Neo4j Keys

Sebastian Link(✉)

School of Computer Science,
The University of Auckland, Auckland 1010, New Zealand
s.link@auckland.ac.nz

Abstract. Keys play a fundamental role in every data model. They stipulate how real-world entities are identified in the database but also how to physically and logically organize access to data. Neo4j is currently the most popular graph database management system. We address fundamental questions about key constraints as formally defined by the Cypher language of Neo4j. Answers include axiomatic and algorithmic solutions to their implication problem.

Keywords: Integrity · Key · Neo4j · Property graph · Reasoning

1 Introduction

Keys are a core enabler for data management. They are fundamental for understanding the structure and semantics of data. Given a collection of entities, a key is a set of attributes whose values uniquely identify an entity in the collection. Keys form the primary mechanism to enforce entity integrity within database management systems (DBMS) [6]. Keys are essential to many classical areas of data management, including data modeling, database design, indexing, transaction processing, and query optimization. Knowledge about keys enables us to i) uniquely reference entities across data repositories, ii) minimize data redundancy at schema design time to process updates efficiently at run time, iii) provide better selectivity estimates in cost-based query optimization, iv) provide a query optimizer with new access paths that can lead to substantial speedups in query processing, v) allow the database administrator to improve the efficiency of data access via physical design techniques such as data partitioning or the creation of indexes and materialized views, and vi) provide new insights into application data. Keys for graphs have already been studied in academia, and the proposed notion is very expressive [10] due to its target application of entity resolution. The notion subsumes keys from XML as well as conditional constraints [10]. This expressiveness has its price, for example, the associated implication problem is NP-complete, and the associated satisfiability and validation problems are both coNP-complete [10].

While graph databases even precede the relational model of data, they have recently experienced a surge of interest due to many new areas of applications. In particular, many commercial DBMSs have emerged and are heavily used. Neo4j is the most popular graph DBMS[1]. It employs an expressive property graph model. In particular,

[1] https://db-engines.com/en/ranking_trend/graph+dbms.

G. Dobbie et al. (Eds.): ER 2020, LNCS 12400, pp. 19–33, 2020.
https://doi.org/10.1007/978-3-030-62522-1_2

objects such as vertices and edges may have properties. Properties are pairs of a property attribute and a property value, reflecting the NoSQL nature of graph databases. In this article we are interested in keys as they are used in practice and defined by Neo4j. Here, keys are expressions of the form $\ell : \mathfrak{K}$ where ℓ denotes a label and $\mathfrak{K}$ is a finite set of property attributes. A key $\ell : \mathfrak{K}$ is satisfied by a property graph whenever every vertex with label ℓ has assigned a property value to every property attribute in $\mathfrak{K}$, and there are no two distinct vertices that each have a label ℓ and every property attribute in $\mathfrak{K}$ has been assigned matching property values for these vertices. This semantics is reminiscent of candidate keys in relational databases: key attributes must not feature any null-marker occurrences and no two distinct rows can carry matching values on all attributes of the key.

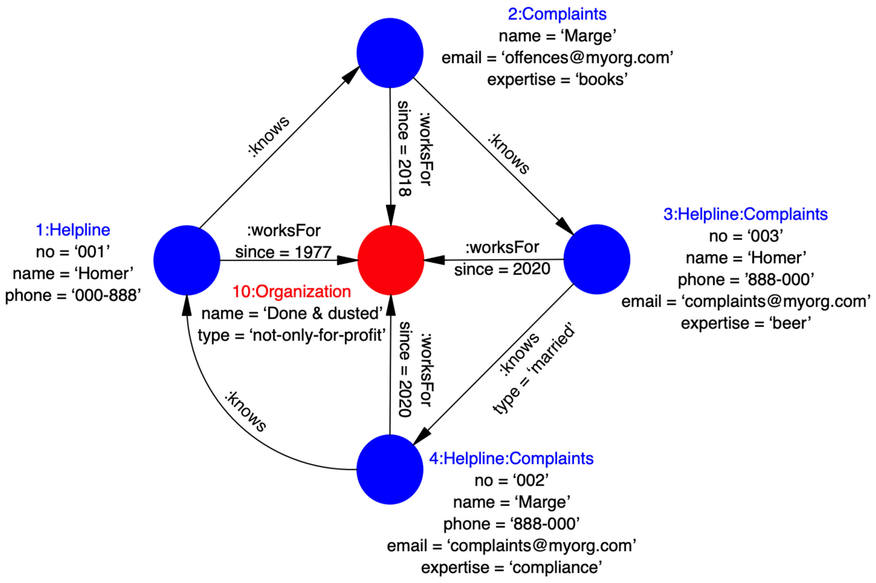

Fig. 1. Our running example of a property graph (Color figure online)

For example, Fig. 1 shows a property graph G that features some staff (blue nodes) working in the *Helpline* and *Complaints* branches of an organization (red node). The graph satisfies the keys *Helpline*:$\{no\}$ and *Helpline*:$\{name, phone\}$. While the first key is unary, that is it has only a single property attribute, the second key is composite. Note that G does not satisfy the keys *Helpline*:$\{no, expertise\}$ nor *Helpline*:$\{name\}$ (vertex 1 and 3) nor *Helpline*:$\{phone\}$ (3 and 4). Indeed, the key *Helpline*:$\{no, expertise\}$ is violated by G because the vertex 1 carries the label *Helpline* but the property attribute *expertise* has not been assigned a property value. Consequently, the familiar notion of a superkey does not carry over from the relational model to the Neo4j graph model. That is, supersets of keys may not be keys. As seen on the example, the reason is that vertices may not define property values for some property attributes. Similarly, G satisfies the key *Complaints*:$\{$*name, email*$\}$, but violates both keys *Complaints*:$\{$*name*$\}$ (vertex 2 and 4), and *Complaints*:$\{$*email*$\}$ (vertex 3 and 4).

Interestingly, Neo4j property graphs can assign multiple labels to any of their vertices. For instance, the property graph in Fig. 1 shows that some staff work for both the *Helpline* and the *Complaints* branch (vertex 3 and 4). We make the interesting observation that Neo4j does not *explicitly* permit the specification of keys for multiple labels, but it does permit it *implicitly* in the sense that vertices with multiple labels inherit the keys explicitly specified on its singleton labels. For example, every property graph that satisfies the two single-label keys *Helpline*:$\{$*no*$\}$ and *Complaints*:$\{$*name,email*$\}$ will also satisfy the multi-label key $\{$*Helpline*,*Complaints*$\}$:$\{$*no,email*$\}$. Indeed, vertices that carry both labels *Helpline* and *Complaints* must have property values defined for property attributes *no* and *email*, and must have unique property value combinations assigned to *no* and *email*. Due to the semantics that Neo4j assigns to keys as well as the fact that vertices carry multiple labels, it is not clear what keys are implicitly satisfied by a property graph, given that it satisfies keys that are enforced explicitly. For instance, G does neither satisfy the key *Helpline*:$\{$*no,expertise*$\}$ due to vertex 1, nor the key $\{$*Helpline,Complaints*$\}$:$\{$*phone,email*$\}$ due to vertices 3 and 4, but it does satisfy the keys $\{$*Helpline*$\}$:$\{$*no,phone*$\}$ and $\{$*Helpline,Complaints*$\}$:$\{$*no,email*$\}$. Hence, the fundamental question about the implication of Neo4j keys arises: Given a set Σ of Neo4j keys, which other Neo4j keys φ are implied by Σ? That is, which keys φ are guaranteed to be satisfied by a property graph G, whenever it satisfies all keys in Σ?

This question is important for data management in Neo4j. In particular, i) keys are strictly enforced by Neo4j under updates, meaning that existence and uniqueness of property values on the property attributes of the keys are checked for all vertices that carry the label of the key, and ii) key specification results in the automatic creation of index structures that are used for data processing, such as speeding up query evaluation by operators like `NodeUniqueIndexSeek` or `NodeUniqueIndexSeekByRange`. In particular, the ability to create index structures for object sets that carry multiple labels have the potential to speed up query evaluation more. Another motivation to study multi-label keys is the realisation of a true multi-label graph model. While multiple labels can be assigned to objects in Neo4j, the co-existence of vertex labels is not being effectively utilized by the underlying DBMS, including the lack of dedicated index structures and operators.

Contributions. We can summarize the contributions of this paper as follows. Firstly, we define multi-labels keys for Neo4j and illustrate their use by applications. Secondly, we study the implication problem associated with Neo4j keys. We derive an axiomatic characterization of the implication problem, as well as an algorithm that decides the implication problem in linear time in the input. Note that the contributions are not restricted to Neo4j, but only to the few concepts we require from the property graph model that Neo4j is based on. Indeed, property graphs in this model are rather general [2].

Organization. We comment on related work in Sect. 2, recall Neo4j's property graph model in Sect. 3, define multi-label keys in Sect. 4, highlight applications of implied keys in Sect. 5, establish axiomatic and algorithmic characterization of the implication problem associated with keys in Sect. 6, and finally conclude and comment on future work in Sect. 7.

2 Related Work

Keys form arguably the most important class of integrity constraint for any data model. For the relational model of data, a key is a set of attributes that ensures that no two different tuples in the relation can have matching values on all the attributes of the key [7,16]. In SQL, candidate keys form minimal sets of attributes that cannot feature any null marker occurrences on any of their attributes and must have a unique combination of values on the attributes across all rows of the table. Keys enforce Codd's principle of entity integrity, which refers to the unique representation of all real-world entities of the underlying application domain within the database [6]. In addition, keys are fundamental to the most important data processing tasks. In query processing, key-foreign key relationships form the foundation for joining tables, and thus for specifying queries soundly. The primary key, which is a distinguished candidate key, determines the physical access model of the relation. Each candidate key and unique constraint gives rise to an index that accelerates query processing. As an integrity constraint, keys enforce the integrity of entities under update operations. For example, we cannot insert new rows that have missing values on some key attribute, and we cannot insert rows that have matching non-null values on the attributes of the key as an existing row in the relation. In database design, the fundamental goal is to obtain a layout of the target database in which data redundancy causing integrity constraints are transformed into keys that prohibit the occurrence of redundant data values [8].

The significance of keys carries over to other data models, in which different notions of keys arise that impact different areas of applications. This includes keys in incomplete data, such as possible and certain keys [14], key sets [11,18], and embedded uniqueness constraints [21,22], keys in description logics [20], keys in semantic models such as the Entity-Relationship model [5,19], keys in object-relational models [13], keys in XML [4,12], and keys over uncertain data such as probabilistic and possibilistic keys [1,3].

Keys have not yet received much attention over graph data models with some noticeable differences [9,10,15,17]. In [9] the authors propose a class of keys for graphs with the primary goal to perform entity matching. The associated implication problem is NP-complete, and those of satisfiability and validation are coNP-complete [10]. In [15] different ways are discussed for mapping relational databases into an RDF graph, with an emphasis on how to represent the original key and foreign key constraints in the resulting RDF graph. A new RDF namespace for the representation of keys and foreign keys is proposed as well. Finally, in [17] the authors put forward some proposals for extending the capabilities of Neo4j in specifying integrity constraints, including an extension of uniqueness constraints limited to single property attributes to uniqueness constraints with multiple property attributes. The authors have provided a simple prototype implementation and experiments.

Keys in Neo4j, in particular their implication problem or their combinatorial behavior, have not been studied in previous work. Since Neo4j is the most popular graph database in practice[2], and keys are of fundamental significance, the work in this paper starts an important line of investigation.

3 Property Graph Model

We recall the basic definitions for the property graph model [2]. For this we assume that the following sets are pairwise disjoint: $\mathcal{O}$ denotes a set of objects, $\mathcal{L}$ denotes a finite set of labels, $\mathcal{K}$ denotes a set of property attributes, and $\mathcal{N}$ denotes a set of values.

A *property graph* is a quintuple $G = (V, E, \eta, \lambda, \nu)$ where $V \subseteq \mathcal{O}$ is a finite set of objects, called *vertices*, $E \subseteq \mathcal{O}$ is a finite set of objects, called *edges*, $\eta : E \to V \times V$ is a function assigning to each edge an ordered pair of vertices, $\lambda : V \cup E \to \mathcal{P}(\mathcal{L})$ is a function assigning to each object a finite set of labels, and $\nu : (V \cup E) \times \mathcal{K} \to \mathcal{N}$ is a partial function assigning values for properties to objects, such that the set of domain values where ν is defined is finite. An example of a property graph is given in Fig. 1.

4 Key Constraints in Cypher

We formally define the syntax and semantics of key constraints. However, we do introduce multi-labelled key constraints, which strictly generalize the key constraints from Cypher that are restricted to a finite label.

According to the Cypher language[3], node key constraints ensure that, for a given label and set of properties: i) All the properties exist on all the nodes with that label, and ii) The combination of the property values is unique.

The Cypher language employs the following syntax to create a key constraint:

```
CREATE CONSTRAINT [constraint_name] ON (ℓ:LabelName)
ASSERT (ℓ.propertyAttribute_1,
      ℓ.propertyAttribute_2,..., ℓ.propertyAttribute_n)
IS NODE KEY
```

For example, in regards to Fig. 1 we may specify:

```
CREATE CONSTRAINT ON (h:Helpline)
ASSERT (h.name, h.phone) IS NODE KEY
```

as a key constraint on vertices with label *Helpline*.

We will now formally define the syntax and semantics of key constraints used by Cypher. Strictly speaking, Cypher only permits the explicit specification of keys on a single label. However, since vertices can carry multiple labels, keys on a single label implicitly also specify keys on vertices that carry multiple labels. Before the formal definition, we define the subset $V_{\mathfrak{L}} \subseteq V$ of vertices in a given property graph that carry at least all the labels of given set $\mathfrak{L}$ of labels, as follows: $V_{\mathfrak{L}} = \{v \in V \mid \mathfrak{L} \subseteq \lambda(v)\}$. For the property graph G from Fig. 1 we have $V_{\{\text{Helpline}\}} = \{1, 3, 4\}$, $V_{\{\text{Complaints}\}} = \{2, 3, 4\}$, and $V_{\{\text{Helpline, Complaints}\}} = \{3, 4\}$. Informally, $V_{\mathfrak{L}}$ is the target set of vertices on which a key should hold.

[2] https://db-engines.com/en/ranking_trend/graph+dbms.
[3] https://neo4j.com/docs/cypher-manual/current/administration/constraints/.

Definition 1. *For a given finite set $\mathcal{L}$ of labels and a given finite set $\mathcal{K}$ of property attributes, a* key constraint *(or* key*) is an expression $\mathfrak{L} : \mathfrak{K}$ where $\mathfrak{L} \subseteq \mathcal{L}$ and $\mathfrak{K} \subseteq \mathcal{K}$. If $\mathfrak{L}$ is a singleton, we call $\mathfrak{L} : \mathfrak{K}$ a* single-labelled *key, and otherwise* multi-labelled*. For a given property graph $G = (V, E, \eta, \lambda, \nu)$ over $\mathcal{O}$, $\mathcal{L}$, $\mathcal{K}$, and $\mathcal{N}$ we say that G* satisfies *the key $\mathfrak{L} : \mathfrak{K}$ over $\mathcal{L}$, and $\mathcal{K}$, denoted by $\models_G \mathfrak{L} : \mathfrak{K}$, if and only if there are no vertices $v_1, v_2 \in V_{\mathfrak{L}}$ such that $v_1 \neq v_2$ and for all $A \in \mathfrak{K}$, $\nu(v_1, A)$ and $\nu(v_2, A)$ are defined and $\nu(v_1, A) = \nu(v_2, A)$.* □

Note that the key $\mathfrak{L} : \emptyset$ expresses that there is at most one vertex in a property graph that carries all the labels in $\mathfrak{L}$. In particular, if $\mathfrak{L} = \emptyset$, then there is at most one vertex in the property graph. Keys with an empty set of property attributes, even for a singleton $\mathfrak{L}$, cannot be defined in Neo4j. For illustration of the definition let us revisit the keys from the introduction.

Example 1. Some keys the property graph G of Fig. 1 satisfies are: *Helpline*:{*no*}, *Helpline*:{*name,phone*}, *Complaints*:{*name,email*}, {*Helpline*,*Complaints*}:{*no, email*}, and {*Helpline*, *Complaints*}:{*name,phone,email*}. However, some keys the property graph G of Fig. 1 does not satisfy are: *Helpline*:{*no,expertise*}, *Helpline*:{*name*}, *Helpline*:{*phone*}, *Complaints*:{*name*}, and {*Helpline*, *Complaints*}: {*phone, email*}.

5 Applications

We illustrate by extensions to our running example the impact of keys on the two most common data processing tasks: updates and queries.

5.1 Update Operations

Once key constraints are specified in Cypher, queries attempting to do any of the following will fail: i) Create new nodes without all the properties or where the combination of property values is not unique, ii) Remove one of the mandatory properties, and iii) Update the properties so that the combination of property values is no longer unique. The following are representative use cases illustrated on our running example:
i) Trying to create the following new node:

```
CREATE  (h:Helpline  {no:004,  name:'Bart', expertise:'pranks'})
```

will fail since no property value is specified for the property attribute *phone*, which is in violation of the key constraint on *Helpline*:{*name,phone*}.
ii) Similarly, removing the value for a key property attribute such as :

```
MATCH (h:Helpline {name:'Homer', phone:'000-888'})REMOVE  h.phone
```

will not remove this property for this vertex.
iii) Updating the values of property attributes so that the combination of property values is no longer unique, for example as in:

```
MATCH (h:Helpline {name: 'Homer', phone:'000-888'})
SET h.phone = '888-000'
RETURN h.name, h.phone
```

will not update the property since the update would violate the key constraint.

5.2 Physical Query Optimization

Furthermore, adding a key constraint for a set of property attributes will also add a composite index on those property attributes, so such an index cannot be added separately. For our example from before, such an index creation would work as follows:

```
CREATE INDEX index_Helpline FOR (h:Helpline)
ON (h.name, h.phone)
```

The index will be used when update operations are being processed to validate whether these comply with the keys or violate any of them. However, the index is also used to speed up the evaluation of queries. For instance, executing the following query without an index

```
MATCH (h:Helpline {name: 'Homer', phone:'000-888'})
RETURN h.no, h.name
```

will do a NodeByLabelScan, since the node label *Helpline* was supplied. This query would still take long on realistically-sized property graph. However, if we create an index as above, then the query optimizer will take advantage of the index and perform a `NodeUniqueIndexSeek` search, which is very efficient.

5.3 Logical Query Optimization and Opportunities with Multiple Labels

We illustrate how the ability to decide implication for keys will enable us to optimize queries logically, for example by rewriting them. While Neo4j does permit the assignment of multiple labels to vertices, this capability does not transfer to index creation. We will demonstrate the benefits of adding such capabilities by way of example.

Similar to SQL, the `DISTINCT` operator removes duplicate rows from the incoming stream of rows. To ensure only distinct elements are returned, `Distinct` will pull in data lazily from its source and build up state. This may lead to increased memory pressure in the system. As an example query consider the following:

```
MATCH (s:Helpline:Complaints)
RETURN DISTINCT s.no, s.email
```

Note that the query looks at returning distinct combinations of number and email of staff who work in both the `Helpline` and `Complaints` branches. If the query engine can conclude that the key $\{Helpline,Complaints\}$:$\{no,email\}$ is implied, it becomes apparent that the `DISTINCT` operator becomes redundant, since by definition of the key, the combination of these values must be unique already. Hence, the query above can be rewritten by removing the `DISTINCT` clause.

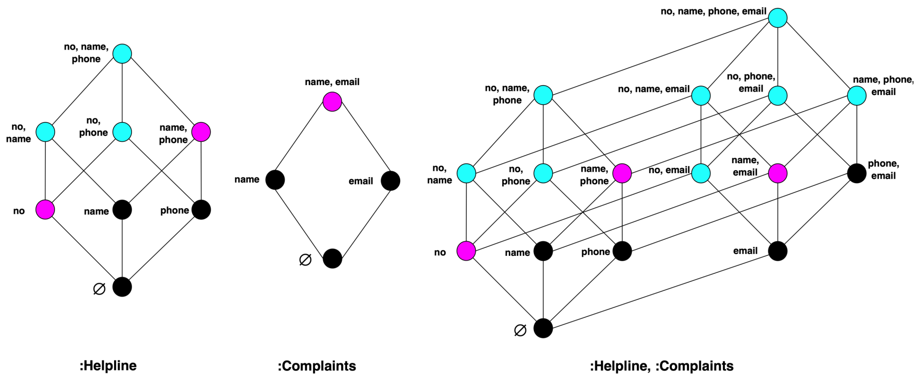

Fig. 2. Powerset Lattices of Keys Implied by Σ_1, Σ_2, and Σ_3 (the given keys are marked in magenta, implied non-given keys marked in cyan, non-implied keys marked in black) (Color figure online)

While not featured in the current Cypher language yet, the definition of multi-label keys and multi-label indices could further help speed up update and query operations. The query above, for example, would engage the single-label index on *Helpline* based on the single-label key $\{$*Helpline*$\}$:$\{$*no*$\}$. However, if we knew that the multi-label key $\{$*Helpline,Complaints*$\}$:$\{$*no,email*$\}$ was implied and a multi-label index would have been created, then we could speed up the search for all query answers even more since the number of vertices that carry both labels would be smaller than the number of vertices with just the *Helpline* label.

6 Reasoning About Neo4j Keys

We formally define the implication problem associated with keys in Neo4j, illustrate it on our running example, and establish axiomatic and algorithmic solutions.

Given a set $\mathcal{L}$ of labels and a set $\mathcal{K}$ of property attributes, let $\Sigma \cup \{\varphi\}$ denote a set of keys over $\mathcal{L}$ and $\mathcal{K}$. The implication problem associated with the class of keys is to decide, given $\Sigma \cup \{\varphi\}$, whether Σ *implies* φ. In fact, Σ *implies* φ, denoted by $\Sigma \models \varphi$, if and only if every property graph G that satisfies all keys in Σ also satisfies φ.

The ability to efficiently decide whether some key constraint φ is implied by Σ is fundamental for the integrity management in Neo4j. Assume that Σ contains keys that are meaningful for the underlying application domain and have been specified, and that φ denotes another meaningful key. If φ is implied by Σ, then we do not need to specify φ because it is specified already implicitly. However, if φ is not implied, then we would need to specify it explicitly on top of all the keys in Σ. As an illustration, let us consider three different sets of keys for our running example: Σ_1 contains the keys *Helpline:*$\{$*no*$\}$ and *Helpline:*$\{$*name,phone*$\}$. Σ_2 consists of the key *Complaints:*$\{$*name,email*$\}$, and Σ_3 consists of all the keys in Σ_1 and Σ_2. Figure 2 shows the powerset lattice that defines the search space for all sets of property attributes that may form a Neo4j key together with the set of labels written at the bottom of the lattice. Sets of property attributes for keys that are given by each of the three key sets are marked in magenta, while sets of property attributes for keys that are implied (but not given) by the three constraint sets are marked in cyan.

6.1 Axiomatic Characterization

We will establish an axiomatization for Neo4j keys in this section. The set $\Sigma^* = \{\varphi \mid \Sigma \models \varphi\}$ denotes the *semantic closure of* Σ, that is, the set of all keys implied by Σ. In principle, the definition of the semantic closure does not tell us whether we can compute it, nevermind how. It is a core reasoning task to investigate whether/how a semantic notion can be characterized syntactically. In fact, we determine the semantic closure Σ^* of a set Σ of keys by applying *inference rules* of the form $\frac{\text{premise}}{\text{conclusion}}$. For a set $\mathfrak{R}$ of inference rules let $\Sigma \vdash_{\mathfrak{R}} \varphi$ denote the *inference* of φ from Σ by $\mathfrak{R}$. That is, there is some sequence $\sigma_1, \ldots, \sigma_n$ such that $\sigma_n = \varphi$ and every σ_i is an element of Σ or is the conclusion that results from an application of 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 keys we have $\Sigma^+_{\mathfrak{R}} \subseteq \Sigma^*$ ($\Sigma^* \subseteq \Sigma^+_{\mathfrak{R}}$). The (finite) set $\mathfrak{R}$ is a (finite) *axiomatization* if $\mathfrak{R}$ is both sound and complete. Table 1 shows two inference rules for the implication of Neo4j keys, which we will show to be sound and complete. We illustrate the use of the inference rules on our running example.

Table 1. Axiomatization $\mathfrak{C}$ of Multi-label Cypher Keys

$\dfrac{\mathfrak{L} : \mathfrak{K}}{\mathfrak{L} \cup \mathfrak{L}' : \mathfrak{K}}$	$\dfrac{\mathfrak{L} : \mathfrak{K}_1 \quad \mathfrak{L} : \mathfrak{K}_2 \cup \mathfrak{K}_3}{\mathfrak{L} : \mathfrak{K}_1 \cup \mathfrak{K}_2}$
(label-extension)	(attribute-extension)

Example 2. Let Σ consist of the three keys *Helpline:*$\{$*no*$\}$, *Helpline:*$\{$*name,phone*$\}$, and *Complaints:*$\{$*name,email*$\}$. A single application of the *attribute-extension* rule to the first two keys above will look as follows:

$$\frac{\mathit{Helpline}{:}\{\mathit{no}\} \qquad \mathit{Helpline}{:}\{\mathit{name,phone}\}}{\mathit{Helpline} : \{\mathit{no}, \mathit{name}\}}$$

and derive the key *Helpline:*$\{$*no, name*$\}$. As a second example we show the following inference from Σ.

$$\frac{\dfrac{\mathit{Helpline}{:}\{\mathit{name,phone}\}}{\{\mathit{Helpline,Complaints}\}{:}\{\mathit{name,phone}\}} \qquad \dfrac{\mathit{Complaints}{:}\{\mathit{name,email}\}}{\{\mathit{Helpline,Complaints}\}{:}\{\mathit{name,email}\}}}{\{\mathit{Helpline,Complaints}\} : \{\mathit{name}, \mathit{phone}, \mathit{email}\}}$$

Here, we infer the key $\{$*Helpline,Complaints*$\}$:$\{$*name,phone,email*$\}$ by first apply the *label-extension* rule to the second and third input key to obtain the same label set on the left-hand side. Subsequently, we can then apply the *attribute-extension* rule to obtain the final conclusion. □

Soundness. We show that any inference of keys by applying rules from $\mathfrak{C}$ results in keys that are implied by the given set.

Lemma 1. *Let $G = (V, E, \eta, \lambda, \nu)$ be a property graph over $\mathcal{O}$, $\mathcal{L}$, $\mathcal{K}$, and $\mathcal{N}$. For all $\mathfrak{L}, \mathfrak{L}' \subseteq \mathcal{L}$, if $\mathfrak{L} \subseteq \mathfrak{L}'$, then $V_{\mathfrak{L}'} \subseteq V_{\mathfrak{L}}$.*

Proof. Let $v \in V$ such that $v \in V_{\mathfrak{L}'}$. Then $\mathfrak{L}' \subseteq \lambda(v)$. Since $\mathfrak{L} \subseteq \mathfrak{L}'$ it follows that $\mathfrak{L} \subseteq \lambda(v)$, too. Hence, $v \in V_{\mathfrak{L}}$. Consequently, $V_{\mathfrak{L}'} \subseteq V_{\mathfrak{L}}$. □

Lemma 2. *The inference rules in $\mathfrak{C}$ are sound for the implication of Neo4j keys.*

Proof. We show the soundness of the *label-extension* rule first. For that purpose we assume that there is a property graph $G = (V, E, \eta, \lambda, \nu)$ that violates the Cypher key $\mathfrak{L} \cup \mathfrak{L}' : \mathfrak{K}$. This means that i) there is some $v \in V_{\mathfrak{L}\cup\mathfrak{L}'}$ and some $A \in \mathfrak{K}$ such that $\nu(v, A)$ is undefined, or ii) there are $v_1, v_2 \in V_{\mathfrak{L}\cup\mathfrak{L}'}$ such that $v_1 \neq v_2$ and $\nu(v_1, \mathfrak{K}) = \nu(v_2, \mathfrak{K})$. From i) and Lemma 1 it would follow that there is some $v \in V_{\mathfrak{L}}$ and some $A \in \mathfrak{K}$ such that $\nu(v, A)$ is undefined. This means that G would also violate $\mathfrak{L} : \mathfrak{K}$. From ii) and Lemma 1 it would follow that there are $v_1, v_2 \in V_{\mathfrak{L}}$ such that $v_1 \neq v_2$ and $\nu(v_1, \mathfrak{K}) = \nu(v_2, \mathfrak{K})$. This means that G would also violate $\mathfrak{L} : \mathfrak{K}$. Hence, we conclude that G would also violate the Cypher key $\mathfrak{L} : \mathfrak{K}$. This proves the soundness of the *label-extension* rule.

It remains to show the soundness of the *attribute-extension* rule. For that purpose we assume that there is a property graph $G = (V, E, \eta, \lambda, \nu)$ that violates the Cypher key $\mathfrak{L} : \mathfrak{K}_1 \cup \mathfrak{K}_2$. Then it follows that either i) there is some $v \in V_{\mathfrak{L}}$ and some $A \in \mathfrak{K}_1 \cup \mathfrak{K}_2$ such that $\nu(v, A)$ is undefined, or ii), for all $v \in V_{\mathfrak{L}}$ and all $A \in \mathfrak{K}_1 \cup \mathfrak{K}_2$, $\nu(v, A)$ is defined, and there are $v_1, v_2 \in V_{\mathfrak{L}}$ such that $v_1 \neq v_2$ and $\nu(v_1, \mathfrak{K}_1 \cup \mathfrak{K}_2) = \nu(v_2, \mathfrak{K}_1 \cup \mathfrak{K}_2)$. From i) it follows that $A \in \mathfrak{K}_1$ or $A \in \mathfrak{K}_2$ such that $\nu(v, A)$ is undefined. Hence, G violates $\mathfrak{L} : \mathfrak{K}_1$ or G violates $\mathfrak{L} : \mathfrak{K}_2 \cup \mathfrak{K}_3$. It remains to consider case ii) above. That is, for all $v \in V_{\mathfrak{L}}$ and for all $A \in \mathfrak{K}_1 \cup \mathfrak{K}_2$, $\nu(v, A)$ is defined, and there are $v_1, v_2 \in V_{\mathfrak{L}}$ such that $v_1 \neq v_2$ and $\nu(v_1, \mathfrak{K}_1 \cup \mathfrak{K}_2) = \nu(v_2, \mathfrak{K}_1 \cup \mathfrak{K}_2)$. In particular, this means that $\nu(v_1, \mathfrak{K}_1) = \nu(v_2, \mathfrak{K}_1)$. Consequently, G violates $\mathfrak{L} : \mathfrak{K}_1$. In summary, we have shown that any property graph that violates $\mathfrak{L} : \mathfrak{K}_1 \cup \mathfrak{K}_2$ will also violate $\mathfrak{L} : \mathfrak{K}_1$ or $\mathfrak{L} : \mathfrak{K}_2 \cup \mathfrak{K}_3$. This proves the soundness of the *attribute-extension* rule.

Example 3. We have seen in Example 2 how the keys

Helpline:{*no,name*} and {*Helpline,Complaints*}*:*{*name,phone,email*}

can be inferred from the set Σ with *Helpline:*{*no*}, *Helpline:*{*name,phone*}, and *Complaints:*{*name,email*}, by using the inference rules in $\mathfrak{C}$. Due to the soundness of the rules it follows that these keys are also implied by Σ. □

Completeness. Before we establish the completeness of our inference rules for Neo4j keys, we introduce some useful notation.

Definition 2. *For a given set Σ of Neo4j keys over $\mathcal{L}$ and $\mathcal{K}$, and a given set $\mathfrak{L} \subseteq \mathcal{L}$ of labels and a given set $\mathfrak{K} \subseteq \mathcal{K}$ of property attributes, let $\Sigma_{\mathfrak{L}} = \{\mathfrak{L}' : \mathfrak{K}' \in \Sigma \mid \mathfrak{L}' \subseteq \mathfrak{L}\}$ denote the set of keys from Σ that are only labelled by labels in $\mathfrak{L}$. Furthermore, let $\mathfrak{K}_{\mathfrak{L},\Sigma} := \bigcup_{\mathfrak{L}':\mathfrak{K}' \in \Sigma_{\mathfrak{L}}} \mathfrak{K}'$ denote the set of property attributes that must be specified by any nodes labelled by labels in $\mathfrak{L}$.* □

We illustrate these new notions by our running example.

Example 4. Let Σ denote the set with *Helpline:*$\{$*no*$\}$, *Helpline:*$\{$*name,phone*$\}$, and *Complaints:*$\{$*name,email*$\}$. Then we have $\Sigma_{\{\mathit{Complaints}\}} = \{\mathit{Complaints}{:}\{\mathit{name,email}\}\}$, $\Sigma_{\{\mathit{Helpline}\}} = \{\mathit{Helpline}{:}\{\mathit{no}\}, \mathit{Helpline}{:}\{\mathit{name,phone}\}\}$, and $\Sigma_{\{\mathit{Helpline,\ Complaints}\}} = \Sigma$. Also, we get $\mathfrak{K}_{\{\mathit{Helpline}\},\Sigma} = \{\mathit{no,name,phone}\}$, $\mathfrak{K}_{\{\mathit{Complaints}\},\Sigma} = \{\mathit{name,email}\}$, and $\mathfrak{K}_{\{\mathit{Helpline,Complaints}\},\Sigma} = \{\mathit{no,name,phone,email}\}$. □

Lemma 3. *Let Σ denote a set of Neo4j keys over $\mathcal{L}$ and $\mathcal{K}$. Then the following hold:*

1. *If $\Sigma_{\mathfrak{L}} \neq \emptyset$, then $\mathfrak{L} : \mathcal{K}_{\mathfrak{L}} \in \Sigma^+$.*
2. *For every $\mathfrak{L}' : \mathfrak{K}' \in \Sigma_{\mathfrak{L}}$ and for all $\mathfrak{K} \subseteq \mathcal{K}$ such that $\mathfrak{K}' \subseteq \mathfrak{K} \subseteq \mathfrak{K}_{\mathfrak{L},\Sigma}$, we have $\mathfrak{L} : \mathfrak{K} \in \Sigma^+$.*

Proof. 1. Let $\Sigma_{\mathfrak{L}} = \{\mathfrak{L}_1 : \mathfrak{K}_1, \ldots, \mathfrak{L}_n : \mathfrak{K}_n\}$ for some non-negative integer n. Since $\Sigma_{\mathfrak{L}}$ is non-empty, it follows that n is a positive integer $n > 0$. Due to the soundness of the label-extension rule it follows that for all $i = 1, \ldots, n$, $\mathfrak{L} : \mathfrak{K}_i \in \Sigma^+$. Due to the soundness of the attribute-extension rule we derive $\mathfrak{L} : \bigcup_{i=1}^{n} \mathfrak{K}_i \in \Sigma^+$, but $\bigcup_{i=1}^{n} \mathfrak{K}_i = \mathfrak{K}_{\mathfrak{L},\Sigma}$. Consequently, $\mathfrak{L} : \mathfrak{K}_{\mathfrak{L},\Sigma} \in \Sigma^+$.

2. Let $\mathfrak{L}' : \mathfrak{K}' \in \Sigma_{\mathfrak{L}}$, and $\mathfrak{K} \subseteq \mathcal{K}$ such that $\mathfrak{K}' \subseteq \mathfrak{K} \subseteq \mathfrak{K}_{\mathfrak{L},\Sigma}$. Since $\mathfrak{L}' : \mathfrak{K}' \in \Sigma_{\mathfrak{L}}$, it follows that $\Sigma_{\mathfrak{L}} \neq \emptyset$. Hence, property *1.* of Lemma 3 implies that $\mathfrak{L} : \mathfrak{K}_{\mathfrak{L},\Sigma} \in \Sigma^+$. Since $\mathfrak{K} \subseteq \mathfrak{K}_{\mathfrak{L},\Sigma}$, it follows that $\mathfrak{K} \cup \mathfrak{K}_{\mathfrak{L},\Sigma} = \mathfrak{K}_{\mathfrak{L},\Sigma}$. Consequently, we have $\mathfrak{L} : \mathfrak{K} \cup \mathfrak{K}_{\mathfrak{L},\Sigma} \in \Sigma^+$. We apply the label-extension rule to $\mathfrak{L}' : \mathfrak{K}' \in \Sigma$ to derive $\mathfrak{L} : \mathfrak{K}' \in \Sigma^+$. We can then apply the *attribute-extension* rule to $\mathfrak{L} : \mathfrak{K}'$ and $\mathfrak{L} : \mathfrak{K} \cup \mathfrak{K}_{\mathfrak{L},\Sigma}$ to derive $\mathfrak{L} : \mathfrak{K}' \cup \mathfrak{K} \in \Sigma^+$. Since $\mathfrak{K}' \subseteq \mathfrak{K}$ it follows that $\mathfrak{L} : \mathfrak{K} \in \Sigma^+$. □

Note that property *1.* of Lemma 3 does not hold when $\Sigma_{\mathfrak{L}} = \emptyset$. In this case, $\mathfrak{K}_{\mathfrak{L}} = \emptyset$, but the Neo4j key $\mathfrak{L} : \emptyset$ is satisfied by a property graph $G = (V, E, \eta, \lambda, \nu)$ if and only if $|V_{\mathfrak{L}}| \leq 1$, that is, if there is at most one vertex in V that carries all the labels of $\mathfrak{L}$.

Theorem 1. *The set $\mathfrak{C}$ forms a finite axiomatization for the implication of Neo4j keys.*

Proof. The soundness of $\mathfrak{C}$ has been established in Lemma 2. It remains to show the completeness. Let $\Sigma \cup \{\mathfrak{L} : \mathfrak{K}\}$ denote a set of Neo4j keys over $\mathcal{L}$ and $\mathcal{K}$ such that $\mathfrak{L} : \mathfrak{K} \notin \Sigma^+$. We need to show that Σ does not imply $\mathfrak{L} : \mathfrak{K}$.

We distinguish between two main cases. In the first main case we assume that $\mathfrak{K} \not\subseteq \mathfrak{K}_{\mathfrak{L},\Sigma}$. That is, there is some property attribute $A \in \mathfrak{K} - \mathfrak{K}_{\mathfrak{L},\Sigma}$. Let us define the property graph $G = (V, E, \eta, \lambda, \nu)$ as follows: $V = \{v\}$, $E = \emptyset$, and therefore there is nothing to define for η, $\lambda(v) = \mathfrak{L}$, and for all $B \in \mathfrak{K}_{\mathfrak{L},\Sigma}$ we define $\nu(v, B) := 0$ and $\nu(v, B)$ remains undefined on other property attributes. For this first main case it follows that G violates the key $\mathfrak{L} : \mathfrak{K}$ since $v \in V_{\mathfrak{L}}$ and $\nu(v, A)$ is undefined since $A \in \mathfrak{K} - \mathfrak{K}_{\mathfrak{L},\Sigma}$. It remains to show in this case that G satisfies $\mathfrak{L}' : \mathfrak{K}'$ for all $\mathfrak{L}' : \mathfrak{K}' \in \Sigma$. If $\mathfrak{L}' : \mathfrak{K}' \in \Sigma_{\mathfrak{L}}$, then we have $\mathfrak{K}' \subseteq \mathfrak{K}_{\mathfrak{L},\Sigma}$. It follows that for all $v \in V_{\mathfrak{L}'}$ and for all $B \in \mathfrak{K}'$, $\nu(v, B)$ is defined. Since there is only one vertex in V, it follows that G must satisfy $\mathfrak{L}' : \mathfrak{K}' \in \Sigma_{\mathfrak{L}}$. Otherwise, $\mathfrak{L}' : \mathfrak{K}' \notin \Sigma_{\mathfrak{L}}$, which means that $\mathfrak{L}' \not\subseteq \mathfrak{L}$. Consequently, $V_{\mathfrak{L}'} = \emptyset$ and G must satisfy $\mathfrak{L}' : \mathfrak{K}' \notin \Sigma_{\mathfrak{L}}$. Hence, we have shown that Σ does not imply $\mathfrak{L} : \mathfrak{K}$ in the first main case.

In the second main case we assume the opposite, that is, $\mathfrak{K} \subseteq \mathfrak{K}_{\mathfrak{L},\Sigma}$. Let us define the property graph $G = (V, E, \eta, \lambda, \nu)$ as follows: $V = \{v_1, v_2\}$, $E = \emptyset$, and therefore there is nothing to define for η, $\lambda(v_1) = \mathfrak{L} = \lambda(v_2)$, for all $B \in \mathfrak{K}$ we define $\nu(v_1, B) = 0 = \nu(v_2, B)$, for all $B \in \mathfrak{K}_{\mathfrak{L},\Sigma} - \mathfrak{K}$ we define $\nu(v_1, B) = 0$ and $\nu(v_2, B) = 1$, and $\nu(v, B)$ and $\nu(v, B)$ remains undefined on other property attributes. It follows that G violates $\mathfrak{L} : \mathfrak{K}$ since there are $v_1, v_2 \in V_{\mathfrak{L}}$ such that $v_1 \neq v_2$ and $\nu(v_1, \mathfrak{K}) = \nu(v_2, \mathfrak{K})$. It remains to show that G satisfies every $\mathfrak{L}' : \mathfrak{K}' \in \Sigma$. In case *a)* we assume $\mathfrak{L}' : \mathfrak{K}' \notin \Sigma_{\mathfrak{L}}$, which means that $\mathfrak{L}' \not\subseteq \mathfrak{L}$. Consequently, $V_{\mathfrak{L}'} = \emptyset$ and G must satisfy $\mathfrak{L}' : \mathfrak{K}' \notin \Sigma_{\mathfrak{L}}$. In case *b)* we assume $\mathfrak{L}' : \mathfrak{K}' \in \Sigma_{\mathfrak{L}}$. Hence, $\Sigma_{\mathfrak{L}} \neq \emptyset$ and $\mathfrak{L} : \mathfrak{K}_{\mathfrak{L},\Sigma} \in \Sigma^+$ by Property *1.* of Lemma 3. In particular, $\mathfrak{K}' \subseteq \mathfrak{K}_{\mathfrak{L},\Sigma}$. Consequently, for all $v \in V_{\mathfrak{L}'}$ and for all $B \in \mathfrak{K}'$ it follows that $\nu(v, B)$ is defined. In case *b.1)* we assume that $\mathfrak{K}' \not\subseteq \mathfrak{K}$. Hence, there is some $A \in \mathfrak{K}' - \mathfrak{K}$ such that $\nu(v_1, A) = 0 \neq 1 = \nu(v_2, A)$ holds. That is, $\mathfrak{L}' : \mathfrak{K}'$ is satisfied by G. Finally, in case *b.2)* we assume that $\mathfrak{K}' \subseteq \mathfrak{K}$. Hence, in this case we have $\mathfrak{K}' \subseteq \mathfrak{K} \subseteq \mathfrak{K}_{\mathfrak{L},\Sigma}$ with $\mathfrak{L}' : \mathfrak{K}' \in \Sigma_{\mathfrak{L}}$. Hence, Property *2.* of Lemma 3 would imply that $\mathfrak{L} : \mathfrak{K} \in \Sigma^+$. This would be a contradiction to our original assumption that $\mathfrak{L} : \mathfrak{K} \notin \Sigma^+$, so case *b.2)* cannot occur. We have just shown that in all possible cases, Σ does not imply $\mathfrak{L} : \mathfrak{K}$. This shows the completeness of $\mathfrak{C}$. □

The proof of Theorem 1 contains a general construction of property graphs showing that a key is not implied by a given set of keys, as illustrated on our running example.

Example 5. Let Σ consist of the keys: *Helpline:*$\{$*no*$\}$, *Helpline:*$\{$*name,phone*$\}$, and *Complaints:*$\{$*name,email*$\}$. The key *Helpline:*$\{$*no,expertise*$\}$ is not implied by Σ. Following the construction of Theorem 1 we would create a property graph G_1 with one vertex v, label $\lambda(v) =$ *:Helpline* and properties $\nu(v, \mathit{no}) = \nu(v, \mathit{name}) = \nu(v, \mathit{phone}) = 0$. In particular, $\mathfrak{K}_{\mathit{Helpline},\Sigma} = \{\mathit{no}, \mathit{name}, \mathit{phone}\}$. Indeed, the property graph G_1 satisfies all keys in Σ, but violates the key *Helpline:*$\{$*no,expertise*$\}$ since the value $\nu(v, \mathit{expertise})$ has remained undefined. As observed before, $\{$*Helpline,Complaints*$\}$*:*$\{$*phone,email*$\}$ is not implied by Σ. Following the construction of Theorem 1 we would create a property graph G_2 with two vertices v_1 and v_2, labels $\lambda(v_1) = \{$*:Helpline*, *:Complaints*$\}$ and properties $\nu(v_1, \mathit{phone}) = \nu(v_2, \mathit{phone}) = \nu(v_1, \mathit{email}) = \nu(v_2, \mathit{email}) = 0$, and $\nu(v_1, \mathit{name}) = \nu(v_1, \mathit{no}) = 0 \neq 1 = \nu(v_2, \mathit{name}) = \nu(v_2, \mathit{no})$. In particular, $\mathfrak{K}_{\{\mathit{Helpline,Complaints}\},\Sigma} = \{\mathit{no}, \mathit{name}, \mathit{phone}, \mathit{email}\}$, and $\mathfrak{K} = \{\mathit{phone}, \mathit{email}\}$. Indeed, G_2 satisfies all keys in Σ, but violates the key $\{$*Helpline,Complaints*$\}$*:*$\{$*phone,email*$\}$ since the two distinct vertices v_1 and v_2 both have labels *:Helpline* and *:Complaints*, and have matching property values on both *phone* and *email*. □

6.2 Algorithmic Characterization

The axiomatization of keys enables us to establish an algorithm that decides the associated implication problem. In fact, we can derive the following characterization for the implication problem, from which we will derive such an algorithm.

Theorem 2. *Let $\Sigma \cup \{\mathfrak{L} : \mathfrak{K}\}$ denote a set of Neo4j keys over $\mathcal{L}$ and $\mathcal{K}$. Then $\Sigma \models \mathfrak{L} : \mathfrak{K}$ if and only if $\mathfrak{K} \subseteq \mathfrak{K}_{\mathfrak{L},\Sigma}$ and there is some $\mathfrak{L}' : \mathfrak{K}' \in \Sigma_{\mathfrak{L}}$ such that $\mathfrak{K}' \subseteq \mathfrak{K}$.*

Proof. **Sufficiency.** If $\mathfrak{K} \subseteq \mathfrak{K}_{\mathfrak{L},\Sigma}$ and there is some $\mathfrak{L}' : \mathfrak{K}' \in \Sigma_{\mathfrak{L}}$ such that $\mathfrak{K}' \subseteq \mathfrak{K}$, then the second property of Lemma 3 shows that $\mathfrak{L} : \mathfrak{K} \in \Sigma^+_{\mathfrak{C}}$. The soundness of $\mathfrak{C}$ means that $\mathfrak{L} : \mathfrak{K}$ is implied by Σ.

Necessity. Suppose that $\mathfrak{K} \not\subseteq \mathfrak{K}_{\mathfrak{L},\Sigma}$. This constitutes the first main case in the proof of Theorem 1. Hence, the property graph created in that case satisfies Σ and violates $\mathfrak{L} : \mathfrak{K}$. Consequently, $\mathfrak{L} : \mathfrak{K}$ is not implied by Σ.

Suppose now that for all $\mathfrak{L}' : \mathfrak{K}' \in \Sigma_{\mathfrak{L}}$ we have that $\mathfrak{K}' \cap (\mathfrak{K}_{\mathfrak{L},\Sigma} - \mathfrak{K}) \neq \emptyset$. Then the property graph from the second main case in the proof of Theorem 1 satisfies Σ and violates $\mathfrak{L} : \mathfrak{K}$. Consequently, $\mathfrak{L} : \mathfrak{K}$ is not implied by Σ. □

Algorithm 1 is based on the characterization of key implication in Theorem 2.

Theorem 3. *Algorithm 1 decides the implication problem $\Sigma \models \mathfrak{L} : \mathfrak{K}$ for the class of keys in $\mathcal{O}(||\Sigma \cup \{\mathfrak{L} : \mathfrak{K}\}||)$ time.*

Proof. The soundness of Algorithm 1 follows from Theorem 2. The upper time bound follows straight from the loop between steps 4–8. □

Example 6. On the input where Σ consist of *Helpline:*$\{$*no*$\}$, *Helpline:*$\{$*name,phone*$\}$, and *Complaints:*$\{$*name,email*$\}$, and φ denotes *Helpline:*$\{$*no,expertise*$\}$, Algorithm 1 returns *FALSE* due to *expertise* $\notin$ $\{$*no,name,phone*$\}$ $=$ $\mathfrak{K}_{\{Helpline\},\Sigma}$ in line 9. On input Σ and $\{$*Helpline,Complaints*$\}$*:*$\{$*no,email*$\}$, Algorithm 1 returns *TRUE* since $\{$*no,email*$\}$ $\subseteq$ $\{$*no,name,phone,email*$\}$ $=$ $\mathfrak{K}_{\{Helpline,Complaints\},\Sigma}$, *Helpline:*$\{$*no*$\}$ $\in$ $\Sigma_{\{Helpline,Complaints\}}$ and $\{no\} \subseteq \{no, email\}$. □

Algorithm 1. Implication of Neo4j Keys

```
Require: Neo4j Key Set $\Sigma \cup \{\mathfrak{L} : \mathfrak{K}\}$
Ensure: TRUE, if $\Sigma \models \mathfrak{L} : \mathfrak{K}$, and FALSE, otherwise
 1: Unique ← FALSE;
 2: Exists ← FALSE;
 3: $\mathfrak{K}_{\mathfrak{L},\Sigma} \leftarrow \emptyset$;
 4: for all $\mathfrak{L}' : \mathfrak{K}' \in \Sigma$ do
 5:     if $\mathfrak{L}' \subseteq \mathfrak{L}$ then                    ▷ This means $\mathfrak{L}' : \mathfrak{K}' \in \Sigma_{\mathfrak{L}}$
 6:         $\mathfrak{K}_{\mathfrak{L},\Sigma} \leftarrow \mathfrak{K}_{\mathfrak{L},\Sigma} \cup \mathfrak{K}'$;      ▷ All properties of this key exist
 7:     if (NOT(Unique) AND $\mathfrak{K}' \subseteq \mathfrak{K}$) then
 8:         Unique ← TRUE;      ▷ Found an input key that makes our candidate unique
 9: if $\mathfrak{K} \subseteq \mathfrak{K}_{\mathfrak{L},\Sigma}$ then return Exists ← TRUE;      ▷ All required properties exist
10: if (Exists AND Unique) then return TRUE      ▷ Properties exist and are unique
11: else return FALSE
```

7 Conclusion and Future Work

We have established that Neo4j keys can be reasoned about efficiently, which is beneficial to update and query operations on property graphs. In particular, multiple labels offer new opportunities for physical and logical data management. In the future, we will study the interaction of Neo4j keys with existence and uniqueness constraints that are also part of the Cypher language. In addition, we will also investigate different semantics and different sets of constraints. An interesting proposal might be to combine Neo4j keys with recently investigated embedded uniqueness constraints [21,22]. These would be expressions of the form $\mathfrak{L} : E : X$ with the semantics that for the set of vertices v that carry all labels in $\mathfrak{L}$ and for which $\nu(v, A)$ is defined for all $A \in E$, the combination of property values over the property attributes in X is unique.

References

1. Balamuralikrishna, N., Jiang, Y., Koehler, H., Leck, U., Link, S., Prade, H.: Possibilistic keys. Fuzzy Sets Syst. **376**, 1–36 (2019)
2. Bonifati, A., Fletcher, G.H.L., Voigt, H., Yakovets, N.: Querying Graphs. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, San Rafael (2018)
3. Brown, P., Link, S.: Probabilistic keys. IEEE Trans. Knowl. Data Eng. **29**(3), 670–682 (2017)
4. Buneman, P., Davidson, S.B., Fan, W., Hara, C.S., Tan, W.C.: Keys for XML. Comput. Netw. **39**(5), 473–487 (2002)
5. Chen, P.P.: The entity-relationship model - toward a unified view of data. ACM Trans. Database Syst. **1**(1), 9–36 (1976)
6. Codd, E.F.: A relational model of data for large shared data banks. Commun. ACM **13**(6), 377–387 (1970)
7. Demetrovics, J.: On the number of candidate keys. Inf. Process. Lett. **7**(6), 266–269 (1978)
8. Fagin, R.: A normal form for relational databases that is based on domains and keys. ACM Trans. Database Syst. **6**(3), 387–415 (1981)
9. Fan, W., Fan, Z., Tian, C., Dong, X.L.: Keys for graphs. PVLDB **8**(12), 1590–1601 (2015)
10. Fan, W., Lu, P.: Dependencies for graphs. ACM Trans. Database Syst. **44**(2), 5:1–5:40 (2019)
11. Hannula, M., Link, S.: Automated reasoning about key sets. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018. LNCS (LNAI), vol. 10900, pp. 47–63. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94205-6_4
12. Hartmann, S., Link, S.: Efficient reasoning about a robust XML key fragment. ACM Trans. Database Syst. **34**(2), 1–33 (2009)
13. Khizder, V.L., Weddell, G.E.: Reasoning about uniqueness constraints in object relational databases. IEEE Trans. Knowl. Data Eng. **15**(5), 1295–1306 (2003)
14. Köhler, H., Leck, U., Link, S., Zhou, X.: Possible and certain keys for SQL. VLDB J. **25**(4), 571–596 (2016). https://doi.org/10.1007/s00778-016-0430-9
15. Lausen, G.: Relational databases in RDF: keys and foreign keys. In: Christophides, V., Collard, M., Gutierrez, C. (eds.) ODBIS/SWDB -2007. LNCS, vol. 5005, pp. 43–56. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70960-2_3
16. Lucchesi, C.L., Osborn, S.L.: Candidate keys for relations. J. Comput. Syst. Sci. **17**(2), 270–279 (1978)
17. Pokorný, J., Valenta, M., Kovacic, J.: Integrity constraints in graph databases. In: The 8th International Conference on Ambient Systems, Networks and Technologies, ANT 2017/The 7th International Conference on Sustainable Energy Information Technology, SEIT 2017, Madeira, Portugal, 16–19 May 2017, vol. 109, pp. 975–981. Elsevier (2017). Procedia Computer Science
18. Thalheim, B.: On semantic issues connected with keys in relational databases permitting null values. Elektronische Informationsverarbeitung und Kybernetik **25**(1/2), 11–20 (1989)

19. Thalheim, B.: Entity-Relationship Modeling - Foundations of Database Technology. Springer, Heidelberg (2000). https://doi.org/10.1007/978-3-662-04058-4
20. Toman, D., Weddell, G.E.: On keys and functional dependencies as first-class citizens in description logics. J. Autom. Reason. **40**(2–3), 117–132 (2008). https://doi.org/10.1007/s10817-007-9092-z
21. Wei, Z., Leck, U., Link, S.: Discovery and ranking of embedded uniqueness constraints. PVLDB **12**(13), 2339–2352 (2019)
22. Wei, Z., Link, S., Liu, J.: Contextual keys. In: Mayr, H.C., Guizzardi, G., Ma, H., Pastor, O. (eds.) ER 2017. LNCS, vol. 10650, pp. 266–279. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69904-2_22