Challenges in RDF validation

Jose Emilio Labra-Gayo, Herminio García-González, Daniel Fernández-Alvarez, and Eric Prud'hommeaux

Abstract The RDF data model forms a cornerstone of the Semantic Web technology stack. Although there have been different proposals for RDF serialization syntaxes, the underlying simple data model enables great flexibility which allows it to be successfully employed in many different scenarios and to form the basis on which other technologies are developed. In order to apply an RDF-based approach in practice it is necessary to communicate the structure of the data that is being stored or represented. Data quality is of paramount importance for the acceptance of RDF as a data representation language and it must be enabled by the use of tools that can check if some data conforms to some specific structure. There have been several recent proposals for RDF validation languages like ShEx and SHACL. In this chapter, we describe both proposals and enumerate some challenges and trends that we foresee with regards to RDF validation. We devote more space to what we consider one of the main challenges, which is to compare ShEx and SHACL and to understand their underlying foundations. To that end, we propose an intermediate language and show how ShEx and SHACL can be converted to it.

1 Introduction

RDF is a key part of the Semantic Web. Its data model is a combination of simplicity and powerful expressiveness which can be applied to represent information in any domain. RDF was proposed around 1997 and became a W3C recommendation in 1999 [1] using an XML based syntax. It was designed as a knowledge representation

Jose Emilio Labra Gayo · Herminio García-González · Daniel Fernández-Alvarez
University of Oviedo
Spain

Eric Prud'hommeaux
W3C, MIT

language with a flexible data model based on graphs. RDF Schema (RDFS) was soon proposed as a means to define RDF vocabularies [2].

At that time, there was some parallel evolution of XML and RDF. XML was promoted as a data exchange technology with validation capabilities (DTDs [3], XML Schema [4], RelaxNG [5], etc.) while RDF acquired a prominent role as a knowledge representation formalism where RDFS could be used to infer new knowledge rather than to validate if RDF data conformed to some schema. New proposals emerged that increased the RDF Schema expressiveness adding the possibility to define full ontologies which crystallized under the OWL W3C recommendation [6].

In order to use RDF in practice, it was necessary to develop query languages. Several proposals appeared (see [7]) and in 2008 the SPARQL language became a W3C recommendation [8]. These technologies: RDF, RDFS, SPARQL and OWL defined the core semantic web technology stack on which most of the semantic web applications were based. In order to publish reusable semantic web data on the web, the linked data principles [9] were proposed as four main guidelines where RDF is specifically mentioned as one of the standards that provides useful information. Linked data became popular [10] and a lot of initiatives have been created publishing linked data information using RDF.

Practical Semantic Web applications require some technology to describe and validate the RDF data that is being employed [11] by the different stakeholders. The producers of RDF need to define the intended structure of the RDF graphs they are generating, while the consumers can check if the received graphs conform to that structure. In recent years, validating RDF data has acquired a lot of traction and there have appeared 2 different technologies: ShEx and SHACL with the same goal: provide an RDF validation and description technology. Given that they share the same goal and were defined almost at the same time, a question arises as to whether one or the other should be employed of fits better for some specific use cases. Solving this question is probably the main challenge to solve in this field at the moment and for that reason we devote to it most of the chapter space. Although the future evolution of both languages will depend on multiple factors, we consider that identifying their foundations using minimal language that can represent both can help. In this chapter, we give a short overview of ShEx and SHACL from a formal point of view, review the main differences of their core language and present the $\mathcal{S}$-language, a minimal language that can be used as an intermediate language for both. We also present 2 algorithms that convert ShEx and SHACL to $\mathcal{S}$. Although ShEx and SHACL can be translated to low level $\mathcal{S}$ code, higher level translations between them that preserve the shapes definitions are more difficult. To finalize the chapter, we identify other challenges and future work that we consider important and on which we are working at this moment.

This chapter is organized as follows: Section 2 describes the RDF data model. Section 3 describes the main RDF validation proposals: Shape Expressions (ShEx) and Shapes Constraint Language (SHACL). Section 3.3 compares both proposals and section 3.4 defines the $\mathcal{S}$ language that can be used to represent both and presents algorithms to translate ShEx and SHACL to $\mathcal{S}$. Section 4 describes challenges in the RDF validation field.

2 RDF data model

There are three main kinds of nodes in RDF: IRIs represented by the set $\mathcal{I}$, blank nodes represented by the set $\mathcal{B}$ and literals represented by $\mathcal{L}$.

An important feature of RDF is the use of IRIs as global identifiers, enabling easy integration and merging of RDF graphs. Literals are pairs of the form (s, d) where s is a string representing the lexical form of the literal and d is an IRI that declares the datatype of that string[1]. Blank nodes are used in RDF to locally identify nodes as a kind of semantic variables [12]. An RDF graph g is defined as a set of triples $\langle s, p, o \rangle$ such that $s \in V_s$, $p \in V_p$ and $o \in V_o$ where $V_s = \mathcal{I} \cup \mathcal{B}$ is the vocabulary of subjects, $V_p = \mathcal{I}$ is the vocabulary of predicates and $V_o = \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$ is the vocabulary of objects.

Example of a simple RDF graph

The following code presents a simple RDF graph using Turtle notation [13], a human-readable syntax for RDF.

```
prefix : <http://example.org/>

:bob    :name        "Robert" ;
        :age         "None";
        :birthPlace  :Oviedo ;
        :enrolledIn  :cs102 .
:alice  :name        "Alice" ;
        :age         23 ;
        :enrolledIn  :cs101 ;
        :birthPlace  :Oviedo ;
        :knows       :carol .
:carol  :name        "Carol" ;
        :enrolledIn  :cs101 .
:cs101  :subject     "Programming" ;
        :students    :alice, :carol .
:cs102  :subject     "Algebra" ;
        :students    :bob .
```

The example is depicted in figure 1. using the RDFShape tool [2] developed by the first author of this paper.

A property path represents a possible route in a graph between two nodes. The concept was introduced in SPARQL 1.1 [14] and allows navigational queries over RDF graphs. Following [15], a property path pp can be defined by the following grammar:

[1] There is also a special kind of literals that have an associated language tag. We omit them in this chapter to simplify the presentation.

[2] RDFShape is deployed at: `http://rdfshape.weso.es`. The following link can be used to show that graph or dynamically visualize other RDF graphs: `https://goo.gl/48KsEP`

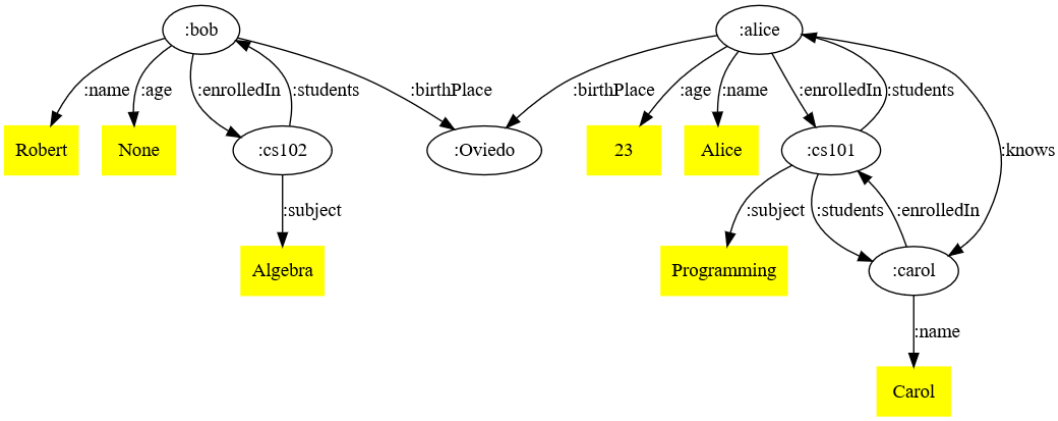

Fig. 1 RDF graph visualization using the RDFShape tool

$$pp ::= p \mid \hat{}pp \mid pp_1 \cdot pp_2 \mid pp^* \mid pp_1 \vee pp_2 \mid !\{p_1 \dots p_n\}$$

where p denotes a single predicate. Two nodes n_1 and n_2 are connected by a property path e in a graph g if $(n_1, n_2) \in [\![e]\!]^g$ where $[\![e]\!]^g$ is defined as:

$$\begin{aligned}
[\![p]\!]^g &= \{(s,o) \mid \langle s,p,o\rangle \in g\} \\
[\![\hat{}pp]\!]^g &= \{(o,s) \mid \langle s,p,o\rangle \in g\} \\
[\![pp_1 \cdot pp_2]\!]^g &= [\![pp_1]\!]^g \circ [\![pp_2]\!]^g \\
[\![pp_1 \vee pp_2]\!]^g &= [\![pp_1]\!]^g \cup [\![pp_2]\!]^g \\
[\![pp^*]\!]^g &= \bigcup_{i \geq 1} [\![pp^i]\!]^g \cup \{(n,n) \mid n \text{ is a node in } g\} \\
[\![!\{p_1 \dots p_n\}]\!]^g &= \{(s,o) \mid \exists p \text{ with } \langle s,p,o\rangle \in g \text{ and } p \notin \{p_1 \dots p_n\}\}
\end{aligned}$$

where $\circ$ is the composition of binary relations, and pp^i is the concatenation $pp \cdot \dots \cdot pp$ of i copies of pp. The SHACL recommendation uses a subset of SPARQL property paths which does not include the negation operator !. SPARQL 1.1 allows also the negation of inverse property sets which we don't need in this chapter.

3 Validating RDF Data

3.1 ShEx

Shex was designed as a human-readable and intuitive language for RDF validation [16]. The syntax adopts Turtle and SPARQL tokens so it is familiar to users of those languages while the semantics is inspired by XML validation languages like XML Schema or RelaxNG, which are based on regular expressions. ShEx is being developed by the W3C Shape Expressions community group and its current version is called ShEx 2.0 [17].

Basic example of a ShEx schema

The following code declares two shapes: **<User>** and **<Course>**.

```
prefix :    <http://example.org/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

<User> IRI {
 :name       xsd:string  ;
 :age        xsd:integer? ;
 :enrolledIn @<Course>+  ;
 :knows      @<User>* ;
 :birthPlace IRI ?
}
<Course> {
 :subject    xsd:string +      ;
 :students   @<User> {1,20}
}
```

Nodes conforming to **<User>** must be IRIs and satisfy the following constraints:

- There must be exactly one property **:name** whose value must belong to the datatype **xsd:string**
- There can be an optional property **:age** whose value must belong to the datatype **xsd:integer**
- There must be one or more properties **:enrolledIn** whose values must conform to shape **<Course>**.
- There can be zero or more properties **:knows** whose values conform to shape **<User>**.
- There can be an optional property **:birthPlace** whose value must be an IRI.

while nodes conforming to **<Course>** must satisfy the rules:

- There can be one or more properties **:subject** whose value must belong to datatype **xsd:string**
- There must be between 1 and 20 properties **:students** whose values must conform to shape **<User>**.

ShEx validation process also defines the concept of Shape maps, which associate shapes with sets of nodes that have to be validated.

Example of a shape map

A simple shape map that associates node **:alice** with shape **<User>** can be declared as:

```
:alice@<User>,:bob@<User>
```

The result of validation in ShEx is also defined in terms of shape maps that associate nodes to shapes indicating if they conform or not. The ShEx validation process may trigger the validation of intermediate nodes which can be returned in the resulting shape map.

Example of a result shape map

The following shape map is the result of evaluating the RDF graph from example 2 against the ShEx schema 3.1 using the query shape map from example 3.1. It declares that **:alice** and **:carol** conform to shape **<User>**, **:cs101** conforms to shape **<Course>** and **:bob** does not conform to shape **<User>**.

```
:alice@<User>,:bob@!<User>,:carol@<User>,:cs101@<Course>
```

In the rest of the chapter we use a subset of ShEx that captures the main features of the language and is defined by the abstract syntax defined in table 1, which follows a similar grammar to the one presented in [18].

se ::=	IRI | BNode | $datatype(iri)$	Node constraints
	se_1 **AND** se_2	Conjunction
|	se_1 **OR** se_2	Disjunction
|	**NOT** se	Negation
|	$@l$	Shape label reference
|	**CLOSED**? (**EXTRA** $p_1 \ldots p_n$)? { te }	Triple expression te with optional **CLOSED** qualifier and optional $n \geq 0$ **EXTRA** predicates p_i
te ::=	$te_1; te_2$	Each of te_1 and te_2
|	$te_1 \mid te_2$	Some of te_1 or te_2
|	$\llcorner \xrightarrow{p} se\{min, max\}$	Between min and max triples with predicate p that conform to shape expression se

Table 1 ShEx abstract syntax used in this chapter

The language has two main terms, shape expressions and triple expressions:

- Shape expressions (denoted by se) define constraints on a node. They can be simple node constraints (IRI, BNode or datatype(iri)), combinations of the logical operators **AND**, **OR** and **NOT**, references to other shapes (denoted by $@l$) and a shape definition with an optional **CLOSED** qualifier and $n \geq 0$ **EXTRA** properties p_i and a triple expression te.
- Triple expressions (denoted by te) define the neighbourhood of a node which represents the triples or arcs incoming and outgoing from it. The basic triple expression is a triple constraint $\llcorner \xrightarrow{p} se\{min, max\}$ which declares that there must be between min and max triples with predicate p whose values conform to shape se. The values of min are integers, while the values of max can be integers or unbounded (denoted by $*$). Triple expressions can also be combined using the each-of operator (**;**) for unordered concatenation or the alternative operator (|). For each triple expression te, we define: $ps(te)$ as the set of properties that appear in te and $shapes^{te}$ as a function that associates for each predicate p the set of shapes $\{se \mid \llcorner \xrightarrow{p} se\{min, max\}$ appears in $te\}$.
- In the triple expression $\llcorner \xrightarrow{p} se\{min, max\}$, if we omit the $\{min, max\}$ part, it is assumed to be $\{1, 1\}$. The cardinalities $\{0, *\}, \{0, 1\}, \{1, *\}$ can be simplified by the symbols $*, ?, +$ respectively.

A ShEx-schema is defined as a pair (L, δ) where L is a set of shape labels and $\delta : L \mapsto se$, associates a shape expression $s \in se$ to each $l \in L$.

The main differences between the ShEx fragment used here and the full ShEx language are:

- We employ only 3 types of node constraints: IRI, BNode and $datatype(iri)$ to declare if a node is an IRI, a blank node or a literal with some datatype while full ShEx contains a longer list of built-in node constraints like XML Schema facets. From a semantic point of view, those node constraints have a similar treatment as the ones we use.
- ShEx has also the possibility to declare node constraints formed by value sets whose elements can be RDF nodes or stem ranges. From a semantic point of view, these node constraints could be translated to a set of values which could be defined in the same way as the 3 built-in node constraints that we propose in this chapter.

An important feature of ShEx is that properties in triple constraints are closed: the system collects the possible values of each property and checks that there are no other values different from those that appear in the triple constraints defined in the triple expression. This feature can be bypassed by the **EXTRA** qualifier which declares that other values different from those that have been declared are admitted.

Example of shape with **EXTRA** qualifier

The following code declares that nodes conforming to shape **<Product>** must have one property **<code>** whose value belongs to datatype **xsd:string**, another property

whose value belongs to **xsd:integer** and are allowed to have any other properties **<code>** whenever they are not strings or integers.

```
<Product> EXTRA <code> {
 <code>      xsd:string  ;
 <code>      xsd:integer ;
}
```

The optional **CLOSED** qualifier in ShEx declares that the only properties allowed are those that appear in the triple expression definition. By default, ShEx allows other properties.

Example of using CLOSED

Given the following RDF data, **:dave** would conform to shape **<User>** defined in example 3.1 even if it has property **:gender** which was not declared. It would not conform, if the **CLOSED** qualifier was added to **<User>** .

```
prefix : <http://example.org/>
:dave  :name       "Dave" ;
       :gender     :Male ;
       :enrolledIn :cs103 .
:cs103 :subject    "Robotics" ;
       :students   :dave .
```

The shape maps specification [19] defines shape maps as sets of associations with the form $ns@l$, where ns is a node selector and $l \in L$ is a shape label. Node selectors can be RDF nodes or triple patterns. A triple pattern has the form $\{s\ p\ o\}$ where s can be an IRI, the keyword $FOCUS$ or the wildcard _, p is a predicate, and o can be an IRI, a literal, the keyword **FOCUS** or the wildcard _. Given a node selector ns and an RDF graph g, the nodes selected by ns in graph g, denoted by $[\![ns]\!]^g$ are defined in table 2.

$$
\begin{aligned}
[\![n]\!]^g &= \{n\} \\
[\![\{\text{FOCUS } p\ o\}]\!]^g &= \{n \mid \langle n, p, o\rangle \in g\} \\
[\![\{\text{FOCUS } p\ \_\}]\!]^g &= \{n \mid \exists o \langle n, p, o\rangle \in g\} \\
[\![\{s\ p\ \text{FOCUS}\}]\!]^g &= \{n \mid \langle s, p, n\rangle \in g\} \\
[\![\{\_\ p\ \text{FOCUS}\}]\!]^g &= \{n \mid \exists s \langle s, p, n\rangle \in g\}
\end{aligned}
$$

Table 2 $[\![ns]\!]^g$ = nodes selected by ns in graph g

Example of a shape map with a node selector

The following shape map:

```
:alice@<User>,{FOCUS rdf:type :Person}@<User>
```

selects node **:alice** and all nodes with **rdf:type :Person** to be validated as **<User>**.

3.2 SHACL

In July, 2017, SHACL was approved as a W3C recommendation[20]. SHACL was influenced by SPIN [21] and OSLC Resource Shapes [22]. The language was divided in two parts: SHACL Core which contains built-in constraint components and SHACL-SPARQL, which defines a mechanism that allows users to create their own constraint components using SPARQL. In this chapter, we will focus on SHACL Core.

SHACL defines Shapes as groups of constraints. There are two main types of shapes: node shapes which constraint the values of some node, and property shapes which constraint the values of a particular property or path.

A difference between the concept of shape in ShEx and SHACL is that in SHACL, shapes can also contain target declarations about which nodes or sets of nodes must be validated. This is accomplished with shape maps in ShEx. In section 3.3 we present an algorithm to convert SHACL target declarations to ShEx shape maps.

Example of a SHACL shapes graph

The following code declares two node shapes **<User>** and **<Course>** which have a meaning similar to the ShEx shapes represented in example 3.1. The **<User>** shape contains 5 constraint components, a node kind declaration (line 2) and 4 anonymous property shapes. The first property shape has path **:name** and declares that values of predicate **:name** must be literals with datatype **xsd:string** and that the number of such values must be exactly one. The rest of property shapes are similar, when there is no **sh:minCount** declaration it is assumed 0, and when there is no **sh:maxCount** declaration, it is assumed unbounded. The third property shape (lines 11-13) has path **:enrolledIn** and uses the constraint **sh:node** to declare that the values of predicate **:enrolledIn** must satisfy the shape **<Course>**. The **<Course>** shape is composed of two property shapes. Notice that this definition is recursive as it contains a cyclic dependency between shapes **<User>** and **<Course>**.

```
1 <User> a sh:NodeShape;
2  sh:nodeKind sh:IRI ;
3  sh:property [ sh:path :name ;
4   sh:minCount 1; sh:maxCount 1;
```

```
  sh:datatype xsd:string ;
 ] ;
 sh:property [ sh:path :age ;
  sh:maxCount 1;
  sh:datatype xsd:integer ;
 ] ;
 sh:property [ sh:path :enrolledIn ;
  sh:node <Course> ;
 ] ;
 sh:property [ sh:path  :knows ;
  sh:node <User> ;
 ] .
<Course> a sh:NodeShape ;
 sh:property [ sh:path :subject ;
  sh:minCount 1;
  sh:datatype xsd:string ;
 ] ;
 sh:property [ sh:path :students ;
  sh:minCount 1; sh:maxCount 20;
  sh:node <User> ;
 ] .
```

SHACL processors take as input an RDF graph to be validated, called the data graph, and another RDF graph which contains the shapes declarations, called shapes graph and denoted by S_g. A shapes graph S_g contains nodes with shape declarations which can be either node shapes (with type **sh:NodeShape**) or property shapes (with type **sh:PropertyShape**). Each shape is formed by a list of target declarations and constraint components:

- Target declarations associate sets of nodes in the data graph with shapes with which they must be validated. For example: **<User> sh:targetNode :alice; sh:targetClass :Person** declares the following nodes to be validated with shape **<User>**: node **:alice** and all nodes that have **rdf:type :Person** or that are **rdfs:subClassOf** some node which has **rdf:type :Person**.
- Constraint components declare constraints on nodes. Table 3 contains the list of constraint components that we will use in this chapter.

Property shapes must contain the **sh:path** declaration that identifies the path of values that are constrained for some focus node. SHACL paths are a subset of SPARQL property paths.

3.3 Comparing Shex and SHACL

Although ShEx and SHACL have a similar goal: describing and validating RDF graphs, they were designed with different priorities and there are several differences between them that need to be taken into account. Chapter 7, of [23] contains a more

Constraint	Parameter	Meaning
`sh:minCount`	Integer n	n or more values must satisfy the constraint
`sh:maxCount`	Integer n	No more than n values must satisfy the constraint
`sh:nodeKind`	`sh:IRI`	Node must be an IRI
`sh:nodeKind`	`sh:BlankNode`	Node must be a blank node
`sh:datatype`	IRI dt	Node must be a literal with datatype dt
`sh:node`	Node n	Node must conform to shape n (unqualified)
`sh:qualifiedValueShape`	Node n	Node must conform to shape n (qualified)
`sh:qualifiedMinCount`	Integer n	n or more qualified values satisfy the constraint
`sh:qualifiedMinCount`	Integer n	No more than n qualified values satisfy the constraint
`sh:property`	Node ps	Each node satisfies the property shape ps

Table 3 Subset of SHACL code constraint components used in this chapter

detailed comparison of both ShEx and SHACL. In this section we overview the main points:

- **Syntactic differences**. ShEx has been designed from the start to be an intuitive, domain-specific language with its own compact syntax similar to Turtle or SPARQL. ShEx schemas can be serialized using compact syntax as well as any other RDF syntax (JSON-LD, Turtle, etc.). Any of those syntaxes are interchangeable and the schemas can be converted between one syntax to the other. On the other hand, SHACL was designed as an RDF vocabulary. There was a proposal for a compact syntax that was not accepted as part of the SHACL recommendation.
- **Underlying philosophy**. ShEx schemas look like grammar specifications which can be employed to define the content of RDF data. There is more emphasis on positive validation results defined by result shape maps than on error reporting. SHACL's emphasis is more on constraint validation and error reporting. The SHACL specification details the format of violation errors while it leaves unspecified the format of positive validations or conformant nodes. In practice, in most SHACL implementations, it is difficult to distinguish between a conformant node and a node that was intentionally or accidentally skipped[3].
- **Invoking validation and shape maps**. ShEx separates the concept of shapes from the association of which nodes will be validated with those shapes. This is a design choice motivated by the need to reuse shape definitions in different contexts. SHACL shapes can have target declarations integrated in their definitions which may make those shapes less reusable in different contexts. It is possible to translate SHACL target declarations to ShEx shape maps that allow property paths in the predicate position. The translation can be defined as:

[3] An exception is our Shaclex library[24] which also provides information about conformant nodes

$$\langle s, \text{sh:targetNode}, n \rangle = n@s$$
$$\langle s, \text{sh:targetClass}, c \rangle = \{\text{FOCUS rdf:type/rdfs:subClassOf* } c\}@s$$
$$\langle s, \text{sh:targetSubjectsOf}, p \rangle = \{\text{FOCUS } p \ \_\}@s$$
$$\langle s, \text{sh:targetObjectsOf}, p \rangle = \{\_ \ p \ \text{FOCUS}\}@s$$

- **Semantic specification.** The ShEx semantic specification is based on formal mathematical terms. It provides a semantics for recursive shapes. The combination of recursion and negation is solved by imposing the requirement that there is no negated reference from one shape to itself directly or indirectly. With this requirement, it is possible to define a well-founded semantics using stratification [18, 25].
  SHACL was initially defined using SPARQL templates and some auxiliary functions although the final recommendation employs a natural language description of the language. In the case of SHACL, the validation of recursive shapes is not defined and is left to SHACL processor implementations. Corman et al. [26] have recently proposed a SHACL semantics based on partial assignments that handles the combination between negation and recursion. However, at the time of this writing, we are not aware of any SHACL implementation that supports that semantics.
- **Modularization and shape reusability.** ShEx has a built-in declaration to import some schema from an IRI while SHACL processors recognize the `owl:imports` property to transitively following and importing the referenced shapes graphs.
  It is possible to extend a shape from another shape in both languages using the conjunction operator. This feature restricts the values of existing properties which may not always be the intended result if some of them are repeated properties.
- **Inference.** ShEx validators are focused on RDF graphs as they are presented to the validator without any interaction between validation and any inference mechanism. On the other hand, SHACL has several features that can interact with inference engines. For example, the `sh:targetClass` declaration has a partial built-in treatment of the property `rdfs:subClassOf` from RDFS, which means that it handles only the closure of that property while ignoring other RDFS properties like `rdfs:domain`, `rdfs:range`, etc. This feature can make full RDFS entailment incompatible with SHACL validators.
- **Property paths and property pair constraints.** ShEx defines triple constraints over a single property defined by an outgoing arc (a predicate) or an inverse arc (represented by symbol `^`). By contrast property shapes in SHACL use property paths, e.g. `:knows/:name`. In ShEx, such steps in a path require intermediate shapes, e.g. `:knows { :name xsd:string }`. Cardinality constraints applied to SHACL property paths are satisfied by any traversal of the property steps while in ShEx, the intermediate shapes would each require explicit cardinality constraints.

Example using property paths

A shape **<Invoice>** whose nodes have exactly one **:payment** with exactly one **:amount** of datatype **xsd:integer** can be declared in ShEx as:

```
<Invoice> {
:payment { :amount xsd:integer } ;
}
```

In SHACL, it is possible to define a similar encoding with two shapes, but one may also be tempted to use a property path as follows:

```
<Invoice> a sh:NodeShape ;
  sh:property [ sh:path (:payment :amount) ;
  sh:datatype xsd:integer ;
  sh:minCount 1 ; sh:maxCount 1
  ] .
```

However, with that shapes graph, node **:wrongInvoice** in the following RDF graph:

```
:wrongInvoice :payment _:1, _:2 .
_:1 :amount 3 .
_:2 :amount 3 .
```

would conform to **<Invoice>** having two payments, which may not be the intended behavior. As can be seen, it may be risky to combine cardinality constraints with SHACL property paths that don't terminate in unique values.

SHACL also added several built-in components that allow to constrain the values identified by two property constraints comparing if they are equal, disjoint, less-than, etc.

Example with SHACL property path comparisons

The following code declares that the values of **:firstName** and **:givenName** must be equal, and that the values of **:birthDate** must be lower than the values of **:loginDate**.

```
<User> a sh:NodeShape ;
 sh:property [
   sh:path     :firstName ;
   sh:equals   :givenName ;
 ] ;
 sh:property [
   sh:path     :birthDate ;
   sh:lessThan :loginDate
 ] .
```

This feature is currently not supported by ShEx, although there are some proposals to extend the language including it.

- **Repeated properties.** The ShEx abstract syntax discriminates shape expressions from triple expressions. Triple expressions define the neighbourhood of a node in a grammar-like way, taking into account repeated properties, while shape expressions act as sets of constraints. SHACL does not have the concept of triple expressions and the constraint components are conjunctive. In this way, repeated properties in SHACL are conjunctive, which may require some special care.

Example with repeated properties

The following ShEx schema declares that a product must have two codes, one with a string value and another with an integer value.

```
<Product> {
 :code xsd:string ;
 :code xsd:integer ;
}
```

A erroneously simple translation to SHACL might be:

```
<Product> a sh:NodeShape ;
  sh:property [ sh:path :code ;
   sh:minCount 1 ; sh:maxCount 1;
   sh:datatype xsd:string
  ] ;
  sh:property [ sh:path :code ;
   sh:minCount 1 ; sh:maxCount 1;
   sh:datatype xsd:integer
  ] ;
.
```

which is not the intended meaning. That expression would be represented in ShEx as:

```
<Product> {
 :code xsd:string ;
} AND {
 :code xsd:integer ;
}
```

and means that there must be exactly one property **`:code`** whose value must be string an integer at the same time. In order to correctly express it in SHACL the repeated properties would need to be qualified.

- **Extension mechanism.** The extension mechanism of ShEx is based on semantic actions while SHACL is based on SHACL-SPARQL. We leave these extension mechanisms out-of-the scope of this chapter and focus only on the core features of the languages: ShEx without semantic actions and SHACL-core.

As can be seen from the previous list, there are significant differences between both languages so it may be difficult to integrate them in the near future. However,

in the next section, we present a unified language that captures the main features of both ShEx and SHACL.

3.4 Language $\mathcal{S}$

Language $\mathcal{S}$ is a simple language that captures the essence of both ShEx and SHACL. The language-$\mathcal{S}$ defines a shape ϕ as:

ϕ	::=	$\top$	true
	|	$@l$	reference to shape with label l
	|	$datatype(iri)$	node has datatype iri
	|	IRI	node is an IRI
	|	BNode	node is a blank Node
	|	$\phi_1 \wedge \phi_2$	conjunction of shape ϕ_1 and shape ϕ_2
	|	$\neg\phi$	negation of shape ϕ
	|	$\_ \xrightarrow{p} \phi\{min, max\}$	between min and max routes with property path p whose values conform to shape ϕ
min	::=	integer	minimal cardinality
max	::=	integer	maximal cardinality or
	|	$*$	unbounded

A $\mathcal{S}$-Schema is defined as a pair (L, δ) where L is a set of labels and δ is a function that associates a shape ϕ for each label $l \in L$.

This language is a combination between the SHACL abstract syntax defined in [26] as language $\mathcal{L}$ and the Shapes-constraint language defined in [27], the main differences with these languages are:

- We added a explicit reference symbol $@l$ to declare a reference to a shape identified by label l.
- We define three primitive constraints: IRI to check that a node is an IRI, BNode to check that a node is a blank node, and $datatype(iri)$ to check that a node is a literal with some datatype identified by iri.
- The expression $\geq_n r.\phi$ from $\mathcal{L}$ which was satisfied if there were n or more arcs with property path r conforming to ϕ has been generalized to the expression $\_ \xrightarrow{p} \phi\{min, max\}$ which is satisfied whenever there are between min and max routes defined by property path p that conform to shape ϕ.

  The expression $\_ \xrightarrow{p} \phi\{min, max\}$ represents qualified routes (similar to the `sh:qualifiedValueShape` constraint component in SHACL) which are not closed as in ShEx, i.e. there can be values of properties $p \in p$ which don't conform to ϕ as long as the values that do conform are between min and max.
- By default, the shapes in $\mathcal{S}$ are open, i.e. there can be other properties apart of the specified in the shape declaration.
- We have omitted the expression $r_1 = r_2$ that compares values of properties for simplicity.

Example of an $\mathcal{S}$ expression

The following $\mathcal{S}$-schema, declares that nodes conforming to shape **<Product>** must be IRIs and must have exactly one property **:code** with a value of datatype **xsd:string**, they can zero or more properties **:seeAlso** whose values conform to shape **<Product>** and they can not have other properties different from **:code** or **:seeAlso** (cardinality $\{0,0\}$).

$$\begin{aligned} Product \mapsto\ & \text{IRI} \wedge \\ & \sqcup \xrightarrow{\text{:code}} datatype(\text{xsd:string})\{1,1\} \wedge \\ & \sqcup \xrightarrow{:seeAlso} @Product\{0,*\} \wedge \\ & \sqcup \xrightarrow{!\{\text{:code,:seeAlso}\}} \top\{0,0\} \end{aligned} \tag{1}$$

The semantics of $\mathcal{S}$ can be defined using a 3-valued logic following the stable reasoning approach[28]: a formula may be true or false, but there are two kinds of truth: certain truth (denoted by 2) and truth-by-default (denoted by 1). False is denoted by 0.

Table 3.4 presents an inductive definition of the semantics of $\mathcal{S}$ where $[\![\phi]\!]^{n,g,\sigma}$ denotes the value of shape ϕ for node n in graph g with regards to the schema σ. The table uses two auxiliary definitions. The first one counts the number of routes according to a property path p departing at node n in graph g for which the evaluation of target node returns a value v.

$$\#_{p,v}^{n,\phi,g,\sigma} = |(n,t) \in [\![p]\!]^g \mid [\![\phi]\!]^{t,g,\sigma} = v| \tag{2}$$

The second one counts all the routes departing at node n with property path p in graph g.

$$\#_{p}^{n,g} = |(n,t) \in [\![p]\!]^g| \tag{3}$$

Notice that the semantics is recursive and a naive implementation of an interpreter based on this definition can create an infinite loop when validating recursive shape declarations.

3.5 From SHACL to $\mathcal{S}$

Given that $\mathcal{S}$ is a generalization of the language presented in [26], the translation from SHACL to $\mathcal{S}$ is similar to the translation presented in appendix 1.2 of [29].

$$
\begin{aligned}
[\![\top]\!]^{n,g,\sigma} &= 2 \\
[\![@l]\!]^{n,g,\sigma} &= [\![\sigma(l)]\!]^{n,g,\sigma} \\
[\![datatype(iri)]\!]^{n,g,\sigma} &= \begin{cases} 2 & \text{if } n \text{ has datatype } iri \\ 0 & \text{otherwise} \end{cases} \\
[\![\text{IRI}]\!]^{n,g,\sigma} &= \begin{cases} 2 & \text{if } n \text{ is an IRI} \\ 0 & \text{otherwise} \end{cases} \\
[\![\text{BNode}]\!]^{n,g,\sigma} &= \begin{cases} 2 & \text{if } n \text{ is a blank node} \\ 0 & \text{otherwise} \end{cases} \\
[\![\phi_1 \wedge \phi_2]\!]^{n,g,\sigma} &= min([\![\phi_1]\!]^{n,g,\sigma}, [\![\phi_2]\!]^{n,g,\sigma}) \\
[\![\neg\phi]\!]^{n,g,\sigma} &= \begin{cases} 2 & \text{if } [\![\phi]\!]^{n,g,\sigma} = 0 \\ 0 & \text{otherwise} \end{cases} \\
[\![\_ \xrightarrow{p} \phi\{min, max\}]\!]^{n,g,\sigma} &= \begin{cases} 2 & \text{if } \#_{p,2}^{\phi,v,g,\sigma} \geq min \wedge \#_{p}^{n,g} - \#_{p,0}^{\phi,n,g,\sigma} \leq max \\ 0 & \text{if } \#_{p}^{n,g} - \#_{p,0}^{\phi,n,g,\sigma} < min \wedge \#_{p,2}^{\phi,v,g,\sigma} > max \\ 1 & \text{otherwise} \end{cases}
\end{aligned}
$$

Table 4 Inductive definition of $[\![\phi]\!]^{n,g,\sigma}$ for shape ϕ, node n, graph g and $\mathcal{S}$-schema σ

For readability, we assume the following restrictions on the shapes graph which simplify the translation:

- Each node can be either node or a property shape, and is marked by its corresponding **sh:NodeShape** or **sh:PropertyShape** type declaration.
- Shapes are normalized so they contain at most one **sh:minCount**, **sh:maxCount** , **sh:qualifiedMinCount** or **sh:qualifiedMaxCount** declaration. There is only one non-qualified constraint component: **sh:nodeKind**, **sh:datatype**, **sh:node** or a qualified constraint component **sh:qualifiedValueShape** in each property shape.
- The conversion is focused on the structural constraints, ignoring target declarations, which can be converted to a data structure similar to ShEx shape maps.

The transformation from SHACL to $\mathcal{S}$ is straightforward except that in the case of non-qualified property shapes, it is necessary to close the property definitions adding a constraint (line 8) that declares that there can't be values which don't satisfy the property shape.

Example converting SHACL to $\mathcal{S}$

Following algorithm 1, the SHACL shapes graph:

```
<User> a sh:NodeShape ;
 sh:nodeKind sh:IRI ;
 sh:property [ sh:path :name ;
  sh:datatype xsd:string  ;
  sh:minCount 1; sh:maxCount 1
 ] ;
```

```
7  sh:property [ sh:path :knows ;
8   sh:node <User>  ;
9  ] .
```

would be converted to:

$$
\begin{aligned}
\texttt{<User>} \mapsto & \text{IRI} \wedge \\
& \sqcup \xrightarrow{\text{:name}} datatype(\text{xsd:string})\{1,1\} \wedge \\
& \sqcup \xrightarrow{\text{:name}} \neg datatype(\text{xsd:string})\{0,0\} \wedge \\
& \sqcup \xrightarrow{\text{:knows}} @\texttt{<User>}\{0,*\} \wedge \\
& \sqcup \xrightarrow{\text{:knows}} \neg @\texttt{<User>}\{0,0\}
\end{aligned}
\tag{4}
$$

3.6 From ShEx to $\mathcal{S}$

Algorithm 2 presents *Sx2s* a conversion between ShEx-schemas to $\mathcal{S}$-Schemas. It takes as input a ShEx-schema (L, f) and associates each label $l \in L$ to its the result of applying *sx2s*() to the shape expression identified by l. Most of the definitions are straightforward, with special care taken to closed or extra properties. In the case of closed triple expressions, it is necessary to add a constraint that limits the appearance of other properties not mentioned in the triple expression (line 10). The function *extraOrClosed* checks if a property is part of the EXTRA set to declare that it allows other values for it different from the shapes mentioned in the triple expression (line 20), otherwise, it limits the appearance of values not satisfying those shapes (line 22).

Example converting ShEx to $\mathcal{S}$

Following algorithm 2, the ShEx-schema:

```
1 <User> {
2  :name       xsd:string  ;
3  :knows      @<User>* ;
4 }
```

is converted to the $\mathcal{S}$-schema (4).

If we add the closed qualifier and an EXTRA definition as:

```
1 <User> CLOSED EXTRA :name {
2  :name       xsd:string  ;
3  :knows      @<User>* ;
```

Algorithm 1: SHACL to $\mathcal{S}$-schema: *Sh2s*

Input: A SHACL shapes graph S_g
Output: An $\mathcal{S}$-Schema

return $\forall n \in S_g \mid \langle n, \text{rdf:type, sh:NodeShape}\rangle \vee \langle n, \text{rdf:type, sh:PropertyShape}\rangle$
$\lambda n \rightarrow sh2s(n)$

defn $sh2s(e) = shape(n) \wedge \bigwedge_{p_s \in propShapes(n)} ps2s(p_s)$
defn $propShapes(n) = \{p_s \mid \langle n, \text{sh:property}, p_s\rangle \in S_g\}$
defn $ps2s(p_s) =$ **if** *qualifiedShape*(p_s)
  $\sqcup \xrightarrow{path(p_s)} shape(p_s)\{qminCard(p_s), qmaxCard(p_s)\}$
else
  $\sqcup \xrightarrow{path(p_s)} shape(p_s)\{minCard(p_s), maxCard(p_s)\} \wedge$
  $\sqcup \xrightarrow{path(p_s)} \neg shape(p_s)\{0,0\}$
defn $path(p_s) = \{p \mid \langle p_s, \text{sh:path}, p\rangle \in S_g\}$
defn $shape(n) = \begin{cases} @l & \text{if } \langle n, \text{sh:node}, l\rangle \in S_g \\ datatype(iri) & \text{if } \langle n, \text{sh:datatype}, iri\rangle \in S_g \\ \text{IRI} & \text{if } \langle n, \text{sh:nodeKind}, shIRI\rangle \in S_g \\ \text{BNode} & \text{if } \langle n, \text{sh:nodeKind, sh:BlankNode}\rangle \in S_g \\ @l & \text{if } \langle n, \text{sh:qualifiedValueShape}, l\rangle \in S_g \end{cases}$
defn $qualifiedShape(p_s) = \begin{cases} true & \text{if } \langle p_s, \text{sh:qualifiedValueShape}, \_\rangle \in S_g \\ false & \text{otherwise} \end{cases}$
defn $minCard(p_s) = \begin{cases} n & \text{if } \langle p_s, \text{sh:minCount}, n\rangle \in S_g \\ 0 & \text{otherwise} \end{cases}$
defn $maxCard(p_s) = \begin{cases} n & \text{if } \langle p_s, \text{sh:maxCount}, n\rangle \in S_g \\ * & \text{otherwise} \end{cases}$
defn $qminCard(p_s) = \begin{cases} n & \text{if } \langle p_s, \text{sh:qualifiedMinCount}, n\rangle \in S_g \\ 0 & \text{otherwise} \end{cases}$
defn $qmaxCard(p_s) = \begin{cases} n & \text{if } \langle p_s, \text{sh:qualifiedMaxCount}, n\rangle \in S_g \\ * & \text{otherwise} \end{cases}$

```
}
```

the $\mathcal{S}$ expression is:

$$
\begin{aligned}
&\text{<User>} \mapsto \text{IRI} \wedge \\
&\quad \sqcup \xrightarrow{\text{:name}} datatype(\text{xsd:string})\{1,1\} \wedge \\
&\quad \sqcup \xrightarrow{\text{:knows}} @\text{<User>}\{0,*\} \wedge \\
&\quad \sqcup \xrightarrow{\text{:name}} \neg datatype(\text{xsd:string})\{0,*\} \wedge \\
&\quad \sqcup \xrightarrow{\text{:knows}} \neg @\text{<User>}\{0,0\} \wedge \\
&\quad \sqcup \xrightarrow{\text{!\{:name,:knows\}}} \top\{0,*\}
\end{aligned}
\tag{5}
$$

Algorithm 2: ShEx schema to $\mathcal{S}$-schema: *Sx2s*

Input: A ShEx schema (L, f)
Output: An $\mathcal{S}$-Schema

return $\forall l \in L : l \rightarrow sx2s(f(l))$

defn $sx2s(s)$ = **match** e
  case s_1 **AND** s_2 $\Rightarrow$ $sx2s(s_1) \wedge sx2s(s_2)$
  case s_1 **OR** s_2 $\Rightarrow$ $sx2s(s_1) \vee sx2s(s_2)$
  case **NOT** s $\Rightarrow$ $\neg sx2s(s)$
  case $@l$ $\Rightarrow$ $@l$
  case **CLOSED**? (**EXTRA** $p_1 \ldots p_n$)? { te } $\Rightarrow$ **let**
    $s_1 = te2s(te))$
    s_2 = **if** ***CLOSED***
      $\_ \xrightarrow{!ps(te)} \top\{0, 0\}$
    else
      $\top$
    in $s_1 \wedge s_2 \wedge \bigwedge_{p \in ps(te)} extraOrClose(p, te, \{p_1 \ldots p_n\})$

defn $te2s(te)$ = **match** te
  case $te_1; te_2$ $\Rightarrow$ $te2s(te_1) \wedge te2s(te_2)$
  case $te_1 \mid te_2$ $\Rightarrow$ $te2s(te_1) \vee te2s(te_2)$
  case $\_ \xrightarrow{p} se\{min, max\}$ $\Rightarrow$ $\_ \xrightarrow{p} sx2s(se)\{min, max\}$

defn $s_1 \vee s_2 = \neg(\neg s_1 \wedge \neg s_2)$

defn $extraOrClosed(p, te, extras)$ = **if** $p \in extras$
  $\_ \xrightarrow{p} notShapes(p, te)\{0, *\}$
else
  $\_ \xrightarrow{p} notShapes(p, te)\{0, 0\}$

defn $notShapes(p, te) = \neg sx2s(\text{OR}_{s \in shapes^{te}(p)} s)$

4 Challenges

In this section, we identify some current challenges and trends related with RDF validation.

4.1 Negation, recursion and semantics

As we described in section 3.3 the approach followed by ShEx and SHACL with regards to negation and recursion is different. ShEx specification proposes an stratification-based semantics limiting the possible schemas to those that have no negative cyclic dependencies, while SHACL leaves recursive shapes out of the spec-

ification. Corman et al. proposed a SHACL semantics based on partial assignments to solve the problem [26]. In their paper, they present an abstract language which is similar to the $\mathcal{S}$-language presented in section 3.4 and they show that SHACL can be defined in terms of that language.

In this chapter we show that the $\mathcal{S}$-language can also be used as the target language for ShEx. We have deliberately omitted any restriction about combining negation and recursion in $\mathcal{S}$to allow further research on possible solutions.

One approach we are currently working on is to define the $\mathcal{S}$-language by conversion to Answer-Set Programming (ASP). We have already implemented a prototype that, given an RDF graph g, a $\mathcal{S}$-schema, and a shape map, generates an ASP encoding that can be run to obtain a result shape map with the validation results (see [24]). We consider that it is be possible to extend the semantics of ShEx or SHACL to handle recursion and negation using answer set programming and stable reasoning techniques.

4.2 Shapes Libraries and reusability

A traditional use case is to describe a library of shapes that can later be reused in different contexts. To that end, it is necessary to be able to reuse an already declared shape in another context by different authors.

Both ShEx and SHACL can compose one shape from another one by conjunction.

Basic example of shapes extension by conjunction

For example, one may define a **<Teacher>** shape as a conjunction of **<User>** and a new shape that declares that there one or more properties **:teaches** whose values have shape **<Course>** as:

```
<User> {
 :name       xsd:string  ;
 :knows      @<User>* ;
}
<Teacher> <User> AND {
 :teaches    @<Course>+
}
```

When there are repeated properties, composing by conjunction can be unintuitive as it restricts the values of those existing properties. ShEx is introducing a new keyword **extends** which takes into account the repeated properties and injects their values in the corresponding triple expressions.

Example using extension and repeated properties

The following definition declares **Book** as a product which has a property **:code** with an integer value.

```
<Product> {
 :code       xsd:string  ;
 :related    @<Product>* ;
}
<Book> extends <Product> {
 :code       xsd:integer
}
```

In this case, the definition of book would be equivalent to:

```
<Book> {
 :code       xsd:integer
 :code       xsd:string  ;
 :related    @<Product>* ;
}
```

If we had tried to use composition by-conjunction, it would not be possible that a node had shape **<Book>** as the value of property **:code** would be declared as the conjunction of integer and string. Likewise, if **<Product>** were **CLOSED**, the conjunction of **<Product>** and **<Book>** would be unsatisfiable.

5 Shapes and the Semantic Web Stack

The appearance of RDF validation languages as new technologies in the Semantic Web field needs to find a place to coexist with the already established ones like SPARQL, RDFS or OWL. Although shapes languages can replace SPARQL for the validation task, they are not intended to replace it for RDF querying. On the contrary, shapes definitions can be very useful for data portal documentation [11] and to drive SPARQL queries. Shapes can help with *subgraph extraction*, identifying subgraphs in large knowledge graphs. For example, one may be interested to extract all nodes that conform to some specific shape and shapes can drive the SPARQL queries which extract those nodes.

RDFS has traditionally been employed not only for inference but also for documenting RDF vocabularies. We expect that more and more vocabularies will gradually convert their documentation to shapes declarations which offer the ability to automatically check conformance of RDF data to those descriptions.

In the case of OWL and ontology languages, shapes languages have not been designed for inference and have a more low level focus. While an ontology engineer is usually focused on domain knowledge, a shapes designer is more focused on graphs, and their topology. Nevertheless, given that a shapes processor has the ability to check or *infer* if some node conforms to some shape, it may be possible to use shapes processors for some inference tasks. As an example, there is a SHACL

proposal to define rules in SHACL [30] which can be used to infer new triples from the asserted ones. It remains to be seen what role will this approach play with regards to other rule based proposals.

5.1 Data transformation

Although semantic web technologies offer a good environment to build new systems, there are a lot of previous projects outside the field. Taking that projects—often information silos—and integrating them into semantic technologies is a challenge that practitioners are facing nowadays.

It can be divided in two main fields: data transformation and data integration. Data transformation refers to the ability to convert data represented in a non semantic format to a semantic format without losing information nor semantics. On its side, data integration is not only how to translate data but how to integrate and reconcile them in a single source of information. Consistency and cohesion are fundamental in this topic.

There are a lot of works in transforming from XML to RDF following different techniques: using a mapping file between XML Schema and OWL to then convert from XML to RDF [31], using XML Schema for the mappings [32], mapping XML Schema to RDF Schema to then provide a mechanism to query RDF data over an XML file [33], using the procedure as in [33] but using DTDs [34], using XSLT [35], embedding XSLT into schemata definitions [36], using XSLT with SPARQL [37], and creating a new language XSPARQL which combines XQuery and SPARQL [38]. These procedures are also supported for other formats like: CSV [39] or relational databases [40],

Providing solutions to integrate different sources of information could promote migration and exploitation of data. This field is being addressed by some works like: RML [41] which extends R2RML to provide transformation and integration of heterogeneous data sources, and YARRRML [42] which follows the same philosophy as RML but is designed to be human-friendly.

Once we are able to transform and integrate data from different formats we need to know if the transformation is still valid and if the integration is valid against a business model. For that purpose, one possibility is to make a one-to-one transformation, like we have explored in XMLSchema2ShEx [43].

Although these kinds of solutions could be valid when transforming from only one format, they are not well fitted in an heterogeneous data sources environment. Firstly, because the input schema and the output schema could not be the same and because heterogeneous data sources could not share the same schemata.

Here come the challenge of mapping and merging technologies which must not only make possible to map and merge heterogeneous data but also take into account that the data must be valid and validated. Therefore, solutions that integrate both tasks of data management could offer an invaluable tool for Semantic Web practitioners.

5.2 Schema inference

The traditional way to introduce schema notions in Semantic Web environments is called the *schema first* approach [44]. This strategy consist on defining a priori the schema that the data should follow. The traditional language to describe the expected content of RDF graphs following a schema first strategy was RDFS. Both ShEx and SHACL can be used for such a purpose, which also allow to define data constraints for validation.

On the other hand, *schema last* approaches compute an already existing source for discovering the schema that has naturally emerged from the data. In XML, for example, there are techniques able to produce schemata in RelaxNG [45] or XML Schema [46]. Different approaches have been proposed to make schema inference over RDF sources in the last decade, aiming for different goals ranging from statistical meta-data extraction [47, 48, 49] using VoID descriptions [50], or more complex structural inferences, such as concept hierarchies [51], graph summarization [52], or development of relational schemata that fits most of the data [44].

Some works are already making inference over RDF graphs to produce ShEx schemata. In [53] the authors compute the English chapter of DBpedia to produce shapes using a compliant subset of ShEx associated to each class in DBpedia's Ontology. Another approach that has already been tackled is to learn SHACL-SPARQL for relations [54].

5.3 Validation, modelling and visualization

A common practice when documenting RDF vocabularies is draw a UML-like class diagram where the different classes are depicted with the possible properties and linked to related classes. Some examples are DCAT [55], the organization ontology [56], the RDF Data Cube [57]. These kind of UML-like diagrams are useful as they can represent the structure of RDF content in an intuitive way. Given that ShEx and SHACL can describe RDF data models, it is not surprising that there are some recent proposals to graphically represent shapes which are backed-up by ShEx or SHACL shapes. As an example, the RDFShape playground developed by the authors of this paper [58] can be used to visualize shapes schemas. The result of visualizing the ShEx schema presented in example 3.1 is depicted in figure 2. The tool translates ShEx-schemas to PlantUML[4] which are converted to SVG on-the-fly.

Another schema authoring strategy is to leverage the extensive user interface investment in existing UML tools. The uml-model[5] system parses UML (encoded as XMI) and exports ShEx.

---

[4] `http://plantuml.com`

[5] `https://github.com/ericprud/uml-model`

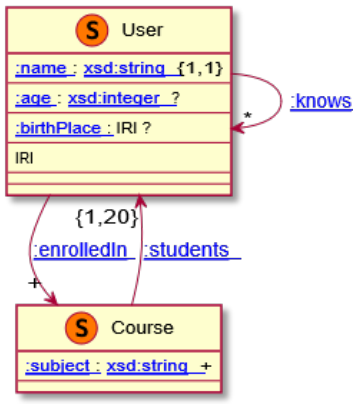

Fig. 2 Visualization of the ShEx Schema presented in example 3.1

We expect that further work is developed not only to visualize existing schemas, but also to edit them. In the case of SHACL, the TopBraid suite by TopQuadrant [6] offers SHACL support via a web-based editor and an integrated development environment. SHACL4P [59] has been implemented as a SHACL plugin for the Protégé editor. Also, the Eclipse Lyo project [7] is currently working on this direction using the Shaclex [24] library.

5.4 Validation usability

As we said in section 3.3, ShEx and SHACL had different priorities with regards to information about the validation results. While ShEx was more focused on information about which nodes were valid, SHACL was more focused on the violation errors. In this way, most ShEx implementations provide a result shape map which has information about the nodes/shapes associations that passed or failed. This information can be enriched with some implementation dependent information.

By contrast, the SHACL recommendation provides a detailed information about the different kinds of violation errors that can appear during validation, while it does not prescribe which information should be given for nodes that conform to shapes. In practice, most SHACL implementations do not provide any information about which nodes have been validated. Sometimes, it is not possible to know if an RDF graph is valid because all nodes pass the prescribing shapes, or it is reported as valid but no node has been selected by wrong target declarations.

We consider that further research must be done, both for improving the messages reported to users during validation, helping during the debugging phases or even to differentiate between different kinds of errors or validations. ShEx provides a generic annotation mechanism making possible to annotate some triple expressions

[6] `https://www.topquadrant.com/`

[7] `https://www.eclipse.org/lyo/`

about their severity. The Validata tool extended ShEx with the keywords: **MUST**, **MAY**, **SHOULD**, etc. [60] which could improve the visualization results. SHACL provides a simple mechanism to declare the severity of some shape with three built-in possibilities: **sh:Info**, **sh:Warning** and **sh:Violation**. However, the validation process is not affected by these severity declarations, and are mainly informative. We consider that further research must be done to improve the error messages provided to the end-user and to provide the shapes author to tailor the information that will be reported by the processors.

Another approach could be to define approximate validation algorithms based on probabilistic reasoning which could offer a more flexible experience where the results are not black or white but have some percentages. This approach has recently been tackled for probabilistic type systems in programming languages also [61].

5.5 Real time and streaming validation

The availability of sensors and similar devices that continually generate data to be processed on-the-fly has caused the emergent popularity of Stream Processing techniques. Stream reasoning extends these approaches with logical inference usually based on RDF data. Several initiatives have proposed to handle RDF streams like C-SPARQL [62], CQELS-QL [63], $SPARQL_{STREAM}$ [64]. The RDF Stream Processing (RSP) W3C community group specified an data model [65] which defines an RDF stream as a potentially unbounded sequence of RDF graphs with time-related metadata.

To our knowledge, most ShEx and SHACL implementations are based on an in-memory RDF graph which is validated against a shapes schema. Adapting this validation model to handle RDF streams poses several challenges.

- Validation of named graphs. Given that the RSP data model represents RDF streams as a sequence of named graphs with timestamps or similar metadata, it seems necessary to extend the RDF validation languages to support RDF datasets, i.e. collections of RDF graphs.
- Expressiveness. RDF stream validators may also provide new operators to take into account the RDF stream windows during validation. LARS [66] is a rule-based framework which extends ASP for stream reasoning seems an interesting approach that can match with the ASP implementation of the $\mathcal{S}$-language.
- Incremental validation. The practical application of validation in a streaming context may require to avoid the complete re-validate of an already validated graph, adopting incremental validation algorithms. In the LARS framework, Ticker [67] present the notion of tick streams that formally represent the aspects of an incremental stream reasoning system. The system uses two strategies, one using Clingo, and the other, truth maintenance techniques. It may be interesting to see if the ASP encoding that can be developed for the $\mathcal{S}$-language can be adapted to be used in Ticker.

- Performance of RDF stream validation. In order to offer real-time answer, it is necessary to validate the timestamped RDF graphs in a very efficient way. To that end, it may be necessary to identify less expressive ShEx or SHACL profiles with less expensive complexity.

6 Conclusions and future work

RDF validation has gained traction in the last years with the development of two technologies: ShEx and SHACL, which can be applied for it. Although both have similar goals, there are several differences and commonalities that must be understood in order to clarify in which use cases we should apply one or the other, or to offer guidelines about the future versions of the languages. We have presented the minimal language $\mathcal{S}$ which can represent both and we have shown two algorithms that convert ShEx and SHACL to $\mathcal{S}$. $\mathcal{S}$ can be used as an intermediate language in which ShEx or SHACL implementations can be based or compared. We have also identified several challenges that we consider of relevance and on which we are currently working on.

7 Acknowledgements

This work is partially funded by the Spanish Ministry of Economy and Competitiveness (Society challenges: TIN2017-88877-R)

References

1. Ora Lassila and Ralph R. Swick: Resource Description Framework (RDF) Model and Syntax Specification. https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/ (1999)
2. Brickley, D., Guha, R.V., Layman, A.: Resource description framework (RDF) schemas. https://www.w3.org/TR/1998/WD-rdf-schema-19980409/ (1998)
3. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler and FranÃğois Yergeau: Extensible Markup Language (XML) 1.0 (Fifth Edition. W3C Recommendation (2008)
4. Shudi Gao, C. M. Sperberg-McQueen and Henry S. Thompson: W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. W3C Recommendation (2012)
5. van der Vlist, E.: Relax NG: A Simpler Schema Language for XML. O'Reilly, Beijing (2004)
6. Deborah L McGuinness and Frank van Harmelen: OWL Web Ontology Language Overview. W3C Recommendation (2004)
7. Haase, P., Broekstra, J., Eberhart, A., Volz, R.: A comparison of rdf query languages. In McIlraith, S.A., Plexousakis, D., van Harmelen, F., eds.: The Semantic Web – ISWC 2004, Berlin, Heidelberg, Springer Berlin Heidelberg (2004) 502–517
8. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation (2008)
9. Berners-Lee, T.: Linked-data design issues. W3C design issue document (June 2006) http://www.w3.org/DesignIssue/LinkedData.html.

10. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. International Journal on Semantic Web and Information Systems **5**(3) (2009) 1–22
11. Labra Gayo, J.E., Prud'hommeaux, E., Solbrig, H.R., Rodríguez, J.M.Á.: Validating and describing linked data portals using RDF shape expressions. In: Proceedings of the 1st Workshop on Linked Data Quality co-located with 10th International Conference on Semantic Systems, LDQ@SEMANTiCS 2014. Volume 1215 of CEUR Workshop Proceedings., CEUR-WS.org (2014)
12. Hogan, A., Arenas, M., Mallea, A., Polleres, A.: Everything you always wanted to know about blank nodes. Web Semantics **27**(C) (August 2014) 42–69
13. Prud'hommeaux, E., Carothers, G.: RDF 1.1 turtle: Terse RDF triple language. http://www.w3.org/TR/turtle/ (2014)
14. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language. W3C Recommendation (2013)
15. Kostylev, E.V., Reutter, J.L., Romero, M., Vrgoč, D.: SPARQL with property paths. In: The Semantic Web - ISWC 2015. Springer International Publishing (2015) 3–18
16. Prud'hommeaux, E., Labra Gayo, J.E., Solbrig, H.: Shape expressions: an RDF validation and transformation language. In: Proceedings of the 10th International Conference on Semantic Systems, SEMANTICS 2014, ACM (2014) 32–40
17. Prud'hommeaux, E., Boneva, I., Labra Gayo, J.E., Kellog, G.: Shape Expressions Language 2.0. https://shexspec.github.io/spec/ (April 2017)
18. Boneva, I., Labra Gayo, J.E., Prud'hommeaux, E.: Semantics and validation of shapes schemas for rdf. In: International Semantic Web Conference. (2017)
19. Prud'hommeaux, E., Baker, T.: ShapeMap Structure and Language. https://shexspec.github.io/ShapeMap/ (July 2017)
20. Knublauch, H., Kontokostas, D.: Shapes Constraint Language (SHACL). W3C Proposed Recommendation (June 2017)
21. Knublauch, H.: SPIN - Modeling Vocabulary. http://www.w3.org/Submission/spin-modeling/ (2011)
22. Ryman, A.G., Hors, A.L., Speicher, S.: OSLC resource shape: A language for defining constraints on linked data. In Bizer, C., Heath, T., Berners-Lee, T., Hausenblas, M., Auer, S., eds.: Linked data on the Web. Volume 996 of CEUR Workshop Proceedings., CEUR-WS.org (2013)
23. Labra Gayo, J.E., Prud'hommeaux, E., Boneva, I., Kontokostas, D.: Validating RDF Data. Volume 7 of Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool Publishers LLC (sep 2017)
24. Labra Gayo, J.E.: Shaclex: Scala Implementation of ShEx and SHACL. DOI: 10.5281/zenodo.1400247 (2018) `http://labra.github.io/shaclex`.
25. Staworko, S., Boneva, I., Labra Gayo, J.E., Hym, S., Prud'hommeaux, E.G., Solbrig, H.R.: Complexity and Expressiveness of ShEx for RDF. In: 18th International Conference on Database Theory, ICDT 2015. Volume 31 of LIPIcs., Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2015) 195–211
26. Corman, J., Reutter, J.L., Savkovic, O.: Semantics and validation of recursive SHACL. In: Proceedings of the 17th International Semantic Web Conference. (October 2018)
27. Boneva, I.: Comparative expressiveness of ShEx and SHACL. Early working draft (March 2016)
28. Cabalar, P., Pearce, D., Valverde, A.: Stable reasoning. Journal of Applied Non-Classical Logics **27**(3-4) (2017) 238–254
29. Corman, J., Reutter, J.L., Savkovic, O.: Semantics and validation of recursive SHACL (extended version). Technical Report KRDB18-01, KRDB Research Centre (October 2018)
30. Knublauch, H., Allemang, D., Steyskal, S.: SHACL advanced features 1.1. W3C Draft Community Group Report (September 2018)
31. Deursen, D.V., Poppe, C., Martens, G., Mannens, E., de Walle, R.V.: XML to RDF Conversion: A Generic Approach. In Nesi, P., Ng, K., Delgado, J., eds.: 2008 International Conference on Automated solutions for Cross Media Content and Multi-channel Distribution., Florence, IEEE (November 2008) 138–144 doi: 10.1109/AXMEDIS.2008.17.

32. Battle, S.: Gloze: XML to RDF and back again. In: Proceedings of the First Jena User Conference, HP Labs, Bristol (May 2006)
33. Thuy, P.T.T., Lee, Y.K., Lee, S., Jeong, B.S.: Transforming valid XML documents into RDF via RDF schema. In Abraham, A., Han, S.Y., eds.: Third International Conference on Next Generation Web Services Practices (NWeSP 2007)., Seoul, IEEE (October 2007) 35–40 doi: 10.1109/NWESP.2007.23.
34. Thuy, P.T.T., Lee, Y.K., Lee, S., Jeong, B.S.: Exploiting XML schema for interpreting XML documents as RDF. In van der Aalst, W., Pu, C., Bertino, E., Feig, E., Hung, P.C.K., eds.: 2008 IEEE International Conference on Services Computing (SCC'08). Volume 2., Honolulu, IEEE (2008) 555–558 doi: 10.1109/SCC.2008.93.
35. Breitling, F.: A standard transformation from XML to RDF via XSLT. Astronomische Nachrichten **330**(7) (2009) 755–760 doi: 10.1002/asna.200811233.
36. Sperberg-McQueen, C.M., Miller, E.: On mapping from colloquial XML to RDF using XSLT. In: Proceedings of Extreme Markup Languages® 2004, Montreal (2004) http://conferences.idealliance.org/extreme/html/2004/Sperberg-McQueen01/EML2004Sperberg-McQueen01.html.
37. Berrueta, D., Labra Gayo, J.E., Herman, I.: XSLT + SPARQL: Scripting the semantic web with SPARQL embedded into XSLT stylesheets. In Bizer, C., Auer, S., Aastrand, G., Tom Heath, G., eds.: 4th Workshop on Scripting for the Semantic Web. Volume 368., Tenerife, CEUR-WS (June 2008)
38. Bischof, S., Decker, S., Krennwallner, T., Lopes, N., Polleres, A.: Mapping between RDF and XML with XSPARQL. Journal on Data Semantics **1**(3) (2012) 147–185 doi: 10.1007/s13740-012-0008-7.
39. Ermilov, I., Auer, S., Stadler, C.: CSV2RDF: User-driven CSV to RDF mass conversion framework. In: Proceedings of the ISEM. Volume 13., Graz, Austria (2013) 04–06
40. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language. W3C Recommendation (September 2012)
41. Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In: LDOW, Seoul, Korea (2014)
42. Heyvaert, P., De Meester, B., Dimou, A., Verborgh, R.: Declarative Rules for Linked Data Generation at your Fingertips! In: Proceedings of the 15th ESWC: Posters and Demos, Heraklion, Greece (2018)
43. Garcia-Gonzalez, H., Labra-Gayo, J.E.: XMLSchema2ShEx: Converting XML validation to RDF validation. Semantic Web (2018) In press: http://www.semantic-web-journal.net/content/xmlschema2shex-converting-xml-validation-rdf-validation-1.
44. Pham, M.D., Boncz, P.: Exploiting emergent schemas to make rdf systems more efficient. In: International Semantic Web Conference, Springer (2016) 463–479
45. Kim, G.H., Ko, S.K., Han, Y.S.: Inferring a relax ng schema from xml documents. In: International Conference on Language and Automata Theory and Applications, Springer (2016) 400–411
46. Klempa, M., Kozak, M., Mikula, M., Smetana, R., Starka, J., Švirec, M., Vitásek, M., Nečaskỳ, M., Mlỳnková, I.H.: Jinfer: A framework for xml schema inference. The Computer Journal **58**(1) (2015) 134–156
47. Rietveld, L., Beek, W., Hoekstra, R., Schlobach, S.: Meta-data for a lot of lod. Semantic Web **8**(6) (2017) 1067–1080
48. Hasnain, A., Mehmood, Q., e Zainab, S.S., Hogan, A.: Sportal: Profiling the content of public sparql endpoints. International Journal on Semantic Web and Information Systems (IJSWIS) **12**(3) (2016) 134–163
49. Mihindukulasooriya, N., Poveda-Villalón, M., García-Castro, R., Gómez-Pérez, A.: Loupe-an online tool for inspecting datasets in the linked data cloud. In: International Semantic Web Conference (Posters & Demos). (2015)
50. Alexander, K., Cyganiak, R., Hausenblas, M., Zhao, J.: Describing linked datasets. In: LDOW. (2009)

51. González, L., Hogan, A.: Modelling dynamics in semantic web knowledge graphs with formal concept analysis. In: Proceedings of the 2018 World Wide Web Conference on World Wide Web, International World Wide Web Conferences Steering Committee (2018) 1175–1184
52. Čebirić, Š., Goasdoué, F., Manolescu, I.: Query-oriented summarization of rdf graphs. Proceedings of the VLDB Endowment **8**(12) (2015) 2012–2015
53. Fernández-Álvarez, D., García-González, H., Frey, J., Hellmann, S., Labra Gayo, J.E.: Inference of latent shape expressions associated to dbpedia ontology. In: International Semantic Web Conference, Springer (2018)
54. Melo, A., Paulheim, H.: Learning SHACL Constraints for Validation of Relation Assertions in Knowledge Graphs. In: Extended Semantic Web Conference ESWC. (2018)
55. Maali, F., Erickson, J., eds.: Data Catalog Vocabulary (DCAT). W3C Recommendation (2014)
56. Reynolds, D.: The Organization Ontology. W3C Recommendation (2014)
57. Cyganiak, R., Reynolds, D.: The RDF Data Cube Vocabulary. W3C Recommendation (2014)
58. Labra Gayo, J.E.: RDFShape: RDF Playground. DOI: 10.5281/zenodo.1412128 (2018) `http://rdfshape.weso.es`.
59. Ekaputra, F.J., Lin, X.: Shacl4p: Shacl constraints validation within protÃľgÃľ ontology editor. In: 2016 International Conference on Data and Software Engineering (ICoDSE). (Oct 2016) 1–6
60. Gray, A.J.G.: Validata: A tool for testing profile conformance. In: Smart Descriptions & Smarter Vocabularies (SDSVoc), Amsterdam, The Netherlands (November 2016)
61. Boston, B., Sampson, A., Grossman, D., Ceze, L.: Probability type inference for flexible approximate programming. SIGPLAN Not. **50**(10) (October 2015) 470–487
62. Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: C-SPARQL: A continuous query language for RDF data streams. International Journal of Semantic Computing **04**(01) (mar 2010) 3–25
63. Le-Phuoc, D., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: The Semantic Web – ISWC 2011. Springer Berlin Heidelberg (2011) 370–388
64. Calbimonte, J.P., Jeung, H., Corcho, O., Aberer, K.: Enabling query technologies for the semantic sensor web. International Journal on Semantic Web and Information Systems **8**(1) (jan 2012) 43–63
65. Jean-Paul Calbimonte, ed.: RDF Stream Processing: Requirements and Design Principles. W3C Draft Community Group Report (2016)
66. Beck, H., Dao-Tran, M., Eiter, T.: LARS: A Logic-Based Framework for Analytic Reasoning over Streams. Technical Report INFSYS RR-1843-17-03, Institute of Information Systems, TU Vienna (October 2017)
67. Beck, H., Eiter, T., Folie, C.: Ticker: A system for incremental asp-based stream reasoning. TPLP **17**(5-6) (2017) 744–763