDBT 2000, 7th International Conference on Extending Database Technology, Konstanz, Germany, March 27-31, 2000, Proceedings* (2000), C. Zaniolo, P. C. Lockemann, M. H. Scholl, and T. Grust, Eds., vol. 1777 of *Lecture Notes in Computer Science*, Springer, pp. 350–364.

[30] Marian, A., and Vassalos, V., Eds. *Proceedings of the 14th International Workshop on the Web and Databases 2011, WebDB 2011, Athens, Greece, June 12, 2011* (2011).

[31] Motik, B., Horrocks, I., and Sattler, U. Adding integrity constraints to OWL. In *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions, Innsbruck, Austria, June 6-7, 2007* (2007), C. Golbreich, A. Kalyanpur, and B. Parsia, Eds., vol. 258 of *CEUR Workshop Proceedings*, CEUR-WS.org.

[32] Novelli, N., and Cicchetti, R. FUN: an efficient algorithm for mining functional and embedded dependencies. In *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings.* (2001), J. V. den Bussche and V. Vianu, Eds., vol. 1973 of *Lecture Notes in Computer Science*, Springer, pp. 189–203.

[33] Papenbrock, T., Ehrlich, J., Marten, J., Neubert, T., Rudolph, J., Schönberg, M., Zwiener, J., and Naumann, F. Functional dependency discovery: An experimental evaluation of seven algorithms. *PVLDB 8*, 10 (2015), 1082–1093.

[34] Papenbrock, T., Kruse, S., Quiané-Ruiz, J., and Naumann, F. Divide & conquer-based inclusion dependency discovery. *PVLDB 8*, 7 (2015), 774–785.

[35] Prokoshyna, N., Szlichta, J., Chiang, F., Miller, R. J., and Srivastava, D. Combining quantitative and logical data cleaning. *PVLDB 9*, 4 (2015), 300–311.

[36] Simmen, D. E., Shekita, E. J., and Malkemus, T. Fundamental techniques for order optimization. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996.* (1996), H. V. Jagadish and I. S. Mumick, Eds., ACM Press, pp. 57–67.

[37] Song, S., and Chen, L. Discovering matching dependencies. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009* (2009), D. W. Cheung, I. Song, W. W. Chu, X. Hu, and J. J. Lin, Eds., ACM, pp. 1421–1424.

[38] Szlichta, J., Godfrey, P., Golab, L., Kargar, M., and Srivastava, D. Effective and complete discovery of order dependencies via set-based axiomatization. *PVLDB 10*, 7 (2017), 721–732.

[39] Szlichta, J., Godfrey, P., and Gryz, J. Fundamentals of order dependencies. *PVLDB 5*, 11 (2012), 1220–1231.

[40] Volkovs, M., Chiang, F., Szlichta, J., and Miller, R. J. Continuous data cleaning. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014* (2014), I. F. Cruz, E. Ferrari, Y. Tao, E. Bertino, and G. Trajcevski, Eds., IEEE Computer Society, pp. 244–255.

[41] Wang, X. S., Bettini, C., Brodsky, A., and Jajodia, S. Logical design for temporal databases with multiple granularities. *ACM Trans. Database Syst. 22*, 2 (1997), 115–170.

[42] Wijsen, J. Temporal fds on complex objects. *ACM Trans. Database Syst. 24*, 1 (1999), 127–176.

[43] Wijsen, J., Ng, R. T., and Calders, T. Discovering roll-up dependencies. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 15-18, 1999* (1999), U. M. Fayyad, S. Chaudhuri, and D. Madigan, Eds., ACM, pp. 213–222.

[44] Wyss, C. M., Giannella, C., and Robertson, E. L. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances - extended abstract. In *Data Warehousing and Knowledge Discovery, Third International Conference, DaWaK 2001, Munich, Germany, September 5-7, 2001, Proceedings* (2001), Y. Kambayashi, W. Winiwarter, and M. Arikawa, Eds., vol. 2114 of *Lecture Notes in Computer Science*, Springer, pp. 101–110.