

PROPER - A Graph Data Model Based on Property Graphs

Nicolas Spyratos^{1(✉)} and Tsuyoshi Sugibuchi²

¹ Laboratoire de Recherche en Informatique, UMR8623 of CNRS,
Université Paris-Sud 11, Orsay, France

Nicolas.Spyratos@lri.fr

² CustomerMatrix Inc., Paris, France
sugibuchi@gmail.com

Abstract. We present a graph data model, called PROPER, which is based on property graphs. Our model consists of a property graph G “augmented” with the concepts of *hyper node* and *hyper edge*. A hyper node is an abstraction of a set of nodes of G having the same properties; and a hyper edge is an abstraction of a set of edges of G having the same label. A graph database over G is defined to be a higher level property graph whose nodes and edges are hyper nodes and hyper edges over G . We introduce a set of operations that generate new hyper nodes and new hyper edges from old, therefore providing the basis for a query language in PROPER. We call this set the “graph algebra”. We also show how certain semantic constructs such as equational constraints and ISA relationships can be defined in our model.

We demonstrate the expressive power of PROPER by showing how a relational database, together with functional dependencies, can be embedded in PROPER in the form of a graph database; and how the relational algebra operations can be mapped as operations of the graph algebra.

1 Introduction

A graph is a collection of nodes and edges in which nodes represent entities and edges represent directed relationships between nodes. An edge has a source node and a target node.

Figure 1(a) shows a graph representing social network data. Each node and each edge is associated with a unique identifier (an integer in our example) and a label. Two nodes or two edges can have the same label; this is the case of nodes 2 and 3, and edges 5 and 7. Moreover, there can be two different edges with the same source and the same target as long as they have different labels; this is the case of edges 6 and 7. The identifiers and labels of nodes and edges come from predefined sets. For example, in Fig. 1(a), identifiers are integers and labels are character strings.

A graph is a general purpose expressive structure allowing to model all kinds of real world scenarios, ranging from business and government applications to

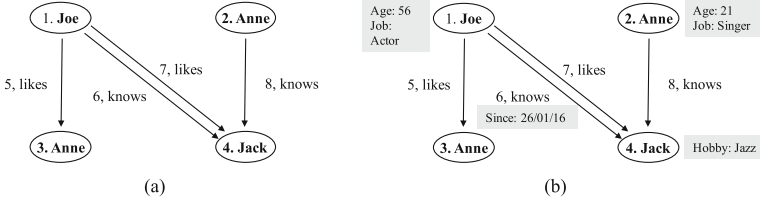


Fig. 1. A graph and a property graph

biology, mobile telephony, geomatics or social networks [5,6]. Unlike the relational model of data, a graph-based data model can describe not only uniform and rule-bound relationships but also exceptional and irregular data.

Graph databases have attracted a lot of attention in recent years [1,2]. Their success is mainly due to the fact that many modern applications are naturally modelled as graph representations. In graph databases, data is stored natively in graph structures and queries are defined in terms of graph traversals and graph matching [7].

The most popular variants of graphs today are the *property graph* [6], the resource description framework (RDF) [3] and the *hypergraph* [4]. The model that we introduce in this paper is based on property graphs. A property graph is a graph in which each node and each edge can be associated with a set of key-value pairs.

Figure 1(b) shows a property graph, which is actually the graph of Fig. 1(a) in which we have associated a set of key-value pairs with some of its nodes and edges. For example, node 1 is associated with two key-value pairs, namely *Age* : 56 and *Job* : ‘Actor’. The labels *Age* and *Job* are the keys, and 56 and ‘Actor’ are the corresponding values. Similarly, node 2 is associated with two key-value pairs; node 4 with one key-value pair; and edge 6 with one key-value pair; while node 3 and edges 5, 7 and 8 are associated with no key-value pairs. Keys and their values come from predefined sets. For example, in Fig. 1(b), keys are character strings; the values of the key *Age* are integers; the values of the key *Job* are character strings, etc.

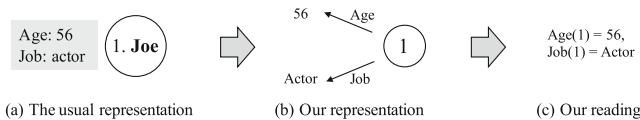


Fig. 2. Representing property-value pairs of a node

In this paper we shall represent the key-value pairs of a node using a different approach, as shown in Fig. 2. Figure 2(a) shows the usual representation using key-value pairs. Figure 2(b) shows our representation, where the key-value pair

Age: 56 is represented by the edge $1 \rightarrow 56$ with label p_{Age} ; and the key-value pair *Job*: ‘Actor’ is represented by the edge $1 \rightarrow \text{‘Actor’}$ with label p_{Job} . The label p_{Age} stands for “property *Age*”, and similarly, the label p_{Job} stands for “property *Job*”. In accordance with our representation, our reading of properties becomes as follows: “ $p_{Age}(1) = 56$ ” (read as “ p_{Age} of 1 equals 56”); and “ $p_{Job}(1) = \text{‘Actor’}$ ” (read as “ p_{Job} of 1 equals ‘Actor’”), as shown in Fig. 2(c). It is important to note that the properties of a node are declared during node creation but their values might not be known at node creation time. In other words the values of one or more properties of a node might be missing.

A basic assumption underlying our model is that no node can have two different properties with the same target (such properties are called “parallel properties” and they differ only in their labels). For example, no node n can have two different properties, p_{Age} and p'_{Age} . Indeed, in such a case, it would be possible to associate node n with two different ages. Of course there are situations where it makes sense to have two different properties with the same target. However, in this paper we shall make the assumption of “no parallel properties” in a node.

Note that a similar situation arises in the relational model, where the basic assumption is that attribute values are atomic (the so called first normal form assumption). An immediate consequence of this assumption is that on any given tuple no attribute can have two different values.

One can visualize the data of a property graph by considering that each node and each edge is “clickable”. By clicking a node/edge, the associated properties appear (and they are also clickable). By clicking a property we obtain the associated value. For example, in Fig. 2(b), if we click node 1 then the properties p_{Age} and p_{Job} appear; and if we click p_{Age} then we obtain the value 56; and if we click p_{Job} then we obtain the value ‘Actor’.

In this paper we present a graph data model, that we call PROPER. It consists of a property graph G “augmented” with the concepts of *hyper node* and *hyper edge*. A hyper node H is an abstraction of a set of nodes of G having the same properties; and a hyper edge E from hyper node H to hyper node H' is an abstraction of a set of edges from nodes of H to nodes of H' having the same label. A graph database over G is defined to be a higher level property graph whose nodes and edges are hyper nodes and hyper edges.

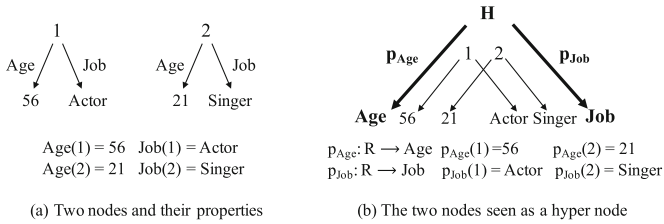


Fig. 3. Abstracting a set of two nodes 1 and 2 as a hyper node H

As an example of hyper node consider two nodes of a property graph G , say 1 and 2, as shown in Fig. 3(a). Following our representation (as explained in Fig. 2) we can “factor out” the labels p_{Age} and p_{Job} and view these two nodes (collectively) as a single node H ; and we can view the properties p_{Age} and p_{Job} (collectively) as two functions, P_{Age} and P_{Job} , on the set $H = \{1, 2\}$, defined as follows (refer also to Fig. 3(b)):

$P_{Age} : H \rightarrow Age$ such that $P_{Age}(1) = p_{Age}(1) = 56$ and $P_{Age}(2) = p_{Age}(2) = 21$

$P_{Job} : H \rightarrow Job$, such that $P_{Job}(1) = p_{Job}(1) = 'Actor'$ and $P_{Job}(2) = p_{Job}(2) = 'Singer'$

Then the set $H = \{1, 2\}$ together with the functions P_{Age} and P_{Job} is a *hyper node* over G .

It is important to note that, in the above definitions of P_{Age} and P_{Job} , the codomains Age and Job are attributes in the sense of the relational model (i.e. they are names, each of which is associated with a set of values called its *domain*). It is also important to note that, as we mentioned earlier, the value of one or more properties of a node might be missing. In other words, the functions P_{Age} and P_{Job} are in general partial functions.

As an example of hyper edge consider the five edges shown in Fig. 4(a). If we collect together the edges 8, 10 and 11 then we have a hyper edge E from hyper node $H = \{1, 2\}$ to hyper node $H' = \{4, 5, 6\}$ as shown in Fig. 4(b). Note that the only requirement in order to have a hyper edge is that all its constituent edges have the same label. In other words, it is not necessary to include in E *all* edges from hyper node H to hyper node H' with label e . Figure 4(c) shows another example of hyper edge E' .

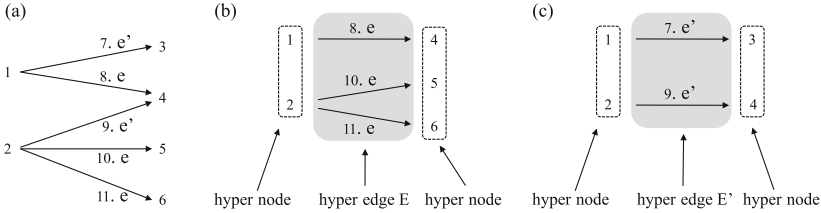


Fig. 4. Abstracting a set of edges as a hyper edge

In our model we use a set of operations over hyper nodes and hyper edges that generate new hyper nodes and new hyper edges from old, therefore providing the basis for a query language. Additionally, We introduce two semantic concepts, namely equational constraints and ISA hyper edges, inspired from functional dependencies of the relational model and ISA relationships of semantic models, respectively.

One important application of graph databases as defined here is graph summarization, whose main goal is to produce a compressed representation of an

input graph [10–12]. Indeed, the way hyper nodes and hyper edges are defined in our model (as abstractions of sets of nodes and of sets of edges, respectively) is actually a summarization tool. Graph summarization is important in order to understand the underlying characteristics of large graphs, and graph summarization techniques are critical in this respect [12]. However, graph summarization lies beyond the scope of the present paper.

Another possible application of graph databases is in providing an expressive data model in which relational databases can be embedded. The need for such an embedding is motivated by the fact that the vast majority of data underpinning the Web are stored in relational databases and relational databases have a proven track record of scalability, efficient storage, optimized query execution, and reliability. However, as compared to relational databases, graph databases are more expressive and data represented in graph databases in general, and in RDF databases in particular can be interpreted, processed and reasoned over by software agents. We shall discuss the embedding of relational databases in graph databases in more detail later on.

The remaining of the paper is organized as follows. Section 2 presents the formal model, namely, hyper nodes, hyper edges, and graph databases; Sect. 3 introduces the operations over hyper nodes and hyper edges; Sect. 4 discusses equational constraints and ISA hyper edges; Sect. 4 presents the embedding of relational databases as graph databases; and Sect. 6 contains concluding remarks and suggestions for future work. Proofs of theorems are omitted due to lack of space.

We emphasize that, although theoretical in nature, our work uses only basic and well known mathematical concepts, namely functions and their basic operations.

2 The Formal Model

In this section, we first define formally the concepts of hyper node and hyper edge over a property graph G and then the concept of graph database over G .

2.1 Hyper Nodes

We have seen in the introduction that (a) a hyper node H over a property graph G is a set of nodes of G having the same properties and (b) the common properties of the nodes in H become the properties of H . In other words, H is an abstraction of a set of nodes and their (common) properties.

Definition 1 (Hyper Node). *Given a property graph G , a hyper node over G is a set H of nodes of G having the same set of properties. Moreover, if $\{p_{A_i} : H \rightarrow A_i / i = 1, 2, \dots, k\}$ is the set of (common) properties of the nodes in H , then the set $\{P_{A_i} : H \rightarrow A_i / i = 1, 2, \dots, k\}$ of properties of H is defined as follows: $P_{A_i}(n) = p_{A_i}(n)$ for all nodes n in H .*

Note that the properties of a hyper node are “induced” by those of its constituent nodes. Indeed, as nodes are added or deleted from the hyper node, the (extensions of the) properties P_{A_i} change.

Also note that, as the properties of a node satisfy the “no parallel properties” assumption, so do the properties of a hyper node. As a consequence, we shall use the notation P_{A_i} to denote unambiguously the property $H \rightarrow A_i$ within a hyper node H ; and we shall use the notation Hp_{A_i} to denote the property $H \rightarrow A_i$ in case there is a property $H' \rightarrow A_i$ in a different node H' having the same codomain A_i .

Henceforth, we shall refer to the set $\{A_i/i = 1, 2, \dots, k\}$ of all targets in H as the *basis* of H . As we mentioned in the introduction, each $A - i$ is an attributes in the sense of the relational model; and it is associated with a set of values called its *domain* and denoted by $dom(A_i)$.

Intuitively, a hyper node should be regarded as a container collecting together nodes (from the underlying graph G) that have the same properties and that are of interest in a specific application.

In practice, a hyper node can be created by (a) asking the system to allocate a new identifier (b) declaring a label H for the hyper node and (c) declaring a set of properties. Once this is done, we can insert in H any node n of interest whose properties “agree” with those of H . The insertion is done by declaring n as an instance of H .

For example, in Fig. 3, nodes 1 and 2 have each the two properties of H , namely p_{Age} and p_{Job} . Therefore, in the absence of any condition they qualify as members of H . In other words, a hyper node can be seen as a schema and any node of G conforming to it can be a member. Clearly, one may define as many hyper nodes over G as needed for a specific application.

2.2 Hyper Edges

Given two hyper nodes H and H' , there might be several edges of G connecting nodes of H to nodes of H' . It is a set of such edges that we call a hyper edge from H to H' . The only requirement is that all edges in the set have the same label.

Definition 2 (Hyper Edge). *Let G be a property graph and let H, H' be hyper nodes over G . A hyper edge from H to H' is a set E of edges from nodes of H to nodes of H' such that they all have the same label.*

Note that a hyper edge can be associated with one or more properties (e.g. the date of its creation), in much the same way as an edge of G can be associated with a set of properties. Also note that a hyper edge sets up a binary relation between the nodes of H and the nodes of H' (we shall come back to this remark in the following section).

It is important to note that we do *not* require that all edges of G from nodes of H to nodes of H' having the same label belong to the hyper edge. It is up to the designer to determine which edges of G , and with what label, will be included

in a hyper edge from H to H' . What we *do* require is that whatever the edges included in a hyper edge, they must all have the same label. Figure 4 shows five edges of a property graph G and two hyper edges E and E' .

2.3 Graph Databases

Roughly speaking, a graph database over a property graph G simply collects together and manages data of interest from G . Formally, a graph database is defined as follows.

Definition 3 (Graph Database). *Let G be a property graph. A graph database over G is a property graph whose nodes and edges are hyper nodes and hyper edges over G .*

In other words, a graph database over a property graph G is simply a property graph of higher level than G . Note that G can be viewed as a “data space” of interest (e.g. G could be a social network); and a graph database over G can be viewed as a “data-generated schema” containing a subset of data of interest from G .

Also note that as a graph database is a property graph, each of its hyper nodes can be associated with one or more properties, *in addition* to the properties induced by its constituent nodes. Similarly, each hyper edge can be associated with one or more properties of its own, *in addition* to the properties associated with its constituent edges.

For example, the hyper node H of Fig. 3(b) has the properties P_{Age} and P_{Job} induced by its constituent nodes 1 and 2. In addition to these properties we might want to associate H with a property giving the date of its creation. This is a property that refers to the hyper node H itself and has nothing to do with the properties of the nodes 1 and 2: it’s simply information regarding H . As such, this information can be viewed as metadata with respect to the data contained in H . Similarly, a hyper edge can be associated with a property saying whether the binary relation that the hyper edge represents is functional or non functional (and if functional, whether one-one, onto, etc.).

In conclusion, a graph database as defined here can be used to describe both data and metadata - and this is an important feature of our model.

3 The Query Language

In a graph database we would like to combine hyper nodes and hyper edges to derive new ones, in much the same way as in a relational database one combines tables to derive new ones (using relational algebra operations). To this end we define in this section a set of operations to which we shall refer (collectively) as the “graph algebra”. A query over a graph database is then defined to be any well formed expression whose operands are hyper nodes and/or hyper edges and whose operations are among those of the graph algebra. Due to lack of space we present only the definitions of operations for hyper nodes. Their definiyons

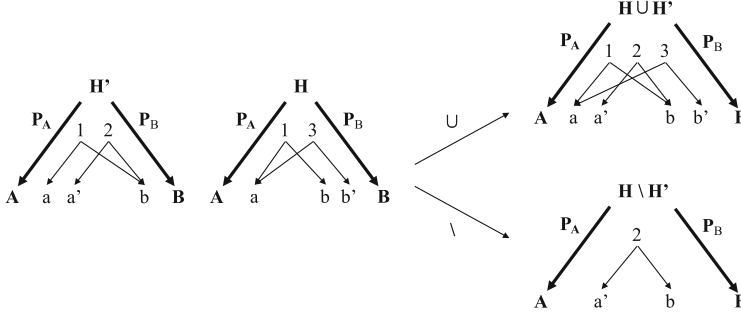


Fig. 5. Union and difference of hyper nodes

for hyper edges are quite similar to those for hyper nodes (recall that, like a hyper node, a hyper edge is a uniquely identified entity associated to a set of properties).

Union. The union of two hyper nodes H_1 and H_2 is a hyper node H whose set of nodes is the union of H_1 and H_2 and whose properties are those of H_1 “extended” to $H_1 \cup H_2$, as shown in Fig. 5.

Definition 4 (Union). Let H_1 and H_2 be two hyper nodes with common basis $\{A_i/i = 1, 2, \dots, k\}$, and with properties $P_{11}, \dots, P_{1k}$ and $P_{21}, \dots, P_{2k}$, respectively. Then the union of H_1 and H_2 denoted by $H_1 \cup H_2$ is a hyper node H with properties $P_1, \dots, P_k$ defined as follows: $H = H_1 \cup H_2$ $P_i(n) = P_{1i}(n)$ if n is in H_1 and $P_i(n) = P_{2i}(n)$ otherwise, for all nodes n in $H_1 \cup H_2$, $i = \{1, 2, \dots, k\}$.

Note that if a node belongs to both H_1 and H_2 it will have the same properties in both hypernodes (i.e. a node of the underlying graph G is associated with the same properties independently of the hyper nodes to which it might belong).

Difference. The difference between a hyper node H_1 and a hyper node H_2 is a hyper node H whose set of nodes is the difference of H_1 and H_2 and whose properties are those of H_1 “restricted” to $H_1 \setminus H_2$, as shown in Fig. 5.

Definition 5 (Difference). Let H_1 and H_2 be two hyper nodes with common basis $\{A_i/i = 1, 2, \dots, k\}$, and with properties $P_{11}, \dots, P_{1k}$ and $P_{21}, \dots, P_{2k}$, respectively. Then the difference of H_1 and H_2 denoted by $H_1 \setminus H_2$ is a hyper node H with properties $P_1, \dots, P_k$ defined as follows: $H = H_1 \setminus H_2$, $P_i(n) = P_{1i}(n)$, for all nodes n in $H_1 \setminus H_2$, $i = 1, 2, \dots, k$.

Restriction. The restriction operation takes as input a hyper node H and a subset S of H and restricts all properties of H to S .

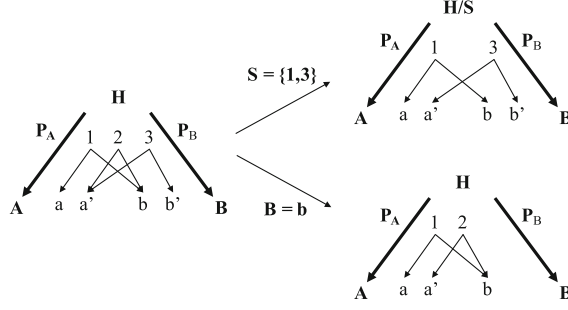


Fig. 6. Hyper node restriction

Definition 6 (Restriction). Let H be a hyper node with properties $P_1, \dots, P_k$, and let S be a subset of H . The restriction of H to S is a hypernode whose set of nodes is S and whose properties are the restrictions $P_1/S, \dots, P_k/S$, where P_i/S denotes the restriction of function P_i to S .

We note that the subset S can be defined either explicitly (by enumerating the nodes belonging to it) or implicitly (by giving property values and computing S through function inverses). Figure 6 illustrates these two ways of defining the set S used in the definition of a hyper node restriction: when S is given explicitly (i.e. $S = \{1, 3\}$), we simply restrict the domain of definition of properties P_A and P_B to $S = \{1, 3\}$; and when the set S is given implicitly through the value b of P_B we compute S as follows: $S = P_A^{-1}(b) = \{1, 2\}$.

Another way to define S implicitly is using properties of H whose codomains are associated to the same domain of values. For example, suppose that properties P_A and P_B have $\text{dom}(A) = \text{dom}(B)$. Then we can specify S as follows: $S = \{n \in H / P_A(n) = P_B(n)\}$.

Projection. The projection operation takes as input a hypernode H and a subset X of the properties of H and removes from H all properties P_i that are not in X .

Definition 7 (Projection). Let H be a hyper node with properties $P_1, \dots, P_k$, and let X be a subset of the set of properties of H . The projection of H on X , is a hyper node, denoted by $\pi_X(H)$, with the same set of nodes as H and with properties those in X .

Figure 7 illustrates hyper node projection.

Pairing. In order to define this operation we need an auxiliary definition, namely the pairing of two functions.

Definition 8 (Pairing of Functions). Let $f : X \rightarrow Y$ and $g : X \rightarrow Z$ be two functions with common source. We define the pairing of f and g , denoted by $f \wedge g$, to be the function $f \wedge g : X \rightarrow Y \times Z$ defined by: $f \wedge g(x) = (f(x), g(x))$.

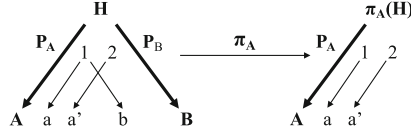


Fig. 7. Hyper node projection

Figure 8(a) shows an example of pairing two functions. Clearly, the definition of pairing can be extended to more than two functions with common source in the obvious way.

It is important to note that pairing works as a “tuple constructor”. Indeed, given an element x in the common source of two or more functions, the pairing puts together their values on x to create a tuple; and the element x works as this tuple’s identifier. When applied to a hyper node, pairing creates a set of tuples by pairing the hyper node properties as stated in the following definition.

Definition 9 (Pairing of a Hyper Node). *Let H be a hyper node with properties $P_1, \dots, P_k$. The pairing of H , denoted by $\text{pair}(H)$, is defined to be a hyper node with the same set of nodes as H and the single property $P_1 \wedge P_2 \wedge \dots \wedge P_k$.*

Figure 8(b) shows an example of pairing a hyper node.

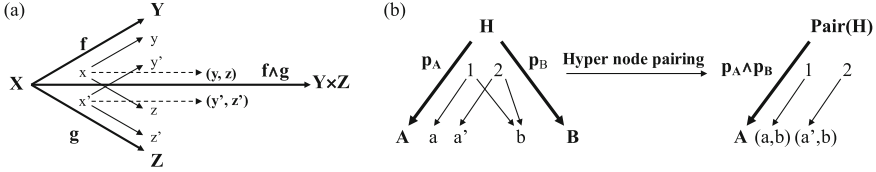


Fig. 8. Hyper node pairing

Product. In order to define this operation we need an auxiliary definition, namely the product of two functions.

Definition 10 (Product of Functions). *Let $f : X \rightarrow Y$ and $g : X' \rightarrow Y'$ be any two functions. We define the product of f and g , denoted by $f \times g$ to be the function $f \times g : X \times X' \rightarrow Y \times Y'$ defined by: $f \times g(x, x') = (f(x), g(x'))$.*

Figure 9 shows an example of product of two functions. Clearly, the definition of product can be extended to more than two functions in the obvious way.

Definition 11 (Product of Hyper Nodes). *Let H and H' be two hyper nodes. The product of H and H' , denoted by $H \times H'$, is defined as follows: $H \times H' = \text{pair}(H) \times \text{pair}(H')$*

Note that the product $H \times H'$ contains pairs of nodes and a single property as shown in the example of Fig. 10.

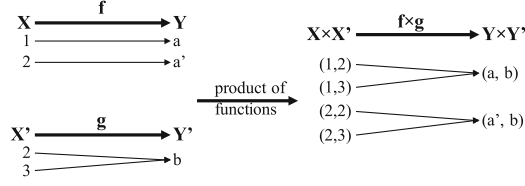


Fig. 9. Product of two functions

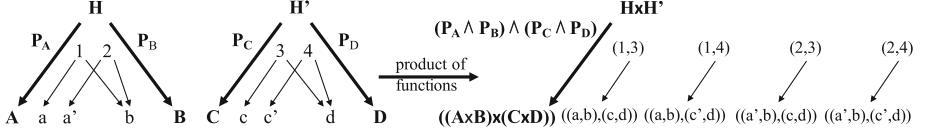


Fig. 10. Product of hyper nodes

Function Renaming. Roughly speaking, renaming a hyper node H whose basis is $A_1, \dots, A_k$ means replacing the names $H, A_1, \dots, A_k$ with possibly new names $H', A'_1, \dots, A'_k$, *without* changing the nodes or their property values.

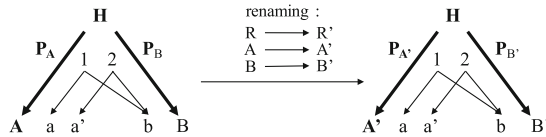
Definition 12 (Hyper Node Renaming). Let H be a hyper node with properties $P_{A_1}, \dots, P_{A_k}$. A renaming function on H is an injective function r that associates the names $H, A_1, \dots, A_k$ with (possibly new) names $H', A'_1, \dots, A'_k$ such that:

- $H = H'$ (as sets of nodes)
- $\text{dom}(A_i) = \text{dom}(A'_i), i = 1, \dots, k$
- $P_{A_i}(n) = P_{A'_i}(n)$, for all nodes n in H , $i = 1, \dots, k$

Then H' is said to be a renaming of H .

Figure 11 illustrates this definition. We note that when we say “possibly new names” in the above definition we simply mean that it is not necessary to change all names (some names may remain unchanged).

In what follows we shall refer to the set of operations introduced in this section as the *graph algebra*. We note that these operations are not independent from each other, in the sense that some of them are defined in terms of others. For example, pairing can be defined using product. Indeed, let $f : X \rightarrow Y$ and $g : X \rightarrow Z$ be two functions with common source. Then to define their pairing,


 Fig. 11. Hyper node renaming $g \circ p = p' \circ f$

based on their product, it is sufficient to restrict the product $f \times g$ to the set of pairs $\{(i, i)/i \in X\}$ and define $f \wedge g(i) = f \times g(i, i)$ for all x in X . However, defining a minimal set of operations for the graph algebra lies outside the scope of the present paper.

We note that, in addition to the operations of the graph algebra, we can use set theoretic operations (Cartesian product of sets, set union, set difference, ...), as well as the usual operations on functions (in particular, function composition).

4 Semantic Constraints

In this section we present semantic constraints that graph databases might be required to satisfy. We consider two types of constraints: (a) equational constraints among the properties of a hyper node and (b) ISA hyper edges between hyper nodes. The definition of equational constraint is motivated by the concept of functional dependency in relational databases while the definition of ISA hyper edge is motivated by the concept of ISA link in semantic models.

4.1 Equational Constraints

As we have seen earlier, a hyper node H over a property graph G has a set of properties induced by its constituent nodes. Now these properties might depend on each other in various ways. In this paper we consider only one type of dependency among the properties of a hyper node, namely equational constraints. In defining an equational constraint we use the following notation: if $X = P_{A_1}, P_{A_2}, \dots, P_{A_k}$ is a set of properties of H then we use $\wedge X$ to denote the pairing $P_{A_1} \wedge P_{A_2}, \dots, P_{A_k}$.

Definition 13 (Equational Constraint). *Let H be a hyper node with set of properties P . An equational constraint in H is an expression of the form $f : \wedge X \rightarrow \wedge Y$, where X and Y are sets of properties of H . We say that f holds in H if $f \circ \wedge X = \wedge Y$.*

For example, if P_A , P_B and P_C are properties of H then $f : P_A \wedge P_B \rightarrow P_C$ is an equational constraint in H . Intuitively, the meaning of this constraint is that the values of P_C are determined by the values of $P_A \wedge P_B$ (using f). When defining a hyper node one may declare equational constraints that the hyper node must satisfy. A hyper node that satisfies its constraints is called consistent and otherwise inconsistent. As in the case of traditional databases, consistency is checked during updates (i.e. when a property is added to the hyper node or when the values of one or more properties are modified). The following theorem expresses three important properties of equational constraints.

Theorem 1. *Let H be a hyper node of a graph database. Then the following hold:*

1. *if Y is a sub-pairing of X then $X \rightarrow Y$ holds for all pairings X, Y of properties of H*

2. if $X \rightarrow Y$ holds then $X \wedge Z \rightarrow Y$ holds, for all pairings X , Y and Z of properties of H
3. if $f : X \rightarrow Y$ and $g : Y \rightarrow Z$ hold then so does $g \circ f : X \rightarrow Z$, for all pairings X , Y and Z of properties of H

These three properties correspond to Armstrong's axioms for functional dependencies in relational databases. An interesting question is whether these properties constitute an axiomatization of equational constraints - a topic lying beyond the scope of this paper.

4.2 ISA Hyper Edges

In its most general form a hyper edge from hyper node H to hyper node H' is a set of edges from nodes of H to nodes of H' such that they all have the same label. Therefore a hyper edge sets up a (directed) relation from H to H' . Of particular interest are hyper edges that connect each node of H to at most one node of H' . Such hyper edges set up a function (partial or total) from H to H' . We call such hyper edges *functional hyper edges*; and from now on, when we say “hyper edge” we shall mean “functional hyper edge”.

Note that functional hyper edges from H to H' can be combined with the properties of H and H' (which are also functions) to produce interesting results. In this section, we focus on hyper edges that represent one-one functions as they model ISA relations.

Definition 14 (ISA Hyper Edge). *A hyper edge E is called an ISA hyper edge if E is injective.*

Figure 12 shows an ISA hyper edge E from hyper node H to hyper node H' . Clearly, if we compose E with each property of H' we obtain new properties of H , namely p'_C and p'_D (in dotted lines). Such properties can be considered as properties of H “inherited” from H' through the ISA hyper edge E .

The intuitive interpretation of property inheritance is that an ISA hyper edge E maps identifiers of individuals in its source to identifiers of the *same* individuals in its target.

For example, a division manager in a company is also an employee of the company, therefore he is identified in two different ways: as an employee and as a manager. Viewed as a manager, he has the properties of an employee together with the additional properties that a manager has. This situation is described by a hyper edge $E : Manager \rightarrow Employee$ such that, for each manager identifier i , $E(i)$ is the identifier of that manager seen as an employee. It is therefore natural that i be associated with the properties of employee *in addition* to his properties as a manager.

We note that the one-one property of an ISA hyper edge has to be maintained during updating. In other words, this property acts as a constraint that needs to be verified during hyper edge updating (e.g. an addition of an edge of G to an ISA hyper edge would be accepted only if it does not violate the one-one property).

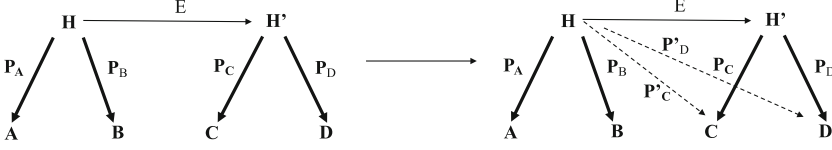


Fig. 12. ISA hyper edge E and property inheritance (P_C and P_D are inherited by H)

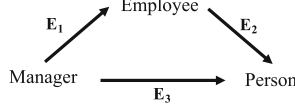


Fig. 13. Parallel path equality: $E_3 = E_2 \circ E_1$

Moreover, the one-one property is not the only property that an ISA hyper edge must satisfy. Indeed, ISA hyper edges must also satisfy what we call “Parallel Path Equality Property” (PPE Property for short).

To understand this property, consider the ISA hyper edges, E_1 , E_2 and E_3 , as shown in Fig. 13. Consider now the two paths of ISA hyper edges from *Manager* to *Person*, namely the direct path E_3 and the indirect path $E_1;E_2$. Intuitively, any manager identifier i corresponds to one and only one person identifier *independently* of the path followed in order to find that identifier in *Person*. This means that we must have: $E_3(i) = (E_2 \circ E_1)(i)$ for all managers i ; in other words, we must have: $E_3 = E_2 \circ E_1$.

Calling “parallel paths” two or more paths with the same source and the same target, we have the following definition.

Definition 15 (Parallel Path Equality Constraint). *Let H and H' be two hyper nodes. Let $P_i = E_{i_1}, \dots, E_{i_k}$ be a set of n parallel ISA paths from H to H' , $i = 1, \dots, n$. Let $\text{comp}(P_i) = E_{i_k} \circ \dots \circ E_{i_1}$ be the composition of the ISA hyper edges along the path P_i , $i = 1, \dots, n$. Then the Parallel Path Equality Constraint (PPE Constraint) is defined as follows: $\text{comp}(P_1) = \dots = \text{comp}(P_n)$.*

It should be noted that the one-one property acts as a constraint on individual ISA hyper edges, whereas the PPE Property acts as a constraint on the whole graph database.

5 Mapping Relational Databases to Graph Databases

In graph databases relationships between data are represented by means of edges between nodes, in sharp contrast to relational databases, where relationships between data in different tables are represented by means of values appearing in tuples of the two tables. As a consequence, graph databases scale more naturally to large data sets as they do not require expensive join operations to compute relationships. In addition, graph databases are more flexible than relational databases as they do not rely on a rigid schema.

On the other hand, as we mentioned in the introduction, the vast majority of data underpinning the Web are stored in relational databases, hence the need for embedding relational databases in graph databases. A survey of current approaches regarding the mapping of relational databases in graph databases, and in particular in RDF databases can be found in [13].

In this section we show how such an embedding can be done by (a) showing how a relational table can be mapped as a hyper node in our model, (b) how operations on relational tables can be mapped as operations on hyper nodes and (c) how the functional dependencies of a relational table can be mapped as equational constraints in a hyper node.

Although embeddings of relational tables to graphs have been proposed in the past, they rely mostly on ad-hoc methods. One notable exception is the work presented in [8], where a systematic embedding of relational databases to property graphs is proposed including the embedding of queries and of key dependencies. In that work, the authors consider each tuple of a relational table R as a function t from the attributes of R to the corresponding attribute domains, as depicted in Fig. 14(a). Let's call this approach the “tuples-as-functions” approach, which is the usual way of viewing tuples in the relational model. Following this approach, the authors of [8] model each tuple identifier t as a node $n(t)$ of a property graph, with the set of pairs $\{(A, t(A))/A \text{ is an attribute of } R\}$ as the set of its property-value pairs; and they model the table R as the set of nodes $\{n(t)/t \text{ is a tuple in } R\}$.

Our approach is fundamentally different than that of [8] in that we consider as functions the attributes of R rather than its tuples. In our “attributes-as-functions” approach, each attribute A of R is seen as a function f_A from the set of tuple identifiers of R to the domain of A , as depicted in Fig. 14(b). (the tuples of R can then be reconstructed by pairing all functions f_A). In fact, this is the approach followed by column databases, as for example in MonetDB [9]. Following this approach we model each tuple identifier t as a node $n(t)$ of a property graph (as in [8]) but this time R is mapped as a hyper node whose set of nodes is the set $\{n(t)/t \text{ is a tuple in } R\}$ and whose set of properties is $\{f_A/A \text{ is an attribute of } R\}$.

By the way, following the “tuples-as-functions” approach, a table is seen as a set of as many functions as there are tuples in the table; whereas, following the “attributes-as-functions” approach, a table is seen as a set of as many

R		
	Age	Job
t_1	56	actor
t_2	21	singer

(a) tuples-as-functions:
 $t_1(\text{Age}) = 56, t_1(\text{Job}) = \text{Actor}$
 $t_2(\text{Age}) = 21, t_2(\text{Job}) = \text{Singer}$

R	Age	Job
t_1	56	actor
t_2	21	singer

(b) attributes-as-functions:
 $\text{Dom}(R) = \text{tuple identifiers}$
 $p_{\text{Age}}(t_1) = 56, \text{Age}(t_2) = 21$
 $p_{\text{Job}}(t_1) = \text{Actor}, \text{Job}(t_2) = \text{Singer}$

Fig. 14. Two ways of looking at a relational table

functions as there are attributes in R . Therefore, the “attributes-as-functions” approach leads to a compact representation of a table (and this is precisely what is exploited in column databases to achieve higher performance in data analytics).

5.1 Mapping a Relational Table

In order to map a relational table R in a graph database as defined in this paper, we view the table name R as an attribute of the table itself having the set of tuple identifiers as its domain (see Fig. 15(a)) and then we map R as a hyper node $H(R)$ (see Fig. 15(b)). Formally we have:

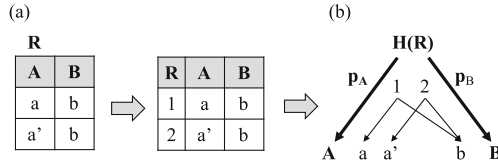


Fig. 15. Mapping a relational table as a hyper node

Definition 16 (Mapping a Relational Table). Let G be a property graph and let $R(A_1, \dots, A_k)$ be a relational table. The image of R in G is defined to be a hyper node $H(R)$ over G such that: (a) each tuple identifier t in R is mapped as a node identifier $n(t)$ of $H(R)$ and (b) each attribute A of R is mapped as a property P_A of $H(R)$ defined by: $P_A(n(t)) = t(A)$.

Note that this mapping is invertible (i.e. one can recover the table R from the hyper node $H(R)$).

5.2 Mapping Relational Algebra Operations

In this section we show how each operation of the relational algebra is mapped as an operation of our graph algebra, and by consequence, how a relational algebra expression is mapped as a graph algebra expression.

Union. In the relational model, the union of two tables R_1 and R_2 is defined only if the two tables have the same set of attributes, and it is mapped as the union of their images $H(R_1)$ and $H(R_2)$.

Definition 17 (Mapping Union). Let G be a property graph, and R_1, R_2 two relational tables with the same attribute set. The image of $R_1 \cup R_2$ in G is defined to be the union $H(R_1) \cup H(R_2)$ of the images of the two tables.

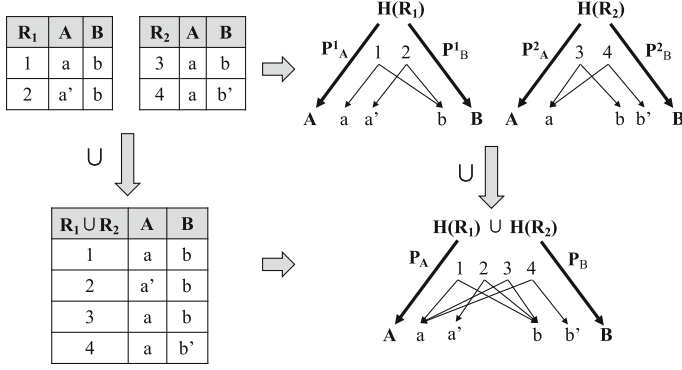


Fig. 16. Mapping relational union as union of hyper nodes

Figure 16 illustrates this definition.

Difference. As for the union, the difference of two tables R_1 and R_2 is defined only if the two tables have the same set of attributes, and it is mapped as the difference of their images $H(R_1)$ and $H(R_2)$.

Definition 18 (Mapping Difference). Let G be a property graph, and R_1, R_2 two relational tables with the same attribute set. The image of $R_1 \setminus R_2$ in G is defined to be the difference $H(R_1) \setminus H(R_2)$ of the images of the two tables.

Figure 17 illustrates this definition.

Selection. To see how relational selection is mapped, consider the selection $\sigma_{A=a}(R)$ as shown in Fig. 18 and let $S = p^{-1}(a)$. Then R is mapped to the hyper node $H(R)/S$ (recall that $H(R)/S$ is the hyper node $H(R)$ with each of its properties restricted to the set S).

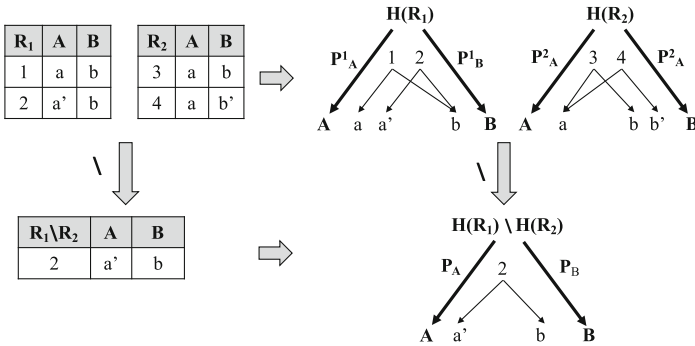


Fig. 17. Mapping difference as difference of hyper nodes

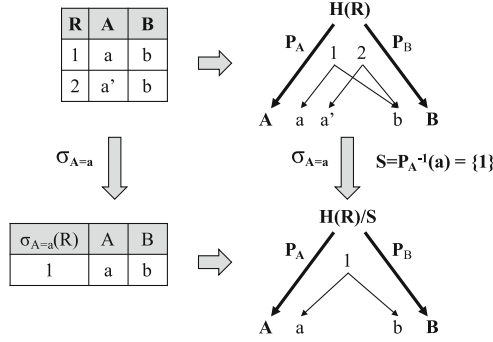


Fig. 18. Mapping relational selection as hyper node restriction

Definition 19 (Mapping Selection). Let R be a relational table and $\sigma_C(R)$ the selection of R under condition C . Then $\sigma_C(R)$ is mapped to $H(R)/S$, where the set S is defined as follows:

- if $C = (A = a)$ then $S = P_A^{-1}(a)$
- if $C = \neg(A = a)$ then $S = R \setminus P_A^{-1}(a)$
- if $C = (A = a) \text{ AND } (B = b)$ then $S = P_A^{-1}(a) \cap P_B^{-1}(b)$
- if $C = (A = a) \text{ OR } (B = b)$ then $S = P_A^{-1}(a) \cup P_B^{-1}(b)$
- if $C = (A = B)$, where $\text{dom}(A) = \text{dom}(B)$ then $S = \{n \in H(R) / P_A(n) = P_B(n)\}$

Figure 18 illustrates the mapping of selection. Conditions of the form $(\text{attribute} = \text{value})$ such as $A = a$ or $B = b$ in the above definition are known as “elementary conditions” in relational model terminology. In general, the condition C in a selection operation is a Boolean combination of elementary conditions. Although in the above definition we only considered conditions with one elementary condition or a combination of two elementary conditions, the extension to more than two elementary conditions should be obvious.

Projection. In the relational model, the projection of a table R over a set of attributes X is the table obtained from R by (a) keeping only the columns contained in X and (b) removing repeated tuples in the result (if any). To see how relational projection is mapped in our model, consider the table R and its image $H(R)$ as shown in Fig. 19. The projection $\pi_A(R)$ is mapped to the hyper node $\pi_A(H(R))$ which results from $H(R)$ if we keep only the property P_A .

Definition 20 (Mapping Projection). Let G be a property graph, let R be a relational table, and let $H(R)$ be the image of R in G . Let $X = \{A_1, \dots, A_m\}$ be a subset of the attribute set of R and $P_1, \dots, P_m$ the properties of $H(R)$ corresponding to the attributes $A_1, \dots, A_m$, respectively. The image of $\pi_X(R)$ in G is defined to be the projection $\pi_X(H(R))$ of the image $H(R)$ of R .

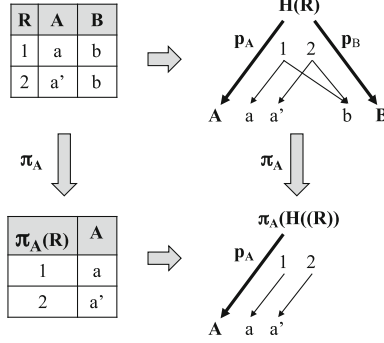


Fig. 19. Mapping projection as hyper node projection

Product. In the relational model, the product of two tables R_1 and R_2 is always defined even if the two tables have different sets of attributes. To see how the relational product is mapped, consider the tables R_1 and R_2 and their images $H(R_1)$ and $H(R_2)$ as shown in Fig. 20. The product $R_1 \times R_2$ is mapped to the product $H(R_1) \times H(R_2)$ of the images of the two tables.

Definition 21 (Mapping Product). Let G be a property graph, and R_1, R_2 two relational tables. The image of $R_1 \times R_2$ in G is defined to be the product $H(R_1) \times H(R_2)$ of the images of the two tables.

Figure 20 illustrates this definition.

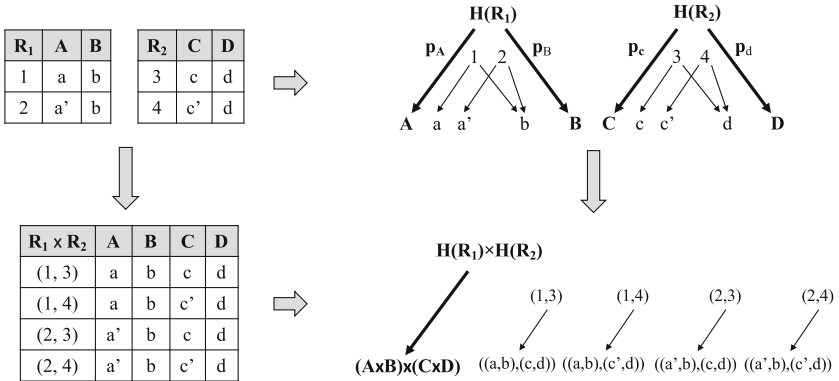


Fig. 20. Mapping the product of two tables as the product of hyper nodes (recall that $H(R) \times H(S) = (p_A \times p_B) \times (p_C \times p_D)$)

Renaming. In the relational model, a renaming function over a table $R(A_1, \dots, A_k)$ is an injective function f that associates the names $R, A_1, \dots, A_k$ with names $R', A'_1, \dots, A'_k$ in the predefined sets from which names of tables and attributes are drawn, such that the tuples of R remain unchanged in the renamed table R' . More precisely, if $f(A_i) = A'_i$, $i = 1, \dots, k$, then the tuples t' of R' are defined by: $t'(f(A'_i)) = t(f(A_i))$, for all $i = 1, \dots, k$ and for all tuples t in R . The image of the renaming of R is defined to be the renaming under f of the hyper node $H(R)$ to which R is mapped

Definition 22 (Mapping Renaming). *Let G be a property graph, let $R(A_1, \dots, A_k)$ be a relational table, and let f be a renaming of R . Then the renamed table $f(R)(f(A_1), \dots, f(A_k))$ is mapped to the renaming of $H(R)$ under f .*

Figure 21 illustrates this definition. We end this section by noting that, although not difficult, proving the correctness of mapping the relational algebra operations to graph algebra expressions is a long and tedious task that we omit here because of lack of space.

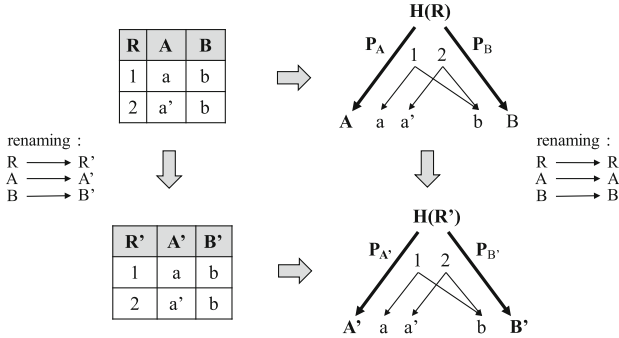


Fig. 21. Mapping relational renaming as a hyper node renaming

5.3 Mapping Functional Dependencies

In the relational model, given a table R , a functional dependency over R is an expression of the form $X \rightarrow Y$ where X and Y are attribute sets in R . We say that $X \rightarrow Y$ holds in R (or that R satisfies $X \rightarrow Y$) if whenever the equality $t(X) = t'(X)$ is true in R then so is the equality $t(Y) = t'(Y)$. This means that a functional dependency $X \rightarrow Y$ is actually a function from X to Y , whose extension can be obtained by projecting R over $X \cup Y$. The question now is how to map this concept to the image $H(R)$ (i.e. to the hyper node to which R is mapped).

Note first that the above definition of functional dependency uses the tuples-as-functions approach, whereas in our work we use the attributes-as-functions

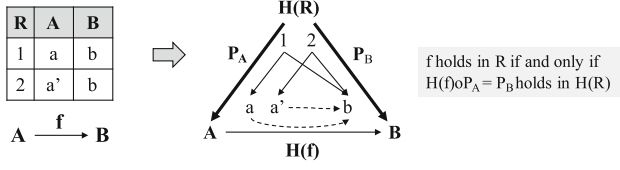


Fig. 22. Mapping a functional dependency as an equational constraint in a hyper node

approach. Therefore, first, we need to express the definition of functional dependency using the attributes-as-functions approach. This is stated in the following proposition.

Proposition 1. *Let $X \rightarrow Y$ be a functional dependency over R . Then we have: $X \rightarrow Y$ holds in R if and only if there is a unique function $h : X \rightarrow Y$ such that $h \circ \text{proj}_X = \text{proj}_Y$.*

Note that in the above proposition the projections proj_X and proj_Y are seen as functions (in the way explained earlier) and that a functional dependency is defined through an equational constraint on the table R . Using the above proposition, we can now define how a functional dependency over R can be mapped in the image $H(R)$. Given a set $X = A_1, \dots, A_k$ of attributes from R , we use the notation $\bar{X}$ to denote the set of corresponding properties in $H(R)$; and we use the notation $\wedge \bar{X}$ to denote the pairing of the properties in $\bar{X}$.

Definition 23 (Mapping a Functional Dependency). *Let G be a property graph, let R be a relational table and let $f : X \rightarrow Y$ be a functional dependency over R , where X and Y are sets of attributes appearing in R . Then the image of $X \rightarrow Y$ in $H(R)$ is defined to be $H(f) : \wedge \bar{X} \rightarrow \wedge \bar{Y}$.*

The following proposition states that this mapping is correct.

Proposition 2. *f holds in R if and only if $H(f) : \wedge \bar{X} \rightarrow \wedge \bar{Y}$ holds in $H(R)$.*

Figure 22 shows schematically how the mapping of f is done, using single attributes on both sides of the functional dependency in order to make things easier to understand.

6 Concluding Remarks

We have seen a graph database model based on property graphs. A notable feature of our model is that a graph database over a property graph is again a property graph but of higher level, therefore enjoying all properties and results known about property graphs. We have also seen a set of operations on graph databases whose well formed expressions constitute the query language of a graph database. Additionally, we have defined two semantic constraints that

a graph database might be required to satisfy, namely equational constraints within a hyper node and ISA relations between hyper nodes.

We have demonstrated the expressive power of our model by showing how a relational database, together with functional dependencies, can be embedded in our model in the form of a graph database; and how queries over relational tables can be mapped as queries over the corresponding graph database.

Future work includes three main research lines. First, we would like to use equational constraints to develop a decomposition theory for graph databases that parallels the one developed for relational databases based on functional dependencies. We believe that most of the results found in the context of relational schema design based on table decomposition can be mapped to graph databases.

The second research item concerns the introduction and study of additional types of semantic constraints, most notably inclusion dependencies and alignment functions between nodes in the bases of two hyper nodes (inspired by similar concepts developed in the context of relational databases).

The third research item concerns the relationship between our graph databases and RDF databases, as the latter form the largest subset of NoSQL databases.

References

1. Angles, R., Gutierrez, C.: Survey of graph database models. *ACM Comput. Surv. (CSUR)* **40**(1), 2012–2015 (2008)
2. Wood, P.T.: Query languages for graph databases. *ACM SIGMOD Rec.* **41**(1), 50–60 (2012)
3. W3C: Resource Description Framework (RDF) Model and Syntax Specification. <https://www.w3.org/TR/PR-rdf-syntax/>
4. Levene, M., Poulouvasilis, A.: An object-oriented data model formalised through hypergraphs. *Data Knowl. Eng.* **6**(3), 205–224 (1991)
5. Easley, D., Kleinberg, J., Crowds, M.: Reasoning about a Highly Connected World. Cambridge University Press, Cambridge (2010)
6. Robinson, I., Webber, J., Eifrem, E.: Graph Databases. O'Reilly Media, New York (2015)
7. Rodriguez, M.A., Neubauer, P.: The graph traversal pattern management, graph data management techniques and applications. In: Sakr, S., Pardede, E. (eds.) IGI Global, ISBN: 9781613500538, August 2011
8. De Virgilio, R., Maccioni, A., Torlone, R.: R2G: a tool for migrating relations to graphs. In: EDBT/ICDT 2014 Joint Conference (2014)
9. Boncz, P., Manegold, S., Kersten, M.: Database architecture optimized for the new bottleneck: memory access. In: Proceedings of VLDB 1999, p. 5465 (1999)
10. Cebiric, S., Goasdoue, F., Manolescu, I.: Query-oriented summarization of RDF graphs. *Proc. VLDB Endow.* **8**(12), 1–39 (2015)
11. Campinas, S., Perry, T., Ceccarelli, D., Delbru, R., Tummarello, G.: Introducing RDF graph summary with application to assisted SPARQL formulation. In: DEXA Workshops (2012)

12. Tian, Y., Hankins, R.A., Patel, J.M.: Efficient aggregation for graph summarization. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (2008)
13. Sahoo, S.S., et al.: A survey of current approaches for mapping of relational databases to RDF. W3C RDB2RDF Incubator Group Report (2009)

Information Search, Integration, and Personalization
10th International Workshop, ISIP 2015, Grand Forks,
ND, USA, October 1-2, 2015, Revised Selected Papers
Grant, E.; Kotzinos, D.; Laurent, D.; Spyratos, N.;
Tanaka, Y. (Eds.)
2016, XI, 157 p. 82 illus., Softcover
ISBN: 978-3-319-43861-0