Normalizing Graph Data

Author Details Omitted Due to Double-blind Process

ABSTRACT

Normalization aims at minimizing sources of potential data inconsistency and costs of update maintenance incurred by data redundancy. For relational databases, different classes of dependencies cause data redundancy and have resulted in proposals such as Third, Boyce-Codd, Fourth and Fifth Normal Form. Features of more advanced data models make it challenging to extend achievements from the relational model to missing, non-atomic, or uncertain data. We initiate research on the normalization of graph data, starting with a class of functional dependencies tailored to property graphs. We show that this class captures important semantics of applications, constitutes a rich source of data redundancy, its implication problem can be decided in linear time, and facilitates the normalization of property graphs flexibly tailored to their labels and properties that are targeted by applications. We normalize property graphs into Boyce-Codd Normal Form without loss of data and dependencies whenever possible for the target labels and properties, but guarantee Third Normal Form in general. Experiments on real-world property graphs quantify and qualify various benefits of graph normalization: 1) Removing redundant property values as sources of inconsistent data, 2) detecting inconsistency as violation of functional dependencies, 3) reducing update overheads by orders of magnitude, and 4) significant speed ups of aggregate queries.

ACM Reference Format:
Author Details Omitted Due to Double-blind Process. 2023. Normalizing Graph Data. In *Proceedings of the 2023 International Conference on Management of Data (SIGMOD '23), June 18–23, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3448016.3459238

1 INTRODUCTION

Normalization minimizes sources of potential data inconsistency and costs of integrity maintenance incurred by updates of redundant data. Based on classes of data dependencies that cause redundancy, classical normalization transforms schemata into normal forms where these dependencies can be enforced by keys only, or come close to it. For example, this is achieved by Boyce-Codd Normal Form for functional dependencies (FDs) [8], Fourth Normal Form for multivalued dependencies [11], Fifth Normal Form for join dependencies [31], Inclusion Dependency Normal Form for functional and inclusion dependencies [21], and Domain-Key Normal Form [12] more generally. Third Normal Form minimizes sources of data redundancy under the additional target of enforcing all FDs without joining relation schemata [5, 19], and Bounded Cardinality Normal Form minimizes the level of data redundancy caused by FDs [23]. Some achievements carry forward to richer data formats, including SQL [18] and models with missing data [20, 33], Nested [26, 32], Object-Oriented [30], Temporal [17], Web [1, 25], and Uncertain Databases [22].

Graph databases experience new popularity due to more mature technology in response to the demand of applications for finding relationships within large amounts of heterogeneous data, such as social network analysis, outlier and fraud detection. Graphs can represent data intuitively and efficiently. Both research and industry have brought forward sophisticated technologies with mature capabilities for processing graph data. Recently, classical classes of data dependencies, such as keys and FDs, have been extended to graph databases, and have been put to use for data cleaning and fraud detection tasks [13]. Indeed, Fan [13] remarks that graph dependencies provide a rare opportunity to capture the semantics of application domains on graph databases, which are schema-less. Interestingly, however, the normalization of graph data has neither been mentioned in the literature nor has it been subject of investigation yet. This is surprising since it is a very natural question to ask what normalization of graph data may actually mean. Indeed, any mature data model needs to facilitate principles of data integrity. Since a strong use case of graph data is analytics, the quality of analysis depends fundamentally on the quality of graph data. This firmly underpins the need to understand sources of data inconsistency and other data quality issues. This includes the challenge of understanding opportunities for more efficient integrity maintenance and query processing, and database design principles within schema-less graph environments. In relational databases, constraints restrict instances of a given schema to those considered meaningful for the underlying application. Normalization restructures the schema and constraints such that data redundancy is minimized and constraint management made more efficient. When attempting to normalize graph data, we do not have a schema and will therefore need to rely on the constraints exclusively. In particular, it means that a normalized set of constraints would not admit any graph with redundant data value occurrences, but without any restriction of such a graph's structure by any schema. This sounds intriguing and the flexibility of graph data may promote restructuring only that part of the graph required by an application. Our contributions towards the aim of initiating research on graph data normalization can be summarized as follows:

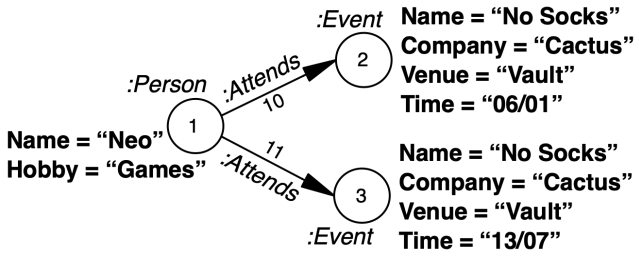
(1) We introduce uniqueness constraints and functional dependencies as declarative means to i) express completeness, integrity, and uniqueness requirements in the form of business rules that govern graph data, and ii) form the source of redundant property values that drive graph normalization goals.

(2) We show that our graph dependencies facilitate normalization since their implication problem can be captured axiomatically by a finite set of Horn rules, and algorithmically by a linear-time decision algorithm.

SIGMOD '23, June 18–23, 2023, Seattle, WA, USA

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3459238

Figure 1: Original Property Graph



(3) We normalize graph data into lossless, dependency-preserving Boyce-Codd Normal Form whenever possible, and guarantee Third Normal Form in general. The normalization is tailored to nodes with those labels and properties that the target application requires. In contrast to relational databases, our normalization can even be applied when FDs do not fully hold. Indeed, our normalization is still lossless and removes redundancy in part of the data that conforms to the FDs.
(4) We demonstrate the extent and benefits of graph normalization experimentally. For some popular real-world property graphs we identify meaningful FDs and quantify how many redundant property values they cause. We provide examples of inconsistencies found as violations of meaningful FDs. We demonstrate that integrity maintenance and aggregate query evaluation improves by orders of magnitude on normalized property graphs. While not achieving the full scale of speed up that graph normalization accomplishes, we show that indexing the left-hand side properties of FDs still gains significant efficiency in integrity maintenance and aggregate query processing without any changes to the graphs.

Our results motivate a long line of future investigation into graph data normalization, including the proposal of normal forms, the study of more expressive graph dependencies, relationships to the conceptual and physical design of graph data, and the design for quality of graph data.

In what follows, we motivate our work with an application scenario in Section 2. Section 3 includes a concise review of relevant work. The formal semantics of property graphs and constraints is given in Section 4. In Section 5, we introduce our normalization framework, including axiomatic and algorithmic characterizations of the implication problem, normal forms and normalization. Experimental results that qualify and quantify the need and benefits of normalization are discussed in Section 6. We conclude and briefly comment on future work in Section 7. Proofs can be found in Section A. Our GitHub repository has details about experiments[1].

2 ILLUSTRATIVE EXAMPLE

In this section we use a minimal example to illustrate ideas and concepts as a motivation for our work in this article.

Our example models an application scenario where people attend events. The model of choice is a property graph, as illustrated in Figure 1. Here, nodes and edges carry any number of labels, and may be associated with pairs of properties and values. In Figure 1 we have a node labelled by *Person* and other nodes labelled by *Event*. We also have edges from Person to Event nodes labelled by *Attends*, expressing that a person attends an event. Event nodes exhibit properties such as *N(ame)*, *C(ompany)*, *V(enue)* and *T(ime)*, expressing that a company (like "Cactus") is in charge of an event with a name (like "No Socks") held at a venue (like "Vault") and time (like "06/01").

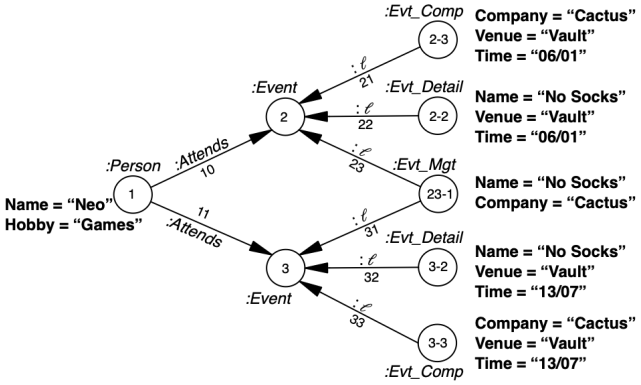
Event data is subject to several business rules expressed by uniqueness constraints (UCs) and FDs: Event nodes with properties C and T can be uniquely identified by the value combination on these two properties, $N \rightarrow C$ (events are managed by at most one company), $NT \rightarrow V$ (the time of events uniquely determines its venue), and $TV \rightarrow N$ (at any time any venue can host at most one event). Intuitively, FDs express that nodes with matching values on all properties of the left-hand side have also matching values on all properties of the right-hand side. However, important features apply to such FDs that result from characteristics of property graphs. Firstly, some nodes may not feature the properties of an FD. In that case, we take the principled approach that missing properties should not have an impact on the validity of constraints. Secondly, since properties may not exist on some nodes, we provide data stewards with the ability to declare requirements on properties as well. In our example, we only stipulate our FDs on nodes that carry the properties. We express the combination of such completeness and integrity requirements as the graph-tailored FDs (gFDs) $NC{:}N \rightarrow C$, $NTV{:}NT \rightarrow V$, and $NTV{:}TV \rightarrow N$. Note that completeness requirements may involve properties that do not occur in some FD at all. For example, we may even stipulate the gFD $NCT{:}N \rightarrow C$. Here, $NC{:}N \rightarrow C$ implies $NCT{:}N \rightarrow C$, but not vice versa since we may have different nodes with matching values on N, non-matching values on C but without property T. Thirdly, gFDs naturally apply to some nodes but not others. In our example, the gFDs shall hold on nodes with label *Event*, but not necessarily on nodes without this label. We express this requirement as the gFD $\sigma = \mathit{Event}{:}NC{:}N \rightarrow C$. Naturally, we want to provide the freedom to tailor the declaration of gFDs to nodes with multiple labels, since some rules may apply to such nodes but not to nodes that do not carry some of those labels. Similarly, we obtain the graph-tailored uniqueness constraint (gUC) $\mathit{Event}{:}CT{:}CT$, stipulating that all Event nodes that carry the properties C and T are uniquely identified by the combination of values on these properties. While a gUC $L{:}P{:}X$ always implies the gFD $L{:}P{:}X \rightarrow P$, the gFD does not imply the gUC. For instance, a graph with two different Event nodes that have matching values on both C and T will satisfy the gFD $\mathit{Event}{:}CT{:}C \rightarrow T$ but violate the gUC $\mathit{Event}{:}CT{:}CT$.

Note that gFD σ causes redundant property value occurrences in the graph of Figure 1. Indeed, each occurrence of company *Cactus* is redundant due to the gFD σ: If one occurrence of *Cactus* is changed to *any* different value, then gFD σ will be violated. Firstly, the redundancy is source for potential data inconsistency since updates of companies may not be propagated to all redundant values. Secondly, the redundancy is also a source for potential inefficiency of update operations, since all redundant occurrences need to be identified and updated consistently. Thirdly, the redundancy may be an inhibitor to more efficient techniques for query evaluation.

As a solution we may choose a different design for the scenario, in which we decompose the properties on Event nodes. Figure 2 shows

[1] https://github.com/GraphDatabaseExperiments/normalization_experiments

Figure 2: Normalized Property Graph



a normalized property graph G_n resulting from a non-obvious decomposition. Indeed, G_n is in Boyce-Codd Normal Form tailored to the application requirements to decompose Event nodes that exhibit **all** properties N, C, V, and T. Note that different application requirements would result in different decompositions. The decomposition is lossless, since a simple join via the ℓ-labelled edges would restore the original graph G. Similarly, the decomposition was dependency-preserving: the gUC *Event*:*CT*:*CT* has added the gUC *Evt_Comp*:*CVT*:*CT*, and the gFDs $NC{:}N \rightarrow C$, $NVT{:}NT \rightarrow V$, $NTV{:}TV \rightarrow N$ have resulted in the gUCs *Evt_Mgt*:*NC*:*N*, *Evt_Detail*:*NTV*:*NT*, and *Evt_Comp*:*NTV*:*TV*. The additional properties in these constraints, such as V in *Evt_Comp*:*CVT*:*CT*, originate from the application requirement for Event nodes that exhibit all properties N, C, V, and T. The new design eliminates all data redundancy caused by gFD σ, for example in G. The fact that company *Cactus* manages the event with name *No Socks* is only represented once (on the new node $23-1$), avoiding data inconsistency, making update and query operations potentially more efficient. This is achieved by the new gUC $\sigma' = Evt\_Mgt{:}NC{:}N$, expressing that there cannot be two different *Evt_Mgt* nodes with properties N and C that have matching values on N.

Normalizing a property graph will intuitively minimize sources of potential inconsistency, and bring forward more efficient updates of companies for events, and of aggregate queries that require the numbers of events that a company manages. Intuitively, the benefits grow as the graph size grows. Hence, identifying opportunities and limits of graph databases in handling normalization is an interesting line of research with huge potential.

3 PREVIOUS WORK

Schema design is a classical topic of database research [24]. The introduction has already mentioned normal forms that result from different classes of data dependencies, and some of their extensions to other data models. The proposals achieve syntactic characterizations of well-designed databases in the sense that only instances are permitted that do not feature any redundant data value occurrences caused by any dependency in the class considered. A cornerstone of normal forms is an efficient characterization of the implication problem associated with the class of data dependencies. Foremost, Boyce-Codd and Third Normal Form are founded on Armstrong's axioms, denoted by $\mathfrak{A}$, for the implication of FDs [2] and its algorithmic characterization by computing attribute set closures in linear time [4]. In fact, this algorithm drives decompositions into these normal forms [6]. While extensions to many richer data models exist, normalization has not been a topic yet for graph data.

In addition, the design for other data quality dimensions has only recently begun to draw attention [3]. In fact, completeness-tailored UCs and FDs were introduced for relations with missing values, and a normalization framework established that tailors classical schema design to the completeness and integrity dimensions of data quality [33]. That approach to schema design has not been investigated in other data models, such as graph databases.

In a different line of research, the database community has spent increasing resources on graph data models. While strong emphasis has been given to query languages, several lines of work on integrity management have emerged recently. The work in [10] proposes a very general notion of keys on property graphs without providing results on their properties or applications. Graph-tailored uniqueness constraints (gUCs) have also been proposed for property graphs to address the shortcoming of concepts available in graph database systems [29], including the lack of constraints that address different data quality dimensions such as data completeness and integrity. However, graph-tailored functional dependencies have not been considered yet and neither has normalization as a consequence. The work in [13, 15] defines expressive graph dependencies more generally, presents hardness results for static reasoning problems and highlights applications in entity resolution and fraud detection. Even more expressive are graph generating dependencies [28], whose applicability to entity resolution has been pointed out. There is consensus that graph dependencies constitute a rare opportunity to specify semantics of application domains in graph databases [10, 13, 15, 28, 29]. Interestingly, the area of logical design and normalization has not been mentioned nor investigated yet for property graphs. In particular, our concept of graph-tailored functional dependencies (gFDs) is new, our results on the combined implication problem for gUCs and gFDs encompass simpler results for gUCs alone, and normalization has not been considered before at all. In our work, we transfer state-of-the-art schema design for FDs from relational to graph databases.

4 GRAPH-TAILORED CONSTRAINTS

We recall basics of the property graph model including the formal definition of graph-tailored uniqueness constraints [29]. We then introduce the new concept of graph-tailored functional dependencies and illustrate it on our running example.

Property Graph Model. We provide the basic definitions for the property graph model [7]. For this we assume that the following sets are pairwise disjoint: $\mathcal{O}$ denotes a set of objects, $\mathcal{L}$ denotes a finite set of labels, $\mathcal{K}$ denotes a set of property attributes, and $\mathcal{N}$ denotes a set of values.

A *property graph* is a quintuple $G = (V, Ed, \eta, \lambda, \nu)$ where $V \subseteq \mathcal{O}$ is a finite set of objects, called *vertices*, $Ed \subseteq \mathcal{O}$ is a finite set of objects, called *edges*, $\eta : Ed \rightarrow V \times V$ is a function assigning to each edge an ordered pair of vertices, $\lambda : V \cup Ed \rightarrow \mathcal{P}(\mathcal{L})$ is a function assigning to each object a finite set of labels, and $\nu : (V \cup Ed) \times \mathcal{K} \rightarrow \mathcal{N}$ is a partial function assigning values for

properties to objects, such that the set of domain values where ν is defined is finite. If $\nu(o, A)$ is defined, we write $\nu(o, A) = \downarrow$ and $\uparrow$ otherwise. Figures 1 and 2 show examples of property graphs.

Graph-tailored UCs. We now recall the syntax and semantics of gUCs [29]. These cover the uniqueness constraints, as used by Neo4j [16], as a special case. For that purpose, we define the subset $V_L \subseteq V$ of vertices in a property graph that carry at least all the labels of the given set $L \subseteq \mathcal{L}$, as follows: $V_L = \{v \in V \mid L \subseteq \lambda(v)\}$.

For a finite set $\mathcal{L}$ of labels and a finite set $\mathcal{K}$ of properties, a *graph-tailored uniqueness constraint* (or *gUC*) over $\mathcal{L}$ and $\mathcal{K}$ is an expression $L{:}P{:}X$ where $L \subseteq \mathcal{L}$ and $X \subseteq P \subseteq \mathcal{K}$. For a property graph $G = (V, Ed, \eta, \lambda, \nu)$ over $\mathcal{O}$, $\mathcal{L}$, $\mathcal{K}$, and $\mathcal{N}$ we say G *satisfies* the gUC $L{:}P{:}X$ over $\mathcal{L}$ and $\mathcal{K}$, denoted by $\models_G L{:}P{:}X$, iff there are no vertices $v_1, v_2 \in V_L$ such that $v_1 \neq v_2$, for all $A \in P$, $\nu(v_1, A)$ and $\nu(v_2, A)$ are defined, and for all $A \in X$, $\nu(v_1, A) = \nu(v_2, A)$.

Neo4j UCs are the special case of gUCs $L{:}P{:}X$ where $L = \{\ell\}$ and $P = X = \{p\}$, that is, $\{\ell\}{:}\{p\}{:}\{p\}$. Hence, they could simply be denoted by $\ell{:}X$. Similarly, Neo4j's composite indices are covered as the special case where $L = \{\ell\}$ and $P = X$, that is, $\{\ell\}{:}X{:}X$. Hence, they could simply be denoted by $\ell{:}X$.

Graph-tailored FDs. We will now introduce graph-tailored functional dependencies. Intuitively, such a dependency expresses that for nodes carrying a given set of labels and values on a given set of properties, the combination of values on some of those properties uniquely determine the values on some other properties.

Definition 4.1. For a finite set $\mathcal{L}$ of labels and a finite set $\mathcal{K}$ of properties, a *graph-tailored functional dependency* (or *gFD*) over $\mathcal{L}$ and $\mathcal{K}$ is an expression $L{:}P{:}X \rightarrow Y$ where $L \subseteq \mathcal{L}$ and $X, Y \subseteq P \subseteq \mathcal{K}$. For a given property graph $G = (V, Ed, \eta, \lambda, \nu)$ over $\mathcal{O}$, $\mathcal{L}$, $\mathcal{K}$, and $\mathcal{N}$ we say that G *satisfies* the gFD $L{:}P{:}X \rightarrow Y$ over $\mathcal{L}$ and $\mathcal{K}$, denoted by $\models_G L{:}P{:}X \rightarrow Y$, iff there are no vertices $v_1, v_2 \in V_L$ such that $v_1 \neq v_2$, for all $A \in P$, $\nu(v_1, A)$ and $\nu(v_2, A)$ are defined, for all $A \in X$, $\nu(v_1, A) = \nu(v_2, A)$ and for some $A \in Y$, $\nu(v_1, A) \neq \nu(v_2, A)$. □

We will now comment on the definition of gFDs. Firstly, the concept provides users with the flexibility to layer rules for nodes with different sets of labels. While $L{:}P{:}X \rightarrow Y$ applies to all nodes when $L = \emptyset$, adding new labels to L allows the user to declare additional rules that only apply to nodes that carry all of the labels in L. Secondly, the property set P addresses completeness requirements of applications on the properties that nodes may have. This means, unless a node exhibits values on all the properties in P, it does not need to conform to the FD $X \rightarrow Y$. Thirdly, we stipulate that the property sets X and Y are subsets of P. This choice is guided by the principle that missing properties should not affect the validity of a business rule. If completeness requirements are not available, we may simply use the gFD $L{:}XY{:}X \rightarrow Y$. Most gFDs that express meaningful rules will have this format, and they imply weaker gFDs $L{:}P{:}X \rightarrow Y$ where P results from XY by adding further properties. This serves multiple purposes as illustrated later, such as tailoring normalization to different requirements, discovering meaningful gFDs from property graphs and sources of inconsistent data.

In relational databases uniqueness constraints can be expressed by FDs. Indeed, FD $X \rightarrow Y$ over relation schema R simply expresses that every pair of records with matching values on all the fields in X also exhibit matching values on all the fields in Y. The special case where $Y = R$ means that there cannot be any pair of different records that has matching values on all the fields in X. This means $X \rightarrow R$ is a key dependency, or X constitutes a key. The reason is that relations are sets of records and no two different records can have matching values on all the fields in R. This observation is significant for normalization which transforms the underlying schema until all FDs exhibited on the schema are keys. Intuitively, any FD that may potentially cause data redundancy has been transformed into a key which cannot cause data redundancy.

This situation is different in data models that permit duplication, including the property graph model. Indeed, no gUC $L{:}P{:}X$ can be expressed by any gFD since we can always have two different nodes with label set L that carry matching values on all properties in P. While this graph satisfies the gFD $L{:}P{:}X \rightarrow P$, it does not satisfy the gUC $L{:}P{:}X$. Since gFDs cause data redundancy, gUCs prohibit data redundancy, and gUCs cannot be expressed by gFDs, we need to study the combined class of gUCs and gFDs.

In our example, the property graph in Figure 1 satisfies the gFD $Event{:}NCV{:}NC \rightarrow V$ but not the gUC $Event{:}NCV{:}NC$.

5 NORMALIZATION FRAMEWORK

We start developing our normalization framework by establishing axiomatic and linear-time algorithmic characterization of the implication problem for the combined class of gUCs and gFDs. We will then introduce Third (3NF) and Boyce-Codd Normal Form (BCNF) for property graphs tailored to labels and properties as required by applications. This will be followed by showing that our normal form proposals minimize (eliminate) data redundancy. Finally, we establish an algorithm that computes a lossless, dependency-preserving 3NF decomposition for a property graph, set of gUCs and gFDs, and the target set of labels and properties. Whenever possible, the output will even be in BCNF.

5.1 Reasoning

We formally define the implication problem for gUCs and gFDs over property graphs, illustrate it on examples, establish axiomatic and algorithmic solutions.

Given a set $\mathcal{L}$ of labels and a set $\mathcal{K}$ of properties, let $\Sigma \cup \{\varphi\}$ denote a set of constraints over $\mathcal{L}$ and $\mathcal{K}$ from a class C. The *implication problem* for C is to decide, given any input set $\Sigma \cup \{\varphi\}$ of constraints from C, whether Σ implies φ. In fact, Σ *implies* φ, denoted by $\Sigma \models \varphi$, if and only if every property graph G over $\mathcal{O}$, $\mathcal{L}$ and $\mathcal{K}$ that satisfies all constraints in Σ also satisfies φ.

The ability to efficiently decide whether φ is implied by Σ is fundamental for node integrity management on property graphs. If φ is implied by Σ, then we do not need to specify φ as it is specified already implicitly by Σ. However, if φ is not implied, failure to specify it explicitly may result in integrity faults that go undetected. The implication problem for FDs in relational databases is complete for *PTIME* [4, 9]. Since FDs form a special case of gFDs, the implication problem on property graphs is *PTIME*-hard.

5.1.1 Axiomatic Characterizations. We will establish an axiomatization for the combined class C of gUCs and gFDs. The set $\Sigma^*_C = \{\varphi \in C \mid \Sigma \models \varphi\}$ denotes the *semantic closure of* Σ, that is, the set of all constraints from C implied by Σ. The semantic closure does not tell us how to compute it. However, we determine the semantic closure Σ^*_C by applying *inference rules* of the form

Table 1: Axiomatization $\mathfrak{E} = \{\mathcal{R}, \mathcal{E}, \mathcal{T}, \mathcal{A}, \mathcal{W}, \mathcal{P}\}$ of gUC/FDs

$\frac{}{L{:}P{:}XY \to X}$ (reflexivity, $\mathcal{R}$)	$\frac{L{:}P{:}X \to Y}{L{:}P{:}X \to XY}$ (extension, $\mathcal{E}$)
$\frac{L{:}P{:}X}{LL'{:}PP'{:}XX'}$ (augmentation, $\mathcal{A}$)	$\frac{L{:}P{:}X \to Y \quad L'{:}P'{:}Y \to Z}{LL'{:}PP'{:}X \to Z}$ (transitivity, $\mathcal{T}$)
$\frac{L{:}P{:}X}{L{:}P{:}X \to P}$ (weakening, $\mathcal{W}$)	$\frac{L{:}P{:}X \to Y \quad L{:}P{:}XY}{L{:}P{:}X}$ (pullback, $\mathcal{P}$)

$\frac{\text{premise}}{\text{conclusion}}$. For a set $\mathfrak{R}$ of inference rules let $\Sigma \vdash_{\mathfrak{R}} \varphi$ denote the *inference* of φ from Σ by $\mathfrak{R}$. That is, there is some sequence $\sigma_1, \ldots, \sigma_n$ such that $\sigma_n = \varphi$ and every σ_i belongs to Σ or is the conclusion that results from applying an inference rule in $\mathfrak{R}$ to some premises in $\{\sigma_1, \ldots, \sigma_{i-1}\}$. Let $\Sigma^+_{\mathfrak{R}} = \{\varphi \mid \Sigma \vdash_{\mathfrak{R}} \varphi\}$ be the *syntactic closure* of Σ under inferences by $\mathfrak{R}$. $\mathfrak{R}$ is *sound* (*complete*) if for every set Σ of constraints from C we have $\Sigma^+_{\mathfrak{R}} \subseteq \Sigma^*_C$ ($\Sigma^*_C \subseteq \Sigma^+_{\mathfrak{R}}$). The (finite) set $\mathfrak{R}$ is a (finite) *axiomatization* if $\mathfrak{R}$ is both sound and complete.

For the rules of $\mathfrak{E}$ in Table 1, we implicitly assume that the gUCs and gFDs are well-formed. As example, for the augmentation rule $\mathcal{A}$ with $L{:}P{:}X$ and $LL'{:}PP'{:}XX'$ we assume that $X \subseteq P$ and $XX' \subseteq PP'$. Indeed, $\mathcal{A}$ by itself is sound and complete for the implication of gUCs. The full version of our paper shows that $\mathfrak{E}$ forms indeed a sound and complete set of inference rules for the implication of gUCs and gFDs over property graphs. The soundness of each rule is easily established by contra-position: assume some property graph violates the conclusion of a rule, one can show by simple arguments that one premise of the rule must be violated by the graph as well. The completeness proof constructs for any given gUC $L{:}P{:}X$ and any given gFD $L{:}P{:}X \to Y$ that cannot be inferred from Σ by using rules in $\mathfrak{E}$, a property graph that satisfies Σ and violates the given gUC or given gFD. This is achieved by introducing two vertices with label set L, matching values on all properties in $X^+_{\Sigma_{L:P}}$ and non-matching values on all remaining properties in P. Here, $X^+_{\Sigma_{L:P}}$ denotes all properties $A \in P$ such that the gFD $L{:}P{:}X \to A$ can be inferred from Σ using $\mathfrak{E}$.

Theorem 5.1. *The set $\mathfrak{E}$ forms a finite axiomatization for the implication of gUCs and gFDs over property graphs.* □

We illustrate inferencing on our running example.

Example 5.2. Suppose Σ consists of $\sigma_1 = \mathit{Event{:}CTV{:}CT} \to V$ and $\sigma_2 = \mathit{Event{:}NTV{:}VT} \to N$. Applying extension $\mathcal{E}$ to σ_1 gives us $\sigma_3 = \mathit{Event{:}CTV{:}CT} \to \mathit{CTV}$. Reflexivity $\mathcal{R}$ gives us $\sigma_4 = \mathit{Event{:}CTV{:}CTV} \to \mathit{VT}$, and applying transitivity $\mathcal{T}$ to σ_4 and σ_2 gives us $\sigma_5 = \mathit{Event{:}CTVN{:}CTV} \to N$. Finally, applying transitivity $\mathcal{T}$ to σ_3 and σ_5 gives us $\varphi = \mathit{Event{:}CNTV{:}CT} \to N$. Hence, Σ implies φ. Note the subtlety in reasoning with the completeness requirements for properties. As we will see below, Σ does not imply $\varphi' = \mathit{Event{:}CNT{:}CT} \to N$.

Algorithm 1 Implication of gUCs and gFDs

Require: Set $\Sigma \cup \{\varphi\}$ of gUCs and gFDs with $\varphi = L{:}P{:}X$ or $\varphi = L{:}P{:}X \to Y$
Ensure: *TRUE*, if $\Sigma \models \varphi$, and *FALSE*, otherwise
1: Compute $X^+_{\Sigma_{L:P}}$ using attribute set closure for FDs [4]
2: **if** $\varphi = L{:}P{:}X$ and $X^+_{\Sigma_{L:P}} = R_P$ **then**
3: **return** *TRUE*
4: **else if** $\varphi = L{:}P{:}X \to Y$ and $Y \subseteq X^+_{\Sigma_{L:P}}$ **then**
5: **return** *TRUE*
6: **else**
7: **return** *FALSE*

$\{\mathcal{R}, \mathcal{E}, \mathcal{T}\}$ forms an axiomatization for gFDs, a natural extension of the Armstrong axioms [2]. We will denote the latter by $\mathfrak{A}$.

5.1.2 Algorithmic Characterization. We use our axiomatization $\mathfrak{E}$ to establish an algorithm that decides implication efficiently.

For a set Σ of gUCs and gFDs, $L \subseteq \mathcal{L}$ and $P \subseteq \mathcal{K}$, we define the following set of FDs over the relation schema $R_P = P \cup \{A_0\}$:

$$\Sigma_{L:P} = \{X \to R_P \mid \exists L'{:}P'{:}X \in \Sigma \wedge L' \subseteq L \wedge P' \subseteq P\} \cup \{X \to Y \mid \exists L'{:}P'{:}X \to Y \in \Sigma \wedge L' \subseteq L \wedge P' \subseteq P\}\ .$$

$A_0 \notin P$ denotes a fresh property that does not occur elsewhere. Indeed, A_0 is only required in R_P when there is no gUC $L'{:}P'{:}X \in \Sigma$ with $L' \subseteq L$ and $P' \subseteq P$. That is, if $L'{:}P'{:}X \in \Sigma$ with $L' \subseteq L$ and $P' \subseteq P$, then $R_P := P$ is sufficient. Next we reduce implication of gUCs and gFDs over property graphs to the implication of classical FDs over relation schemata.

Theorem 5.3. *For every set $\Sigma \cup \{L{:}P{:}X, L{:}P{:}X \to Y\}$ over $\mathcal{L}$ and $\mathcal{K}$ and R_P, we have*
(1) $\Sigma \models L{:}P{:}X \to Y$ if and only if $\Sigma_{L:P} \models X \to Y$
(2) $\Sigma \models L{:}P{:}X$ if and only if $\Sigma_{L:P} \models X \to R_P$ □

Theorem 5.3 gives rise to Algorithm 1, which computes the property set closure $X^+_{\Sigma_{L:P}}$ of X for $\Sigma_{L:P}$ over R_P using the classical algorithm [4]. The decision branches in Algorithm 1 reflect the characterization by Theorem 5.3. This results in the linear time decidability of implication for gUCs and gFDs, based on that for classical FDs [4]. Hence, implication for gUC/FDs is *PTIME*-complete.

Corollary 5.4. *Algorithm 1 decides the implication problem for the combined class of gUCs and gFDs in linear input time.* □

We illustrate the algorithm on our running example.

Example 5.5. Consider $\Sigma = \{\sigma_1, \sigma_2\}$ and φ' from Example 5.2. Hence, $\Sigma_{\mathit{Event{:}CNT}} = \emptyset$ and $N \notin X^+_{\Sigma_{\mathit{Event{:}CNT}}} = \mathit{CT}$, which means that Algorithm 1 returns a negative answer. However, for $\Sigma_{\mathit{Event{:}CNTV}} = \{\mathit{CT} \to V, \mathit{VT} \to N\}$, such that for φ we get $N \in X^+_{\Sigma_{\mathit{Event{:}CNTV}}} = \mathit{CTVN}$, so that Algorithm 1 returns a positive answer.

5.2 Normal Forms for Graph Data

We define Boyce-Codd and Third Normal Form for sets of gUCs and gFDs. Based on opportunities that graph data provides, we start by explaining our approach. We then introduce and illustrate our proposals, and present results on their achievements.

5.2.1 Approach. Since property graphs have no schema, it is challenging to define classical normal forms for graph data. Indeed, we address this challenge using our class of graph-tailored constraints. Moreover, graph data is flexible, which provides further opportunities in defining normal forms. Indeed, applications target graph objects based on their labels and properties. We therefore understand these features as requirements: The application targets only those nodes that exhibit a given set L of labels and a given set P of properties. With that approach, we then normalize that part of the graph which meets these targets. Hence, normalization becomes flexible and driven by application requirements.

5.2.2 Boyce-Codd Normal Form (BCNF). Classical BCNF casts a syntactic definition that prevents any possible occurrence of redundant data values by stipulating that every FD, which could potentially cause redundancy, is actually a key dependency (and can therefore never cause any redundancy). We will now define BCNF for sets of gUCs and gFDs. The definition is tailored to prevent redundant property value occurrences on graphs that satisfy the constraints and whose nodes meet the requirements for labels and properties.

Definition 5.6. (L:P-Boyce-Codd Normal Form)
Let Σ denote a set of gUCs and gFDs over $\mathcal{L}$ and $\mathcal{K}$. For sets $L \subseteq \mathcal{L}$ and $P \subseteq \mathcal{K}$, we say that Σ is in L:P-Boyce-Codd Normal Form (L:P-BCNF) if and only if for every gFD L:P:$X \rightarrow Y \in \Sigma^+_{\mathfrak{C}}$ it is true that $Y \subseteq X$ or L:P:$X \in \Sigma^+_{\mathfrak{C}}$. □

We illustrate the definition on our running example.

Example 5.7. Property graph G from Figure 1 satisfies Σ with gUC *Event*:*CT*:*CT*, and gFDs *Event*:*NTV*:*NT* $\rightarrow$ *V*, *Event*:*NTV*:*VT* $\rightarrow$ *N* and *Event*:*NC*:*N* $\rightarrow$ *C*. Indeed, Σ is in *Event*:*CT*-BCNF, but neither in *Event*:*NC*-, *Event*:*NCT*-, *Event*:*NTV*-, nor *Event*:*NCTV*-BCNF. In contrast, property graph G' from Figure 2 satisfies Σ' with gUCs *Evt_Mgt*:*NC*:*N*, *Evt_Comp*:*CVT*:*CT*, *Evt_Comp*:*CVT*:*VT*, *Evt_Detail*:*NVT*:*NT*, and *Evt_Detail*:*NVT*:*VT*. Σ' is in *Evt_Mgt*:*NC*-BCNF, *Evt_Comp*:*CVT*-BCNF, and *Evt_Detail*:*NVT*-BCNF.

For any label set L and property set P, we can check whether Σ is in L:P-BCNF by checking if $(R_P, \Sigma_{L:P})$ is in BCNF. That is, our BCNF definition is tailored to label and property sets of graphs.

Theorem 5.8. *For every label set L and property set P, it holds that Σ is in L:P-BCNF if and only if $(R_P, \Sigma_{L:P})$ is in BCNF.* □

Continuing with Example 5.7, Σ is not in *Event*:*NTV*-BCNF since $R_P = NTVA_0$ is not in BCNF for $\Sigma_{Event:NTV} = \{VT \rightarrow N, NT \rightarrow V\}$ (note that $A_0 \in R_P$ and, $VT \rightarrow R_P, NT \rightarrow R_P \notin \Sigma^+_{Event:NTV}$). Similarly, Σ is not in *Event*:*NCTV*-BCNF since $R_P = NCTV$ is not in BCNF for $\Sigma_{Event:NCTV} = \{N \rightarrow C, CT \rightarrow NV, NT \rightarrow V, VT \rightarrow N\}$ ($N \rightarrow R_P \notin \Sigma^+_{Event:NCTV}$).

The condition for Σ to be in L:P-BCNF is independent of how Σ is represented. That is, for every gUC/FD set Θ where $\Sigma^+_{\mathfrak{C}} = \Theta^+_{\mathfrak{C}}$, it follows that Σ is in L:P-BCNF if and only if Θ is in L:P-BCNF. This feature is due to Definition 5.6 where we check all gFDs in $\Sigma^+_{\mathfrak{C}}$, which may be of size exponential in Σ. However, we can show it is sufficient to check Σ itself, making the test for L:P-BCNF efficient.

Theorem 5.9. *Σ in L:P-BCNF iff for every gFD L':P':$X \rightarrow Y \in \Sigma$ where $L' \subseteq L$ and $P' \subseteq P$, $Y \subseteq X$ or L:P:$X \in \Sigma^+_{\mathfrak{C}}$.* □

Theorem 5.9 allows us to check in time quadratic in $|\Sigma|$ whether Σ is in L:P-BCNF. We simply need to test if $X^+_{\Sigma_{L:P}} = R_P$ for every L':P':$X \rightarrow Y \in \Sigma$ where $L' \subseteq L$, $P' \subseteq P$ and $Y \not\subseteq X$. We can compute $X^+_{\Sigma_{L:P}}$ in time linear in $|\Sigma_{L:P} \cup \{X\}|$ using the classical attribute set closure algorithm [4].

Corollary 5.10. *The condition whether Σ is in L:P-BCNF can be checked in time quadratic in $|\Sigma|$.* □

5.2.3 Third Normal Form. While it is always possible to achieve a lossless decomposition into BCNF, some FDs may be lost during the process. That is, some FDs may require a join of schemata resulting from the decomposition before their validity can be tested. Since this is prohibitively expensive, the community has made dependency-preservation another goal of normalization. It is always possible to achieve a lossless, dependency-preserving decomposition into 3NF. Current state-of-the-art finds a lossless, dependency-preserving decomposition into 3NF, which is in BCNF whenever possible. We target this result for property graphs.

Towards defining 3NF, we say property $A \in P$ is *L:P-prime for* Σ iff there is some L:P:$X \in \Sigma^+_{\mathfrak{C}}$ such that $A \in X$, and for all proper subsets $Y \subset X$, L:P:$Y \notin \Sigma^+_{\mathfrak{C}}$. Hence, A is contained in some minimal L:P-key for Σ. Unlike relational databases, there may not be a key at all and thus no prime property.

Definition 5.11. Let Σ be a set of gUCs and gFDs over $\mathcal{L}$ and $\mathcal{K}$. For $L \subseteq \mathcal{L}$ and $P \subseteq \mathcal{K}$, Σ is in *L:P-Third Normal Form (L:P-3NF)* if and only if for every gFD L:P:$X \rightarrow Y \in \Sigma^+_{\mathfrak{C}}$ it is true that $Y \subseteq X$ or L:P:$X \in \Sigma^+_{\mathfrak{C}}$ or every property in $Y - X$ is L:P-prime. □

Example 5.7 has shown that Σ is not in *Event*:*NCTV*-BCNF. Based on the gUC *Event*:*CT*:*CT* we obtain other gUCs *Event*:*NCTV*:*VT* and *Event*:*NCTV*:*NT* in $\Sigma^+_{\mathfrak{C}}$, which are *Event*:*NCTV*-minimal. In particular, every property in *NCTV* is *Event*:*NCTV*-prime. Hence, Σ is in *Event*:*NCTV*-3NF.

Similar to L:P-BCNF, the definition to L:P-3NF is grounded in classical 3NF but tailored to graph features.

Theorem 5.12. *For every label set L and property set P it holds that Σ is in L:P-3NF if and only if $(R_P, \Sigma_{L:P})$ is in 3NF.* □

Given the set of L:P-prime properties for Σ, we can validate in time quadratic in Σ whether Σ is in L:P-3NF.

Theorem 5.13. *Σ in L:P-3NF if and only if for every gFD L':P':$X \rightarrow Y \in \Sigma$ where $L' \subseteq L$ and $P' \subseteq P$ it is true that $Y \subseteq X$ or L:P:$X \in \Sigma^+_{\mathfrak{C}}$ or every property in $Y - X$ is L:P-prime.* □

While testing L:P-BCNF is efficient, validating L:P-3NF is likely intractable due to the *NP*-completeness of deciding whether a property is L:P-prime, already for the case where $L = \emptyset$ and $P = R$ is a relation schema [4]. Likewise, it is *coNP*-complete to decide whether for a given Σ, $\Sigma_{L:P:S}$ is in L:P-BCNF where $S \subseteq P$ and

$$\Sigma_{L:P:S} = \{L'{:}P'{:}X \rightarrow Y \in \Sigma^+_{\mathfrak{C}} \mid L' \subseteq L \wedge P' \subseteq P \wedge XY \subseteq S \subseteq P\} .$$

Theorem 5.14. *Deciding for Σ, if $\Sigma_{L:P:S}$ is in L:P-BCNF, is* coNP*-complete. Deciding whether Σ is in L:P-3NF is* NP*-complete.* □

5.3 Achievements of Normal Forms

The objective of our normalization is to minimize sources of property values that may occur redundantly in graphs that satisfy the

underlying set of gUCs and gFDs. We will now illustrate in which formal sense this is actually achieved.

Let Σ denote a set of gUCs and gFDs over $\mathcal{L}$ and $\mathcal{K}$, and G be a property graph over $\mathcal{O}$, $\mathcal{L}$ and $\mathcal{K}$ that satisfies Σ. Let $L \subseteq \mathcal{L}$ and $P \subseteq \mathcal{K}$. Let v denote a node of G that carries all labels in L and has values defined for all properties in P. Let $A \in P$. An *L:P-replacement of v on A* is any property graph G' that results from G by only changing the current property value $\nu(v, A)$ to some different value. The property value occurrence $\nu(v, A)$ is *L:P-redundant for* Σ if and only if for **every** *L:P*-replacement G' of v on A, the graph G' violates some constraint $L{:}P : \sigma \in \Sigma^+_{\mathfrak{C}}$.

Definition 5.15. Σ is in *L:P-Redundancy Free Normal Form (RFNF)* iff there is no property graph G that satisfies Σ, no node $v \in V_{L:P}$ in G, and no property $A \in P$ such that $\nu(v, A)$ is *L:P*-redundant for Σ.

In graph G of Figure 1, each occurrence of "Cactus", namely in $\nu(2, \text{Company})$ and $\nu(3, \text{Company})$, is *Event:NCVT*-redundant. For instance, if G' results from G by replacing $\nu(2, \text{Company})$ by a value different from "Cactus", G' will violate gFD $\mathit{Event{:}NCVT{:}N} \rightarrow C \in \Sigma^+_{\mathfrak{C}}$. Hence, Σ is not in *Event:NCVT*-RFNF. In contrast, the occurrence of $\nu(23{-}1, \text{Company}) =$ "Cactus" in graph G' of Figure 2 is not *Evt_Mgt:NC*-redundant. Indeed, Σ' is in *Evt_Mgt:NC*-RFNF.

Theorem 5.16. *For all sets Σ of gUC/FDs, for all label sets L and property sets P, we have Σ is in L:P-RFNF iff $(R_P, \Sigma_{L:P})$ is in RFNF.* □

In illustrating Theorem 5.16, the relation r corresponding to node set $V_{\mathit{Event:NCTV}}$ of graph G in Figure 1 is

Event	*Company*	*Venue*	*Time*
No Socks	Cactus	Vault	06/01
No Socks	Cactus	Vault	13/07

and r satisfies $\Sigma_{\mathit{Event:NCTV}} = \{N \rightarrow C, NT \rightarrow V, TV \rightarrow N, CT \rightarrow NV\}$, and each occurrence of "Cactus" is redundant.

As targeted, BCNF captures Redundancy-Free Normal Form.

Corollary 5.17. *For all sets Σ of gUC/FDs, for all label sets L and property sets P, we have Σ is in L:P-RFNF iff Σ is in L:P-BCNF.*

Using information theory, Kolahi and Libkin [19] demonstrated that among all dependency-preserving decompositions, those that are in 3NF admit the highest guaranteed information content possible for any data value occurrences. In other words, 3NF pays the least price in terms of data redundancy in order to achieve dependency-preservation. Due to Theorem 5.12, these results carry over to *L:P*-3NF, pending our definition of decompositions below.

5.4 Normalizing Property Graphs

We will now show how to restructure, without loss of information and guided by target sets L of node labels and P of properties, a given gUC/FD set Σ and a given property graph G that satisfies Σ such that the restructered constraint set is satisfied by the restructured graph and is in $L : P$-3NF, and $L : P$-BCNF whenever possible.

We first describe the general method informally, illustrate it on our running example, and then provide the technical definitions.

5.4.1 Method. Intuitively, the normalization process is as follows.

1) Given L, P, G and Σ, for each node $v \in V_{L:P}$ and each element S of a decomposition for P (a set $\mathcal{D}$ of subsets for R_P), we introduce new nodes v_S with fresh label ℓ_S and directed edges (v_S, v) with fresh label ℓ, and transfer the properties in S from v to v_S. For each S, these operations result in the projection $G^{\ell}_{L:P}[S]$ of $G_{L:P}$ onto S, where $G_{L:P}$ is the restriction of G onto $V_{L:P}$.

2) We then materialize the "redundancy elimination" by identifying new nodes v_S and v'_S whenever they exhibit matching values on all properties in S. Technically, this is achieved by a congruence relation $\equiv_S$, and forming the quotient graph $G^{\ell}_{L:P}[S]/{\equiv_S}$.

3) We then take the union of quotient graphs over all elements S of the decomposition $\mathcal{D}$. The resulting property graph $G^{\ell}_{L:P}[\mathcal{D}]$ is an *L:P*-decomposition of G onto $\mathcal{D}$.

4) Similarly, the *L:P*-decomposition $\Sigma^{\ell}_{L:P}[\mathcal{D}]$ of Σ onto $\mathcal{D}$ is obtained by adding gUCs $\ell_S{:}S{:}X$ for each $X \rightarrow R_P \in \Sigma_{L:P}[S]$ and adding gFDs $\ell_S{:}S{:}X \rightarrow Y$ for each $X \rightarrow Y \in \Sigma_{L:P}[S]$ for $Y \neq R_P$.

This construction can easily be inverted by collapsing all edges (v_S, v) labelled ℓ and transferring back the property/value pairs from v_S to the node v they originated from. The original dependencies imply new ones on the new nodes, transforming gFDs into gUCs whenever possible, which is why property value redundancy is removed as far as possible. Due to labels, we can simply add the new ones, and remove them when the decomposition is inverted.

Consider again Example 5.7. The set Σ' is a lossless, dependency-preserving *Event:CNTV*-decomposition of Σ into BCNF. The property graph G' in Figure 2 is the Event:*NCTV*-decomposition of G in Figure 1, based on the decomposition $\mathcal{D} = \{NC, NVT, CVT\}$ of R_{NCTV}. For the G-nodes 2 and 3 we obtain new G'-nodes $2-1$, $2-2$, $2-3$, $3-1$, $3-2$ and $3-3$, of which $2-1$ and $3-1$ are identified to node $23-1$ by value equality on *NC*.

5.4.2 Technical Definitions. For property graph G, $L \subseteq \mathcal{L}$, and $P \subseteq \mathcal{K}$, we define $G_{L:P}$ to denote the restriction of G to the vertex set $V_{L:P}$. For a property set $S \subseteq P$, and a label $\ell \in \mathcal{L}$ that does not occur in G, we define the *L:P*-projection $G^{\ell}_{L:P}[S]$ of $G_{L:P}$ onto S by

- $V_{L:P}[S] := V_{L:P} \cup \bigcup_{v \in V_{L:P}} \{v_S\}$
- $Ed_{L:P}[S] := \bigcup_{v \in V_{L:P}} \{(v_S, v)\}$
- $\lambda_{L:P}[S] := \begin{cases} \ell_S & \text{, if } v \in V_{L:P}[S] \\ \lambda(v_S, v) := \ell & \text{, if } (v_S, v) \in Ed_{L:P}[S] \end{cases}$
- $\mu_{L:P}[S] := \begin{cases} \mu(v) & \text{, if } v \in V_{L:P} \\ id(v_S) & \text{, if } v_S \in V_{L:P}[S] - V_{L:P} \\ id(v_S, v) & \text{, if } (v_S, v) \in Ed_{L:P}[S] \end{cases}$
- $\nu_{L:P}[S] := \begin{cases} \nu(v_S, A) := \nu(v, A) & \text{, if } A \in S \wedge \lambda(v_S, v) = \ell \\ \nu(v_S, A) := \uparrow & \text{, if } A \notin S \wedge \lambda(v_S, v) = \ell \\ \nu(v, A) & \text{, if } A \notin S \wedge v \in V_{L:P} \\ \uparrow & \text{, if } A \in S \wedge v \in V_{L:P} \end{cases}$

For a property set $S \subseteq \mathcal{K}$ and two nodes v, v' of a property graph, we define $v \equiv_S v'$ if and only if for all $A \in S$, $\nu(v, A) = \nu(v', A)$. That is, the two nodes are equivalent on the property set S if and only if they have matching values on all the properties in S. Of course, $\equiv_S$ defines an equivalence relation between the nodes of a property graph G, so we may define the quotient graph $G/{\equiv_S}$.

For two property graphs G and G' over $\mathcal{O}$, $\mathcal{L}$, and $\mathcal{K}$ we define the union $G \cup G'$ as the property graph obtained as $V_G \cup V_{G'}$, $Ed_G \cup Ed_{G'}$, $\lambda_G \cup \lambda_{G'}$, $\mu_G \cup \mu_{G'}$ but where $\nu_G \cup \nu_{G'}$ is defined by $\nu(v, A) \uparrow$ for any property $A \in \mathcal{K}$ whenever $\nu_G(v, A)$ and $\nu_{G'}(v, A)$ have non-matching values (eg. only one of them is defined).

Finally, for a property graph G and label $\ell \in \mathcal{L}$ we define $\overset{\ell}{\bowtie} G$ as the following property graph:

- $V := V_G - \{v' \in V_G \mid \exists (v', v) \in Ed_G, \lambda(v', v) = \ell\}$
- $Ed := Ed\mid_V, \lambda := \lambda_G\mid_V, \mu := \mu_G\mid_V$, and
- $\nu := \begin{cases} \nu(v, A) := \nu_G(v, A) & \text{, if } v \in V \wedge \nu_G(v, A)\downarrow \\ \nu(v, A) := \nu_G(v', A) & \text{, if } (v', v) \in Ed_G \wedge \\ & \quad \lambda_G(v', v) = \ell \wedge \nu_G(v', A)\downarrow \end{cases}$

Recall from relational databases that a *decomposition* of an attribute set R is a set $\mathcal{D}$ of subsets of R such that $\bigcup_{S\in\mathcal{D}} S = R$. Likewise, for a set Σ of FDs over R, and a subset $S \subseteq R$, $\Sigma[S] = \{X \to Y \in \Sigma^+_{\mathfrak{A}} \mid XY \subseteq S\}$ is the projection of Σ onto S.

Definition 5.18. Let Σ denote a set of gUCs and gFDs, and let L and P be sets of labels and properties, respectively. Given decomposition $\mathcal{D}$ of R_P, we define the *L:P-projection* $\Sigma^\ell_{L:P}[\mathcal{D}]$ *of* Σ *onto* $\mathcal{D}$ by

$$\Sigma \cup \{\ell_S{:}S{:}X \mid X \to R_P \in \Sigma_{L:P}[S] \text{ for } S \in \mathcal{D}\} \cup \{\ell_S{:}S{:}X \to Y \mid X \to Y \in \Sigma_{L:P}[S] \wedge Y \neq R_P \wedge S \in \mathcal{D}\} .$$

We say the *L:P-decomposition* $\Sigma^\ell_{L:P}[\mathcal{D}]$ *of* Σ *is in BCNF (3NF)* iff for all $S \in \mathcal{D}$, $\Sigma^\ell_{L:P}[\mathcal{D}]$ is in $\ell_S : S$-BCNF (3NF). The *L:P*-decomposition $\Sigma^\ell_{L:P}[\mathcal{D}]$ of Σ is *dependency-preserving* iff $\Sigma_{L:P}$ and $\bigcup_{S\in\mathcal{D}} \left(\Sigma^\ell_{L:P}[\mathcal{D}]\right)_{\ell_S:S}$ are covers of one another. For decomposition $\mathcal{D}$ of R_P, the *L:P-decomposition* $G^\ell_{L:P}[\mathcal{D}]$ *of a property graph* G *onto* $\mathcal{D}$ by $G^\ell_{L:P}[\mathcal{D}] := \bigcup_{S\in\mathcal{D}} G^\ell_{L:P}[S]/\equiv_S$. The *L:P*-decomposition $\Sigma^\ell_{L:P}[\mathcal{D}]$ of Σ is *lossless* iff for every property graph G that satisfies Σ, the *L:P*-decomposition $G^\ell_{L:P}[\mathcal{D}]$ of G onto $\mathcal{D}$ satisfies $G_{L:P} = \overset{\ell}{\bowtie} G^\ell_{L:P}[\mathcal{D}]$. □

Our decomposition is always lossless, no matter whether any constraint holds. However, only when a gFD is converted into a gUC, all redundancy caused by the gFD is eliminated. Indeed, normalizing a property graph will eliminate redundancy on those equivalence classes where the underlying gFD holds (on classes where it fails to hold, the corresponding nodes cannot be identified).

Algorithm 2 normalizes a given property graph G and gUC/FD set Σ tailored to a label set L and property set P. Our techniques make it possible for lines (1-15) to utilize a state-of-the-art normalization algorithm from relational databases which achieves a lossless, dependency-preserving 3NF decompostion $\mathcal{D}$ into BCNF whenever possible. Following our previous definitions, $\mathcal{D}$ is then converted into the output $\left(G^\ell_{L:P}[\mathcal{D}], \Sigma^\ell_{L:P}[\mathcal{D}]\right)$.

Theorem 5.19. *On input* $((G, \Sigma), L \cup \{\ell\}, P)$ *such that* G *satisfies* Σ*, Algorithm 2 returns the property graph* $G^\ell_{L:P}[\mathcal{D}]$ *that satisfies* $\Sigma^\ell_{L:P}[\mathcal{D}]$*, which is a lossless, dependency-preserving L:P-decomposition of* Σ *into 3NF that is in BCNF whenever possible.* □

6 EXPERIMENTS

Our experiments will showcase the extent of both opportunities and benefits of normalizing graph data. This will be done quantitatively and qualitatively using popular real-world property graphs, but also synthetic graph data for scalability tests. The research questions we aim to answer by our experiments are:

Q1) What gFDs do property graphs exhibit?
Q2) What gFDs cause much data redundancy?
Q3) How much inconsistency can gFDs avoid?
Q4) What does graph normalization actually look like?

Algorithm 2 NormaG

Require: Property graph G that satisfies gUC/FD set Σ; label set $L \cup \{\ell\}$; property set P
Ensure: Property graph $G^\ell_{L:P}[\mathcal{D}]$ that satisfies $\Sigma^\ell_{L:P}[\mathcal{D}]$, which is a lossless, dependency-preserving *L:P*-decomposition of Σ into 3NF (which is in BCNF whenever possible)

1: Compute atomic closure $\overline{\Sigma}_a$ of $\Sigma_{L:P}$ on R_P [27];
2: $\Sigma_a \leftarrow \overline{\Sigma}_a$
3: **for all** $X \to A \in \Sigma_a$ **do**
4: **for all** $Y \to B \in \overline{\Sigma}_a (YB \subseteq XA \wedge XA \not\subseteq Y^+)$ **do**
5: **if** $\Sigma_a - \{X \to A\} \models X \to A$ **then**
6: $\Sigma_a \leftarrow \Sigma_a - \{X \to A\}$ {Eliminate critical schemata}
7: $\mathcal{D} \leftarrow \emptyset$
8: **for all** $X \to A \in \Sigma_a$ **do**
9: **if** $\Sigma_a - \{X \to A\} \models X \to A$ **then**
10: $\Sigma_a \leftarrow \Sigma_a - \{X \to A\}$ {Eliminate redundant schemata}
11: **else**
12: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(XA, \overline{\Sigma}_a[XA])\}$
13: Remove all $(S, \overline{\Sigma}_a[S]) \in \mathcal{D}$ if $\exists (S', \overline{\Sigma}_a[S']) \in \mathcal{D} (S \subseteq S')$
14: **if** there is no $(R', \Sigma') \in \mathcal{D}$ where $R' \to R_E \in \Sigma^+_{L:P}$ **then**
15: Choose a minimal key K for R_P with respect to $\Sigma_{L:P}$
16: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(K, \overline{\Sigma}_a[K])\}$
17: **return** $\left(G^\ell_{L:P}[\mathcal{D}], \Sigma^\ell_{L:P}[\mathcal{D}]\right)$

Table 2: Details on graph datasets used in the experiments

Graph data	L	$\lvert V_L\rvert$	$\lvert P\rvert$	$\%V_{L:P}$	#gFDs	AvgRed	#gUCs
Northwind	Order	830	14	35.54%	555	25.13	49
Offshore	Entity	814,345	18	26.14%	1414	47,919.13	102

Q5) How much better is integrity managed after normalization?
Q6) How much faster are aggregate queries after normalization?
Q7) How do benefits of normalization scale?

Answers to the first three questions will illustrate why normalization is necessary. For Question 4, we will demonstrate how normalizing a real-world graph looks like, and the last three questions will underline benefits of normalization at operational level.

6.1 Data Sets and Measures

Details of experiments are on our Github repository https://github.com/GraphDatabaseExperiments/normalization_experiments. The graphs we analyzed include *Northwind* (github.com/neo4j-graph-examples/northwind) with 1,035 nodes and 3,139 edges, with information on customers and products they ordered, and *Offshore* (github.com/ICIJ/offshoreleaks-data-packages) with 2,016,524 nodes and 3,336,971 edges which gives information on companies registered in different parts of the world. Table 2 shows the node labels L we target, the number $\lvert V_L\rvert$ of nodes with label L, the number $\lvert P\rvert$ of all properties carried by those nodes, the percentage $\%V_{L:P}$ of those nodes that have all properties, the numbers #gFDs and #gUCs in a minimal cover of those constraints that hold on the data sets, and the average number of redundant property values caused by gFDs. The latter number shows that gFDs capture significant amounts of redundant property values, particularly in *Offshore*.

Figure 3: #gFDs by Property Size $|P|$

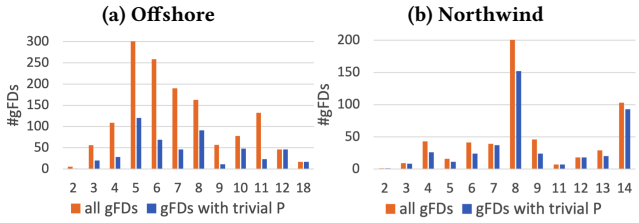


Figure 4: Consistency Profiles for gFDs

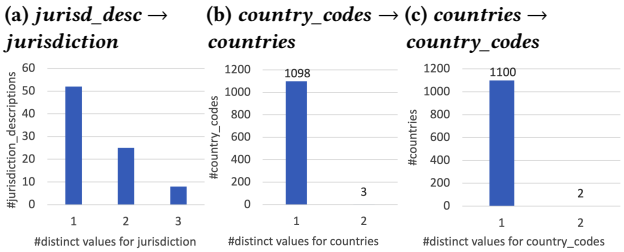


Table 3: gFDs on Offshore Ranked by Redundancy Caused

$P \setminus XY$	X	Y	#red	#inc
incorporation_date	jurisd_desc, lastEditTimestamp, sourceID	jurisdiction	788,408	20,000
incorporation_date	jurisd_desc, sourceID, valid_until	jurisdiction	788,365	175,871
	incorporation_date, jurisd_desc, valid_until	service_provider	754,283	1,047
ibcRUC	jurisd_desc, valid_until	service_provider	555,353	20,000
ibcRUC	jurisd_desc, lastEditTimestamp	service_provider	555,353	175,888
ibcRUC	jurisdiction, lastEditTimestamp, valid_until	sourceID	555,338	20,000
country_codes	jurisd_desc, lastEditTimestamp, sourceID	jurisdiction	504,944	20,000
countries	jurisd_desc, lastEditTimestamp, sourceID	jurisdiction	504,944	20,000
country_codes	jurisd_desc, sourceID, valid_until	jurisdiction	504,902	113,055
countries	jurisd_desc, sourceID, valid_until	jurisdiction	504,902	113,055
	country_codes, sourceID	countries	504,424	83,647
	countries, sourceID	country_codes	504,424	83,647
	country_codes, valid_until	countries	504,418	83,647
	countries, valid_until	country_codes	504,418	83,647
	countries, jurisd_desc	country_codes	504,227	83,653
	countries, jurisdiction	country_codes	504,151	83,647
valid_until	country_codes, jurisd_desc, sourceID	jurisdiction	504,110	83,647
valid_until	countries, jurisd_desc, sourceID	jurisdiction	504,110	83,647
	country_codes, jurisd_desc, valid_until	jurisdiction	504,106	83,647
	countries, jurisd_desc, valid_until	jurisdiction	504,106	83,647
	country_codes, lastEditTimestamp	countries	503,977	19,988
	countries, lastEditTimestamp	country_codes	503,977	19,988
incorporation_date	countries	country_codes	490,969	83,989

We used *Neo4j* and its query language *Cypher* as representative graph database system. As performance measure we used *database hits*, an abstract unit of the storage engine related to requests for operations on nodes or edges. We also used Python 3.9.13. Experiments were conducted on a 64-bit operating system with an Intel Core i7 Processor with 16GB RAM. Details of experiments are available in https://github.com/GraphDatabaseExperiments/normalization_experiments.

6.2 What gFDs do graphs exhibit?

We mined gFDs with fixed target labels *Entity* (Offshore) and *Order* (Northwind). Figure 3 classifies the gFDs $L : P : X \rightarrow Y$ by the size $|P|$ of their property set P. If $P = XY$, we call P trivial.

The mined gFDs include interesting examples. On *Offshore*, for instance, we have the gFDs

- *Entity* : *address*, *country_codes*, *countries*: *countries*, *address* → *country_codes*
- *Entity* : *service_provider*, *country_codes*, *countries*: *countries* → *country_codes*.

In particular, for every mined $L : P : X \rightarrow Y$, removing any property from P or X will result in a gFD that is violated by the dataset. Hence, the gFD *Entity*:*country_codes*, *countries*:*countries* → *country_codes* is not satisfied by the dataset.

6.3 What gFDs cause much data redundancy?

Interesting for normalization purposes are gFDs that cause many occurrences of redundant property values. Ultimately, human users decide which constraints express meaningful business rules. However, ranking gFDs by the number of redundant property value occurrences they cause can provide helpful guidance for such decisions. As an example for *Offshore*, Table 3 shows the gFDs with Label *Entity* that cause the most number of redundant value occurrences (#*red*). These numbers are huge, and ought to be targeted by normalization. While the following gFDs $L : P : X \rightarrow Y$ may appear to be meaningful:

(1) *Entity* : *jurisd_desc*, *jurisdiction*:*jurisd_desc* → *jurisdiction*
(2) *Entity* : *country_codes*, *countries*:*country_codes* → *countries*
(3) *Entity* : *country_codes*, *countries*:*countries* → *country_codes*

neither of them actually holds. Nevertheless, adding few properties to P or X results in various gFDs that do hold and exhibit many redundant property values. This makes us wonder whether gFDs (1), (2) or (3) are only violated due to data inconsistencies that are a result of data redundancy and the fact these gFDs are not enforced.

6.4 How much inconsistency can gFDs avoid?

We have seen various gFDs that cause many redundant value occurrences. If these gFDs represent actual business rules, they form a primary target for graph normalization. We will now illustrate how to inform decisions whether gFDs are meaningful and violations constitute inconsistencies. We will discuss a negative and positive case, further strengthening the use of graph constraints and normalization to avoid data redundancy and sources of inconsistency.

Table 3 lists the potential level of inconsistency (#*inc*) associated with a gFD *Entity* : $P : X \rightarrow Y$ on *Offshore*. Hence, if the gFD is not enforced, there may be up to #*inc* nodes in $V_{Entity:P}$ that have matching values on all properties in X but have each different values on properties in Y. For each of the gFDs, #*inc* represents the worst-case scenario of not enforcing the constraint.

Let us examine gFD (1) which is violated due to *Entity*-nodes with matching values for property *jurisd_desc* and different values for property *jurisdiction*. For instance, nodes with *jurisd_desc* 'Bahamas' have either *jurisdiction* 'BAH', 'BHS' or 'BA'. Similarly, there are multiple *jurisdiction*s associated with the same *jurisd_desc*-value in 32 other cases. This consistency profile is illustrated in Figure 4(a), where we list the number of *jurisdiction_descriptions* that have n distinct *jurisdictions* associated with them, for $n = 1, 2, 3$. It is

Table 4: Summary of Normalizing *Offshore*

P	$\|P_i\|$	#gFDs	#FDs	#red	#dbhits	time (ms)	$\|\mathcal{D}_i\|$
P_1	4	3	2	684,608	8,930,544	5,566	2
P_2	5	5	5	1,008,998	28,845,388	22,697	4
P_3	6	10	5	1,369,802	39,474,122	17,284	5

plausible that multiple jurisdictions can be associated with the same jurisdiction description. Hence, gFD (1) may not be meaningful.

In contrast, gFD (2) exhibits a different consistency profile, as shown in Figure 4(b). There are only three different *country_codes* that have two distinct *countries* linked to them, while the 1098 other codes are linked to unique countries. Indeed, for *country_code* 'COK' there are 464 nodes with value "Cook Islands" and 1389 nodes with value "COK" for property *countries*. The only other inconsistencies are linked to values "GBR;VGB" and "VGB;COK" for *country_codes*.

Finally, gFD (3) exhibits a similar profile as gFD (2), shown in Figure 4(c). Here, the only exceptions are given by *country_code* "GBR;VGB" linked to "United Kingdom; British Virgin Islands" and "United Kingdom; Virgin Islands, British"; and "COK" associated with both "Cook Islands" and "COK". The impact of such inconsistency should not be underestimated. A query asking about "Cook Islands", which is popular due to its status as a 'Tax Haven', can only recall one quarter of all relevant nodes due to the incorrect value of "COK" on this property for three quarter of the nodes.

Hence, mined gFDs and their ranking provide useful heuristics to identify meaningful gFDs and data inconsistency in the form of their violations. Meaningful gFDs and consistent graph data form input desirable for normalization.

6.5 What does graph normalization look like?

While our running example is sufficiently small to illustrate our concepts and ideas, we will now examine three applications of Algorithm 2 to the property graph *Offshore*. All three applications target nodes with label *Entity (E)* but different property sets:
$P_1 = \{$*jurisd_desc(jd), countries(c), service_provider(sp), country_codes(cc)*$\}$,
$P_2 = \{$*jd, valid_until (v), c, sourceID (s), cc*$\}$ and $P_3 = P_1 \cup P_2$.
As set Σ we consider gFDs $E : P \backslash XY : X \to Y$ as follows:

$$E{:}sp{:}c \to cc; E{:}\emptyset{:}c, jd \to cc; E{:}sp{:}cc \to c; E{:}\emptyset{:}c, s \to cc;$$
$$E{:}\emptyset{:}c, v \to cc; E{:}\emptyset{:}cc, s \to c; E{:}\emptyset{:}cc, v \to c;$$
$$E{:}\emptyset{:}sp \to s, v; E{:}sp{:}s \to v; E{:}sp{:}v \to s.$$

For $R_1 = R_{P_1} = \{jd, c, sp, cc, a_1\}$ and $\Sigma_1 = \Sigma_{E:P_1} = \{c \to cc, cc \to c\}$ we get the BCNF decomposition $\mathcal{D}_1$ of (R_1, Σ_1) into

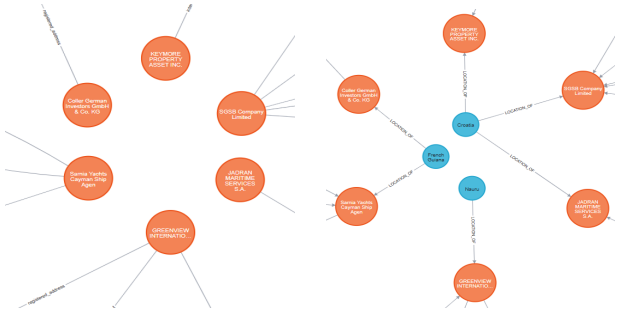
- $R_1^1 = \{c, cc\}$ with $\Sigma_1^1 = \{c \to cc; cc \to c\}$, and
- $R_1^2 = \{jd, sp, c, a_1\}$ with $\Sigma_1^2 = \emptyset$.

Hence, we obtain the gUCs $\ell_{R_1^1}{:}R_1^1{:}\{c\}$ and $\ell_{R_1^1}{:}R_1^1{:}\{cc\}$, and $\Sigma^{\ell}_{E:P_1}[\mathcal{D}_1]$ is in BCNF.

For $R_2 = R_{P_2} = \{jd, v, c, s, cc, a_2\}$ and $\Sigma_2 = \Sigma_{E:P_2} = \{c, jd \to cc; cs, \to cc; c, v \to cc; cc, s \to c; cc, v \to c\}$ we obtain the BCNF decomposition $\mathcal{D}_2$ of (R_2, Σ_2):

- $R_2^1 = \{c, cc, v\}$ with $\Sigma_2^1 = \{c, v \to cc; cc, v \to c\}$
- $R_2^2 = \{c, cc, s\}$ with $\Sigma_2^2 = \{c, s \to cc; cc, s \to c\}$
- $R_2^3 = \{c, cc, jd\}$ with $\Sigma_2^3 = \{c, jd \to cc\}$
- $R_2^4 = \{jd, s, v, c, a_2\}$ with $\Sigma_1^4 = \emptyset$.

Figure 6: *Offshore* Snippet: Original vs. Normalized



Hence, we get the gUCs $\ell_{R_2^1}{:}R_2^1{:}\{c, v\}$; $\ell_{R_2^1}{:}R_2^1{:}\{cc, v\}$; $\ell_{R_2^2}{:}R_2^2{:}\{c, s\}$; $\ell_{R_2^1}{:}R_2^1{:}\{cc, s\}$; $\ell_{R_2^3}{:}R_2^3{:}\{c, jd\}$ and $\Sigma^{\ell}_{E:P_2}[\mathcal{D}_2]$ is in BCNF.

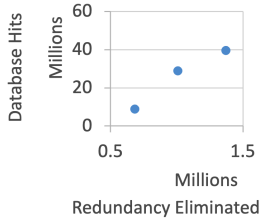
For $R_3 = R_{P_3} = \{jd, sp, v, c, s, cc, a_3\}$ and $\Sigma_3 = \Sigma_{E:P_3} = \{c \to cc; cc, \to c; sp \to s, v; s \to v; v \to s\}$ we obtain the BCNF decomposition $\mathcal{D}_3$ of (R_3, Σ_3):

- $R_3^1 = \{c, cc\}$ with $\Sigma_3^1 = \{c \to cc; cc \to c\}$
- $R_3^2 = \{sp, s\}$ with $\Sigma_3^2 = \{sp \to s\}$
- $R_3^3 = \{sp, v\}$ with $\Sigma_3^3 = \{sp \to v\}$
- $R_3^4 = \{s, v\}$ with $\Sigma_3^4 = \{s \to v; v \to s\}$
- $R_3^5 = \{jd, sp, c, a_3\}$ with $\Sigma_3^4 = \emptyset$.

Hence, we obtain the gUCs $\ell_{R_3^1}{:}R_3^1{:}\{c\}$; $\ell_{R_3^1}{:}R_3^1{:}\{cc\}$; $\ell_{R_3^2}{:}R_3^2{:}\{sp\}$; $\ell_{R_3^3}{:}R_3^3{:}\{sp\}$; $\ell_{R_3^4}{:}R_3^4{:}\{s\}$; $\ell_{R_3^4}{:}R_3^4{:}\{v\}$ and $\Sigma^{\ell}_{E:P_3}[\mathcal{D}_3]$ is in BCNF.

We then used Cypher to compute, for $i = 1, 2, 3$, the *Entity* : P_i-decomposition $G^{\ell}_{Entity:P_i}[\mathcal{D}_i]$ of graph G (*Offshore*). The results are summarized in Table 4. For each property set P_i we show its size $|P_i|$, the number #gFDs of gFDs *Entity* : $P' : X \to Y \in \Sigma$ such that $P' \subseteq P_i$, the number #FDs in a cover of $\Sigma_{Entity:P_i}$, the number #red of distinct redundant value occurrences in G caused by the gFDs, the number #dbhits of database hits for computing $G^{\ell}_{Entity:P_i}[\mathcal{D}_i]$, the time to compute $G^{\ell}_{Entity:P_i}[\mathcal{D}_i]$, and the size $|\mathcal{D}_i|$ of decomposition $\mathcal{D}_i$. Figure 5 illustrates that the graph normalization query uses its access and time effectively to eliminate redundant property value occurrences. Finally, Figure 6 shows a glimpse into the effect of normalizing *Offshore* into $L : P_1$-BCNF.

Figure 5: Good Riddance



6.6 How does integrity management improve?

The primary target for graph normalization is the elimination of redundant property values such that sources of inconsistency are avoided and the overhead of update operations is minimized. We will now quantify these benefits of graph normalization. We will compare the performance of updates between the original and normalized graph for gFDs $L : P : X \to Y$ as follows:
For *Offshore*: $L = \{E\}$, $P = \{sp, s, v\}$, $X = \{sp\}$, $Y = \{s, v\}$

Figure 7: Update Comparison: Original vs. Normalized

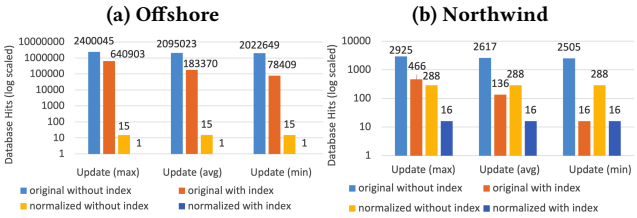
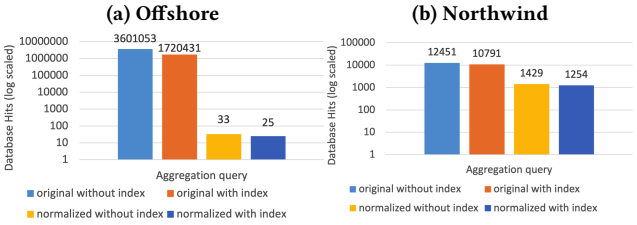


Figure 8: Aggregate Queries: Original vs. Normalized



For *Northwind*: $L = \{O(rder)\}$, $P = \{customerID(cI), shipCity(sC), shipName(sN), shipPostalCode(sP), shipCountry(sCo), shipAddress(sA), shipRegion(sR)\}$, $X = \{cI\}$, $Y = \{sC, sN, sP, sCo, sA, sR\}$.

Using Neo4j we performed the following update in the original *Offshore* graph using Cypher query:

```
MATCH (e : Entity) WHERE EXISTS(e.service_provider) AND EXISTS(e.sourceID)
AND EXISTS(e.valid_until) AND e.service_provider = 'Appleby'
SET e.valid_until = 'Appleby data is current through 2015'
```

and the following update in the original *Northwind* graph using:

```
MATCH (o : Order) WHERE EXISTS(o.customerID) AND EXISTS(o.shipCity) AND
EXISTS(o.shipName) AND EXISTS(o.shipPostalCode) AND EXISTS(o.shipCountry)
AND EXISTS(o.shipAddress) AND EXISTS(o.shipRegion) AND
o.customerID = 'CENTC' SET o.shipCountry = 'Estados Unidos Mexicanos' .
```

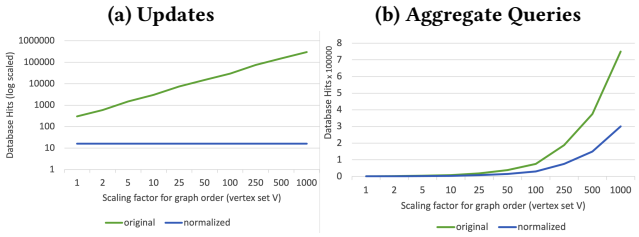
The queries were run using values for *sevice_provider* and *customerID* with the min, avg, and max number of redundant occurrences. We then performed these updates on the graphs normalized by the gFDs above, and compared the number of database hits. We also performed the operations using an index for V_L on the property X. The different results can be seen in Figure 7.

Due to high redundancy in *Offshore*, graph normalization improves update performance by multiple orders of magnitude, irrespective of min, avg or max numbers. Indices result in further optimization. On *Northwind*, with less redundancy, update performance still improves by an order of magnitude. For values with min or avg occurrences, indexing the original graph shows good performance. However, indexing the normalized graph excels, and is only matched for values with minimum redundancy. These benefits already hold by applying a single FD for normalization only.

6.7 How do aggregate queries improve?

Next we illustrate the benefit of speeding up aggregate queries, using the parameters for node set $V_{L:P}$ from the previous section.

Figure 9: Scaling Comparison: Original vs. Normalized



As typical aggregate queries, we access information on the numbers of orders associated with a given *customerID* in *Northwind*, and on the numbers of entities for a given *service_provider* in *Offshore*:

```
MATCH (e : Entity) WHERE EXISTS(e.service_provider) AND EXISTS(e.sourceID) AND
EXISTS(e.valid_until) WITH (e.service_provider) AS provider, COUNT(*) AS amount
RETURN min(amount), max(amount), avg(amount)
```

and

```
MATCH (o : Order) WHERE EXISTS(o.customerID) AND EXISTS(o.shipCity) AND
EXISTS(o.shipName) AND EXISTS(o.shipPostalCode) AND EXISTS(o.shipCountry) AND
EXISTS(o.shipAddress) AND EXISTS(o.shipRegion) WITH o.customerID AS orders,
COUNT (*) AS amount RETURN min(amount), max(amount), avg(amount) .
```

We compare their performance to corresponding queries on the normalized graph. Results are shown in Figure 8, including those after introducing an index for V_L on the property X. Normalization improves query performance by several orders of magnitude on *Offshore*, and one order of magnitude on *Northwind*. The index does improve performance, but has not as big an impact as for updates.

6.8 How do benefits of normalization scale?

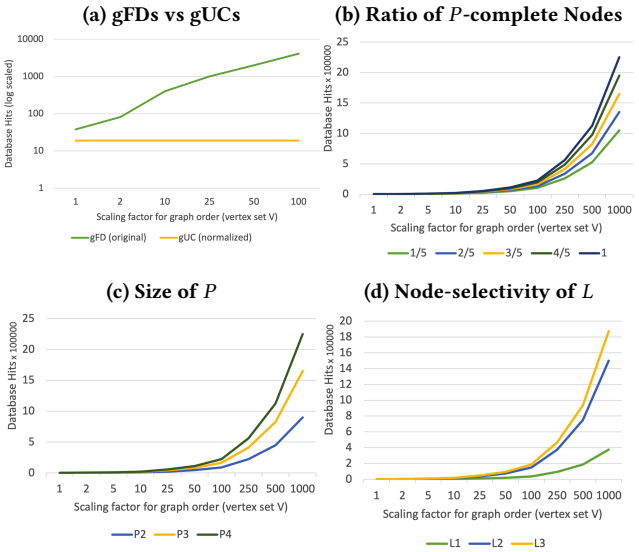
We will report how graph size impacts on update and aggregate query performance, but also on the validation of our constraints and their features. We utilized synthetic datasets as follows.

We created graphs that consist of a node labelled *Company* with edges to *Employee*-nodes that have properties *name*, *department* and *manager*, with additional properties for some experiments. Our underlying business rule says that every department has at most one manager, resulting in the gFD $\varphi = \{Employee\} : \{department, manager\} : \{department\} \rightarrow \{manager\}$. For each experiment, we perform a query on the same baseline graph and scale this graph by factor k to have k times as many *Employee*-nodes while keeping the number of departments fixed.

Figure 9 compares the performance of (a) updating manager names, and (b) querying the minimum, average and maximum number of employees per department, both between the original and normalized graph (with respect to gFD φ), respectively. In particular, Figure 9(a) conveys the main message that normalization scales update performance perfectly. Indeed, access to the normalized graph using gUCs remains constant while access to the original graph using gFDs keeps on growing. For aggregate queries the performance improvement is also very noticeable.

Figure 10(a) underlines the perfect scalability of validating gUCs resulting from gFD φ on the normalized graph in contrast to growing access necessary for validating φ on the original graph. From (b) it can be seen how validation performance scales with the ratio

Figure 10: Validation at Scale



of nodes that have all properties in P. From (c) we observe how validation performance scales in the size of the underlying property set P. Indeed, Pi contains $i + 1$ properties. Finally, (d) shows how validation of φ scales in the node selectivity of labels in φ. Indeed, the database hits required are directly proportional to the number of nodes with the given label set present.

7 CONCLUSION AND FUTURE WORK

We have introduced the area of normalization to graph data, tailoring the expressive class of functional dependencies to graph application requirements for node labels and properties. For this class of constraints, we have transferred the full normalization framework from relational to graph databases, including BCNF, 3NF, and the State-of-the-Art algorithm that returns a lossless, dependency-preserving BCNF decomposition whenever possible. Our experiments with real-world graph data illustrate how our constraints capture many redundant property value occurrences and potential inconsistency, and how our algorithms transform graphs to eliminate/minimize them. Our experiments have further demonstrated the efficacy of graph normalization. Indeed, the reduction of overheads for update maintenance and the speed up of aggregate queries by orders of magnitude, as well as the effort required to normalize the graph are all proportional to the amount of redundancy removed.

In future work, we will address other classes of constraints and normal forms, but also different data quality dimensions including variety, veracity and velocity. We also propose the area of *conditional normalization* as a promising direction of research. Indeed, if a gFD holds conditionally for some property values, the corresponding gUC is satisfied on the normalized graph for the same property values, making conditional versions of constraints [14] particularly attractive to graph normalization.

REFERENCES

[1] Marcelo Arenas. 2006. Normalization theory for XML. *SIGMOD Record* 35, 4 (2006), 57–64.
[2] William W. Armstrong. 1974. Dependency Structures of Data Base Relationships. In *IFIP Congress*. 580–583.
[3] Carlo Batini and Andrea Maurino. 2018. Design for Data Quality. In *Encyclopedia of Database Systems, Second Edition*, Ling Liu and M. Tamer Özsu (Eds.).
[4] Catriel Beeri and Philip A. Bernstein. 1979. Computational Problems Related to the Design of Normal Form Relational Schemas. *ACM Trans. Database Syst.* 4, 1 (1979), 30–59.
[5] Philip A. Bernstein. 1976. Synthesizing Third Normal Form Relations from Functional Dependencies. *ACM Trans. Database Syst.* 1, 4 (1976), 277–298.
[6] Joachim Biskup, Umeshwar Dayal, and Philip A. Bernstein. 1979. Synthesizing Independent Database Schemas. In *SIGMOD*. 143–151.
[7] Angela Bonifati, George H. L. Fletcher, Hannes Voigt, and Nikolay Yakovets. 2018. *Querying Graphs*. Morgan & Claypool Publishers.
[8] Edgar F. Codd. 1972. Further normalization of the database relational model. In *Courant Computer Science Symposia 6: Data Base Systems*. 33–64.
[9] William F. Dowling and Jean H. Gallier. 1984. Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *J. Log. Program.* 1, 3 (1984), 267–284.
[10] Renzo Angles et al. 2021. PG-Keys: Keys for Property Graphs. In *SIGMOD '21: International Conference on Management of Data, Virtual Event*. 2423–2436.
[11] Ronald Fagin. 1977. Multivalued Dependencies and a New Normal Form for Relational Databases. *ACM Trans. Database Syst.* 2, 3 (1977), 262–278.
[12] Ronald Fagin. 1981. A Normal Form for Relational Databases That Is Based on Domains and Keys. *ACM Trans. Database Syst.* 6, 3 (1981), 387–415.
[13] Wenfei Fan. 2019. Dependencies for Graphs: Challenges and Opportunities. *ACM J. Data Inf. Qual.* 11, 2 (2019), 5:1–5:12.
[14] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.* 33, 2 (2008).
[15] Wenfei Fan and Ping Lu. 2019. Dependencies for Graphs. *ACM Trans. Database Syst.* 44, 2 (2019), 5:1–5:40.
[16] Emil Eifrem Ian Robinson, Jim Webber. 2015. *Graph Databases*. O'Reilly Media.
[17] Christian S. Jensen, Richard T. Snodgrass, and Michael D. Soo. 1996. Extending Existing Dependency Theory to Temporal Databases. *IEEE Trans. Knowl. Data Eng.* 8, 4 (1996), 563–582.
[18] Henning Köhler and Sebastian Link. 2016. SQL Schema Design: Foundations, Normal Forms, and Normalization. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016*. 267–279.
[19] Solmaz Kolahi and Leonid Libkin. 2010. An information-theoretic analysis of worst-case redundancy in database design. *ACM Trans. Database Syst.* 35, 1 (2010), 5:1–5:32.
[20] Mark Levene and George Loizou. 1999. Database Design for Incomplete Relations. *ACM Trans. Database Syst.* 24, 1 (1999), 80–125.
[21] Mark Levene and Millist Vincent. 2000. Justification for Inclusion Dependency Normal Form. *IEEE Trans. Knowl. Data Eng.* 12, 2 (2000), 281–291.
[22] Sebastian Link and Henri Prade. 2019. Relational database schema design for uncertain data. *Inf. Syst.* 84 (2019), 88–110.
[23] Sebastian Link and Ziheng Wei. 2021. Logical Schema Design that Quantifies Update Inefficiency and Join Efficiency. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. 1169–1181.
[24] Heikki Mannila and Kari-Jouko Räihä. 1992. *Design of Relational Databases*. Addison-Wesley.
[25] Wai Yin Mok. 2016. Utilizing Nested Normal Form to Design Redundancy Free JSON Schemas. *Int. J. Recent Contributions Eng. Sci. IT* 4, 4 (2016), 21–25.
[26] Wai Yin Mok, Yiu-Kai Ng, and David W. Embley. 1996. A Normal Form for Precisely Characterizing Redundancy in Nested Relations. *ACM Trans. Database Syst.* 21, 1 (1996), 77–106.
[27] Sylvia L. Osborn. 1979. Testing for Existence of a Covering Boyce-Codd normal Form. *Inf. Process. Lett.* 8, 1 (1979), 11–14.
[28] Larissa Capobianco Shimomura, George Fletcher, and Nikolay Yakovets. 2020. GGDs: Graph Generating Dependencies. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management*. 2217–2220.
[29] Philipp Skavantzos, Kaiqi Zhao, and Sebastian Link. 2021. Uniqueness Constraints on Property Graphs. In *Advanced Information Systems Engineering - 33rd International Conference, CAiSE 2021*. 280–295.
[30] Zahir Tari, John Stokes, and Stefano Spaccapietra. 1997. Object Normal Forms and Dependency Constraints for Object-Oriented Schemata. *ACM Trans. Database Syst.* 22, 4 (1997), 513–569.
[31] Millist Vincent. 1997. A Corrected 5NF Definition for Relational Database Design. *Theor. Comput. Sci.* 185, 2 (1997), 379–391.
[32] Millist Vincent and Mark Levene. 2000. Restructuring Partitioned Normal Form Relations without Information Loss. *SIAM J. Comput.* 29, 5 (2000), 1550–1567.
[33] Ziheng Wei and Sebastian Link. 2021. Embedded Functional Dependencies and Data-completeness Tailored Database Design. *ACM Trans. Database Syst.* 46, 2 (2021), 7:1–7:46.

A PROOFS

A.1 Reasoning

We will now show that $\mathfrak{E}$ from Table 1 forms indeed a sound and complete set of inference rules for the implication of gUCs and gFDs over property graphs. First, we establish soundness of the rules, meaning that any consecutive applications of the inference rules can never result in gUC or gFD not implied by the input set.

Lemma A.1. *The inference rules in Table 1 are sound for the implication of gUCs and gFDs in property graphs.*

Proof. Soundness of the augmentation rule $\mathcal{A}$ was already proven in [29]. We will now prove soundness of the remaining rules.

For soundness of the reflexivity axiom $\mathcal{R}$ we note that for every property graph G with two vertices that carry all labels in L and have values for all properties in P, values on properties in X match whenever values on all properties in $X \cup Y$ match.

For soundness of the extension rule $\mathcal{E}$ we use contra-position and assume there is a property graph G that violates the gFD $L : P : X \to XY$. We need to show that G also violates the gFD $L : P : X \to Y$. Since G violates $L : P : X \to XY$, there are vertices $v_1, v_2 \in V_L$ such that $v_1 \neq v_2$ and for all $A \in P$, $\nu(v_1, A)$ and $\nu(v_2, A)$ are defined, for all $A \in X$, $\nu(v_1, A) = \nu(v_2, A)$, and for some $A \in XY$, $\nu(v_1, A) \neq \nu(v_2, A)$. Consequently, there is some $A \in Y$ such that $\nu(v_1, A) \neq \nu(v_2, A)$. We have just shown that G also violates the gFD $L : P : X \to Y$.

For soundness of the transitivity rule $\mathcal{T}$ we use contra-position and assume there is a property graph G that violates the gFD $LL' : PP' : X \to Z$. We need to show that G also violates the gFD $L : P : X \to Y$ or the gFD $L' : P' : Y \to Z$. Since G violates $LL' : PP' : X \to Z$, there are vertices $v_1, v_2 \in V_{LL'}$ such that $v_1 \neq v_2$ and for all $A \in PP'$, $\nu(v_1, A)$ and $\nu(v_2, A)$ are defined, for all $A \in X$, $\nu(v_1, A) = \nu(v_2, A)$, and for some $A \in Z$, $\nu(v_1, A) \neq \nu(v_2, A)$. If G violates the gFD $L : P : X \to Y$, then there is nothing more to show. Hence, we assume that G satisfies the gFD $L : P : X \to Y$. Consequently, we know that for all $A \in Y$, $\nu(v_1, A) = \nu(v_2, A)$. However, under this assumption we can see directly that G violates the gFD $L' : P' : Y \to Z$.

For soundness of the weakening rule $\mathcal{W}$ we use contra-position and assume there is a property graph G that violates the gFD $L : P : X \to Y$. We need to show that G also violates the gUC $L : P : X$. Since G violates $L : P : X \to Y$, there are vertices $v_1, v_2 \in V_L$ such that $v_1 \neq v_2$ and for all $A \in P$, $\nu(v_1, A)$ and $\nu(v_2, A)$ are defined, for all $A \in X$, $\nu(v_1, A) = \nu(v_2, A)$, and for some $A \in Y$, $\nu(v_1, A) \neq \nu(v_2, A)$. Consequently, we have just shown that G also violates the gUC $L : P : X$.

For soundness of the pullback rule $\mathcal{P}$ we use contra-position and assume there is a property graph G that violates the gUC $L : P : X$. We need to show that G also violates the gFD $L : P : X \to Y$ or the gUC $L : P : XY$. Since G violates $L : P : X$, there are vertices $v_1, v_2 \in V_L$ such that $v_1 \neq v_2$ and for all $A \in P$, $\nu(v_1, A)$ and $\nu(v_2, A)$ are defined, and for all $A \in X$, $\nu(v_1, A) = \nu(v_2, A)$. If G violates the gUC $L : P : XY$, then there is nothing more to show. Hence, we assume that G satisfies the gUC $L : P : XY$. Consequently, we know for some $A \in Y$, $\nu(v_1, A) \neq \nu(v_2, A)$. However, under this assumption G violates the gFD $L : P : X \to Y$. □

Lemma A.2. *The following rules are sound*

$$\frac{L{:}P{:}X \to YZ}{L{:}P{:}X \to Y}\ (\textit{decomposition}, \mathcal{D}) \qquad \frac{L{:}P{:}X \to Y \quad L{:}P{:}X \to Z}{L{:}P{:}X \to YZ}\ (\textit{union}, \mathcal{U}) \qquad \frac{L{:}P{:}X \to Y}{LL'{:}PP'{:}X \to Y}\ (\textit{increase}, \mathcal{I})$$

for the implication of gFDs.

Proof. The *decomposition*-rule $\mathcal{D}$ can be inferred from the rules in $\mathfrak{E}$ as follows:

$$\dfrac{L : P : X \to YZ \qquad \overline{(\mathcal{R}) \quad L : P : YZ \to Y}}{(\mathcal{T}) \qquad L : P : X \to Y}$$

and the *union*-rule $\mathcal{U}$ can be inferred from the rules in $\mathfrak{E}$ as follows:

$$\dfrac{\dfrac{\dfrac{L{:}P{:}X \to Y}{(\mathcal{E}) \quad L{:}P{:}X \to XY} \qquad \dfrac{\dfrac{\overline{(\mathcal{R}) \quad L{:}P{:}XY \to X} \qquad L{:}P{:}X \to Z}{(\mathcal{T}) \qquad L{:}P{:}XY \to Z}}{(\mathcal{E}) \qquad L{:}P{:}XY \to XYZ} \qquad \overline{(\mathcal{R}) \quad L{:}P{:}XYZ \to YZ}}{(\mathcal{T}) \qquad L{:}P{:}XY \to YZ}}{(\mathcal{T}) \qquad L{:}P{:}X \to YZ}$$

The *increase*-rule $\mathcal{I}$ can be inferred from the rules in $\mathfrak{E}$ as follows:

$$\dfrac{L : P : X \to Y \qquad \overline{(\mathcal{R}) \quad L' : P' : Y \to Y}}{(\mathcal{T}) \qquad LL' : PP' : X \to Y}$$

This completes the proof. □

Theorem A.3 (Theorem 5.1 restated). *The set $\mathfrak{E}$ forms a finite axiomatization for the implication of gUCs and gFDs over property graphs.*

Proof. The soundness of $\mathfrak{E}$ was already established in Lemma A.1.

We show the completeness of $\mathfrak{E}$ by contra-position. Let $\Sigma \cup \{\varphi\}$ denote a set of gUCs and gFDs over $\mathcal{L}$ and $\mathcal{K}$ such that $\varphi \notin \Sigma^+_{\mathfrak{E}}$. We will show that Σ does not imply φ by defining a property graph G that satisfies all gUCs and gFDs in Σ and violates φ. We distinguish two cases where φ denotes either the gUC $L : P : X$ or the gFD $L : P : X \to Y$, respectively. In either case, let

$$X^+_{\Sigma_{L:P}} = \{A \in P \mid \Sigma \vdash_{\mathfrak{E}} L : P : X \to A\}$$

denote the *property set closure* of X with respect to $L : P$ and Σ. The soundness of the *union*-rule $\mathcal{U}$, established in Lemma A.2, shows that $L : P : X \to X^+_{\Sigma_{L:P}} \in \Sigma^+_{\mathfrak{E}}$ holds.

Let us define the property graph $G = (V, Ed, \eta, \lambda, \nu)$ as follows: $V = \{v_1, v_2\}$, $Ed = \emptyset$, and therefore there is nothing to define for η, $\lambda(v_1) = L = \lambda(v_2)$, for all $A \in X^+_{\Sigma_{L:P}}$ we define $\nu(v_1, A) = 0 = \nu(v_2, A)$, and for all $A \in E - X^+_{\Sigma_{L:P}}$ we define $\nu(v_1, A) = 0$ and $\nu(v_2, A) = 1$.

Case 1. Since $X \subseteq X^+_{\Sigma_{L:P}}$ holds, it follows from the construction of G that G violates $L : P : X$. It therefore remains to show in this case that G satisfies every gUC and every gFD in Σ. Let σ denote such an element of Σ.

Case 1.a) Here, σ denotes the gUC $L' : P' : X' \in \Sigma$. Assume, to the contrary, that G violates σ. It follows from the construction of G that $L' \subseteq L$, $P' \subseteq P$ and $X' \subseteq X^+_{\Sigma_{L:P}}$ hold. Applying the *extension*-rule $\mathcal{E}$ to $L' : P' : X' \in \Sigma$ gives us $L : P : X^+_{\Sigma_{L:P}} \in \Sigma^+_{\mathfrak{E}}$. Since $L : P : X \to X^+_{\Sigma_{L:P}} \in \Sigma^+_{\mathfrak{E}}$ holds, we can apply the *pullback*-rule $\mathcal{P}$ to infer $L : P : X \in \Sigma^+_{\mathfrak{E}}$. This is a contradiction to our assumption that $\varphi = L : P : X \notin \Sigma^+_{\mathfrak{E}}$. Consequently, our assumption that G violates σ must have been wrong, and we conclude that r satisfies σ in this case.

Case 1.b) Here, σ denotes the gFD $L' : P' : X' \to Y' \in \Sigma$. Assume again, to the contrary, that G violates σ. It follows from the construction of G that $L' \subseteq L$, $P' \subseteq P$, $X' \subseteq X^+_{\Sigma_{L:P}}$, and $Y' \cap (P - X^+_{\Sigma_{L:P}}) \neq \emptyset$ all hold. From $L : P : X \to X^+_{\Sigma_{L:P}} \in \Sigma^+_{\mathfrak{E}}$ and $X' \subseteq X^+_{\Sigma_{L:P}}$ we infer $L : P : X \to X' \in \Sigma^+_{\mathfrak{E}}$ by applying the *decompose*-rule $\mathcal{D}$ from Lemma A.2. Applying the *transitivity*-rule $\mathcal{T}$ to $L : P : X \to X' \in \Sigma^+_{\mathfrak{E}}$ and $L' : P' : X' \to Y' \in \Sigma$, we infer $LL' : PP' : X \to Y' \in \Sigma^+_{\mathfrak{E}}$. Since $L' \subseteq L$, $P' \subseteq P$, we actually have $L : P : X \to Y' \in \Sigma^+_{\mathfrak{E}}$. According to the definition of the property set closure, we must then have that $Y' \subseteq X^+_{\Sigma_{L:P}}$. This, however, is a contradiction to $Y' \cap (P - X^+_{\Sigma_{L:P}}) \neq \emptyset$. Consequently, our assumption that G violates σ must have been wrong, and we conclude that G satisfies σ in this case.

Case 2. If $Y \subseteq X^+_{L:E,\Sigma}$, then $L : E : X \to Y \in \Sigma^+_{\mathfrak{E}}$ contrary to our assumption. Hence, $Y \cap (E - X^+_{L:E,\Sigma}) \neq \emptyset$. Since $X \subseteq X^+_{L:E,\Sigma}$, $Y \cap (E - X^+_{E,\Sigma}) \neq \emptyset$, it follows from the construction of G that G violates $L : E : X \to Y$. It therefore remains to show in this case that G satisfies every gUC and every gFD in Σ. Let σ denote such an element of Σ.

Case 2.a) Here, σ denotes the gUC $L' : P' : X' \in \Sigma$. Assume, to the contrary, that G violates σ. It follows from the construction of G that $L' \subseteq L$, $P' \subseteq P$ and $X' \subseteq X^+_{\Sigma_{L:P}}$ both hold. Applying the *extension*-rule $\mathcal{E}$ to $L' : P' : X' \in \Sigma$ gives us $L : P : X^+_{\Sigma_{L:P}} \in \Sigma^+_{\mathfrak{E}}$. Since $L : P : X \to X^+_{\Sigma_{L:P}} \in \Sigma^+_{\mathfrak{E}}$ holds, we can apply the *pullback*-rule $\mathcal{P}$ to infer $L : P : X \in \Sigma^+_{\mathfrak{E}}$. From the *weakening*-rule $\mathcal{W}$ we can then infer $L : P : X \to P \in \Sigma^+_{\mathfrak{E}}$, and since $L : P : P \to Y \in \Sigma^+_{\mathfrak{E}}$, we can apply the *transitivity*-rule $\mathcal{T}$ to $L : P : X \to P \in \Sigma^+_{\mathfrak{E}}$ and $L : P : P \to Y \in \Sigma^+_{\mathfrak{E}}$ to infer the contradiction that $L : P : X \to Y \in \Sigma^+_{\mathfrak{E}}$. Hence, our assumption must have been wrong, and G satisfies σ in this case.

Case 2.b) Here, σ denotes the gFD $L' : P' : X' \to Y' \in \Sigma$. Assume again, to the contrary, that G violates σ. It follows from the construction of G that $L' \subseteq L$, $P' \subseteq P$, $X' \subseteq X^+_{\Sigma_{L:P}}$, and $Y' \cap (P - X^+_{\Sigma_{L:P}}) \neq \emptyset$ all hold. From $L : P : X \to X^+_{\Sigma_{L:P}} \in \Sigma^+_{\mathfrak{E}}$ and $X' \subseteq X^+_{\Sigma_{L:P}}$ we infer $L : P : X \to X' \in \Sigma^+_{\mathfrak{E}}$ by applying the *decomposition*-rule $\mathcal{D}$ from Lemma A.2. Applying the *transitivity*-rule $\mathcal{T}$ to $L : P : X \to X' \in \Sigma^+_{\mathfrak{E}}$ and $L' : P' : X' \to Y' \in \Sigma$, we infer $LL' : PP' : X \to Y' \in \Sigma^+_{\mathfrak{E}}$. Since $L' \subseteq L$ and $P' \subseteq P$, we actually have $L : P : X \to Y' \in \Sigma^+_{\mathfrak{E}}$. According to the definition of the property set closure, we must then have that $Y' \subseteq X^+_{\Sigma_{L:P}}$. This, however, is a contradiction to $Y' \cap (P - X^+_{\Sigma_{L:P}}) \neq \emptyset$. Consequently, our assumption that G violates σ must have been wrong, and we conclude that G satisfies σ in this case.

This concludes the completeness proof. □

Theorem A.4 (Theorem 5.3 restated). *For every set $\Sigma \cup \{L : P : X, L : P : X \to Y\}$ over $\mathcal{L}$ and $\mathcal{K}$ and $R_P = P \cup \{A_0\}$ with a fresh attribute $A_0 \notin P$, the following holds*

(1) $\Sigma \models L : P : X \to Y$ if and only if $\Sigma_{L:P} \models X \to Y$
(2) $\Sigma \models L : P : X$ if and only if $\Sigma_{L:P} \models X \to R_P$

Proof. **1.** Suppose $\Sigma \not\models L : P : X \to Y$. Then there is a property graph $G = (V, Ed, \eta, \lambda, \nu)$ that satisfies all gUCs and gFDs in Σ but violates $L : P : X \to Y$. In particular, there are two vertices $v, v' \in V$ whose label sets include all labels in L, both vertices carry all properties in P, and the values $\nu(v, A)$ and $\nu(v', A)$ are matching on all the properties $A \in X$ but there is some property $B \in Y$ on which $\nu(v, B)$ and $\nu(v', B)$ are not matching. We are now defining the following two-tuple relation $r := \{t_v, t_{v'}\}$ over $R_P = P \cup \{A_0\}$ as follows $t_v(A) := \nu(v, A)$ for all $A \in P$ and $t_v(A_0) := 0$, and $t_{v'}(A) := \nu(v', A)$ for all $A \in P$ and $t_{v'}(A_0) := 1$. It follows that r violates $X \to Y$ since $XY \subseteq P$. It remains to show that r satisfies all $X' \to R_P$ and $X' \to Y' \in \Sigma_{L:P}$. For $X' \to R_P \in \Sigma_{L:P}$ it follows that $L' : P' : X' \in \Sigma$ for some $L' \subseteq L$ and $P' \subseteq P$. Since we know that the property graph G satisfies $L' : P' : X'$ it must be the case that $t_v(X') \neq t_{v'}(X')$. Consequently, r satisfies $X' \to R_P$. Since we know that the property graph G satisfies $L' : P' : X' \to Y$ the following must hold: if $t_v(X') = t_{v'}(X')$, then $t_v(Y') = t_{v'}(Y')$. Consequently, r satisfies $X' \to Y'$.

Suppose now that $\Sigma_{L:P} \models X \to Y$ does not hold. Then there is a two-tuple relation $r = \{t, t'\}$ over R_P that satisfies $\Sigma_{L:P}$ and violates $X \to Y$. We define a property graph $G = (V, Ed, \eta, \lambda, \nu)$ where $V = \{v_t, v_{t'}\}$, $Ed = \emptyset$, $\lambda(v_t) = L = \lambda(v_{t'})$. For every $A \in P$ we define $\nu(v_t, A) := t(A)$ and $\nu(v_{t'}, A) := t'(A)$. Clearly, the property graph G violates $L : P : X \to Y$. It remains to show that G satisfies all gUCs $L' : P' : X'$ and all gFDs $L' : P' : X' \to Y'$ in Σ. In the case where $L' \subseteq L$ and $P' \subseteq P$, it follows that $X' \to R_P$ and $X' \to Y' \in \Sigma_{L:P}$. Hence, since r satisfies $X' \to R_P$ and $X' \to Y'$, it follows that G satisfies $L' : P' : X'$ and $L' : P' : X' \to Y'$. Note, in particular, that r contains two distinct tuples, so there is some attribute $B \in R_P$ such that $t(B) \neq t'(B)$ holds. Consequently, we must actually have $\nu(v_t, A) \neq \nu(v_{t'}, A)$ for some $A \in X'$. In the other case we have $L' \not\subseteq L$ or $P' \not\subseteq P$. In this case, by definition, any gUC $L' : P' : X'$ and any gFD $L' : P' : X' \to Y'$ would be satisfied since there are no nodes that apply to them. Hence, we have just shown that $\Sigma \not\models L : P : X \to Y$.

2. Suppose $\Sigma \not\models L : P : X$. Then there is a property graph $G = (V, Ed, \eta, \lambda, \nu)$ that satisfies all gUCs and gFDs in Σ but violates $L : P : X$. In particular, there are two vertices $v, v' \in V$ whose label sets include all labels in L, both vertices v, v' carry all properties in P, and for all properties $A \in X$, the values $\nu(v, A)$ and $\nu(v', A)$ are matching. We now define the following two-tuple relation $r := \{t_v, t_{v'}\}$ over $R_P = P \cup \{A_0\}$ as follows $t_v(A) := \nu(v, A)$ for all $A \in P$ and $t_v(A_0) := 0$, and $t_{v'}(A) := \nu(v', A)$ for all $A \in P$ and $t_{v'}(A_0) := 1$. It follows that r violates $X \to R_P$ since $XY \subseteq P$. It remains to show that r satisfies all $X' \to R_P$ and $X' \to Y' \in \Sigma_{L:P}$. For $X' \to R_P \in \Sigma_{L:E}$ it follows that $L' : P' : X' \in \Sigma$ for some $L' \subseteq L$ and $P' \subseteq P$. Since we know that the property graph G satisfies $L' : P' : X'$ it must be the case that $t_v(X') \neq t_{v'}(X')$. Consequently, r satisfies $X' \to R_P$. Since we know that G satisfies $L' : P' : X' \to Y$ the following must hold: if $t_v(X') = t_{v'}(X')$, then $t_v(Y') = t_{v'}(Y')$. Consequently, r satisfies $X' \to Y'$.

Suppose now that $\Sigma_{L:P} \models X \to R_P$ does not hold. Then there is a two-tuple relation $r = \{t, t'\}$ over R_P that satisfies $\Sigma_{L:P}$ and violates $X \to R_P$. We define a property graph $G = (V, Ed, \eta, \lambda, \nu)$ with $V = \{v_t, v_{t'}\}$, $Ed = \emptyset$, and $\lambda(v_t) = L = \lambda(v_{t'})$. For every $A \in P$ we define $\nu(v_t, A) := t(A)$ and $\nu(v_{t'}, A) := t'(A)$. Since r violates $X \to R_P$, it follows that $t(X) = t'(X)$. Hence, the property graph G violates $L : P : X$. It remains to show that G satisfies all gUCs $L' : P' : X'$ and all gFDs $L' : P' : X' \to Y'$ in Σ. In the case where $L' \subseteq L$ and $P' \subseteq P$, it follows that $X' \to R_P$ and $X' \to Y' \in \Sigma_{L:P}$. Hence, since r satisfies $X' \to R_P$ and $X' \to Y'$, it follows that G

satisfies $L' : P' : X'$ and $L' : P' : X' \to Y'$. Note, in particular, that r contains two distinct tuples, so there is some attribute $B \in R_P$ such that $t(B) \neq t'(B)$ holds. Consequently, we must actually have $\nu(v_t, A) \neq \nu(v_{t'}, A)$ for some $A \in X'$. In the other case we have $L' \not\subseteq L$ or $P' \not\subseteq P$. In this case, by definition, any gUC $L' : P' : X'$ and any gFD $L' : P' : X' \to Y'$ would be satisfied since there are no nodes that apply to them. Hence, we have just shown that $\Sigma \not\models L : P : X$. □

A.2 Normal Forms

Theorem A.5 (Theorem 5.8 restated). *For every label set L and property set P, it holds that Σ is in L:P-BCNF if and only if $(R_P, \Sigma_{L:P})$ is in BCNF.*

Proof. By Theorem 5.3(1) we have $L{:}P{:}X \to Y \in \Sigma^+_{\mathfrak{E}}$ if and only if $X \to Y \in (\Sigma_{L:P})^+_{\mathfrak{A}}$, and by Theorem 5.3(2) we have $L{:}P{:}X \in \Sigma^+_{\mathfrak{E}}$ if and only if $X \to R_P \in (\Sigma_{L:P})^+_{\mathfrak{A}}$. Hence, Σ is in L:P-BCNF if and only if for every FD $X \to Y \in (\Sigma_{L:P})^+_{\mathfrak{A}}$ it is true that $Y \subseteq X$ or $X \to R_P \in (\Sigma_{L:P})^+_{\mathfrak{A}}$. This, however, means that $(R_P, \Sigma_{L:P})$ is in Boyce-Codd Normal Form. □

Theorem A.6 (Theorem 5.9 restated). *Σ in L:P-BCNF iff for every gFD $L'{:}P'{:}X \to Y \in \Sigma$ where $L' \subseteq L$ and $P' \subseteq P$, $Y \subseteq X$ or $L{:}P{:}X \in \Sigma^+_{\mathfrak{E}}$.*

Proof. By Theorem 5.8 it follows that Σ in L:P-BCNF if and only if for every FD $X \to Y \in (\Sigma_{L:P})^+_{\mathfrak{A}}$ it is true that $Y \subseteq X$ or $X \to R_P \in (\Sigma_{L:P})^+_{\mathfrak{A}}$. Since $(R_P, \Sigma_{L:P})$ is in BCNF if and only if for every FD $X \to Y \in \Sigma_{L:P}$ it is true that $Y \subseteq X$ or $X \to R_P \in (\Sigma_{L:P})^+_{\mathfrak{A}}$, and $X \to Y \in \Sigma_{L:P}$ if and only if there is some $L'{:}P'{:}X \to Y \in \Sigma$ such that $L' \subseteq L$ and $P' \subseteq P$, we conclude that Σ in L:P-BCNF if and only if for every gFD $L'{:}P'{:}X \to Y \in \Sigma$ where $L' \subseteq L$ and $P' \subseteq P$ it is true that $Y \subseteq X$ or $L{:}P{:}X \in \Sigma^+_{\mathfrak{E}}$. □

Theorem A.7 (Theorem 5.12 restated). *For every label set L and property set P it holds that Σ is in L:P-3NF if and only if $(R_P, \Sigma_{L:P})$ is in 3NF.*

Proof. The proof is similar to the proof of Theorem 5.8 and uses the fact that a property $A \in P$ is L:P-prime for Σ if and only if A is prime for $\Sigma_{L:P}$. □

Theorem A.8 (Theorem 5.16 restated). *For all sets Σ of gUC/FDs, for all label sets L and property sets P, we have Σ is in L:P-RFNF iff $(R_P, \Sigma_{L:P})$ is in RFNF.*

Proof. We show first: if Σ is not in L:P-RFNF, then $(R_P, \Sigma_{L:P})$ is not in RFNF. By hypothesis, there is some property graph G that satisfies Σ, some node $v \in V_{L:P}$, and some property $A \in P$ such that $\nu(v, A)$ is L:P-redundant for Σ. Hence, for every L:P-replacement G' of v on A, G' violates some gUC $L{:}P{:}X$ or gFD $L{:}P{:}X \to Y$ in $\Sigma^+_{\mathfrak{E}}$ such that $L' \subseteq L$ and $P' \subseteq P$. Consequently, there must be some node $v' \in V_{L:P}$ such that $v \neq v'$, $\nu(v, XA) = \nu(v', XA)$ and $A \in Y - X$ for some non-trivial gFD $L'{:}P'{:}X' \to Y' \in \Sigma$ with $L' \subseteq L$ and $P' \subseteq P$. In particular, $X' \to Y' \in \Sigma_{L:P}$. Let $r := \{t_v, t_{v'}\}$ be a relation over R_P such that for all $B \in R_P$, $t_v(B) = t_{v'}(B)$ iff $B \in (X')^+_{\Sigma_{L:P}}$. Note that $R_P - (X')^+_{\Sigma_{L:P}}$ is non-empty as otherwise $\Sigma_{L:P}$ would imply $X' \to R_P$, which means that $L{:}P{:}X'$ would be implied by Σ, and the latter would imply that $v = v'$ since $\nu(v, X') = \nu(v', X')$. Since $R_P - (X')^+_{\Sigma_{L:P}}$ is non-empty, $t_v \neq t_{v'}$. Furthermore, r satisfies $\Sigma_{L:P}$ for the following reasons. Suppose $t_v(X) = t_{v'}(X)$ for some FD $X \to Y \in \Sigma_{L:P}$. Then $X \subseteq (X')^+_{\Sigma_{L:P}}$, and $X' \to X$ is implied by $\Sigma_{L:P}$. Hence, $X' \to Y$ is implied by $\Sigma_{L:P}$, too. Consequently, $Y \subseteq (X')^+_{\Sigma_{L:P}}$ and $t_v(Y) = t_{v'}(Y)$ holds, too. Hence, $t_v(A)$ in r is redundant for $\Sigma_{L:P}$ since for every replacement $\bar{t}_v$ of t_v on A, $\bar{r} := (r - \{t_v\}) \cup \{\bar{t}_v\}$ violates the FD $X' \to Y'$ in $\Sigma_{L:P}$. Hence, $(R_P, \Sigma_{L:P})$ is not in RFNF.

We now show: if $(R_P, \Sigma_{L:P})$ is not in RFNF, then Σ is not in L:P-RFNF for Σ. By hypothesis, there is some relation r over R_P that satisfies $\Sigma_{L:P}$, some tuple $t \in r$, and attribute $A \in R_P$ such that the data value occurrence $t(A)$ is redundant for $\Sigma_{L:P}$. That is, for every replacement $\bar{t}$ of t on A, $\bar{r} := (r - \{t\}) \cup \{\bar{t}\}$ violates some constraint in $\Sigma_{L:P}$. Hence, there is some tuple $t' \in r$ such that $t \neq t'$, $t(X'A) = t'(X'A)$ and $A \in Y' - X'$ for some FD $X' \to Y' \in \Sigma_{L:P}$. In particular, $Y' \subset R_P$ as otherwise $t = t'$. Hence, there is some $L'{:}P'{:}X' \to Y' \in \Sigma$ such that $L' \subseteq L$ and $P' \subseteq P$. In particular, $A \in Y' \subseteq P' \subseteq P$. Let $G^{t,t'}$ be defined as the following property graph: $V = \{v_t, v_{t'}\}$ with $\mu(v_t) \neq \mu(v_{t'})$, $\lambda(v_t) = L = \lambda_{v_{t'}}$, $Ed = \emptyset$, and $\nu(v_t, A) := t(A)$ and $\nu(v_{t'}, A) := t'(A)$ for all $A \in P$. It follows that $G^{t,t'}$ satisfies every gUC $L''{:}P''{:}X''$ and every gFD $L''{:}P''{:}X'' \to Y''$ in Σ. Indeed, if $L'' \not\subseteq L$ or $P'' \not\subseteq P$, then $G_{L'':P''} = \emptyset$ and the gUCs and gFDs above are satisfied. Otherwise $L'' \subseteq L$ and $P'' \subseteq P$, and then $G_{L'':P''} = G$ satisfies X'' and $X'' \to Y''$ because $\{t, t'\}$ satisfies $X'' \to R_P$ and $X'' \to Y''$ in $\Sigma_{L:P}$. Hence, there are a property graph $G^{t,t'}$ that satisfies Σ, a node $v_t \in V_{L:P}$ in $G^{t,t'}$, and a property $A \in P$ such that $\nu(v_t, A)$ is L:P-redundant for Σ. Hence, Σ is not in L:P-RFNF. □

Corollary A.9 (Corollary 5.17 restated). *For all sets Σ of gUC/FDs, for all label sets L and property sets P, we have Σ is in L:P-RFNF iff Σ is in L:P-BCNF.*

Proof. Theorem 5.8 has shown that Σ is in L:P-BCNF iff $(R_P, \Sigma_{L:P})$ is in BCNF. However, we know that classical BCNF captures classical BCNF, that is, $(R_P, \Sigma_{L:P})$ is in BCNF iff $(R_P, \Sigma_{L:P})$ is in RFNF. However, the latter holds iff Σ is in L:P-RFNF by Theorem 5.16. □

A.3 Normalization

Theorem A.10 (Theorem 5.19 restated). *On input $((G, \Sigma), L \cup \{\ell\}, P)$ such that G satisfies Σ, Algorithm 2 returns the property graph $G^{\ell}_{L:P}[\mathcal{D}]$ that satisfies $\Sigma^{\ell}_{L:P}[\mathcal{D}]$, which is a lossless, dependency-preserving L:P-decomposition of Σ into Third Normal Form that is in Boyce-Codd Normal Form whenever possible.*

Proof. Lines 1-16 ensure that a lossless, dependency-preserving decomposition $\mathcal{D}$ of $(R_P, \Sigma[L{:}P])$ into 3NF is returned that is in BCNF whenever possible. $\Sigma^{\ell}_{L:P}[\mathcal{D}]$ is always lossless, and $\bigcup_{S \in \mathcal{D}} \left(\Sigma^{\ell}_{L:P}[\mathcal{D}]\right)_{\ell_S:S}$ is a cover of $\Sigma_{L:P}$ by definition of $\Sigma^{\ell}_{L:P}[\mathcal{D}]$. Finally, for every $S \in \mathcal{D}$, it is true that $\Sigma^{\ell}_{L:P}[\mathcal{D}]$ is in ℓ_S:S-3NF (or -BCNF, respectively) if and only if $(S, \Sigma_{L:P}[S])$ is in 3NF (BCNF). Hence, the result follows by construction. □