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.

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 ‘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 blog \( y' \), 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

\[
\psi_1: \text{its title and the id of its primary artist, or} \\
\psi_2: \text{its title and the year of initial release.}
\]

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 \( \psi_1 \), we also need a key to identify artists:

\[
\psi_3: \text{the name of the artist, and the id of an album recorded by the artist.}
\]

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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or re-publish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

PODS ’17, May 14–19, 2017, Chicago, IL, USA.
© 2017 ACM. ISBN 978-1-4503-4198-1/17/05. . . $15.00
DOI: http://dx.doi.org/10.1145/3034786.3056114
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 $R(X \rightarrow Y)$ is defined on a relation schema $R$ with attributes $X$ and $Y$, where $R$ specifies the “scope” of the FD, i.e., $X \rightarrow Y$ 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 $\psi_1 \rightarrow \psi_2$ 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” $X \rightarrow Y$ 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 $\Sigma$ of GEDs, whether there exists a nonempty finite model $G$ of $\Sigma$ that satisfies $\Sigma$, denoted by $G \models \Sigma$ as usual.

(b) The implication problem is to decide whether a set $\Sigma$ of GEDs entails another GED $\varphi$, 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 $\Sigma$ 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 $\Sigma$ literals vs. their absence, and

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 $\Sigma$ of GEDs and another GED $\varphi$, $\Sigma \models \varphi$ if and only if $\varphi$ is provable.
from $\Sigma$ using $A$. 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’s).

We can express, for instance, denial constraints [3] as GDCs, disjunctive GEDs [17] as GED’s, and “domain constraints” for attributes of an entity to have a finite domain as both GDCs and GED’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 $\Sigma^p_2$-complete and $\Pi^p_2$-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 notions. Assume three countably infinite sets $\Sigma$, $\Gamma$ and $U$ of labels, attributes and constants, respectively.

Graphs. A graph $G$ is specified as $\langle V, E, L, F_A \rangle$, 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 $(v, \iota, v')$ denotes an edge from node $v$ to $v'$, and the edge is labeled with $\iota$, referred to as its label; (c) each node $v \in V$ has label $L(v)$ from $\Gamma$; and (d) each node $v \in V$ carries a tuple $F_A(v) = \langle A_1 = a_1, \ldots, A_n = a_n \rangle$ of attributes of a finite arity, where $A_i \in \Sigma \Gamma$ 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 $A \in \Sigma \Gamma$ and a node $v \in V$, $v.A$ may not exist, except that $v$ has a unique $v.id$ as found in practice.

Graph patterns. A graph pattern is a directed graph $Q[x] = (V_Q, E_Q, L_Q)$, 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) $x$ denotes the nodes in $V_Q$ as a list of variables.

Labels of pattern nodes and edges are drawn from $\Gamma$. Moreover, we allow wildcard ‘.’ as a special label in $Q$.

A pattern $Q_i[y]$ is a copy of $Q_i[x]$ via a bijection $f$: $x \mapsto y$ if $Q_i[y]$ is $Q_i[x]$ with variables renamed by $f$. More specifically, let $Q_i[x] = (V_Q, E_Q, L_Q)$ and $Q_i[y] = (V_Q[y], E_Q, L_Q[y])$. Then (a) $x$ and $y$ are disjoint, and (b) $f$ is an isomorphism from $Q_i$ to $Q_i$; i.e., for each $x \in x$, $L_Q(x) = L_Q(f(x))$; and $(x_1, x_2) is an edge in $E_Q[y]$ if and only if $(f(x_1), f(x_2)) is in $E_Q[x]$ for the same label $\iota$.

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 $Q_3[x, y]$; (2) $Q_4[x, y]$ 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 $Q_5[x, x']$ with variables $x$ and $x'$, specifying a relationship between an album entity $x$ and an artist entity $x'$; and (b) a copy $Q_7[y, y']$ of $Q_6[x, x']$ with variables renamed; similarly for $Q_7[x, y]$.

Matches. We say that a label $\iota$ matches $\iota'$, denoted by $\iota \triangleright \iota'$, if either (a) $\iota$ and $\iota'$ are in $\Gamma$ and $\iota = \iota'$, or (b) $\iota'$ is a constant $c$.

A match of pattern $Q[x]$ in graph $G$ is a homomorphism $h$ from $Q$ to $G$ such that (a) for each node $u \in V_Q$, $L_Q(u) \triangleright L(h(u))$; and (b) for each edge $e = (u, \iota, v')$ in $Q$, there exists an edge $e' = (h(u), h(\iota), h(v'))$ in $G$ such that $\iota \triangleright \iota'$. Note that when $\iota$ is ‘.’, there may exist multiple edges $e'$ with $\iota \triangleright \iota'$.

The match picks one of them, and denotes it by $h(\iota')$.

Abusing the notations, we also denote the match as a vector $h(x)$ 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 $\varphi$ is defined as $Q[x] \rightarrow Q[y]$ where $Q[x]$ is a graph pattern, and $X$ and $Y$ are two (possibly empty) sets of literals of $\bar{x}$; we refer to $Q[x]$ and $X \rightarrow Y$ as the pattern and FD of $\varphi$, respectively.

(a) constant literal $x.A = c$, where $c$ is a constant in $U$, and $A$ is an attribute in $\Sigma \Gamma$ that is not id; or

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

Figure 1: Graph patterns
Intuitively, GED $\varphi$ is a combination of (1) a topological constraint imposed by pattern $Q$, to identify entities in a graph, and (2) an FD $X \rightarrow Y$, 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 CFs [21]. An id literal $x.id = y.id$ 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 $\varphi_1 = Q_1[x,y](X_1 \rightarrow Y_1)$. Here $X_1$ consists of a single constant literal $x.type = \text{"video game"}$, $Y_1$ consists of a literal $y.type = \text{"programmer"}$, and $y$ 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 $\varphi_2 = Q_2[x,y,z](\emptyset \rightarrow y.name = z.name)$. 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 $\varphi_3 = Q_3[x,y](x.A = x.A \rightarrow y.A = x.A)$, 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 `$\cdot$', representing generic entities regardless of labels.

(4) GED $\varphi_4 = Q_4[x,y](\emptyset \rightarrow false)$, 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 $\varphi_1$-$\varphi_3$ catch the errors described in Example 1, e.g., $\varphi_3$ can detect the inconsistency between birds and moa.

(5) GED $\varphi_5 = Q_5[x,x',z_1,z_2,y_1,\ldots,y_k](X_5 \rightarrow Y_5)$ specifies the rule of Example 1 for catching spam. Here $X_5$ consists of three constant literals $x' is.real = 1$, $z_1.keyword = c$ and $z_2.keyword = c$, $y_5 is.x.is.fake = 1$, 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: $\psi_1 = Q_6[x,x',y,y'](X_6 \rightarrow x.id = y.id)$, $\psi_2 = Q_7[x,y](X_7 \rightarrow x.id = y.id)$.

For artist: $\psi_3 = Q_8[x,x',y,y'](X_8 \rightarrow x'.id = y.id)$.

Here $X_6$ consists of $x.title = y.title$ and id literal $x'.id = y.id$; $X_7$ includes $x.title = y.title$ and x.id = y.id; and $X_8$ consists of $x.name = y.name$ and id literal $x.id = y.id$. 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 ($\psi_1$), or their title and release attributes ($\psi_2$). Similarly, to identify artist entities $x'$ and $y'$ as required by $\psi_1$, we need to check the ids of a pair of albums they recorded ($\psi_2$) in turn.

Semantics. To interpret GED $\varphi = Q_6[x](X \rightarrow Y)$, we use the following notations. Consider a match $h(x)$ of $Q$ in a graph $G$, and a literal $l$ of $x$. We say that $h(x)$ satisfies $l$, denoted by $h(x) \models l$, if (a) when $l$ is $x.A = c$, then attribute $A$ exists at node $v = h(x)$, and $A = c$; (b) when $l$ is $x.A = y.B$, then attributes $A$ and $B$ exist at $v = h(x)$ and $v' = h(y)$, respectively, and $A = A'$. (c) when $l$ is $x.id = y.id$, then $h(x)$ and $h(y)$ refer to the same node; hence, they have the same set of attributes and edges.

We denote by $h(x) \models X$ if the match $h(x)$ satisfies all the literals in $X$; in particular, if $X = \emptyset$, then $h(x) \models X$ for any match $h(x)$ of $Q$ in $G$; similarly for $h(x) \models Y$. We write $h(x) \models X \rightarrow Y$ if $h(x) \models X$ implies $h(y) \models Y$.

A graph $G$ satisfies GED $\varphi$, denoted by $G \models \varphi$, if for all matches $h(x)$ of $Q$ in $G$, $h(x) \models X \rightarrow Y$.

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

Given the semantics, we also write $Q[\bar{x}](X \rightarrow Y)$ as $Q[\bar{x}](\lambda \bar{u} \in \bar{x} \rightarrow \lambda \bar{u}' \in \bar{Y}'
$. 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 $h(x)$ trivially satisfies $X \rightarrow Y$ by the definition of $h(x) \models X$. (b) In contrast, if $x.A = c$ is in $Y$, then for $h(x) \models Y$, $h(x)$ must have $A$-attribute by the definition; similarly for $x.A = y.B$.

As a consequence, we can use, e.g., $Q[\bar{x}](\emptyset \rightarrow A \cdot x.A = x.A)$ to enforce that all entities $x$ of “type” $\tau$ must have an $A$ attribute, where $Q$ consists of a single vertex $x$ labeled $\tau$. 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. GEDS of [23] are syntactically defined as GEDs without id literals, i.e., $Q[\bar{x}](X \rightarrow Y)$ in which neither $X$ nor $Y$ contains $x.id = y.id$. 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, $\varphi_1 \models \varphi_3$ in Example 3 are GFDS, but $\psi_1$,$\psi_3$ are not.

(2) Keys. A key $\psi$ of [19] is defined as $Q[\bar{x}, x_0]$, where $Q[\bar{x}]$ is a graph pattern and $x_0$ is a designated node in $\bar{x}$. A graph $G$ satisfies $\psi$ 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, $h(x_0)$ and $h'(x_0)$ 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 $Q[\bar{x}](X \rightarrow x_0.id = y_0.id)$, where (a) $Q[\bar{x}]$ is composed of patterns $Q_1[\bar{y}]$ and $Q_2[\bar{y}]$, and $Q_1[\bar{y}]$ is a copy of $Q_2[\bar{y}]$ via a bijection $f: \bar{y} \mapsto \bar{y}$ (see Example 2), (b) $\bar{x}$ 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, $\psi_1$, $\psi_2$ and $\psi_3$ of Example 3 are GKeys. GKeys express recursive keys of [19] in terms of id literals.

The key $\psi = Q[\bar{x}, x_0]$ of [19] can be expressed as a GKey $Q'[\bar{x}](X \rightarrow x_0.id = y_0.id)$, 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 GEDs that we adopt the homomorphism semantics for graph pattern matching.
To illustrate this, consider GKey $\psi_3$ 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, $h[\bar{x}] \not\cong X_8$ and hence, $h[\bar{x}]$ trivially satisfies $\psi_3$. 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 satisfiability of a set $\Sigma$ of GEDs (see Section 5.1), where a model of $\Sigma$ requires that every GED in $\Sigma$ finds a match of its pattern, to assure that the GEDs in $\Sigma$ do not conflict with each other. Consider a GKey $\varphi = Q[x, y](\emptyset \rightarrow x.\text{id} = y.\text{id})$, 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 $\varphi$ cannot find a model in any sensible graph.

(3) GEDs. We also study the class of GFDs that include no constant literals, referred to as variable GFDs and denoted by GFDs. For instance, $\varphi_2$ and $\varphi_3$ are GFDs, but $\varphi_1$, $\varphi_3$ and $\varphi_5$ are not. Intuitively, (a) GFDs are an extension of relational CFDs to graphs, (b) while GFDs extend FDS, carrying neither constant literals nor id literals.

Similarly, we study GEDs without constant literals, referred to as variable GEDs and denoted by GEDs. Obviously GFDs are a proper sub-class of GEDs; e.g., $\psi_1$-$\psi_3$ of Example 3 are GFDs, but they are not GFDs.

(4) Forbidding GEDs. GEDs can express limited negation, in the form of $Q[\bar{x}](X \rightarrow \text{false})$, where $\text{false}$ 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 GEDs (GEDs) in the same setting. An EGD has the form $\forall z(\phi(z) \rightarrow y_1 = y_2)$, where $\phi$ is a conjunction of relation atoms $R(w_1, \ldots, w_j)$ and equality atoms $w_i = w_j$, $w_i$ and $w_j$ are variables in $\bar{x}$, and so are $y_1$ and $y_2$ (cf. [11]). Here each variable $w$ corresponds to an attribute $R_{eq}[A_w]$ in a relation atom of $\phi$. The EGD can be expressed as a pair of GFDs:

(1) $\varphi_R = Q_R[\bar{x}](\emptyset \rightarrow Y_R)$, where (a) $Q_R$ is a pattern such that for each relation atom $R$ in $\phi$, there exists a node $x_R \in \bar{x}$ in $Q_R$ that is labeled with $R$; and $Q_R$ has no edges; and (b) $Y_R$ consists of $x_RA_R = x_RA_R$ for each variable $x \in \bar{x}$, which indicates attribute $R_{eq}[A_w]$; intuitively, $\varphi_R$ ensures that the relations in $\phi$ 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 $\phi$, which corresponds to $R_i[A_{w_i}] = R_i[A_{w_j}]$ as remarked above, $X_E$ includes a literal $x_{R_i}R_i.A_{R_i} = x_{R_i}R_i.A_{R_i}$; and (b) $Y_E$ is $x_{R_i}R_i.A_{R_i} = x_{R_j}R_j.A_{R_j}$, which corresponds to $y_1 = y_2$. This enforces that $\phi$ 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 $\Sigma$ of GEDs. We study the chase of $G$ by $\Sigma$, to (a) check the satisfiability of $\Sigma$ (resp. implication of GED $\varphi$ by $\Sigma$) when $G$ encodes the patterns of $\Sigma$ (resp. $\varphi$; see Section 5), (b) optimize graph pattern queries $Q$ with $\Sigma$ when $G$ represents $Q$, and (c) identify entities and catch errors by using $\Sigma$ 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 $[x]_{Eq}$, is a set of nodes $y \in V$ that are identified as $x$. For each attribute $x.A$ of $x$, its equivalence class $[x.A]_{Eq}$ is a set of attributes $y.B$ and constants $c$, if $x.A = y.B$ and $x.A = c$ are enforced by GEDs in $\Sigma$ (see below), respectively. The relation is reflexive, symmetric and transitive, such that

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

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

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

(d) if node $y \in [x]_{Eq}$, then for each attribute $y.B$ of $y$, $[x.B]_{Eq} = [y.B]_{Eq}$; 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 $y \in [x]_{Eq}$ such that $L(x) \neq L(y)$ and $L(y) \neq L(x)$ (label conflict), or

(b) there exists $y.B \in [x.A]_{Eq}$ 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 $\sim$ to compare labels (recall $\sim$ 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 $\sim$ is asymmetric: $x \sim y$ does not mean that $y \sim x$.

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 $G_{Eq} = (V', E', L', F'_A)$ obtained from $G$ as follows: for each node $x \in V'$, (a) $x_{Eq}$ is a node in $V'$, denoting $[x]_{Eq}$; (b) for each edge $(x, y) \in E$, $(x_{Eq}, y_{Eq})$ is an edge in $E'$; similarly for each edge $(y, t, x) \in E$; (c) $L'[x_{Eq}] = \sim'$; if all nodes in $[x]_{Eq}$ are labeled ‘.‘; otherwise $L'[x_{Eq}] = L(x)$, where $z \in [x]_{Eq}$ with $L(z) \neq \sim'$; and (d) $F'_A(x_{Eq}) = \bigcup_{y \in [x]_{Eq}} F_A(y)$, the union of the attributes of all the nodes in $[x]_{Eq}$.
When \( \mathbf{Eq} \) is consistent, \( G_{\mathbf{Eq}} \) is well defined. In particular, when \( x \) and \( y \) are identified as the same node, \( F_A'([x]_{\mathbf{Eq}}) \) merges the attributes of \( x \) and \( y \), and \( \text{e} \) 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 \( z_1, z_2 \in [x]_{\mathbf{Eq}} \), if \( L(z_1) \neq \cdot' \) and \( L(z_2) \neq \cdot' \), then \( L(z_1) = L(z_2) \).

When \( \mathbf{Eq} \) is inconsistent, \( G_{\mathbf{Eq}} \) is undefined.

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

We define a chase step of \( G \) by \( \mathbf{Eq}_i \) as

\[
\mathbf{Eq}_i \Rightarrow (\varphi, h) \mathbf{Eq}'
\]

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

1. If \( l = x.A = c \) and \( c \notin h(\cdot) \cdot.A \cdot \cdot \cdot \cdot \cdot \), then \( \mathbf{Eq}' \) extends \( \mathbf{Eq} \) by adding \( \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \·
at most \( |E| \leq 4 \cdot |G| \cdot |\Sigma| \). Based on the bound, one can readily verify that the length of \( \rho \) 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 \( G_{\text{Eq}} \models \Sigma \) by the definition of terminal chasing sequences and the Church-Rosser property.

\[ \Box \]

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 \( \Sigma \) of GEDs. A model of \( \Sigma \) is a graph \( G \) such that (a) \( G \models \Sigma \), and (b) for each \( \phi \in \Sigma \), \( Q \phi(X \rightarrow Y) \) in \( \Sigma \), \( Q \) has a match in \( G \).

Intuitively, if \( \Sigma \) has a model, then the GEDs in \( \Sigma \) 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 \( \Sigma \) of GEDs.
- Question: Does there exist a model of \( \Sigma \)?

We say that \( \Sigma \) 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 \( \Sigma_1 \) consisting of

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

where \( Q_1 \) and \( Q_2 \) are depicted in Fig. 3. One can verify that each of \( \phi_1 \) and \( \phi_2 \) has a model when they are taken separately; however, \( \Sigma_1 \) does not have a model. To see this, consider a homomorphism \( f \) from \( Q_2 \) to \( Q_1 \), mapping \( x_1 \) to \( x \), and \( x_2 \) to \( x \), \( y_1 \) to \( y \), 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, \( \phi_1 \) and \( \phi_2 \) 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 \( \Sigma_2 \) consisting of \( \phi_1 \) and \( \phi_2' = Q_2'[\emptyset]([x_1.A = x_1.B]) \), 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, \( \Sigma_2 \) is not satisfiable. To see this, suppose by contradiction that \( \Sigma_2 \) has a model \( G \), in which \( Q_2' \) has a match \( h_2' \). 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.\[ \Box \]

Characterization. We develop a sufficient and necessary condition to characterize the satisfiability of a set \( \Sigma \) of GEDs.

Consider a set \( \Sigma \) of GEDs \( Q_i[x_i](X_i \rightarrow Y_i) \) for \( i \in [1,n] \), where \( Q_i \equiv (V_i,E_i,L_i) \), 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 \( G_S \) of \( \Sigma \) is defined to be a graph \( (V_S,E_S,L_S,F_S) \), where \( V_S \) is the union of \( V_i \)'s, and similarly for \( E_S \) and \( L_S \); but \( F_S \) is empty.

Intuitively, \( G_S \) is the union of all graph patterns in \( \Sigma \), in which patterns from different GEDs are disjoint.

We chase the pattern graph \( G_S \) with \( \Sigma \), and characterize the satisfiability of \( \Sigma \) based on the chase (Section 4).

Theorem 2: A set \( \Sigma \) of GEDs is satisfiable if and only if \( \text{chase}(G_S, \Sigma) \) is consistent.\[ \Box \]

Example 6: Recall the set \( \Sigma_1 \) of GEDs from Example 5. Its canonical graph is the union \( G_{\Sigma_1} \) of \( Q_1 \) and \( Q_2 \) shown in Fig. 3. One can verify that \( \text{chase}(G_{\Sigma_1}, \Sigma_1) \) is inconsistent, i.e., there exists a terminal chasing sequence of \( G_{\Sigma_1} \) by \( \Sigma_1 \) with result \( \bot \). This confirms the observation of Example 5 that \( \Sigma_1 \) is not satisfiable; similarly for \( \Sigma_2 \).

Proof: (a) If \( \text{chase}(G_S, \Sigma) \) is consistent, i.e., there exists a valid terminal chasing sequence \( \rho \) of \( G_S \) by \( \Sigma \), we show that one can build a model of \( \Sigma \) from \( G_S \) based on \( \rho \), using Theorem 1. We take special care to handle \( \bot \) in \( \Sigma \).

(b) If \( \Sigma \) is satisfiable, i.e., \( \Sigma \) has a model \( G \), we show that each terminal chasing sequence \( \rho \) of \( G_S \) by \( \Sigma \) is valid. For a pattern \( Q \), an equivalence relation \( \text{Eq} \) on the nodes and attributes of \( G \) and a match \( h \) of \( Q \) in \( G \), we represent \( \text{Eq} \) as a set of equality literals, and write \( h \models \text{Eq} \) if \( h \models \) for all literals \( i \) in \( \text{Eq} \). We construct a match \( h \) of \( G_S \) in \( G \) by treating \( G_S \) as a graph pattern, and show that \( h \models \text{Eq} \) for each chase step \( \text{Eq}_i \Rightarrow \text{Eq}_{i+1} \) of \( \rho \).\[ \Box \]

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

Theorem 3: The satisfiability problem is

- \( \text{coNP}\)-complete for GEDs, GFDs, GKeys, GEDs;
- \( \text{it is in } O(1) \text{ time for GFDs.} \)

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 GEDs. As will be seen in the proof, the problem is \( \text{coNP}\)-hard even when \( \Sigma \) consists of a fixed number of GEDs.

(2) In the absence of constant and id literals, the problem is trivial: any set of GEDs find a model.

For relational EGDs, the satisfiability problem is not an issue. The satisfiability problem for relational CFDs is \( \text{NP}\)-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 $\Sigma$ 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 $\Sigma$ consists of GFDs, $\text{chase}(G, \Sigma)$ 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 GEDs and GEDs. We prove (a) and (b) by (different) reductions from the invariant of the 3-colorability problem. The 3-colorability problem is to decide, given an undirected graph $G$, whether there exists a proper 3-coloring of $G$ such that for each edge $(u, v)$ in $G$, $v(u) \neq v(v)$. The problem is NP-complete even when $|G|$ is connected [25].

The proof for (a) uses two GFDs of the form $Q(x)(0 \rightarrow Y)$, 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 $Q(x)(0 \rightarrow x.id = y.id)$ without constant literals.

5.2 The Implication

A set $\Sigma$ of GEDs implies another GED $\varphi$, 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 $\Sigma$ of GEDs and another GED $\varphi$.
- 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 $\varphi = Q(x)(X \rightarrow Y)$, where pattern $Q$ = $(V, Eq, Lq)$. We use the following notations.

1. The canonical graph of $Q$ is $G_Q = (V_Q, Eq, Lq, F_A)$, where $F_A$ is empty, along the same lines as $G_S$.
2. We use $Eq_X$ to denote the equivalence relation of $X$ in $G_Q$, such that for any literal $l$ in $X$, $x \in [u]_{Eq_X}$ if $l$ is $u = v$, denoting $x.A = c, x.A = y.B$ or $x.id = y.id$. Moreover, $Eq_X$ contains $[x]_{Eq_X} = \{x\}$ for each $x \in V_Q$.
3. We use $\text{chase}(G_Q, Eq_X, \Sigma)$ to denote the result of the chase of $G_Q$ by $\Sigma$ starting with $Eq_X$. Notice that it is inconsistent if $Eq_X$ is inconsistent (see Section 4).
4. We say that a literal $l$ can be deduced from an equivalence relation $Eq$ if $v \in [l]_{Eq}$, 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 $\Sigma$ of GEDs and $\varphi = Q(x)(X \rightarrow Y)$, $\Sigma \models \varphi$ if and only if either (1) $\text{chase}(G_Q, Eq_X, \Sigma)$ is inconsistent; or (2) $\text{chase}(G_Q, Eq_X, \Sigma)$ is consistent and $Y$ can be deduced from $\text{chase}(G_Q, Eq_X, \Sigma)$.

Intuitively, if $\text{chase}(G_Q, Eq_X, \Sigma)$ is inconsistent, then for all graphs $G \models \Sigma$ and for all matches $h(x)$ of pattern $Q$ in $G$, $h(x) \neq X$. Condition (1) covers this case. Otherwise, if $\text{chase}(G_Q, Eq_X, \Sigma)$ is consistent, condition (2) ensures that $Y$ is a logical consequence of $\Sigma, Q$ and $X$.

Example 7: Consider a set $\Sigma = \{\{\varphi_1, \varphi_2\}\}$ and $\varphi$.

- $\varphi_1 = Q[x_1, x_2](x_1.A = x_2.A \rightarrow x_1.id = x_2.id)$,
- $\varphi_2 = Q[x_1, x_2](x_1.B = x_2.B \rightarrow x_1.A = x_1.B)$,
- $\varphi = Q[x_1, x_2, x_3, x_4](X \rightarrow Y)$,

where $Q, Q_1$ and $Q_2$ are shown in Fig. 4, $X$ is $x_1.A = x_3.A \land x_2.B = x_4.B$, and $Y$ is $x_1.id = x_3.id \lor x_2.id = x_4.id$. Canonical graph $G_Q$ has the same form as $Q$ of Fig. 4. Then $\text{chase}(G_Q, Eq_X, \Sigma)$ 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 \(\wedge\): $x_3 \in [x_3]_{Eq_X}$ and $x_4 \in [x_4]_{Eq_X}$, where $Eq$ is the result of the chase. This explains why we use $\bowtie$ 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 $Eq_X, Eq_1, \ldots, Eq_k$ of $G_Q$ by $\Sigma$ starting with $Eq_X$, valid or not. We show the following lemmas.

(a) For any graph $G$ and pattern $Q(x)$, if $G \models \Sigma, h(x)$ is a match of $Q$ in $G$ and $h(x) \models X$, then $h(x) \models Eq_k$.
(b) For consistent $\text{chase}(G_Q, Eq_X, \Sigma)$, $Y$ can be deduced from $\text{chase}(G_Q, Eq_X, \Sigma)$ if and only if for any graph $G$ and match $h(x)$ of $Q$ in $G$, $h(x) \models Eq_k$ implies $h(x) \models Y$.

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, GFDs and GEDs.

As opposed to Theorem 3, the implication analysis for GFDs is NP-hard, in the absence of constant and id literals, although $\text{chase}(G_Q, Eq_X, \Sigma)$ is always consistent in this case. This is because to check whether $Y$ can be deduced from $\text{chase}(G_Q, Eq_X, \Sigma)$, we need to examine all possible homomorphic mappings of patterns of $\Sigma$ in $G_Q$. The intractability remains intact even when $\Sigma$ 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 GFDs and GKeys, since GEDs, GFDs and GEDs subsume GFDs. We prove these by (differ-
ent) reductions from the 3-colorability problem, capitalizing on Theorem 4. In the reductions, we use \( \Sigma \) consisting of a single GFD, \( \phi \) (resp. GKey \( \psi \)), where \( \phi \) and \( \psi \) have the form \( Q[\bar{x}](Y \rightarrow Y) \) and \( Y \) consists of variable literals only (resp. \( Q[\bar{x}](\emptyset \rightarrow x.id = y.id) \) for GKeys \( \psi \) and \( \phi \)).

5.3 The Validation Problem

The validation problem for GEDs is stated as follows.

- **Input:** A finite set \( \Sigma \) 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, GFDs and GEDs.

As for the implication problem, the validation analysis is intractable even for GFDs, which is an extension of relational FDs that carries neither constant literals nor id literals. The intractability remains intact when \( \Sigma \) consists of a single GFD, 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 GFDs by (different) reductions from the complement of the 3-colorability problem. These suffice since GFDs are a special case of GFDs, GEDs and GEDs.

In the reductions, we use \( \Sigma \) consisting of only a single GFD, \( Q[\bar{x}](X \rightarrow Y) \) (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 later publication.

Nonetheless, there are tractable cases that allow us to make effective use of GEDs. For example, one may consider a set \( \Sigma \) 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 \( A \) 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.

<table>
<thead>
<tr>
<th>Table 2: Axiom system ( A_{GED} ) for GEDs</th>
</tr>
</thead>
</table>
| For a set \( \Sigma \) of GEDs and a GED \( \varphi \), a proof of \( \varphi \) from \( \Sigma \) using inference rules of \( A \) is a sequence of GEDs \( \varphi_1, \ldots, \varphi_n = \varphi \), such that each \( \varphi_i \) either is a GED in \( \Sigma \), or can be deduced from \( \varphi_j \)'s by applying an inference rule (or axiom) in \( A \), for \( j < i \) (see [1] for details about proofs).
| We say that \( \varphi \) is provable from \( \Sigma \) using \( A \), denoted by \( \Sigma \vdash_A \varphi \), if there exists a proof of \( \varphi \) from \( \Sigma \) using \( A \). We write it as \( \Sigma \vdash \varphi \) when \( A \) is clear from the context.
| For all GED sets \( \Sigma \) and GEDs \( \varphi \) and:
| * sound if \( \Sigma \vdash_A \varphi \) implies \( \Sigma \models \varphi \);
| * complete if \( \Sigma \models \varphi \) implies \( \Sigma \vdash_A \varphi \)
| for all GED sets \( \Sigma \) and GEDs \( \varphi \); and
| * independent if for any rule \( r \in A \), there exist GEDs \( \Sigma \) and \( \varphi \) such that \( \Sigma \vdash_A \varphi \) but \( \Sigma \vdash_{A \setminus r} \varphi \).
| Here \( \Sigma \setminus r \) denotes \( \Sigma \) excluding \( r \). That is, removing any rule from \( A \) would make it no longer complete. We remark that we focus on finite implication, considering finite graphs.
| We refer to \( A \) as a finite axiom system or a finite axiomatization of GEDs if \( A \) 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 \( A_{GED} \) 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) \( GQ \) the canonical graph of pattern \( Q \) (Section 5.2); (e) \( EQ \) the equivalence relation of a set \( X \) of literals in \( GQ \); and (f) \( (GQ)_eq \) the coercion of \( Eq \) on \( GQ \) (Section 4). The consistency of an equivalence relation \( Eq \) is defined in Section 4. To simplify the presentation, we allow \( c = \infty \). 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, \( A_{GED} \) has six rules for GEDs over graphs.
| **Example 8:** (a) We first prove the following property: if \( \Sigma \vdash \varphi \) and \( \varphi = Q[\bar{x}](X \rightarrow Y) \) and \( Y \subseteq Y \), then \( \Sigma \vdash Q[\bar{x}](X \rightarrow Y) \), where \( Y_i \) 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. \( Q[\bar{x}](X \rightarrow Y) \)
| 2. \( Q[\bar{x}](X \rightarrow (v_i = u_i)) \) (1) and GED4
| 3. \( Q[\bar{x}](X \rightarrow (u_i = v_i)) \) (2) and GED4
| \ldots
| (2n+1) \( Q[\bar{x}](X \rightarrow (u_n = v_n)) \) (2n) and GED3
(2n+2) \( Q[\phi](X \rightarrow (u_1 = v_1)(u_2 = v_2)) \) (3), (5) and GED6

\[\ldots\]

(3n) \( Q[\phi](X \rightarrow Y_1) \) (3n-1), (2n+1) and GED6

It can also be proven for inconsistent \( X \cup Y \). To simplify the presentation, we denote this property as GED7 and apply it in proofs, although GED7 is not in \( \mathcal{A}_{\text{GED}} \).

(b) Recall the augmentation rule of Armstrong’s axioms: if \( X \rightarrow Y \) then \( XZ \rightarrow YZ \). Analogously, consider \( \Sigma \vdash \phi_1 \), where \( \phi_1 = Q[\bar{x}](X \rightarrow Y) \), and GED \( \phi = Q[\bar{x}](XZ \rightarrow YZ) \). We show that \( \Sigma \vdash \phi \) using \( \mathcal{A}_{\text{GED}} \) as follows. First consider the case when \( \text{Eq}_X \cup \text{Eq}_Y \) is consistent:

- (1) \( Q[\bar{x}](X \rightarrow XZ \wedge X_0) \) GED1
- (2) \( Q[\bar{x}](X \rightarrow X) \) GED7
- (3) \( Q[\bar{x}](X \rightarrow Y) \) \( \phi_1 \)
- (4) \( Q[\bar{x}](X \rightarrow XY) \) GED6
- (5) \( Q[\bar{x}](X \rightarrow YZ) \) GED6
- (6) \( Q[\bar{x}](X \rightarrow Y_1) \) GED7

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

- (3) \( Q[\bar{x}](X \rightarrow Y_1) \) GED3

(c) Let \( \Sigma \vdash \phi_2 \) and \( \Sigma \vdash \phi_2 \), where \( \phi_1 = Q[\bar{x}](X \rightarrow Y) \) and \( \phi_2 = Q[\bar{x}](Y \rightarrow Z) \). We show that \( \Sigma \vdash Q[\bar{x}](X \rightarrow Z) \) using \( \mathcal{A}_{\text{GED}} \). When \( \text{Eq}_X \cup \text{Eq}_Y \) is consistent, we have:

- (1) \( Q[\bar{x}](X \rightarrow X \wedge X_0) \) GED1
- (2) \( Q[\bar{x}](X \rightarrow X) \) GED7
- (3) \( Q[\bar{x}](X \rightarrow Y) \) \( \phi_1 \)
- (4) \( Q[\bar{x}](X \rightarrow XY) \) GED6
- (5) \( Q[\bar{x}](X \rightarrow Z) \) GED6
- (6) \( Q[\bar{x}](X \rightarrow Z) \) GED7

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

- (4) \( Q[\bar{x}](X \rightarrow XY) \) GED6
- (5) \( Q[\bar{x}](X \rightarrow Z) \) GED5

These prove the transitivity of Armstrong’s axioms. □

Axiomatization. GEDs are finitely axiomatizable.

**Theorem 7:** The set \( \mathcal{A}_{\text{GED}} \) 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 \( \mathcal{A}_{\text{GED}} \), based on the chase and Theorem 4. Below we illustrate each rule in \( \mathcal{A}_{\text{GED}} \).

(a) GED1 extends the reflexivity of Armstrong’s axioms to cover id literals. Similarly, GED3 and GED4 ensure that equality literals are symmetric and transitive.

(b) GED2 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 \( \text{Eq}_X \cup \text{Eq}_Y \) is inconsistent, then chase\((G_Q,\text{Eq}_X, \Sigma)\) is inconsistent, since \( \text{Eq}_X \) and \( \text{Eq}_Y \) are included in its result. GED3 says that if this happens, then any set \( Y_1 \) of literals of \( \bar{x} \) is a “logical consequence” of the inconsistent \( X, \Sigma \) and \( Q \), following condition (1) of Theorem 4.

(d) When \( \text{Eq}_X \cup \text{Eq}_Y \) is consistent, \( Q_1 \) can be embedded in \( (G_Q)\text{Eq}_X \cup \text{Eq}_Y \) via a match \( h \), and if \( h(\bar{x}_1) \models X_1 \), then one can verify that if chase\((G_Q,\text{Eq}_X, \Sigma)\) is consistent, then \( h(Y_1) \) can be deduced from chase\((G_Q,\text{Eq}_X, \Sigma)\). Hence GED6 follows from condition (2) of Theorem 4.

Observe that GED2 and GED6 are unique for graph dependencies, which are needed to handle id-based entity identification and embedding of graph patterns, respectively.

(2) **Completeness.** Assume that \( \Sigma \models Q[\bar{x}](X \rightarrow Y) \). To prove that \( \Sigma \models Q[\bar{x}](X \rightarrow Y) \), for a terminal chasing sequence \( \rho \) of \( G_Q \) by \( \Sigma \), where \( \rho = \text{Eq}_1 \models \text{Eq}_2 \models \ldots \models \text{Eq}_k \), we treat \( \text{Eq}_k \) as a set of equality literals. Then we show the following claims by induction on the length of \( \rho \).

**Claim 1:** For each \( 1 \leq i \leq k \), \( \Sigma \models Q[\bar{x}](X \rightarrow \text{Eq}_i) \).

**Claim 2:** If there exist GED \( \phi \in \Sigma \) and match \( h \) of the pattern of \( \phi \) such that \( \text{Eq}_k \models (\phi, h) \) \( \text{Eq}_{k+1} \) and \( \text{Eq}_{k+1} \) is inconsistent in \( G_{\text{Eq}_k} \), then \( \Sigma \models Q[\bar{x}](X \rightarrow \text{Eq}_{k+1}) \).

We can verify that \( \Sigma \models Q[\bar{x}](X \rightarrow Y) \) using the claims as follows. By Theorem 4, if \( \Sigma \models Q[\bar{x}](X \rightarrow Y) \), then we need to consider two cases: (a) \( \text{chase}(G_Q,\text{Eq}_X, \Sigma) \) is inconsistent; and otherwise, (b) \( Y \) can be deduced from \( \text{chase}(G_Q,\text{Eq}_X, \Sigma) \). In case (a), Claim 2 and GED3 put together can derive \( \Sigma \models Q[\bar{x}](X \rightarrow Y) \). In case (b), we can show that \( \Sigma \models Q[\bar{x}](X \rightarrow Y) \) following Claim 1.

(3) **Independence.** For each rule GED \( k \) in \( \mathcal{A}_{\text{GED}} \), we show that there exist \( \Sigma \) of GEDs and another GED \( \phi \), such that the proof of \( \Sigma \vdash \phi \) necessarily uses GED\( k \).

Take GED3 as an example. Consider \( \Sigma = \emptyset \) and \( \phi = Q_3[x](x.A = 1) \land (x.A = 2) \rightarrow x.A = 3 \), where \( Q_3 \) consists of a single node \( x \). We show by contradiction that without using GED3, we cannot prove \( \Sigma \models \phi \). Indeed, no other rule allows us to deduce \( Q[\bar{x}](X \rightarrow Y) \) when \( Y \) contains a constant that appears in neither \( X \) nor \( \Sigma \).

□

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 \( \phi \) is defined as \( Q[\bar{x}](X \rightarrow Y) \), where \( Q \) is a pattern, and \( X \) and \( Y \) are sets of literals of one of the following forms: (a) \( x.A \in c \), (b) \( x.A \in y.B \), for constant \( c \in U \), and non-id attributes \( A, B \in \Gamma \), and (c) \( x.id = y.id \); 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” \( \tau \) to have an attribute with a finite domain, e.g., Boolean, as follows:

\( \phi_1: Q_\tau[x](\emptyset \rightarrow x.A = x.A) \),
\( \phi_2: Q_\tau[x](x.A \neq 0 \land x.A \neq 1 \rightarrow \text{false}) \).

Here \( Q_\tau \) consists of a single node labeled \( \tau \); \( \phi_1 \) is a GED that enforces each \( \tau \)-node \( x \) to have an \( A \)-attribute, and \( \phi_2 \) 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 \( \Sigma^p_2 \)-complete, \( \Pi^p_2 \)-complete and coNP-complete, respectively.

The lower bounds of these problems remain intact when \( \Sigma \) 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 \( \Sigma \) 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 \( \Sigma^p_2 \) 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 \( \Sigma^p_2 \)-complete when \( G \) is a complete graph and \( F \) contains no self cycles [37].

The reduction is a little complicated. We use a set \( \Sigma \) of three 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 \( Q[x](X \rightarrow \text{false}) \).

(2) For implication, we also show a small model property: if \( \Sigma \models \varphi \), 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 \( G_h \models \varphi \). Based on the property, we give an \( \Sigma^p_2 \) algorithm to check \( \Sigma \models \varphi \). The lower bound is verified by reduction from the complement of GGCP, using a \( \Sigma \) 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. GEDs. A GED \( \psi \) with disjunction, denoted by GED*, has the same syntactic form \( Q[x](X \rightarrow Y) \) as GEDs, but \( Y \) is interpreted as the disjunction of its literals. That is, for a match \( h[x] \) of \( Q \) in a graph \( G \), \( h[x] \models Y \) if there exists a literal \( l \in Y \) such that \( h(x) \models l \). Hence we also write \( \psi \) as \( Q[x](\bigwedge_{l \in X} l \rightarrow \bigvee_{l' \in Y} l') \).

The other notions such as satisfiability and implication remain the same as their GED counterparts. GED's subsume GEDs. Each GED \( Q[x](X \rightarrow Y) \) can be expressed as a set of \( Q[x](X \rightarrow l) \) of GED's, one for each \( l \in Y \). In contrast, some GED'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*:

\[ \psi: Q[x](0 \rightarrow x.A = 0 \lor x.A = 1) \]

It specifies a domain constraint: each \( \tau \)-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 \( \Sigma \) consists of a fixed number of GED’s with constant and variable literals only.

**Theorem 9:** The satisfiability, implication and validation problems for GED's are \( \Sigma^p_2 \)-complete, \( \Pi^p_2 \)-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 \( \Sigma \) consisting of three GED’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 \( \Pi^p_2 \)-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 example, 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 $A_{GED}$ differs from [2, 16, 29, 30] in the following. (a) Besides value-based reasoning, $A_{GED}$ 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) $A_{GED}$ 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 of 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, GFDs and GEDs, 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


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.