Dependencies for Graphs

Dependencies for Graphs

Wenfei Fan
University of Edinburgh and Beihang University
wenfei@inf.ed.ac.uk

Ping Lu
BDBC, Beihang University
luping@buaa.edu.cn

ABSTRACT

This paper proposes a class of dependencies for graphs, referred to as *graph entity dependencies* (GEDs). A GED is a combination of a graph pattern and an attribute dependency. In a uniform format, GEDs express graph functional dependencies with constant literals to catch inconsistencies, and keys carrying id literals to identify entities in a graph.

We revise the chase for GEDs and prove its Church-Rosser property. We characterize GED satisfiability and implication, and establish the complexity of these problems and the validation problem for GEDs, in the presence and absence of constant literals and id literals. We also develop a sound and complete axiom system for finite implication of GEDs. In addition, we extend GEDs with built-in predicates or disjunctions, to strike a balance between the expressive power and complexity. We settle the complexity of the satisfiability, implication and validation problems for the extensions.

Keywords

graph dependencies; conditional functional dependencies; keys; EGDs; TGDs; satisfiability, implication, validation; axiom system; built-in predicates; disjunction

1. INTRODUCTION

As primitive integrity constraints for relations, functional dependencies (FDs) are found in almost every database textbook. FDs specify a fundamental part of the semantics of data, and have proven important in conceptual design, query optimization, and prevention of update anomalies, among other things. Moreover, FDs and their extensions such as conditional functional dependencies (CFDs) [21] and denial constraints [3] have been widely used in practice to detect semantic inconsistencies and repair data.

Among our most familiar FDs are keys. As a special case of FDs, keys provide an invariant connection between tuples and the real-world entities they represent, and are crucial to data models and transformations.

PODS '17, May 14–19, 2017, Chicago, IL, USA.

DOI: http://dx.doi.org/10.1145/3034786.3056114

The need for FDs and keys is also evident in graphs. Unlike relational data, real-life graphs often do not come with a schema, and dependencies such as FDs and keys provide one of few means for us to specify the integrity and semantics of the data. They are useful in consistency checking, spam detection, entity resolution and knowledge base expansion.

Example 1: Consider the following from knowledge bases and social networks, which are modeled as graphs.

(1) Consistency checking. It is common to find inconsistencies in real-life knowledge bases, *e.g.,*

- psychologist Tony Gibson is credited for creating Ghetto Blaster, while the video game was actually created by programmer Tony 'Gibbo' Gibson (Yago3);
- both Saint Petersburg and Helsinki are labeled as the capital of Finland (Yago3);
- it is claimed that all birds can fly, and that moa are birds, although moa are "flightless" (DBPedia);
- Philip Sclater is marked as both a child and a parent of William Lutley Sclater (DBPedia).

As shown in [23], such inconsistencies can be captured by FDs defined on graphs, referred to as GFDs.

(2) Spam detection. Fake accounts are common in social networks [14]. A rule for identifying spam is as follows.

- If account x' is confirmed fake, both accounts x and x' like blogs $P_1, \ldots, P_k$, x posts blog y, x' posts y', and if both y and y' have a peculiar keyword c, then x can also be identified fake.

Such rules can also be expressed as GFDs [23].

(3) Knowledge base expansion [19]. We want to decide whether to add a newly extracted album to a knowledge base G. To avoid duplicates, we need keys to identify an album entity in G, defined in terms of

ψ_1: its title and the id of its primary artist, or
ψ_2: its title and the year of initial release.

These can be expressed as keys for graphs studied in [19]. Note that the title of an album and the name of its artist cannot uniquely identify an album. For instance, an American band and a British band are both called "Bleach", and both bands had an album "Bleach".

To cope with ψ_1, we also need a key to identify artists:

ψ_3: the name of the artist, and the id of an album recorded by the artist.

As opposed to our familiar keys for relations, these keys are "recursively defined": to identify an album, we may need to identify its primary artist, and vice versa. □

Dependencies	Satisfiability	Implication	Validation	Connection with GEDs
GEDs	coNP-complete (Th. 3)	NP-complete (Th. 5)	coNP-complete (Th. 6)	$Q[\bar{x}](X \to Y)$
GFDs	coNP-complete (Th. 3)	NP-complete (Th. 5)	coNP-complete (Th. 6)	GEDs without id literals
GKeys	coNP-complete (Th. 3)	NP-complete (Th. 5)	coNP-complete (Th. 6)	$Q[\bar{x}](X \to x.\mathsf{id} = y.\mathsf{id})$
GED$_\mathsf{x}$s	coNP-complete (Th. 3)	NP-complete (Th. 5)	coNP-complete (Th. 6)	GEDs without constant literals
GFD$_\mathsf{x}$s	$O(1)$ (Th. 3)	NP-complete (Th. 5)	coNP-complete (Th. 6)	GFDs without constant literals
GDCs	Σ_2^p-complete (Th. 8)	Π_2^p-complete (Th. 8)	coNP-complete (Th. 8)	adding built-in predicates
GED$^\vee$s	Σ_2^p-complete (Th. 9)	Π_2^p-complete (Th. 9)	coNP-complete (Th. 9)	disjunctive Y in $Q[\bar{x}](X \to Y)$

Table 1: Complexity for reasoning about GEDs

Moreover, FDs and keys help us optimize queries that are costly on large graphs in the real world, *e.g.,* Facebook, which have billions of nodes and trillions of edges [27].

Keys and FDs on graphs are a departure from their relational counterparts. (1) A relational FD $R(X \to Y)$ is defined on a relation schema R with attributes X and Y, where R specifies the "scope" of the FD, *i.e.,* $X \to Y$ is to be applied to tuples in an instance of R. In contrast, graphs are semistructured and often schemaless. To cope with this, we need a combination of (a) a topological constraint to identify entities, *i.e.,* to specify its "scope", and (b) an "FD" on the attributes of the entities identified. (2) Relational FDs and keys are "value-based", while keys and FDs for graphs are often necessarily "id-based" as shown by ψ_1–ψ_3 of Example 1. That is, they are based on node identity. In particular, if two vertices are identified as the same entity, then they must have the same attributes and edges.

There has been work on FDs for RDF [2,13,16,28,30,32,42] in particular and for general property graphs [23] in general, and on keys for RDF [19]. However, many questions remain open. For example, as opposed to relational FDs and keys, none of these FD proposals can express keys for graphs [19].

The practical need calls for a full treatment of the topic, to answer the following questions. (1) Is there a simple class of graph dependencies for us to uniformly express FDs and keys? (2) Can we adapt the chase [39] to reason about the dependencies? (3) What is the complexity of fundamental problems associated with the dependencies? (4) Is there a finite axiom system for their implication analysis, like Armstrong's axioms for traditional FDs [5]? (5) How can we strike a balance between their expressivity and complexity?

Contributions. This paper tackles these questions.

(1) GEDs. We propose a class of dependencies, referred to as *graph entity dependencies* and denoted by GEDs (Section 3). A GED is a combination of (a) a graph pattern Q as a topological constraint, and (b) an "FD" $X \to Y$ with sets X and Y of equality literals. Pattern Q identifies a set of entities in a graph, and the FD is enforced on these entities. GEDs may specify conditions carrying literals with constants, like relational CFDs [21]. They may carry id literals to identify vertices in a graph, beyond equality on attribute values.

GEDs subsume GFDs of [23] and keys of [19] as special cases (subject to adaption of graph pattern matching with graph homomorphism instead of subgraph isomorphism, to uniformly express GFDs and keys; see Section 3). They can express traditional FDs, CFDs and equality-generating dependencies (EGDs [7]), when relations are represented as graphs. That is, GEDs can do the job of keys, FDs, CFDs and EGDs for graph-structured data, *e.g.,* to specify integrity, detect inconsistencies, identify entities and optimize queries.

(2) The chase revised. We extend the chase [39] to GEDs (Section 4). Chasing with GEDs is more involved than with traditional FDs: it may run into conflicts introduced by id literals or constant literals, and may "generate" new attributes when enforcing GEDs on a schemaless graph. Nonetheless, we show that the chase with GEDs is finite and has the Church-Rosser property. That is, all chasing sequences of a graph (pattern) by a set of GEDs are finite and yield the same result, regardless of the order of GEDs applied.

(3) Classical problems for GEDs. We investigate three fundamental problems associated with GEDs (Section 5).

(a) The *satisfiability* problem is to decide, given a set Σ of GEDs, whether there exists a nonempty finite model G of Σ that satisfies Σ, denoted by $G \models \Sigma$ as usual.

(b) The *implication* problem is to decide whether a set Σ of GEDs entails another GED φ, denoted by $\Sigma \models \varphi$, *i.e.,* for any finite graph G, if $G \models \Sigma$, then $G \models \varphi$.

(c) The *validation* problem is to decide, given a finite graph G and a set Σ of GEDs, whether $G \models \Sigma$.

These problems not only are of theoretical interest, but also find practical applications. The satisfiability analysis helps us check whether a set of GEDs makes sense before the GEDs are used as rules for data cleaning or query optimization. The implication analysis serves as an optimization strategy to get rid of redundant rules. The validation analysis can detect violations of GEDs, and catch "dirty" entities.

To understand where the complexity arises, we consider two dichotomies when studying these problems:

- the presence of id literals vs. their absence, and
- the presence of constant values vs. their absence.

For instance, keys of [19] are recursively defined in terms of id literals while GFDs of [23] are not. In these settings, we characterize GED satisfiability and implication, based on the chase. We also establish complexity bounds of these problems for GEDs, GFDs, keys and other sub-classes, all matching (see Table 1). As opposed to relational FDs, these problems are all intractable for GEDs, and the intractability is robust even for restricted special cases. The complexity is, however, comparable to, *e.g.,* (a) relational CFDs, for which the satisfiability and implication problems are NP-complete and coNP-complete, respectively [21], and (b) EGDs, for which the implication problem is NP-complete [8].

(4) Finite axiomatizability. We study the finite axiomatizability of GEDs (Section 6). One naturally wants a finite set $\mathcal{A}$ of inference rules that is sound and complete for the implication analysis of GEDs, along the same lines as Armstrong's axioms for relational FDs (see [1]). That is, for any set Σ of GEDs and another GED φ, $\Sigma \models \varphi$ if and only if φ is provable

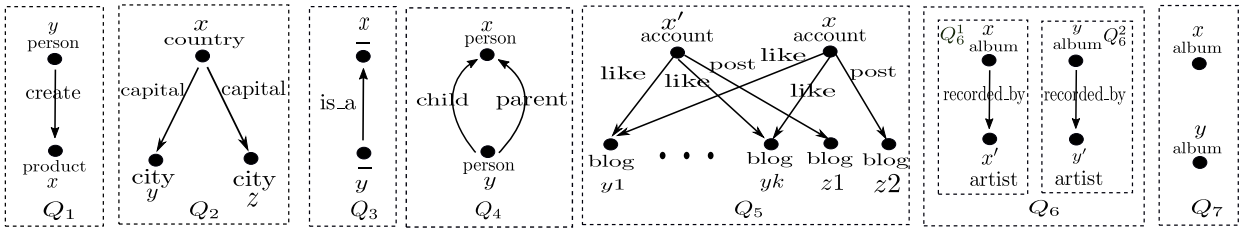

Figure 1: Graph patterns

from Σ using $\mathcal{A}$. Here we focus on finite graphs and study finite implication, rather than unrestricted implication.

We provide a set of six inference rules for GEDs, and show that it is sound and complete for GED implication analyses, based on the revised chase. We also show that the axiom system is independent (non-redundant and minimal), *i.e.,* removing any rule makes it no longer complete.

(5) Extensions. To strike a balance between the expressivity and complexity, we investigate extensions of GEDs (Section 7). We extend GEDs by supporting

- built-in predicates $=, \neq, <, >, \leq, \geq$ (GDCs); or
- limited disjunction of literals (GED$^\vee$s).

We can express, for instance, denial constraints [3] as GDCs, disjunctive EGDs [17] as GED$^\vee$s, and "domain constraints" for attributes of an entity to have a finite domain as both GDCs and GED$^\vee$s, among other things.

With the increased expressive power, the extensions increase the complexity of static analyses. We show that their satisfiability and implication problems become Σ_2^p-complete and Π_2^p-complete, as opposed to coNP-complete and NP-complete for GEDs, respectively. Their validation problems remain coNP-complete, the same as for GEDs (Table 1).

The dependency classes studied in the paper and their complexity results are summarized in Table 1, annotated with their corresponding theorems. This work is a preliminary step toward developing a dependency theory for graphs. The intractability results reveal the challenges inherent to entities with a graph structure. The revised chase, characterizations of satisfiability and implication, and axiom system provide insight into the analyses of graph dependencies.

The related work will be discussed in Section 8.

2. PRELIMINARIES

Before we define GEDs, we first review some basic notations. Assume three countably infinite sets Γ, Υ and U of labels, attributes and constants, respectively.

Graphs. A *graph* G is specified as (V, E, L, F_A), where (a) V is a finite set of nodes; (b) $E \subseteq V \times \Gamma \times V$ is a finite set of edges, in which (v, ι, v') denotes an edge from node v to v', and the edge is labeled with ι, referred to as its *label*; (c) each node $v \in V$ has label $L(v)$ from Γ; and (d) each node $v \in V$ carries a tuple $F_A(v) = (A_1 = a_1, \ldots, A_n = a_n)$ of *attributes* of a finite arity, where $A_i \in \Upsilon$ and $a_i \in U$, written as $v.A_i = a_i$, and $A_i \neq A_j$ if $i \neq j$. In particular, each v has a special attribute id denoting its *node identity*.

That is, we consider finite directed graphs in which nodes and edges are labeled. Nodes carry attributes for, *e.g.,* properties and keywords and rating, as in property graphs. Unlike relational databases, we assume no schema for graphs. Hence for an attribute $A \in \Upsilon$ and a node $v \in V$, $v.A$ may not exist, except that v has a unique $v.\mathsf{id}$ as found in practice.

Graph patterns. A *graph pattern* is a directed graph $Q[\bar{x}] = (V_Q, E_Q, L_Q)$, where (1) V_Q (resp. E_Q) is a finite set of pattern nodes (resp. edges) as before; (2) L_Q is a function that assigns a label $L_Q(u)$ to each node $u \in V_Q$; and (3) $\bar{x}$ denotes the nodes in V_Q as a list of variables.

Labels of pattern nodes and edges are drawn from Γ. Moreover, we allow wildcard '_' as a special label in Q.

A pattern $Q_2[\bar{y}]$ is a *copy* of $Q_1[\bar{x}]$ via a bijection f: $\bar{x} \mapsto \bar{y}$ if $Q_2[\bar{y}]$ is $Q_1[\bar{x}]$ with variables renamed by f. More specifically, let $Q_1[\bar{x}] = (V_{Q1}, E_{Q1}, L_{Q1})$ and $Q_2[\bar{y}] = (V_{Q2}, E_{Q2}, L_{Q2})$. Then (a) $\bar{x}$ and $\bar{y}$ are disjoint, and (b) f is an isomorphism from Q_1 to Q_2; *i.e.,* for each $x \in \bar{x}$, $L_{Q1}(x) = L_{Q2}(f(x))$; and (x_1, ι, x_2) is an edge in E_{Q_1} if and only if $(f(x_1), \iota, f(x_2))$ is in E_{Q_2}, with the same label ι.

Example 2: Figure 1 depicts seven graph patterns: (1) $Q_1[x, y]$ specifies a person entity and a product entity, which are connected by a create edge; similarly for $Q_2[x, y, z]$ and $Q_4[x, y]$; (2) $Q_3[x, y]$ depicts a generic is_a relationship between two entities labeled with wildcard '_'; (3) $Q_5[x, x', z_1, z_2, y_1, \ldots, y_k]$ specifies two accounts x and x', $k + 2$ blogs $z_1, z_2, y_1, \ldots, y_k$ and their relationships; (4) $Q_6[x, x', y, y']$ consists of (a) a pattern $Q_6^1[x, x']$ with variables x and x', specifying a relationship between an album entity x and an artist entity x'; and (b) a copy $Q_6^2[y, y']$ of $Q_6^1[x, x']$ with variables renamed; similarly for $Q_7[x, y]$. □

Matches. We say that a label ι *matches* ι', denoted by $\iota \asymp \iota'$, if either (a) ι and ι' are in Γ and $\iota = \iota'$, or (b) $\iota' \in \Gamma$ and ι is '_', *i.e.,* wildcard matches any label in Γ.

A *match* of pattern $Q[\bar{x}]$ in graph G is a homomorphism h from Q to G such that (a) for each node $u \in V_Q$, $L_Q(u) \asymp L(h(u))$; and (b) for each edge $e = (u, \iota, u')$ in Q, there exists an edge $e' = (h(u), \iota', h(u'))$ in G such that $\iota \asymp \iota'$. Note that when ι is '_', there may exist multiple edges e' with $\iota \asymp \iota'$. The match picks one of them, and denotes it by $h(\iota_v^u)$.

Abusing the notations, we also denote the match as a vector $h(\bar{x})$ if it is clear from the context, where $h(\bar{x})$ consists of $h(x)$ for all variables $x \in \bar{x}$. Intuitively, $h(\bar{x})$ is a list of entities identified by pattern Q in graph G.

3. GRAPH ENTITY DEPENDENCIES

We now define *graph entity dependencies* (GEDs).

GEDs. A GED φ is defined as $Q[\bar{x}](X \to Y)$, where $Q[\bar{x}]$ is a graph pattern, and X and Y are two (possibly empty) sets of literals of $\bar{x}$; we refer to $Q[\bar{x}]$ and $X \to Y$ as the *pattern* and FD of φ, respectively.

A *literal* of $\bar{x}$ is one of the following: for $x, y \in \bar{x}$,

(a) *constant literal* $x.A = c$, where c is a constant in U, and A is an attribute in Υ that is not id;

(b) *variable literal* $x.A = y.B$, where A and B are attributes in Υ that are not id; or

(c) id *literal* $x.\mathsf{id} = y.\mathsf{id}$.

Intuitively, GED φ is a combination of (1) a *topological constraint* imposed by pattern Q, to identify entities in a graph, and (2) an FD $X \to Y$, to be applied to the entities identified by Q. Constant literals $x.A = c$ enforce bindings of semantically related constants, along the same lines as relational CFDs [21]. An id literal $x.\mathsf{id} = y.\mathsf{id}$ states that x and y denote the same vertex (entity).

Example 3: We can use GEDs to detect the inconsistencies, catch spam and identify entities observed in Example 1. These GEDs are defined with graph patterns of Fig. 1.

(1) GED $\varphi_1 = Q_1[x, y](X_1 \to Y_1)$. Here X_1 consists of a single constant literal $x.\mathsf{type} =$ "video game", Y_1 consists of a literal $y.\mathsf{type} =$ "programmer", and type is an attribute of person and product (not shown in Q_1). It states that a video game can only be created by programmers.

(2) GED $\varphi_2 = Q_2[x, y, z](\emptyset \to y.\mathsf{name} = z.\mathsf{name})$. It says that if a country x has two capitals y and z, then y and z must have the same name. Here name is an attribute, X is empty, and Y consists of a single variable literal.

(3) GED $\varphi_3 = Q_3[x, y](x.A = x.A \to y.A = x.A)$, where A is an attribute of x, *e.g.,* can_fly. It says that if y is_a x and if x has property A, then y inherits $x.A$, *i.e.,* y also has attribute A and $y.A = x.A$. Note that x and y are labeled '_', representing generic entities regardless of labels.

(4) GED $\varphi_4 = Q_4[x, y](\emptyset \to \mathsf{false})$, where false is a syntactic sugar for Boolean constant (see details shortly). It states that pattern Q_4 is "illegal", *i.e.,* no person can be both a child and a parent of another person.

GEDs φ_1–φ_4 catch the errors described in Example 1, *e.g.,* φ_3 can detect the inconsistency between birds and moa.

(5) GED $\varphi_5 = Q_5[x, x', z_1, z_2, y_1, \ldots, y_k](X_5 \to Y_5)$ specifies the rule of Example 1 for catching spam. Here X_5 consists of three constant literals $x'.\mathsf{is\_fake} = 1$, $z_1.\mathsf{keyword} = c$ and $z_2.\mathsf{keyword} = c$, Y_5 is $x.\mathsf{is\_fake} = 1$, and c is a constant. The GED says that for accounts and blogs that match Q_5, if account x' is confirmed fake and if blogs z_1 and z_2 both contain a peculiar keyword c, then x is also a fake account.

(6) The keys of Example 1 can be expressed as GEDs:

For album: $\psi_1 = Q_6[x, x', y, y'](X_6 \to x.\mathsf{id} = y.\mathsf{id})$,
$\psi_2 = Q_7[x, y](X_7 \to x.\mathsf{id} = y.\mathsf{id})$.
For artist: $\psi_3 = Q_6[x, x', y, y'](X_8 \to x'.\mathsf{id} = y'.\mathsf{id})$.

Here X_6 consists of $x.\mathsf{title} = y.\mathsf{title}$ and id literal $x'.\mathsf{id} = y'.\mathsf{id}$; X_7 includes $x.\mathsf{title} = y.\mathsf{title}$ and $x.\mathsf{release} = y.\mathsf{release}$; and X_8 consists of $x'.\mathsf{name} = y'.\mathsf{name}$ and id literal $x.\mathsf{id} = y.\mathsf{id}$, defined with attributes title, release, name and id.

To identify a pair of album entities x and y, we check either their title attributes and the ids of their artists (ψ_1), or their title and release attributes (ψ_2). Similarly, to identify artist entities x' and y' as required by ψ_1, we need to check the ids of a pair of albums they recorded (ψ_3) in turn. □

Semantics. To interpret GED $\varphi = Q[\bar{x}](X \to Y)$, we use the following notations. Consider a match $h(\bar{x})$ of Q in a graph G, and a literal l of $\bar{x}$. We say that $h(\bar{x})$ *satisfies* l, denoted by $h(\bar{x}) \models l$, if (a) when l is $x.A = c$, then attribute $v.A$ *exists* at node $v = h(x)$, and $v.A = c$; (b) when l is $x.A = y.B$, then attributes A and B *exist* at $v = h(x)$ and $v' = h(y)$, respectively, and $v.A = v'.B$; and (c) when l is $x.\mathsf{id} = y.\mathsf{id}$, then $h(x)$ and $h(y)$ refer to the same node; hence, they have the same set of attributes and edges.

We denote by $h(\bar{x}) \models X$ if the match $h(\bar{x})$ satisfies *all* the literals in X; in particular, if X is $\emptyset$, then $h(\bar{x}) \models X$ for any match $h(\bar{x})$ of Q in G; similarly for $h(\bar{x}) \models Y$. We write $h(\bar{x}) \models X \to Y$ if $h(\bar{x}) \models X$ implies $h(\bar{x}) \models Y$.

A graph G *satisfies* GED φ, denoted by $G \models \varphi$, if *for all* matches $h(\bar{x})$ of Q in G, $h(\bar{x}) \models X \to Y$.

We say that a graph G *satisfies* a set Σ of GEDs if for all $\varphi \in \Sigma$, $G \models \varphi$, *i.e.,* G satisfies every GED in Σ.

Given the semantics, we also write $Q[\bar{x}](X \to Y)$ as

$$Q[\bar{x}](\textstyle\bigwedge_{l \in X} l \to \bigwedge_{l' \in Y} l').$$

Existence of attributes. Note that attributes are not specified in pattern Q, and that we consider schemaless graphs. For a literal $x.A = c$, node $h(x)$ does not necessarily have attribute A, to accommodate the semistructured nature of graphs. Observe the following. (a) When $x.A = c$ is a literal in X, if $h(x)$ has *no* A-attribute, then $h(\bar{x})$ trivially satisfies $X \to Y$ by the definition of $h(\bar{x}) \models X$. (b) In contrast, if $x.A = c$ is in Y, then for $h(\bar{x}) \models Y$, node $h(x)$ must have A-attribute by the definition; similarly for $x.A = y.B$.

As a consequence, we can use, *e.g.,* $Q[x](\emptyset \to x.A = x.A)$ to enforce that all entities x of "type" τ must have an A attribute, where Q consists of a single vertex x labeled τ. This is in the flavor of tuple generating dependencies [7], limited to attributes. Such constraints cannot be expressed as EGDs [7] or FDs for relations and RDF [2, 16, 30, 42].

However, GEDs *cannot* enforce attribute $x.A$ to have a finite domain, *e.g.,* Boolean, as opposed to database schema.

Special cases. We list some special cases of GEDs.

(1) GFDs. GFDs of [23] are syntactically defined as GEDs without id literals, *i.e.,* $Q[\bar{x}](X \to Y)$ in which neither X nor Y contains $x.\mathsf{id} = y.\mathsf{id}$. They adopt the semantics of subgraph isomorphism for graph pattern matching.

We refer to GEDs of this form also as GFDs, and interpret graph pattern matching in terms of homomorphism. For instance, φ_1–φ_5 in Example 3 are GFDs, but ψ_1–ψ_3 are not.

(2) Keys. A key ψ of [19] is defined as $Q[\bar{x}, x_o]$, where $Q[\bar{x}]$ is a graph pattern and x_o is a designated node in $\bar{x}$. A graph G satisfies ψ if for any two matches $h(\bar{x})$ and $h'(\bar{x})$ of $Q[\bar{x}]$ in G such that $h(\bar{x})$ and $h'(\bar{x})$ are isomorphic, $h(x_o)$ and $h'(x_o)$ denote the same node. Pattern Q is defined as a set of RDF triples, carrying variables and constants, and interpreted under the semantics of subgraph isomorphism.

We define a *key for graphs*, denoted by GKey, as a GED of the form $Q[\bar{z}](X \to x_0.\mathsf{id} = y_0.\mathsf{id})$, where (a) $Q[\bar{z}]$ is composed of patterns $Q_1[\bar{x}]$ and $Q_2[\bar{y}]$, and $Q_2[\bar{y}]$ is a copy of $Q_1[\bar{x}]$ via a bijection f: $\bar{x} \mapsto \bar{y}$ (see Example 2), (b) $\bar{z}$ consists of $\bar{x}$ followed by $\bar{y}$, (c) $x_0 \in \bar{x}$ and $y_0 = f(x_0)$ are designated nodes in Q, and (d) X is a set of literals as before.

For instance, ψ_1, ψ_2 and ψ_3 of Example 3 are GKeys. GKeys express recursive keys of [19] in terms of id literals.

The key $\psi = Q[\bar{x}, x_o]$ of [19] can be expressed as a GKey $Q'[\bar{z}](X \to x_0.\mathsf{id} = y_0.\mathsf{id})$, where X consists of literals to express constant and variable bindings embedded in pattern Q, and Q' is composed of Q and a copy of Q, interpreted in terms of homomorphism instead of subgraph isomorphism.

It is to uniformly express keys and GFDs that we adopt the homomorphism semantics for graph pattern matching.

To illustrate this, consider GKey ψ_3 given in Example 3. The GKey catches no violations if it is interpreted under subgraph isomorphism. Indeed, for any match $h[\bar{x}]$ of pattern Q_6 in a graph G, $h(x)$ and $h(y)$ have to be distinct nodes as required by isomorphism. As a result, $h[\bar{x}] \not\models X_8$ and hence, $h[\bar{x}]$ trivially satisfies ψ_3. As opposed to [19] that interprets a key with three isomorphic mappings, we interpret GEDs with a single match of pattern, and thus isomorphism is too strict to allow two variables to be mapped to the same node.

The issue becomes more subtle when it comes to the satistiability of a set Σ of GEDs (see Section 5.1), where a model of Σ requires that every GED in Σ finds a match of its pattern, to assure that the GEDs in Σ do not conflict with each other. Consider a GKey $\varphi = Q[x, y](\emptyset \rightarrow x.\mathsf{id} = y.\mathsf{id})$, where Q consists of two isolated nodes, which are labeled with "UoE". This GKey states that all nodes representing "UoE" are essentially the same node. One can verify that under the semantics of subgraph isomorphism, GKeys like φ cannot find a model in any sensible graph.

(3) GED$_\mathsf{x}$s. We also study the class of GFDs that include no constant literals, referred to as *variable* GFDs and denoted by GFD$_\mathsf{x}$s. For instance, φ_2 and φ_3 are GFD$_\mathsf{x}$s, but φ_1, φ_4 and φ_5 are not. Intuitively, (a) GFDs are an extension of relational CFDs to graphs, (b) while GFD$_\mathsf{x}$s extend FDs, carrying neither constant literals nor id literals.

Similarly, we study GEDs without constant literals, referred to as *variable* GEDs and denoted by GED$_\mathsf{x}$s. Obviously GFD$_\mathsf{x}$s are a proper sub-class of GEDs; *e.g.,* ψ_1–ψ_3 of Example 3 are GED$_\mathsf{x}$s, but they are not GFD$_\mathsf{x}$s.

(4) Forbidding GEDs. GEDs can express limited negation, in the form of $Q[\bar{x}](X \rightarrow \mathsf{false})$, where false is an abbreviation for, *e.g., Y* consisting of $y.A = c$ and $y.A = d$ for distinct constants c and d, where y is a variable in $\bar{x}$. Following [16], we refer to such GEDs as *forbidding constraints*.

(5) Relational dependencies. Following [23], one can show that FDs and CFDs can be expressed as GEDs if relation tuples are represented as nodes in a graph. In fact, equality-generating dependencies (EGDs) can be expressed as GFDs (GEDs) in the same setting. An EGD has the form $\forall \bar{z}(\phi(\bar{z}) \rightarrow y_1 = y_2)$, where ϕ is a conjunction of relation atoms $R(w_1, \ldots, w_l)$ and equality atoms $w_i = w_j$, w_i and w_j are variables in $\bar{z}$, and so are y_1 and y_2 (cf. [1]). Here each variable w corresponds to an attribute $R_w[A_w]$ in a relation atom of ϕ. The EGD can be expressed as a pair of GFDs:

(1) $\varphi_R = Q_E[\bar{x}](\emptyset \rightarrow Y_R)$, where (a) Q_E is a pattern such that for each relation atom R in ϕ, there exists a node $x_R \in \bar{x}$ in Q_E that is labeled with R; and Q_E has no edges; and (b) Y_R consists of $x_R.A_R = x_R.A_R$ for each variable $x \in \bar{z}$, which indicates attribute $R[A_R]$; intuitively, φ_R ensures that the relations in ϕ have the attributes required; and

(2) $\varphi_E = Q_E[\bar{x}](X_E \rightarrow Y_E)$, where (a) for each equality atom $w_i = w_j$ in ϕ, which corresponds to $R_i[A_{R_i}] = R_j[A_{R_j}]$ as remarked above, X_E includes a literal $x_{R_i}.A_{R_i} = x_{R_j}.A_{R_j}$; and (b) Y_E is $x_{R_{y1}}.A_{R_{y1}} = x_{R_{y2}}.A_{R_{y2}}$, which corresponds to $y_1 = y_2$. This enforces that ϕ entails $y_1 = y_2$.

One might be tempted to encode GEDs as relational dependencies. As will be discussed in Section 8, such encoding makes it awkward to express id literals, and the relational techniques do not simplify the analyses of GEDs.

4. THE CHASE REVISITED FOR GEDS

We next revise the chase [39] for GEDs over graphs (Section 4.1), and show that chasing with GEDs has the Church-Rosser property (Section 4.2). As will be seen in later sections, the chase helps us characterize the static analyses of GEDs and develop finite axiomatization for GEDs.

4.1 Chasing with GEDs

Consider a graph $G = (V, E, L, F_A)$ and a finite set Σ of GEDs. We study the chase of G by Σ, to (a) check the satisfiability of Σ (resp. implication of GED φ by Σ) when G encodes the patterns of Σ (resp. φ; see Section 5), (b) optimize graph pattern queries Q with Σ when G represents Q, and (c) identify entities and catch errors by using Σ in a knowledge base or a social graph G, among other things.

Equivalence relations. We define the chase as a sequence of equivalence relations Eq on nodes x and attributes $x.A$ in G. For each node x in V, its equivalence class, denoted by $[x]_{\mathsf{Eq}}$, is a set of nodes $y \in V$ that are identified as x. For each attribute $x.A$ of x, its equivalence class $[x.A]_{\mathsf{Eq}}$ is a set of attributes $y.B$ and constants c, if $x.A = y.B$ and $x.A = c$ are enforced by GEDs in Σ (see below), respectively. The relation is reflexive, symmetric and transitive, such that

(a) if node $y \in [x]_{\mathsf{Eq}}$, then $x \in [y]_{\mathsf{Eq}}$ and $[x]_{\mathsf{Eq}} = [y]_{\mathsf{Eq}}$; that is, we merge $[x]_{\mathsf{Eq}}$ and $[y]_{\mathsf{Eq}}$ into one; similarly, if attribute $y.B \in [x.A]_{\mathsf{Eq}}$, then $[y.B]_{\mathsf{Eq}} = [x.A]_{\mathsf{Eq}}$;

(b) if there is attribute $y.B$ such that $y.B \in [x.A]_{\mathsf{Eq}}$ and $y.B \in [z.C]_{\mathsf{Eq}}$, then $[x.A]_{\mathsf{Eq}} = [z.C]_{\mathsf{Eq}}$; similarly for constant c if $c \in [x.A]_{\mathsf{Eq}}$ and $c \in [z.C]_{\mathsf{Eq}}$;

(c) if there exists node y such that $y \in [x]_{\mathsf{Eq}}$ and $y \in [z]_{\mathsf{Eq}}$, then $[x]_{\mathsf{Eq}} = [z]_{\mathsf{Eq}}$ by transitivity; and

(d) if node $y \in [x]_{\mathsf{Eq}}$, then for each attribute $y.B$ of y, $[x.B]_{\mathsf{Eq}} = [y.B]_{\mathsf{Eq}}$; similarly for attribute $x.A$; that is, if x and y are the same node, then they have the same attributes and corresponding values.

Consistency. Inconsistencies may be introduced by id literals and constant literals when enforcing GEDs.

We say that Eq is *inconsistent* in G if

(a) there exists node $y \in [x]_{\mathsf{Eq}}$ such that $L(x) \not\asymp L(y)$ and $L(y) \not\asymp L(x)$ (label conflict), or

(b) there exists $y.B \in [x.A]_{\mathsf{Eq}}$ such that $x.A = c$ and $y.B = d$ for distinct c, d in U (attribute conflict).

Otherwise we say that Eq is *consistent*.

We use $\asymp$ to compare labels (recall $\asymp$ from Section 2). This is to cope with wildcard in a pattern Q when we chase Q as a graph (see Section 5 for such examples). In this case, we treat '_' in Q as a special label. Recall that $\asymp$ is asymmetric: $x \asymp y$ does not mean that $y \asymp x$.

Coercion. When an equivalence relation Eq is consistent in graph G, we can enforce Eq on G and revise G by merging nodes and their corresponding attributes and edges, and by equalizing and extending attributes, as follows.

We define the *coercion of a consistent* Eq *on* G as graph $G_{\mathsf{Eq}} = (V', E', L', F'_A)$ obtained from G as follows: for each node $x \in V$, (a) x_{Eq} is a node in V', denoting $[x]_{\mathsf{Eq}}$; (b) for each edge $(x, \iota, y) \in E$, $(x_{\mathsf{Eq}}, \iota, y_{\mathsf{Eq}})$ is an edge in E'; similarly for each edge $(y, \iota, x) \in E$; (c) $L'(x_{\mathsf{Eq}})$ is '_' if all nodes in $[x]_{\mathsf{Eq}}$ are labeled '_'; otherwise $L'(x_{\mathsf{Eq}}) = L(z)$, where $z \in [x]_{\mathsf{Eq}}$ with $L(z) \neq$ '_'; and (d) $F'_A(x_{\mathsf{Eq}}) = \bigcup_{y \in [x]_{\mathsf{Eq}}} F_A(y)$, the union of the attributes of all the nodes in $[x]_{\mathsf{Eq}}$.

When Eq is consistent, G_{Eq} is well defined. In particular, when x and y are identified as the same node, $F'_A([x]_{\mathsf{Eq}})$ merges the attributes of x and y; moreover, if A is an attribute of both x and y, then $x.A = y.A$, and hence $F'_A(\cdot)$ is well defined. In addition, for all nodes $z_1, z_2 \in [x]_{\mathsf{Eq}}$, if $L(z_1) \neq$ '_' and $L(z_2) \neq$ '_', then $L(z_1) = L(z_2)$.

When Eq is inconsistent, G_{Eq} is undefined.

Chasing. We start with Eq_0, an initial equivalence relation that includes $[x]_{\mathsf{Eq}_0} = \{x\}$ and $[x.A]_{\mathsf{Eq}_0} = \{x.A, c\}$, for each node $x \in V$ and attribute $x.A = c$ in $F_A(x)$. Each chase step i extends Eq_{i-1} to get Eq_i, by applying a GED.

We define *a chase step of G by Σ at* Eq as

$$\mathsf{Eq} \Rightarrow_{(\varphi,h)} \mathsf{Eq}',$$

where $\varphi = Q[\bar{x}](X \to Y)$ is a GED in Σ, and $h(\bar{x})$ is a match of pattern Q in the coercion G_{Eq} of Eq on graph G such that (a) $h(\bar{x}) \models X$, and (b) Eq' is the equivalence relation of the extension of Eq by adding one literal $l \in Y$; more specifically, l and Eq' satisfies one of the following conditions:

(1) if l is $x.A = c$ and $c \notin [h(x).A]_{\mathsf{Eq}}$, then Eq' extends Eq by (a) including a new equivalence class $[h(x).A]_{\mathsf{Eq}'}$ if $h(x).A$ is not in Eq, and (b) adding c to $[h(x).A]_{\mathsf{Eq}'}$;

(2) if l is $x.A = y.B$ and $h(y).B \notin [h(x).A]_{\mathsf{Eq}}$, then Eq' extends Eq by adding (a) $[h(x).A]_{\mathsf{Eq}'}$ if $h(x).A$ is not in Eq, and (b) $h(y).B$ to $[h(x).A]_{\mathsf{Eq}'}$; and

(3) if l is $x.\mathsf{id} = y.\mathsf{id}$ and $h(y) \notin [h(x)]_{\mathsf{Eq}}$, then Eq' extends Eq by adding $h(y)$ to $[h(x)]_{\mathsf{Eq}'}$.

The step is *valid* if Eq' is consistent in G_{Eq}.

Note that cases (1) and (2) above may expand the set of attributes of $h(x)$ when enforcing φ: attribute $h(x).A$ in Y is added if it is not already an attribute of $h(x)$, as required by $h(\bar{x}) \models Y$ (Section 3), since otherwise the chase will not lead to a graph that satisfies φ (see Theorem 1 below).

A *chasing sequence* ρ of G by Σ is a sequence

$$\mathsf{Eq}_0, \ \ldots, \ \mathsf{Eq}_k,$$

where for all $i \in [0, k-1]$, there exist a GED $\varphi = Q[\bar{x}](X \to Y)$ in Σ and a match h of pattern Q in coercion graph G_{Eq_i} such that $\mathsf{Eq}_i \Rightarrow_{(\varphi,h)} \mathsf{Eq}_{i+1}$ is a valid chase step.

The sequence is *terminal* if there exist no GED $\varphi \in \Sigma$, match h of pattern Q of φ in G_{Eq_k} and equivalence relation Eq_{k+1} such that $\mathsf{Eq}_k \Rightarrow_{(\varphi,h)} \mathsf{Eq}_{k+1}$ is a valid chase step. More specifically, it terminates in one of the following cases.

(a) No GEDs in Σ can be applied to expand the chasing sequence. If so, we say that the sequence is *valid*, and refer to $(\mathsf{Eq}_k, G_{\mathsf{Eq}_k})$ as *its result*. It is easy to verify that in a valid ρ, for all $i \in [0, k]$, Eq_i is consistent in G_{Eq_i}.

(b) Either Eq_0 is inconsistent to start with (see the case in Section 5.2), or there exist φ, h and Eq_{k+1} such that $\mathsf{Eq}_k \Rightarrow_{(\varphi,h)} \mathsf{Eq}_{k+1}$ but Eq_{k+1} is inconsistent in G_{Eq_k}. If so, we say that ρ is *invalid*, with result $\perp$ (undefined).

In particular, a forbidding constraint $Q[\bar{x}](X \to \mathsf{false})$ can be applied only when G is "inconsistent" or "dirty", and as a result, it makes the chasing sequence invalid.

Example 4: Consider graph G shown in Fig. 2, where v_1 and v_2 have attribute A with $v_1.A = 1$ and $v_2.A = 1$.

(1) Consider a set Σ_1 consisting of a single GED $\phi_1 = Q_1[x, y](x.A = y.A \to x.\mathsf{id} = y.\mathsf{id})$ with Q_1 in Fig. 2. Then $\mathsf{Eq}_0 \Rightarrow_{(\phi_1,h_1)} \mathsf{Eq}_1$ is a chase step, where (a) Eq_0 consists of $[v]_{\mathsf{Eq}_0} = \{v\}$ for v ranging over v_1, v_2, v'_1, v'_2, and $[v_1.A]_{\mathsf{Eq}_0} = [v_2.A]_{\mathsf{Eq}_0} = \{v_1.A, v_2.A, 1\}$; (b) h_1: $x \mapsto v_1$ and $y \mapsto v_2$; and (c) Eq_1 extends Eq_0 by letting $[v_1]_{\mathsf{Eq}_1} = [v_2]_{\mathsf{Eq}_1} = \{v_1, v_2\}$. The coercion G_1 of Eq_1 on G is shown in Fig. 2, which merges v_1 and v_2. One can verify that $\mathsf{Eq}_0 \Rightarrow_{(\phi_1,h_1)} \mathsf{Eq}_1$ is a terminal chasing sequence of G by Σ_1 since no more GEDs can be applied. Moreover, it is valid, yielding result (Eq_1, G_1).

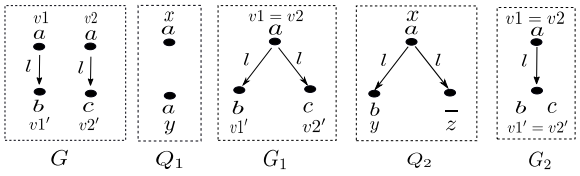

Figure 2: Graphs and patterns used in chasing

(2) Consider $\Sigma_2 = \{\phi_1, \phi_2\}$, where $\phi_2 = Q_2[x, y, z](\emptyset \to y.\mathsf{id} = z.\mathsf{id})$ (Q_2 in Fig. 2). Now $\mathsf{Eq}_1 \Rightarrow_{(\phi_2,h_2)} \mathsf{Eq}_2$ is a chase step, where h_2: $x \mapsto v_1$, $y \mapsto v'_1$, $z \mapsto v'_2$; and (b) Eq_2 extends Eq_1 by adding v'_2 to $[v'_1]_{\mathsf{Eq}_1}$. Then $\mathsf{Eq}_0 \Rightarrow_{(\phi_1,h_1)} \mathsf{Eq}_1$ is still terminal, but it is invalid as there exists a chase step $\mathsf{Eq}_1 \Rightarrow_{(\phi_2,h_2)} \mathsf{Eq}_2$, where Eq_2 is inconsistent in G_1. As shown in Fig. 2, the coercion G_2 of Eq_2 on G is to merge v'_1 and v'_2 with distinct labels. The result of this sequence is $\perp$. □

As opposed to chase of relations or RDF with EGDs or FDs [2, 7, 16, 30], a chasing sequence with GEDs operates on a graph (pattern), and may be invalid due to label or attribute conflicts. Moreover, it supports "attribute generation" (cases (1) and (2) of chase steps above) to cope with schemaless graphs. In addition, the relational and RDF chasing rules do not deal with id literals. When $x.\mathsf{id} = y.\mathsf{id}$ is enforced, all their attributes and edges have to be merged.

4.2 The Church Rosser Property

The chase with relational FDs has the Church-Rosser property (cf. [1]). We show that chasing with GEDs retains the property. To present this, we use the following notions. We consider finite sets Σ of GEDs as usual.

(a) Chasing with GEDs is *finite* if for all sets Σ of GEDs and graphs G, all chasing sequences of G by Σ are finite.

(b) Chasing with GEDs has the *Church-Rosser property* if for all Σ and G, all terminal chasing sequences of G by Σ have the same result, regardless of in what order the GEDs are applied. That is, terminal sequences are either (a) all valid with the same $(\mathsf{Eq}, G_{\mathsf{Eq}})$, or (b) all invalid with $\perp$.

While chasing with GEDs may get into conflicts, all terminal valid chasing sequences yield the same result.

Theorem 1: *Chasing with* GEDs *is finite and has the Church-Rosser property. Moreover, for any set Σ of* GEDs *and graph G, if there exists a valid terminal chasing sequences of G by Σ, then $G_{\mathsf{Eq}} \models \Sigma$, where $(\mathsf{Eq}, G_{\mathsf{Eq}})$ is the result of the terminal sequence.* □

By Theorem 1, we can define *the result of chasing G by* Σ as the result of any terminal chasing sequence of G by Σ, denoted by $\mathsf{chase}(G, \Sigma)$. We say that $\mathsf{chase}(G, \Sigma)$ is *consistent* if there exists such a valid terminal chasing sequence, with result $(\mathsf{Eq}, G_{\mathsf{Eq}})$. It is *inconsistent* otherwise, *i.e.,* when all terminal chasing sequences are invalid.

Proof: (a) We show that in any chasing sequence ρ of G by Σ, the equivalence relation Eq_i in any chase step has size

at most $|\mathsf{Eq}_i| \leq 4 \cdot |G| \cdot |\Sigma|$. Based on the bound, one can readily verify that the length of ρ is at most $8 \cdot |G| \cdot |\Sigma|$.

(b) We show the Church-Rosser property by contradiction. Assume that there exist two terminal chasing sequences with different results. We show that one of the sequences must not be terminal, by distinguishing the case when both sequences are valid and the case when only one is valid.

(c) We show that $G_{\mathsf{Eq}} \models \Sigma$ by the definition of terminal chasing sequences and the Church-Rosser property. □

5. REASONING ABOUT GEDS

We next study three fundamental problems associated with GEDs and their sub-classes identified in Section 3. We characterize their static analyses and establish their complexity in various settings (Sections 5.1 and 5.2). We also investigate their validation problem (Section 5.3).

5.1 The Satisfiability Problem

We study a strong notion of satisfiability. Consider a set Σ of GEDs. A *model* of Σ is a graph G such that (a) $G \models \Sigma$, and (b) for each $Q[\bar{x}](X \to Y)$ in Σ, Q has a match in G.

Intuitively, if Σ has a model, then the GEDs in Σ are sensible and do not conflict with each other. Hence we can apply these GEDs without worrying about their conflicts.

The *satisfiability* problem for GEDs is as follows.

- Input: A finite set Σ of GEDs.
- Question: Does there exist a model of Σ?

We say that Σ is *satisfiable* if it has a model.

For relational FDs, the satisfiability problem is trivial: there always exists a nonempty relation that satisfies a given set of FDs (cf. [20]). When it comes to GEDs over graphs, however, the satisfiability analysis is more intriguing.

Example 5: (1) Consider a set Σ_1 consisting of

$\phi_1 = Q_1[x, y, z](x.A = x.B \to y.\mathsf{id} = z.\mathsf{id})$,
$\phi_2 = Q_2[x_1, y_1, z_1, x_2, y_2, z_2](\emptyset \to x_1.A = x_1.B)$,

where Q_1 and Q_2 are depicted in Fig. 3. One can verify that each of ϕ_1 and ϕ_2 has a model when they are taken separately; however, Σ_1 does not have a model. To see this, consider a homomorphism f from Q_2 to Q_1, mapping x_1 and x_2 to x, y_1 and y_2 to y, and z_1 and z_2 to z. Hence for any match h of Q_1 in a graph G, the composition of h and f makes a match of Q_2 in G. When taken together, ϕ_1 and ϕ_2 require us to merge two nodes y and z with distinct labels.

(2) GEDs may interact with each other even when their patterns are not homomorphic. To see this, consider Σ_2 consisting of ϕ_1 and $\phi_2' = Q_2'[\bar{x}](\emptyset \to x_1.A = x_1.B)$, where Q_2' extends Q_2 by adding a connected component C_2, as shown in Fig. 3. Obviously, Q_1 is not homomorphic to Q_2' and vice versa. However, Σ_2 is not satisfiable. To see this, suppose by contradiction that Σ_2 has a model G, in which Q_2' has a match $h_2(\bar{x})$. Then for any match h_1 of Q_1 in G, we can construct a match h_2' of Q_2' such that (a) over C_2, h_2' is the same as h_2, and (b) over Q_2, h_2' is the composition of h_1 and f given above. Then the same conflict emerges as in (1). □

The example also illustrates complications introduced by the homomorphism semantics for pattern matching. Under the semantics of subgraph isomorphism [23], Q_2 and Q_2' cannot find a match in Q_1 and introduce no conflicts.

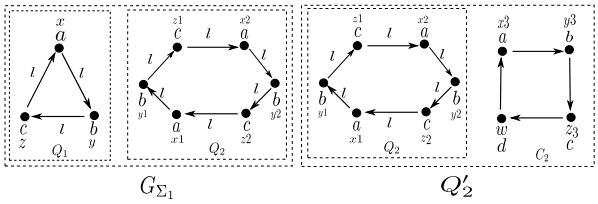

Figure 3: The satisfiability of GEDs

Characterization. We develop a sufficient and necessary condition to characterize the satisfiability of a set Σ of GEDs.

Consider a set Σ of GEDs $\varphi_i = Q_i[\bar{x}_i](X_i \to Y_i)$ for $i \in [1, n]$, where $Q_i = (V_i, E_i, L_i)$, and we assume *w.l.o.g.* that V_i and V_j are disjoint if $i \neq j$, after naming the nodes in Q_i.

The *canonical graph* G_Σ of Σ is defined to be a graph $(V_\Sigma, E_\Sigma, L_\Sigma, F_A^\Sigma)$, where V_Σ is the union of V_i's, and similarly for E_Σ and L_Σ; but F_A^Σ is empty.

Intuitively, G_Σ is the union of all graph patterns in Σ, in which patterns from different GEDs are disjoint.

We chase the pattern graph G_Σ with Σ, and characterize the satisfiability of Σ based on the chase (Section 4).

Theorem 2: *A set Σ of* GEDs *is satisfiable if and only if* $\mathsf{chase}(G_\Sigma, \Sigma)$ *is consistent.* □

Example 6: Recall the set Σ_1 of GEDs from Example 5. Its canonical graph is the union G_{Σ_1} of Q_1 and Q_2 shown in Fig. 3. One can verify that $\mathsf{chase}(G_{\Sigma_1}, \Sigma_1)$ is inconsistent, *i.e.,* there exists a terminal chasing sequence of G_{Σ_1} by Σ_1 with result $\bot$. This confirms the observation of Example 5 that Σ_1 is not satisfiable; similarly for Σ_2. □

Proof: (a) If $\mathsf{chase}(G_\Sigma, \Sigma)$ is consistent, *i.e.,* there exists a valid terminal chasing sequence ρ of G_Σ by Σ, we show that one can build a model of Σ from G_Σ based on ρ, using Theorem 1. We take special care to handle '_' in Σ.

(b) If Σ is satisfiable, *i.e.,* Σ has a model G, we show that each terminal chasing sequence ρ of G_Σ by Σ is valid. For a pattern Q, an equivalence relation Eq on the nodes and attributes of G and a match h of Q in G, we represent Eq as a set of equality literals, and write $h \models \mathsf{Eq}$ if $h \models l$ for all literals l in Eq. We construct a match h of G_Σ in G by treating G_Σ as a graph pattern, and show that $h \models \mathsf{Eq}_{i+1}$ for each chase step $\mathsf{Eq}_i \Rightarrow_{(\varphi,h)} \mathsf{Eq}_{i+1}$ of ρ. □

Complexity. Using Theorem 2, we give the complexity of the satisfiability analysis of GEDs and its sub-classes.

Theorem 3: *The satisfiability problem is*

- coNP*-complete for* GEDs, GFDs, GKeys, $\mathsf{GED_x}$s*;*
- *it is in $O(1)$ time for* $\mathsf{GFD_x}$s. □

Theorem 3 tells us the following. (1) The intractability of the satisfiability analysis is rather robust: it arises either from constant literals in GFDs, or from id literals in GKeys and $\mathsf{GED_x}$s. As will be seen in the proof, the problem is coNP-hard even when Σ consists of a fixed number of GEDs. (2) In the absence of constant and id literals, the problem is trivial: any set of $\mathsf{GFD_x}$s can find a model.

For relational EGDs, the satisfiability problem is not an issue. The satisfiability problem for relational CFDs is NP-complete [21]. A close examination reveals that it is intractable only under a database schema that requires attributes to have a finite domain, *e.g.,* Boolean [21]. It is in

PTIME in the absence of finite-domain attributes. As remarked in Section 3, while GEDs can express CFDs when relations are represented as graphs, they cannot enforce an attribute to have a finite domain. That is, the satisfiability problem for GEDs is intractable in the absence of finite-domain attributes. Hence its intractability is not inherited from CFDs, as indicated by coNP-complete vs. NP-complete.

Proof: We give an NP algorithm to check whether a set Σ of GEDs is not satisfiable, based on Theorem 2. This is possible because of the bound on terminal chasing sequences by GEDs given in the proof of Theorem 1.

In particular, when Σ consists of $\mathsf{GFD_x}$s, $\mathsf{chase}(G_\Sigma, \Sigma)$ is always consistent. Indeed, in the absence of constant and id literals, no chase step can result in conflicts.

For the lower bounds, we show that the problem is coNP-hard for (a) GFDs, and (b) GKeys without constant literals; these suffice since such GKeys are a special case of $\mathsf{GED_x}$s and GEDs. We prove (a) and (b) by (different) reductions from the complement of the 3-colorability problem. The 3-colorability problem is to decide, given an undirected graph G, whether there exists a proper 3-coloring ν of G such that for each edge (u, v) in G, $\nu(u) \neq \nu(v)$. The problem is NP-complete even when G is connected [25].

The proof for (a) uses two GFDs of the form $Q[\bar{x}](\emptyset \rightarrow Y)$, where Y consists of variable and constant literals. It is different from the one given in [23], where GFDs are interpreted via subgraph isomorphism, while we adopt graph homomorphism here. For (b) we use three GKeys of the form $Q[\bar{x}](\emptyset \rightarrow x.\mathsf{id} = y.\mathsf{id})$ without constant literals. □

5.2 The Implication Problem

A set Σ of GEDs *implies* another GED φ, denoted by $\Sigma \models \varphi$, if for all graphs G, if $G \models \Sigma$ then $G \models \varphi$.

The *implication* problem for GEDs is as follows:

- Input: A finite set Σ of GEDs and another GED φ.
- Question: Does $\Sigma \models \varphi$?

The implication analysis helps us optimize data quality rules and graph pattern queries, among other things.

Characterization. We characterize the implication $\Sigma \models \varphi$ as follows. Assume $\varphi = Q[\bar{x}](X \rightarrow Y)$, where pattern $Q = (V_Q, E_Q, L_Q)$. We use the following notations.

(a) The *canonical graph* of Q is $G_Q = (V_Q, E_Q, L_Q, F_A)$, where F_A is empty, along the same lines as G_Σ.

(b) We use Eq_X to denote *the equivalence relation of X in G_Q*, such that for any literal l in X, $v \in [u]_{\mathsf{Eq}}$, where l is $u = v$, denoting $x.A = c$, $x.A = y.B$ or $x.\mathsf{id} = y.\mathsf{id}$. Moreover, Eq_X contains $[x]_{\mathsf{Eq}_X} = \{x\}$ for each $x \in V_Q$.

(c) We use $\mathsf{chase}(G_Q, \mathsf{Eq}_X, \Sigma)$ to denote the result of the chase of G_Q by Σ starting with Eq_X. Note that it is inconsistent if Eq_X is inconsistent (see Section 4).

(d) We say that a literal l can be *deduced* from an equivalence relation Eq if $v \in [u]_{\mathsf{Eq}}$, where l is $u = v$. That is, the equality specified by l can be deduced from the transitivity of equality literals, and the semantics of id literals in Eq.

We say that a set Y of literals can be *deduced* from Eq if each literal of Y can be deduced from Eq.

Theorem 4: *For a set Σ of* GEDs *and $\varphi = Q[\bar{x}](X \rightarrow Y)$, $\Sigma \models \varphi$ if and only if either (1)* $\mathsf{chase}(G_Q, \mathsf{Eq}_X, \Sigma)$ *is inconsistent; or (2)* $\mathsf{chase}(G_Q, \mathsf{Eq}_X, \Sigma)$ *is consistent and Y can be deduced from* $\mathsf{chase}(G_Q, \mathsf{Eq}_X, \Sigma)$. □

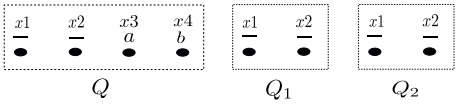

Figure 4: The implication of GEDs

Intuitively, if $\mathsf{chase}(G_Q, \mathsf{Eq}_X, \Sigma)$ is inconsistent, then for all graphs $G \models \Sigma$ and for all matches $h(\bar{x})$ of pattern Q in G, $h(\bar{x}) \not\models X$. Condition (1) covers this case. Otherwise, if $\mathsf{chase}(G_Q, \mathsf{Eq}_X, \Sigma)$ is consistent, condition (2) ensures that Y is a logical consequence of Σ, Q and X.

Example 7: Consider a set $\Sigma_1 = \{\phi_1, \phi_2\}$ and φ:

$\phi_1 = Q_1[x_1, x_2](x_1.A = x_2.A \rightarrow x_1.\mathsf{id} = x_2.\mathsf{id})$,
$\phi_2 = Q_2[x_1, x_2](x_1.B = x_2.B \rightarrow x_1.A = x_1.B)$,
$\varphi = Q[x_1, x_2, x_3, x_4](X \rightarrow Y)$,

where Q, Q_1 and Q_2 are shown in Fig. 4, X is $x_1.A = x_3.A \wedge x_2.B = x_4.B$, and Y is $x_1.\mathsf{id} = x_3.\mathsf{id} \wedge x_2.\mathsf{id} = x_4.\mathsf{id}$. Canonical graph G_Q has the same form as Q of Fig. 4. Then $\mathsf{chase}(G_Q, \mathsf{Eq}_X, \Sigma)$ yields all literals in Y, and $\Sigma \models \varphi$.

Note that x_3 and x_4 have distinct labels, and each is identified with a node labeled '_': $x_3 \in [x_1]_{\mathsf{Eq}}$ and $x_4 \in [x_2]_{\mathsf{Eq}}$, where Eq is the result of the chase. This explains why we use $\asymp$ when comparing labels (see Section 4). □

Theorem 4 tells us that to decide whether $\Sigma \models \varphi$, it suffices to chase the canonical graph G_Q of pattern Q.

Proof: We verify conditions (1) and (2) of Theorem 4 by using Lemmas (a) and (b) below. Consider a terminal chasing sequence $\mathsf{Eq}_X, \mathsf{Eq}_1, \ldots, \mathsf{Eq}_k$ of G_Q by Σ starting with Eq_X, valid or not. We show the following lemmas.

(a) For any graph G and pattern $Q[\bar{x}]$, if $G \models \Sigma$, $h(\bar{x})$ is a match of Q in G and $h(\bar{x}) \models X$, then $h(\bar{x}) \models \mathsf{Eq}_k$.

(b) For consistent $\mathsf{chase}(G_Q, \mathsf{Eq}_X, \Sigma)$, Y can be deduced from $\mathsf{chase}(G_Q, \mathsf{Eq}_X, \Sigma)$ if and only if for any graph G and match $h(\bar{x})$ of Q in G, $h(\bar{x}) \models \mathsf{Eq}_k$ implies $h(\bar{x}) \models Y$. □

Complexity. Based on the characterization, we settle the complexity of the implication analysis of GEDs.

Theorem 5: *The implication problem is* NP*-complete for* GEDs, GFDs, GKeys, $\mathsf{GFD_x}$s *and* $\mathsf{GED_x}$s. □

As opposed to Theorem 3, the implication analysis for $\mathsf{GFD_x}$s is NP-hard, in the absence of constant and id literals, although $\mathsf{chase}(G_Q, \mathsf{Eq}_X, \Sigma)$ is always consistent in this case. This is because to check whether Y can be deduced from $\mathsf{chase}(G_Q, \mathsf{Eq}_X, \Sigma)$, we need to examine all possible homomorphic mappings of patterns of Σ in G_Q. The intractability remains intact even when Σ consists of a single GED.

The lower bound for GEDs does not follow from its counterpart for CFDs, which is coNP-complete [21], for the same reason as for the satisfiability analysis. While the implication problem for EGDs is NP-complete [8], the proofs are quite different, especially for the upper bound for GEDs and lower bound for GKeys, in the presence of id literals.

Proof: We give an NP algorithm to check $\Sigma \models \varphi$ based on the characterization of Theorem 4 and the bound given in the proof of Theorem 1. For the lower bounds, we show that the problem is NP-hard for $\mathsf{GFD_x}$s and GKeys, since GEDs, GFDs and $\mathsf{GED_x}$s subsume $\mathsf{GFD_x}$s. We prove these by (differ-

ent) reductions from the 3-colorability problem, capitalizing on Theorem 4. In the reductions, we use Σ consisting of a single GFD_x ϕ (resp. GKey ψ), where ϕ and φ have the form $Q[\bar{x}](\emptyset \to Y)$ and Y consists of variable literals only (resp. $Q[\bar{x}](\emptyset \to x.\mathsf{id} = y.\mathsf{id})$ for GKeys ψ and φ). □

5.3 The Validation Problem

The *validation problem* for GEDs is stated as follows.

- Input: A finite set Σ of GEDs and a graph G.
- Question: Does $G \models \Sigma$?

As remarked earlier, the validation analysis is the basis of inconsistency and spam detection, to find violations of GEDs in a knowledge base or a social graph.

Recall that validations of relational FDs and CFDs are in PTIME. It is harder for GEDs unless P = NP.

Theorem 6: *The validation problem is* coNP*-complete for* GEDs*,* GFDs*,* GKeys*,* GFD_xs *and* GED_xs. □

As for the implication problem, the validation analysis is intractable even for GFD_xs, which is an extension of relational FDs that carries neither constant literals nor id literals. The intractability remains intact when Σ consists of a single GFD_x or a single GKey. The proof is quite different from the validation analysis of relational EGDs [8].

Proof: We provide an NP algorithm to check whether $G \not\models \Sigma$, for GEDs. We show the lower bounds for GKeys and GFD_xs by (different) reductions from the complement of the 3-colorability problem. These suffice since GFD_xs are a special case of GFDs, GED_xs and GEDs.

In the reductions, we use Σ consisting of only a single GFD_x $Q[\bar{x}](X \to Y)$ (resp. GKeys), where $X = \emptyset$ and Y consists of a single variable literal (resp. id literal). □

Tractable cases. The main conclusion of this section is that the intractability of the analyses of GEDs is quite robust. In fact, even for GEDs defined in terms of tree patterns, the satisfiability, implication and validation problems remain intractable. This is because the analyses require to enumerate and examine all matches of a pattern Q in a graph G in the worst case, not just to check whether there exists a match of Q in G. We defer the proof to a latter publication.

Nonetheless, there are tractable cases that allow us to make effective use of GEDs. For example, one may consider a set Σ of GEDs in which graph patterns have a size at most k, for a predefined bound k. This is practical. Indeed, real-life graph patterns often have a small size: 98% of SPARQL queries have no more than 4 nodes and 5 edges, and single-triple patterns account for 97.25% of patterns in SWDF and 66.41% of DBPedia [24]. One can readily verify that the satisfiability, implication and validation problems for GEDs are in PTIME when patterns have a bounded size k.

6. FINITE AXIOMATIZABILITY

We next study the finite axiomatizability of GEDs.

We naturally want a finite set $\mathcal{A}$ of inference rules to characterize GED implication, along the same lines as Armstrong's axioms for relational FDs [5]. As observed in [1], the finite axiomatizability of a dependency class is a stronger property than the existence of an algorithm for testing its implication. An axiom system reveals insight of logical implication, and can be used to generate symbolic proofs.

GED_1	$\Sigma \vdash Q[\bar{x}](X \to X \wedge X_\mathsf{id})$, where X_id is $\bigwedge_{i\in[1,n]}(x_i.\mathsf{id} = x_i.\mathsf{id})$, and $\bar{x}$ consists of x_i for all $i \in [1,n]$.
GED_2	If $\Sigma \vdash Q[\bar{x}](X \to Y)$ and literal $(u.\mathsf{id} = v.\mathsf{id}) \in Y$, then $\Sigma \vdash Q[\bar{x}](X \to u.A = v.A)$ for all attributes $u.A$ that appear in Y.
GED_3	If $\Sigma \vdash Q[\bar{x}](X \to Y)$ and $(u = v) \in Y$, then $\Sigma \vdash Q[\bar{x}](X \to v = u)$.
GED_4	If $\Sigma \vdash Q[\bar{x}](X \to Y)$, $(u_1 = v) \in Y$ and $(v = u_2) \in Y$, then $\Sigma \vdash Q[\bar{x}](X \to u_1 = u_2)$.
GED_5	If $\Sigma \vdash Q[\bar{x}](X \to Y)$ and $\mathsf{Eq}_X \cup \mathsf{Eq}_Y$ is inconsistent, then $\Sigma \vdash Q[\bar{x}](X \to Y_1)$ for any set Y_1 of literals of $\bar{x}$.
GED_6	If $\Sigma \vdash Q[\bar{x}](X \to Y)$, $\mathsf{Eq}_X \cup \mathsf{Eq}_Y$ is consistent, $\Sigma \vdash Q_1[\bar{x}_1](X_1 \to Y_1)$, and if there exists a match h of Q_1 in $(G_Q)_{\mathsf{Eq}_X \cup \mathsf{Eq}_Y}$ such that $h(\bar{x}_1) \models X_1$, then $\Sigma \vdash Q[\bar{x}](X \to Y \wedge h(Y_1))$.

Table 2: Axiom system $\mathcal{A}_\mathsf{GED}$ for GEDs

For a set Σ of GEDs and a GED φ, *a proof of* φ *from* Σ *using inference rules of* $\mathcal{A}$ is a sequence of GEDs

$$\varphi_1, \ \ldots, \ \varphi_n = \varphi,$$

such that each φ_i either is a GED in Σ, or can be deduced from φ_j's by applying an inference rule (or axiom) in $\mathcal{A}$, for $j < i$ (see [1] for details about proofs).

We say that φ is *provable from* Σ *using* $\mathcal{A}$, denoted by $\Sigma \vdash_\mathcal{A} \varphi$, if there exists a proof of φ from Σ using $\mathcal{A}$. We write it as $\Sigma \vdash \varphi$ when $\mathcal{A}$ is clear from the context.

We say that for GEDs, an inference system $\mathcal{A}$ is

- *sound* if $\Sigma \vdash_\mathcal{A} \varphi$ implies $\Sigma \models \varphi$;
- *complete* if $\Sigma \models \varphi$ implies $\Sigma \vdash_\mathcal{A} \varphi$;

for all GED sets Σ and GEDs φ; and

- *independent* if for any rule $\mathsf{r} \in \mathcal{A}$, there exist GEDs Σ and φ such that $\Sigma \vdash_\mathcal{A} \varphi$ but $\Sigma \not\vdash_{\mathcal{A}\setminus\mathsf{r}} \varphi$.

Here $\mathcal{A}\setminus\mathsf{r}$ denotes $\mathcal{A}$ excluding r. That is, removing any rule from $\mathcal{A}$ would make it no longer complete. We remark that we focus on finite implication, considering finite graphs.

We refer to $\mathcal{A}$ as a *finite axiom system* or a *finite axiomatization* of GEDs if $\mathcal{A}$ is sound, complete and independent for GEDs. We say that GEDs are *finitely axiomatizable* if there exists a finite axiomatization of GEDs [1].

Inference rules. We give a set $\mathcal{A}_\mathsf{GED}$ of rules for GEDs in Table 2, in which we denote by (a) $Q[\bar{x}]$ a pattern; (b) X a set of literals of $\bar{x}$; (c) $h(X)$ the set of literals obtained by substituting $h(x)$ for all $x \in X$, for a match h of Q in a graph; (d) G_Q the canonical graph of pattern Q (Section 5.2); (e) Eq_X the equivalence relation of a set X of literals in G_Q; and (f) $(G_Q)_\mathsf{Eq}$ the coercion of Eq on G_Q (Section 4). The consistency of an equivalence relation Eq is defined in Section 4. To simplify the presentation, we allow $c = x.A$ as a literal in intermediate results of a proof, for constant c.

Recall that Armstrong's axioms consist of three rules for relational FDs: reflexivity, augmentation and transitivity [5]. Four rules are needed for CFDs [21] and EGDs [38]. In contrast, $\mathcal{A}_\mathsf{GED}$ has six rules for GEDs over graphs.

Example 8: (a) We first prove the following property: if $\Sigma \vdash \varphi$, $\varphi = Q[\bar{x}](X \to Y)$ and $Y_1 \subseteq Y$, then $\Sigma \vdash Q[\bar{x}](X \to Y_1)$, where Y_1 is a set $\{u_i = v_i \mid i \in [1,n]\}$ of literals that are also in Y. When $X \cup Y$ is consistent, we have

(1) $Q[\bar{x}](X \to Y)$	φ
(2) $Q[\bar{x}](X \to (v_1 = u_1))$	(1) and GED_3
(3) $Q[\bar{x}](X \to (u_1 = v_1))$	(2) and GED_3
...	
(2n+1) $Q[\bar{x}](X \to (u_n = v_n))$	(2n) and GED_3

(2n+2) $Q[\bar{x}](X \to (u_1 = v_1)(u_2 = v_2))$ (3), (5) and GED_6
...
(3n) $Q[\bar{x}](X \to Y_1)$ (3n-1), (2n+1) and GED_6

It can also be proven for inconsistent $X \cup Y$. To simplify the presentation, we denote this property as GED_7 and apply it in proofs, although GED_7 is not in $\mathcal{A}_{\mathsf{GED}}$.

(b) Recall the augmentation rule of Armstrong's axioms: if $X \to Y$ then $XZ \to YZ$. Analogously, consider $\Sigma \vdash \varphi_1$, where $\varphi_1 = Q[\bar{x}](X \to Y)$, and GED $\varphi = Q[\bar{x}](XZ \to YZ)$. We show that $\Sigma \vdash \varphi$ using $\mathcal{A}_{\mathsf{GED}}$ as follows. First consider the case when $\mathsf{Eq}_X \cup \mathsf{Eq}_Z$ is consistent:

(1) $Q[\bar{x}](XZ \to XZ \wedge X_{\mathsf{id}})$	GED_1
(2) $Q[\bar{x}](XZ \to XZ)$	(1) and GED_7
(3) $Q[\bar{x}](X \to Y)$	φ_1
(4) $Q[\bar{x}](XZ \to XYZ)$	(2), (3) and GED_6
(5) $Q[\bar{x}](XZ \to YZ)$	(3) and GED_7

When $\mathsf{Eq}_X \cup \mathsf{Eq}_Z$ is inconsistent, the proof consists of steps (1) and (2) above, followed by:

(3) $Q[\bar{x}](XZ \to YZ)$ (2) and GED_5

(c) Let $\Sigma \vdash \varphi_1$ and $\Sigma \vdash \varphi_2$, where $\varphi_1 = Q[\bar{x}](X \to Y)$ and $\varphi_2 = Q[\bar{x}](Y \to Z)$. We show that $\Sigma \vdash Q[\bar{x}](X \to Z)$ using $\mathcal{A}_{\mathsf{GED}}$. When $\mathsf{Eq}_X \cup \mathsf{Eq}_Y$ is consistent, we have:

(1) $Q[\bar{x}](X \to X \wedge X_{\mathsf{id}})$	GED_1
(2) $Q[\bar{x}](X \to X)$	(1) and GED_7
(3) $Q[\bar{x}](X \to Y)$	φ_1
(4) $Q[\bar{x}](X \to XY)$	(2), (3) and GED_6
(5) $Q[\bar{x}](Y \to Z)$	φ_2
(6) $Q[\bar{x}](X \to XYZ)$	(4), (5) and GED_6
(7) $Q[\bar{x}](X \to Z)$	(6) and GED_7

If Eq_X is inconsistent, the proof has steps (1), (2) and

(3) $Q[\bar{x}](X \to Z)$ (2) and GED_5

If $\mathsf{Eq}_X \cup \mathsf{Eq}_Y$ is inconsistent, it has steps (1)–(3) and

(4) $Q[\bar{x}](X \to XY)$	(2), (3) and GED_6
(5) $Q[\bar{x}](X \to Z)$	(4) and GED_5

These prove the transitivity of Armstrong's axioms. □

Axiomatization. GEDs are finitely axiomatizable.

Theorem 7: *The set $\mathcal{A}_{\mathsf{GED}}$ of rules given in Table 2 is sound, complete and independent for* GEDs. □

Proof: We outline a proof, highlighting intuition.

(1) Soundness. The soundness is verified by induction on the length of proofs by using $\mathcal{A}_{\mathsf{GED}}$, based on the chase and Theorem 4. Below we illustrate each rule in $\mathcal{A}_{\mathsf{GED}}$.

(a) GED_1 extends the reflexivity of Armstrong's axioms to cover id literals. Similarly, GED_3 and GED_4 ensure that equality literals are symmetric and transitive.

(b) GED_2 enforces the semantics of id literals: if x and y refer to the same node, then they have the same sets of attributes with the same values $x.A = y.A$.

(c) If $\mathsf{Eq}_X \cup \mathsf{Eq}_Y$ is inconsistent, then $\mathsf{chase}(G_Q, \mathsf{Eq}_X, \Sigma)$ is inconsistent, since Eq_X and Eq_Y are included in its result. GED_5 says that if this happens, then any set Y_1 of literals of $\bar{x}$ is a "logical consequence" of the inconsistent X, Σ and Q, following condition (1) of Theorem 4.

(d) When $\mathsf{Eq}_X \cup \mathsf{Eq}_Y$ is consistent, Q_1 can be embedded in $(G_Q)_{\mathsf{Eq}_X \cup \mathsf{Eq}_Y}$ via a match h, and if $h(\bar{x}_1) \models X_1$, then one can verify that if $\mathsf{chase}(G_Q, \mathsf{Eq}_X, \Sigma)$ is consistent, then $h(Y_1)$ can be deduced from $\mathsf{chase}(G_Q, \mathsf{Eq}_X, \Sigma)$. Hence GED_6 follows from condition (2) of Theorem 4.

Observe that GED_2 and GED_6 are unique for graph dependencies, which are needed to handle id-based entity identification and embedding of graph patterns, respectively.

(2) Completeness. Assume that $\Sigma \models Q[\bar{x}](X \to Y)$. To prove that $\Sigma \vdash Q[\bar{x}](X \to Y)$, for a terminal chasing sequence ρ of G_Q by Σ, where ρ is $\mathsf{Eq}_1 = \mathsf{Eq}_X, \mathsf{Eq}_2, \ldots, \mathsf{Eq}_k$, we treat Eq_i as a set of equality literals. Then we show the following claims by induction on the length of ρ.

Claim 1: For each $1 \leq i \leq k$, $\Sigma \vdash Q[\bar{x}](X \to \mathsf{Eq}_i)$.

Claim 2: If there exist GED $\varphi \in \Sigma$ and match h of the pattern of φ such that $\mathsf{Eq}_k \Rightarrow_{(\varphi,h)} \mathsf{Eq}_{k+1}$ and Eq_{k+1} is inconsistent in G_{Eq_k}, then $\Sigma \vdash Q[\bar{x}](X \to \mathsf{Eq}_{k+1})$.

We can verify that $\Sigma \vdash Q[\bar{x}](X \to Y)$ using the claims as follows. By Theorem 4, if $\Sigma \models Q[\bar{x}](X \to Y)$, then we need to consider two cases: (a) $\mathsf{chase}(G_Q, \mathsf{Eq}_X, \Sigma)$ is inconsistent; and otherwise, (b) Y can be deduced from $\mathsf{chase}(G_Q, \mathsf{Eq}_X, \Sigma)$. In case (a), Claim 2 and GED_5 put together can derive $\Sigma \vdash Q[\bar{x}](X \to Y)$. In case (b), we can show that $\Sigma \vdash Q[\bar{x}](X \to Y)$ following Claim 1.

(3) Independence. For each rule GED_k in $\mathcal{A}_{\mathsf{GED}}$, we show that there exist a set Σ of GEDs and another GED φ, such that the proof of $\Sigma \vdash \varphi$ necessarily uses GED_k.

Take GED_5 as an example. Consider $\Sigma = \emptyset$ and $\varphi = Q_5[x]((x.A = 1) \wedge (x.A = 2) \to x.A = 3)$, where Q_5 consists of a single node x. We show by contradiction that without using GED_5, we cannot prove $\Sigma \vdash \varphi$. Indeed, no other rule allows us to deduce $Q[\bar{x}](X \to Y)$ when Y contains a constant that appears in neither X nor Σ. □

7. EXTENSIONS OF GEDS

We next extend GEDs by supporting built-in predicates (Section 7.1) or disjunctions (Section 7.2). We show that the extensions complicate the static analyses.

7.1 Denial Constraints for Graphs

We first extend GEDs with built-in predicates, referred to as *graph denial constraints*, denoted by GDCs.

GDCs. A GDC ϕ is defined as $Q[\bar{x}](X \to Y)$, where Q is a pattern, and X and Y are sets of literals of one of the following forms: (a) $x.A \oplus c$, (b) $x.A \oplus y.B$, for constant $c \in U$, and non-id attributes $A, B \in \Upsilon$, and (c) $x.\mathsf{id} = y.\mathsf{id}$; here $\oplus$ is one of built-in predicates $=, \neq, <, >, \leq, \geq$.

Along the same lines as GEDs, we define $G \models \phi$ for a graph G; similarly for other notions. Obviously GEDs are a special case of GDCs when $\oplus$ is equality '=' only. One can verify that GDCs can express denial constraints of [3] when relation tuples are represented as vertices in a graph.

Example 9: We can express "domain constraints" as GDCs, to enforce each node of "type" τ to have an attribute with a finite domain, *e.g.,* Boolean, as follows:

ϕ_1: $Q_e[x](\emptyset \to x.A = x.A)$,
ϕ_2: $Q_e[x](x.A \neq 0 \wedge x.A \neq 1 \to \mathsf{false})$.

Here Q_e consists of a single node labeled τ, ϕ_1 is a GED that enforces each τ-node x to have an A-attribute, and ϕ_2 ensures that $x.A$ can only takes values 0 or 1. □

Complexity. The increased expressive power of GDCs comes with a price. Recall that the satisfiability, implication and validation problems for GEDs are coNP-complete,

NP-complete and coNP-complete, respectively. In contrast, the static analyses of GDCs have a higher complexity unless P = NP, although their validation problem gets no harder.

Theorem 8: *The satisfiability, implication and validation problems for* GDCs *are* Σ_2^p*-complete,* Π_2^p*-complete and* coNP*-complete, respectively.* □

The lower bounds of these problems remain intact when Σ consists of a fixed number of GDCs with variable and constant literals only. The proof of Theorem 8 is more involved than their counterpart for GEDs (Theorems 3, 5 and 6).

Proof: (1) To prove the upper bound of the satisfiability problem, we establish a small model property, as opposed to the proof of Theorem 3 that is based on the chase. We show that if a set Σ of GDCs has a model, then it has a model of size at most $4 \cdot |\Sigma|^3$. The proof requires attribute value normalization. Based on the property, we give an Σ_2^p algorithm to check whether a set of GDCs is satisfiable.

We show the lower bound by reduction from a generalized graph coloring problem (GGCP) [37, 40]. GGCP is to decide, given two undirected graphs $F = (V_F, E_F)$ and $G = (V_G, E_G)$, whether there exists a two-coloring of F such that G is not a monochromatic subgraph of F. A monochromatic subgraph of F is a subgraph in which nodes are assigned the same color. The problem is Σ_2^p-complete when G is a complete graph and F contains no self cycles [37].

The reduction is a little complicated. We use a set Σ of four GDCs to encode 2-coloring, monochromatic G and graph F. These GDCs use constant and variable literals with $\neq$ and $\leq$, but employ no id literals. One of them is a forbidding constraint of the form $Q[\bar{x}](X \rightarrow \mathsf{false})$.

(2) For implication, we also show a small model property: if $\Sigma \not\models \varphi$, then there exists a graph G_h such that $|G_h| \leq 2 \cdot |\varphi| \cdot (|\varphi| + |\Sigma| + 1)^2$, $G_h \models \Sigma$ and $G_h \not\models \varphi$. Based on the property, we give an Σ_2^p algorithm to check $\Sigma \not\models \varphi$. The lower bound is verified by reduction from the complement of GGCP, using Σ of three GDCs of the form above.

(3) For validation, the lower bound follows from Theorem 6 since GEDs are a special case of GDCs. For the upper bound, we use the algorithm for checking $G \not\models \Sigma$ developed for GEDs in the proof of Theorem 6. We show that the algorithm also works for GDCs and better still, remains in NP. □

7.2 Adding Disjunction

We next extend GEDs by adding limited disjunctions.

GED$^\vee$s. A GED ψ *with disjunction*, denoted by GED$^\vee$, has the same syntactic form $Q[\bar{x}](X \rightarrow Y)$ as GEDs, but Y is interpreted as the disjunction of its literals. That is, for a match $h(\bar{x})$ of Q in a graph G, $h(\bar{x}) \models Y$ if *there exists* a literal $l \in Y$ such that $h(\bar{x}) \models l$. Hence we also write ψ as

$$Q[\bar{x}](\textstyle\bigwedge_{l \in X} l \rightarrow \bigvee_{l' \in Y} l').$$

The other notions such as satisfiability and implication remain the same as their GED counterparts.

GED$^\vee$s subsume GEDs. Each GED $Q[\bar{x}](X \rightarrow Y)$ can be expressed as a set of $Q[\bar{x}](X \rightarrow l)$ of GED$^\vee$s, one for each $l \in Y$. In contrast, some GED$^\vee$s are not expressible as GEDs.

Example 10: Recall GDCs from Example 9 that enforce $x.A$ to be Boolean. It is expressible as a GED$^\vee$:

$$\psi\colon Q_e[x](\emptyset \rightarrow x.A = 0 \vee x.A = 1).$$

It specifies a domain constraint: each τ-node x has an A-attribute and that $x.A$ can only take Boolean values. □

Complexity. Disjunctions also complicate the static analyses but do not make the validation analysis harder. The lower bounds remain intact when Σ consists of a fixed number of GED$^\vee$s with constant and variable literals only.

Theorem 9: *The satisfiability, implication and validation problems for* GED$^\vee$s *are* Σ_2^p*-complete,* Π_2^p*-complete and* coNP*-complete, respectively.* □

Proof: The proof is similar to the one for Theorem 8. For satisfiability (resp. implication), the upper bound is also verified by means of a small model property, and the lower bound by reduction from (resp. the complement of) GGCP, by using a set Σ consisting of three GED$^\vee$s. □

8. RELATED WORK

We categorize related work as follows.

Relational dependencies. FDs were introduced in [15] and have been well studied for relations. Armstrong's axioms were proposed for FDs in [5], and the chase in [39]. EGDs and TGDs were introduced in [7]. There were also renewed interests in extending FDs to improve data quality, *e.g.,* denial constraints [3] and CFDs [21] (see [1,18,20] for surveys).

For relational FDs, the satisfiability, implication and validation problems are in $O(1)$, linear time and PTIME, respectively (cf. [1]). Similar to the strong notion of satisfiability studied in this work, a consistency problem was shown NP-complete and undecidable for EGDs and TGDs [26], respectively; their implication problems are also NP-complete and undecidable, respectively [8]; and the validation problem was shown coNP-complete for EGDs [8] and Π_2^p-complete for TGDs [36]. The satisfiability, implication and validation problems are NP-complete, coNP-complete and in PTIME for CFDs [21], respectively. The satisfiability and implication problems are NP-complete and coNP-complete for denial constraints [6], respectively. An axiom system of four rules was developed for EGDs in [38], while TGDs are not finitely axiomatizable for finite implication. A set of four rules was shown sound and complete for CFDs [21].

GEDs carry graph patterns and id literals. Their satisfiability, implication and validation problems are intractable. However, their static analyses bear complexity comparable to their counterparts for denial constraints, CFDs and EGDs. Moreover, GEDs have the finite axiomatizability and the Church-Rosser property of the chase, as for relational FDs.

One might want to encode GEDs as relational dependencies and employ relational techniques to reason about GEDs. However, (a) id literals and graph patterns with wildcard complicate the encoding; and (b) it is not clear what we can get from an encoding. To express GEDs we need both EGDs and limited TGDs. Reasoning about generic TGDs is beyond reach [8, 26]. While some special cases have been studied, *e.g.,* oblivious terminating TGDs and EGDs [33, 34], their syntactic characterization is not yet in place, and their fundamental problems such as satisfiability and validation are still open. It is not clear whether GEDs can be expressed in the special forms, and even so, what results can GEDs inherit from them. In light of this, we opt to give a clean native definition of GEDs and develop their proofs directly. (c) The chase and axiom system for GEDs are quite different from their counterparts in the relational setting. For instance, chasing with GEDs may expand a graph with new attributes and run into conflicts, in contrast to with EGDs.

FDs *for graphs.* Graph constraints are being investigated by W3C [31] and industry (*e.g.,* [35]). The constraints currently supported are quite simple, *e.g.,* uniqueness constraints, cardinality constraints and property paths; a "standard" form of FDs is not yet in place. However, there have been several research proposals for FDs on RDF graphs. This line of work started from [32]. It defines keys, foreign keys and FDs by extending relational methods to RDF, and interpreting the "scope" of an FD with a class type that represents a relation name. Using clustered values, [42] defines FDs with conjunctive path patterns, which were extended to CFDs [28]. FDs are also defined by mapping relations to RDF [13], with tree patterns in which nodes represent relation attributes. As opposed to class names [32], tree patterns [13] and path patterns [28, 42], GEDs are specified with (possibly cyclic) graph patterns with variables and node identities.

Closer to this work are [2, 16, 29, 30] for RDF. A class of EGDs was formulated in [2] in terms of RDF triple patterns with variables, which are interpreted with homomorphism and triple embedding. Along the same lines, a class of FDs, tuple-generating dependencies (TGDs) and forbidding dependencies were defined for RDF in [16]. The FDs were extended in [30] to support constants like CFDs [21]. Chasing algorithms were developed in [2, 29, 30] for the implication analysis of EGDs and FDs. The decidability of the implication and validation problems was established in [16] for the EGDs (and hence FDs), among other things. Finite axiom systems were provided for the EGDs, TGDs, and for EGDs and TGDs put together, consisting of 9, 5 and 16 rules, respectively [2, 16]. Several axiom systems were also provided for various classes of FDs over relations of an arbitrary arity [29, 30], with 13 rules for the general case.

This work differs from [2, 16, 29, 30] in the following.

(1) GEDs are defined for general property graphs, not limited to RDF. (a) GEDs distinguish node identity from value equality. Their id literals enforce that nodes identified have the same attributes and edges. (b) GEDs can uniformly express GFDs, keys of [19] and forbidding dependencies (Section 3). (c) GEDs support constant literals, beyond [2,16,29].

(2) Our revised chase differs from the prior work in the following. (a) We study the chase of a graph (pattern) by GEDs, not limited to the implication analysis. For instance, the chase also helps us characterize the satisfiability analysis. (b) Chasing with GEDs has to deal with id literals, a major cause of invalid steps. It may also add new attributes as enforced by GEDs. (c) We establish the Church-Rosser property of the chase, which was not considered in [2, 16, 29, 30].

(3) We provide characterizations of the static analyses of GEDs, and the complexity of the satisfiability, implication and validation problems for GEDs in various settings. The satisfiability problem was not studied for EGDs or FDs of [2, 16, 29, 30]. Moreover, the complexity bounds remain to be developed for their implication and validation problems.

(4) The axiom system $\mathcal{A}_{\mathsf{GED}}$ differs from [2, 16, 29, 30] in the following. (a) Besides value-based reasoning, $\mathcal{A}_{\mathsf{GED}}$ deals with id-based deduction to enforce the semantics of node identities. (b) It adopts graph pattern matching in property graphs, beyond RDF and relations. (c) $\mathcal{A}_{\mathsf{GED}}$ allows attribute generation (Section 4), which is not supported by the axiom systems for EGDs and FDs [2, 16, 29, 30]. While this can be derived from TGDs and EGDs of [16] put together, the finite axiomatizability for finite implication of TGDs requires further investigation [8].

As remarked in Section 5, a class of keys was studied for RDF [19]. Over property graphs, a form of GFDs [23] was defined with a graph pattern Q that is interpreted via subgraph isomorphic mapping. These GFDs can express CFDs [21] when tuples are represented as vertices in a graph, but cannot express keys of [19]. The satisfiability, implication and validation problems are shown coNP-complete, NP-complete and coNP-complete, respectively, for GFDs of [23].

This work differs from our prior work [19, 23] as follows.

(1) GEDs extend GFDs [23] by supporting id literals, and can express the GFDs of [23]. Moreover, to simplify the definition of the keys of [19] and to reason about GFDs and GKeys in a uniform framework, GEDs adopt the graph homomorphism semantics for graph pattern matching, as opposed to subgraph isomorphism [19, 23] (see Section 3).

(2) We revise the chase for GEDs, which was not studied in [23]. A form of chase was studied for keys [19], which is a simple case of the general process studied here.

(3) We establish the complexity of the satisfiability, implication and validation problems for GEDs in various settings. These were not studied in [19], and were considered for GFDs of [23] only. As remarked earlier, we employ characterizations and proof techniques different from [23] to cope with different semantics of graph pattern matching, *e.g.,* the chase to prove upper bounds. We also give lower bounds for GKeys, $\mathsf{GFD}_{\mathsf{x}}$s and $\mathsf{GED}_{\mathsf{x}}$s, which were not studied before.

(4) We provide finite axiomatization for GEDs, which was not considered for GFDs and GKeys [19, 23].

(5) To the best of our knowledge, no previous work has studied graph dependencies defined in terms of built-in predicates or disjunction, including [19, 23].

The chase has also been studied for data exchange with relational (disjunctive) EGDs [11] or FDs [9], for ontology querying [12], and for optimizing SPARQL queries [41] with the constraints of [32]. In contrast, we study the chase of a graph by GEDs, and deal with id literals.

FDs *for XML.* Keys [10, 22] and FDs [4] have also been studied for XML, which are quite different from GEDs in formulation and semantics. As a consequence, the results on XML do not apply to GEDs and vice versa.

9. CONCLUSION

We have proposed GEDs, which can uniformly express GFDs and keys for graphs. For GEDs, we have revised the chase with the Church-Rosser property, provided characterizations for their static analyses, settled the complexity of their satisfiability, implication and validation problems in various settings (Table 1), and shown the finite axiomatizability of their finite implication. We have also studied extensions of GEDs with built-in predicates or disjunction.

One topic for future work is to identify practical special cases in which the static analyses and validation are tractable. Another topic is to develop parallel scalable algorithms for reasoning about GEDs, to warrant speedup with the increase of processors. It is also interesting to study other practical forms of graph dependencies, *e.g.,*TGDs.

10. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] W. Akhtar, A. Cortés-Calabuig, and J. Paredaens. Constraints in RDF. In *SDKB*, pages 23–39, 2010.

[3] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, 1999.

[4] M. Arenas and L. Libkin. A normal form for XML documents. In *PODS*, pages 85–96, 2002.

[5] W. W. Armstrong. Dependency structures of data base relationships. In *IFIP Congress*, pages 580–583, 1974.

[6] M. Baudinet, J. Chomicki, and P. Wolper. Constraint-generating dependencies. *JCSS*, 59(1):94–115, 1999.

[7] C. Beeri and M. Y. Vardi. The implication problem for data dependencies. In *Automata, Languages and Programming*, pages 73–85, 1981.

[8] C. Beeri and M. Y. Vardi. On the complexity of testing implications of data dependencies. Technical report, The Hebrew University of Jeruslem, 1981.

[9] A. Bonifati, I. Ileana, and M. Linardi. Functional dependencies unleashed for scalable data exchange. In *SSDBM*, pages 2:1–2:12, 2016.

[10] P. Buneman, S. B. Davidson, W. Fan, C. S. Hara, and W. C. Tan. Keys for XML. In *WWW*, pages 201–210, 2001.

[11] M. Calautti, S. Greco, C. Molinaro, and I. Trubitsyna. Exploiting equality generating dependencies in checking chase termination. *PVLDB*, 9(5):396–407, 2016.

[12] A. Calì and A. Pieris. On equality-generating dependencies in ontology querying - preliminary report. In *AMW*, 2011.

[13] D. Calvanese, W. Fischl, R. Pichler, E. Sallinger, and M. Simkus. Capturing relational schemas and functional dependencies in RDFS. In *AAAI*, 2014.

[14] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro. Aiding the detection of fake accounts in large scale social online services. In *NSDI*, pages 197–210, 2012.

[15] E. F. Codd. Relational completeness of data base sublanguages. *In: R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California*, 1972.

[16] A. Cortés-Calabuig and J. Paredaens. Semantics of constraints in RDFS. In *AMW*, pages 75–90, 2012.

[17] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, pages 89–124, 2005.

[18] R. Fagin and M. Y. Vardi. The theory of data dependencies - an overview. In *ICALP*, pages 1–22, 1984.

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

[20] W. Fan and F. Geerts. Foundations of data quality management. *Synthesis Lectures on Data Management*, 4(5):1–217, 2012.

[21] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *TODS*, 33(1), 2008.

[22] W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. *J. ACM*, 49(3):368–406, 2002.

[23] W. Fan, Y. Wu, and J. Xu. Functional dependencies for graphs. In *SIGMOD*, 2016.

[24] M. A. Gallego, J. D. Fernández, M. A. Martínez-Prieto, and P. de la Fuente. An empirical study of real-world SPARQL queries. In *USEWOD workshop*, 2011.

[25] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical computer science*, 1(3):237–267, 1976.

[26] M. H. Graham, A. O. Mendelzon, and M. Y. Vardi. Notions of dependency satisfaction. *Journal of the ACM (JACM)*, 33(1):105–129, 1986.

[27] I. Grujic, S. Bogdanovic-Dinic, and L. Stoimenov. Collecting and analyzing data from e-government Facebook pages. In *ICT Innovations*, 2014.

[28] B. He, L. Zou, and D. Zhao. Using conditional functional dependency to discover abnormal data in RDF graphs. In *SWIM*, pages 1–7, 2014.

[29] J. Hellings, M. Gyssens, J. Paredaens, and Y. Wu. Implication and axiomatization of functional constraints on patterns with an application to the RDF data model. In *FoIKS*, 2014.

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

[31] H. Knublauch and D. Kontokostas. Shapes constraint language (SHACL). W3C Working Draft, Feb. 2017. *https://www.w3.org/TR/shacl/#dfn-shacl-instance*.

[32] G. Lausen, M. Meier, and M. Schmidt. SPARQLing constraints for RDF. In *EDBT*, pages 499–509, 2008.

[33] B. Marnette. Generalized schema-mappings: from termination to tractability. In *PODS*, pages 13–22, 2009.

[34] B. Marnette and F. Geerts. Static analysis of schema-mappings ensuring oblivious termination. In *ICDT*, pages 183–195, 2010.

[35] Neo4j Team. The Neo4j developer manual v3.1 (chapter 3.5.2: Constraints), 2017. *http://neo4j.com/docs/developer-manual/current/*.

[36] R. Pichler and S. Skritek. The complexity of evaluating tuple generating dependencies. In *ICDT*, pages 244–255, 2011.

[37] V. Rutenburg. Complexity of generalized graph coloring. In *MFCS*, pages 573–581, 1986.

[38] F. Sadri. *Data dependencies in the relational model of databases, a generalization*. PhD thesis, Princeton Unversity, 1980.

[39] F. Sadri and J. D. Ullman. The interaction between functional dependencies and template dependencies. In *SIGMOD*, pages 45–51, 1980.

[40] M. Schaefer and C. Umans. Completeness in the polynomial-time hierarchy: A compendium. *SIGACT news*, 33(3):32–49, 2002.

[41] M. Schmidt, M. Meier, and G. Lausen. Foundations of SPARQL query optimization. In *ICDT*, pages 4–33, 2010.

[42] Y. Yu and J. Heflin. Extending functional dependency to detect abnormal data in RDF graphs. In *ISWC*, 2011.

Acknowledgments. Fan and Lu are supported in part by ERC 652976, NSFC 61421003, 973 Program 2014CB340302, Beijing Advanced Innovation Center for Big Data and Brain Computing, EPSRC EP/M025268/1, Shenzhen Peacock Program 1105100030834361, Guangdong Innovative Research Team Program 2011D005, the Foundation for Innovative Research Groups of NSFC, and two Innovative Research Grants from Huawei Technologies. Lu is also supported in part by NSFC 61602023.