Containment of Shape Expression Schemas for RDF

Sławek Staworko
CRIStAL, INRIA LINKS, CNRS
University of Lille
France
slawomir.staworko@inria.fr

Piotr Wieczorek
Institute of Computer Science
University of Wrocław
Poland
piotr.wieczorek@cs.uni.wroc.pl

ABSTRACT

We study the problem of containment of *shape expression schemas* (ShEx) for RDF graphs. We identify a subclass of ShEx that has a natural graphical representation in the form of *shape graphs* and whose semantics is captured with a tractable notion of *embedding* of an RDF graph in a shape graph. When applied to pairs of shape graphs, an embedding is a sufficient condition for containment, and for a practical subclass of *deterministic shape graphs*, it is also a necessary one, thus yielding a subclass with tractable containment. Containment for general shape graphs is EXP-complete. Finally, we show that containment for arbitrary ShEx is decidable.

CCS CONCEPTS

• **Information systems → Graph-based database models**; **Resource Description Framework (RDF)**; • **Theory of computation → Database theory**; Database interoperability.

KEYWORDS

RDF, Schema, ShEx, containment, counter-example

ACM Reference Format:
Sławek Staworko and Piotr Wieczorek. 2019. Containment of Shape Expression Schemas for RDF. In *38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3294052.3319687

1 INTRODUCTION

Although RDF has been originally introduced schema-free, it has since become a standalone database format and the need for a schema language has been identified, with the emergence of new RDF applications previously reserved to relational and semi-structured databases [2, 37]. Recently introduced by W3C, and under continuous development, *shape expression schema* (ShEx) is a formalism for defining valid RDF graphs [21–23, 38]. ShEx allows to define a set of types, each type defined with a rule describing the admissible types of the outbound neighborhood of a node. Inspired by versatility of schema languages for XML [4, 19], the rules of ShEx are based on regular expressions.

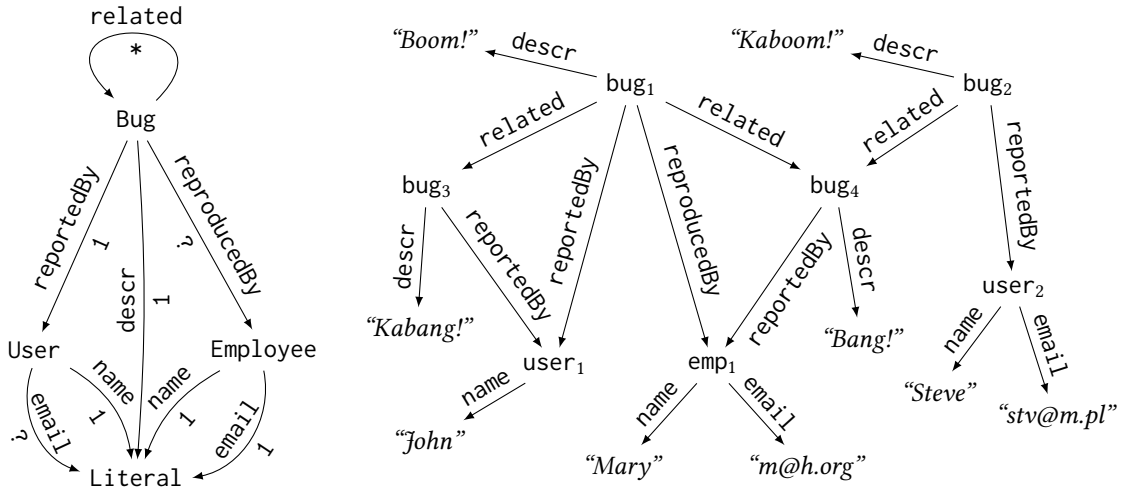
An example of shape expression schema for RDF graphs storing bug reports is presented in Figure 1. The schema requires a bug report to have a description and a user who reported it. Optionally, a bug report may have an employee who successfully reproduced the bug. Also, a bug report can have a number 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.

In this paper, we investigate the classical problem of *containment*: given two schemas S and S', is the set of instances satisfying S contained in the set of instances satisfying S'? This problem has applications to a vast number of problems that perform non-trivial reasoning tasks such as data exchange, query optimization, and inference [1, 3, 9, 10, 14, 16, 40]. The task at hand is difficult for a number of reasons.

Because the neighborhood of a node in an RDF graph is unordered, regular expressions define bag languages, also known as *commutative languages* [18], where the relative order among symbols is irrelevant. This lack of order gives raise to a significant degree of nondeterminism when working with *regular bag expressions* (RBE). For instance, membership for RBE i.e., deciding whether a bag of symbols belongs to the language defined by an RBE, is NP-complete [20]. Similarly, validation for ShEx i.e., deciding whether a RDF graph satisfies a ShEx, is NP-complete too [34]. The need for nondeterminism can be limited by disallowing disjunction and permitting the Kleene closure on atomic symbols only. This yields the class RBE_0 with tractable membership and tractable validation for the corresponding class of shape expression schemas ShEx_0. Similarly, single-occurrence regular bag expressions (SORBE) have tractable membership and give rise to *deterministic* shape expression schemas (DetShEx), where the same symbol can be used only once. Their validation is also tractable [34]. Both restrictions offer enough room to

PODS'19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-6227-6/19/06…$15.00
https://doi.org/10.1145/3294052.3319687



$\texttt{Bug} \longrightarrow \texttt{descr} :: \texttt{Literal},\ \texttt{reportedBy} :: \texttt{User},\ \texttt{reproducedBy} :: \texttt{Employee}^{?},\ \texttt{related} :: \texttt{Bug}^{*}$

$\texttt{User} \longrightarrow \texttt{name} :: \texttt{Literal},\ \texttt{email} :: \texttt{Literal}^{?}$

$\texttt{Employee} \longrightarrow \texttt{name} :: \texttt{Literal},\ \texttt{email} :: \texttt{Literal}$

Figure 1: An RDF graph with bug reports (top right) together with a shape expression schema (bottom) and the corresponding shape graph (top left).

accommodate practical uses, and in particular, the schema in Figure 1 belongs to them both.

Since ShEx is a schema language based on types, comparing two schemas requires the ability to compare types, and consequently, testing $S \subseteq S'$ revolves around questions whether a type t of S is *covered* by the union of types $s_1, \ldots, s_k$ of S'. Indeed, suppose that in the schema in Figure 1 we replace the type User with two types depending on whether or not the user has an email address:

$$\texttt{User}_1 \longrightarrow \texttt{name} :: \texttt{Literal}$$
$$\texttt{User}_2 \longrightarrow \texttt{name} :: \texttt{Literal},\ \texttt{email} :: \texttt{Literal}$$

and adapt the rest of the schema by replacing Bug with

$$\begin{aligned}\texttt{Bug}_1 \longrightarrow\ & \texttt{descr} :: \texttt{Literal},\ \texttt{reportedBy} :: \texttt{User}_1,\\ & \texttt{reproducedBy} :: \texttt{Employee}^{?},\\ & \texttt{related} :: \texttt{Bug}_1^{*},\ \texttt{related} :: \texttt{Bug}_2^{*}\\ \texttt{Bug}_2 \longrightarrow\ & \texttt{descr} :: \texttt{Literal},\ \texttt{reportedBy} :: \texttt{User}_2,\\ & \texttt{reproducedBy} :: \texttt{Employee}^{?},\\ & \texttt{related} :: \texttt{Bug}_1^{*},\ \texttt{related} :: \texttt{Bug}_2^{*}\end{aligned}$$

Although no longer deterministic (the symbol related is used twice in the type definitions of $\texttt{Bug}_1$ and $\texttt{Bug}_2$), the modified schema is equivalent to the original one as the type Bug is covered by the union of the types $\texttt{Bug}_1$ and $\texttt{Bug}_2$, and the type User by the union of $\texttt{User}_1$ and $\texttt{User}_2$ (the latter also being equivalent to Employee).

Naturally, the fact that a type might be covered by a union of types is a source of complexity of the containment problem, and it is an interesting question if there is a class of schemas for which comparison on pairs of types alone would suffice. To answer this question, we use *shape graphs*, which are natural graphical representation of ShEx_0 (cf. Figure 1), and propose a graph-theoretic notion of an *embedding* between pairs of shape graphs. In essence, an embedding identifies in a simulation-like manner when a type is covered by another type, and therefore, is a sufficient condition for containment. We also identify a class $\mathsf{DetShEx}_0^-$ for which embedding is a necessary condition for containment. $\mathsf{DetShEx}_0^-$ is the class of deterministic shape expressions schema using RBE_0, which furthermore forbids the use of + and, intuitively, requires every ? to be referenced through $*$. In particular, the schema in Figure 1 belongs to $\mathsf{DetShEx}_0^-$ because the type User using $\texttt{email} :: \texttt{Literal}^{?}$ is (indirectly) referenced by $\texttt{related} :: \texttt{Bug}^{*}$ in type Bug.

Because embeddings are carefully crafted to be tractable, we obtain a class with tractable containment. The additional restrictions of $\mathsf{DetShEx}_0^-$ are necessary as we show the containment problem for full $\mathsf{DetShEx}_0$ to be intractable. Interestingly, for a schema S in $\mathsf{DetShEx}_0^-$ we construct a *characterizing* graph G such that G is satisfied by any schema S' in $\mathsf{DetShEx}_0^-$ if and only if $S \subseteq S'$.

Checking the containment $S \subseteq S'$ involves implicit negation: checking whether there is no *counter-example*, an instance that satisfies S and does not satisfy S'. The implicit negation allows to encode disjunction even in classes of schemas that explicitly forbid using disjunction in type definitions, such as ShEx_0. This renders ShEx_0 very powerful and allows for pairs of schemas for which the smallest counter-example is of exponential size. Not surprisingly, we observe a significant impact on complexity: testing containment for shape graphs is EXP-complete.

The picture of containment for arbitrary shape expression schemas is far from obvious. It is known that ∃MSO on graphs is alone insufficient to capture ShEx and needs to be enriched with Presburger arithmetic [34]. However, monadic extensions of Presburger arithmetic quickly become undecidable [13, 31]. The question whether containment for ShEx is decidable at all is non-trivial but we answer it positively and give an initial characterization of its complexity: coNEXP-hard and in $\text{co2NEXP}^{\text{NP}}$. The precise complexity of containment for ShEx remains an open question.

Our study has a number of outcomes:

- a thorough characterization of complexity of containment for classes of shape expression schemas;
- a set of bounds on the size of a minimal graph that satisfies one schema and violates another;
- a tractable notion of embeddings that is a sufficient condition for containment, and a necessary one of a subclass of deterministic shape expression schemas.

Related work. There is a large body of literature dealing with the containment problem for various database schema formalisms. The expressivity of ShEx has been studied in [34]. ShEx is not comparable to neither FO logic on graphs, nor to ∃MSO on graphs. In order to capture the cardinality contraints e.g., $(a \parallel b)^\star$, expressible by RBE, ∃MSO has to be extended with Presburger Arithmetic (PA). Such extensions, as we have already mentioned, can easily get to be undecidable [13, 31]. It is, also, a classic result that MSO theory of $\mathbb{N}$ with addition is undecidable, it has been shown in [30] that even MSO theory of naturals with the double function, $\langle \mathbb{N}, +1, 2x, 0 \rangle$, is undecidable.

Schema languages for trees have been extensively studied in the context of XML. Most of the work, however, has been devoted to the case of ordered trees. Typically, schema languages for ordered trees (DTD, XML Schema) are captured by tree automata. A survey of basic decision problems for tree automata can be found in [36]. In particular, containment for nondeterministic tree automata is EXPTIME-complete [33] but becomes polynomial for deterministic tree automata. The containment of DTDs over the usual (ordered) nondeterministic regular expressions is PSPACE-complete and remains intractable even for very simple nondeterministic regular expressions [24] but drops to PTIME [25] if deterministic regular expressions are used as mandated by the XML standard [7]. We point out that in our case, however, determinism alone is not sufficient, in order to obtain tractability of the containment we need to enforce additional structural conditions on the shape graphs.

Various forms of DTDs with regular expressions interpreted under the commutative closure [28] have been studied. In the context of trees many formalisms have been introduced to express the numerical constrains on the occurrences of the different symbols among the children of some node, e.g., Presburger automata [31, 32], sheaves automata[12], TQL logic[8]. Unfortunately, the containment problem in all this settings is NP-hard [20].

The problem of containment of regular expressions with interleaving is EXPSPACE-complete [26]. A number of expressive formalisms with interleaving have been shown in [15] to have highly intractable inclusion.

Disjunctive multiplicity schemas (DMS) for unordered XML have been studied in [5]. A DMS is a formalism that defines for each label the language of allowed children labels using disjunctive multiplicity expressions (DIME) that are similar to RBE_0 but allow a limited form of disjunction. DMS assume similar form of determinism as DTDs since types of nodes are identified with their labels. Additionaly, DIMEs require each of the alphabet symbols from Σ to appear at most once. Nevertheless, the containment for DMS is in PTIME.

Organization. The paper is organized as follows. In Section 2 we present basic notions and introduce embeddings of shape graphs. In Section 3 we investigate decidability of containment for the full fragment of ShEx by bounding the size of a counter-example. In Section 4 we analyze the complexity of containment for shape graphs (ShEx_0) and show a tight exponential bound on the size of a counter-example. In Section 5 we identify a tractable subclass of deterministic shape expression schemas and investigate the complexity of constructing embeddings. We summarize our work and outline directions of further study in Section 6.

2 BASIC NOTIONS

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

Intervals. We use pairs of numbers including the infinite constant ∞ to represent intervals: the pair $[n;m]$, with $n \leq m \leq \infty$, represents the set $\{i \mid n \leq i \leq m\}$. We assume that both n and m are stored in binary. We use a number

of operators on intervals: every interval $I = [n; m]$ has its lower bound $\min(I) = n$ and its upper bound $\max(I) = m$. The point-wise addition of two intervals $A \oplus B = \{a + b \mid a \in A,\ b \in B\}$ has a natural interpretation: $[n_1; m_1] \oplus [n_2; m_2] = [n_1 + n_2; m_1 + m_2]$. Note that $[0; 0]$ is the neutral element of $\oplus$, and hence, for $k = 0$ the expression $I_1 \oplus \ldots \oplus I_k$ evaluates to $[0; 0]$. Also $[n_1; m_1] \subseteq [n_2; m_2]$ iff $n_2 \leq n_1 \leq m_1 \leq m_2$.

Four *basic intervals* are commonly employed in popular schema languages for semi-structured databases, listed here together with their shorthand notation: 1 stands for $[1; 1]$, ? for $[0; 1]$, + for $[1; \infty]$, and $*$ for $[0; \infty]$. We use I to denote the set of all intervals and M to denote the the set of basic intervals.

Bags. Let Δ be a finite set of symbols. Unordered words are represented with bags. Formally, a *bag* over Δ is a function $w : \Delta \to \mathbb{N}$ that maps a symbol to the number of its occurrences. The empty bag ε has 0 occurrences of every symbol i.e., $\varepsilon(a) = 0$ for every $a \in \Delta$. Bags are often presented using the notation $\{\!\{a, \ldots\}\!\}$ with elements possibly being repeated. For example, when $\Delta = \{a, b, c\}$, $w_0 = \{\!\{a, a, a, c, c\}\!\}$ represents the function $w_0(a) = 3$, $w_0(b) = 0$, and $w_0(c) = 2$. A *bag language* is a set of bags.

The equivalent of concatenation for unordered words is the *bag union* $w_1 \uplus w_2$ of two bags w_1 and w_2 is $[w_1 \uplus w_2](a) = w_1(a) + w_2(a)$ for all $a \in \Delta$. We also extend this operator to bag languages: $L_1 \uplus L_2 = \{w_1 \uplus w_2 \mid w_1 \in L_1, w_2 \in L_2\}$. Also, for a given bag language L, we define $L^0 = \{\varepsilon\}$ and $L^i = L \uplus L^{i-1}$ for $i > 0$.

Regular bag expressions. *Regular bag expressions* (RBE) are analogues of regular expressions for defining bag languages and use disjunction "|," unordered concatenation "||," and unordered repetition. Formally, they are defined with the following grammar:

$$E ::= \epsilon \mid a \mid (E|E) \mid (E \parallel E) \mid E^I,$$

where $a \in \Delta$ and I is an interval. Their semantics is defined as follows: $L(\epsilon) = \{\varepsilon\}$, $L(a) = \{\{\!\{a\}\!\}\}$, $L(E_1 \mid E_2) = L(E_1) \cup L(E_2)$, $L(E_1 \parallel E_2) = L(E_1) \uplus L(E_2)$, and $L(E^I) = \bigcup_{i \in I} L(E)^i$. By RBE_0 we denote the class of expressions of the form $a_1^{M_1} \parallel \ldots \parallel a_n^{M_n}$, where $a_i \in \Sigma$ and $M_i \in \mathsf{M}$ for $i \in \{1, \ldots, n\}$. We point out that occurrences of symbols need not be distinct e.g., $a \| a^+ \| b^*$ is RBE_0.

Graphs. We employ a general graph model that allows to capture RDF graphs as well as an important subclass of shape expression schemas (ShEx_0). Because shape expressions schemas do not constrain the predicates of the edges of an RDF graph, we assume a fixed set Σ of predicates names used to label edges of graphs. To represent (a subclass of) shape expression schemas as graphs, we additionally label each edge with an occurrence interval, which intuitively indicate the admissible number of edges of the given kind (cf. Definition 2.2). Also, the general graph model allows multiple edges connecting the same pair of nodes with the same predicate label, which is not allowed in standard RDF.

Definition 2.1. A *graph* is a tuple

$$G = (N_G, E_G, \mathsf{source}_G, \mathsf{target}_G, \mathsf{lab}_G, \mathsf{occur}_G),$$

where N_G is a finite set of *nodes*, E_G is a finite set of *edges*, the functions $\mathsf{source}_G : E_G \to N_G$ and $\mathsf{target}_G : E_G \to N_G$ identify resp. the origin node and end point node of an edge, $\mathsf{lab}_G : E_G \to \Sigma$ assigns a (predicate) label to an edge, and $\mathsf{occur}_G : E_G \to \mathsf{I}$ assigns an occurrence interval to an edge.

A graph is *simple* if it uses only the interval 1 and has no two edges with the same origin, the same end point, and the same label. By G_0 we denote the set of all simple graphs. A *shape graph* is a graph that uses only basic occurrence intervals (in M) and we denote the class of all shape graphs with ShEx_0. □

For the purposes of studying containment of shape expression schemas the class of simple graph captures adequately RDF graphs. Although RDF nodes are labeled with URIs, literal values, and blank identifiers, and shape expression schemas can constraint node labels, in general these constraints can, to some extent, be "simulated." For instance, if the schema imposes a type of admissible literal nodes (integer, date, etc.), literal nodes can be modified to include an outgoing edge labeled with the type name.

Shape expression schemas constrain the outbound neighborhood of a node, and for that purpose we identify the set of all outgoing edges of a node $n \in N_G$ with

$$\mathsf{out}_G(n) = \{e \in E_G \mid \mathsf{source}_G(e) = n\}.$$

Sometimes, if a node n has an outgoing edge leading to m, we shall call m a *child* of n (even if n and m are the same node). Also, we call an a-edge any edge labeled with $a \in \Sigma$, and analogously, an I-edge any edge with occurrence interval I.

Shape Expression Schemas. Again, we assume a fixed set of predicate labels Σ. Given a set of type names Γ, a *shape expression* over Γ is an RBE over $\Sigma \times \Gamma$ and in the sequel we write $(a, t) \in \Sigma \times \Gamma$ simply as $a :: t$. A *shape expression schema* (ShEx) is a pair $H = (\Gamma_H, \delta_H)$, where Γ_H is a finite set of types, and δ_H is a *type definition* function that maps elements of Γ_H to shape expressions over Γ_H. Typically, we present a ShEx H as a collection of rules of the form $t \to E$ to indicate that $\delta_H(t) = E$, where E is a shape expression. For a class of RBEs C, by $\mathsf{ShEx}(C)$ we denote the class of shape expression schemas using only shape expressions in C.

We recall the formal semantics of ShEx [34] and illustrate it on the example of a simple graph G_0 and a schema H_0 in Figure 2. A *typing* of a simple graph G w.r.t. H is a relation $T \subseteq N_G \times \Gamma_H$. For instance, a typing of G_0 w.r.t. H_0 is (for

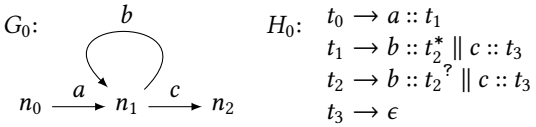


Figure 2: A simple graph G_0 and a schema H_0.

clarity, we employ functional notation)

$$T_1(n_0) = \{t_0\},\ \ T_1(n_1) = \{t_1, t_2\},\ \ T_1(n_2) = \{t_3\}.$$

Note that a node may have a number of types. The *signature* of a node $n \in N_G$ w.r.t. T is an RBE expression

$$\mathrm{sign}^T_G(n) = \Big\|_{e \in \mathrm{out}_G(n)} \big(\big|_{t \in T(\mathrm{target}_G(e))} \mathrm{lab}_G(e) :: t\big)$$

For instance, the signature of n_1 in G_0 w.r.t. T_1 is

$$\mathrm{sign}^{T_1}_{G_0}(n_1) = (b :: t_1 \mid b :: t_2) \parallel c :: t_3.$$

A node n *satisfies* a shape expression E w.r.t. a typing T iff $L(\mathrm{sign}^T_G(n)) \cap L(E) \neq \emptyset$. For instance, n_1 satisfies the type definition $\delta_{H_0}(t_2)$ of H_0 w.r.t. T_1. The typing T is *valid* iff every node satisfies the type definition of every type assigned to the node i.e., $L(\mathrm{sign}^T_G(n)) \cap L(\delta_H(t)) \neq \emptyset$ for every $(n, t) \in T$. Valid typings of G w.r.t. H form a semi-lattice, with the union as the meet operation [34]. Consequently, there exists a unique maximal typing, which we denote by $\mathrm{Typing}_{G:H}$, and in the sequel we say that a node n *has type* t if $(n, t) \in \mathrm{Typing}_{G:H}$. Now, G *satisfies* H if every node of G has at least one type i.e., $\mathrm{dom}(\mathrm{Typing}_{G:H}) = N_G$. By $L(H)$ we denote the set of all simple graphs that satisfy H.

Containment. In this paper, we investigate the containment problem for ShEx: given two shape expression schemas H and K we say that H is *contained* in K, in symbols $H \subseteq K$, if $L(H) \subseteq L(K)$. A *counter-example* for $H \subseteq K$ is any graph $G \in L(H) \setminus L(K)$.

Embeddings. We define a natural notion of embedding that allows to treat shape graphs as an alternative representation of $\mathsf{ShEx}(\mathsf{RBE}_0)$.

Definition 2.2. Given two graphs G and H, a binary relation $R \subseteq N_G \times N_H$ is a *simulation* of G in H iff for any $(n, m) \in R$ there exists a *witness* of simulation of n by m w.r.t. R, i.e., a function $\lambda_{n,m} : \mathrm{out}_G(n) \to \mathrm{out}_H(m)$ such that for every $e \in \mathrm{out}_G(n)$

1. $\mathrm{lab}_G(e) = \mathrm{lab}_H(\lambda_{n,m}(e))$,
2. $(\mathrm{target}_G(e), \mathrm{target}_H(\lambda_{n,m}(e))) \in R$,

and for every $f \in \mathrm{out}_H(m)$

3. $\bigoplus\{\mathrm{occur}_G(e) \mid e \in E_G,\ \lambda_{n,m}(e) = f\} \subseteq \mathrm{occur}_H(f)$.

An *embedding* of G in H is a simulation R of G in H such that $\mathrm{dom}(R) = N_G$, and we write $G \preccurlyeq H$ if G there is an embedding of G in H. □

Figure 3 presents an example of an embedding between the simple graph G_0 and the shape graph corresponding to the shape expression schema H_0 in Figure 2.

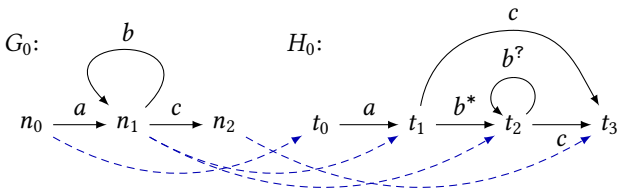


Figure 3: An embedding of G_0 in H_0.

The set of simulations of G in H is a semi-lattice (with the meet operation interpreted with the set union), and consequently, there exists exactly one *maximal simulation* of G in H. We use embeddings to treat graphs as schemas. The *language* of a graph H is the set of all simple graphs that can be embedded in H i.e., $L(H) = \{G \in \mathrm{G}_0 \mid G \preccurlyeq H\}$.

There is a natural correspondence between shape expression schemas using RBE_0 only and shape graphs, and we show that the existence of a witness of a simulation is equivalent to type satisfaction.

Proposition 2.3. ShEx_0 *captures precisely* $\mathsf{ShEx}(\mathsf{RBE}_0)$*, i.e., for every* $\mathsf{ShEx}(\mathsf{RBE}_0)$ *schema there is a shape graph in* ShEx_0 *defining the same language and for every shape graph in* ShEx_0 *there is a* $\mathsf{ShEx}(\mathsf{RBE}_0)$ *schema defining the same language.*

Embeddings are closed under composition, which immediately gives the following.

Lemma 2.4. *For any G and H, $G \preccurlyeq H$ implies $G \subseteq H$.*

The converse does not hold as illustrated in Figure 4, where two equivalent graphs are given but embedding holds only in one direction. This example basically illustrates that a shape

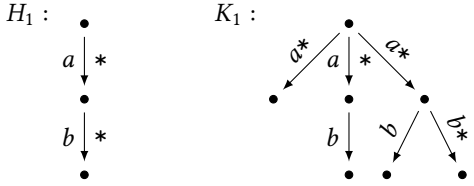


Figure 4: Containment does not imply an embedding: $H_1 \subseteq K_1$ but $H_1 \not\preccurlyeq K_1$.

expression $b :: t^*$ is equivalent to $\epsilon \mid b :: t \mid b :: t^+$, a (disjoint) union that enumerates cases of the original expression. In Section 5 we identify a practical subclass of shape graphs for which embedding is also a necessary condition for containment and then, we analyze the complexity of constructing an embedding.

3 SHAPE EXPRESSION SCHEMAS

In this section, we address the question of decidability of containment for ShEx, which is far from obvious as ShEx caries some expressive power of ∃MSO on graphs combined with Presburger arithmetic [34] and monadic extensions of Presburger arithmetic easily become undecidable [13]. We show a triple-exponential upper bound on the size of a counter-example, which can be compressed to double-exponential size. The compression does not change the complexity of validation, which permits us to give a preliminary upper bound on the complexity of testing containment for ShEx

3.1 Counter-example

To illustrate the analysis of the size of a counter-example we present in Figure 5 two shape expression schemas H_2 and K_2 and a counter-example G_2 for the containment $H_2 \subseteq K_2$. We

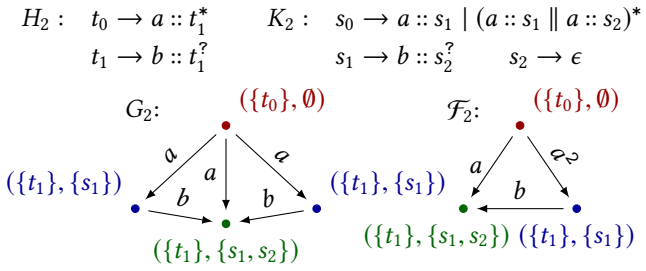


Figure 5: Two schemas $H_2 \not\subseteq K_2$ (top), a counter-example G_2 (bottom left), and its compression $\mathcal{F}_2$ (bottom right).

associate with every node of G_2 its *kind* (T, S), the set of all types T of H_2 and the set of all types S of K_2 that the node satisfies. Our goal is to construct a graph that has at most one node of any possible kind. We point out that in any counter-example there is at least one node that does not satisfy any type of K_2 (while it satisfies some types of H_2). In G_2 there is a single such node and its kind is $(\{t_0\}, \emptyset)$. This node has two a-children of the kind $(\{t_1\}, \{s_1\})$ and one a-child of the kind $(\{t_1\}, \{s_1, s_2\})$. Notice that none of the $(\{t_1\}, \{s_1\})$-nodes can be removed or the $(\{t_0\}, \emptyset)$-node would satisfy the type s_0 and the resulting graph would not longer be a counter-example. Instead, we *fuse* the $(\{t_1\}, \{s_1\})$-nodes into a single one, and use a singleton interval 2 to indicate two copies of the a-edge. Essentially, this allows to *compress* the counter-example into a graph with at most exponentially many nodes. We then use the existing results on solutions to Presburger arithmetic formulas to characterize bounds on the sizes of the intervals necessary in the compressed counter-examples.

Compression. Simple graphs do not allow multiple edges with the same label between the same pair of nodes. We propose a model that allows it by attaching to every edge a *cardinality* indicating the number of such edges. More precisely, a *singleton* interval is an interval of the form $[k; k]$ for any natural k, and a *compressed* graph is a graph that uses only singleton intervals on its edges and like simple graphs allows only one edge per label in Σ between a pair of nodes. Given a compressed graph $\mathcal{F}$, its *unpacking* is a simple graph obtained by making a sufficient number of copies of each node, each copy has the same outbound neighborhood but receiving at most one incoming edge.Since intervals are stored in binary, the unpacking of a compressed graph $\mathcal{F}$ is of size at most exponential in the size of $\mathcal{F}$.

Proposition 3.1. *The size of the unpacking of a compressed graph $\mathcal{F}$ is at most exponential in the size of $\mathcal{F}$.*

We adapt the definition of validation of ShEx to compressed graphs by extending the definition of node signature. Given a shape expression schema H and a compressed graph $\mathcal{F}$, the *signature* of a node $n \in N_{\mathcal{F}}$ w.r.t. a typing $T \subseteq N_{\mathcal{F}} \times \Gamma_H$ is

$$\mathrm{sign}^T_G(n) = \Big\|_{e \in \mathrm{out}_{\mathcal{F}}(n)} \Big(\Big|_{t \in T(\mathrm{target}_{\mathcal{F}}(e))} \mathrm{lab}_{\mathcal{F}}(e) :: t \Big)^{\mathrm{occur}_{\mathcal{F}}(e)}.$$

Again, the typing T is *valid* iff $L(\mathrm{sign}^T_G(n)) \cap L(\delta_H(t)) \neq \emptyset$ for every $(n, t) \in T$, there exists a unique maximal valid typing $\mathrm{Typing}_{\mathcal{F}:H}$ of $\mathcal{F}$ w.r.t. H, and also $\mathcal{F}$ satisfies H if $\mathrm{dom}(\mathrm{Typing}_{\mathcal{F}:H}) = N_{\mathcal{F}}$. Naturally, if $\mathcal{F}$ satisfies H, then its unpacking also satisfies H. Checking the satisfaction of ShEx for compressed graphs remains in NP and to prove it we employ known results on Presburger arithmetic that we present next.

Presburger Arithmetic. The *Presburger arithmetic* (PA) is the first-order logical theory of natural numbers with addition that has decidable satisfiability [29]. We point out that any natural number n can be easily defined with an existentially quantified formula of length linear in $\log(n)$. Since we use PA formulas to define bags, we use a convenient notation. When the set of symbols $\Delta = \{a_1, \ldots, a_k\}$ is known from the context, a bag w over Δ can be represented as a (Parikh) vector of k natural numbers $\langle w(a_1), \ldots, w(a_k) \rangle$ and if a vector of variables $\bar{x}$ is used to describe a bag over Δ, we use elements of Δ to index elements of $\bar{x}$: x_{a_i} designates $w(a_i)$, for $1 \leq i \leq k$. Also, we write $\varphi(w)$ to say that φ is valid for w.

We extend RBE with intersection $L(E_1 \cap E_2) = L(E_1) \cap L(E_2)$ because intersection is used to define satisfiability of a graph w.r.t. a schema and intersection is easily expressed in Presburger arithmetic. Now, for a regular bag expression E we recursively construct a formula $\psi_E(\bar{x}) = \psi(\bar{x}, 1)$ as

follows.

$$\begin{aligned}
\psi_\epsilon(\bar{x}, n) &:= \textstyle\bigwedge_a x_a = 0 \\
\psi_a(\bar{x}, n) &:= x_a = n \wedge \textstyle\bigwedge_{b \neq a} x_b = 0 \\
\psi_{E^{[k;\ell]}}(\bar{x}, n) &:= (n = 0 \wedge \textstyle\bigwedge_a x_a = 0) \vee \\
&\quad (n > 0 \wedge \exists m.\ k \leq m \wedge m \leq \ell \wedge \psi_E(\bar{x}, m)) \\
\psi_{E_1|E_2}(\bar{x}, n) &:= \exists \bar{x}_1, \bar{x}_2, n_1, n_2.\ n = n_1 + n_2 \wedge \\
&\quad \bar{x} = \bar{x}_1 + \bar{x}_2 \wedge \psi_{E_1}(\bar{x}_1, n_1) \wedge \psi_{E_2}(\bar{x}_2, n_2) \\
\psi_{E_1\|E_2}(\bar{x}, n) &:= \exists \bar{x}_1, \bar{x}_2.\ \bar{x} = \bar{x}_1 + \bar{x}_2 \wedge \psi_{E_1}(\bar{x}_1, n) \wedge \\
&\quad \psi_{E_2}(\bar{x}_2, n) \\
\psi_{E_1 \cap E_2}(\bar{x}, n) &:= \psi_{E_1}(\bar{x}, n) \wedge \psi_{E_2}(\bar{x}, n)
\end{aligned}$$

The main claim, proven with a simple induction, is that $\psi_E(w, n)$ iff $w \in L(E)^n$ for any bag w over Δ and any $n \geq 0$. It follows that $L(E) \neq \emptyset$ iff $\exists \bar{x}.\psi_E(\bar{x})$ is valid. Validity of existentially quantified PA formulas is known to be in NP [17], and consequently, we obtain an upper bound on complexity of validation of compressed graphs.

Proposition 3.2. *Validation of compressed graphs w.r.t.* ShEx *is in* NP.

The following result is instrumental in our analysis of upper bounds on the size of a counter-example for ShEx.

Proposition 3.3 ([39]). *Let $\Phi = Q_1\bar{x}_1 \ldots Q_k\bar{x}_k.\varphi$ be a closed formula of Presburger arithmetic in prenex normal form with k quantifier alternations over the variables $\bar{x} = \bar{x}_1 \cup \ldots \cup \bar{x}_k$ (φ is quantifier-free). Then Φ is valid if and only if Φ is valid when restricting the first-order variables of Φ to be interpreted over elements of $\{0, \ldots, B\}$, where $\log(B) = O(|\varphi|^{3|\bar{x}|^k})$.*

Compressed counter-example. We take two schemas H and K such that $H \not\subseteq K$ and fix a counter-example $G \in L(H) \setminus L(K)$. We know that there is at least one node of G that satisfies at least one type of H but no type of K. In general, for a node n of G we identify a pair (T, S) consisting of a set T of types of H and a set of types S of K that n satisfies. We say that the node n is of the (T, S)-*kind* and we identify the set C of all kinds present in G.

$$\begin{aligned}
&\mathit{kind}(n) = (\mathsf{Typing}_{G:H}(n), \mathsf{Typing}_{G:K}(n)), \\
&C = \{\mathit{kind}(n) \mid n \in N_G\}.
\end{aligned}$$

Shape expression schemas may only inspect the labels of the outgoing edges of a node and the types of the nodes at the end points of the edges. Consequently, if we replace a node by a node of the same kind, or more precisely we redirect all incoming edges of the node to the other node, then the types of the nodes in the graph do not change, in particular, it remains a counter-example. Furthermore, we can fuse the set of all nodes of the same kind into a single node that belongs to the same kind, and still obtain a graph that is a counter-example. When fusing several nodes we gather the incoming edges into a fused node but for the outgoing edges we use only the outgoing edges of one (arbitrarily chosen) of the fused nodes, while discarding the outgoing edges of the remaining nodes. We point out that the obtained graph needs not longer to be simple, fusing a set of nodes may lead to several incoming edges with the same label originating from the same node. Such multiple edges can, however, be easily compressed to a single one.

We describe the construction of the compressed counter-example $\mathcal{F}$ more precisely. First for every kind $\kappa \in C$ we pick an (arbitrarily chosen) representative node $n_\kappa \in G$ such that $\mathit{kind}(n_\kappa) = \kappa$. The set of nodes of $\mathcal{F}$ is the set of all kinds of G, $N_\mathcal{F} = C$. For every edge connecting two representative nodes $\mathcal{F}$ has a corresponding edge:

$$\begin{aligned}
E_\mathcal{F} = \{\langle \kappa, a, \kappa' \rangle \mid \exists e \in E_G.\ &\mathsf{source}_G(e) = n_\kappa, \\
&\mathsf{target}_G(e) = n_{\kappa'},\ \mathsf{lab}_G(e) = a\}
\end{aligned}$$

and for $\langle \kappa, a, \kappa' \rangle \in E_\mathcal{F}$

$$\begin{aligned}
&\mathsf{source}_\mathcal{F}(\langle \kappa, a, \kappa' \rangle) = \kappa, \quad &&\mathsf{lab}_\mathcal{F}(\langle \kappa, a, \kappa' \rangle) = a, \\
&\mathsf{target}_\mathcal{F}(\langle \kappa, a, \kappa' \rangle) = \kappa', \quad &&\mathsf{occur}_\mathcal{F}(\langle \kappa, a, \kappa' \rangle) = [k; k],
\end{aligned}$$

where

$$k = |\{e \in \mathsf{out}_G(n_\kappa) \mid \mathsf{lab}_G(e) = a,\ \mathit{kind}(\mathsf{target}_G(e)) = \kappa'\}|.$$

The main claim is that G and $\mathcal{F}$ satisfy precisely the same schemas. Furthermore, the number $|C|$ of possible kinds is at most exponential in the number of types in H and K, and from the above construction, $\mathcal{F}$ has at most one node per kind.

Bounding the node degree. The remaining question is how big the cardinalities of the edges of $\mathcal{F}$ must be. We answer this question with the help of Proposition 3.3 by describing the outbound neighborhood of a node of $\mathcal{F}$ with Presburger arithmetic formula.

For the kind $(T, S) \in C$ the formula $\Phi_{(T,S)}$ examines the existence of an outbound neighborhood of a node of that kind that satisfies all types in T and all types in S. This neighborhood is captured as a bag $\bar{x}$ over $\Delta_C = \{a :: (T', S') \mid a \in \Sigma,\ (T', S') \in C\}$, where an occurrence of the symbol $a :: (T', S')$ corresponds to one outgoing edge labeled with a and leading to a node of the kind (T', S').

$$\begin{aligned}
\Phi_{(T,S)} := \exists \bar{x}.\ &\bigwedge_{t \in T} \varphi_t(\bar{x}) \wedge \bigwedge_{t \in \Gamma_H \setminus T} \neg\varphi_t(\bar{x}) \wedge \\
&\bigwedge_{s \in S} \varphi_s(\bar{x}) \wedge \bigwedge_{s \in \Gamma_K \setminus S} \neg\varphi_s(\bar{x}).
\end{aligned}$$

The formulas $\varphi_t(\bar{x})$ and $\varphi_s(\bar{x})$ verify whether the types t of H and s of K are satisfied in this neighborhood. This is done in two phases and we present it only for $\varphi_t(\bar{x})$; the formula $\varphi_s(\bar{x})$ is defined analogously. The variable $x_{a::(T',S')}$ represents the number of outgoing edges with label a to nodes that satisfy all types in T' (and types in S'). In the

context of satisfying definition of the type t each outgoing edge is used with exactly one type. Consequently, the next formula partitions the number of outgoing edges $x_{a::(T',S')}$ into all types in T'. Here, we use a vector $\bar{y}$ of variables over $\{a :: (T',S') \to a :: t' \mid a \in \Sigma,\ (T',S') \in C,\ t' \in T'\}$, where $y_{a::(T',S')\to a::t'}$ represents the part of $x_{a::(T',S')}$ edges that is to be used with the type t'.

$$\varphi_t(\bar{x}) := \exists \bar{y}.\ \bigwedge_{a::(T',S')\in\Delta_C} x_{a::(T',S')} = \sum_{t'\in T'} y_{a::(T',S')\to a::t'} \wedge \varphi'_t(\bar{y}).$$

Finally, the edges with the same label and type of the end point are aggregated in the vector $\bar{z}$ representing a bag over $\Delta_H = \Sigma \times \Gamma_H$, which is then is fed to the formula $\psi_{\delta_H(t)}$ that defines the type definition of t (cf. proof of Proposition 3.2).

$$\varphi'_t(\bar{y}) := \exists \bar{z}.\ \bigwedge_{a::t'\in\Delta_H} z_{a::t'} = \sum_{\substack{a::(T',S')\in\Delta_C \\ \text{s.t. } t'\in T'}} y_{a::(T',S')\to a::t'} \wedge \psi_{\delta_H(t)}(\bar{z}).$$

The formula $\Phi_{(T,S)}$ can be easily converted to prenex normal form, and then, it is of exponential length, uses an exponential number of quantified variables, and has only one alternation of quantifiers. Since $\Phi_{(T,S)}$ is valid for any $(T,S) \in C$, by Proposition 3.3 the satisfying values for the variables $\bar{x}$ are bound by a triple exponential, and consequently, have a binary representation whose size is bounded by double-exponential function in the size of H and K.

Theorem 3.4. *For any two* ShEx *H and K, if $H \not\subseteq K$, then there exists a compressed graph $\mathcal{F}$ that satisfies H, does not satisfy K, and whose size is at most double-exponential in the size of H and K.*

3.2 Complexity

Very recently containment for RBE has been shown to be coNEXP-complete [18], and immediately, we obtain this lower bound.

Proposition 3.5. *Containment for* ShEx *is* coNEXP*-hard.*

The upper bound follows from Theorem 3.4 and Proposition 3.2. A (universally) nondeterministic Turing machine for an input pair (H,K) guesses a compressed graph $\mathcal{F}$ and uses an NP oracle to verify that $\mathcal{F}$ satisfies the schema H and violates the schema K. The input pair is accepted if the test is passed on every computation path.

Corollary 3.6. *Containment for* ShEx *is in* co2NEXP$^{\text{NP}}$.

4 SHAPE GRAPHS

In this section we consider shape graphs ShEx_0, which correspond to the subclass $\mathsf{ShEx}(\mathsf{RBE}_0)$ of shape expression schemas that use only RBE_0 expression for type definitions (cf. Proposition 2.3). First, we show that the size of a counter-example is at most exponential and that the bound is tight. Then, we show that the complexity of the containment problem for ShEx_0 is EXP-complete.

4.1 Counter-example

In Section 3 we have presented an argument showing that a smallest counter-example has at most exponential number of nodes, and next we show that for ShEx_0 this bound is in fact tight.

Lemma 4.1. *For any n, there exist two shape graphs H and K such that $H \not\subseteq K$ and the smallest graph $G \in L(H) \setminus L(K)$ is of size exponential in n.*

Proof. In our construction the counter-example i.e., $G \in L(H) \setminus L(K)$, is essentially a binary tree of depth n modeled with the rules (for $i \in \{1,\dots,n\}$)

$$t^{(i)} \to L :: t^{(i+1)} \parallel R :: t^{(i+1)}$$

The leaves of this tree store each a subset of $A = \{a_1,\dots,a_n\}$, modeled with the two rules

$$t^{(n+1)} \to a_1 :: t_o^? \parallel \dots \parallel a_n :: t_o^? \quad t_o \to \epsilon$$

The schema H consists exactly of all the above rules while the schema K contains all but the rule defining type $t^{(1)}$. Clearly, at this point a counter-example of $H \not\subseteq K$ exists, one whose root node has type $t^{(1)}$ in H but no type in K, however, it may be small as it suffices to use a dag. To eliminate small counter-examples, by adding them to the language of K, we ensure that all leaves of the counter-example are labeled with distinct subsets of A. In essence, we require in the counter-example a node at level i to have all leaves of its left subtree labeled with subsets containing a_i and all leaves of its right subtree labeled with subsets missing a_i. For that purpose, we introduce types $s^{(j)}_{i,1,d}$ ($s^{(j)}_{i,0,d}$), which identify nodes at level j that are using (missing resp.) the symbol a_i; the additional parameter $d \in \{L,R\}$ is used to handle disjunction and essentially indicates the subtree from which the usage information comes from. The rules for leaves are (for $i \in \{1,\dots,n\}$, $M \in \{0,1\}$, and $d \in \{L,R\}$)

$$s^{(n+1)}_{i,M,d} \to a_1 :: t_o^? \parallel \dots \parallel a_{i-1} :: t_o^? \parallel a_i :: t_o^M \parallel a_{i+1} :: t_o^? \parallel \dots \parallel a_n :: t_o^?$$

The information of using a symbol a_i in a branch is propagated upward but only to the level $i+1$ with the rules (for $i \in \{1,\dots,n\}$, $j \in \{i+1,\dots,n\}$, and $M \in \{0,1\}$)

$$s^{(j)}_{i,M,L} \to L :: s^{(j+1)?}_{i,M,L} \parallel L :: s^{(j+1)?}_{i,M,R} \parallel R :: t^{(j)}$$
$$s^{(j)}_{i,M,R} \to L :: t^{(j)} \parallel R :: s^{(j+1)?}_{i,M,L} \parallel R :: s^{(j+1)?}_{i,M,R}$$

Finally, a tree is invalid for our purposes if a node at depth i is missing the symbol a_i in a leaf of its left subtree or is using the symbol a_i in a leaf of its right subtree. This situation is identified and propagated to the root node with the rules

(for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, i-1\}$)

$$p^{(i)}_{i,L} \to L :: s^{(i+1)?}_{i,0,L} \parallel L :: s^{(i+1)?}_{i,0,R} \parallel R :: t^{(i+1)}$$
$$p^{(i)}_{i,R} \to L :: t^{(i+1)} \parallel R :: s^{(i+1)?}_{i,1,L} \parallel R :: s^{(i+1)?}_{i,1,R}$$
$$p^{(j)}_{i,L} \to L :: p^{(j+1)?}_{i,L} \parallel L :: p^{(j+1)?}_{i,R} \parallel R :: t^{(j+1)}$$
$$p^{(j)}_{i,R} \to L :: t^{(j+1)} \parallel R :: p^{(j+1)?}_{i,L} \parallel R :: p^{(j+1)?}_{i,R}$$

Now, the claim, proven with a simple induction, is that for any $G \in L(H)$ unless G contains an exponential tree, any node that has the type $t^{(1)}$ of H also has a type $p^{(1)}_{i,d}$ of K for some $i \in \{1, \ldots, n\}$, $M \in \{0, 1\}$, and $d \in \{L, R\}$. □

The lower bound on the size of a minimal counter-example for ShEx_0 is tight.

Theorem 4.2. *For any $H, K \in \mathsf{ShEx}_0$ such that $H \not\subseteq K$ there exists a graph $G \in L(H) \setminus L(K)$ whose size is at most exponential in the size of H and K.*

The proof consists of two parts. The first shows that there are at most exponentially many kinds of nodes, and we use the same argument in the proof of Theorem 3.4 in Section 3. The second part uses a pumping argument to show that the outbound degree of a minimal counter-example is polynomially bounded.

4.2 Complexity

The lower bound on the complexity of containment for ShEx_0 is obtained with a reduction from nondeterministic top-down tree automata known to be EXP-complete [11]. The reduction is non-trivial because RBE_0 does not allow directly the disjunction necessary to express nondeterminism, and the proof needs to account for graphs that might have cycles and do not represent trees.

Theorem 4.3. *Containment for ShEx_0 is* EXP*-hard.*

Since validation for ShEx_0 is polynomial [34], the bound on the size of a counter-example yields a coNEXP procedure for testing containment. We provide, however, a tight EXP bound with an exponential procedure for deciding containment of shape graphs. We outline the main ideas with the examples that follow. The first example gives a rough sketch of the framework, while the remaining examples present more challenging aspects of the problem at hand and how we address them.

Example 4.4. We consider the two shape graphs H_3 and K_3 in Figure 6, that represent the two shape expression schemas presented in Section 1. First, we introduce the notion of type covering, which essentially for every type t of H_3 identifies sets of types of K_3 that capture t. For instance, the type U is covered by $\{U_1, U_2\}$ and the type B is covered by the types $\{B_1, B_2\}$. Then, we prove an important property of *support*

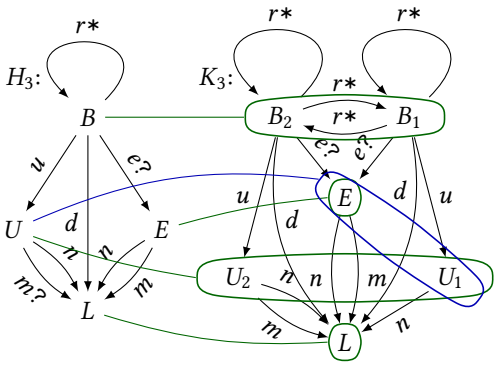


Figure 6: Covering between two shape graphs.

of type t being covered by a set of types S: the definition of the type t can be *unfolded* to a disjunction of type definitions of (a subset of) S. For instance, we take the type definition of U

$$\delta_{H_3}(U) = n :: L \parallel m :: L^?,$$

and decompose $m :: L^?$ into the following disjunction

$$(n :: L) \mid (n :: L \parallel m :: L)$$

which is equal to $\delta_{K_3}(U_1) \mid \delta_{K_3}(U_2)$. This indeed shows that U is covered by $\{U_1, U_2\}$. Here, the types L and E are the same in both schemas and for simplicity use the same name.

Because shape expression schemas are recursive, the notion of support needs to be defined in a (co)inductive fashion. For instance, for the type definition of B

$$\delta_{H_3}(B) = r :: B^* \parallel u :: U \parallel d :: L \parallel e :: E^?$$

since U is covered by $\{U_1, U_2\}$ we get

$$(r :: B^* \parallel u :: U_1 \parallel d :: L \parallel e :: E^?) \mid (r :: B^* \parallel u :: U_2 \parallel d :: L \parallel e :: E^?)$$

and since B is covered by $\{B_1, B_2\}$

$$(r :: B_1^* \parallel r :: B_2^* \parallel u :: U_1 \parallel d :: L \parallel e :: E^?) \mid (r :: B_1^* \parallel r :: B_2^* \parallel u :: U_2 \parallel d :: L \parallel e :: E^?)$$

which is equal to $\delta_{K_3}(B_1) \mid \delta_{K_3}(B_2)$. This shows that B is covered by $\{B_1, B_2\}$ even though this very fact is assumed to hold when constructing the unfolding. □

While unfolding atoms $a{::}t^1$ and $a{::}t^?$ is relatively straightforward, the next example shows that unfolding atoms $a :: t^*$ is more complicated than the previous example might suggest.

Example 4.5. Consider the following two schemas:

H_4: $t_0 \to a :: t^*$
$t \to a :: t_\varnothing \parallel a :: t^?_\varnothing$
$t_\varnothing \to \epsilon$

K_4: $s_0 \to a :: s_1^*$ $\quad s_0' \to a :: s_2^+ \parallel a :: s_1^*$
$s_1 \to a :: s_\varnothing$ $\quad s_2 \to a :: s_\varnothing \parallel a :: s_\varnothing$
$s_\varnothing \to \epsilon$

Clearly, $t_\varnothing$ is covered by $\{s_\varnothing\}$, and consequently, t is covered by $\{s_1, s_2\}$. We also point out that s_1 and s_2 are incomparable and so are s_0 and s_0'. Essentially, s_0 allows only outgoing edges that lead to nodes of type s_1, while s_0' requires at least one outgoing edge that leads to a node of type s_2 and an arbitrary number of edges that lead to nodes of type s_1. Naturally, t_0 is covered by $\{s_0, s_0'\}$. When constructing the unfolding of $\delta_{H_4}(t) = a :: t_0^*$ we observe that since t is covered by $\{s_1, s_2\}$ the type definition $a :: t^*$ can be unfolded to

$$a :: s_1^* \mid (a :: s_2 \parallel a :: t^*)$$

and then again to

$$a :: s_1^* \mid (a :: s_2 \parallel a :: s_1^* \parallel a :: s_2^*)$$

which after a simple normalization yields $\delta_{K_4}(s_0) \mid \delta_{K_4}(s_0')$. □

The next example illustrates why we trace the lineage of atoms in unfolding and why we additionally consider the problem of testing emptiness of intersection. In this example we use arbitrary intervals for brevity, and for instance, $a::t^{[1;2]}$ is short for $a :: t \parallel a :: t^?$.

Example 4.6. Consider the following two schemas:

$$\begin{aligned}
H_5\colon\ & t_0 \to c :: t \\
& t \to a :: t_\varnothing^{[1;2]} \parallel b :: t_\varnothing^{[1;2]} \\
& t_\varnothing \to \epsilon \\
K_5\colon\ & p_0 \to c :: p_1 \quad p_0' \to c :: p_2 \quad s_0 \to c :: s_1 \\
& p_1 \to a :: t_\varnothing \parallel b :: t_\varnothing \qquad s_1 \to a :: t_\varnothing^2 \parallel b :: t_\varnothing^{[1;2]} \\
& p_2 \to a :: t_\varnothing^{[1;2]} \parallel b :: t_\varnothing^2 \qquad s_2 \to a :: t_\varnothing^3 \parallel b :: t_\varnothing^{[1;2]} \\
& p_3 \to a :: t_\varnothing^{[2;3]} \parallel b :: t_\varnothing \qquad t_\varnothing \to \epsilon
\end{aligned}$$

It is easy to see that t is covered by $\{p_1, p_2, p_3\}$ and p_3 is covered by $\{s_1, s_2\}$. We shall prove that t_0 is covered by $\{p_0, p_0', s_0\}$ by constructing an unfolding of $\delta_{H_5}(t_0) = c::t$. We first use the fact that t is covered by $\{p_1, p_2, p_3\}$ and obtain the following disjunction

$$c :: p_1 \mid c :: p_2 \mid c :: p_3 = \delta_{K_5}(p_0) \mid \delta_{K_5}(p_0') \mid c :: p_3$$

To further unfold the atom $c :: p_3$ one use the fact that p_3 is covered by $\{s_1, s_2\}$, which yields

$$\delta_{K_5}(p_0) \mid \delta_{K_5}(p_0') \mid \delta_{K_5}(s_0) \mid c :: s_2$$

The remaining atom $c :: s_2$ is in fact *void* and can be discarded. Indeed, if we inspect its lineage, we observe that the type t has been initially replaced by p_3 and then by s_2. However, the intersection of the types $t \cap p_3 \cap s_2$ is empty i.e., there is no graph with a node having the three types. □

4.2.1 Emptiness of intersection. To identify void atoms in derivations we present a method for identifying sets of types whose intersection is empty, a problem of independent interest. The method is based on techniques that are simpler versions of those central in proving Lemma 4.11, which are too complex to be presented in full detail.

We introduce the notion of *rooted graph*, which is a simple graph with one distinguished root node. We fix a shape graph H and for a shape expression E by $[\![E]\!]$ we denote the set of rooted graphs whose root nodes satisfy E. We extend this notion to sets of shape expressions $[\![\mathcal{E}]\!] = \bigcup\{[\![E]\!] \mid E \in \mathcal{E}\}$. Since shape expressions use RBE_0 and the unordered concatenation operator is commutative, we view shape expressions as unordered collections (bags) of atoms of the form $a :: t^M$ without repetitions of atoms using $*$. We also use the empty atom ϵ that is the neutral element of the unordered concatenation operator i.e., $E = E \parallel \epsilon$. W.l.o.g. we assume that the interval + is not used: indeed $a :: t^+$ can be replaced by $a :: t \parallel a :: t^*$.

Our method is based on pumping and tagging shape expressions, which reduces the problem to (disjunctions) of single-occurrence expressions that use only the interval 1, denoted $\mathsf{SORBE}_0(1)$. Let $m = \max\{|\delta_H(t_1)|, \ldots, |\delta_H(t_k)|\}+1$, where $|E|$ is the number of atoms in E and $\Gamma_H = \{t_1, \ldots, t_k\}$. Pumping removes ? and $*$ by producing expressions with none or one occurrences of atoms using ? and up to m occurrences of atoms using $*$. For instance, for $m = 2$ pumping $a :: t^? \parallel b :: s \parallel c :: t^*$ yields

$$\begin{aligned}
& b :: s, && a :: t \parallel b :: s, \\
& b :: s \parallel c :: t, && a :: t \parallel b :: s \parallel c :: t, \\
& b :: s \parallel c :: t \parallel c :: t, && a :: t \parallel b :: s \parallel c :: t \parallel c :: t.
\end{aligned}$$

Note that after pumping the obtained shape expression use only the interval 1. To obtain single-occurrence expressions we use tagging, which considers all permutations of atoms in an expression and numbers the symbols accordingly. For instance, the expression $a :: t \parallel a :: s \parallel b :: t$ has the following taggings

$$\begin{aligned}
& a_1 :: t \parallel a_2 :: s \parallel b_3 :: t, && a_1 :: t \parallel a_3 :: s \parallel b_2 :: t, \\
& a_2 :: t \parallel a_1 :: s \parallel b_3 :: t, && a_2 :: t \parallel a_3 :: s \parallel b_1 :: t, \\
& a_3 :: t \parallel a_1 :: s \parallel b_2 :: t, && a_3 :: t \parallel a_2 :: s \parallel b_1 :: t.
\end{aligned}$$

Our procedure constructs the set $\mathcal{X}_H$ of all subsets of types of H whose intersection is nonempty

$$\mathcal{X}_H = \{\{t_1, \ldots, t_k\} \subseteq \Gamma_H \mid [\![\delta_H(t_1)]\!] \cap \ldots \cap [\![\delta_H(t_k)]\!] \neq \emptyset\}$$

Initially, we begin $\mathcal{X}_0 = \mathcal{P}(\Gamma_H)$ and we iteratively refine it as follows. Suppose, at an iteration we have a set $\mathcal{X} \subseteq \mathcal{P}(\Gamma_H)$. Let $E_1, \ldots, E_k$ be $\mathsf{SORBE}_0(1)$ expressions each having the same number n of atoms and using precisely the same labels $a_1, \ldots, a_n$ i.e., $E_i = a_1::t_{i,1} \| \ldots \| a_n::t_{i,n}$ for $i \in \{1, \ldots, k\}$. We say that $E_1, \ldots, E_k$ are *supported* by $\mathcal{X}$ if $\{t_{1,j}, \ldots, t_{k,j}\} \in \mathcal{X}$ for every $j \in \{1, \ldots, n\}$. Take any $\{t_1, \ldots, t_k\} \in \mathcal{X}$ and for $i \in \{1, \ldots, k\}$ let $\mathcal{E}_i$ be the set of $\mathsf{SORBE}_0(1)$ expressions obtained by pumping and tagging $\delta_H(t_i)$. We say that

$\{t_1, \ldots, t_k\}$ is *supported* by $\mathcal{X}$ if for every $i \in \{1, \ldots, k\}$ there is $E_i \in \mathcal{E}_i$ such that $E_1, \ldots, E_k$ are supported by $\mathcal{X}$. The one-step refinement function is

$$\mathit{Refine}(\mathcal{X}) = \{T \in \mathcal{X} \mid T \text{ is supported by } \mathcal{X}\}.$$

Naturally, the above function is monotone, and therefore, when iteratively applied to $\mathcal{X}_0 = \mathcal{P}(\Gamma_H)$ it has a fix-point $\mathit{Refine}^*(\mathcal{X}_0)$. We claim that this fix point is in fact $\mathcal{X}_H$. To prove that this procedure works in time exponential in the size of H it suffices to observe that each iteration of *Refine* removes from $\mathcal{X}$ at least one element, and therefore, the number of iteration is at most exponential in the number of types of H, and each iteration works in time exponential since the sets $\mathcal{E}_i$ have a number of expressions exponential in m and each expression is of size at most m^2.

Lemma 4.7. *Checking emptiness of intersection of a set of types of a shape graph is in* EXP.

4.2.2 Type coverings. We begin with a natural notion of coverage that captures the disjunction introduced by schemas. For technical reasons and in the interest of simplicity of presentation, the definitions in the reminder of this section are for a single shape graph $\mathcal{H}$ that is obtained by taking the disjoint union $H \uplus K$ of the two shape graphs H and K.

Definition 4.8. Given a shape expression schema $\mathcal{H}$, a type $t \in N_{\mathcal{H}}$ is *covered* by a set of types $S \subseteq N_{\mathcal{H}}$ iff in any simple graph G all nodes satisfying the type t also satisfy one of the types in S i.e.,

$$\forall G \in \mathsf{G}_0.\ \mathsf{Typing}^{-1}_{G:\mathcal{H}}(t) \subseteq \bigcup_{s \in S} \mathsf{Typing}^{-1}_{G:\mathcal{H}}(s).$$

The *type covering* of $\mathcal{H}$ is the relation

$$\mathsf{Covering}_{\mathcal{H}} = \{(t, S) \in N_{\mathcal{H}} \times \mathcal{P}(N_{\mathcal{H}}) \mid t \text{ is covered by } S\}.$$

□

Type covering allows us to decide containment in a straightforward fashion.

Proposition 4.9. *$H \subseteq K$ if and only if for every $t \in N_H$ there is $S \subseteq N_K$ such that $(t, S) \in \mathsf{Covering}_{H \uplus K}$.*

We now fix a shape graph $\mathcal{H} = H \uplus K$ and propose an iterative algorithm for constructing the type covering of $\mathcal{H}$ using a local characterization of *support* for elements of type covering (Definition 4.10). The algorithm begins with the full relation $R = N_{\mathcal{H}} \times \mathcal{P}(N_{\mathcal{H}})$ and iteratively removes any (t, S) from R that is not supported by R. It returns the maximal self-supported relation R which is precisely the type covering (Lemma 4.11). We show that testing whether an element of R is supported can be done in exponential time (Lemma 4.12), and since R has an exponential number of elements and at most exponential number of iterations is performed, the algorithm works in exponential time (Theorem 4.13).

Unfolding. Again, we view shape expressions as unordered collections (bags) of atoms of the form $a :: t^M$, without repetitions of atoms using $*$, and we assume that the interval $+$ is not used. We propose a method that for a pair $(t, S) \in R$ uses R and basic properties of RBE_0 to unfold the type definition of t into a disjunction of shape expressions. If t can be unfolded into a disjunction contained in S, then (t, S) is supported by R and there is no reason to believe that t is not covered by S, and consequently, no reason to remove (t, S) from R (at this iteration).

The unfolding is defined with a set of unfolding operations on RBE_0 atoms that return a disjunction of RBE_0 expressions. This set captures: the disjunction from RBE_0

$$a::t^? \to \epsilon \mid a::t \qquad a::t^* \to \epsilon \mid (a::t \parallel a::t^*) \tag{1}$$

from every $(t, \{s_1, \ldots, s_m\}) \in R$ the straightforward disjunction

$$a::t^1 \to a::s_1^1 \mid \ldots \mid a::s_m^1 \tag{2a}$$

$$a::t^? \to a::s_1^? \mid \ldots \mid a::s_m^? \tag{2b}$$

as well as disjunction from $a :: t^*$ atoms (for $0 \leq k \leq m$)

$$\begin{aligned} a::t^* \to (a::s_1^* \parallel \ldots \parallel a::s_k^*) \mid \\ (a::s_{k+1} \parallel a::t^*) \mid \ldots \mid (a::s_m \parallel a::t^*) \end{aligned} \tag{2c}$$

and finally, the containment of RBE_0

$$\epsilon \to a::t^? \qquad a::t \to a::t^? \qquad a::t^? \to a::t^* \tag{3}$$

An *unfolding tree* w.r.t. R is a unranked tree whose nodes are labeled with RBE_0 expressions and if a non-leaf node is labeled with an expression $E \parallel e$, then there is an unfolding operation (w.r.t. R) $e \to e_1 \mid \ldots \mid e_m$ and the children of the node are labeled with the expressions $E \parallel e_1, \ldots, E \parallel e_m$. Furthermore, with a type t in a node we associate its *lineage*: the set of types $\{t, t_1, \ldots\}$ of its ancestors that have led to the type. A type used at a node is *void* if the intersection of its lineage is empty. A type t that is void at a node n of an unfolding tree is *pruned* as follows: if t is used in an atom $a :: t$, then the node and all its descendants are removed; if t is used in an atom $a :: t^?$ or $a :: t^*$, then the atom is removed from that the expression at that node and so is any derived atom in the descendants of the node. Now, an *unfolding* of an RBE_0 E w.r.t. R is a unfolding tree w.r.t. R whose root is labeled with all types pruned.

Support. We use unfolding to identify a defining property of the type covering.

Definition 4.10. Given a shape graph $\mathcal{H}$ and $R \subseteq N_{\mathcal{H}} \times \mathcal{P}(N_{\mathcal{H}})$, a pair $(t, S) \in N_{\mathcal{H}} \times \mathcal{P}(N_{\mathcal{H}})$ is *supported* by R if $\delta_{\mathcal{H}}(t)$ has an unfolding w.r.t. R contained in $\delta_{\mathcal{H}}(S) = \{\delta_{\mathcal{H}}(s) \mid s \in S\}$. $R \subseteq N_{\mathcal{H}} \times \mathcal{P}(N_{\mathcal{H}})$ is *self-supported* if every $(t, S) \in R$ is supported by R. □

Because the union of two self-supported relations is also self-supported, there exists precisely one maximal self-supported relation. The support property is an alternative definition of the type covering, which is the base of our algorithm for constructing the type covering.

Lemma 4.11. *For any $\mathcal{H} \in \mathsf{ShEx}_0$, the type covering of $\mathcal{H}$ is the maximal self-supported relation.*

Proof. The proof is non-trivial, technically complex, and we outline only the main key ideas. We show that type covering is self-supported i.e., if t is covered by S then $\delta_{\mathcal{H}}(t)$ has an unfolding (w.r.t. the type uncovering) contained in $\delta_{\mathcal{H}}(S)$, using a series of complex arguments. We generalize the notion of covering to shape expressions in the natural fashion.

First, we use a pumping and tagging technique, similar to the one in Section 4.2.1. We obtain the set of $\mathsf{SORBE}_0(1)$ expression $\mathcal{E}$ from $\delta_{\mathcal{H}}(t)$ and the set $\mathcal{E}'$ from $\delta_{\mathcal{H}}(S)$, and we show that every $E \in \mathcal{E}$ is covered by $\mathcal{E}'$. We view a shape expression $E = a_1 :: t_1 \,\|\, \ldots \,\|\, a_n :: t_n$ as defining a n-cube Q that is covered by a set $\{Q_1, \ldots, Q_k\}$ of n-cubes corresponding to expressions from $\mathcal{E}'$. We then show that Q is covered by a single n-cube, which is captured with (3) operations or it can be decomposed into smaller n-cubes each covered by a proper subset of $\{Q_1, \ldots, Q_k\}$, which is captured with (2a) operations. This yields an unfolding of E into $\mathcal{E}'$ that we use as a skeleton for constructing an unfolding of $\delta_{\mathcal{H}}(t)$ into $\delta_{\mathcal{H}}(S)$. Furthermore, we observe that the height of the constructed unfolding is polynomially-bounded.

To show that the type covering is the maximal self-supported relation, we first prove the following claim.

Claim. *Let $\mathcal{R}$ be the maximal self-supported relation for $\mathcal{H}$. For any $(t, S) \in \mathcal{R}$ that is supported by $\mathcal{R}$ with an unfolding using the facts $(t_1, S_1), \ldots, (t_k, S_k) \in \mathcal{R}$, for any graph G and any node $n \in N_G$ of type t, if for every child m of n the fact that m has a type t_i also implies that m has a type in S_i, then n has a type in S.*

We use the above claim to show that for any $(t, S) \in \mathcal{R}$, in any graph G, any node of type t has also a type in S, and consequently, $(t, S) \in \mathsf{Covering}_{\mathcal{H}}$. □

Search graph. The polynomial bound on the depth of an unfolding allows to show that the number of relevant expressions derived from $\delta_{\mathcal{H}}(t)$ is bounded exponentially, even if we store the lineage information with every type. We construct an oriented hypergraph, whose nodes are all relevant expressions and oriented hyperedges represent one-step unfoldings with void atoms pruned with the help of Lemma 4.7. We reduce the problem of checking the existence of an unfolding to the reachability problem in alternating graphs, known to be P-complete. This gives us an exponential upper bound on testing the existence of a relevant unfolding.

Lemma 4.12. *For $\mathcal{H} \in \mathsf{ShEx}_0$ and $R \subseteq N_{\mathcal{H}} \times \mathcal{P}(N_{\mathcal{H}})$, checking that $(t, S) \in R$ is supported by R can be done in time exponential in the size of $\mathcal{H}$.*

Lemmas 4.11 and 4.12 together with Proposition 4.9 give.

Theorem 4.13. *Containment for ShEx_0 is in* EXP.

5 DETERMINISM

In this section we identify a tractable subclass of deterministic shape expression schemas that is arguably of practical use (recall that determinism forbids using the same edge label twice in type definition). We show that containment of deterministic shape expression schemas is intractable even if type definitions use RBE_0 only (Theorem 5.7). Interestingly, if the set of intervals is further restricted to 1 and $*$, containment is equivalent to the existence of embedding, which we show to be tractable (Theorem 5.4). Adding support for other basic intervals is tricky as unrestricted use of ? leads to intractability. Consequently, we employ the technique of *characterizing example* [27, 35] to find a relatively rich and tractable subclass of deterministic shape graphs that allows unrestricted use of 1 and $*$ and a restricted use of ?. We believe this subclass is of potential practical interest, and in particular it includes the schema in Figure 1. While our technique does allow to include +, the further restriction this addition causes render the class impractical, and consequently, we forbid + altogether (in practice using $*$ instead of + is often acceptable).

We first illustrate the method of characterizing example on the example in Figure 7, where we generate a characterizing example G_6 for the schema H_6 (which is a representation of the schema in Figure 1). The central property of G_6 is that any

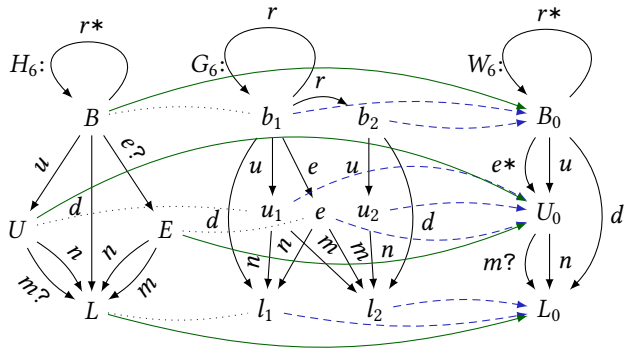


Figure 7: Constructing embedding of H_6 into W_6 from embedding of the characterizing example G_6 into W_6.

schema K (in $\mathsf{DetShEx}_0^-$) that satisfies G_6 is satisfied by all

graphs that satisfy H_6. In other words, H_6 can be embedded in any K such that $G_6 \in L(K)$ and hence for any $G' \in L(H_6)$ we have $G' \in L(K)$. The proof takes the embedding of G_6 in K and constructs an embedding of H_6 in K. Such a construction is feasible due to a key observation, which we illustrate on the schema W_6. Since the node b_1 has the type B_0, by determinism of W_6 the nodes b_1 and b_2 have the same type B_0. Transitively, the nodes u_1 and u_2 have the same type U_0 and similarly the nodes l_1 and l_2 have the same type L_0. The characterizing example G_6 is constructed in such a manner as to make sure that every type of H_6 is described by a set of nodes of G_6, all embedded into a corresponding type of W_6, which ensures embedding of the type of H_6 in the corresponding type of W_6. For instance, u_1 and u_2 are embedded into U_0 and since u_1 has an outgoing m-edge and u_2 does not, the definition of the corresponding type in W_6 must use $m :: L_0^?$ or $m :: L_0^*$.

We now define formally the subclass of shape expression schemas in question. Given a shape graph H and a type $t \in N_H$, a *reference* to t is any edge $e \in E_H$ that leads to t i.e. $\text{target}_H(e) = t$. A reference e is $*$*-closed* if $\text{occur}_H(e) = *$ or all references to $\text{source}_H(e)$ are $*$-closed.

Definition 5.1. A shape graph H is *deterministic* if for every node $n \in N_H$ and every label $a \in \Sigma$, n has at most one outgoing edge labeled with a. By $\mathsf{DetShEx}_0$ we denote the class of all deterministic shape graphs. By $\mathsf{DetShEx}_0^-$ we denote 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. □

Intuitively, we require that any type using `?` must be referenced and can only be referenced (directly or indirectly) through $*$. The schema in Figure 1 belongs to $\mathsf{DetShEx}_0^-$ since both uses of the `?` operator are closed by the edge `related` with interval $*$.

5.1 Characterizing example

Interestingly, the class $\mathsf{DetShEx}_0^-$ allows construction of graphs that characterize any schema in $\mathsf{DetShEx}_0^-$ up to containment.

Lemma 5.2. *For any $H \in \mathsf{DetShEx}_0^-$, there exists a simple graph $G \in L(H)$ of size polynomial in the size of H such that for any $K \in \mathsf{DetShEx}_0^-$ we have that $G \preccurlyeq K$ implies $H \preccurlyeq K$.*

The precise construction of the graph G that characterizes H is in Appendix B. Here, we outline the main ideas and illustrate them on an example in Figure 8. In essence, for every type $t \in N_H$ the graph G needs to contain a number of nodes of type t that serve the purpose of characterizing t. If t has an outgoing $*$-edge e labeled with a that leads to the type s, then at least one node n of G that characterizes t needs to have at least two outgoing edges labeled with a that lead to nodes A that characterize the type s. When n is mapped to a

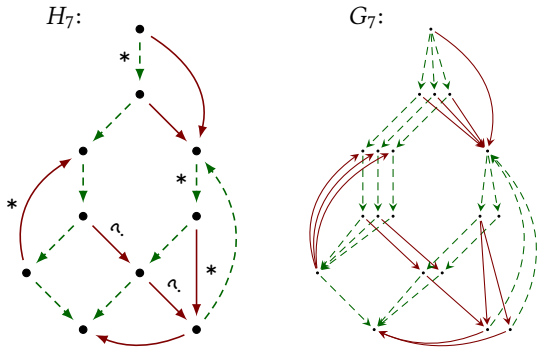


Figure 8: Characterizing example. Different colors denote different labels.

type t' of K that has an outgoing edge e' labeled with a and leading to s', all a-children A must to be mapped to s'. This shows that e' is an $*$-edge. Interestingly, this observation propagates to descendants of A. If the type s has an outgoing edge with label b that leads to the type u, then any b-child of a node in A must have the type u, and furthermore, they are all mapped to a type u' that is reachable from s' with an edge labeled with b, etc.

Now, for a type t with an outgoing `?`-edge labeled with a we need two nodes in G that characterize t, which guarantee that the corresponding type in K uses the right occurrence interval: one node with one outgoing edge with label a and one node with no such outgoing edge. Naturally, we need to make sure that those two nodes are mapped to the same type in K and this is accomplished by making sure there is an ascending path from every `?`-edge to every closest $*$-edge. In general, every type in H is characterized by a number of nodes that is at most 2 plus the number of `?`-edges in H. Lemma 5.2 renders containment and embeddings equivalent.

Corollary 5.3. *For $H, K \in \mathsf{DetShEx}_0^-$, $H \subseteq K$ iff $H \preccurlyeq K$.*

5.2 Complexity

To characterize the complexity of containment for $\mathsf{DetShEx}_0^-$ we study the complexity of testing embedding between two graphs. Interestingly, it turns out that constructing embeddings for shape graphs, which use only basic occurrence intervals, is tractable and becomes intractable if arbitrary intervals may be used. This rise in computational complexity does not come from binary encoding of intervals, in fact the results remain negative even if the arbitrary intervals are encoded in unary. We also point out that shape graphs using only basic intervals are as expressive as graphs using arbitrary intervals because RBE_0 with basic intervals and repetition are equivalent to RBE_0 with arbitrary intervals. The difference in complexity is not a contradiction but merely

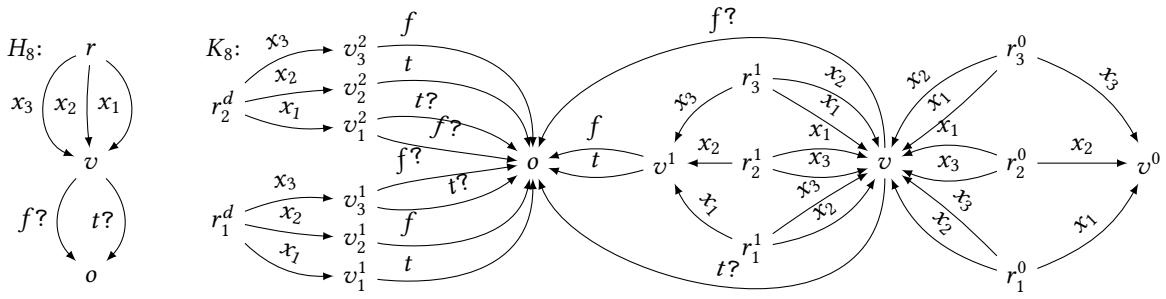


Figure 9: An example of reduction on $\varphi = (x_1 \wedge \neg x_2) \vee (x_2 \wedge \neg x_3)$.

reflects the fact that containment does not necessarily imply embedding.

Theorem 5.4. *Testing the existence of embeddings between shape graphs is in* P.

The proof of the above theorem consisting of a polynomial algorithm constructing embeddings between two graphs can be found in Appendix A. As a result of Corollary 5.3 and Theorem 5.4, we obtain.

Corollary 5.5. *Containment for* $\mathsf{DetShEx}_0^-$ *is in* P.

Constructing an embedding becomes intractable if arbitrary intervals can be employed.

Theorem 5.6. *Testing the existence of embeddings between graphs with arbitrary intervals is* NP*-complete.*

Finally, we observe that lifting the additional restrictions we impose on $\mathsf{DetShEx}_0^-$ leads to intractability.

Theorem 5.7. *Containment for* $\mathsf{DetShEx}_0$ *is* coNP*-hard.*

Proof (sketch). The proof is by reduction from tautology of DNF formulas, which we illustrate on the example of $\varphi = (x_1 \wedge \neg x_2) \vee (x_2 \wedge \neg x_3)$. We construct two deterministic schemas H_8 and K_8 presented in Figure 9. The schema H_8 is satisfied by all graphs defining a valuation of the variables of φ: a node with the root type r has outgoing edges labeled with the name of the variable leading to a node of type v that represents the value of the variable t or f. Because $\mathsf{DetShEx}_0$ does not allow disjunction, nodes of type v may also have both outgoing edges t and f, or neither of them. These cases are covered in K_8 by the types r_i^1's and r_i^0's respectively. The types r_j^d's capture precisely the valuations that satisfy the clauses of φ. Hence, K_8 is not satisfied by the graphs that correctly define a valuation that does not satisfy φ. Naturally, $H_8 \subseteq K_8$ iff φ is a tautology. □

6 CONCLUSIONS AND FUTURE WORK

This work was prompted by our recent work on data exchange for RDF [6] and ongoing work on schema inference for RDF, where not only do we ask the questions of type implication but are also interested in instances satisfying constraints expressed with the help of ShEx. In this paper, we have considered ShEx and its two practical subclasses ShEx_0 and $\mathsf{DetShEx}_0^-$. While the precise complexity of containment for ShEx remains open, the complexity results we have obtained, summarized in Figure 10, provide a good separation of the presented schema classes. Determinism shows promise in allowing reduction in complexity. For instance, containment for DetShEx is in co2NEXP since validation for DetShEx is in P. But its precise impact on complexity of containment needs to be studied further. It is an open question whether using arbitrary intervals in shape graphs has an impact on the complexity of testing containment; interestingly the answer to this question is negative for ShEx and positive for $\mathsf{DetShEx}_0^-$. The class of regular bag expression DIME that permits restricted use of disjunction yet allows for tractable containment for schemas for unordered XML [5] and it would also be interesting to see if there are any computational benefits that can be drawn for shape expression schemas using DIME.

$\mathsf{DetShEx}_0^-$	ShEx_0	ShEx
P	EXP-complete	coNEXP-hard co2NEXP$^{\mathrm{NP}}$

Figure 10: Summary of complexity results

ACKNOWLEDGMENT

We would like to thank the referees for many useful comments. This work has been supported by Polish National Science Center grant UMO-2014/15/D/ST6/00719. Part of this work has been done when one of authors was a fellow at the University of Edinburgh, funded by the EU DIACHRON project.

REFERENCES

[1] A. Abbas, P. Genevès, C. Roisin, and N. Layaïda. Optimising SPARQL query evaluation in the presence of shex constraints. In *BDA 2017*, pages 1–12, Nancy, France, November 2017.

[2] M. Arenas, C. Gutierrez, and J. Pérez. Foundations of RDF databases. In *Reasoning Web*, International Summer School on Semantic Technologies for Information Systems, pages 158–204, 2009. Invited Tutorial.

[3] M. Arenas, J. Pérez, Reutter J., C. Riveros, and J. Sequeda. Data exchange in the relational and RDF worlds. International Workshop on Semantic Web Information Management (SWIM), June 2011. Invited talk.

[4] G. J. Bex, F. Neven, and J. Van den Bussche. DTDs versus XML Schema: A practical study. In *WebDB*, pages 79–84, 2004.

[5] I. Boneva, R. Ciucanu, and S. Staworko. Schemas for unordered XML on a DIME. *Theoretical Computer Science (TCS)*, 57(2):337–376, 2015.

[6] I. Boneva, J. Lozano, and S. Staworko. Relational to RDF data exchange in presence of a shape expression schema. In *Alberto Mendelzon International Workshop on Foundations of Data Management*, May 2018.

[7] A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1998.

[8] L. Cardelli and G. Ghelli. TQL: a query language for semistructured data based on the ambient logic. *Mathematical Structures in Computer Science*, 14(3):285–327, 2004.

[9] Š. Čebirić, F. Goasdoué, P. Guzewicz, and I. Manolescu. Compact Summaries of Rich Heterogeneous Graphs. Research Report RR-8920, INRIA Saclay ; Université Rennes 1, July 2018.

[10] Š. Čebirić, F. Goasdoué, and I Manolescu. A Framework for Efficient Representative Summarization of RDF Graphs. In *International Semantic Web Conference (ISWC)*, Vienna, Austria, October 2017.

[11] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. 1997. Release 2007.

[12] S. Dal-Zilio and D. Lugiez. XML schema, tree logic and sheaves automata. In *Rewriting Techniques and Applications*, pages 246–263, 2003.

[13] C. C. Elgot and M. O. Rabin. Decidability and undecidability of extensions of second (first) order theory of (generalized) successor. *Journal of Symbolic Logic*, 31(2):169–181, 1966.

[14] M. F. Fernandez and D. Suciu. Optimizing regular path expressions using graph schemas. In *International Conference on Data Engineering (ICDE)*, pages 14–23, 1998.

[15] W. Gelade, W. Martens, and F. Neven. Optimizing schema languages for XML: Numerical constraints and interleaving. *SIAM Journal on Computing*, 38(5):2021–2043, 2009.

[16] F. Goasdoué, P. Guzewicz, and I. Manolescu. Incremental structural summarization of RDF graphs. In *International Conference on Extending Database Technology (EDBT)*, Lisbon, Portugal, March 2019.

[17] Erich Grädel. *The complexity of subclasses of logical theories*. PhD thesis, Universität Basel, 1987.

[18] C. Haase and P. Hofman. Tightening the complexity of equivalence problems for commutative grammars. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 41:1–41:14, 2016.

[19] A. Klarlund, T. Schwentick, and D. Suciu. XML: Model, schemas, types, logics and queries. In J. Chomicki, R. van der Meyden, and G. Saake, editors, *Logics for Emerging Applications of Databases*. Springer-Verlag, 2003.

[20] E. Kopczynski and A. To. Parikh images of grammars: Complexity and applications. In *Logic in Computer Science (LICS)*, pages 80–89, 2010.

[21] J. E. Labra Gayo, E. Prud'hommeaux, I. Boneva, and D. Kontokostas. Validating RDF data. *Synthesis Lectures on the Semantic Web: Theory and Technology*, 7(1):1–328, 2017.

[22] J. E. Labra Gayo, E. Prud'hommeaux, I. Boneva, S. Staworko, H. R. Solbrig, and S. Hym. Towards an RDF validation language based on regular expression derivatives. In *EDBT/ICDT Workshops (GraphQ & LWDM)*, pages 197–204, March 2015.

[23] J. E. Labra Gayo, E. Prud'hommeaux, H. Solbrig, and J. M. Alvarez Rodriguez. Validating and describing linked data portals using RDF Shape Expressions. In *Workshop on Linked Data Quality*, September 2015.

[24] W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for XML schemas and chain regular expressions. *SIAM Journal on Computing*, 39(4):1486–1530, 2009.

[25] W. Martens, F. Neven, T. Schwentick, and G. J. Bex. Expressiveness and complexity of XML schema. *ACM Transactions on Database Systems (TODS)*, 31(3):770–813, 2006.

[26] A. J. Mayer and L. J. Stockmeyer. The complexity of word problems - this time with interleaving. *Information and Computation*, 115(2):293–311, 1994.

[27] G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *Journal of the ACM*, 51(1):2–45, 2004.

[28] F. Neven and T. Schwentick. XML schemas without order. 1999.

[29] D. C. Oppen. A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic. *Journal of Computer and System Sciences*, 16(3):323–332, 1978.

[30] R. M. Robinson. Restricted set-theoretical definitions in arithmetic. In *Proceedings of the American Mathematical Society*, pages 238–242, 1958.

[31] H. Seidl, T. Schwentick, and A. Muscholl. Numerical document queries. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 155–166, 2003.

[32] H. Seidl, T. Schwentick, and A. Muscholl. Counting in trees. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*., pages 575–612, 2008.

[33] Helmut Seidl. Haskell overloading is dexptime-complete. *Information Processing Letters*, 52(2):57–60, October 1994.

[34] S. Staworko, I. Boneva, J. E. Labra Gayo, S. Hym, E. G. Prud'hommeaux, and H. Solbrig. Complexity and expressiveness of ShEx for RDF. In *International Conference on Database Theory (ICDT)*, pages 195–211, March 2015.

[35] S. Staworko and P. Wieczorek. Characterizing XML twig queries with examples. In *International Conference on Database Theory (ICDT)*, pages 144–160, March 2015.

[36] M. Veanes. On computational complexity of basic problems of finite tree automata. Technical Report 133, UPMAIL, January 1997.

[37] W3C. RDF validation workshop report: Practical assurances for quality RDF data. September 2013.

[38] W3C. Shape expressions language 2.0, 2017.

[39] V. Weispfenning. The complexity of almost linear diophantine problems. *Journal of Symbolic Computation*, 10(5):395–404, 1990.

[40] H. Zhang, Y. Duan, X. Yuan, and Y. Zhang. ASSG: adaptive structural summary for RDF graph data. In *International Semantic Web Conference (ISWC)*, pages 233–236, October 2014.

A CONSTRUCTING EMBEDDINGS

Theorem 5.4. Testing the existence of embeddings between shape graphs is in P.

Proof. We fix shape graphs G and H and present a iterative procedure for constructing an embedding of G in H. The procedure begins with $R_0 = N_G \times N_H$ and iteratively refines it $R_i = \mathit{Refine}(R_{i-1})$ by removing any pair of nodes with no simulation witness

$$\mathit{Refine}(R) = \{(n,m) \in R \mid \text{there exists a witness } \lambda \text{ of simulation of } n \text{ by } m \text{ w.r.t. } R\}.$$

This process terminates at the earliest iteration k when a fix-point is reached $\mathit{Refine}^*(R_0) = R_k = \mathit{Refine}(R_k)$. The fix-point is in fact the maximal simulation of G in H and naturally, it is an embedding if its domain contains all nodes of G. The core difficulty is in testing the existence of a witness of simulation.

We fix a relation $R \subseteq N_G \times N_H$ and a pair of nodes $(n,m) \in R$. We abstract the problem of existence of a witness of simulation of n by m w.r.t. R as a *flow routing* problem, where we are given a set of sources $V = \mathrm{out}_G(n)$, a set of sinks $U = \mathrm{out}_H(m)$, and a source-to-sink connection table

$$E = \{(v,u) \in V \times U \mid \mathrm{lab}_G(v) = \mathrm{lab}_H(u) \wedge (\mathrm{target}_G(v), \mathrm{target}_H(u)) \in R\},$$

every source $v \in V$ outputs a volume of water between $v.\mathit{min} = \min(\mathrm{occur}_G(v))$ and $v.\mathit{max} = \max(\mathrm{occur}_G(v))$, and every sink $u \in U$ requires an input of at least $u.\mathit{min} = \min(\mathrm{occur}_H(u))$ but no more than $u.\mathit{max} = \max(\mathrm{occur}_H(u))$. The flow routing problem is to find a valid *routing* $\lambda : V \to U$ i.e., a routing such that $(v, f(v)) \in E$ for every source $v \in V$ and there are no deficits or overflows at any source. Formally, given a routing λ we estimate the inflow at a sink u with

$$\mathit{min\text{-}inflow}_\lambda(u) = \textstyle\sum_{\lambda(v)=u} v.\mathit{min},$$
$$\mathit{max\text{-}inflow}_\lambda(u) = \textstyle\sum_{\lambda(v)=u} v.\mathit{max}.$$

A sink u is in *deficit* if $\mathit{min\text{-}inflow}_\lambda(u) < u.\mathit{min}$ and u is in *overflow* if $\mathit{max\text{-}inflow}_\lambda(u) > u.\mathit{max}$. Observe that the conditions 1 and 2 in Definition 2.2 are ensured by the definition of E while the condition 3 follows from lack of deficits and overflows. Also, w.l.o.g. we can assume that in E every source is paired with at least one sink.

In essence, the algorithm for constructing a valid routing (1) starts with an empty routing, (2) assigns it assigns a sink to every source while distributing any overflow by pushing it forth to other sinks, and (3) solves any deficit at a sink by pulling back the input from sources assigned to other sinks.

The main reason why this approach is successful is the use of basic occurrence intervals in shape graphs, which implies that the lower bounds are only 0 and 1 while the upper bounds are 1 and ∞. When constructing the routing λ we need to pay attention to *saturated* sinks that are unable to accept any additional inflow. However, saturated sinks are exactly those u's with $u.\mathit{max} = 1$ and $\mathit{max\text{-}inflow}_\lambda(u) = 1$. Furthermore, w.l.o.g. we can assume that $v.\mathit{max} \leq u.\mathit{max}$ for $(v,u) \in E$, and in particular a source with ∞ upper bound can only be routed to a sink with upper bound ∞. Consequently, any overflow created by the algorithm at a sink u is *singular* i.e., $\mathit{max\text{-}inflow}_\lambda(u) = 2$ and $u.\mathit{max} = 1$.

Given a (partial) routing λ and a source v with no assigned sink, the algorithm assigns to v any admissible sink u_0 i.e., such that $(v,u_0) \in E$. If an overflow is created at u_0, the algorithm attempts to find an acyclic path π from u_0 to *fin* in the *push-forth graph* $G^{\rightarrow}_\lambda = (N, A)$, where the nodes are $N = V \cup U \cup \{\mathit{fin}\}$ and the oriented edges A are (for $v \in V$ and $u \in U$):

- $u \to v$ if $\lambda(v) = u$ and u is saturated; an additional inflow of 1 at sink u must be redirected further and this can be done by redirecting the output of v to another sink.
- $v \to u$ if $(v,u) \in E$ but $\lambda(v) \neq u$; the source v can be routed to u and any additional inflow at u is at most 1.
- $u \to \mathit{fin}$ if u is not saturated; the sink can accept an additional inflow of 1.

Rerouting λ in accordance with a path from u to *fin* gives us a overflow-free routing.

When a total overflow-free routing λ is constructed, the algorithm identifies any sink u_0 with a deficit and tries to solve it by finding an acyclic path π from u_0 to *fin* in the *pull-back graph* $G^{\leftarrow}_\lambda = (N, A)$, where the nodes are $N = V \cup U \cup \{\mathit{fin}\}$, and oriented edges A are (for $v \in V$ and $u \in U$):

- $u \to v$ if $\lambda(v) \neq u$ and $v.\mathit{min} = 1$; rerouting v to u will solve a deficit of 1 at u and may create a overflow at u but only if $u = u_0$ and then the overflow is singular.
- $v \to u$ if $\lambda(v) = u$, $u.\mathit{min} = 1$, and v is the only source such that $\lambda(v) = u$ and $v.\mathit{min} = 1$; rerouting v away from u will create a deficit of 1 at u.
- $v \to \mathit{fin}$ if $\lambda(v).\mathit{min} \neq 1$, $v.\mathit{min} = 1$, and there is $v' \neq v$ such that $\lambda(v') = \lambda(v)$ and $v'.\mathit{min} = 1$; rerouting the source v from the sink $\lambda(v)$ will not create a deficit at $\lambda(v)$.

Rerouting λ in accordance with π renders λ deficit-free at u_0. If the rerouting creates a singular overflow at u_0, the algorithm uses the push-forth graph $G^{\rightarrow}_\lambda$ to find an acyclic path π' from u_0 to *fin* that is deficit-free i.e., with no edge $u \to v$ such that $u.\mathit{min} = v.\mathit{min} = 1$, which guarantees that further rerouting λ in accordance with π' yields a overflow-free routing with one sink node u_0 less in deficit.

Naturally, the algorithm is polynomial because the sizes of the push-forth and pull-back graphs are bounded by the size of E, and all constructed paths are acyclic. □

B CHARACTERIZING EXAMPLES FOR DETERMINISTIC SHAPE GRAPHS

Lemma 5.2. For any $H \in \mathsf{DetShEx}_0^-$, there exists a simple graph $G \in L(H)$ of size polynomial in the size of H such that for any $K \in \mathsf{DetShEx}_0^-$ we have that $G \preccurlyeq K$ implies $H \preccurlyeq K$.

Proof. Let $E_? = (e_1, \ldots, e_m)$ be all edges in H with occurrence interval ?, in an arbitrary but fixed order. The graph G is constructed as follows. The graph has $m + 2$ nodes per type of H: $N_G = N_H \times \{0, 1, \ldots, m + 1\}$, and for simplicity we shall write t^i for $(t, i) \in N_G$. The nodes are used to characterize the occurrence intervals used on the outgoing edges of the type. For an edge $e \in E_H$ such that $t = \mathsf{source}_H(e)$, $s = \mathsf{target}_H(e)$, and $a = \mathsf{lab}_H(e)$, we construct the following edges:

(1) if $\mathsf{occur}_H(e) = 1$, then t^i has one outgoing edge with label a that leads to s^i for $i \in \{0, \ldots, m + 1\}$;
(2) if $\mathsf{occur}_H(e) = ?$ and $e = e_\ell$ i.e., ℓ is the position of e on the list $\mathcal{E}_?$, then t^i has one outgoing edge with label a that leads to s^i for $i \in \{0, \ldots, m + 1\} \setminus \{\ell\}$ and t_ℓ has no outgoing edges with label a;
(3) if $\mathsf{occur}_H(e) = *$, then for $i \in \{0, \ldots, m\}$ the node t^i has no outgoing edge with label a and the node t^{m+1} has an outgoing edge with label a that leads to t_j for every $j \in \{0, \ldots, m + 1\}$.

We now assume that $G \in L(K)$ and construct the following embedding of H in K:

$$R = \{(t, u) \mid u \in \mathsf{Typing}_{G:K}(t^{m+1}),\ \mathsf{in}_H(t) = \emptyset\} \cup \{(t, u) \mid u \in \mathsf{Typing}_{G:K}(t^{m+1}),\ \exists e \in \mathsf{in}_H(t).\ \exists f \in \mathsf{in}_K(u).\ \mathsf{lab}_H(e) = \mathsf{lab}_K(f) \wedge \mathsf{source}_H(e) = s \wedge \mathsf{source}_K(f) \in \mathsf{Typing}_{G:K}(s^{m+1})\},$$

where $\mathsf{in}_H(t) = \{e \in E_H \mid target_H(e) = t\}$. Essentially, R uses types of the nodes t^{m+1} but only essential ones: all types for root nodes (with no incoming edges) and for nodes with incoming edges only those types that are needed to ensure satisfaction of their predecessors. It is easy to see that $\mathsf{dom}(R) = N_H$. To prove that it is indeed an embedding we make several observations. First, we point out that for deterministic shape graphs the concept of witness is redundant since an edge with a given label can be mapped only to an edge with the same label. Consequently, we only need to make sure that R maps nodes in a manner consistent with the labels of the connecting edges and in quantities within the bounds of the occurrence interval.

Now we take any edge $e \in E_H$ and let $t = \mathsf{source}_H(e)$, $s = \mathsf{target}_H(e)$, and $a = \mathsf{lab}_H(e)$. We take any $u \in R(t)$, which implies that t^{m+1} of G has the type u. Since t^{m+1} has an edge labeled a and leading to s^{m+1}, there is an edge $f \in E_K$ with label a from u to some $v \in R(s)$. We can make the following observations about the occurrence interval on f:

- In general $\mathsf{occur}_G(f) \in \{1, ?, *\}$, which is adequate if $\mathsf{occur}_H(e) = 1$;
- If $\mathsf{occur}_H(e) = *$, then we observe that t^{m+1} has $m+2 \geq 2$ outgoing edges labeled with a, all of witch must be embedded in f, and therefore, $\mathsf{occur}_K(f) = *$.
- If $\mathsf{occur}_H(e) = ?$, then we diligently construct a path $e_k, \ldots, e_1, e_0 = e$ such that $\mathsf{occur}_H(e_k) = *$, $\mathsf{occur}_H(e_j) \neq *$ for $j \in \{k-1, \ldots, 1\}$, and $\mathsf{target}_H(e_i) = \mathsf{source}(e_{i-1})$ for $i \in \{k, \ldots, 1\}$, together with two sequences of types $(t_{k+1}, \ldots, t_1)$ and $(u_{k+1}, \ldots, u_1)$ such that $t_i = \mathsf{source}_H(e_{i-1})$ for $i \in \{k + 1, \ldots, 1\}$, $(t_i, u_i) \in R$ for $i \in \{k + 1, \ldots, 1\}$ and there is an edge $f_i \in E_K$ from u_{i+1} to u_i for $i \in \{k + 1, \ldots, 1\}$. We let $t_1 = t$ and $u_1 = u$. Assume we have constructed the path up to e_i with type $t_i = \mathsf{source}_H(e_i)$ and the type u_i such that $(t_i, u_i) \in R$. If $\mathsf{occur}_H(e_i) = *$, we terminate the construction of the path. Otherwise, from construction of R there is a type of u' that is a parent of u_i with an edge with label a' as well as a type $t' \in N_H$ connected to t_i with an edge $e' \in E_H$ such that $(t', u') \in R$. We set $t_{i+1} = t$ and $u_{i+1} = u'$, and $e_{i+1} = e'$. This process terminates thanks to the definition of $\mathsf{DetShEx}_0^-$.
  Now, let the edge e be the ℓ-th element in the ordering $E_?$. we take two corresponding paths in G:
  $$t_{k+1}^{m+1}, t_k^{m+1} \ldots, t_1^{m+1} \quad \text{and} \quad t_{k+1}^{m+1}, t_k^{\ell} \ldots, t_1^{\ell};$$
  both following the edges $e_k, \ldots, e_1$. With a simple inductive proof we show that both t_1^ℓ and t_1^{m+1} have the same type u_1, and since t_1^ℓ does not have any outgoing edge labeled with a, the interval $\mathsf{occur}_K(f)$ must be either ? or $*$.

This concludes the proof. □