HAL
open science

Complexity and Expressiveness of ShEx for RDF

Slawomir Staworko, Iovka Boneva, Jose Emilio Labra Gayo, Samuel Hym, Eric Gordon Prud'Hommeaux, Harold Solbrig

To cite this version:

Slawomir Staworko, Iovka Boneva, Jose Emilio Labra Gayo, Samuel Hym, Eric Gordon Prud'Hommeaux, et al.. Complexity and Expressiveness of ShEx for RDF. 18th International Conference on Database Theory (ICDT 2015), Mar 2015, Brussels, Belgium. 10.4230/LIPIcs.ICDT.2015.195 . hal-01218552

HAL Id: hal-01218552

https://hal.science/hal-01218552

Submitted on 22 Oct 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Complexity and Expressiveness of ShEx for RDF

Sławek Staworko*[1], Iovka Boneva[1], Jose E. Labra Gayo[2], Samuel Hym[3], Eric G. Prud'hommeaux[4], and Harold Solbrig[5]

1 LINKS, INRIA & CNRS, University of Lille, France
2 University of Oviedo, Spain
3 LIFL, University of Lille & CNRS, France
4 W3C, Stata Center, MIT
5 Mayo Clinic College of Medicine, Rochester, MN, USA

Abstract

We study the expressiveness and complexity of Shape Expression Schema (ShEx), a novel schema formalism for RDF currently under development by W3C. ShEx assigns types to the nodes of an RDF graph and allows to constrain the admissible neighborhoods of nodes of a given type with regular bag expressions (RBEs). We formalize and investigate two alternative semantics, multi- and single-type, depending on whether or not a node may have more than one type. We study the expressive power of ShEx and study the complexity of the validation problem. We show that the single-type semantics is strictly more expressive than the multi-type semantics, single-type validation is generally intractable and multi-type validation is feasible for a small (yet practical) subclass of RBEs. To curb the high computational complexity of validation, we propose a natural notion of determinism and show that multi-type validation for the class of deterministic schemas using single-occurrence regular bag expressions (SORBEs) is tractable.

1998 ACM Subject Classification H.2.2 Schema and subschema.

Keywords and phrases RDF, Schema, Graph topology, Validation, Expressiveness, Complexity.

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Schemas have a number of important functions in databases. They describe the structure of the database, and its knowledge is essential to any user trying to formulate and execute a query or an update over a database instance. Typically, schemas allow for efficient algorithms for validating the conformance of a given database instance. Schemas also capture the intended meaning of the data stored in database instances and are important for static analysis tasks such as query optimization.

Relational and XML databases have a number of well-established and widely accepted schema formalisms e.g., the SQL Data Definition Language for relational databases and W3C XML Schema and RELAX NG for XML databases. One of the reasons why the RDF data model at its conception has been schema-free was to promote its use and ensure its wide-spread adoption. Indeed, a number of existing RDF applications, such as the linked open data initiative[1], could not have had the same success if the published data had to comply with a rigid schema. However, RDF is slowly but surely becoming an independent database model [1], with applications that were previously considered only in the context of

* Contact author: slawomir.staworko@inria.fr
[1] http://linkeddata.org/

© Sławek Staworko and Iovka Boneva and Jose E. Labra Gayo and Samuel Hym and Eric G. Prud'hommeaux and Harold Solbrig;
licensed under Creative Commons License CC-BY
Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

relational and XML databases, for instance data exchange [16, 2]. Such classical applications often rely on the conformance of the data with a set of constraints. Not only has the 'declarative definition of the structure of a graph for validation and description' been clearly identified [39] but also we currently see an emergence of approaches to address this need: apart from the existing but somewhat inadequate RDF Schemas (RDFS) [9], we have OSLC Resource Shapes (ResSh) [31], integrity constraints expressed in OWL [34], and SPARQL queries generated with SPIN templates [6]. In this paper, we investigate Shape Expression Schemas (ShEx) [29, 24], a novel schema formalism for RDF currently under development by W3C [40].

A ShEx allows to define a set of types that impose structural constraints on nodes and their immediate neighborhood. Figure 1 presents a simple example of a Shape Expression Schema for an RDF database storing bug reports.

```
<BugReport> {
  descr xsd:string,
  reportedBy @<User>,
  reportedOn xsd:dateTime,
  (reproducedBy @<Employee>,
   reproducedOn xsd:dateTime)?,
  related @<BugReport>*
}

<User> {
  name xsd:string,
  email xsd:string?
}

<Employee> {
  (name xsd:string |
   (first-name xsd:string,
    last-name xsd:string)),
  email xsd:string,
}
```

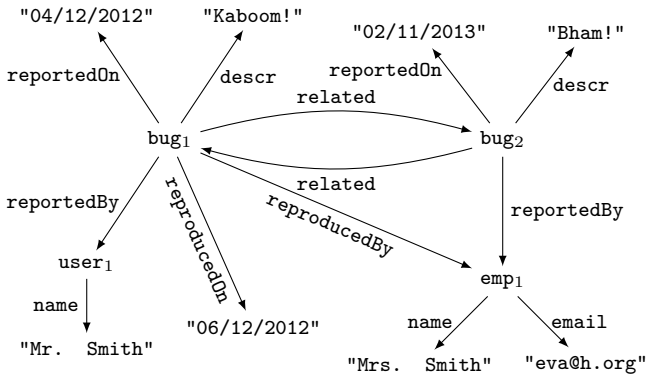

Figure 1 An example of a Shape Expression Schema and a valid RDF graph.

Essentially, the schema above says that a bug report has a description, a user who reported it, and on what date. Optionally, a bug report may also have an employee who successfully reproduced the bug, and on what date. Finally, a bug report can have a number of related bug reports. For every user we wish to store his/her name and optionally email. For an employee we wish to store his/her name, either as one string or split into the first and last name, and email address.

A Shape Expression Schema defines a set of types to be associated to graph nodes. Each type defines the admissible collection of outgoing edges and the types of the nodes they lead to. Naturally, such a schema bears a strong resemblance to RELAX NG and DTDs which also use regular expressions to describe the allowed collections of children of an XML node. The most important difference comes from the fact that in XML, the children of a node are ordered, and the regular expressions in DTDs and RELAX NG schemas define admissible sequences of children, whereas for RDF graphs, no order on the neighborhood of a given node can be assumed. As the regular expressions used in ShEx define bags (multisets) of symbols rather than sequences, we call them *regular bag expressions* (RBEs).

The semantics of Shape Expression Schemas is quite natural. An RDF graph is valid if it is possible to assign types to the nodes of the graph in a manner that satisfies all shape expressions of the schema. A natural question arises: can a node be assigned more than

one type? In most applications the *multi-type semantics*, which permits assigning multiple types to a node, seems to be more natural. For instance, the RDF graph in Figure 1 requires assigning both the type `User` and the type `Employee` to the node $\texttt{emp}_1$ because $\texttt{emp}_1$ has reported $\texttt{bug}_2$ and reproduced $\texttt{bug}_1$. However, there are applications where the single-type semantics may be more suitable e.g., when modeling graph transformations that visit every node exactly once.

The requirement of exactly one type per node makes the single-type semantics more restrictive, and therefore, capable of defining more refined families of graphs than the multi-type semantics. We show that the single-type semantics is in fact strictly more powerful than the multi-type semantics. We also show both semantics are closed under intersection but neither is closed under union or complement. We then compare the expressive power of ShEx with two standard yardstick logics for graphs: first-order logic ($\mathsf{FO_G}$) and existential monadic second-order logic ($\exists\mathsf{MSO_G}$). ShEx are not comparable with $\mathsf{FO_G}$ but if the RBEs use the Kleene closure on symbols only (e.g. a^*), then ShEx using such expressions are captured by $\exists\mathsf{MSO_G}$. Finally, in our study we compare the expressive power of ShEx with graph grammars, which are generally incomparable, and graph acceptors/automata, which are typically less expressive.

Next, we study the complexity of the validation problem i.e., checking if a given graph has a valid typing w.r.t. the given schema. Naturally, this problem comes in two flavors, depending on the chosen semantics, and we show significant computational differences between them. While validation for both semantics is generally intractable, the multi-type semantics admits tractable validation for a subclass $\mathsf{RBE_0}$ of disjunction-free expressions that use the Kleene closure on symbols only. This fragment of RBEs is quite practical as it can, for instance, very easily capture the topology of RDF graphs obtained by exported relational databases in virtually any of the proposed approaches for this task (for survey, see [33]). More interestingly, however, we show that the complexity of multi-type validation for ShEx using a class $\mathcal{C}$ of RBEs is closely related (Turing reducible) to the complexity of the satisfiability problem for $\mathcal{C}$ with intersection. We show that in general this problem is NP-complete, which stands in contrast with its analogue for regular word expressions known to be PSPACE-complete [23].

To lower the complexity of validation, we introduce the notion of determinism. Essentially, determinism requires that every shape expression uses at most one type with every label. The shape expressions in Figure 1 are deterministic but the following shape expression is not.

```
<BugReport> {
   descr xsd:string,
   (reportedBy @<User> | reportedBy @<Employee>),
   reportedOn xsd:dateTime,
   (reproducedBy @<Employee>,
    reproducedOn xsd:dateTime)?
   related @<BugReport>*
}
```

This shape expression is not deterministic because `reportedBy` is used with two types: `User` and `Employee`. For deterministic shape expression schemas, we are able to relate the complexity of multi-type validation to the problem of checking membership of a bag of symbols to the language of RBEs. While this problem is known to be NP-complete [22], it is generally simpler than the satisfiability problem, and a number of tractable subclasses has already been identified [7, 22]. All known tractable classes of RBEs require the expressions to be single-occurrence i.e., every symbol of the alphabet is used at most once in an RBE. In the present paper, we show that the full class of *single-occurrence regular bag expressions* (SORBE) has in fact tractable membership. Finally, we consider the problem of validating

only a fragment of a graph with preassigned types for its root nodes and argue that for deterministic ShEx using SORBEs, multi-type validation can be performed efficiently, and show that single-type validation can be performed with a single pass over the graph.

While checking that the data values satisfy constraints is an important part of database validation, in our study we focus only on the core capacity of Shape Expression Schemas to define the graph topology, and therefore, ignore data values. While the exact array of types of value constraints of ShEx is yet to be elaborated, our results identify important computational obstacles that arise from the topology shaping properties of ShEx alone, and furthermore, we present tractable algorithms that can serve as a basis for RDF validation with a number of data value constraints. Our methodology can be compared to using automata to model schema languages for XML databases: while virtually all schema languages for XML allow to define constraints on data values, from simple domain checks in DTDs to key constraints in XML Schema, the results on automata often translate directly to results for schema languages for XML.

The main contributions of the present paper are:

1. We formalize two alternative semantics for ShEx, multi-type and single-type, depending on whether or not a node may have more than one type.
2. We study the expressive power and basic properties of ShEx.
3. We provide a comprehensive understanding of the complexity of validation and show a very close relationship to the complexity of the problem of satisfiability of regular bag expressions with intersection. We show that multi-type validation is tractable for a practical subclass of RBE_0.
4. We propose a notion of determinism for ShEx that allows to curb the complexity of validation and (Turing) reduces validation to checking membership of a bag to the language of RBE. Additionally, we show that single-occurrence regular bag expressions (SORBE) enjoy a tractable membership problem which makes them an attractive candidate for use in deterministic shape expression schemas.

Related work. A number of approaches for validation of RDF has been previously proposed. RDF Schema (RDFS) in essence allows to define a light ontology consisting of types of objects (classes), inclusion dependencies between types (a hierarchy), and specification of the domain and the range of the graph edges of a given label. However, the W3C recommendation does not fix a semantics but only suggest two possible usages: 1) as an ontology allowing to infer types of RDF objects and 2) as a constraint language. It should be noted that only the first use is formalized [5] and it is a common belief that, despite its name, RDF Schema is a basic ontology language rather than a schema language. We point out that the constraints definable with RDFS can be easily captured with ShEx but the converse is not the case. OSLC Resource Shapes (ResSh) [31] essentially extend RDFS by allowing to specify cardinality constraints ?, $*$, $+$, and 1 on types which renders it equivalent to ShEx using single-occurrence RBE_0.

While OWL [14] is an ontology language geared towards inference, it can enforce certain constraints by capturing constraint violations with rules that trigger an inconsistency. However, the class of constraints that can be enforced by OWL is limited due to the fact that OWL is interpreted under open world assumption (OWA) and without unique name assumption (UNA). Consequently, a modified semantics for OWL has been proposed [34] that uses OWA when inferring new facts while employing closed word assumption (CWA) and UNA when enforcing constraints. Since OWL with its standard semantics is widely accepted, concerns have been raised [31] about the potential confusion arising from the mixed

semantics. Such formalisms is, however, very powerful and expressive, easily captures a rich fragment of ShEx (equivalent to $\exists\mathsf{MSO}_\mathsf{G}$), but its computational properties are yet to be characterized. While [36] outlines a method of translating OWL constraints into SPARQL queries, the size of the resulting SPARQL queries seems to depend on the size of the OWL constrains. Currently, we do not know if it is reasonable to assume the size of schema for RDF to be fixed as it may involve large vocabularies of types used by ontologies. This gives a PSPACE upper bound while ShEx enjoys much lower (combined) complexity. Similar criticism applies to other solutions based on SPARQL queries, such as SPIN [6], while they are very powerful and expressive, their use may require significant computational resources.

Finally, we point out an important difference in semantics: while we investigate the existence and construction of a valid typing, all approaches above assume the typing to be given (with `rdf:type` edges) and only verify that the typing is valid. Our approach is more general, we show how to verify the validity of a given typing and that it is the main source of complexity. In particular, we propose a method constructing a maximal (multi-type) valid typing and a method extending the given (possibly invalid) typing to one that is valid.

Organization. In Section 2 we present basic notions. In Section 3 we introduce Shape Expression Schemas (ShEx) and define the single- and multi-type semantics. In Section 4 we study the complexity of the validation problem for ShEx. In Section 5 we introduce a natural notion of determinism for ShEx and identify a rich class of single-occurrence RBEs that together render multi-type validation tractable. In Section 6 we analyze the expressive power of ShEx. Finally, we conclude and discuss related and future work in Section 7. Because of space restriction we omit the proofs: they can be found in the technical report [8].

2 Preliminaries

Because we wish to investigate only the capacity of ShEx to shape the graph topology, we model RDF databases with standard graphs whose edges are labeled by elements of a finite set. In [8] we show how a more general model can be employed without affecting the results.

2.1 Graphs

We assume a finite set Σ of edge labels. An *edge-labeled graph* (or simply a *graph*) is a pair $G = (V, E)$, where V is a finite set of nodes and $E \subseteq V \times \Sigma \times V$ is the set of edges. In Figure 2 we present a number of examples of edge-labeled graphs.

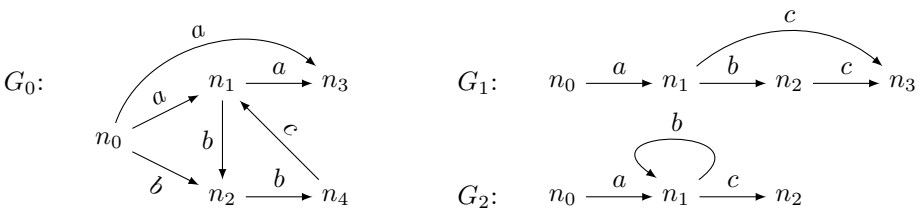

Figure 2 Edge-labeled oriented graphs.

In our approach, we shape the topology of a graph based on the immediate outbound neighborhood of nodes. The *labeled outbound neighbourhood* of the node n in the graph $G = (V, E)$ is essentially the set of edges outgoing from n, and is defined as $\mathit{out\text{-}lab\text{-}node}_G(n) = \{(a, m) \in \Sigma \times V \mid (n, a, m) \in E\}$. For instance, $\mathit{out\text{-}lab\text{-}node}_{G_0}(n_0) = \{(a, n_1), (b, n_2), (a, n_3)\}$. On occasions, we use only the collection of outgoing labels, ignoring their target nodes. Note, however, that this collection needs not be representable as neither a set nor by a list of labels

because a node may have multiple outgoing edges with the same label and its neighborhood is not ordered. Take for instance node n_0 in the graph G_0: it has two outgoing a-edges and one outgoing b-edge. Consequently, we employ bags, also known as multisets, which essentially specify the number of occurrences of every symbol.

2.2 Bags of symbols

Let Δ be a finite set of symbols (which is not necessarily Σ). 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 $a \in \Delta$. We write $a \in w$ as a short for $w(a) \neq 0$.

We present bags 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 $w_0(c) = 2$. Now, for a given graph $G = (V, E)$ and its node $n \in V$, we define the *bag of outbound labels* of n in G as the bag $\mathit{out\text{-}lab}_G(n) = \{\!|a \mid (n, a, m) \in E|\!\}$. For instance, for the graph G_0 in Figure 2 and the node n_0 we have $\mathit{out\text{-}lab}_{G_0}(n_0) = \{\!|a, a, b|\!\}$.

The *bag union* $w_1 \uplus w_2$ of two bags w_1 and w_2 is defined as $[w_1 \uplus w_2](a) = w_1(a) + w_2(a)$ for all $a \in \Delta$. For instance, $\{\!|a, c, c|\!\} \uplus \{\!|a, b|\!\} = \{\!|a, a, b, c, c|\!\}$. A *bag language* is a set of bags. The bag union of two languages L_1 and L_2 is the language $L_1 \uplus L_2 = \{w_1 \uplus w_2 \mid w_1 \in L_1, w_2 \in L_2\}$. Also, for a given bag language L, we define $L^0 = \{\varepsilon\}$ and $L^i = L \uplus L^{i-1}$ for $i \geq 0$.

2.3 Regular bag expressions

A number of XML schema languages, including DTD, XML Schema, and RelaxNG, uses regular expressions to define the content model (local structure) of types. The popularity of using regular expressions (for words) in validation comes from the fact that they are easy to grasp and use by a wide range of potential users. Because in the context of RDF graphs the nodes are not ordered, we employ regular expressions for bags, which replace the ordered concatenation operator with its unordered version $\|$, and implicitly the Kleene star by the unordered Kleene star. This family of expressions have been successfully employed to model XML with unordered and mixed content model [7, 12].

A *regular bag expression* (RBE) defines bags by using disjunction "$|$", unordered concatenation "$\|$", and unordered Kleene star "$*$". Formally, RBEs over Δ are defined with the following grammar $E ::= \epsilon \mid a \mid E^* \mid (E\text{“}|\text{”}E) \mid (E\text{“}\|\text{”}E)$, where $a \in \Delta$. Their semantics is defined as follows: $L(\epsilon) = \{\varepsilon\}$, $L(a) = \{\{\!|a|\!\}\}$, $L(E_1 \mid E_2) = L(E_1) \cup L(E_2)$, $L(E_1 \| E_2) = L(E_1) \uplus L(E_2)$, and $L(E^*) = \bigcup_{i \geq 0} L(E)^i$. We use two standard macros: $E^? := (\epsilon \mid E)$ and $E^+ := (E \| E^*)$. We also use *intervals* on symbols $a^{[n;m]}$, where $n \in \mathbb{N}$ and $m \in \mathbb{N} \cup \{\infty\}$, with the natural semantics: $L(a^{[n;m]}) = \bigcup_{n \leq i \leq m} L(a)^i$.

We use different syntactic restrictions on RBEs, by indicating which of the syntactic ingredients can be used, among the following: a^M means that multiplicities among $\{1, ?, *, +\}$ can be used only on symbols; a^I means that arbitrary intervals on symbols can be used; $\|$, $|$, and $*$ mean that the corresponding operator can be used. For instance, $\mathsf{RBE}(a^M, \|, |)$ is the class of RBE that allows the use of the Kleene star operator $*$ on symbols only; $\mathsf{RBE}(a, \|, |, *)$ is the class of all RBE. By $\mathsf{RBE_0}$ we denote the simplest class of $\mathsf{RBE}(a^I, \|)$ that we believe to be of practical relevance. Finally, by $\mathsf{RBE_1}$ we denote the RBEs of the form $(a_{1,1} \mid \cdots \mid a_{1,k_1}) \| \cdots \| (a_{n,1} \mid \ldots \mid a_{n,k_n})$, with $a_{i,j} \in \Delta$. In the sequel, we also use RBE to denote the family of bag languages definable with regular bag expressions, and it should be clear from the context whether "RBE" stands for the class of expressions, or for the class of languages.

A number of important facts are known about RBE: it is closed under intersection, union, and complement [26], testing membership $w \in L(E)$ is NP-complete [22], and so is testing the emptiness of $L(E_1) \cap L(E_2)$ [12]. Also, when a bag of symbols is viewed as vector of natural numbers (obtained by fixing some total order on Δ), RBE is equivalent to the class of semilinear sets and the class of vectors definable with Presburger arithmetic [18, 28].

3 Shape Expression Schemas

In this section we formally introduce shape expressions schemas and propose two semantics that we study in the remainder of the paper. We assume a finite set of edge labels Σ and a finite set of types Γ. A *shape expression* is an RBE over $\Sigma \times \Gamma$. In the sequel we write $(a, t) \in \Sigma \times \Gamma$ simply as $a :: t$. A *shape expression schema* (ShEx), or simply *schema*, is a tuple $S = (\Sigma, \Gamma, \delta)$, where Σ is a finite set of edge labels, Γ is a finite set of types, and δ is a *type definition* function that maps elements of Γ to bag languages over $\Sigma \times \Gamma$. We only use shape expressions for defining bag languages of δ. Typically, we present a ShEx as a collection of rules of the form $t \to E$ to indicate that $\delta(t) = L(E)$, where $t \in \Gamma$ and E is a shape expression over Σ and Γ (naturally, no two rules shall have the same left-hand side). If for some type t a rule is missing, the default rule is $t \to \epsilon$. For a class of RBEs $\mathcal{C}$ by ShEx($\mathcal{C}$) we denote the class of shape expression schemas using only shape expressions in $\mathcal{C}$. Two example schemas follow:

$$\begin{aligned} S_0 : t_0 &\to a :: t_1 \parallel b :: t_2 \\ t_1 &\to (a :: t_1 \mid b :: t_2)^* \\ t_2 &\to b :: t_2 \mid c :: t_1 \end{aligned} \qquad \begin{aligned} S_1 : t_0 &\to a :: t_1 \qquad t_3 \to \epsilon \\ t_1 &\to b :: t_2 \parallel c :: t_3 \\ t_2 &\to (b :: t_2)^? \parallel c :: t_3 \end{aligned}$$

The semantics of ShEx is natural: a graph is valid if it is possible to assign types to the nodes of the graph in a manner that satisfies the type definitions of the schema. Two variants of semantics can be envisioned depending on whether or not more than one type can be assigned to a node.

3.1 Single-type semantics

We fix a graph $G = (V, E)$ and a schema $S = (\Sigma, \Gamma, \delta)$. A *single-type typing* (or simply an *s-typing*) of G w.r.t. S is a function $\lambda : V \to \Gamma$ that associates with every node $n \in V$ its type $\lambda(n)$. An example of an s-typing of G_0 w.r.t. S_0 is

$$\lambda_0(n_0) = t_0, \qquad \lambda_0(n_1) = t_1, \qquad \lambda_0(n_2) = t_2, \qquad \lambda_0(n_3) = t_1, \qquad \lambda_0(n_4) = t_2.$$

Next, we identify the conditions that an s-typing needs to satisfy. Given a typing λ and a node $n \in V$ we define the *labeled and typed out-neighborhood* of n w.r.t. λ as bag over $\Sigma \times \Gamma$ as $\textit{out-lab-type}^{\lambda}_{G}(n) = \{\!\!\{a :: \lambda(m) \mid (n, a, m) \in E\}\!\!\}$. For instance, for the graph G_0 (Figure 2) and the typing λ_0 we have $\textit{out-lab-type}^{\lambda_0}_{G_0}(n_1) = \{\!\!\{a :: t_1, b :: t_2\}\!\!\}$ and $\textit{out-lab-type}^{\lambda_0}_{G_0}(n_4) = \{\!\!\{c :: t_1\}\!\!\}$.

Now, λ is a *valid* s-typing of S on G if and only if every node satisfies the type definition of its associated type i.e., for every $n \in V$, $\textit{out-lab-type}^{\lambda}_{G}(n) \in \delta(\lambda(n))$. By $L_s(S)$ we denote the set of all graphs that have a valid s-typing w.r.t. the shape expression schema S. For a class $\mathcal{C}$ of bag languages by $\mathsf{ShEx}_s(\mathcal{C})$ we denote the class of graph languages definable under the single-type semantics with shape expression schemas using shape expressions from $\mathcal{C}$ only. Naturally, λ_0 is a valid typing of G_0 w.r.t. S_0. G_1 also has a valid s-typing of S_1:

$$\lambda_1(n_0) = t_0, \qquad \lambda_1(n_1) = t_1, \qquad \lambda_1(n_2) = t_2, \qquad \lambda_1(n_3) = t_3.$$

G_2, however, does not have a valid s-typing w.r.t. S_1.

3.2 Multi-type semantics

Again, we assume a fixed graph $G = (V, E)$ and a fixed schema $S = (\Sigma, \Gamma, \delta)$. A *multi-type typing* (or simply an *m-typing*) of G w.r.t. S is a function $\lambda : V \to 2^\Gamma$ that associates with every node of G a set of types. For instance, an m-typing of G_2 w.r.t. S_1 is

$$\lambda_2(n_0) = \{t_0\}, \qquad \lambda_2(n_1) = \{t_1, t_2\}, \qquad \lambda_2(n_2) = \{t_3\}.$$

The labeled and typed out-neighborhood of a node is defined in the same way but note that this time it is a bag over $\Sigma \times 2^\Gamma$. For instance, $\mathit{out\text{-}lab\text{-}type}^{\lambda_2}_{G_2}(n_1) = \{\!\!\{b :: \{t_1, t_2\}, c :: \{t_3\}\}\!\!\}$.

Now, a flattening of a bag over $\Sigma \times 2^\Gamma$ is any bag over $\Sigma \times \Gamma$ obtained by choosing one type from every occurrence of every set. For instance, $\mathit{out\text{-}lab\text{-}type}^{\lambda_2}_{G_2}(n_1)$ has two flattenings: $\{\!\!\{b :: t_1, c :: t_3\}\!\!\}$ and $\{\!\!\{b :: t_2, c :: t_3\}\!\!\}$. Formally, a *flattening* of a bag w over $\Sigma \times 2^\Gamma$ is any bag in the language of the following RBE_1 expression $\mathit{Flatten}(w) = \|_{a::T \Subset w}(|_{t \in T}\ a :: t)$, where $a :: T \Subset w$ indicates that the symbol $a :: T$ is to be considered $w(a :: T)$ times. For instance,

$$\mathit{Flatten}(\{\!\!\{a :: \{t_0, t_1\}, a :: \{t_0, t_1\}, b :: t_1\}\!\!\}) = (a :: t_0 \mid a :: t_1) \parallel (a :: t_0 \mid a :: t_1) \parallel (b :: t_1).$$

By $\mathit{Out\text{-}lab\text{-}type}^{\lambda}_{G}(n)$ we denote the set of all flattenings of $\mathit{out\text{-}lab\text{-}type}^{\lambda}_{G}(n)$ i.e.

$$\mathit{Out\text{-}lab\text{-}type}^{\lambda}_{G}(n) = L(\mathit{Flatten}(\mathit{out\text{-}lab\text{-}type}^{\lambda}_{G}(n))).$$

For instance, $\mathit{Out\text{-}lab\text{-}type}^{\lambda_2}_{G_2}(n_1) = \{\{\!\!\{b :: t_1, c :: t_3\}\!\!\}, \{\!\!\{b :: t_2, c :: t_3\}\!\!\}\}$. Note that while $\mathit{Flatten}(\mathit{out\text{-}lab\text{-}type}^{\lambda}_{G}(n))$ is an expression of size polynomial in the size of G and S, the cardinality of the set $\mathit{Out\text{-}lab\text{-}type}^{\lambda}_{G}(n)$ may be exponential in the size of G and S. Now, λ is a *valid* m-typing of G w.r.t. S if and only if:

1. it assigns at least one type to every node, $\lambda(n) \neq \emptyset$ for $n \in V$,
2. every node satisfies the type definition of every type assigned to the node i.e., for every $n \in V$ and every $t \in \lambda(n)$, $\mathit{Out\text{-}lab\text{-}type}^{\lambda}_{G}(n) \cap \delta(t) \neq \emptyset$.

For instance, λ_2 is a valid multi-type typing of G_2 w.r.t. S_1. A *semi-valid* m-typing is a m-typing that satisfies the second condition but might violate the first one. By $L_{\mathrm{m}}(S)$ we denote the set of all graphs that have a valid m-typing w.r.t. S. For a class $\mathcal{C}$ of bag languages by $\mathsf{ShEx}_{\mathrm{m}}(\mathcal{C})$ we denote the class of graph languages definable under the multi-type semantics with shape expressions schemas using shape expressions in $\mathcal{C}$ only.

4 Validation

In this section we consider the problem of *validation*: checking whether a given graph has a valid typing w.r.t. a given ShEx. This problem has two parameters: 1) the kind of typing, either single-type or multi-type and 2) the class of regular bag expressions used for type definitions in the schema.

We first point out that the complexity of single-type validation for ShEx(RBE) is NP-complete. The NP upper bound follows from the fact that the membership problem for RBE is in NP. The upper bound is shown with a reduction from 3-colorability of graphs.

▶ **Theorem 1.** *Single-type validation for* ShEx(RBE) *is NP-complete.*

Proof (sketch) The following schema which, under the single-type semantics, defines the set of graphs with homomorphism into K_3 i.e., all 3-colorable graphs:

$$t_r \to _ :: t_b^* \parallel _ :: t_g^* \qquad t_g \to _ :: t_r^* \parallel _ :: t_b^* \qquad t_b \to _ :: t_g^* \parallel _ :: t_r^* \qquad \square$$

For the remaining of this section, we focus on multi-type validation and return briefly to the single-type semantics in the next section.

4.1 Semi-lattice of m-typings

We begin by presenting a downward refinement method that allows to construct a valid m-typing. Take a graph $G = (V, E)$ and a ShEx S, and let $mTyping(G, S)$ be the set of all valid m-typings of the graph G w.r.t. the schema S. $mTyping(G, S)$ is a semi-lattice with the meet operation $\sqcup$ and the (induced) partial order defined as follows:

$$(\lambda_1 \sqcup \lambda_2)(n) = \lambda_1(n) \cup \lambda_2(n) \quad \text{for } n \in V, \qquad \text{and} \qquad \lambda_1 \sqsubseteq \lambda_2 \quad \text{iff} \quad \forall n \in V.\ \lambda_1(n) \subseteq \lambda_2(n).$$

The refinement method works as follows. We begin with a typing that assigns to every node the set of all types, $\lambda^\circ(n) = \Gamma$ for all $n \in V$, and then we iteratively remove the types that are not satisfied. Every iteration is an application of the *one-step refinement operator* on m-typings defined as follows (with $n \in V$):

$$[\mathit{Refine}(\lambda)](n) = \{t \in \lambda(n) \mid \mathit{Out\text{-}lab\text{-}type}_G^\lambda(n) \cap \delta(t) \neq \emptyset\}.$$

Clearly, $\mathit{Refine}(\lambda) \sqsubseteq \lambda$, and therefore, the fix-point $\mathit{Refine}^*(\lambda)$ is well-defined. We claim that the procedure outlined above indeed constructs the maximal valid m-typing if one exists.

▶ **Lemma 2.** *For any $\lambda \in mTyping(G, S)$, $\lambda \sqsubseteq \mathit{Refine}^*(\lambda^\circ)$, where $\lambda^\circ(n) = \Gamma$ for all $n \in V$.*

In particular, G satisfies S if and only if $\mathit{Refine}^*(\lambda^\circ)$ is valid, and then, $\mathit{Refine}^*(\lambda^\circ)$ is the $\sqsubseteq$-maximal valid m-typing of G on S. We point out that there does not necessarily exist a unique $\sqsubseteq$-minimal valid m-typing.

4.2 Complexity of Validation

Using the above refinement procedure, we show that multi-type validation is NP-complete for arbitrary regular bag expressions and later identify a tractable fragment.

In essence, performing the refinement procedure requires testing the nonemptyness of the intersection $\mathit{Out\text{-}lab\text{-}type}_G^\lambda(n) \cap \delta(t)$. Recall that $\mathit{Out\text{-}lab\text{-}type}_G^\lambda(n)$ is defined by an RBE_1 expression, i.e. an expression of the form $(a_{1,1} \mid \cdots \mid a_{1,k_1}) \parallel \cdots \parallel (a_{n,1} \mid \ldots \mid a_{n,k_n})$. Therefore, for a class of RBEs $\mathcal{C}$ used we identify the following decision problem:

$$\mathsf{INTER}_1(\mathcal{C}) = \{(E_0, E) \in \mathsf{RBE}_1 \times \mathcal{C} \mid L(E_0) \cap L(E) \neq \emptyset\}.$$

Tractability of INTER_1 is a necessary and sufficient condition for tractability of multi-type validation for $\mathsf{ShEx}(\mathcal{C})$. On the one hand, we show that for any class $\mathcal{C}$ of RBEs there exists a polynomial-time reduction from $\mathsf{INTER}_1(\mathcal{C})$ to validation for $\mathsf{ShEx}_\mathrm{m}(\mathcal{C})$. On the other hand, the refinement procedure performs a polynomial number of INTER_1 tests (with polynomially-sized inputs). This observation allows us to characterize precisely the complexity of multi-type validation for ShEx(RBE): the lower bound follows from an existing complexity results [12] on testing emptiness of RBEs with intersection and the upper bound follows from a more general result we prove, namely testing satisfiability of RBEs with intersection can be reduced to finding integer solutions to a system of linear equations known to be in NP [27].

▶ **Theorem 3.** *Multi-type validation for* ShEx(RBE) *is NP-complete.*

4.3 The tractable subclass RBE_0

When ShEx use only expressions in $\mathsf{RBE}_0 = \mathsf{RBE}(a^I, \|)$, the multi-type validation is tractable, which we show by providing a polynomial algorithm for $\mathsf{INTER}_1(\mathsf{RBE}_0)$. We point out that ShEx using RBE_0 are capable, for instance, of capturing the topology of RDF graphs obtained from exporting relational databases to RDF [33].

We reduce $\mathsf{INTER}_1(\mathsf{RBE}_0)$ to the circulation problem in flow networks. Recall that a flow network is a directed graph with arcs having additionally assigned the amount of flow they require (minimum flow) and the amount of flow they can accept (maximum flow). The *circulation problem* is to find a valid flow i.e., an assignment of flow values to arcs of the flow network so that the flow constraints of every arc are satisfied and at every node the sum of incoming flow is equal to the sum of outgoing flow. This problem has been well-studied and a number of efficient polynomial algorithms exist (cf. [19]).

We illustrate the reduction on the example of $E_0 = (a \mid c) \parallel (b \mid c)$ and $E = a^? \parallel b^* \parallel c$. The corresponding network $N_{E_0,E}$ is presented in Figure 3 with the maximum flow (the minimum flow) of an arc is indicated above (below respectively). An example of a valid

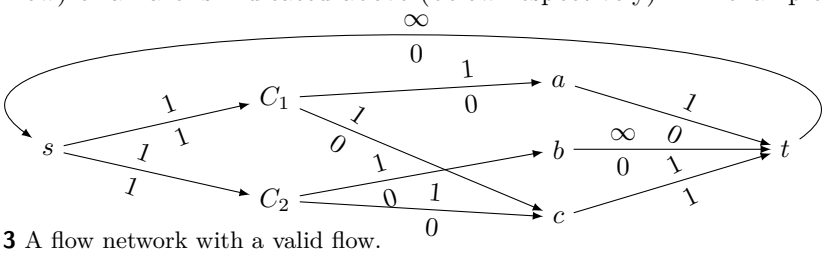

Figure 3 A flow network with a valid flow.

flow f in $N_{E_0,E}$, that corresponds to $\{\!|a, c|\!\} \in L(E_0) \cap L(E)$, is $f(s, C_1) = 1$, $f(C_1, a) = 1$, $f(C_2, b) = 0$, $f(s, C_2) = 1$, $f(C_1, c) = 0$, $f(C_2, c) = 1$, $f(a, t) = 1$, $f(b, t) = 0$, $f(c, t) = 1$, and $f(t, s) = 2$.

▶ **Theorem 4.** *Multi-type validation for* $\mathsf{ShEx}(\mathsf{RBE}_0)$ *is in PTIME.*

5 Determinism

Determinism is a classical tool for decreasing the complexity of validation [20], which we explore next. We propose a suitable and natural notion of determinism for ShEx and show that multi-type validation for deterministic ShEx is not harder than membership of a bag to the language of an RBE. This allows us to identify a large and practical class of single-occurrence regular bag expressions (SORBE) that render validation tractable (Section 5.2). We then investigate the problem of partial validation, where the conformance of only a fragment of the input graph is to be checked (Section 5.3), and which we believe to be an important practical use case of ShEx. We present an optimal algorithm for partial validation which is tractable for classes of deterministic ShEx with tractable membership, for both multi-type and single-type semantics.

5.1 Deterministic Shape Expressions

Essentially, the idea of determinism for ShEx $S = (\Gamma, \delta)$ is that, knowing the type t of a node $n \in V$ and the label a of an outgoing edge $(n, a, m) \in E$ we should know the type t' that must be satisfied by m if n is to satisfy the type t.

Formally, a shape expression E is *deterministic* if every label $a \in \Sigma$ is used with at most one type $t \in \Gamma$ in E. For instance, $E_1 = a :: t_1 \parallel b :: t_2^* \parallel a :: t_1 \parallel c :: t_2$ is deterministic but

$E_2 = a :: t_1 \parallel b :: t_2^* \parallel a :: t_3 \parallel c :: t_2$ is not because the symbol a is used with two different types t_1 and t_3. Now, a shape expression schema $S = (\Sigma, \Gamma, \delta)$ is *deterministic* if it uses only deterministic shape expressions, and then, by $\delta(t, a)$ we denote the unique type used with the symbol a in the expression used to define $\delta(t)$ (if a is used in this expression).

Recall that the tractability of the refinement method for multi-type validation presented in Section 4.1 depends on the tractability of testing that $\mathit{Out\text{-}lab\text{-}type}_G^\lambda(n) \cap \delta(t)$ is nonempty. When the schema is deterministic, $\mathit{Out\text{-}lab\text{-}type}_G^\lambda(n) \cap \delta(t)$ is nonempty if and only if 1) the bag $\mathit{out\text{-}lab\text{-}type}_G^\delta(n, t) = \{\!\!\{a :: \delta(t, a) \mid (n, a, m) \in E\}\!\!\}$ belongs to $\delta(t)$ and 2) for every $(n, a, m) \in E$, $\delta(t, a)$ belongs to $\lambda(m)$. Using this argument, we show that the sufficient and necessary condition for tractable validation against deterministic ShEx using a class $\mathcal{C}$ of RBEs is tractable membership problem for $\mathcal{C}$, a decision problem defined formally as

$$\mathsf{MEMB}(\mathcal{C}) = \{(w, E) \mid E \in \mathcal{C},\ w \in L(E)\}.$$

5.2 Single-occurrence RBE (SORBE)

While the problem of membership of a bag to a language defined by an RBE is in general intractable [22], we identify a rich and practical class of RBEs with tractable membership. This class is obtained by disallowing repeating symbols, while allowing arbitrary intervals on symbols, a restriction on regular expressions commonly imposed in the context of document content models with evidence justifying its use [3, 11, 12, 17, 25]. Formally, a *single-occurrence regular bag expression* (SORBE) over Δ is an RBE that allows at most one occurrence of every symbol of Δ and allows the use of the wildcard $+$ inside expressions and arbitrary intervals on symbols a^I. Note that this also enables the use of the wildcard $E^?$ since it can be defined using ϵ and the union operator without repeating any symbol of Δ.

▶ **Theorem 5.** MEMB(SORBE) *is in PTIME.*

Proof. We fix a bag of symbols w over Δ. For a regular bag expression E, by $\Delta(E)$ we denote the subset of Δ containing exactly the symbols used in E. For a subset $X \subseteq \Delta$ by w_X we denote the bag over X obtained from w by removing all occurrences of symbols outside of X. W.l.o.g. we assume that the Kleene's plus E^+ is used only if $\varepsilon \notin L(E)$ (otherwise E^+ can be replaced by E^*).

The algorithm recursively constructs for an expression E a set of integers $I(E)$ such that $i \in I(E)$ iff $w_{\Delta(E)} \in L(E)^i$. This set is represented by an interval. Recall that an *interval* $[n; m]$ is a finite representation of the set $\{i \mid n \leq i \leq m\}$. It is *empty* if $m < n$ and we use $\emptyset$ to denote (the equivalence class of) all empty intervals. Also, the intersection of two intervals can be obtained easily $[n_1; m_1] \cap [n_2; m_2] = [\max\{n_1, n_2\}; \min\{m_1, m_2\}]$ and the *component-wise addition* $A \oplus B = \{a + b \mid a \in A,\ b \in B\}$ can be implemented as $[n_1; m_1] \oplus [n_2; m_2] = [n_1 + n_2; m_1 + m_2]$ with $m + \infty = \infty + m = \infty$ for any $m \in \mathbb{N} \cup \{\infty\}$. The algorithm works as follows (with $0/\infty = 0$ and $i/\infty = 1$ for $i \geq 1$):

$$I(\epsilon) = [0; \infty],$$
$$I(a^{[n,m]}) = \big[\lceil w(a)/m \rceil; \lfloor w(a)/n \rfloor\big],$$
$$I(E_1 \mid E_2) = I(E_1) \oplus I(E_2),$$
$$I(E_1 \parallel E_2) = I(E_1) \cap I(E_2),$$

$$I(E^*) = \begin{cases} [0; \infty] & \text{if } w_{\Delta(E)} = \varepsilon, \\ [1; \infty] & \text{if } w_{\Delta(E)} \neq \varepsilon \text{ and } I(E) \neq \emptyset, \\ \emptyset & \text{otherwise,} \end{cases}$$

$$I(E^+) = \begin{cases} [0; 0] & \text{if } w_{\Delta(E)} = \varepsilon, \\ [1; \max I(E)] & \text{if } w_{\Delta(E)} \neq \varepsilon \text{ and } I(E) \neq \emptyset, \\ \emptyset & \text{otherwise.} \end{cases}$$

Note that assigning $I(E^+) = [0;0]$ when $w_{\Delta(E)} = \varepsilon$ is valid since we assume $\varepsilon \notin L(E)$. Naturally, $w \in L(E)$ if and only if $1 \in I(E)$ and w uses only symbols present in E. □

We, therefore, immediately get

▶ **Corollary 6.** *Multi-type validation for deterministic* ShEx *using* SORBE *is in PTIME.*

We employ the single type requirement to reduce the NP-complete problem of exact set cover to single-type validation against a deterministic ShEx using only single-occurrence RBE_0 expressions.

▶ **Theorem 7.** *Single-type validation for deterministic* ShEx *using* SORBE *is NP-complete.*

5.3 Optimal validation algorithm

Some applications might not require testing validity of the whole graph, but rather checking the validity of only a fragment that will be accessed by the application. Such a fragment can be identified by a set of root nodes, entry points for navigating the graph, and typically, the application will require the entry points to satisfy certain types. In this section, we show how this scenario can be modeled with ShEx and present an efficient algorithm that works with deterministic shape expressions.

For this, take a schema $S = (\Sigma, \Gamma, \delta)$ such that Γ contains a special *universal type* $t_\top$ with the definition $\delta(t_\top) = (\Sigma \times \Gamma)^*$. The language of S is the universal graph language, as any node of any graph can be typed with $t_\top$. In essence, the universal type allows to forgo validation of any node because any node implicitly satisfies $t_\top$.

To carry out validation on a fragment of a graph identified by the entry points we introduce the notion of pre-typing, an assignment of required types to a selected set of nodes. Formally, a *pre-typing* of a graph G (w.r.t. S) is a partial mapping $\lambda_ : V \to 2^\Gamma$. Now, the objective is to find a *valid extension* of $\lambda_$ i.e., a valid m-typing λ of G w.r.t. S such that $\lambda_ \sqsubseteq \lambda$. The universal type $t_\top$ combined with pre-typing is a very powerful modelling tool. For instance, the rule $t_0 \to a :: t_1^* \parallel (\|_{b \in \Sigma, b \neq a} b :: t_\top)$ indicates that we are interested in checking correct typing for nodes reachable by an a-labelled edge, but all remaining nodes can have arbitrary types, therefore do not need to be typed.

Since we are not interested in typing the whole graph G, we focus on the smallest possible valid extension of a given pre-typing λ. Interestingly, we can show the following.

▶ **Lemma 8.** *For a deterministic* ShEx $S = (\Sigma, \Gamma, \delta)$ *with universal type, a graph* $G = (V, E)$*, and a pre-typing* $\lambda_ : V \to 2^\Gamma$*, if* $\lambda_$ *admits a valid extension, then it admits a unique* $\sqsubseteq$*-minimal valid extension.*

We present an algorithm that constructs the minimal valid extension of a given pre-typing $\lambda_$ of a given graph G w.r.t. a given deterministic Shape Expression Schema S with universal type. For technical reasons, we represent a typing as binary relation between the set of nodes and the set of types, and deliberately omit the universal type. More precisely, we use a relation $R_\lambda \subseteq V \times (\Gamma \setminus \{t_\top\})$ to represent the typing $\lambda(n) = \{t \mid (n,t) \in R_f\}$ if $(n,t) \in R_f$ for some $t \in \Gamma$, and $\lambda(n) = \{t_\top\}$ otherwise. Furthermore, we abuse notation and use λ instead of R_λ. Recall that $\delta(t,a)$ is the unique type used together with the symbol a in $\delta(t)$.

This algorithm is a modified graph flooding algorithm that maintains a frontier set F of pairs (n,t) for which it remains to be verified that the node n satisfies type t. Initially, this set contains only the pairs specified by the pre-typing (line 1). The algorithm fails whenever for some $(n,t) \in F$ the outgoing edges of n do not satisfy the structural constraints given by

Algorithm 1 $\mathsf{MinValidExt}(S, G, \lambda_)$

Input: $S = (\Sigma, \Gamma, \delta)$ a deterministic ShEx,
$G = (V, E)$,
$\lambda_ \subseteq V \times \Gamma$ a pre-typing;
Output: $\lambda \subseteq V \times \Gamma$ the minimal valid extension of $\lambda_$.

1: **let** $F := \lambda_$
2: **let** $\lambda := \emptyset$
3: **while** $F \neq \emptyset$ **do**
4: choose $(n, t) \in F$ and remove it from F
5: **let** $\textit{out-lab-type}^{\delta}_{G}(n, t) := \{\!\!\{(a, \delta(t, a)) \mid (a, m) \in \textit{out-lab-node}_G(n)\}\!\!\}$
6: **if** $\textit{out-lab-type}^{\delta}_{G}(n, t) \notin \delta(t)$ **then**
7: **fail**
8: $\lambda := \lambda \cup \{(n, t)\}$
9: **for** $(a, m) \in \textit{out-lab-node}_G(n)$ **do**
10: **if** $\delta(t, a) \neq t_\top$ **and** $(m, \delta(t, a)) \notin \lambda$ **then**
11: $F := F \cup \{(m, \delta(t, a))\}$
12: **return** λ

$\delta(t)$ (lines 5–7). If, however, the constraints are satisfied, any node m reachable from n is added to F with an appropriate type unless the type is universal.

Note that a run of the algorithm considers the pair (t, n) at most once, and therefore the main loop is executed at most $|V| \times |\Gamma|$ times. Once F is empty, the constructed λ represents the *minimal valid extension* of $\lambda_$. This algorithm is optimal in the sense that it constructs the minimal representation of the minimal valid extension and considers assigning a type to a node only if it is required to construct the extension. Naturally, for single occurrence RBEs the algorithm works in polynomial time.

▶ **Theorem 9.** *Given a deterministic* ShEx(SORBE) *S, a graph G, and a pre-typing $\lambda_ : V \to \Gamma$, the algorithm* $\mathsf{MinValidExt}(S, G, \lambda_)$ *constructs in polynomial time the minimal valid extension of $\lambda_$ if it exists, or fails otherwise.*

A slight modification of this algorithm works for the single-type semantics too: in the inner loop (lines 9–11) it suffices to add a check that neither F nor λ contain (m, t') for some $t' \neq \delta(t, a)$, which prevents assigning two different types to the same node. As a result such modified algorithm constructs an s-typing λ (with universal type omitted). Also, note that with the single-type modification the while loop is executed at most $|V|$ times, and the algorithm considers each edge of the graph at most once, i.e. the algorithm makes a single pass over the graph.

6 Expressive power

This section outlines the results of our study of expressive power of ShEx. First, we consider the first-order logic on graphs ($\mathsf{FO_G}$) over the standard signature consisting of relation names $(E_a)_{a \in \Sigma}$, and the existential monadic second-order logic on graphs ($\exists\mathsf{MSO_G}$) allowing only formulas of the form $\exists X_1, \ldots, X_n \varphi$, where $X_1, \ldots, X_n$ are monadic second order variables and φ is an FO formula using additional atomic formulae of the form $x \in X_i$.

We say that a class of graph languages $\mathcal{C}$ *separates* a graph H from a graph G if there is $L \in \mathcal{C}$ such that $G \in L$ and $H \notin L$. Consider the fork $G_<$ and the diamond $G_\diamond$ graphs in Figure 4. We observe that single-type semantics can easily separate $G_\diamond$ from $G_<$ while it

Figure 4 Fork and diamond graphs.

can be easily shown that the multi-type semantics cannot. However, even the single-type semantics cannot separate $G_<$ from $G_\diamond$ while this separation is trivial for $\mathsf{FO_G}$ and $\exists\mathsf{MSO_G}$. Also, let L_{cycle} be the set of graphs labeled with $\Sigma = \{a, b\}$ such that for every node n with an incoming b-edge, there is a cycle reachable from n. It is a classic result that $\mathsf{FO_G}$ cannot define sets of graphs which contain cycles of unbounded size [15]. However, L_{cycle} can be defined in both semantics with the following schema:

$$S_{cycle} : t_0 \to (a :: t_\bullet \mid a :: t_0)^* \parallel (b :: t_\bullet)^* \qquad t_\bullet \to (a :: t_\bullet^+ \mid b :: t_\bullet) \parallel (a :: t_0)^* \parallel (b :: t_\bullet)^*$$

The following table present a comparison of expressive power, indicating whether a language L satisfying the constraints given in the first column can be expressed by each of the formalisms.

	$\mathsf{FO_G}$	$\mathsf{ShEx_m}$	$\mathsf{ShEx_s}$	$\exists\mathsf{MSO_G}$
$L : G_\diamond \in L,\ G_< \in L$	✓	✓	✓	✓
$L : G_\diamond \notin L,\ G_< \in L$	✓	×	✓	✓
$L : G_\diamond \in L,\ G_< \notin L$	✓	×	×	✓
L_{cycle}	×	✓	✓	✓

It should also be noted that RBEs can express cardinality constraints e.g., $(a \parallel b)^*$ means that the number of a must be equal to the number of b, that cannot be captured by $\exists\mathsf{MSO_G}$. However, if we limit the expressive power of the bag languages used in schemas to those that can be captured by $\exists\mathsf{MSO_G}$ e.g., any subclass of $\mathsf{RBE}(a^I, \parallel, |)$, the expressive power of schemas is captured by $\exists\mathsf{MSO_G}$, a result easily proven with a simple adaptation of the standard translation of an automaton to an existential monadic second-order formula [37]. The restriction on use of Kleene closure in defining unordered content models has been previously advocated for complexity reasons in the context of XML [12], and we provide yet another reason.

The single-type semantics is in fact very powerful and can easily capture graph languages defined by homomorphism into a fixed graph, as illustrated in the proof of Theorem 3. It seems unlikely that the multi-type semantics is as powerful as suggested by complexity arguments in the present paper. Both semantics have the same closure properties:

▶ **Theorem 10.** ShEx_s *and* ShEx_m *are not closed under union and complement. Both* ShEx_s *and* ShEx_m *are closed under intersection.*

The closure under intersection can be extended to a powerset technique, similar to the determinisation technique of finite automata [21], that allows to show that the single-type semantics is in fact more powerful than the multi-type semantics.

▶ **Theorem 11.** ShEx_s *properly contains* ShEx_m.

ShEx can be viewed as an automaton on edge-labeled (possibly infinite) trees obtained by unraveling the input graph and we believe that such a model would correspond closely to Presburger automata [32] if the latter were extended to infinite trees, taking a universal acceptance condition. Interestingly, in this analogy the single-type semantics corresponds to deterministic automata while multi-type semantics corresponds to nondeterministic ones. More recently, k-Pebble automata on graphs have been proposed [30] but they are not comparable with ShEx because capable of expressing arbitrary FO properties. Recognizable sets of graphs as defined in [13] go beyond $\mathsf{MSO_G}$, and therefore, capture $\mathsf{ShEx}(\mathsf{RBE}(a^I, \|, |))$.

Finally, ShEx are incomparable with both the node replacement (NR) graph grammars, and the hyperedge replacement (HR) graph grammars. On the one hand, the language $\{G_\diamond\}$ is definable by both HR and NR graph grammars with single initial graph and no rules. On the other hand, ShEx can define languages that are not definable by neither HR nor NR grammars: HR grammars can define only languages of graphs of bounded tree-width while NR grammars cannot define a language containing infinitely many square grids.

7 Conclusions

We have investigated Shape Expressions Schemas (ShEx), a novel formalism of schemas for RDF graphs currently under development by W3C. We have proposed two alternative semantics, single-type and multi-type, studied their expressive powers and the complexity of the problem of validation. We have also proposed a notion of determinism in order to curb down the complexity of validation. While the single-type semantics is in general intractable, for multi-type validation we have identified two essential bottleneck complexity problems on RBE, membership and satisfiability of RBEs with intersection, depending on whether or not deterministic expressions are used. Summary of complexity results can be found in Table 1.

	$\mathsf{RBE_0} = \mathsf{RBE}(a^M, \|)$	RBE	SORBE	SORBE det.	SORBE det. + $\lambda_$ + $t_\top$
multi-type	PTIME (Thm. 3)	NP-c. (Thm. 4)		PTIME (Cor. 6 and Thm. 9)	
single-type	NP-c. (Thm. 1)		NP-c. (Thm. 7)		PTIME (Section 5.3)

Table 1 Summary of main complexity results for the validation problem.

Our results on expressive power suggest that an unrestricted use of the Kleene closure may render the proposed formalism too powerful and so far there exists little evidence of its practical usability in the context of unordered content model [12, 7]. Complexity results suggest that the single-type semantics may be too expensive for practical application unless we wish to validate only a fragment of graph with a given pretyping. As for the multi-type semantics, validation is tractable for a small yet practical fragment $\mathsf{RBE_0}$ and if we use determinism, a richer class of SORBEs can be handled efficiently.

Future work. In the future, we plan to investigate the impact of data value constraints on complexity of validation. Our preliminary study shows that adding *local* data value constraints, such as domain check, does not affect our results. However, the impact of value constraints of *global* nature, such as key dependencies, remains to be investigated. We also plan to thoroughly evaluate experimentally the proposed algorithms and compare with existing validation approaches (SPIN, ICV) on both real-life and synthetically generated data e.g., RDF export of TPC-H benchmark data [38]. Our preliminary experiments [8] are very promising. Also, we would like to study the complexity of classical static analysis

problems such as schema containment and query validity in the presence of schema. Finally, we would like devise inference algorithms for ShEx drawing inspiration from learning XML twig queries [35] and schemas for XML [4, 3, 10].

References

1. 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.
2. M. Arenas, J. Pérez, Reutter J., C. Riveros, and J. Sequeda. Data exchange in the relational and RDF worlds. Invited talk at the International Workshop on Semantic Web Information Management (SWIM), June 2011.
3. G. J. Bex, F. Neven, T. Schwentick, and S. Vansummeren. Inference of concise regular expressions and DTDs. *ACM Transactions on Database Systems (TODS)*, 35(2), 2010.
4. G. J. Bex, F. Neven, and S. Vansummeren. Inferring XML schema definitions from XML data. In *International Conference on Very Large Data Bases (VLDB)*, pages 998–1009, 2007.
5. Chimezie O. Birte Glimm. SPARQL 1.1 Entailment Regimes. `http://www.w3.org/TR/sparql11-entailment/`, 2012.
6. J. Bolleman, S. Gehant, and N. Redaschi. Catching inconsistencies with the semantic web: A biocuration case study. In *International Workshop on Semantic Web Applications and Tools for Life Sciences (SWAT4LS)*, 2012.
7. I. Boneva, R. Ciucanu, and S. Staworko. Schemas for unordered XML on a DIME. *Theory of Computing Systems*, 2014. To appear. Available at `http://arxiv.org/abs/1311.7307`.
8. I. Boneva, J. Emilio Labra Gayo, S. Hym, E. G. Prud'hommeaux, H. Solbrig, and S. Staworko. Validating RDF with shape expressions. *ArXiv e-prints*, April 2014. Available at `http://arxiv.org/abs/1404.1270`.
9. D. Brickley and R. V. Guha. RDF Schema 1.1. `http://www.w3.org/TR/rdf-schema`, February 2004.
10. R. Ciucanu and S. Staworko. Learning schemas for unordered XML. In *International Symposium on Database Programming Languages (DBPL)*, 2013.
11. D. Colazzo, G. Ghelli, L. Pardini, and C. Sartiani. Linear inclusion for XML regular expression types. In *International Conference on Information and Knowledge Management (CIKM)*, pages 137–146, 2009.
12. D. Colazzo, G. Ghelli, and C. Sartiani. Efficient inclusion for a class of XML types with interleaving and counting. *Information Systems*, 34(7):643–656, 2009.
13. B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12 – 75, 1990.
14. M. Dean and M. Schreiber. OWL Web Ontology Language Reference. `http://www.w3.org/TR/owl-ref`, February 2004.
15. H.-D. Ebbinghaus and J. Flum. *Finite model theory.* Springer, 1995.
16. J. D. Fernández, M. A. Martínez-Prieto, C Gutiérrez, A. Polleres, and A. Arias. Binary RDF representation for publication and exchange (HDT). *Journal of Web Semantics*, 19:22–41, 2013.
17. G. Ghelli, D. Colazzo, and C. Sartiani. Linear time membership in a class of regular expressions with interleaving and counting. In *International Conference on Information and Knowledge Management (CIKM)*, pages 389–398, 2008.
18. S. Ginsburg and Spanier E. H. Semigroups, presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, December 1966.
19. A. V. Goldberg, E. Tardos, and R. E. Tarjan. Network flow algorithms. In *Algorithms and Complexity*, Volume 9, *Paths, Flows, and VLSI-Layout*, 1990.

20 B. Groz, S. Maneth, and S. Staworko. Deterministic regular expressions in linear time. In *ACM Symposium on Principles of Database Systems (PODS)*, May 2012.

21 J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 2nd edition, 2001.

22 E. Kopczynski and A. To. Parikh images of grammars: Complexity and applications. In *LICS*, pages 80–89, 2010.

23 D. Kozen. Lower bounds for natural proof systems. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 254–266, 1977.

24 J. E. Labra Gayo, E. Prud'hommeaux, H. Solbrig, and J. M. Álvarez Rodríguez. Validating and describing linked data portals using RDF Shape Expressions. In *Workshop on Linked Data Quality*, September 2015.

25 M. Montazerian, P. T. Wood, and S. R. Mousavi. XPath query satisfiability is in PTIME for real-world DTDs. In *International XML Database Symposium (Xsym)*, pages 17–30, 2007.

26 D. C.O 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.

27 C. H. Papadimitriou. On the complexity of integer programming. *Journal of the ACM*, 28(4):765–768, October 1981.

28 R. J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.

29 E. Prud'hommeaux, J. E. Labra Gayo, and H. Solbrig. Shape Expressions: An RDF validation and transformation language. In *International Conference on Semantic Systems*, September 2015.

30 J. L. Reutter and T. Tan. A formalism for graph databases and its model of computation. In *AMW*, volume 749 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.

31 A. Ryman, A. Le Hors, and S. Speicher. OSLC Resource Shape: A language for defining constraints on linked data. In *Proceedings of the WWW2013 Workshop on Linked Data on the Web (LDOW)*. CEUR-WS.org, 2013.

32 H. Seidl, T. Schwentick, and A. Muscholl. Counting in trees. *Logic and Automata*, pages 575–612, 2008.

33 J. Sequeda, H. Tirmizi, S, Ó. Corcho, and D. P. Miranker. Survey of directly mapping SQL databases to the Semantic Web. *Knowledge Engineering Review*, 26(4):445–486, 2011.

34 E. Sirin. Data validation with OWL integrity constraints. In *International Conference on Web Reasoning and Rule Systems (RR)*, pages 18–22, 2010.

35 S. Staworko and P. Wieczorek. Learning twig and path queries. In *International Conference on Database Theory (ICDT)*, March 2012.

36 J. Tao, E. Sirin, J. Bao, and D. L. McGuinness. Integrity constraints in OWL. In *International Conference on Artificial Intelligence (AAAI)*, 2010.

37 J. W. Thatcher and Wright J. B. Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2:57–82, 1968.

38 TPC. TPC benchmarks, `http://www.tpc.org/`.

39 W3C. RDF validation workshop report: Practical assurances for quality RDF data. `http://www.w3.org/2012/12/rdf-val/report`, September 2013.

40 W3C. Shape expressions schemas, 2013. `http://www.w3.org/2013/ShEx/Primer`.