Inference of Shape Graphs for Graph Databases

Benoît Groz
University Paris Sud, France

Aurélien Lemay
University of Lille, France

Sławek Staworko
University of Lille, France

Piotr Wieczorek
University of Wrocław, Poland

Abstract

We investigate the problem of constructing a shape graph that describes the structure of a given graph database. We employ the framework of grammatical inference, where the objective is to find an inference algorithm that is both sound, i.e., always producing a schema that validates the input graph, and complete, i.e., able to produce any schema, within a given class of schemas, provided that a sufficiently informative input graph is presented. We identify a number of fundamental limitations that preclude feasible inference. We present inference algorithms based on natural approaches that allow to infer schemas that we argue to be of practical importance.

2012 ACM Subject Classification
Information systems → Graph-based database models

Keywords and phrases RDF, Schema, Inference, Learning, Fitting, Minimality, Containment

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.14

Funding Piotr Wieczorek: This author has been partially supported by Polish National Science Center grant NCN 2016/23/B/ST6/01438.

1 Introduction

Traditionally, in relational databases, defining the schema is the mandatory first step before a database can even be populated with data. However, novel database models, such as graph databases, quite intentionally allow to store and process data without declaring any schema. This facilitates the evolution of the structure of a database while the applications around it are being developed. In fact, often a suitable schema formalism is proposed after a particular database model has established its place in practice. In those circumstances a natural problem of schema inference arises: given a schema-less database construct a schema that captures the structure of the database. This problem has been identified as an important research direction [1] and is well motivated since the knowledge of database structure is instrumental in any meaningful data processing task such as querying or transformation.

In the present paper, we present a principled approach to the problem of schema inference for graph databases. We consider RDF graphs and Shape Expression Schemas (ShEx) [55, 47]. ShEx builds on the success of XML Schema and allows to describe the structure of an RDF graph by defining patterns of arrangement of RDF nodes. More precisely, ShEx specifies a collection of node types, each type defined by a regular expression that constrains the types of the outbound neighborhood of a node. Take for instance the RDF graph storing bug reports, presented in Figure 1 together with its shape expression schema. The schema requires a bug report to have a description and a user who submitted it. Optionally, a bug report may have an employee who verified it. Also, a bug report can have a number of related bug reports. A user has a name and an optional email address while an employee has a name and a mandatory email address. We point out that just like with XML Schema the
nodes of the RDF graph need not be typed and it is the task of a validation algorithm to find a valid node typing [42, 35, 47]. Furthermore, some nodes may need to be assigned more than one type, for instance the node emp, needs to satisfy the types User and Employee.

We focus our investigation on a practical subclass ShEx$_0$ that defines types with collections of atoms with arities ranging over 1, ?, +, and *. This fragment does not allow disjunction or union types but can adequately approximate schemas with those features. ShEx$_0$ is particularly suited to capture the topology of RDF graphs obtained by exporting relational databases in a number of formalisms proposed for this task, such as R2RML, Direct Mapping, and YARRRML [46, 45, 13]. More importantly, the class ShEx$_0$ enjoys a sought-after feature of an equivalent graphical representation in the form of a shape graph whose nodes are types and edges are labeled by a symbol and a multiplicity (see Figure 1).

We present a principled approach to the problem of graph database schema inference, where we start with the fundamental question: What is an inference algorithm? We answer it using the framework of grammatical inference [28], which in recent years has been successfully applied to a number of database formalisms ranging from queries [17, 48] to schemas [7, 22] and transformations [38, 36]. In essence, an inference algorithm needs to be both sound, i.e., produce a schema that validates the input graph, and complete, i.e., able to infer any goal schema with a sufficiently informative input graph. This formal framework allows us to tackle another fundamental question: When is inference feasible? Indeed, we identify classes of schemas that are not learnable, which reveals two principal challenges in schema inference: 1) distinguishing unbounded arities * and + from bounded ones 1 and ?, and 2) distinguishing recursive types from their finite non-recursive unravelings.

A suitable ShEx$_0$ inference algorithm needs to identify the types and their definitions. ShEx$_0$ assigns to every node of a graph a set of types by using the notion of embedding, which is an extension of the standard graph simulation. Consequently, our first attempt is to use graph simulation to identify sets of nodes whose outbound neighborhood shares structural similarity, and therefore, should have the same type. Interestingly, this approach yields a very reasonable inference algorithm that generalizes the structural information of the input graph, and in particular, easily introduces recursion in the constructed schema. The generalization is however very eager, and in particular, the algorithm clumps together any two types that are related by subtype relation such as Employee $\subseteq$ User where every node of

$$
\begin{align*}
\text{Bug} & \rightarrow \text{descr: str}, \text{submittedBy: User}, \text{verifiedBy: Employee?}, \text{related: Bug*} \\
\text{User} & \rightarrow \text{name: str}, \text{email: str?} \\
\text{Employee} & \rightarrow \text{name: str}, \text{email: str}
\end{align*}
$$

Figure 1 An RDF graph with bug reports (top right) together with a shape expression schema (bottom) and the corresponding shape graph (top left). str is a built-in type for literal string nodes.
type *Employee* has also type *User*. Indeed, this inference algorithm produces a shape that is *singular*, having no two types that cannot be distinguished with the help of simulation, and consequently it prohibits schemas with subtypes like the schema in Figure 1.

To address this shortcoming we investigate using the *context* in which nodes are used: their *inbound* neighborhood. Indeed, the nodes of type *User* and the nodes of type *Employee* are used in different contexts, although their use may partially overlap e.g., the node *emp*$_1$ is used both as an employee and as a user. An inference algorithm based on using context information constructs a schema that belongs to the class of *deterministic* shape graph, which permits every edge label to be present at most once in a type definition. This class of schemas is incomparable with singular shape graph. For instance, the schema in Figure 1 is deterministic but is not singular. We find, however, that an indiscriminate use of the context information leads to *overfitting*, and in particular, it may fail to introduce recursion and may produce voluminous schemas. It appears that an inference algorithm must choose carefully the information to identify the set of types. We propose, as a proof of concept, a hybrid inference algorithm that extends the simulation-based approach with a modest amount of context information that consists of a single incoming edge. This algorithm produces schemas that are *context-singular*, a generalization of singular schemas that allows subtyping between types used in different contexts.

Finally, we survey a number of graph databases and inspect their schemas. Our analysis shows that *ShEx*$_0$ and the proposed subclasses lack certain features present in real-world schemas, most notably union types, but we can approximate them very well. This makes our algorithms an excellent solution for preparing the first draft of a schema to be refined by a knowledgeable architect.

**Organization.** The paper is organized as follows. In Section 2 we present basic notions, including shape graphs and their embeddings. In Section 3 we formally state the problem of inference. In Section 4 we present a simulation-based inference algorithm for the class of singular shape graphs. In Section 5 we present negative implications of unrestrained use of context information in inference. In Section 6 we propose a hybrid approach combining the simulation-based approach with elements of context information. In Section 7 we present an analysis of schemas of a number of graph databases. We discuss related work in Section 8. We summarize our findings and outline future research directions in Section 9.

## 2 Basic notions

Throughout this paper we apply a functional notation to relations, and conversely, often view functions as relations. For instance, for a binary relation $R \subseteq A \times B$ we set $\text{dom}(R) = \{a \in A \mid \exists b \in B. (a, b) \in R\}$, $\text{ran}(R) = \{b \in B \mid \exists a \in A. (a, b) \in R\}$, $R(a) = \{b \in B \mid (a, b) \in R\}$ for $a \in A$, and $R^{-1}(b) = \{a \in A \mid (a, b) \in R\}$ for $b \in B$.

**Intervals.** We use the standard notation $[i; j]$ for $0 \leq i \leq j \leq \infty$ to denote intervals that represent nonempty sets of consecutive natural numbers. By $\mathbb{N}_0$ we denote the set of all intervals. Our schema formalisms use only four *basic intervals* ($\mathbb{M}_0$) given with their shorthand notation: 1 is $[1; 1]$, $?$ is $[0; 1]$, $+$ is $[1; \infty]$ and $\ast$ is $[0; \infty]$. A *simple interval* has the form $[i; i]$ for $0 \leq i < \infty$, and often we use $i$ for $[i; i]$. Throughout this paper, we employ the point-wise addition operation $[i_1; j_1] \oplus [i_2; j_2] = [i_1 + i_2; j_1 + j_2]$. A set of intervals $M$ is *proper* if any interval is contained in an interval in $M$. For a proper $M$ we define the function $\text{fit}_M$ that maps any finite nonempty set of natural numbers (occurrences) into a smallest interval in $M$ containing it.
Graphs and schemas. We employ a generic model of a graph that we use to model both RDF and the subclass ShEx$_0$ that allows type definitions consisting of collections of atoms using basic intervals [47, 49]. Our methods do not inspect data values of literal nodes, and consequently, we will abstract them using their type alone but this requires us to record the number of some types of edges as illustrated in Figure 2a.

![Diagram](image.png)

**Figure 2** Graphs and embeddings.

When drawing graphs we typically omit the label 1 and simple intervals are denoted with natural numbers e.g., 2 stands for $[2; 2]$ which indicates two identical edges. We point out a dichotomy of node kinds and their types: literal nodes can only have literal types and non-literal nodes can only have non-literal types. While it might be tempting to remove literal types from consideration and focus on data-free graphs, we keep literal types and nodes because without them it is difficult to define reasonable schema families without subtypes.

We assume a finite set $\Sigma$ of edge labels and an enumerable set of node identifiers $N = U \cup L$, where $U$ is an infinite set of identifiers used to represent both URIs and blank nodes, and $L$ is a finite set of types of literal nodes e.g., str representing strings. We assume that $U \cap L = \emptyset$.

**Definition 1.** A graph $G$ is a triple $(N_G, E_G, \text{arity}_{G})$, where $N_G \subseteq N$ is a finite set of nodes, $E_G \subseteq (N_G \cap U) \times \Sigma \times N_G$ is a set of labeled oriented edges, and $\text{arity}_{G}$ maps every edge to an interval. A shape graph is a graph that uses only basic intervals and by ShEx$_0$ we denote the set of all shape graphs. A simple graph is a graph that assigns to every edge a simple interval and this simple interval is 1 unless the edge leads to a node in $L$. By $G_0$ we denote the set of simple graphs. A simple graph $G$ is data-free if $N_G \cap L = \emptyset$.

In Figure 2b we present a more concise version of the graphs from Figure 1 adapted to our graph model. For $M \subseteq M_0$, we write ShEx$_0(M)$ to restrict the choice of arities to $M$. For an edge $e = (n, p, m)$ we let source$\,(e) = n$, lab$\,(e) = p$, and target$\,(e) = m$. For a given node $n$ of $G$ we identify its outbound neighborhood out$_G(n) = \{ e \in E_G \mid \text{source}(e) = n \}$. Given two graphs $G_1$ and $G_2$ by $G_1 \uplus G_2$ we denote their disjoint union. In the sequel, we often refer to shape graphs as schemas and their nodes as types. We also abuse the standard tree terminology and say that nodes $n$ and $n'$ are siblings if there is a node $m$ that has outgoing edges to $n$ and to $n'$.

Embeddings. We recall next, and illustrate in Figure 2b, the notion of embedding that allows to define the semantics of shape graphs as schemas [49] and is instrumental in our inference methods. Here, by $\text{id}_A$ we denote the identity relation on a given set $A$. 
The embedding equivalent of $G$ of embedding of $R$ languages and has been successfully applied to a number of database formalisms ranging grammatical inference constructing a shape graph for a given input simple graph. To this end we adopt the

In this paper we investigate the problem of shape graph inference, which consists of constructing a shape graph for a given input simple graph. To this end we adopt the framework of grammatical inference [28], which has originally been proposed for word languages and has been successfully applied to a number of database formalisms ranging...
Inference of Shape Graphs for Graph Databases

from queries [17, 48] to schemas [7, 22] to transformations [38, 36]. In essence, the framework requires the inference algorithm to be capable of inferring any goal schema given a sufficiently informative input. We are interested in inference from positive data: the input consists of a graph that belongs to the goal language. To prevent collusion, namely a solution where a characteristic input graph encodes the goal schema using an elaborate scheme, the inference algorithm is required to be robust under extension: it must infer the goal schema even if the characteristic graph is accompanied by other potentially less informative graphs that satisfy the schema. Formally, \( G \) extends \( G' \) consistently with schema \( H \) iff there is \( G'' \) such that \( G = G' \union G'' \) and \( G, G', G'' \in L(H) \).

Definition 3. A class of shape graphs \( \mathcal{C} \) is learnable from a class of graphs \( \mathcal{G} \) iff there is an inference algorithm learner such that

- **Soundness** For every input graph \( G \in \mathcal{G} \) the inference algorithm returns a schema learner\((G) = H \) such that \( H \in \mathcal{C} \) and \( G \in L(H) \).
- **Completeness** For every goal schema \( H \in \mathcal{C} \) there exists a characteristic graph \( G_H \in L(H) \) such that for any \( G \) that extends \( G_H \) consistently with \( H \) we have learner\((G) \equiv H \).

Furthermore, we say that \( \mathcal{C} \) is learnable in polynomial time if the inference algorithm works in polynomial time. We also say that \( \mathcal{C} \) is learnable in polynomial data if there exists a polynomial that bounds the size of the characteristic graph as a function of the size of the goal schema.

As an illustrative example of grammatical inference consider the problem of inferring basic intervals \( M_0 \) from a given sample of its members \( \{i_1, \ldots, i_k\} \). Take the function \( \text{fit}_{M_0} \) that returns the smallest basic interval that contains all the elements given on the input. Naturally, \( \text{fit}_{M_0} \) is a sound inference algorithm since it returns an interval that contains all the elements given on the input. It is also a complete inference algorithm because we can construct characteristic samples for every element of \( M_0 \). For instance, for the goal interval \( * = [1; \infty] \) a characteristic sample is \( \{1, 2\} \). Indeed, \( \text{fit}_{M_0}(\{1, 2\}) = * \) but more importantly \( \text{fit}_{M_0}(X) = \star \) for any \( X \) subset of \( [1; \infty] \) that includes \( \{1, 2\} \).

Now, consider the full class of intervals \( I_0 \) and take the function \( \text{fit}_{I_0} \) as an inference algorithm. It is naturally sound since \( \text{fit}_{I_0}(X) = [\min(X); \max(X)] \) but it is not complete since it cannot infer an unbounded interval \( [i_0; \infty] \) from a finite input set. In fact, \( I_0 \) is not learnable because one can show that there is no sound and complete inference algorithm for \( I_0 \) (from positive samples alone). Next, we present a general characterization of classes of languages that are not learnable. We will use it as a tool for identifying key challenges in schema inference.

3.2 Limit point

We recall that a class of languages \( \mathcal{C} \) has the limit point property iff \( \mathcal{C} \) contains an ascending chain of languages \( L_1 \subseteq L_2 \subseteq \ldots \) whose limit point \( L_\infty = \bigcup_i L_i \) also belongs to \( \mathcal{C} \). It is a folklore result that the limit point property precludes learnability (from positive examples) [2]: essentially, no finite amount of examples from the goal language allows to distinguish between the limit point \( L_\infty \) and some language \( L_i \) in the ascending chain. We formally show that it also holds for the framework we adopt in this paper.

Proposition 4. No class of shape graphs with the limit point property is learnable from simple graphs.

The limit point argument allows us to identify a number of limitations of inference of shape expressions schemas.
Lemma 5. For any $M \subseteq \{1, ?, *, +\}$ with at least two elements the class $\text{ShEx}_0(M)$ has the limit point property.

Proof sketch. It suffices to consider sets of 2 multiplicities and in Figure 5 we present three cases. The limit point arguments for all remaining cases are constructed very similarly to $L_0$:

(a) Limit point for $\text{ShEx}(?, *)$.

(b) Limit point for $\text{ShEx}(1, ?)$.

(c) Limit point for $\text{ShEx}(\ast, *)$.

Figure 5 Limit points for various subclasses of $\text{ShEx}_0$.

ShEx($?, \ast$) in Figure 5a (details in appendix).

Consequently, we obtain the following corollary.

Corollary 6. $\text{ShEx}_0$ is not learnable from simple graphs.

A closer look at the limit point arguments offers an insight into the reasons why inferring $\text{ShEx}_0$ is not feasible. Firstly, we observe in Figure 5a that one of the challenges in distinguishing between $L_i$ and $L_\infty$ is in deciding based on finite input example when to use unbound arity $\ast$ as opposed to $?$ (similar argument holds for $+$ and $?$, etc.). Secondly, in Figures 5b and 5c we observe an analogous difficulty in deciding between other pairs of arities, but more importantly, we also observe a different difficulty: whether to choose a simple recursive type ($L_\infty$) or allow some arbitrary non-recursive unfolding of it ($L_i$). To summarize, the constructions in Figure 5 point to two key challenges of inferring shape schemas for graph databases: 1) distinguishing between bounded and unbounded arities, and 2) introducing recursive types especially from a possibly acyclic input example.

3.3 Graph aggregate

The inference algorithms that we propose in this paper construct schemas with an operation that generalizes the standard graph quotient. Recall that the graph quotient operation essentially reduces to a single node every set of nodes that are equivalent w.r.t. a given equivalence relation. In the context of schema inference, we are interested in grouping together nodes that we deem to have the same type but we point out that the underlying relation “$n$ and $m$ are deemed to have the same type” needs not be transitive in general. In fact, we do not even assume that this relation is reflexive because we may wish to intentionally ignore nodes of the input graph that we do not find informative enough.

Definition 7. A node grouping of a simple graph $G = (N_G, E_G, \text{arity}_G)$ is a collection $P \subseteq 2^{N_G}$ of sets of nodes of $G$ such that any $N \in P$ with $N \cap \mathcal{L} \neq \emptyset$ is a singleton. Given a proper set of intervals $M$, the aggregate of $G$ by $P$ w.r.t. $M$, in symbols $G \div_M P$, is the shape graph $H = (N_H, E_H, \text{arity}_H)$, where...
\[N_H = P,\]
\[E_H = \{(n, a, N') \in P \times \sum \times P \mid \exists (n, a, n') \in E_G, \ n \in N \land n' \in N'\},\]
\[\text{arity}_H(n, a, N') = \text{fit}_M \left( \bigoplus \{\text{arity}_G(n, a, n') \mid n' \in N' \land (n, a, n') \in E_G \} \mid n \in N \right).\]

The requirement \(N \cap \mathcal{L} \neq \emptyset\) follows from the dichotomy of node kinds and their types. To ensure that the aggregate is technically a (shape) graph, we assume that any subset of \(N\) is also a node identifier and for any \(\ell \in \mathcal{L}\) we interpret \(\{\ell\}\) as \(\ell\). In the sequel, when the set of intervals \(M\) is known from the context, we omit it in the aggregate and simply write \(G \div P\).

In essence, for a given input graph \(G\), our inference algorithms explore the lattice \((\mathcal{P}_G, \subseteq)\) of all node groupings \(\mathcal{P}_G\) of \(G\) from \(P_L = \{(n) \mid n \in N_G\} \to P_T = \{N_G \setminus \mathcal{L}\} \cup \{\ell\} \mid \ell \in N_G \cap \mathcal{L}\).

The structure of this lattice is consistent with the containment relation: \(P_1 \subseteq P_2\) if \(G \div P_1 \subseteq G \div P_2\). This connection to containment is essential for guiding inference algorithms with characteristic graphs. When the goal schema is \(G \div P\) for a node grouping \(P\) then for every relevant \(P' \subseteq P\) the characteristic graph needs to contain an example showing \(G \div P' \not\supseteq G \div P\).

**Example 8.** Consider \(G_0\) and \(H_0\) in Figure 2b, and take the node grouping that corresponds to the typing of \(G_0\) w.r.t. \(H_0\): \(P = \{\{b_1, b_2, b_3, b_4\}, \{u_1, u_2, e_1\}, \{e_1\}, \{\text{str}\}\}\). The aggregate \(G_0 \div_{M_0} P\) is in fact (equivalent to) \(H_0\).

The above example shows that \(G_0\) carries sufficient information that with the right partition allows to infer the goal schema \(H_0\).

The task is to identify a method that allows to group nodes of the input graph that are deemed to have the same type and at the same time can introduce unbounded arities and recursion in type definitions. A natural attempt at grouping nodes of a graph would be to use bisimulation. But as we illustrate in Figure 4 this fails to introduce unbounded arities and recursive types. In fact, it has been studied in the literature [27] and found to fail to generalize appropriately the input graph.

### 4 Singular shape graphs

In this section we address the two key challenges of inference by grouping nodes with the help of graph simulation relation. This yields an inference algorithm for a subclass of singular shape graphs. The inference algorithm allows to infer unbounded arities and recursive types even from acyclic input graph. The price is the ability to distinguish types from their subtypes.

Shape graphs assign types to nodes using an embedding, which is an adaptation of the standard notion of graph simulation. Consequently, we explore an approach of grouping together nodes that are comparable with the simulation relation. To that effect, for a given simple graph \(G\) we construct the graph \(G^*\) by changing the arity of every edge to \(\ast\), and then, any autoembedding of \(G^*\) is a simulation relation on \(G\) and vice versa. Hence, we use the maximal autoembedding \(\ll_{G^*}\) to assess the relative amount of schema information that a node supplies: if \(m \ll_{G^*} n\), we consider \(m\) to provide no more schema information than \(n\) does. We introduce useful short-hands. We say that \(n\) sim-dominates \(m\) if \(m \ll_{G^*} n\) and \(m \neq n\). Also, \(n\) properly sim-dominates \(m\) is \(n\) sim-dominates \(m\) and \(n \not\ll_{G^*} m\). We say that \(n\) and \(m\) are sim-equivalent if \(m \ll_{G^*} n\) and \(n \ll_{G^*} m\). Finally, a node is sim-maximal if it is sim-dominated only by sim-equivalent nodes.
Using simulation to group nodes has an important drawback: the inability to distinguish types from their subtypes. Indeed, if $t_1 \subseteq t_2$, then any node that has type $t_1$ is simulated by nodes of type $t_2$ that characterize the type $t_2$. We return to this problem in Section 6 where we show how to handle inference of restricted cases of subtypes. We point out that permitting arbitrary subtypes allows to construct ascending chains, which can be then easily used in limit point arguments, as evidenced for instance in Figure 5b. This allows us to conclude that inferring subtypes is inherently problematic. Furthermore, we show that even forbidding subtypes does not suffice to guarantee learnability.

**Example 11.** To keep the example simple, we limit the intervals used in shape graphs to $\{1, ?, \ast\}$. In figures we use a dotted green arrow $n \leftarrow m$ to indicate that $n$ sim-dominates $m$.

\[
\begin{align*}
G_2 & \quad n_0 \xrightarrow{a} n_1 \xrightarrow{a} n_2 \xrightarrow{a} n_3 \xrightarrow{a} n_4 \xrightarrow{b} \text{str} \\
G_2 \div P_0 & \quad \{n_0\} \xrightarrow{a} \{n_1\} \xrightarrow{a} \{n_2\} \xrightarrow{b} \text{str} \\
H_2 & = G_2 \div P_{\text{max}} \quad \{n_0, n_3\} \xrightarrow{a} \{n_1, n_3\} \xrightarrow{a} \{n_0, n_2, n_4\} \xrightarrow{b} \text{str}
\end{align*}
\]

**Figure 6** Example of inference using graph simulation. $G_2 \div P_0$ does not recognize $G_2$.

Take the graph $G_2$ in Figure 6, identify its sim-maximal nodes $\text{Max} = \{n_0, n_1, n_2, \text{str}\}$, and consider the node grouping $P_0 = \{\{n_0\}, \{n_1\}, \{n_2\}, \{\text{str}\}\}$. This grouping is too sparse: the aggregate $G_2 \div P_0$ does not recognize $G_2$. Consider then the grouping where every sim-maximal node is accompanied by all sim-dominated nodes: $P_{\text{max}} = \{\{n_0\}, \{n_1, n_3\}, \{n_2, n_4\}, \{\text{str}\}\}$. While $G_2 \div P_{\text{max}}$ recognizes $G_2$, it is not singular because $\{n_0\} \lessdot_{G_2 \div P_{\text{max}}} \{n_2, n_4\}$. Although originally $n_0$ was sim-maximal, afterwards it became subsumed by the group $\{n_2, n_4\}$. It is then absorbed by $\{n_2, n_4\}$ and we get the node grouping $P_{\text{max}}^+ = \{\{n_1, n_3\}, \{n_0, n_2, n_4\}, \{\text{str}\}\}$, which yields the singular shape graph $H_2$ that recognizes $G_2$.

The node grouping $P_{\text{max}}^+$ is not a good starting point for an inference algorithm because it might be overly eager as we illustrate next. Consider the graph $G_3$ in Figure 7. The node $m_2$ is sim-dominated by $m_1$ and $m_3$. The grouping $P_{\text{max}} = \{\{m_0\}, \{m_1, m_2\}, \{m_2, m_3, m_4\}, \{\text{str}\}\}$ forces $\ast$ on the edge from $\{m_0\}$ to $\{m_1, m_2\}$. It is, however, not necessary to add $m_2$ to any group: $P_1 = \{\{m_0\}, \{m_1\}, \{m_3, m_4\}, \{\text{str}\}\}$ already yields $H_3$ that is singular, recognizes $G_3$, and is a tighter fit than $G_3 \div P_{\text{max}}$. 

\[\vdash\]
For inferring singular shape graphs we propose the algorithm \texttt{sim-learner}. It is parameterized by a proper set of intervals \( M \) which restricts the intervals used in the output singular shape graph.

\textbf{Algorithm 1} Inference algorithm for \texttt{SinShEx}_0(M).

\begin{algorithm}
\textbf{algorithm} \texttt{sim-learner}_M(G)
\textbf{input:} \( G = (N_G, E_G) \) a simple graph\n\textbf{param:} \( M \) a proper set of intervals
\textbf{output:} inferred shape graph in \texttt{SinShEx}_0(M)

\textbf{begin}\n1: \( \text{Max} := \{ n \in N_G \mid \forall m \in N_G. \ n \ll_{G^{*}} m \Rightarrow m \ll_{G^{*}} n \} \)
2: \( P := \emptyset \)
3: \textbf{for} \( n \in \text{Max} \) \textbf{do}\n4: \( P[n] := \{ m \in \text{Max} \mid m \ll_{G^{*}} n \} \)
5: \textbf{while} \( G \not\approx G \div_M P \) \textbf{do}\n6: \quad \text{Choose} \( n \in \text{Max} \) and \( m \in N_G \setminus P[n] \) based on priority rules \( \Phi \), such that \( m \ll_{G^{*}} n \)
7: \quad \text{add} \( m \) to \( P[n] \)
8: \textbf{while} \( \exists S, T \in P. \ S \neq T \land S \ll_{(G \div_M P)^{*}} T \) \textbf{do}\n9: \quad \text{merge} \( S \) and \( T \) in \( P \)
10: \textbf{return} \( G \div_M P \)
\textbf{end}\n\end{algorithm}

The algorithm constructs a node grouping \( P \) of the input graph \( G \) and it constructs only groupings where every sim-maximal node \( n \) belongs to precisely one node group, which can be accessed with a lookup operation \( P[n] \). The algorithm begins by identifying all sim-maximal nodes (line 1) and grouping together all sim-equivalent nodes (lines 2-4). Next, as long as the aggregate \( G \div_M P \) does not recognize the input graph, the algorithm iteratively identifies a node \( m \) that is absorbed by a node group \( P[n] \) such that \( m \not\ll P[n] \) (lines 5-7). Among all eligible pairs of \( m \) and \( P[n] \) we choose one according to the following order \( \Phi \):

1. **siblings first** any \( m \) that has a sibling \( n \in \text{Max} \) such that \( P[n] \) contains all siblings of \( m \) that sim-dominate it.
2. **single choice** any \( m \) and \( n \in \text{Max} \) such that all nodes that sim-dominate \( m \) are in \( P[n] \).
3. **all else** any \( n \in \text{Max} \) and \( m \in N_G \) such that \( m \in N_G \setminus P[n] \) and \( n \) sim-dominates \( m \).

Finally, the algorithm merges any node groups that prevent the shape graph \( G \div_M P \) from being singular (lines 8-10). We point out that the preference order \( \Phi \) may indicate multiple equally preferred pairs and the algorithm may choose an arbitrary one: regardless of the choice the output always recognizes the input graph and when the input contains a characteristic graph, the algorithm will always converge on the goal schema. \( \Phi \) prevents the algorithm \texttt{sim-learner} from performing unnecessary generalization: when executed on the
graph $G_3$ in Figure 7 it returns the schema $H_3$. Furthermore, when run on $G_2$ in Figure 6 the algorithm returns the schema $H_2$, which illustrates that it is capable of inferring recursive types even from acyclic input graphs.

In general the algorithm is sound for SinShEx, but here we only claim that sim-learner $1, \ast$ is sound and complete for the shape graphs using $1$ and $\ast$.

**Theorem 12.** SinShEx$_0(1, \ast)$ is learnable in polynomial time and data from simple graphs.

In Section 6.2 we take advantage of $\Phi$ to show completeness for a larger class but it remains an open question if the full class of singular shape graphs is learnable in polynomial time and data (but the existing evidence suggests it is unlikely [49]). The full class is however learnable with an algorithm that exhaustively explores the exponential space of possible singular shape graphs and uses an exponentially-large characteristic graph to help navigate this space.

**Theorem 13.** SinShEx$_0$ is learnable from simple graphs.

### 5 Deterministic shape graphs

In this section we investigate the potential ramifications of attempting to improve the process of identifying the set of types by exploiting the context in which nodes are used in the graph. More precisely, two nodes $n$ and $n'$ are used in the same context of node $m$ and label $a$ if there are two edges $(m, a, n)$ and $(m, a, n')$, and in that case the nodes $n$ and $n'$ should have the same type. Such premise leads us to the class of deterministic shape expressions schemas that allow at most one type to be used in a given context.

**Definition 14.** A type $t$ of a shape graph $H$ is deterministic iff for every label $a \in \Sigma$ the type $t$ has at most one outgoing edge with label $a$. A deterministic shape graph has only deterministic types and by DetShEx$_0$ we denote the set of all deterministic shape graphs.

Interestingly, the powerset method for determinization of finite automata can be adopted for inference of DetShEx$_0$. We illustrate this algorithm in Figure 8 but refrain from formally introducing it because we do not wish to promote it in this paper.

![Figure 8 Determinization of a graph.](image)

Later in this section we provide its alternative semantic characterization that will allow us to identify the pitfalls of an indiscriminate use of context information for shape graph inference.

Some caveats follow. Firstly, not every simple graph can be recognized by a deterministic shape graph. For instance, the graph $G_1$ in Figure 2a satisfies no deterministic schema because $G_1$ has a node with two outgoing $b$-edges, one of which leads to a literal node and the other to a non-literal node. Consequently, we limit the inputs to data-free graphs, simple graphs with no literal nodes. Secondly, the limit point in Figure 5b shows that DetShEx$_0$ is not learnable, and thus, we recall from [49] a subclass DetShEx$_0$ that we show learnable with the determinization algorithm. Formally, a reference to a type $t \in N_H$ is any edge $e \in E_H$
that leads to \( t \) i.e. \( \text{target}_H(e) = t \). A reference \( e \) is *-closed if \( \text{arity}_H(e) = * \) or all references to \( \text{source}_H(e) \) are *-closed. \( \text{DetShEx}_0 \) is the class of deterministic shape graphs that do not use + and any type using ? is referenced at least once and all references to it are *-closed.

**Theorem 15.** \( \text{DetShEx}_0 \) is not learnable from data-free graphs. \( \text{DetShEx}_0 \) is learnable from data-free graphs in polynomial data.

It can be proved (see appendix) that the determinization algorithm outputs the containment-minimal shape graph in \( \text{DetShEx}_0 \) that satisfies the input data-free graph \( G \). This makes the determinization algorithm an embodiment of the minicon principle which postulates for constructing the minimal consistent schema. This principle has a negative ramifications.

**Proposition 16.** For any \( n \) there exists a graph \( G_n \) of size quadratic in \( n \) such that the containment-minimal shape graph in \( \text{DetShEx}_0 \) that validates \( G \) is of size exponential in \( n \).

We conclude that an indiscriminate use of the context information has negative impact on inference of shape graphs: the obtained schemas risk being too verbose and overfitted. Also, nontrivial classes of shape graphs are too rich to benefit from blindly following the minicon principle.

### 6 Hybrid approach

In this section we pursue a hybrid approach that extends the simulation-based approach with a limited amount of context information to enable inference of shape graphs with subtypes. We then identify the subclass that our hybrid algorithm is able to infer among shape graphs using arbitrary basic intervals.

#### 6.1 Context-singular shape graphs

We extend the simulation-based approach with context information limited to the labels of incoming edges. This allows to infer subtypes as long as they are used in different contexts. For instance, this extension can infer the schema in Figure 1, where Employee \( \subseteq \) User but Employee is used in the context of verifiedBy and User is used only in the context submittedBy.

Here, the contexts in which a type \( t \) is used in a shape graph \( H \) are described by the labels of incoming edges: \( \text{ctx-lab}_H(t) = \{ a \in \Sigma \mid \exists s \in N_H. (s,a,t) \in E_H \} \). Singularity of shape graph requires that no type is sim-dominated by another type. We relax this by allowing a sim-dominated type as long as it is used in at least one context where none of its sim-dominating types are.

**Definition 17.** Given a shape graph \( H \), a type \( t \) of \( H \) is context-singular if and only if \( \text{ctx-lab}_H(t) \not\subseteq \bigcup \{ \text{ctx-lab}_H(t') \mid t' \in N_H, t \ll_H t', t \neq t' \} \). \( H \) is context-singular if it has only context-singular types. \( \text{CtxSinShEx}_0 \) is the set of all context-singular shape graphs.

To adapt \( \text{sim-learner} \) to inference of context-singular shape graphs we introduce an uncoupling operation that for a given graph creates a distinct clone of a node for every of its incoming edges, so that all incoming edges of a node have the same label. Formally, the
uncoupling of a graph \( G \) is \( \bar{G} = (N_G, E_G, \text{arity}_G) \), where

\[
N_G = \bigcup \{ \text{uncouple}(n) \mid n \in N_G \},
\]

\[
\text{uncouple}(n) = \begin{cases} 
\{n\} & \text{if } n \in \mathcal{L}, \\
\{(\bot, n)\} & \text{if } \text{ctx-lab}_{G}(n) = \emptyset, \\
\{(a, n) \mid a \in \text{ctx-lab}_{G}(n)\} & \text{otherwise}, 
\end{cases}
\]

\[
E_G = \{((\eta, a, \text{try}(a, m)) \mid (a, m) \in E_G, \eta \in \text{uncouple}(n)\}
\]

\[
\text{try}(a, m) = \begin{cases} 
m & \text{if } m \in \mathcal{L}, \\
(a, m) & \text{otherwise}, 
\end{cases}
\]

\[
\text{arity}_G(\eta, a, \text{try}(a, m)) = \text{arity}_G(n, a, m) \text{ for } (n, a, m) \in E_G \text{ and } \eta \in \text{uncouple}(n).
\]

When each node has incoming edges with the same label, using the context information is simpler. For instance, a type \( t \) of an uncoupled shape graph \( H \) is context-singular iff there is no type \( t' \) such that \( t \ll H \), \( t' \) and \( \text{ctx-lab}_{H}(t) \subseteq \text{ctx-lab}_{H}(t') \). Consequently, for uncoupled graphs we define a contextualized variant of the autoembedding relation: \( n \ll_{\text{ctx}} m \) iff \( n \ll_K m \) and \( \text{ctx-lab}_K(n) \subseteq \text{ctx-lab}_K(m) \). We adapt terminology and say that \( n \ll_{\text{ctx}} m \) iff \( m \ll_{\text{ctx}} n \) and \( n \neq m \). Also, \( n \) is ctx-maximal if any node that ctx-dominates \( n \) is also ctx-dominated by \( n \).

Now, the inference algorithm hybrid-learner is a modification of sim-learner where we replace every occurrence of \( G \) by \( \bar{G} \) and every occurrence of \( \ll \) by \( \ll_{\text{ctx}} \). We illustrate hybrid-learner on an example.

**Example 18.** We run hybrid-learner in Figure 9 on the graph \( G_0 \) introduced in Figure 2. In the uncoupling of \( G_0 \) we indicate with \( m \rightarrow n \) that \( m \ll_{\text{ctx}}^{\text{sim}} n \).

- Figure 9 Execution of hybrid-learner on \( G_0 \). \( m \rightarrow n \) means that \( n \) sim-dominates \( m \).

In this example we use the complete set of basic intervals \( \mathcal{M}_0 = \{1, ?, +, \ast\} \). The algorithm begins with the node grouping \( P_0 = \{(r, b_1), (v, e_1), (s, e_1), (s, e_2)\} \), \( \{\text{str}\} \) obtained from the ctx-maximal nodes in \( \bar{G}_0 \). Because \( \bar{G}_0 \) is not recognized \( \bar{G}_0 \), the algorithm uses \( \text{Phi} \) to identify nodes that can be absorbed: \( (r, b_1) \) can be absorbed in \( P_0 \) only. After this operation the node grouping \( P_1 \) still does not yield a shape graph that recognizes \( \bar{G}_0 \). In the next step, \( (r, b_2) \) being ctx-dominated by its sibling \( (r, b_1) \) is absorbed in \( P_1[(r, b_1)] \). The resulting node grouping \( P_2 \) again fails to recognize \( \bar{G}_0 \). Finally, the node \( (s, e_1) \) is absorbed in \( P_2[(s, e_1)] \) and the resulting node grouping \( P_3 \) yields \( \bar{G}_0 \equiv H_0 \) which recognizes \( \bar{G}_0 \).

The algorithm hybrid-learner is a sound inference algorithm for CtxSinShEx and analogously to sim-learner it is also complete for CtxSinShEx(1, *). The full class of context-singular shape graphs is learnable with an exhaustive exploration of the exponential search space with exponentially large characteristic graphs.
Inference of Shape Graphs for Graph Databases

14:14

Theorem 19. CtxSinShEx_0(1, ∗) is learnable in polynomial time and data from simple graphs. CtxSinShEx_0 is learnable from simple graphs.

6.2 Absorption-protected shape graphs

In this section we characterize the subclass of CtxSinShEx_0, using all basic intervals \(M_0 = \{1, ?, +, ∗\}\), for which hybrid-learner is sound and complete. To do that we first identify the information that needs to be contained in the input graph for hybrid-learner to infer the right arity on the edge of a the goal shape graph.

Let us fix a goal shape graph \(H\) and an edge \((t, a, s) \in E_H\). For hybrid-learner to output \(H\), it must be able to construct from a characteristic graph \(G\) two groups of nodes \(P_t\) and \(P_s\) that characterize the interval \(arity_H(t, a, s)\). In particular, they must contain sufficient information to indicate whether the interval is unbounded (\(+\) and \(∗\)) and whether it is nullable (\(?\) and \(∗\)).

Characterizing an unbounded interval is relatively simple. It suffices to introduce in \(G\) a ctx-maximal node \(n_t\) of type \(t\) that has at least two outgoing \(-\)edges to nodes \(n_s\) and \(m_s\) of type \(s\), one of which, say \(n_s\), should also be ctx-maximal. Naturally, the nodes \(n_t\) and \(n_s\) will be put by hybrid-learner in the groups \(P_t\) and \(P_s\) respectively. Because of the first rule of the preference order \(Φ\), the node \(m_s\) will also be added to \(P_m\).

Characterizing a nullable interval is more challenging since it requires making sure that \(P_t\) and \(P_s\) absorb information that is more volatile. Namely, we need to make sure that \(G\) has a ctx-maximal node \(n_t\) and a node \(m_t\) of type \(t\) without any \(-\)edge to a node of type \(s\). Since \(n_t\) is ctx-maximal, it will be absorbed in \(P_t\) but since \(m_t\) is not a ctx-maximal node, \(G\) must protect it from being absorbed by any other group other than \(P_t\). The node \(m_t\) can be absorbed into a group \(P_{-}\), representing a type \(s\), if \(s\) dominates a version \(η\) of the type \(t\) from which we remove the edge \((t, a, s)\). Naturally, \(m_t\) is protected if no such node exists, and then the second rule of \(Φ\) ensures that \(m_t\) ends up in \(P_t\). Otherwise, the type \(t\) allows protection from an absorption if \(t\) has an incoming edge with unbounded arity and none of the siblings of \(t\) dominate \(η\). Then \(m_t\) also ends up in \(P_t\) because of the first rule of \(Φ\) for same reasons as explained for unbounded intervals.

To characterize the classes of schemas that allow protection from absorption we need to introduce a construction of the type \(η\). Formally, let \(H\) be a shape graph \(H, η ∈ N_L \setminus NH\) a fresh node identifier, and \(e ∈ E_H\) an edge from \(t\) to \(s\) with unbounded arity. The absorption-protection test graph \(APT^*(H, e, η)\) is a shape graph \(H'\) defined as:

\[
N_{H'} = N_H ∪ \{η\}, \quad E_{H'} = E_H ∪ \{(η, a, s')|(t, a, s') ∈ out_H(t) \setminus \{e\}\}, \quad \text{arity}_{H'}(f) = ∗ \text{ for } f ∈ E_{H'}.
\]

Essentially, we add the type \(η\), for every outgoing edge \(e'\) from \(t\) other than \(e\), we add an outgoing edge with the same label as \(e'\) from \(η\) to the same node as the endpoint of \(e'\), and we set arities of all edges to \(∗\). Next, we formally characterize the class of shape graphs that hybrid-learner is capable of identifying (we use sim-domination because \(η\) in \(APT^*(H, e, η)\) does not have any incoming edges).

Definition 20. A type \(t\) of a shape graph \(H\) is absorption-protected iff for every edge \(e ∈ out_H(t)\) with nullable arity one of the following conditions holds:

1. \(t\) is the only node that sim-dominates \(η\) in \(APT^*(H, e, η)\) or,
2. \(t\) has an incoming edge with unbounded arity and \(t\) is the only node among its siblings that sim-dominates \(η\) in \(APT^*(H, e, η)\).

An absorption-protected shape graph is a context-singular shape graph that has only absorption-protected types. By AbsShEx_0 we denote the set of all absorption-protected shape graphs.
We illustrate absorption protection in the following example.

Example 21. In Figure 10, $H_1$ is not absorption-protected because of the edge $e = (t_1, a?, str)$. The copy $\eta$ of $t_1$ without the edge $e$ is sim-dominated by the type $t_2$ of $H_1$. $H_2$ is absorption-protected. Even if for the type $s_1$ we create a copy $\eta$ without the edge $(s_1, a?, str)$ and find it is sim-dominated by $s_2$, the type $s_1$ has an incoming unbounded $r$-edge and the type $s_2$ is not a sibling of $s_1$.

$$H_1 \xrightarrow{\text{APT}^+}(H_1, e, \eta) \text{ where } e = (t_1, a?, str) \quad H_2$$

Figure 10 Two shape graphs, $H_1$ is not absorption-protected because $t_1$ is not, but $H_2$ is.

We point out that the shape graph in Figure 1 is absorption protected, which explains why in Example 18 the algorithm correctly infers the schema $H_0$. We finish by formally stating the learnability result.

Theorem 22. AbsShEx$_0$ is learnable in polynomial time and data from simple graphs.

Schemas of existing graph databases

In this section we present an analysis (Table 1) of a number of graph databases in order to find out to what extent their schemas can be captured with shape graphs and the subclasses that we propose, and consequently, to assess the usefulness of the proposed inference algorithms.

We have analyzed: the Berlin SPARQL Benchmark [10], the Tpc-h benchmark converted to RDF with Direct Mapping [51, 54], the Nobel Prize database [4], an RDF serialization of the DBLP database (retrieved from https://www.rdfhdt.org/datasets/ on 14.01.2021), the Springer Nature knowledge graph SciGraph [32], the Covid19 knowledge graph [43], and SHACL constraints in the version 4 of Yago knowledge base [50]. For each database we have defined appropriate schemas using ShEx, which is a fully fledged schema language and properly contains ShEx$_0$ (cf. [47]). Complete schemas and more details can be found in the appendix.

Table 1 Analysis of schemas for a number of graph databases.

<table>
<thead>
<tr>
<th></th>
<th>Base types total</th>
<th>Det</th>
<th>$\sigma$-ind</th>
<th>Sin</th>
<th>CtxSin</th>
<th>Abs</th>
<th>Union types total</th>
<th>Clauses total</th>
<th>union</th>
<th>disj.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLP</td>
<td>11</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>7</td>
<td>2 (3)</td>
<td>135</td>
<td>2 (9)</td>
<td>1</td>
</tr>
<tr>
<td>BERLIN</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>0 (0)</td>
<td>47</td>
<td>0 (0)</td>
<td>0</td>
</tr>
<tr>
<td>TPC-H (DM)</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>0 (0)</td>
<td>67</td>
<td>0 (0)</td>
<td>0</td>
</tr>
<tr>
<td>NOBEL</td>
<td>11</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>10</td>
<td>8</td>
<td>1 (0)</td>
<td>66</td>
<td>2 (0)</td>
<td>1</td>
</tr>
<tr>
<td>SCIGRAPH</td>
<td>18</td>
<td>10</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>16</td>
<td>1 (2)</td>
<td>162</td>
<td>1 (7)</td>
<td>1</td>
</tr>
<tr>
<td>COVID19</td>
<td>22</td>
<td>11</td>
<td>11</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>2 (0)</td>
<td>295</td>
<td>3 (0)</td>
<td>0</td>
</tr>
<tr>
<td>YAGO</td>
<td>49</td>
<td>14</td>
<td>28</td>
<td>28</td>
<td>30</td>
<td>21</td>
<td>2 (13)</td>
<td>1174</td>
<td>4 (490)</td>
<td>0</td>
</tr>
</tbody>
</table>

We have found that the ShEx schemas for the above databases require the use of disjunction and union types, features that are absent in ShEx$_0$ [47, 55]. We illustrate them on the following example inspired by DBLP.

Book $\rightarrow$ (creator: Person+ | editor: Person+), title: Literal, ref : (Article $\cup$ Book)*.
It states that a book has either creators (authors) or editors but never both, has a single title, and references an arbitrary number of articles and books. We say that this base type definition has 3 clauses, one uses disjunction, and one uses a union type. Union types are trivial when used only under $\ast$ since $\alpha : (t_1 \cup t_2)\ast$ is equivalent to $\alpha : t_1\ast$, $\alpha : t_2\ast$. For every schema we have measured the quantities of the following objects: nontrivial (and trivial) union types, clauses, clauses with disjunction, clauses with nontrivial (and trivial) use of a union type.

To help assessing the usefulness of the proposed inference algorithms, for every $\text{ShEx}$ schema we have constructed its straightforward $\text{ShEx}_0$-approximation as illustrated on the example for the above type definition of $\text{Book}$

$$\text{Book} \rightarrow \text{creator} : \text{Person}\ast, \text{editor} : \text{Person}\ast, \text{title} : \text{Literal}, \text{ref} : \text{Article}\ast, \text{ref} : \text{Book}\ast.$$  

Then in each $\text{ShEx}_0$ we have measured the number of all (base) types, those that are singular ($\text{Sin}$), context-singular ($\text{CtxSin}$), and absorption-protected ($\text{Abs}$). Additionally, we have counted the number of types that are signature-independent ($\sigma$-ind): types whose set of labels of outgoing edges is incomparable with others.

We first analyze our findings for the graph databases and knowledge graphs (first 6 rows). We observe that the numbers of nontrivial union types and disjunctions are relatively small. Also, deterministic types are relatively low in some databases because of use of union types: even trivial use of union types in a type definition precludes determinism. We point to the differences in signature-independent and singular types, on average 69% vs. 79% resp., which indicates that using simulation to identify types has advantages over an approach that reduces a type to the labels of its outgoing types. While the numbers of singular types are relatively large, 79% on average, the numbers of context-singular types, 94% on average, are even more impressive, and the significant improvement validates our approach of incorporating the context information in the inference process.

For the knowledge base YAGO the results indicate that our algorithms would offer little improvement over simpler approaches ($\sigma$-ind) that attempt to identify types by using the labels of their outgoing edges. We attribute this to the use of rich hierarchies of classes which introduce many subtype relationships between types. We point out, however, that those class hierarchies are defined with ontologies that specify rules that allow to type the graph. Our methods do not take advantage of this typing information, and therefore, are not geared towards inference for knowledge bases. Alternative tools for inference from typed graphs should be used (see our preliminary results on the topic in [30]).

We conclude that inference algorithms for the proposed classes $\text{CtxSinShEx}_0$ and $\text{AbsShEx}_0$ are an attractive choice for the first draft of the schema for (untyped) graph databases and knowledge graphs to be later on improved by a data specialist, especially when the type hierarchy is relatively flat.

8 Related work

Several approaches have been investigated in the late 90’s to summarize graph data, defining schema models, and algorithms to infer schemas from graph data, then viewed as semi-structured data [15, 24, 41, 40]. A large part of the literature focuses on graph summarization [14], where the purpose is to obtain a concise representation of an input graph that can be used to formulate queries against the data. Our approach takes a different perspective of the same problem, where the input data is considered as evidence from which we must infer an unknown target schema. Using the grammatical inference framework enables us to offer strong guarantees, and lay the ground for a principled fundamental approach. To
the best of our knowledge this is a first, and this complements the traditional approach in
the literature that consists in evaluating algorithms empirically on real-life graph data to
measure the tradeoff between compactness and the accuracy of the description.

A Dataguide [29] of a graph represents as a DFA all the paths that can be navigated
from a root node in the graph. Dataguides are thus similar in spirit to our determinization
procedure, but do not represent the arity of paths and assume that specific entry points have
been specified; the root nodes. This approach, like determinization, can identify subtypes,
and cycles (provided the cycles are presented in the input data). Several approaches have been
proposed to infer schemas for tree-structured data [25, 39, 21, 8, 9] or JSON data [5, 6]. The
schemas inferred include DTDs, tree automata, and XML Schema, and thus are expressive
enough to represent the arity of edges. However, these approaches focus on tree-structure and
do not consider cycles (as expressed via mechanisms such as ID/REF). The tree structure
means that the schema can be inferred recursively, and thus the task is essentially about
inferring efficiently a regular expression representing for the children of each tree node.

More recently, the inference of schemas for graphs has focused on RDF and property
graphs [34]. Several applications [23, 56, 33, 52] rely on graph (bi)simulation to build a
quotient graph w.r.t. an equivalence relation. The quotient graph merges all nodes that
belong to the same equivalence class. Those proposals generally do not assign arities to the
edges with a few exceptions such as [53]. Variants [16, 44, 52] consider both forward and
backward bisimilarity, which means that types can be defined based on both the incoming and
outgoing edges. Full bisimulation is rather rare in heterogenous graphs and even limiting it
to the 1-hop neighbourhood does not help [26, 20, 18, 27], so the authors introduce weak and
strong equivalence relations that relax rather aggressively the bisimulation relation through
transitive co-occurrence of edge labels, to achieve more compact graph summaries. Type
hierarchies essentially collapse in summaries built on weak or strong equivalence whenever
they would collapse for hybrid-learner. Actually, weak equivalence systematically collapses
hierarchies (except for nodes without outgoing edges). Even these summaries may become
large for big, encyclopedic graphs, such as DBPedia or YAGO. One solution they suggest
is to use methods based on mining frequent patterns [19]. An alternative approach to such
problematic graphs is to use semi automatic methods where the construction of schema is
interactively guided by domain experts [12].

Other ideas include clustering nodes according to thresholds on the number of shared
edge labels [53]. In [3] integer linear programming is used to optimize the tradeoff between
the number of partitions and how well each partition fits to the schema, however, the fitness
is defined using outbound neighborhood, and not the context. Finally, some approaches
complement the structural information with types that may be already present in graphs [16,
27] or, in the case of property graphs, by inferring the typing from node labels [37].

Among results on grammatical inference, [31] shows that a family of well-behaved inference
algorithms can be interpreted as constructing a least upper bound for the input in the lattice
of languages from the class, where the order is defined as the containment of languages. Our
inference algorithm follows a similar approach. It does not compute a least upper bound
among the whole class of graph schemas, but rather inside the lattice of the graph aggregates
of the input graph (see Section 3.3).

9 Conclusions and future work

We have presented a principled approach to the problem of inference of shape graphs, which
are a subclass of shape expression schemas for graph databases. We have employed the
framework of grammatical inference to identify key challenges: identifying arities on edges
and identifying recursive types. To confront these challenges we have explored an approach
of inferring shape graphs using simulation relation on the nodes of the input graph. This
Inference of Shape Graphs for Graph Databases

This approach addresses well the identified challenges at the price of identifying subtypes, which we argue to be a fundamental limitation. To overcome it we extend the simulation based approach with a modest amount of context information. The resulting inference algorithm allows to infer a rich class of schemas, which we argue to be of practical importance with an empirical study of real-life schemas of graph databases.

Future research directions include exploring techniques for inference of disjunctions [11] and adapting the proposed methods to the emerging standard for Property Graph Schemas. We also plan to pursue the question of inference of shape graphs from typed graphs, our preliminary results are very promising [30].

References


22 R. Ciucanu and S. Staworko. Learning schemas for unordered XML. In International Symposium on Database Programming Languages (DBPL), 2013.


14:20  Inference of Shape Graphs for Graph Databases

51  TPC. TPC benchmarks. URL: http://www.tpc.org/.
54  W3C. A direct mapping of relational data to RDF, 2012. URL: http://www.w3.org/TR/rdb-direct-mapping/.
55  W3C. Shape expressions schemas, 2013. URL: http://www.w3.org/2013/ShEx/Primer.