Normalizing Property Graphs

Philipp Skavantzos
The University of Auckland
Auckland, New Zealand
philipp.skavantzos@auckland.ac.nz

Sebastian Link
The University of Auckland
Auckland, New Zealand
s.link@auckland.ac.nz

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.

PVLDB Reference Format:
Philipp Skavantzos and Sebastian Link. Normalizing Property Graphs. PVLDB, ?(?): XXX-XXX, 2023.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/GraphDatabaseExperiments/normalization_experiments.

1 INTRODUCTION

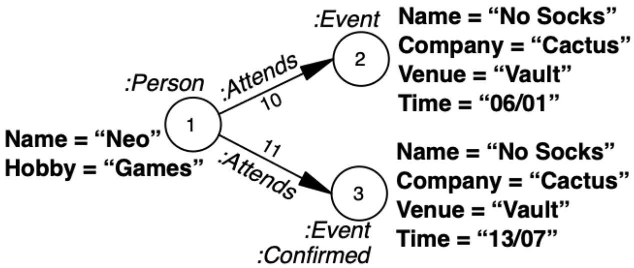
Normalization minimizes sources of potential data inconsistency and costs of integrity maintenance incurred by updates of redundant data. Based on classes of data dependencies that cause redundancy, classical normalization transforms schemata into normal forms where these dependencies can be enforced by keys only, or come close to it. For example, this is achieved by Boyce-Codd Normal Form (BCNF) for functional dependencies (FDs) [10], Fourth Normal Form for multivalued dependencies [13], Fifth Normal Form for join dependencies [35], Inclusion Dependency Normal Form for functional and inclusion dependencies [24], and Domain-Key Normal Form [14] more generally. Third Normal Form (3NF) minimizes sources of data redundancy under the additional target of enforcing all FDs without joining relation schemata [6, 22], and Bounded Cardinality Normal Form minimizes the level of data redundancy caused by FDs [26]. Some achievements carry forward to richer data formats, including SQL [21] and models with missing data [23, 38], Nested [30, 36], Object-Oriented [34], Temporal [20], Web [2, 29], and Uncertain Databases [25].

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 [15]. Indeed, Fan [15] 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 property graphs, 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 normalizing property graphs can be summarized as follows:

(1) We introduce uniqueness constraints and functional dependencies as declarative means to i) express completeness, integrity, and

Proceedings of the VLDB Endowment, Vol. ?, No. ? ISSN 2150-8097.
doi:XX.XX/XXX.XX

Figure 1: Original Property Graph



uniqueness requirements in the form of business rules that govern property graph data, and ii) form the source of redundant property values that drive goals for graph normalization.
(2) We show that our graph dependencies facilitate normalization as their implication problem can be captured axiomatically finitely by Horn rules, and algorithmically by a linear-time decision algorithm.
(3) We normalize property graphs into lossless, dependency-preserving BCNF whenever possible, and guarantee 3NF in general. Normalization is tailored to nodes with labels and properties the target application requires. Unlike the relational case, our normalization can even be applied when FDs do not fully hold. Indeed, our normalization is still lossless and removes redundancy in that part conforming to the FDs.
(4) We demonstrate the extent and benefits of property 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 work on graph data normalization, including normal forms, the study of more expressive graph dependencies, relationships to 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. Section 4 contains a brief guide of concepts and notation. The semantics of property graphs and constraints is given in Section 5. In Section 6, we introduce our normalization framework, including the implication problem, normal forms and normalization. Experimental results that qualify and quantify the need and benefits of normalization are discussed in Section 7. We conclude and briefly comment on future work in Section 8. More details, including the full paper, are available at the Artifact URL.

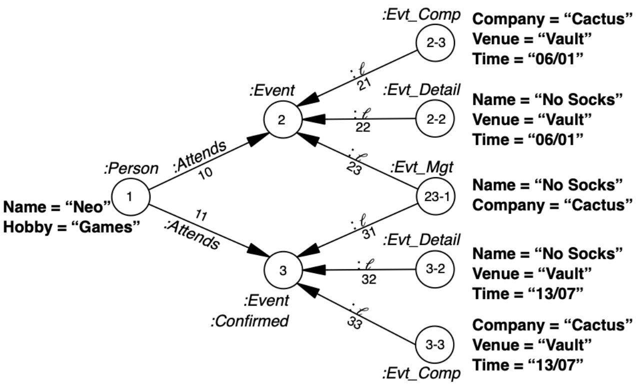
2 ILLUSTRATIVE EXAMPLE

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

The property graph G_0 in Figure 1 models an application scenario where people attend events. Nodes and edges carry any number of labels, and may carry pairs of properties and values. In Figure 1 we have nodes labeled by *Person*, *Event*, and *Confirmed*. The latter label assures that *Event* nodes model events that have been confirmed. We also have edges from Person to (confirmed) Event nodes labeled by *Attends*, expressing that a person attends a (confirmed) 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 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 side have also matching values on all properties of the right side. However, characteristics of property graphs motivate additional features of graph dependencies. Firstly, properties may not exist on some nodes. Secondly, dependencies apply to different types of nodes. As a consequence, we want to provide data stewards with the ability to tailor uniqueness constraints and functional dependencies to i) completeness requirements for properties, and ii) the labels carried by nodes. For i), we take the principled approach that missing properties should not have an impact on the validity of constraints [38]. Hence, graph-tailored UCs (gUC) and FDs (gFD) feature a property set P that restricts the set of vertices on which the constraint holds to those nodes for which all properties in P exist, and P contains at least all the properties that occur in the constraint. Moreover, a label set L is included in the specification of gUCs and gFDs that further restricts the set of vertices on which the constraint holds to those nodes which carry all the labels in L. We obtain the following constraints in our example: the gUC $Event{:}CT{:}CT$ (σ_1), stipulating that all *Event* nodes that have properties C and T, are uniquely identified by the combination of values on these properties. The gFD $Event{:}NC{:}N \rightarrow C$ (σ_2) stipulates that *Event* nodes with properties N and C have matching values on C whenever they have matching values on N. Similarly, the gFDs $Event{:}NTV{:}NT \rightarrow V$ (σ_3) and $Event{:}NTV{:}TV \rightarrow N$ (σ_4) express that *Event* nodes with properties N, T, and V have matching values on V (N, respectively) whenever they have matching values on N and T (T and V, respectively). Finally, the gFD $Event, Confirmed : NCTV : NC \rightarrow T$ (σ_c) expresses that nodes with both labels *Event* and *Confirmed*, and with properties N, C, T, and V have matching values on T whenever they have matching values on N and C. The constraints exhibit non-trivial interactions. For instance, $\Sigma = \{\sigma_1, \ldots, \sigma_4\}$ implies $\sigma_5 = Event{:}NCTV{:}TV$, and every gUC $L{:}P{:}X$ implies the gFD $L{:}P{:}X \rightarrow P$, but not vice versa. For instance, G_0 satisfies the gFD $Event{:}NCV{:}NC \rightarrow V$, but not the gUC $Event{:}NCV{:}NC$. In particular, no gUC can be expressed by any set of gFDs. As another example, G_0 satisfies σ_c, in particular, but not the gFD $Event{:}NCTV{:}NC \rightarrow T$. This illustrates the subtlety of reasoning with multiple labels. Similarly, if G_0' results from G_0 by adding label *Confirmed* to node 2 and removing property *Venue* from node 2, G_0' would still satisfy the gFD σ_c but not the gFD $Event, Confirmed{:}NCT{:}NC \rightarrow T$ resulting from σ_c by removing property V from $P = NCTV$.

Figure 2: Normalized Property Graph



We observe that every graph satisfying $\Sigma = \{\sigma_1, \ldots, \sigma_4, \sigma_c\}$, such as G_0, cannot exhibit any redundant property values on any nodes that carry both labels *Event*, *Confirmed* and all properties N, C, T, V. Interestingly, however, G_0 is one property graph that does exhibit a redundant data value occurrence on nodes that carry only label *Event* and all properties $NCTV$. Indeed, each occurrence of company *Cactus* is redundant due to σ_2: If one occurrence of *Cactus* is changed to *any* different value, then σ_2 will be violated.

While G_0 is in BCNF for target label set $\{Event, Confirmed\}$ and property set $\{N, C, T, V\}$, and in 3NF for target label set $\{Event\}$ and property set $\{N, C, T, V\}$, it is not in BCNF for label set $\{Event\}$ and property set $\{N, C, T, V\}$. Figure 2 shows the normalized property graph G_n resulting from a non-obvious decomposition. Indeed, G_n is in BCNF tailored to the application requirements that decompose *Event* nodes exhibiting **all** properties in $NCTV$. The decomposition is lossless, as a simple join via ℓ-labeled edges would restore the original graph G_0. It is also dependency-preserving: gUC σ_1 adds gUC $\sigma_1' = Evt\_Comp{:}CVT{:}CT$, and gFDs $\sigma_2, \ldots \sigma_5$ result in gUCs $\sigma_2' = Evt\_Mgt{:}NC{:}N$, $\sigma_3' = Evt\_Detail{:}NTV{:}NT$, $\sigma_4' = Evt\_Detail{:}NTV{:}TV$, and even $\sigma_5' = Evt\_Comp{:}CVT{:}TV$. Note that σ_c is beyond scope of any decomposition not targeted at nodes with label *Confirmed*. The additional properties in the new constraints, such as V in σ_1', originate from the requirement for *Event* nodes to exhibit all properties $NCTV$. The new design eliminates all data redundancy caused by σ_2, for example in G_0. The fact that *Cactus* manages the event *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 σ_2', expressing that there cannot be two different *Evt_Mgt* nodes with properties N, C and matching values on N. Omitting the two *Evt_Comp*-nodes and their outgoing edges from G_n would still result in a lossless BCNF decomposition. However, the gUC σ_1 would be lost. In other words, the *Evt_Comp*-nodes ensure G_n is dependency-preserving.

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

3 PREVIOUS WORK

Normalization is a classical topic [27], but no framework exists for graph data yet. Normal forms characterize well-designed databases that only admit instances with no redundant data value caused by any dependency in the class considered. Fundamental are efficient solutions to the implication problem: BCNF and 3NF are founded on Armstrong's axioms ($\mathfrak{A}$) [3] and linear decision algorithms [5]. The latter drive decompositions into BCNF and 3NF [7].

Schema design for other data quality dimensions is in its infancy [4]. Completeness-tailored UCs and FDs were introduced for relations with missing values, and a normalization framework established that tailors relational design to completeness and integrity requirements [38]. Our present work extends this approach to property graphs with fundamental differences: in contrast to [38] we cannot assume an underlying schema, we deal with graphs rather than relations, and we require an extension to accommodate labels.

Recently, much attention has been given to graph query languages, but several lines of work on integrity management have emerged, too. The comprehensive key proposal [12] sets out an expressive framework for specifying keys on nodes, edges, and properties. In particular, the gUC $\{L_1, \ldots, L_m\}{:}\{P_1, \ldots, P_n\}{:}\{U_1, \ldots, U_k\}$ can be specified as the exclusive PG-key below.

FOR $x{:}L_1 \cdots{:}L_m$ WHERE $x.P_1$ ... AND $x.P_n$ IS NOT NULL EXCLUSIVE $x.U_1,\ldots,x.U_k$

While expressive and flexible, there are no technical results for PG-keys yet. Our class of gUCs was proposed in [33] to address the lack of constraints for data quality dimensions. They form a sub-class of PG-keys that enjoys good computational properties.

The work in [17, 18, 32] defines expressive graph dependencies, including FDs that compare values of properties or constants for all pairs of entities identified by a graph pattern. Their expressiveness is different from gFDs which allow multiple labels and require all properties in P to exist. While implication for FDs in [18] is *NP*-complete and they target entity resolution and fraud detection, implication for gFDs is decidable in linear time and they target normalization. Graph dependencies provide a rare opportunity to specify application semantics in graph databases [12, 17, 18, 32, 33].

In contrast to normalization, [37] propose a schema design framework for graph data that is based on minimizing access to data that will likely co-occur in query results while keeping independent concepts separate. In contrast to normalization that is based on data dependencies, [37] requires a conceptual schema as input.

Schema information is beneficial for data management, including (property) graphs [1, 28]. Schemata may interact non-trivially with dependencies, such as PG-keys or gFDs. This motivates further research, including the normalization of graph schemata.

Already Codd [9] suggests online and a-posteriori enforcement, where integrity is preserved either as part of every update, or inconsistencies are reported casually, respectively. Full normalization with our framework supports online enforcement while casual normalization materializes a-posteriori enforcement, and both can be balanced by tailoring normalization to target dependencies.

Our work is the first to address normalization for property graphs. The class of gFDs is new, and results on the combined implication for gUC/gFDs encompass simpler findings for gUCs alone. In our work, we transfer state-of-the-art normalization for FDs from relational to graph databases.

Table 1: Summary of Concepts & Notation To Be Introduced

Concept	*Notation*
Basic concepts for graphs and graph constraints:	
Property graph	$G = (V, Ed, \eta, \lambda, \nu)$
Label set/Property set	L/P
Property subsets	$X, Y \subseteq P$
gFD	$L : P : X \to Y$
gUC	$L : P : X$
gUC/gFD set	Σ
Translation into relational framework:	
$L : P$-FD set for Σ	$\Sigma_{L:P}$ (classical FDs originating from Σ)
Relation schema of P	$R_P := P \cup \{A_0\}$ with fresh attribute A_0
Decomposition of R_P	$\mathcal{D} = \{S \mid S \subseteq R_P\}$ where $\bigcup_{S\in\mathcal{D}} = R_P$
Projection of $\Sigma_{L:P}$ onto $S \subseteq R_P$	$\Sigma_{L:P}[S] = \{X \to Y \in \Sigma^+_{L:P} \mid XY \subseteq S\}$
Normal forms for property graphs and their achievements:	
Σ in $L : P$-BCNF/3NF/RFNF	$(R_P, \Sigma_{L:P})$ in BCNF/3NF/RFNF
Normalization of property graphs:	
$L : P$-decomposition of Σ wrt $\mathcal{D}$	$\Sigma^{\ell}_{L:P}[\mathcal{D}]$ with fresh edge label ℓ
$L : P$-projection of G onto $S \subseteq R_P$	$G^{\ell}_{L:P}[S]$ with fresh edge label ℓ
S-equivalence between nodes v, v'	$v \equiv_S v'$ (values match on properties in S)
$L : P$-decomposition of G wrt $\mathcal{D}$	$\bigcup_{S\in\mathcal{D}} (G^{\ell}_{L:P}[S]/\equiv_S)$

4 GUIDE FOR CONCEPTS AND NOTATION

Table 1 provides a brief outline which concepts and notation we will develop throughout. In Sec. 5 we will repeat concepts for property graphs, and introduce graph-tailored constraints called gFDs and gUCs. Our approach will enable us to translate graph constraints into classical FDs in Sec. 6.1, to take advantage of the existing theory. This will enable us to define BCNF and 3NF for property graphs in Sec. 6.2. In Sec. 6.3, we will transfer achievements for BCNF and 3NF from relational databases to property graphs, and establish a framework for normalizing property graphs in Sec. 6.4.

5 GRAPH-TAILORED CONSTRAINTS

We recall basics of property graphs, including gUCs [33]. We then introduce gFDs and illustrate them on our running example.

The *property graph model* [8] is based on the following disjoint sets: $\mathcal{O}$ for a set of objects, $\mathcal{L}$ for a finite set of labels, $\mathcal{K}$ for a set of properties, and $\mathcal{N}$ for 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*, $\eta : Ed \to V \times V$ is a function assigning to each edge an ordered pair of vertices, $\lambda : V \cup Ed \to \mathcal{P}(\mathcal{L})$ is a function assigning to each object a finite set of labels, and $\nu : (V \cup Ed) \times \mathcal{K} \to \mathcal{N}$ 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 (gUCs) were introduced in [33], and cover UCs used by Neo4j [19] as a special case. For define the subset $V_L \subseteq V$ of vertices in a property graph that carry all labels of the given set $L \subseteq \mathcal{L}$ as follows: $V_L = \{v \in V \mid L \subseteq \lambda(v)\}$.

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 gUCs L:P:X where $L = \{\ell\}$ and $P = X = \{p\}$, that is, $\{\ell\}$:$\{p\}$:$\{p\}$. Hence, we denoted them by ℓ:p. Neo4j's composite indices are covered as the special case where $L = \{\ell\}$ and $P = X$, that is, $\{\ell\}$:X:X. Hence, we denote them by ℓ:X.

We will now introduce *graph-tailored FDs*. Intuitively, they express that 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 5.1. A *graph-tailored functional dependency* (or *gFD*) over $\mathcal{L}$ and $\mathcal{K}$ is an expression $L{:}P{:}X \to Y$ where $L \subseteq \mathcal{L}$ and $X, Y \subseteq P \subseteq \mathcal{K}$. For a property graph $G = (V, Ed, \eta, \lambda, \nu)$ over $\mathcal{O}$, $\mathcal{L}$, $\mathcal{K}$, $\mathcal{N}$, we say G *satisfies* the gFD $L{:}P{:}X \to Y$ over $\mathcal{L}$ and $\mathcal{K}$, denoted by $\models_G L{:}P{:}X \to Y$, 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 $A \in Y$, $\nu(v_1, A) \neq \nu(v_2, A)$. □

The concept of gFDs provides users with the flexibility to layer rules for nodes with different sets of labels. While $L{:}P{:}X \to Y$ 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. Unless a node exhibits values on all properties in P, it does not need to comply with the FD $X \to Y$. Thirdly, 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 $L{:}XY{:}X \to Y$. Most gFDs that express meaningful rules will have this format, and they imply weaker gFDs $L{:}P{:}X \to Y$ where P contains XY. This has multiple benefits as illustrated later, such as tailoring normalization to different requirements, discovering gFDs and sources of inconsistent data from property graphs.

Over relation schema R, the UC X can be expressed by the FD $X \to R$. Indeed, 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 cause data redundancy has been transformed into a key which cannot cause data redundancy.

This situation is different in property graphs that permit duplication. Indeed, no gUC L:P:X can be expressed by any gFD since we can always have two different nodes with label set L and matching values on all properties in P. While this graph satisfies the gFD $L{:}P{:}X \to P$, it does not satisfy the gUC L:P:X. For example, graph G_0 in Figure 1 satisfies the gFD $Event{:}NCV{:}NC \to V$ but not the gUC $Event{:}NCV{:}NC$. 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.

6 NORMALIZATION FRAMEWORK

We will first establish axiomatic and algorithmic characterizations of the implication problem for gUCs and gFDs. We will then introduce 3NF and BCNF for property graphs tailored to labels and properties as required by applications. We will show that our normal forms minimize (eliminate) data redundancy. Finally, we will 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.

Table 2: Axiomatization $\mathfrak{E} = \{\mathcal{R}, \mathcal{E}, \mathcal{T}, \mathcal{A}, \mathcal{W}, \mathcal{P}\}$ of gUC/FDs

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

6.1 Reasoning

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 O, $\mathcal{L}$ and $\mathcal{K}$ that satisfies all constraints in Σ also satisfies φ.

Deciding whether φ is implied by Σ is fundamental for node integrity management on property graphs. If φ is implied by Σ, then φ is already specified implicitly by Σ. Otherwise, failure to specify φ explicitly may result in integrity faults that go undetected. The implication problem for FDs in relational databases is complete for *PTIME* [5, 11]. Since FDs form a special case of gFDs, the implication problem on property graphs is *PTIME*-hard.

6.1.1 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. We aim at computing Σ^*_C by applying *inference rules* of the form $\dfrac{\text{premise}}{\text{conclusion}}$. 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.

We assume the rules of $\mathfrak{E}$ in Table 2 contain well-formed gUCs and gFDs. As example, for the rule $\mathcal{A}$ with $L{:}P{:}X$ and $LL'{:}PP'{:}XX'$ we assume $X \subseteq P$ and $XX' \subseteq PP'$. $\mathcal{A}$ by itself is sound and complete for the implication of gUCs. The full version shows that $\mathfrak{E}$ is sound and complete for the implication of gUCs and gFDs. The soundness is established by contra-position: assume some property graph violates the conclusion of a rule, one shows that some premise of the rule must be violated as well. The completeness proof constructs for any given gUC $L{:}P{:}X$ and gFD $L{:}P{:}X \to Y$ that cannot be inferred from Σ by $\mathfrak{E}$, a property graph that satisfies Σ and violates the given gUC or 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 $L{:}P{:}X \to A \in \Sigma^+_{\mathfrak{E}}$.

Algorithm 1 Implication of gUCs and gFDs

Require: Set $\Sigma \cup \{\varphi\}$ of gUC/FDs; $\varphi = L{:}P{:}X$ or $\varphi = L{:}P{:}X \to Y$
Ensure: *TRUE*, if $\Sigma \models \varphi$, and *FALSE*, otherwise

```
Compute X⁺_{Σ_{L:P}} by linear-time attribute set closure for FDs [5]
if φ = L:P:X and X⁺_{Σ_{L:P}} = R_P then
    return TRUE
else if φ = L:P:X → Y and Y ⊆ X⁺_{Σ_{L:P}} then
    return TRUE
else
    return FALSE
```

Theorem 6.1. *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 6.2. Let Σ contain $\phi'_1 = Event{:}CTV{:}CT \to V$ and $\phi'_2 = Event{:}NTV{:}VT \to N$. Applying ($\mathcal{E}$) to ϕ'_1 gives us $\phi'_3 = Event{:}CTV{:}CT \to CTV$. ($\mathcal{R}$) gives us $\phi'_4 = Event{:}CTV{:}CTV \to VT$, and applying ($\mathcal{T}$) to ϕ'_4 and ϕ'_2 gives us $\phi'_5 = Event{:}CTVN{:}CTV \to N$. Finally, applying ($\mathcal{T}$) to ϕ'_3 and ϕ'_5 gives us $\varphi = Event{:}CNTV{:}CT \to N$. Hence, Σ implies φ. Note the subtlety in reasoning with the requirements for properties. As we will see below, Σ does not imply $\varphi' = Event{:}CNT{:}CT \to N$.

$\{\mathcal{R}, \mathcal{E}, \mathcal{T}\}$ forms an axiomatization for gFDs, a natural extension of the Armstrong axioms [3]. We will denote the latter by $\mathfrak{A}$.

6.1.2 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 $R_P = P \cup \{A_0\}$:

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

$A_0 \notin P$ is a fresh property not occurring elsewhere. 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 FDs over relation schemata.

Theorem 6.3. *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 $\Sigma_{L:P} \models X \to Y$, and (2) $\Sigma \models L{:}P{:}X$ if and only if $\Sigma_{L:P} \models X \to R_P$* □

Theorem 6.3 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 [5]. The decision branches in Algorithm 1 reflect the characterization by Theorem 6.3. Hence, *PTIME*-completeness carries over from the classical case [5, 11].

Corollary 6.4. *Algorithm 1 decides the* PTIME*-complete implication problem for gUCs and gFDs in linear input time.* □

We illustrate the algorithm on our running example.

Example 6.5. Let $\Sigma = \{\phi'_1, \phi'_2\}$ and φ' from Example 6.2. Hence, $\Sigma_{Event:CNT} = \emptyset$ and $N \notin (CT)^+_{\Sigma_{Event:CNT}} = CT$, which means that Algorithm 1 returns a negative answer. However, for $\Sigma_{Event:CNTV} = \{CT \to V, VT \to N\}$, such that for φ we get $N \in (CT)^+_{\Sigma_{Event:CNTV}} = CTVN$, and Algorithm 1 returns a positive answer.

6.2 Normal Forms for Property Graphs

We define BCNF and 3NF for property graphs. Based on opportunities that graph data provides, we first explain our approach, describe our proposals, and present results on their achievements.

6.2.1 Approach. Since property graphs have no schema, it is challenging to define classical normal forms for graph data. We address this challenge using our class of graph-tailored constraints. The flexibility of graph data provides further opportunities. Since applications target graph objects based on their labels and properties, we view these features as requirements: The application targets only 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 the targets. Hence, normalization becomes flexible and driven by application requirements.

6.2.2 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 (unable to ever cause any redundancy). We will now define BCNF for gUCs and gFDs, aimed at preventing redundant property values on graphs that satisfy the constraints.

Definition 6.6. (L:P-BCNF) 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 $L{:}P{:}X \to Y \in \Sigma^+_{\mathfrak{C}}$ it is true that $Y \subseteq X$ or $L{:}P{:}X \in \Sigma^+_{\mathfrak{C}}$. □

We illustrate the definition on our running example.

Example 6.7. Property graph G_0 from Figure 1 satisfies $\Sigma = \{\sigma_1, \ldots, \sigma_5\}$. 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_n from Figure 2 satisfies $\Sigma' = \{\sigma'_1, \ldots, \sigma'_5\}$ from Section 2, which 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 6.8. *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.* □

Following Example 6.7, Σ is not in *Event*:*NTV*-BCNF as $R_P = NTVA_0$ is not in BCNF for $\Sigma_{Event:NTV} = \{VT \to N, NT \to V\}$ ($A_0 \in R_P$ and $VT \to R_P \notin \Sigma^+_{Event:NTV}$). Σ is not in *Event*:*NCTV*-BCNF as $R_P = NCTV$ is not in BCNF for $\Sigma_{Event:NCTV} = \{N \to C, CT \to NV, NT \to V, VT \to N\}$ ($N \to R_P \notin \Sigma^+_{Event:NCTV}$).

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 $\Sigma^+_{\mathfrak{C}} = \Theta^+_{\mathfrak{C}}$, Σ is in L:P-BCNF iff Θ is in L:P-BCNF. This is due to Definition 6.6 that checks all gFDs in $\Sigma^+_{\mathfrak{C}}$, which may be exponential in Σ. We can show it suffices to check Σ itself, so testing L:P-BCNF is efficient.

Theorem 6.9. *Σ is 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 $L{:}P{:}X \in \Sigma^+_{\mathfrak{C}}$.* □

Theorem 6.9 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 $L'{:}P'{:}X \to Y \in \Sigma$ 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 [5].

Corollary 6.10. *The condition whether Σ is in L:P-BCNF can be checked in time quadratic in $|\Sigma|$.* □

6.2.3 3NF. While a lossless BCNF decomposition is always achievable, some FDs may be lost. These require a join of schemata resulting from the decomposition before their validity can be tested. As this is expensive, dependency-preservation is another goal of normalization. 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 key for $\Sigma_{L:P}$. If no key exists, there is no prime property.

Definition 6.11. 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 $L{:}P{:}X \to Y \in \Sigma^+_{\mathfrak{C}}$ 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 6.7 showed that Σ is not in *Event*:*NCTV*-BCNF. Due to gUC σ_1 we obtain gUCs *Event*:*NCTV*:*VT* and *Event*:*NCTV*:*NT* in $\Sigma^+_{\mathfrak{C}}$, which are *Event*:*NCTV*-minimal. Hence, every property in *NCTV* is *Event*:*NCTV*-prime, and 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 6.12. *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, the quadratic time required to validate 3NF for $\Sigma_{L:P}$ extends to L:P-3NF for Σ.

Theorem 6.13. *Σ in L:P-3NF 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 $L{:}P{:}X \in \Sigma^+_{\mathfrak{C}}$ or every property in $Y - X$ is L:P-prime.* □

Testing L:P-BCNF is efficient, but validating L:P-3NF is likely intractable as it is *NP*-complete to decide if a property is L:P-prime, already when $L = \emptyset$ and $P = R$ is a relation schema [5]. It is *coNP*-complete to decide if for Σ, $\Sigma_{L:P:S}$ is in L:P-BCNF where $S \subseteq P$ and $\Sigma_{L:P:S} = \{L'{:}P'{:}X \to Y \in \Sigma^+_{\mathfrak{C}} \mid L' \subseteq L \wedge P' \subseteq P \wedge XY \subseteq S \subseteq P\}$.

Theorem 6.14. *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.* □

6.3 Achievements of Normal Forms

We aim at minimizing sources of property values that may occur redundantly in graphs that satisfy the given gUCs and gFDs. We will now illustrate in which formal sense this is actually achieved.

Let v denote a node of property graph G that carries all labels in L and 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 changing value $\nu(v, A)$ to some different value. The 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 in $(\Sigma_{L:P})^+_{\mathfrak{A}}$.

Definition 6.15. 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.

Figure 3: Projections of G_0 onto $\mathcal{D} = \{NC, NVT, CVT\}$

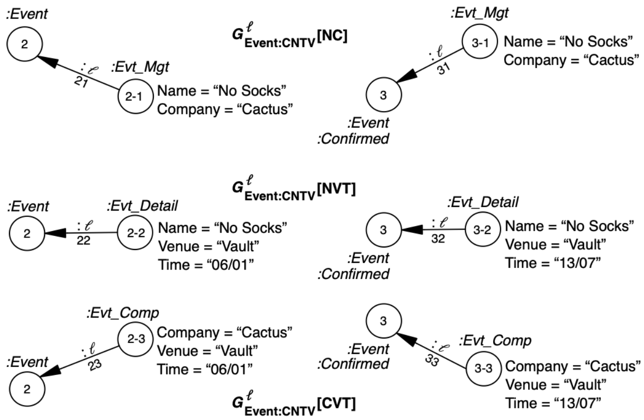
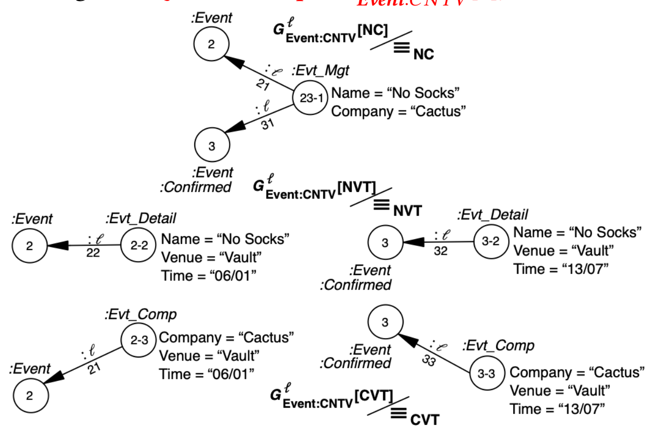


Figure 4: Quotient Graphs $G^{\ell}_{\textit{Event:CNTV}}[S]/{\equiv_S}$ of G_0



In graph G_0 of Figure 1, each occurrence $v(2, \text{Company})$ and $v(3, \text{Company})$ of "Cactus" is *Event*:$NCVT$-redundant. For instance, if G_0' results from G_0 by replacing $v(2, \text{Company})$ by a value different from "Cactus", G_0' will violate gFD *Event*:$NCVT$:$N \to C \in \Sigma^+_{\mathfrak{C}}$. Hence, Σ is not in *Event*:$NCVT$-RFNF. In contrast, the occurrence of $v(23-1, \text{Company})$ = "Cactus" in graph G_n of Figure 2 is not *Evt_Mgt*:NC-redundant. Indeed, Σ' is in *Evt_Mgt*:NC-RFNF.

Theorem 6.16. *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 6.16, the relation r corresponding to node set $V_{\textit{Event:NCTV}}$ of graph G_0 in Figure 1 is

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

and r satisfies $\Sigma_{\textit{Event:NCTV}} = \{N \to C, NT \to V, TV \to N, CT \to NV\}$, and each occurrence of "Cactus" is redundant. This example is representative that BCNF captures RFNF. Indeed, Theorem 6.16 lifts the result from relational databases to property graphs as targeted.

Corollary 6.17. *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.*

For relational databases, it is known that 3NF exhibits the fewest sources of data redundancy among all dependency-preserving decompositions [22]. Due to Theorem 6.12, these results carry over to L:P-3NF, pending our definitions below.

6.4 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.

6.4.1 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 $\mathcal{D}$ and the original graph G. 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 ℓ_S:S:X for each $X \to R_P \in \Sigma_{L:P}[S]$ and adding gFDs ℓ_S:S:$X \to Y$ for each $X \to Y \in \Sigma_{L:P}[S]$ for $Y \neq R_P$.

This construction can easily be inverted by collapsing all edges (v_S, v) labeled 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 6.7. The set Σ' is a lossless, dependency-preserving *Event*:$CNTV$-decomposition of Σ into BCNF. The property graph G_n in Figure 2 is the Event:$NCTV$-decomposition of G_0 in Figure 1, based on the decomposition $\mathcal{D} = \{NC, NVT, CVT\}$ of R_{NCTV}. Indeed, Figure 3 shows the three projections $G^{\ell}_{\textit{Event:CNTV}}[S]$ of $G_{\textit{Event:CNTV}}$ onto $S \in \mathcal{D}$ from step 1) of the process above, including new nodes 2-1 (=2_{NC}), 2-2(=2_{NVT}), 2-3(=2_{CVT}), 3-1(=3_{NC}), 3-2(=3_{NVT}) and 3-3(=3_{CVT}), with node labels ℓ_{NC} = *:Evt_Mgt*, ℓ_{NVT} = *:Evt_Details* and ℓ_{CVT} = *:Evt_Comp*, and directed edges 21=(2-1,2), 22=(2-2,2), 23=(2-3,2), 31=(3-1,3), 32=(3-2,3), and 33=(3-3,3) with edge label ℓ.

Step 2) of the process is illustrated in Figure 4 where the quotient graphs of the projections are shown. Here, the only vertices identified are 2-1 and 3-1 based on their value equality on NC. Step 3) results in G_n (Figure 2) by taking the union of quotient graphs from Figure 4 and the original graph. Finally, step 4) results in the constraint set $\Sigma \cup \Sigma'$ where $\Sigma' = \Sigma^{\ell}_{\textit{Event:NCTV}}[\mathcal{D}] = \{\sigma_1', \ldots, \sigma_5'\}$.

6.4.2 Formal Definitions.

For a 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

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

For example, Figure 3 shows the projections of G_0 onto $S \in \mathcal{D} = \{NC, NVT, CVT\}$ with identifiers of new nodes v_S (edges (v_S, v)) marked within node circles (alongside the edges, respectively), and node labels ℓ_S carry have real names such as ℓ_{NC} = *:Evt_Mgt*.

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 example, the quotient graphs of G_0 onto $S \in \mathcal{D} = \{NC, NVT, CVT\}$ are shown in Figure 4, where nodes 2-1 and 3-1 are equivalent on NC.

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). For example, property graph G_n from Figure 2 is the union of quotient graphs from Figure 4 and G_0.

For a property graph G and label $\ell \in \mathcal{L}$ we define $\overset{\ell}{\bowtie} G$ as follows:

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

As example, $G_0 = \overset{\ell}{\bowtie} G_n$ with G_0 from Figure 1 and G_n from Figure 2.

In relational databases, a *decomposition* of attribute set R is a set $\mathcal{D}$ of subsets of R such that $\bigcup_{S \in \mathcal{D}} S = R$, for example $\mathcal{D} = \{NC, NVT, CVT\}$ of $CNTV$. For an FD set Σ over R, and subset $S \subseteq R$, $\Sigma[S] = \{X \to Y \in \Sigma^+_{\mathfrak{A}} \mid XY \subseteq S\}$ is the projection of Σ onto S. As example, for $\Sigma = \Sigma_{Event:NCTV} = \{N \to C, NT \to V, TV \to N, CT \to NV\}$ we have $\Sigma[NC] = \{N \to C\}$, $\Sigma[NTV] = \{TV \to N, NT \to V\}$ and $\Sigma[CTV] = \{CT \to V, VT \to C\}$.

Definition 6.18. For gUC/gFD set Σ, label set L, property set P, and decomposition $\mathcal{D}$ of R_P, we define the *L:P-projection* $\Sigma^{\ell}_{L:P}[\mathcal{D}]$ *of* Σ *onto* $\mathcal{D}$ by $\Sigma \cup \{\ell_S{:}S{:}X \mid X \to R_P \in \Sigma_{L:P}[S]$ 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 $S \in \mathcal{D}$, $\Sigma^{\ell}_{L:P}[S]$ is in BCNF (3NF). The L:P-decomposition $\Sigma^{\ell}_{L:P}[\mathcal{D}]$ of Σ is *dependency-preserving* iff $\Sigma_{L:P}$ and $\bigcup_{S \in \mathcal{D}} \Sigma^{\ell}_{L:P}[S]$ are covers of one another. The *L:P-decomposition* $G^{\ell}_{L:P}[\mathcal{D}]$ *of a property graph* G *onto* $\mathcal{D}$ is defined 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 $G_{L:P} = \overset{\ell}{\bowtie} G^{\ell}_{L:P}[\mathcal{D}]$. □

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 [31];
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 $R' \to R_P \in \Sigma^+_{L:P}$ **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** $\left(G^{\ell}_{L:P}[\mathcal{D}], \Sigma^{\ell}_{L:P}[\mathcal{D}]\right)$

As example, for $\Sigma = \{\sigma_1, \ldots, \sigma_4\}$, L = *Event*, P = *CNTV*, and BCNF-decomposition $\mathcal{D} = \{NC, NVT, CVT\}$ of P, $\Sigma^{\ell}_{L:P}[\mathcal{D}] = \Sigma \cup \Sigma'$ where $\Sigma' = \{\sigma'_1, \ldots, \sigma'_5\}$. Indeed, $\Sigma^{\ell}_{L:P}[\mathcal{D}]$ is in BCNF since it is in *Evt_Mgt*:*NC*-BCNF, *Evt_Comp*:*CVT*-BCNF, and *Evt_Detail*:*NVT*-BCNF, see Example 6.7. The decomposition is also dependency-preserving since $\Sigma_{Event:NCTV}$ and the union of $\Sigma_{Event:NCTV}[NC]$, $\Sigma_{Event:NCTV}[NTV]$ and $\Sigma_{Event:NCTV}[CTV]$ cover one another.

Our decomposition is always lossless, but 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.

Algorithm 2 normalizes a property graph G and gUC/FD set Σ tailored to label set L and property set P. Our techniques make it possible for lines (1-16) to apply state-of-the-art normalization from relational databases that achieves a lossless, dependency-preserving 3NF decomposition $\mathcal{D}$ into BCNF whenever possible. $\mathcal{D}$ is then converted into the output $\left(G^{\ell}_{L:P}[\mathcal{D}], \Sigma^{\ell}_{L:P}[\mathcal{D}]\right)$ in line (17).

Theorem 6.19. *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.* □

Given G_0 from Figure 1, L = *Event* and P = *CNTV*, Algorithm 2 returns G_n from Figure 2 and gUC/gFD set $\Sigma \cup \Sigma'$ from Section 2.

Table 3: Details on the graph datasets from 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

7 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?
Q5) How much better is integrity managed after normalization?
Q6) How much faster are aggregate queries after normalization?
Q7) How do the benefits of normalization scale?

Q1)-Q3) will illustrate why normalization is necessary. Q4) will showcase normalization on a real-world graph, and Q5-Q7) will underline benefits of normalization at the operational level.

7.1 Data Sets and Measures

Details of experiments are on our Github repository https://github.com/GraphDatabaseExperiments/normalization_experiments. We analyzed graphs *Northwind* (https://github.com/neo4j-graph-examples/northwind) with 1,035 nodes, 3,139 edges, and sales data; and *Offshore* (https://github.com/ICIJ/offshoreleaks-data-packages) with 2,016,524 nodes, 3,336,971 edges, and global company data. Table 3 shows the node labels L we target, the number $|V_L|$ of nodes with label L, the number $|P|$ of properties for those nodes, the percentage $\%V_{L:P}$ of nodes with these properties, the numbers #gFDs and #gUCs in a minimal cover of constraints that hold on the data sets, and the average number of redundant property values caused by gFDs. Note the high number on *Offshore*.

We used *Neo4j* and its query language *Cypher* as currently most popular graph database (https://db-engines.com/en/ranking/graph+dbms), its support of unique constraints, indexes, and the measure of *database hits*, an abstract unit of the storage engine related to requests for operations on nodes or edges. For comparison with a cloud-based provider, we also used *Amazon Neptune*, which has no support of indexes or database hits. We also measured run times. We 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 the Artifact URL.

7.2 What gFDs do graphs exhibit?

We mined gFDs with fixed target labels *Entity* (Offshore) and *Order* (Northwind). Figure 5 classifies the gFDs $L : P : X \to Y$ by the size $|P|$ of their property set P. If $P = XY$, we call P trivial.

The mined gFDs include interesting examples. On *Offshore*[1], for instance, we have the gFDs

Figure 5: #gFDs by Property Size $|P|$

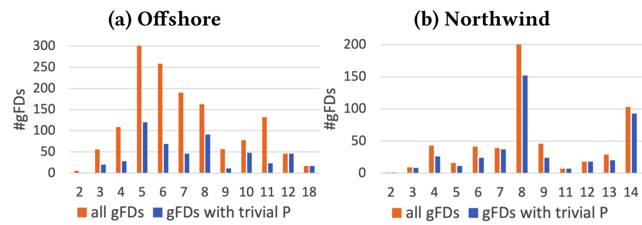


Table 4: gFDs on Offshore Ranked by Redundancy Caused

$P \backslash XY$	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

- *Entity* : *address*, *country_codes*, *countries*: *countries*, *address* $\longrightarrow$ *country_codes*
- *Entity* : *service_provider*, *country_codes*, *countries*: *countries* $\longrightarrow$ *country_codes*.

In particular, for every mined L:P:$X \longrightarrow Y$, 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* $\longrightarrow$ *country_codes* is not satisfied by the dataset.

7.3 What gFDs cause much data redundancy?

Interesting for normalization 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. For *Offshore*, Table 4 shows 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:$X \longrightarrow Y$ may appear to be meaningful:

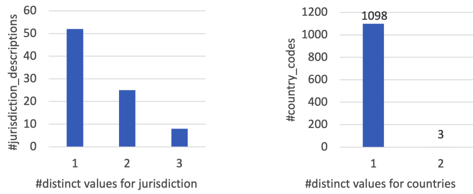
(1) *Entity* : *jurisd_desc*, *jurisdiction*:*jurisd_desc* $\longrightarrow$ *jurisdiction*
(2) *Entity* : *country_codes*, *countries*:*country_codes* $\longrightarrow$ *countries*

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) or (2) are only violated due to data inconsistencies that are a result of data redundancy and the fact these gFDs are not enforced.

[1]Properties described here: https://guides.neo4j.com/sandbox/icij-paradise-papers/datashape.html

Figure 6: Consistency Profiles for gFDs

(a) ***jurisd_desc*** **$\rightarrow$** ***jurisdiction*** **(b)** ***country_codes*** **$\rightarrow$** ***countries***



7.4 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 4 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 *jurisdictions* associated with the same *jurisd_desc*-value in 32 other cases. This consistency profile is illustrated in Figure 6(a), where we list the number of *jurisdiction_descriptions* that have n distinct *jurisdictions* associated with them, for $n = 1, 2, 3$. It is 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 6(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*.

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.

7.5 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 $P_3 = P_1 \cup P_2$.

Table 5: Summary of Normalizing *Offshore*

P	$\lvert P_i \rvert$	#gFDs	#FDs	#red	#dbhits	time (ms)	$\lvert \mathcal{D}_i \rvert$
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

As set Σ we use gFDs E:P:$X \rightarrow Y$ (we write $P = P \backslash XY$) as follows:

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

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

- $R_1^1 = \{c, cc\}$ with $\Sigma_1^1 = \{c \rightarrow cc; cc \rightarrow c\}$, 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. The only properties with E:P_1-redundant values are countries and country_codes. These have been eliminated by the decomposition without losing dependencies.

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

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

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. While L:P_2-redundant values still occur on countries and country_codes only, there are more sources (left-hand sides of FDs) for them compared to P_1. Correspondingly, our decomposition contains more schemata to eliminate the redundancies and preserve more FDs.

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

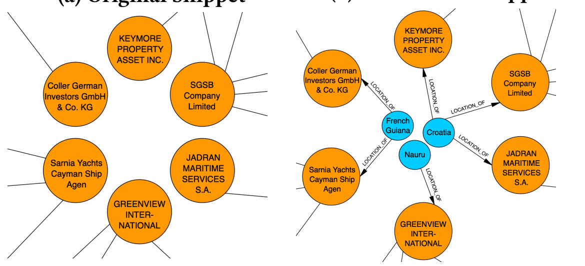
- $R_3^1 = \{c, cc\}$ with $\Sigma_3^1 = \{c \rightarrow cc; cc \rightarrow c\}$
- $R_3^2 = \{sp, s\}$ with $\Sigma_3^2 = \{sp \rightarrow s\}$
- $R_3^3 = \{sp, v\}$ with $\Sigma_3^3 = \{sp \rightarrow v\}$
- $R_3^4 = \{s, v\}$ with $\Sigma_3^4 = \{s \rightarrow v; v \rightarrow s\}$
- $R_3^5 = \{jd, sp, c, a_3\}$ 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. Given $P_1 \cup P_2$, the additional sources for E:P_2-redundancy in countries and country_codes become obsolete again, but new schemata are required to eliminate new E:$P_1 \cup P_2$-redundant values on sourceID and valid_until, and preserve all FDs.

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 5. For each property set P_i we show its size $|P_i|$, the number #gFDs of gFDs *Entity*:P':$X \rightarrow Y \in \Sigma$ 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,

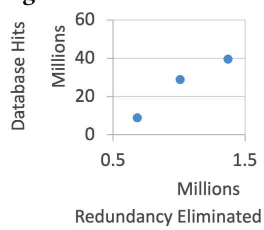
Figure 8: Illustration how redundant property values in an *Offshore* snippet are eliminated by normalization

(a) Original Snippet (b) Normalized Snippet



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 7 illustrates that the graph normalization query uses its access and time effectively to eliminate redundant property value occurrences. Finally, Figure 8 shows a glimpse into the effect of normalizing *Offshore* into $L{:}P_1$-BCNF. The figure illustrates how redundant values on the property *countries* are eliminated on some nodes. In fact, the number of outgoing edges indicate for each new node (in blue) how many redundant occurrences of the countries-value have been eliminated by it.

Figure 7: Good Riddance



7.6 How does integrity management improve?

We will now quantify the benefits of graph normalization by comparing update performance between the original and normalized graph for gFDs $L : P : X \to Y$ as follows:

For *Offshore*: $L = \{E\}$, $P = \{sp, s, v\}$, $X = \{sp\}$, $Y = \{s, v\}$

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

We applied the following update on the original *Offshore* graph:

```
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'
```

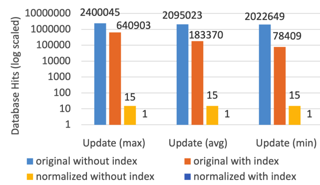
and the following update on the original *Northwind* graph:

```
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' .
```

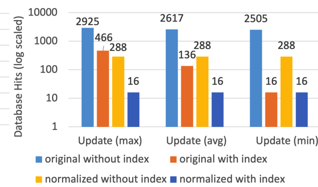
The queries were run using values for *sevice_provider* and *customerID* with the min, avg, and max number of redundant occurrences. We then performed these updates on the graphs normalized by the gFDs above, compared the number of database hits, and the runtime. We also performed the operations using an index for V_L on the property X. The different results can be seen in Figure 10. Normalization for *Offshore* took 6,475 ms (103,995 ms) in Neo4j (Neptune), and 494 ms (3769 ms) for *Northwind*.

Figure 10: Update Comparison: Original vs. Normalized

(a) Offshore - db hits



(b) Northwind - db hits



(c) Offshore - times in ms on Neo4j (Neptune)

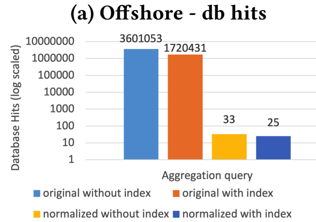
data (index)	redundancy max	avg	min
orig, no	806 (36905)	699 (10114)	650 (4244)
orig, yes	404	100	38
norm, no	0.2 (1,001)		
norm, yes	0.2		

(d) Northwind - times in ms on Neo4j (Neptune)

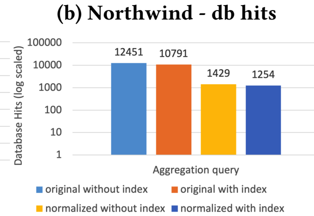
data (index)	redundancy max	avg	min
orig, no	1.5 (135)	0.9 (122)	0.9 (118)
orig, yes	0.4	0.3	0.2
norm, no	0.4 (98)		
norm, yes	0.2		

Figure 11: Aggregate Queries: Original vs. Normalized

(a) Offshore - db hits



(b) Northwind - db hits



(c) Offshore - times in ms on Neo4j (Neptune)

data/index	without index	with index
original	679 (9253)	414
normalized	0.3 (7022)	0.2

(d) Northwind - times in ms on Neo4j (Neptune)

data/index	without index	with index
original	3 (128)	2.9
normalized	0.5 (113)	0.5

Neptune queries are cloud-based, so cannot be compared to Neo4j. Important is the runtime difference between original and normalized graphs. Due to high redundancy in *Offshore*, normalization improves update performance by multiple orders of magnitude. On *Northwind*, with less redundancy, update performance still improves by an order of magnitude. The benefits already apply for normalization with a single FD. While indices result in further optimization for database hits and runtime, these are marginal compared to normalization. Normalized graphs outperform the original graph when indexed, which is similar for queries as shown next.

7.7 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.

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
```

Figure 12: Scaling Comparison: Original vs. Normalized

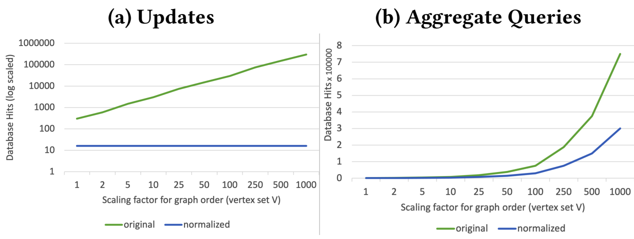
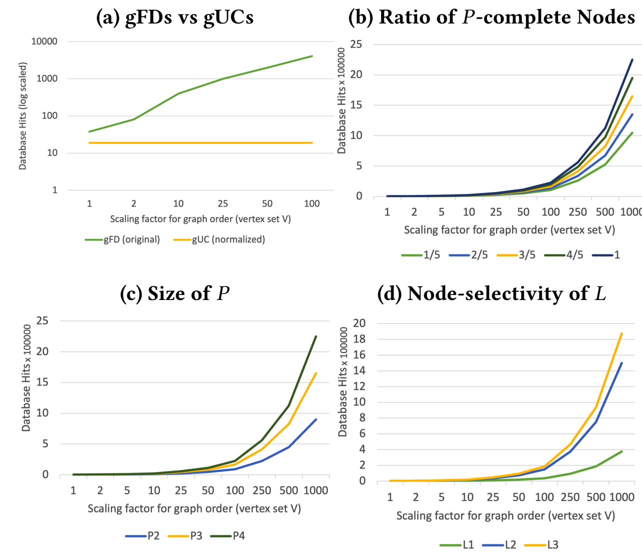


Figure 13: Validation at Scale



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 11, 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.

7.8 How do the 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 $\varphi = \{Employee\} : \{department, manager\} : \{department\} \to \{manager\}$. 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 12 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 12(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 13(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 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.

8 CONCLUSION AND FUTURE WORK

Our research is the first to address the challenging area of normalizing property graphs. Challenges include the unavailability of a schema, the desire to customize normalization of property graphs to flexible requirements of applications, the robustness of normalization under different interpretations of missing properties, the abilities to express and eliminate many redundant property values, and to transfer achievements of BCNF and 3NF from relational databases to property graphs. Indeed, we have turned these challenges into an opportunity by enabling our class of graph-tailored functional dependencies to express application-specific requirements for node labels and properties; plus specifying their semantics to be robust under different interpretations of missing property values. Having created this opportunity, we have then transferred comprehensive achievements from relational databases to property graphs, 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 property graph normalization. Indeed, the reduction of overheads for update maintenance and the speed up of aggregate queries by orders of magnitude, and the effort required to normalize the property graph are all proportional to the amount of redundancy removed.

In future work, we will address other classes of constraints and normal forms. 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 [16] particularly attractive to graph normalization.

REFERENCES

[1] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Alastair Green, Jan Hidders, Bei Li, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Stefan Plantikow, Ognjen Savkovic, Michael Schmidt, Juan Sequeda, Slawek Staworko, Dominik Tomaszuk, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, and Dusan Zivkovic. 2022. PG-Schema: Schemas for Property Graphs. *CoRR* abs/2211.10962 (2022). https://doi.org/10.48550/arXiv.2211.10962

[2] Marcelo Arenas. 2006. Normalization theory for XML. *SIGMOD Record* 35, 4 (2006), 57–64.

[3] William W. Armstrong. 1974. Dependency Structures of Data Base Relationships. In *IFIP Congress*. 580–583.

[4] Carlo Batini and Andrea Maurino. 2018. Design for Data Quality. In *Encyclopedia of Database Systems, Second Edition*, Ling Liu and M. Tamer Özsu (Eds.).

[5] 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.

[6] Philip A. Bernstein. 1976. Synthesizing Third Normal Form Relations from Functional Dependencies. *ACM Trans. Database Syst.* 1, 4 (1976), 277–298.

[7] Joachim Biskup, Umeshwar Dayal, and Philip A. Bernstein. 1979. Synthesizing Independent Database Schemas. In *SIGMOD*. 143–151.

[8] Angela Bonifati, George H. L. Fletcher, Hannes Voigt, and Nikolay Yakovets. 2018. *Querying Graphs*. Morgan & Claypool Publishers.

[9] E. F. Codd. 1970. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM* 13, 6 (1970), 377–387.

[10] Edgar F. Codd. 1972. Further normalization of the database relational model. In *Courant Computer Science Symposia 6: Data Base Systems*. 33–64.

[11] 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.

[12] Renzo Angles et al. 2021. PG-Keys: Keys for Property Graphs. In *SIGMOD '21: International Conference on Management of Data, Virtual Event*. 2423–2436.

[13] Ronald Fagin. 1977. Multivalued Dependencies and a New Normal Form for Relational Databases. *ACM Trans. Database Syst.* 2, 3 (1977), 262–278.

[14] 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.

[15] Wenfei Fan. 2019. Dependencies for Graphs: Challenges and Opportunities. *ACM J. Data Inf. Qual.* 11, 2 (2019), 5:1–5:12.

[16] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.* 33, 2 (2008).

[17] Wenfei Fan and Ping Lu. 2019. Dependencies for Graphs. *ACM Trans. Database Syst.* 44, 2 (2019), 5:1–5:40.

[18] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional Dependencies for Graphs. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. 1843–1857.

[19] Emil Eifrem Ian Robinson, Jim Webber. 2015. *Graph Databases*. O'Reilly Media.

[20] 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.

[21] 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.

[22] 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.

[23] Mark Levene and George Loizou. 1999. Database Design for Incomplete Relations. *ACM Trans. Database Syst.* 24, 1 (1999), 80–125.

[24] Mark Levene and Millist Vincent. 2000. Justification for Inclusion Dependency Normal Form. *IEEE Trans. Knowl. Data Eng.* 12, 2 (2000), 281–291.

[25] Sebastian Link and Henri Prade. 2019. Relational database schema design for uncertain data. *Inf. Syst.* 84 (2019), 88–110.

[26] 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.

[27] Heikki Mannila and Kari-Jouko Räihä. 1992. *Design of Relational Databases*. Addison-Wesley.

[28] Stephan Mennicke. 2019. Modal Schema Graphs for Graph Databases. In *Conceptual Modeling - 38th International Conference, ER 2019, Salvador, Brazil, November 4-7, 2019, Proceedings*. 498–512.

[29] 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.

[30] 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.

[31] Sylvia L. Osborn. 1979. Testing for Existence of a Covering Boyce-Codd normal Form. *Inf. Process. Lett.* 8, 1 (1979), 11–14.

[32] 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.

[33] 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.

[34] 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.

[35] Millist Vincent. 1997. A Corrected 5NF Definition for Relational Database Design. *Theor. Comput. Sci.* 185, 2 (1997), 379–391.

[36] Millist Vincent and Mark Levene. 2000. Restructuring Partitioned Normal Form Relations without Information Loss. *SIAM J. Comput.* 29, 5 (2000), 1550–1567.

[37] Roberto De Virgilio, Antonio Maccioni, and Riccardo Torlone. 2014. Model-Driven Design of Graph Databases. In *Conceptual Modeling - 33rd International Conference, ER 2014, Atlanta, GA, USA, October 27-29, 2014. Proceedings*. 172–185.

[38] 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 2 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 2 are sound for the implication of gUCs and gFDs in property graphs.*

Proof. Soundness of the augmentation rule $\mathcal{A}$ was already proven in [33]. 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*

$$\dfrac{L{:}P{:}X \to YZ}{L{:}P{:}X \to Y} \qquad \dfrac{L{:}P{:}X \to Y \quad L{:}P{:}X \to Z}{L{:}P{:}X \to YZ} \qquad \dfrac{L{:}P{:}X \to Y}{LL'{:}PP'{:}X \to Y}$$

(decomposition, $\mathcal{D}$)   (union, $\mathcal{U}$)   (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:

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

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

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

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

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

This completes the proof. ◻

Theorem A.3 (Theorem 6.1 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 $L' : P' : X' \rightarrow Y' \in \Sigma$. 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 $L : P : X \rightarrow X^+_{\Sigma_{L:P}} \in \Sigma^+_{\mathfrak{C}}$ and $X' \subseteq X^+_{\Sigma_{L:P}}$ we infer $L : P : X \rightarrow X' \in \Sigma^+_{\mathfrak{C}}$ by applying the *decompose*-rule $\mathcal{D}$ from Lemma A.2. Applying the *transitivity*-rule $\mathcal{T}$ to $L : P : X \rightarrow X' \in \Sigma^+_{\mathfrak{C}}$ and $L' : P' : X' \rightarrow Y' \in \Sigma$, we infer $LL' : PP' : X \rightarrow Y' \in \Sigma^+_{\mathfrak{C}}$. Since $L' \subseteq L$, $P' \subseteq P$, we actually have $L : P : X \rightarrow Y' \in \Sigma^+_{\mathfrak{C}}$. 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 $Y \subseteq X^+_{L:P,\Sigma}$, then $L : P : X \rightarrow Y \in \Sigma^+_{\mathfrak{C}}$ contrary to our assumption. Hence, $Y \cap (P - X^+_{L:P,\Sigma}) \neq \emptyset$. Since $X \subseteq X^+_{L:P,\Sigma}$, $Y \cap (P - X^+_{P,\Sigma}) \neq \emptyset$, it follows from the construction of G that G violates $L : P : X \rightarrow Y$. 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 $L : P : X^+_{\Sigma_{L:P}} \in \Sigma^+_{\mathfrak{C}}$. Since $L : P : X \rightarrow X^+_{\Sigma_{L:P}} \in \Sigma^+_{\mathfrak{C}}$ holds, we can apply the *pullback*-rule $\mathcal{P}$ to infer $L : P : X \in \Sigma^+_{\mathfrak{C}}$. From the *weakening*-rule $\mathcal{W}$ we can then infer $L : P : X \rightarrow P \in \Sigma^+_{\mathfrak{C}}$, and since $L : P : P \rightarrow Y \in \Sigma^+_{\mathfrak{C}}$, we can apply the *transitivity*-rule $\mathcal{T}$ to $L : P : X \rightarrow P \in \Sigma^+_{\mathfrak{C}}$ and $L : P : P \rightarrow Y \in \Sigma^+_{\mathfrak{C}}$ to infer the contradiction that $L : P : X \rightarrow Y \in \Sigma^+_{\mathfrak{C}}$. Hence, our assumption must have been wrong, and G satisfies σ in this case.

Case 2.b) Here, σ denotes the gFD $L' : P' : X' \rightarrow Y' \in \Sigma$. 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 $L : P : X \rightarrow X^+_{\Sigma_{L:P}} \in \Sigma^+_{\mathfrak{C}}$ and $X' \subseteq X^+_{\Sigma_{L:P}}$ we infer $L : P : X \rightarrow X' \in \Sigma^+_{\mathfrak{C}}$ by applying the *decomposition*-rule $\mathcal{D}$ from Lemma A.2. Applying the *transitivity*-rule $\mathcal{T}$ to $L : P : X \rightarrow X' \in \Sigma^+_{\mathfrak{C}}$ and $L' : P' : X' \rightarrow Y' \in \Sigma$, we infer $LL' : PP' : X \rightarrow Y' \in \Sigma^+_{\mathfrak{C}}$. Since $L' \subseteq L$ and $P' \subseteq P$, we actually have $L : P : X \rightarrow Y' \in \Sigma^+_{\mathfrak{C}}$. 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 6.3 restated). *For every set $\Sigma \cup \{L : P : X, L : P : X \rightarrow Y\}$ 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) $\Sigma \models L : P : X \rightarrow Y$ if and only if $\Sigma_{L:P} \models X \rightarrow Y$
(2) $\Sigma \models L : P : X$ if and only if $\Sigma_{L:P} \models X \rightarrow R_P$

Proof. **1.** Suppose $\Sigma \not\models L : P : X \rightarrow Y$. Then there is a property graph $G = (V, Ed, \eta, \lambda, \nu)$ that satisfies all gUCs and gFDs in Σ but violates $L : P : X \rightarrow Y$. 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 $X \rightarrow Y$ since $XY \subseteq P$. It remains to show that r satisfies all $X' \rightarrow R_P$ and $X' \rightarrow Y' \in \Sigma_{L:P}$. For $X' \rightarrow R_P \in \Sigma_{L:P}$ 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 $X' \rightarrow R_P$. Since we know that the property graph G satisfies $L' : P' : X' \rightarrow Y$ the following must hold: if $t_v(X') = t_{v'}(X')$, then $t_v(Y') = t_{v'}(Y')$. Consequently, r satisfies $X' \rightarrow Y'$.

Suppose now that $\Sigma_{L:P} \models X \rightarrow Y$ does not hold. Then there is a two-tuple relation $r = \{t, t'\}$ over R_P that satisfies $\Sigma_{L:P}$ and violates $X \rightarrow Y$. 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 $L : P : X \rightarrow Y$. It remains to show that G satisfies all gUCs $L' : P' : X'$ and all gFDs $L' : P' : X' \rightarrow Y'$ in Σ. In the case where $L' \subseteq L$ and $P' \subseteq P$, it follows that $X' \rightarrow R_P$ and $X' \rightarrow Y' \in \Sigma_{L:P}$. Hence, since r satisfies $X' \rightarrow R_P$ and $X' \rightarrow Y'$, it follows that G satisfies $L' : P' : X'$ and $L' : P' : X' \rightarrow Y'$. 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 $L' : P' : X' \rightarrow Y'$ would be satisfied since there are no nodes that apply to them. Hence, we have just shown that $\Sigma \not\models L : P : X \rightarrow Y$.

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 $X \rightarrow R_P$ since $XY \subseteq P$. It remains to show that r satisfies all $X' \rightarrow R_P$ and $X' \rightarrow Y' \in \Sigma_{L:P}$. For $X' \rightarrow R_P \in \Sigma_{L:E}$ 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 $X' \rightarrow R_P$. Since we know that G satisfies $L' : P' : X' \rightarrow Y$ the following must hold: if $t_v(X') = t_{v'}(X')$, then $t_v(Y') = t_{v'}(Y')$. Consequently, r satisfies $X' \rightarrow Y'$.

Suppose now that $\Sigma_{L:P} \models X \rightarrow R_P$ does not hold. Then there is a two-tuple relation $r = \{t, t'\}$ over R_P that satisfies $\Sigma_{L:P}$ and violates $X \rightarrow R_P$. 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 $X \rightarrow R_P$, 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 $L' : P' : X' \rightarrow Y'$ in Σ. In the case where $L' \subseteq L$ and $P' \subseteq P$, it follows that $X' \rightarrow R_P$ and $X' \rightarrow Y' \in \Sigma_{L:P}$. Hence, since r satisfies $X' \rightarrow R_P$ and $X' \rightarrow Y'$, 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 $L' \nsubseteq L$ or $P' \nsubseteq P$. 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 6.8 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 6.3(1) we have $L{:}P{:}X \to Y \in \Sigma^+_{\mathfrak{C}}$ if and only if $X \to Y \in (\Sigma_{L:P})^+_{\mathfrak{A}}$, and by Theorem 6.3(2) we have $L{:}P{:}X \in \Sigma^+_{\mathfrak{C}}$ 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 6.9 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 $L{:}P{:}X \in \Sigma^+_{\mathfrak{C}}$.*

Proof. By Theorem 6.8 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 $L{:}P{:}X \in \Sigma^+_{\mathfrak{C}}$. □

Theorem A.7 (Theorem 6.12 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 6.8 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 6.16 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 $\Sigma^+_{\mathfrak{C}}$ 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 $L'' \nsubseteq L$ or $P'' \nsubseteq P$, 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 6.17 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 6.8 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 6.16. □

A.3 Normalization

Theorem A.10 (Theorem 6.19 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 $\bigcup_{S \in \mathcal{D}} \Sigma^{\ell}_{L:P}[S]$ is a cover of $\Sigma_{L:P}$ by definition of $\Sigma^{\ell}_{L:P}[S]$. Finally, for every $S \in \mathcal{D}$, it is true that $\Sigma^{\ell}_{L:P}[\mathcal{D}]$ is in $\ell_S{:}S$-3NF (or -BCNF, respectively) if and only if $(S, \Sigma_{L:P}[S])$ is in 3NF (BCNF). Hence, the result follows by construction. □