Semantic Web 0 (0) 1
IOS Press

XMLSchema2ShEx: Converting XML validation to RDF validation

Editor(s): Axel Polleres, Vienna University of Economics and Business (WU Wien), Austria
Solicited review(s): Felix Sasaki, Cornelsen Verlag GmbH, Germany; Emir Muñoz, National University of Ireland Galway, Ireland; Simon Steyskal, Vienna University of Economics and Business (WU Wien), Austria

Herminio Garcia-Gonzalez [a,*], Jose Emilio Labra-Gayo [a]
[a] *Department of Computer Science, University of Oviedo, Oviedo, Asturias, Spain*
Email: herminiogg@gmail.com, labra@uniovi.es

Abstract. RDF validation is a field where the Semantic Web community is currently focusing attention. Besides, there is a recent trend to migrate data from different sources to semantic web formats. Therefore, in order to facilitate this transformation, we propose: a set of mappings that can be used to convert from XML Schema to Shape Expressions (ShEx), a prototype that implements a subset of the proposed mappings, an example application to obtain a ShEx schema from an XML Schema and a discussion on conversion implications of non-deterministic schemata. We demonstrate that an XML and its corresponding XML Schema are still valid when converted to their RDF and ShEx counterparts. This conversion, along with the development of other format mappings, could drive to an improvement of data interoperability due to the reduction of the technological gap.

Keywords: ShEx, XML Schema, Shape Expressions, formats mapping, data validation

1. Introduction

Data validation is a key area when normalisation and confidence are desired. Normalisation—which can be defined, in this context, as using an homogeneous schema or structure across different sources of similar information—is desired as a way of making a dataset more reliable and even more useful to possible consumers because of its standardised schema. Validation can excel data cleansing, querying and standardisation of datasets. In words of P.N. Fox et al. [16]: *"Procedures for data validation increase the value of data and the users' confidence in predictions made from them. Well-designed data management systems may strengthen data validation itself, by providing better estimates of expected values than were available previously."*. Therefore, validation is a key field of data management.

XML Schema [5] was designed as a language to make XML validation possible with more expressiveness than DTDs [4]. Using XML Schema developers can define the structure, constraints and documentation of an XML vocabulary. Besides DTD and XML Schema, other alternatives for XML validation (such as Relax NG [11] and Schematron [18]) were proposed.

In the Semantic Web, RDF was missing a standard constraints validation language which covers the same features that XML Schema does for XML. Some alternatives were OWL [17] and RDF Schema [10]; however, they do not cover completely what XML Schema does for XML [38]. For this purpose, Shape Expressions (ShEx) [32,33] was proposed to fulfill the requirement of a constraints validation language for RDF, and SHACL [20] (another proposed language for RDF validation) has recently become a W3C recommendation.

As many documents and data are persisted in XML, the need for migration and interoperability to more

*Corresponding Author. Email: herminiogg@gmail.com

flexible data is nowadays more pressing than ever, many authors have proposed conversions from XML to RDF [27,12,2,6], with the goal of transforming XML data to Semantic Web formats.

Although these conversions enable users to migrate their data to Semantic Web, means for validating the output data after converting XML to RDF are missing. Therefore, we should ensure that the conversion has been done correctly and that both versions—in different languages—are defining the same meaning.

Conversions between XML and RDF, and between XML Schema and ShEx are necessary to alleviate the gap between semantic technologies and more traditional ones (e.g., XML, JSON, CSV, relational databases). With that in mind, providing generic transformation tools from non-semantic technologies to semantic technologies can enhance the migration possibilities; in other words, if we can create tools that ease the transformation and adaptation among technologies we will encourage future migrations. Taking Text Encoding Initiative (TEI) [14] as an example, digital humanities can take benefit from Semantic Web approaches [37,35]. There are many manuscripts transcribed to XML—using TEI—that can be converted to RDF. But transcribers are hesitant to deal with the underlying technology although they can benefit from it [26]. Those are the cases where generic approaches, as the one introduced here, can offer a solution and where automatic conversion of schemata has its place when transformations are to be checked.

Taking into account what we previously presented, the questions that we want to address in the present work are the following:

- RQ1: What components should have a mapping from XML Schema to ShEx?
- RQ2: How to ensure that both schemata are equivalent?
- RQ3: Is it possible to ensure a backwards conversion in all cases?
- RQ4: Are non-deterministic schemata (i.e., ambiguous schemata) possible to translate and validate?

In this paper, we describe a solution on how to make the conversion from XML Schema to ShEx. We describe how each element in XML Schema can be translated into ShEx. Moreover, we present a prototype that can convert a subset of what is defined in the following sections.

The rest of the paper is structured as follows: Section 2 presents the background; Section 3 gives a brief introduction to ShEx; Section 4 describes a possible set of mappings between XML Schema and ShEx; Section 5 presents a prototype used to validate a subset of previously presented mappings and how this conversion works against existing RDF validators; Section 6 discusses the implications of Non-Deterministic schemata on our work. Finally, Section 7 draws some conclusions and future lines of work and improvement.

2. Background

The related work of XML ecosystem conversion can be divided in three main categories: conversions from XML to Semantic Web formats, conversions from XML schemata to non Semantic Web schemata and conversions from XML schemata to RDF schemata.

2.1. From XML to Semantic Web formats

Along with schemata conversions, data transformation has to be tackled. Therefore many authors have worked on this topic of converting from XML to Semantic Web formats and more specifically to RDF. For this conversions there are plenty of strategies that have been proposed and followed by other authors.

In [27], authors describe their experience on developing this transformation for business to business industry in the case of the Semantic Mediation tools. An XML Schema to RDF Schema transformation is performed as part of the requirement of the Semantic Mediation tool.

In [12], a transformation between XML and RDF depending on an ontology is described. This transformation takes an XML document, a mapping document and an ontology document and makes the transformations to RDF instances compliant with the input ontology. Using the mapping file, conversions between the XML Schema and the ontology are established.

In [1], the author explains how XML can be converted to RDF—and vice versa—using XML Schema as the base for the mappings. This work is then expanded in [2] where the author tries to solve the lift problem (the problem of how to map heterogeneous data sources in the same representational framework) from XML to RDF and backwards by using the Gloze mapping approach on top of Apache Jena.

In [40], the authors present a mechanism to query XML data as RDF. Firstly, a matching from XML Schema to RDF Schema class hierarchy is performed. Then XML elements can be interpreted as RDF triples.

The same procedure but using DTDs is described in [39].

In [9], the author presents a technique for making standard transformations between XML and RDF using XSLT. A case study in the field of astronomy is used to illustrate the solution.

Another approach using XSLT is [36] where authors describe a mapping mechanism using XSLT that can be attached to schemata definition.

In [3], a transformation from RDF to other kind of formats, including XML, is proposed using in XSLT stylesheets embedded SPARQL which by means of these extensions, could query, merge and transform data from the Semantic Web.

In [6], authors describe XSPARQL which is a framework that enables the transformation between XML and RDF based on XQuery and SPARQL and solves the disadvantages of using XSLT for these transformations.

However, these works (except [27]) are not covering the schemata mapping problem.

2.2. From XML schemata to other schemata

Although data migration is important, during this process it is desirable to transform the constraint rules or schemas too. This is also a way to verify that the transformations have been done correctly. Therefore, many authors have proposed different techniques and transformation from XML Schema.

In [29], a transformation from XML Schema to JSON Schema is proposed. These transformations are made using equivalent constraints when it is possible and concrete transformations when no equivalent constraints exists.

In [31], an algorithm that converts from XML Schemata to ER diagrams is proposed. This algorithm (called Xere mapping) is proposed as a part of the Xere technique to assist the integration of XML data.

In [24], the authors propose an algorithm to convert from a relational schema to an XML Schema and two algorithms to convert from a XML Schema to a relational schema. All these techniques preserve the structure and the semantics.

However, none of these works bring XML schemata to Semantic Web technologies.

2.3. From XML schemata to RDF schemata

In the Semantic Web community there has been an effort to convert XML schemata to OWL [15,34] and to RDF Schema [27]. Moreover, when no schema is available the transformation can be performed from XML to OWL [7,21,30,23].

However, RDF Schema and OWL were not designed as RDF validation languages. Their use of Open World and Non-Unique Name Assumptions can pose some difficulties to define the integrity constraints that RDF validation languages require [38].

2.4. FHIR approach

Another approach for transformation between schemas is to take a domain model as the main representation of data structure and constraints and then transform between that model and other schema formats like XML Schema, JSON Schema or ShEx. This has been the approach followed by FHIR[1]. However, this technique needs the creation of a domain model as an abstract representation which is not the goal of our work.

2.5. RDF validation languages and its conversions

Various languages have recently been developed for RDF validation. Shapes Constraint Language (SHACL) [20] has been developed by the W3C Data Shapes Working Group and Shape Expressions (ShEx) [33] is being developed by the W3C Shape Expressions Community Group.

To the best of our knowledge, no conversion between XML Schema and ShEx/SHACL has been proposed to date. This might be due to the recent introduction of ShEx and SHACL.

In this paper, ShEx is used to describe the mappings due to its compact syntax and its support for recursion whereas in SHACL recursion depends on the implementation. However, we consider that converting the mappings proposed in this paper to SHACL is feasible and can be an interesting line of future work given that it has already been accepted as a W3C recommendation and that there are some ways to simulate recursion by target declarations or property paths.

3. Brief introduction to ShEx

ShEx was proposed as a language for RDF validation in 2014 [33]. It was one of the foundations for the W3C Data Shapes Working Group which de-

[1] https://www.hl7.org/fhir/

veloped the Shapes Constraint Language (SHACL) for the same purpose. SHACL was also inspired by SPIN [19] and although both languages can perform RDF validation there are some differences between them like the support of recursion or the emphasis on validation versus constraint checking (see chapter 7 of [22] for more details). In this paper, we will focus on ShEx because it has a well-defined semantics for recursion [8] and its semantics are more inspired by grammar-based formalisms like Relax NG.

ShEx syntax was inspired by Turtle, SPARQL and Relax NG with the aim to offer a concise and easy to use syntax. In July 2017, version 2.0 was released together with a draft community group report and the community group is currently developing version 2.1.

ShEx uses shapes to group different validations associated with the same node 'type'. That is, a shape can define how a node and its triples should be in order to be valid. Listing 1 illustrates an example of a ShEx document defining a shape with a `:PurchaseOrder` type.

```
PREFIX : <http://example.com/>
PREFIX schema: <http://schema.org>
PREFIX
    xs: <http://www.w3.org/2001/XMLSchema#>

:PurchaseOrder {
  :orderId          /Order\\d{2}/ ;
  schema:customer   @:User ;
  schema:orderDate  xs:date ? ;
  schema:orderedItem @:Item +
}
:Item {
  schema:name xs:string ;
  :quantity xs:positiveInteger OR
            xs:integer MININCLUSIVE 1
}
:User {
  a [ schema:Person ] ;
  :purchaseOrder @:PurchaseOrder*
}
```

Listing 1: ShEx shape example

Prefixes are defined at the beginning of the snippet and use the same syntax as in Turtle. Triple constraints are defined inside the shape where a purchase order must have an `orderId` value that matches the regular expression `Order\d{2}`, it must have a `schema:customer` value which must be a node that conforms to shape `:User`, a `schema:orderDate` whose value must be of type `xs:date` and can have one or more (represented by the plus sign) `schema:orderedItem` whose values must conform to the `:Item` shape.

The `:Item` shape must have a `schema:name` value of type `xs:string` and a `:quantity` value of type `xs:positiveInteger`, while the `:User` shape declares that the values must have type `schema:Person`, and can contain zero or more values of `:purchaseOrder` which must conform to the `:PurchaseOrder` shape.

```
### Pass validation as :PurchaseOrder
:order1 :orderId      "Order23" ;
 schema:customer      :alice ;
 schema:orderDate     "2017-03-02"^^xs:date;
 schema:orderedItem   :item1 .
:alice a              schema:Person ;
       :purchaseOrder :order1 .
:item1 schema:name    "Lawn" ;
      :quantity       2 .

### Fails validation as :PurchaseOrder
:order2 :orderId    "MyOrder" ;
 schema:customer    :bob;
 schema:orderDate   2017;
 schema:orderedItem :item1 .
:bob a              schema:Person ;
    :purchaseOrder  :unknown.
```

Listing 2: RDF validation example

In Listing 2 there is an example of two purchase orders defined in RDF. The first one passes validation and conforms to the shapes declaration given in Listing 1 whereas `:order2` fails validation for several reasons: the value of `:orderId` does not conform to the required regular expression, the value of `schema:customer` does not conform to shape `:User` and the value of `schema:orderDate` does not have datatype `xs:date`.

ShEx supports different serialization formats:

- ShExC: a concise human readable compact syntax which is the one presented in previous example.
- ShExJ: a JSON-LD syntax which is used as an abstract syntax in the ShEx specification [32].
- ShExR: an RDF representation syntax based on ShExJ.

ShEx defines an extension mechanism through which users can embed portions of code written in a programming language or SPARQL. This feature is known as Semantic Actions and are introduced between definition of triples with the *%interpreter{}%* syntax where interpreter is the name of the interpreter

to be used (e.g., JS, SPARQL, JAVA). See Listings 22 and 23 for Semantic Actions examples.

In this paper, ShExC syntax was used because it is easy to read and understand. The goal of this introduction was to provide a basic understanding of ShEx. For more examples and a longer comparison between ShEx and SHACL readers can consult [22].

4. Mappings between XML Schema and ShEx

XML Schema defines a set of elements and datatypes for validation that need to be converted to ShEx. In this section, we describe different XML Schema elements and a possible conversion to ShEx. All examples use the default prefix `:` for URIs. It is intended to be replaced by different prefixes depending on the required namespaces. For XML Schema elements and datatypes `xs` prefix is used in the examples.

4.1. Element

Elements are treated as a triple predicate and object, i.e., we convert them to a triple constraint whose predicate is the name of the element:

```
### XML Schema
<xs:element name="birthday" type="xs:date"/>

### ShEx
:birthday xs:date ;
```

Listing 3: Element mapping

The `name` attribute is used as the fragment of the URI in the predicate and the type is transcribed directly, as ShEx has built-in support for XML Schema datatypes. If the `ref` attribute is present, the type must be defined somewhere in the document to link the corresponding type or shape. When an `xs:element` type is a `xs:complexType`, the type should be referenced to a new shape where the `xs:complexType` is converted (see Section 4.3 where we explain how to convert `xs:complexType` to a shape). See Listings 3, 4 and 5 for a list of examples on how to convert an element.

```
### XML Schema
<xs:element name="purchaseOrder"
            type="PurchaseOrderType"/>

<xs:complexType name="PurchaseOrderType">
    ...
</xs:complexType>

### ShEX
:purchaseOrder @<PurchaseOrderType> ;
```

Listing 4: Element mapping with linked type

```
### XML Schema
<xs:element name="item"
            minOccurs="0"
            maxOccurs="unbounded">
    <xs:complexType>
        ...
    </xs:complexType>
</xs:element>

### ShEx
:item @<item> * ;
```

Listing 5: Element mapping with nested type

As presented in Listing 5, when an element has its complex type nested the shape name will be the `name` of the element.

4.1.1. Cardinality

Cardinality in ShEx is defined with the following symbols: '*' for 0 or more repetitions, '+' for 1 or more repetitions, '?' for 0 or 1 repetitions (optional element) or '{m, n}' for m to n repetitions where m is `minOccurs` and n `maxOccurs`. As in XML Schema, the default cardinality in ShEx is 1 for lower and upper bounds. Therefore, transformation of `minOccurs` and `maxOccurs` in the previously defined cardinality marks is done as showed in Listing 6.

```
### XML Schema
<xs:element name="nameZeroUnbounded"
            type="xs:string"
            minOccurs="0"
            maxOccurs="unbounded">
<xs:element name="nameOneUnbounded"
            type="xs:string"
            minOccurs="1"
            maxOccurs="unbounded">
<xs:element name="nameOptional"
            type="xs:string"
            minOccurs="0"
            maxOccurs="1">
<xs:element name="nameFourToTen"
            type="xs:string"
            minOccurs="4"
            maxOccurs="10">

### ShEx
```

```
:nameZeroUnbounded xs:string * ;
:nameOneUnbounded xs:string + ;
:nameOptional xs:string ? ;
:nameFourToTen xs:string {4, 10} ;
```

Listing 6: Cardinality mapping

4.2. Attribute

ShEx treats attributes like elements because it makes no difference between an attribute and an element. This difference is part of XML data model whereas the RDF data model does not have the concept of attributes. One possibility to transform attributes is to use their **name** and **type** as performed with elements (see Section 4.1). This allows better readability of the corresponding RDF data, but limits roundtrip conversions between XML to RDF and back.

4.3. ComplexType

Complex types are translated directly to ShEx shapes. The **name** of the **xs:complexType** will be the name of the shape to which elements can refer to (see Listing 7 for an example). Complex types consist of various statements, so we provide a detailed transformation of each possibility in the following sections.

```
### XML Schema
<xs:complexType name="PurchaseOrderType">
    ...
</xs:complexType>

### ShEx
<PurchaseOrderType> {
    ...
}
```

Listing 7: Complex type mapping

4.3.1. Sequence

While sequences in XML Schema define sequential order of elements, representing the same modeling in ShEx is complex due to RDF graph structure. There are several ways to represent order in RDF, the most obvious one is using RDF lists (cf., other ways to represent it [13,25]).

The example in Listing 8 shows how the mapping is done for a **xs:sequence** using RDF lists:

```
### XML Schema
<xs:complexType name="Address">
    <xs:sequence>
        <xs:element name="street"
            type="xs:string"/>
        <xs:element name="city"
            type="xs:string"/>
        <xs:element name="state"
            type="xs:string"/>
        <xs:element name="zip"
            type="xs:decimal"/>
    </xs:sequence>
</xs:complexType>

### ShEx
<address> {
    rdf:first @<street> ;
    rdf:rest @<i1> ;
}
<i1> {
    rdf:first @<city> ;
    rdf:rest @<i2> ;
}
<i2> {
    rdf:first @<state> ;
    rdf:rest @<i3> ;
}
<i3> {
    rdf:first @<zip> ;
    rdf:rest [ rdf:nil ] ;
}
<street> {
    :street xs:string ;
}
<city> {
    :city xs:string ;
}
<state> {
    :state xs:string ;
}
<zip> {
    :zip xs:decimal ;
}
```

Listing 8: Sequence mapping

4.3.2. Choice

Choices in XML Schema are the disjunction operator to select between two options, for instance: choice between two elements. This operator is supported in ShEx using the *oneOf* operator ('|'). The object and predicate of the RDF statement must be one of the enclosed ones. Therefore, translation is performed as shown in the snippet of Listing 9:

```
### XML Schema
<xs:choice>
```

```
  <xs:element name="name"
              type="xs:string"/>
  <xs:all>
    <xs:element name="givenName"
                type="xs:string"
                maxOccurs="unbounded"/>
    <xs:element name="familyName"
                type="xs:string" />
  </xs:all>
</xs:choice>

### ShEx
( :name xs:string |
  :givenName xs:string + ;
  :familyName xs:string
) ;
```

Listing 9: Choice mapping

4.3.3. All

While sequences are an ordered set of elements, `xs:all` is instead a set of unordered elements. Indeed, `xs:all` has a better representation using ShEx elements and the transformation is simpler than the `xs:sequence` one as there is no need to keep track of the order of elements. See Listing 10 for an example.

```
### XML Schema
<xs:all>
    <xs:element name="street"
                type="xs:string"/>
    <xs:element name="city"
                type="xs:string"/>
    <xs:element name="state"
                type="xs:string"/>
    <xs:element name="zip"
                type="xs:decimal"/>
</xs:all>

### ShEx
:street  xs:string  ;
:city    xs:string  ;
:state   xs:string  ;
:zip     xs:decimal ;
```

Listing 10: All mapping

4.4. XSD Types

XSD Types can be used in ShEx as they are used on XML Schema, e.g., whenever a string type is required we can use `xs:string`. Therefore, translation is done directly using the same types that are defined in the XML Schema document.

4.4.1. Enumerations (using NMTokens)

Enumerations in XML Schema can be used to declare the possible values that an element can have. In ShEx, this is supported using the symbols '[' and ']'. The enclosed values are the possible values that the RDF object can take. See Listing 11 for an example.

```
### XML Schema
<xs:simpleType name="PublicationType">
   <xs:restriction base="xs:NMTOKEN">
     <xs:enumeration value="Book"/>
     <xs:enumeration value="Magazine"/>
     <xs:enumeration value="Journal"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="pubType"
            ref="PublicationType"/>
<xs:attribute name="country"
              type="xs:NMTOKEN"
              fixed="US"/>

### ShEx
:pubType ["Book" "Magazine" "Journal"] ;
:country ["US"] ;
```

Listing 11: Enumarations (using NMTokens) mapping

4.4.2. Pattern

`xs:pattern` is used in XML Schema to define the format and allowed contents of a string value. `xs:pattern` in ShEx uses a syntax similar to the JavaScript language except that backslash is required to be escaped, i.e., double backslash has to be used to correctly escape. Therefore, the conversion is a transformation between XML Schema and JavaScript Regular Expression syntaxes as shown in Listing 12.

```
### XML Schema
<xs:simpleType name="SKU">
  <xs:restriction base="xs:string">
    <xs:pattern value="\d{3}-[A-Z]{2}"/>
  </xs:restriction>
</xs:simpleType>
<xs:attribute name="partNum"
              type="SKU"
              use="required"/>

### ShEx
:partNum /\\d{3}-[A-Z]{2}/ ;
```

Listing 12: Pattern mapping

4.5. SimpleType

Simple types in XML Schema are based on XSD Types (see Section 4.4) and allow some enhancements like: restrictions, lists and unions. Depending on the content, translation is performed following different strategies which we detail bellow. For translation of restrictions, see Section 4.7.

4.5.1. List

Lists inside simple types define a way of creating collections of a base XSD type in XML Schema. These lists are supported in RDF using RDF Collections[2]. As previously discussed, there can be several approaches to represent ordered lists in RDF (see Section 4.3.1). A commonly accepted approach is the use of RDF lists: the `rdf:first` edge points to the first element and the `rdf:rest` edge to the rest of the list which recursively follows the same structure until the `rdf:nil` element is declared to represent the end of the list. This way, it is possible to create the desired list and preserve the order. Figure 1 shows how an RDF list is constructed for a better understanding of this section. Hence, translation into ShEx is made by using RDF lists and the use of recursion that defines a type with a pointer to itself in the `rdf:rest` edge. See Listing 13 for an example.

```
### XML Schema
<xs:simpleType name="IntegerList">
  <xs:list itemType="xs:integer" />
</xs:simpleType>

### ShEx
<IntegerList> {
 rdf:first xs:integer ;
 rdf:rest @<IntegerList> OR [rdf:nil];
}
```

Listing 13: List mapping

4.5.2. Union

Unions are the mechanism that XML Schema offers to make new types that are the combination of two simple types. With this kind of disjunction, a new type which allows any value admitted by any of the members of the `xs:union` is created. For the translation into ShEx we create a new type that is the combination of the types involved in the `xs:union` as shown in Listing 14.

[2] https://www.w3.org/TR/rdf11-mt/#rdf-collections

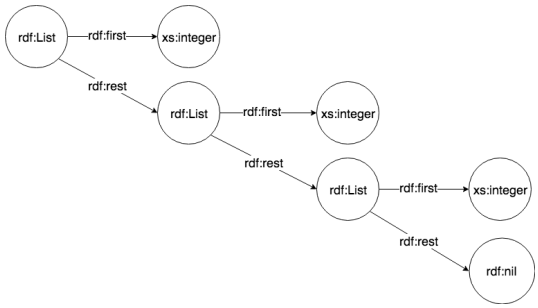

Fig. 1. Example of a RDF list construction

```
### XML Schema
<xs:attribute name="fontsize">
 <xs:simpleType>
   <xs:union memberTypes="Fontbynumber
                          Fontbystringname"
   />
    </xs:simpleType>
</xs:attribute>

<xs:simpleType name="Fontbynumber">
    <xs:restriction
        base="xs:positiveInteger">
        <xs:maxInclusive value="72"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="Fontbystringname">
    <xs:restriction base="xs:string">
        <xs:enumeration value="small"/>
        <xs:enumeration value="medium"/>
        <xs:enumeration value="large"/>
    </xs:restriction>
</xs:simpleType>

### ShEx
:fontsize
  @<Fontbynumber> OR @<Fontbystringname>

<Fontbynumber>
   xs:positiveInteger MAXINCLUSIVE 72

<Fontbystringname> ["small"
                    "medium"
                    "large"
                   ]
```

Listing 14: Union mapping

4.6. Complex Content and Simple Content

Complex contents and simple contents are a way to define a new type from a base type using restrictions or extensions. The base type is the one that is

used as a base for the restriction (or extension) clause and the new type is the one that is been restricted (or extended). Complex content allows to extend or restrict a base `xs:complexType` with mixed content or elements only. Simple content allows to extend or restrict a `xs:complexType` with character data or with a `xs:simpleType`. For the translation into ShEx, the respective `xs:restriction` or `xs:extension` have to be taken into account to define the new type.

4.6.1. Restriction

Restrictions are used in XML Schema to restrict possible values of a base type. A new type can be defined using restrictions applied to a base type. Depending on how the type and the restrictions are defined, the translation strategies vary.

- Simple Content: If `xs:simpleContent` is present, XSD Facets/Restrictions must be used (see Section 4.7 for more information). When restricting using a `xs:simpleType`, the transformation is done using the known base type (see Section 4.4) and putting some format restrictions to it. Translation into ShEx will be performed using the base type and translating the XSD Facets as they are defined in every specific case (see Section 4.7).
- Complex Content: If `xs:complexContent` is present, the base `xs:complexType` is restricted using `xs:all`, `xs:group`, `xs:choice`, `xs:sequence`, `xs:attribute` or `xs:attributeGroup`. Complex content restriction will restrict allowed values and elements types. This is a case of inheritance by restriction. For translation into ShEx, the `xs:restriction` elements must be taken and transformed directly into a new shape that defines the resulting child shape[3].

4.6.2. Extension

With extensions in XML Schema, it is possible to define a new type as an extension of a previously defined one. This is a case of classic inheritance, where the child inherits its parent elements that are added to its own defined elements. Depending on the content, i.e., `xs:complexContent` or `xs:simpleContent`, different translation strategies can be used.

- Simple content: If `xs:simpleContent` is present, extension of the base type is performed by adding more attributes or attribute groups to the new type. Therefore, the translation into ShEx is made by the concatenation of both the type and its `xs:extension` to create the new shape.
- Complex content: If `xs:complexContent` is present, extension of base type is performed by adding more attributes and elements to a new base one. Therefore, translation is done by combining the base type and its `xs:extension` to create a new shape.

Restrictions and extensions in ShEx are not supported directly in the current version (i.e., ShEx has no support for extensions, restriction or inheritance) with the same semantics as XML Schema. Therefore, we use the normal syntax provided by ShEx and create the two resulting shapes—by solving the `xs:restriction` or `xs:extension` before the translation to ShEx—from the respective `xs:restriction` or `xs:extension` as can be seen in Listing 15. However, this translation suffers from a loss of semantics—which is in line with RQ3—which makes impossible a backwards conversion.

```
### XML Schema
<xs:simpleType name="mountainBikeSize">
  <xs:restriction base="xs:string">
    <xs:enumeration value="small" />
    <xs:enumeration value="medium" />
    <xs:enumeration value="large" />
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="FamilyMountainBikes">
  <xs:simpleContent>
    <xs:extension base="mountainBikeSize">
      <xs:attribute name="familyMember">
       <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="child" />
          <xs:enumeration value="male" />
          <xs:enumeration value="female" />
        </xs:restriction>
       </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

### ShEx
<MountainBikeSize> ["small" "medium" "large"]

<FamilyMountainBikes> {
 :mountainBikeSize @<MountainBikeSize> ;
 :familyMember ["child" "male" "female"];
}
```

[3] Future versions of ShEx are planning to include inheritance. See: https://github.com/shexSpec/shex/issues/50

Listing 15: Restrictions and extensions mapping, where extensions and restrictions are directly transformed into the equivalent shape

4.7. XSD Types Restrictions/Facets

4.7.1. Enumeration

`xs:enumeration` restriction uses a base type to restrict the possible values of a type. It is declared using a set of possible values. In ShEx, this is defined using the '[' and ']' operators. The values that are allowed are enclosed inside the square brackets. This is the same mechanism how the example in Section 4.5.1 works. However, Listing 16 shows a more complex example using extensions and restrictions.

```
### XML Schema
<xs:simpleType name="Mountainbikesize">
  <xs:restriction base="xs:string">
    <xs:enumeration value="small"/>
    <xs:enumeration value="medium"/>
    <xs:enumeration value="large"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType
  name="FamilyMountainBikeSizes">
  <xs:simpleContent>
    <xs:extension base="mountainbikesize">
       <xs:attribute name="familyMember"
                     type="xs:string" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType
    name="ChildMountainBikeSizes">
  <xs:simpleContent>
     <xs:restriction
          base="FamilyMountainBikeSizes" >
        <xs:enumeration value="small"/>
        <xs:enumeration value="medium"/>
     </xs:restriction>
  </xs:simpleContent>
</xs:complexType>

### ShEx
<MountainBikeSize> ["small" "medium" "large"]

<FamilyMountainBikes> {
 :mountainBikeSize @<MountainBikeSize> ;
 :familyMember ["child" "male" "female"];
}
```

```
<ChildMountainBikeSizes>
   @<FamilyMountainBikes> AND {
   :mountainBikeSize ["small" "medium"]
}
```

Listing 16: Enumeration mapping

4.7.2. Fraction digits

`xs:fractionDigits` are used in XML Schema when a decimal type is defined (e.g., `xs:decimal`) and the number of decimal digits is desired to be restricted in the representation. ShEx supports this feature in a similar way as XML Schema. Hence, `FRACTIONDIGITS` keyword is used followed by the integer number of fraction digits that should be allowed. See Listing 17 for an example.

```
### XML Schema
<xs:element name="itemValue">
  <xs:simpleType>
    <xs:restriction base="xs:decimal">
      <xs:fractionDigits value="2"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

### ShEx
:itemValue xs:decimal FRACTIONDIGITS 2 ;
```

Listing 17: Fraction digits mapping

4.7.3. Total digits

This feature allows to restrict the total number of digits permitted in a numeric type. In ShEx, this is possible using `TOTALDIGITS` keyword as shown in Listing 18.

```
### XML Schema
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:totalDigits value="3"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

### ShEx
:age xs:integer
     TOTALDIGITS 3 ;
```

Listing 18: Total digits mapping

4.7.4. Length

`xs:length` is used to restrict the number of characters allowed in a string type. In ShEx, this is supported with the `LENGTH` keyword followed by the integer number that defines the desired length as shown in Listing 19.

```
### XML Schema
<xs:element name="group">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

### ShEx
:group xs:string LENGTH 1 ;
```

Listing 19: Length mapping

4.7.5. Max Length and Min Length

`xs:maxLength` and `xs:minLength` are used to restrict the number of characters allowed in a text type. But instead of restricting to a fixed number of characters, with these features restriction to a length interval is possible. In ShEx, the definitions of minimum and maximum length are made by using the `MINLENGTH` and `MAXLENGTH` keywords as shown in Listing 20.

```
### XML Schema
<xs:element name="comments">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="1000"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

### ShEx
:comment xs:string
         MINLENGTH 1
         MAXLENGTH 1000;
```

Listing 20: Max length and min length mapping

4.7.6. Max-min exclusive and max-min inclusive

These features allow restricting number types to an interval of desired values. This is the same notion as in open and closed intervals. In ShEx, these features are supported directly. Therefore, transformation is done as shown in Listing 21.

```
### XML Schema
<xs:element name="cores">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minExclusive value="0"/>
      <xs:maxExclusive value="9"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="coresOpenInterval">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1"/>
      <xs:maxInclusive value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

### ShEx
:cores xs:integer
       MINEXCLUSIVE 0
       MAXEXCLUSIVE 9 ;
:coresOpenInterval xs:integer
                   MININCLUSIVE 1
                   MAXINCLUSIVE 8 ;
```

Listing 21: Max exclusive, min exclusive, min inclusive and max inclusive mapping

4.7.7. Whitespace

`xs:whiteSpace` allows to specify how white spaces in strings are handled. In XML Schema, there are three options:

- Preserve: This option will not remove any white space character from the given string.
- Replace: This option will replace all white space characters (line feeds, tabs, spaces and carriage returns) with spaces.
- Collapse: This option will remove all white spaces characters:
  - Line feeds, tabs, spaces and carriage returns are replaced with spaces.
  - Leading and trailing spaces are removed.
  - Multiple spaces are reduced to a single space.

In ShEx, `xs:whiteSpace` options are not supported. Their behaviour could be simulated using semantic actions (see Listing 22).

```
### XML Schema
<xs:complexType name="whiteSpaces">
 <xs:all>
  <xs:element name="preserve">
    <xs:simpleType>
```

```
      <xs:restriction base="xs:string">
        <xs:whiteSpace
           value="preserve"/>
        </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="replace">
    <xs:simpleType>
       <xs:restriction base="xs:string">
          <xs:whiteSpace
             value="replace"/>
          </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="collapse">
   <xs:simpleType>
     <xs:restriction base="xs:string">
        <xs:whiteSpace
           value="collapse"/>
     </xs:restriction>
   </xs:simpleType>
 </xs:element>
</xs:all>
</xs:complexType>

### ShEx
<whiteSpaces> {
  :preserve xs:string ;
  :replace  xs:string
   %js{
     _.o.lex = _.o.lex
        .replace("/\r|\n|\r\n|\s/g", " ");
    return true;
   }
   % ;
   :collapse xs:string
   %js{
     var replacedText = _.o.lex
        .replace("/\r|\n|\r\n|\s/g", " ");
     _.o.lex = replacedText.trim();
     return true;
   }
  %
}
```

Listing 22: WhiteSpace mapping

4.7.8. *Unique*

`xs:unique` is used in XML Schema to define that an element of some type is unique, i.e., there cannot be the same values among elements defined in the rule. This is useful for cases like IDs, where a unique ID is the way to identify an element. Currently, ShEx does not support `Unique` function but it is expected to be supported in future versions[4]. As a temporal solution, semantic actions could be used to implement this kind of constraint (see Listing 23).

```
### XML Schema
<xs:element name="Person"
            maxOccurs="unbounded">
  <xs:complexType>
    <xs:all>
       <xs:element name="name"
                   type="xs:string" />
         <xs:element name="surname"
                     type="xs:string" />
           <xs:element name="id"
                       type="xs:integer" />
    </xs:all>
  </xs:complexType>
  <xs:unique name="onePersonPerID">
     <xs:selector xpath="."/>
     <xs:field xpath="id"/>
  </xs:unique>
</xs:element>

### ShEx
%js{
    var ids = [];
    return true;
}
%
<Person> {
  :name xs:string ;
  :surname xs:string ;
  :id xs:integer
  %js{ if(ids.indexOf(_.o.lex) >= 0)
         return false;
       ids.push(_.o.lex);
       return true;
  }%
}
```

Listing 23: Unique mapping

5. XMLSchema2ShEx prototype

In addition to the proposed mappings from XML Schema to Shape Expressions, and in order to answer RQ2, a prototype has been developed. This prototype uses a subset of the presented mappings and converts a given XML Schema input to a ShEx output.

The prototype has been developed in Scala and is available online[5]. It is a work-in-progress implemen-

[4] https://www.w3.org/2001/sw/wiki/ShEx/Unique_UNIQUE

[5] https://github.com/herminiogg/XMLSchema2ShEx

Table 1

Supported and pending of implementation features in XMLSchema2ShEx prototype. * Not natively supported in ShEx 2.0.

Supported features	Complex type, Simple type, All, Attributes, Restriction, Element, Max exclusive, Min exclusive, Max inclusive, Min inclusive, Enumeration, Pattern, Cardinality
Pending implementation	Choice, List, Union, Extension, Fraction Digits, Length, Max Length, Min Length, Total digits, Whitespace*, Unique*

tation, so not all the mappings are supported yet (see Table 1 for a list of supported features).

The tool is built on top of Scala parser combinators [28]. Once the XML Schema input is analysed and verified, it is converted to ShEx based on different elements and types declared on it. These conversions are made recursively and printed to the ouput in ShEx Compact Format (ShExC).

The input XML Schema document example presented in Listing 24 is used to ensure that the prototype can work and do the transformation as expected. This example includes complex types, attributes, elements, simple types and patterns among others. Complex types are converted to shapes, elements and attributes to triple predicates and objects, restrictions (max/minExclusive and max/minInclusive) to numeric intervals, cardinality attributes to ShEx cardinality and so on. Although it is a small example, it has the structure of typical XML Schemas used nowadays and the prototype can convert it properly as it is stated in Listing 24.

```
### XML Schema
<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://tempuri.org/po.xsd"
 xmlns="http://tempuri.org/po.xsd"
 elementFormDefault="qualified">

 <xs:element name="purchaseOrder"
             type="PurchaseOrderType"/>

  <xs:element name="comment"
              type="xs:string"/>

  <xs:complexType name="PurchaseOrderType">
    <xs:all>
       <xs:element name="shipTo"
                   type="USAddress"/>
       <xs:element name="billTo"
                   type="USAddress"/>
       <xs:element ref="comment"
                   minOccurs="0"/>
       <xs:element name="items"
                   type="Items"/>
    </xs:all>
    <xs:attribute name="orderDate"
                  type="xs:date"/>
 </xs:complexType>

 <xs:complexType name="USAddress">
    <xs:all>
       <xs:element name="name"
                   type="xs:string"/>
       <xs:element name="street"
                   type="xs:string"/>
       <xs:element name="city"
                   type="xs:string"/>
       <xs:element name="state"
                   type="xs:string"/>
       <xs:element name="zip"
                   type="xs:integer"/>
    </xs:all>
    <xs:attribute name="country"
                  type="xs:NMTOKEN"
                  fixed="US"/>
</xs:complexType>

<xs:complexType name="Items">
  <xs:all>
    <xs:element name="item"
                minOccurs="0"
                maxOccurs="unbounded">
     <xs:complexType>
      <xs:all>
       <xs:element
                name="productName"
                type="xs:string"/>
       <xs:element
                name="quantity">
         <xs:simpleType>
            <xs:restriction
               base="xs:positiveInteger">
              <xs:maxExclusive
                   value="100"/>
            </xs:restriction>
         </xs:simpleType>
         </xs:element>
           <xs:element name="USPrice"
              type="xs:decimal"/>
         <xs:element ref="comment"
              minOccurs="0"/>
         <xs:element name="shipDate"
              type="xs:date" minOccurs="0"/>
     </xs:all>
   <xs:attribute name="partNum" type="SKU"
        use="required"/>
  </xs:complexType>
  </xs:element>
 </xs:all>
```

```
 </xs:complexType>

 <xs:simpleType name="SKU">
   <xs:restriction base="xs:string">
     <xs:pattern value="\d{3}-[A-Z]{2}"/>
   </xs:restriction>
 </xs:simpleType>
</xs:schema>

### ShEx
PREFIX : <http://www.example.com/>
PREFIX
    xs: <http://www.w3.org/2001/XMLSchema#>

<Items> {
  :item         @<item> *  ;
}
<item> {
  :productName  xs:string  ;
  :quantity     xs:positiveInteger
                MAXEXCLUSIVE 100 ;
  :USPrice      xs:decimal  ;
  :comment      xs:string ?  ;
  :shipDate     xs:date ?  ;
  :partNum      /\\d{3}-[A-Z]{2}/ ;
}
<PurchaseOrderType> {
  :shipTo     @<USAddress>  ;
  :billTo     @<USAddress> ;
  :comment    xs:string ?  ;
  :items      @<Items>  ;
  :orderDate  xs:date ;
}
<USAddress> {
  :name       xs:string ;
  :street     xs:string  ;
  :city       xs:string  ;
  :state      xs:string ;
  :zip        xs:integer ;
  :country    ["US"]  ;
}
```

Listing 24: XML Schema to ShEx example

5.1. Validation example

```
### XML
<?xml version="1.0"?>
<purchaseOrder
  xmlns="http://tempuri.org/po.xsd"
  orderDate="1999-10-20">
  <shipTo country="US">
     <name>Alice Smith</name>
     <street>123 Maple Street</street>
     <city>Mill Valley</city>
     <state>CA</state>
     <zip>90952</zip>
  </shipTo>
  <billTo country="US">
     <name>Robert Smith</name>
     <street>8 Oak Avenue</street>
     <city>Old Town</city>
     <state>PA</state>
     <zip>95819</zip>
  </billTo>
  <comment>
     Hurry, my lawn is going wild!
  </comment>
  <items>
  <item partNum="872-AA">
     <productName>
       Lawnmower
     </productName>
     <quantity>1</quantity>
     <USPrice>148.95</USPrice>
      <comment>
                Confirm this is electric
      </comment>
      </item>
  <item partNum="926-AA">
    <productName>
        Baby Monitor
    </productName>
    <quantity>1</quantity>
    <USPrice>39.98</USPrice>
    <shipDate>1999-05-21</shipDate>
  </item>
 </items>
</purchaseOrder>

### RDF
:order1
  :shipTo [
    :name "Alice Smith" ;
    :street "123 Maple Street" ;
    :city "Mall Valley" ;
    :state "CA" ;
    :zip 90952 ;
    :country "US"
  ] ;
 :billTo [
    :name "Robert Smith" ;
    :street "8 Oak Avenue" ;
    :city "Old Town" ;
    :state "PA" ;
    :zip 95819 ;
    :country "US"
  ] ;
 :comment "Hurry, my lawn is going wild!";
 :items [
    :item [
        :productName "Lawnmower" ;
        :quantity "1"^^xs:positiveInteger ;
        :USPrice 148.95 ;
        :comment "Confirm this is electric";
        :partNum "872-AA"
    ] ;
    :item [
```

```
        :productName "Baby Monitor" ;
        :quantity "1"^^xs:positiveInteger ;
        :USPrice 39.98 ;
        :shipDate "1999-05-21"^^xs:date ;
        :partNum "926-AA"
    ] ;
];
:orderDate "1999-10-20"^^xs:date .
```

Listing 25: XML to RDF example

Once conversion from XML Schema to ShEx is done, it must be verified that the same validation that was performed on XML data using XML Schema, but now on RDF data using ShEx, is working equivalently. The translation of a valid XML to RDF is executed which is presented in Listing 25. The conversion presented in the snippet uses blank nodes to represent the nested types. This is done to avoid creating a fictitious node every time a triple is pointing to another triple (in other words, every time it has a nested type). The conversion was performed following similar equivalences to those proposed in the mappings. That is, complex types to triple subjects or predicates, simple types to triple objects, cardinality translated directly and so on.

For RDF validation using ShEx there are various implementations in different programming languages that are being developed[6]. One of these implementations is made in Scala by one of the authors of this paper and it is available online[7].

Using the examples given above the validation can be performed with the mentioned tool which allows the RDF and the ShEx inputs in various formats and then the option to validate the RDF against ShEx or SHACL schema. As seen in Figure 2, validation is performed trying to match the shapes with the existing graphs, whenever the tool matches a pattern it shows the evidence in green and a short explanation of why this graph has matched.

6. Non-Deterministic schemata

There is an issue that arises in XML Schema documents that should be solved when proposing a transformation from XML Schema. This is the topic of Non-Deterministic schemata where the parser is unable to determine the sequence to validate due to the Unique Particle Attribution. This issue appears, for example, in a choice between two sequences that begin with the same element. This event can be formulated with the regular expression: `(ab | ac)` and in XML Schema as shown in Listing 26.

```
### XML Schema
<xs:complexType name="nondeterministic">
    <xs:choice>
        <xs:sequence>
            <xs:element name="a"/>
            <xs:element name="b"/>
        </xs:sequence>
        <xs:sequence>
            <xs:element name="a"/>
            <xs:element name="c"/>
        </xs:sequence>
    </xs:choice>
</xs:complexType>

### ShEx
<nondeterministic> {
  a @<ab> OR @<ac> ;
}

<ab> {
  rdf:first @<a> ;
  rdf:rest @<ab1> ;
}

<ac> {
  rdf:first @<a> ;
  rdf:rest @<ac1> ;
}

<ab1> {
  rdf:first @<b> ;
  rdf:rest [rdf:nil]  ;
}

<ac1> {
  rdf:first @<c> ;
  rdf:rest [rdf:nil]  ;
}

<a> {
  :namea xs:string ;
}

<b> {
  :nameb xs:string ;
}

<c> {
  :namec xs:string ;
}
```

Listing 26: Non-Deterministic schema and its ShEx counterpart

[6] A list of ShEx implementations is available at: `https://shex.io`

[7] `http://shaclex.herokuapp.com`

Fig. 2. Validation result using Shaclex validator. The RDF data is entered in the left text area whereas the ShEx schema is entered on the right text area. In the bottom, a ShapeMap is declared to make the validator know where and how to begin the validation, in this case we commanded to validate :order1 node with <PurchaseOrderType> shape. In the top of the page, the result is shown detailing how each node was validated and what are the evidences or failures for the validation. A link to the validation example can be found in Supplementary Material.

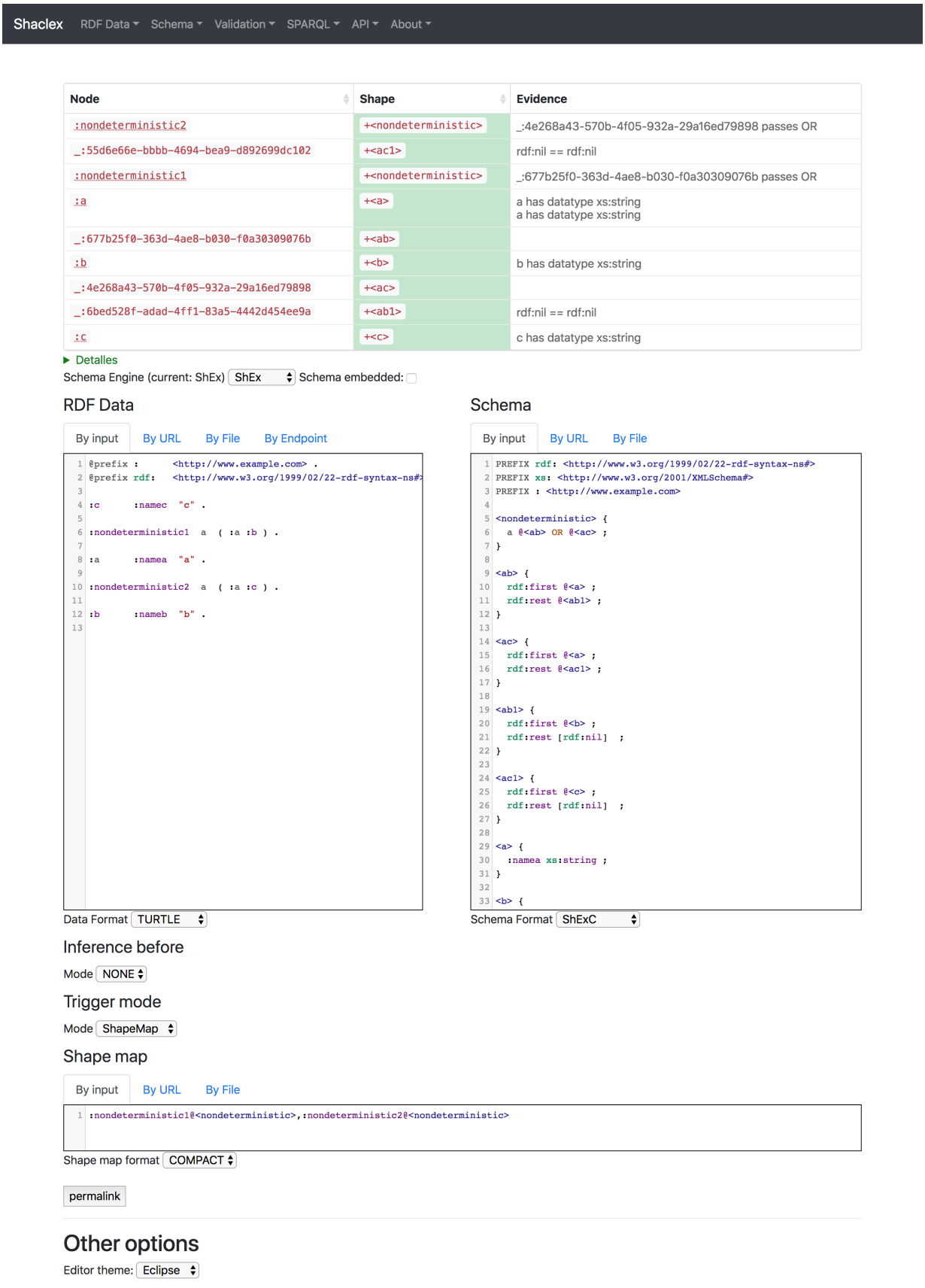

Fig. 3. Validation result using Shaclex validator of a ShEx schema converted from a non-deterministic XML Schema document. In the Shape map input area text we have indicated to Shaclex validator to check if :nondeterministic1 and :nondeterministic2 hold the form of shape <non-deterministic>. In the top of the page the satisfactory result is shown in green.

These sequences are translated as shown in Section 4.3.1 and the final result can be seen in Listing 26. The question is that if this non-determinism is also transferred to the converted schemata. In order to check the actual behaviour we have run this example on Shaclex validator which shows that the validation is performed correctly (see Figure 3).

This behaviour is motivated by two things: firstly, the structure of RDF lists is different from XML Schema sequences which makes the validation to be performed in a different form; consequently, the validation in ShEx is performed recursively trying to match shape by shape. Therefore, if an element match with a shape this will scale up into the recursion tree without creating ambiguity problems.

7. Conclusions and Future work

In this work, a possible set of mappings between XML Schema and ShEx has been presented. With this set of mappings, automation of XML Schema conversions to ShEx is a new possibility for schema translation which is demonstrated by the prototype that has been developed and presented in this paper. Using an existing validator helped to demonstrate that an XML and its corresponding XML Schema are still valid when they are converted to RDF and ShEx.

One future line of work that should be tackled is the loss of semantics: with this kind of transformations some of the elements could not be converted back to their original XML Schema constructs. Nevertheless, it is a difficult problem due to the difference between ShEx and XML data models and it would involve some sort of modifications and additions to the ShEx semantics (like the previously mentioned inheritance).

To cover more business cases and make this solution more compatible with existing systems, there is the need to create mappings for Schematron and Relax NG as a future work. Relax NG is grammar-based but Schematron is rule based, which will make conversion from Relax NG to ShEx more straightforward than from Schematron to ShEx, as ShEx is also grammar-based. Another line of future work is to adapt the presented mappings to SHACL: most of the mappings follow a similar structure. Moreover, the rule-based Schematron conversion seems more feasible using the advanced SHACL-SPARQL features which allow to expand the core SHACL language by using SPARQL queries to validate complex constraints.

With the present work, validation of existing transformations between XML and RDF is now possible and convenient. This kind of validations makes the transformed data more reliable and trustworthy and it also facilitates migrations from non-semantic data formats to semantic data formats.

Conversions from other formats (such as JSON Schema, DDL, CSV Schema, etc.) will also be investigated to permit an improvement of data interoperability by reducing the technological gap.

Acknowledgments

This work has been partially funded by the Vicerectorate for Research of the University of Oviedo under the call of "*Programa de Apoyo y Promoción de la Investigación 2017*".

References

[1] Steve Battle. Round-tripping between XML and RDF. In Jeremy J. Carroll, editor, *International Semantic Web Conference (ISWC)*, Posters, Hiroshima, November 2004.

[2] Steve Battle. Gloze: XML to RDF and back again. In *Proceedings of the First Jena User Conference*, HP Labs, Bristol, May 2006.

[3] Diego Berrueta, Jose Emilio Labra Gayo, and Ivan Herman. XSLT + SPARQL: Scripting the semantic web with SPARQL embedded into XSLT stylesheets. In Christian Bizer, Sören Auer, Gunnar Aastrand, and Grimnes Tom Heath, editors, *4th Workshop on Scripting for the Semantic Web*, volume 368, Tenerife, June 2008. CEUR-WS.

[4] Geert Jan Bex, Frank Neven, and Jan den Bussche. DTDs versus XML schema: a practical study. In Luis Gravano and Sihem Amer-Yahia, editors, *Proceedings of the 7th international workshop on the web and databases: colocated with ACM SIGMOD/PODS 2004*, ICPS, pages 79–84, Paris, June 2004. ACM. doi: 10.1145/1017074.1017095.

[5] Paul V. Biron, Ashok Malhotra, World Wide Web Consortium, et al. XML Schema part 2: Datatypes. `https://www.w3.org/TR/xmlschema-2/`, 2004.

[6] Stefan Bischof, Stefan Decker, Thomas Krennwallner, Nuno Lopes, and Axel Polleres. Mapping between RDF and XML with XSPARQL. *Journal on Data Semantics*, 1(3):147–185, 2012. doi: 10.1007/s13740-012-0008-7.

[7] Hannes Bohring and Sören Auer. Mapping XML to OWL Ontologies. In Klaus P. Jantke, Klaus-Peter Fähnrich, and Wolfgang S. Wittig, editors, *Marktplatz Internet: von E-Learning bis E-Payment, 13. Leipziger Informatik-Tage*, volume 72 of *LNI*, pages 147–156, Leipzig, September 2005. GI.

[8] Iovka Boneva, Jose Emilio Labra Gayo, and Eric Prud'hommeaux. Semantics and Validation of Shapes Schemas for RDF. In Claudia d'Amato, Miriam Fernandez, Valentina Tamma, Freddy Lecue, Philippe Cudré-Mauroux, Juan Sequeda, Christoph Lange, and Jeff Heflin, editors, *Inter-*

national Semantic Web Conference, volume 10587 of *Lecture Notes in Computer Science*, pages 104–120, Vienna, October 2017. Springer Verlag. doi: 10.1007/978-3-319-68288-4_7.

[9] Frank Breitling. A standard transformation from XML to RDF via XSLT. *Astronomische Nachrichten*, 330(7):755–760, 2009. doi: 10.1002/asna.200811233.

[10] Dan Brickley and R.V. Guha. RDF Schema 1.1. https://www.w3.org/TR/rdf-schema/, 2014.

[11] James Clark and Makoto Murata. Relax NG specification. http://relaxng.org/spec-20011203.html, 2001.

[12] Davy Van Deursen, Chris Poppe, Gäetan Martens, Erik Mannens, and Rik Van de Walle. XML to RDF Conversion: A Generic Approach. In Paolo Nesi, Kia Ng, and Jaime Delgado, editors, *2008 International Conference on Automated solutions for Cross Media Content and Multi-channel Distribution.*, pages 138–144, Florence, November 2008. IEEE. doi: 10.1109/AXMEDIS.2008.17.

[13] Nick Drummond, Alan L Rector, Robert Stevens, Georgina Moulton, Matthew Horridge, Hai Wang, and Julian Seidenberg. Putting OWL in order: Patterns for sequences in OWL. In Bernardo Cuenca Grau, Pascal Hitzler, Conor Shankey, and Evan Wallace, editors, *Proceedings of the OWLED'06 Workshop on OWL: Experiences and Directions*, Athens, Georgia, November 2006. CEUR-WS.

[14] TEI Consortium, eds. TEI P5: Guidelines for Electronic Text Encoding and Interchange. http://www.tei-c.org/Guidelines/P5/, 2017.

[15] Matthias Ferdinand, Christian Zirpins, and David Trastour. Lifting XML Schema to OWL. In Nora Koch, Piero Fraternali, and Martin Wirsing, editors, *Web Engineering: 4th International Conference, ICWE 2004*, volume 3140 of *Lecture Notes in Computer Science*, pages 354–358, Munich, July 2004. Springer Berlin Heidelberg. doi: 10.1007/978-3-540-27834-4_44.

[16] P. N. Fox, R. Mead, M. Talbot, and J. D. Corbett. Data management and validation. In R. A. Kempton and P. N. Fox, editors, *Statistical Methods for Plant Variety Evaluation*, volume 3 of *Plant Breeding*, chapter 3, pages 19–39. Chapman & Hall, London, 1997. doi: 10.1007/978-94-009-1503-9_3.

[17] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview (Second Edition). https://www.w3.org/TR/owl2-overview/, 2012.

[18] Rick Jelliffe. The Schematron: An XML structure validation language using patterns in trees. http://xml.ascc.net/resource/schematron/schematron.html, 2001.

[19] Holger Knublauch. SPIN - Modeling Vocabulary. http://www.w3.org/Submission/spin-modeling/, 2011.

[20] Holger Knublauch and Dimitris Kontokostas. Shapes constraint language (SHACL). https://www.w3.org/TR/shacl/, June 2017.

[21] Nassim Kobeissy, Marc Girod Genet, and Djamal Zeghlache. Mapping XML to OWL for seamless information retrieval in context-aware environments. In Fusun Ozguner, Buyurman Baykal, Ali Akoglu, and Ozgur Ercetin, editors, *IEEE International Conference on Pervasive Services*, pages 349–354, Istanbul, July 2007. IEEE. doi: 10.1109/PERSER.2007.4283938.

[22] Jose Emilio Labra Gayo, Eric Prud'hommeaux, Iovka Boneva, and Dimitri Kontokostas. *Validating RDF Data*, volume 7 of *Synthesis Lectures on the Semantic Web: Theory and Technology*. Morgan and Claypool Publishers, 2018. doi: 10.2200/S00786ED1V01Y201707WBE016.

[23] Damien Lacoste, Kiran Prakash Sawant, and Suman Roy. An efficient XML to OWL converter. In Arun Bahulkar, K. Kesavasamy, T. V. Prabhakar, and Gautam Shroff, editors, *Proceedings of the 4th India software engineering conference*, pages 145–154, Thiruvananthapuram, 2011. ACM. doi: 10.1145/1953355.1953376.

[24] Dongwon Lee, Murali Mani, and Wesley W Chu. Schema Conversion Methods between XML and Relational Models. In Michel Klien and Borys Omelayenko, editors, *Knowledge Transformation for the Semantic Web*, volume 95 of *Frontiers in Artificial Intelligence and Applications*, chapter 1, pages 1–17. IOS Press, 2003.

[25] Sergei Melnik and Stefan Decker. Representing Order in RDF. http://infolab.stanford.edu/~stefan/daml/order.html, January 2001.

[26] Albert Meroño-Peñuela. Semantic web for the humanities. In Philipp Cimiano, Oscar Corcho, Valentina Presutti, Laura Hollink, and Sebastian Rudolph, editors, *The Semantic Web: Semantics and Big Data: 10th International Conference, ESWC 2013*, volume 7882 of *Lecture Notes in Computer Science*, pages 645–649, Montpellier, May 2013. Springer Berlin Heidelberg. doi: 10.1007/978-3-642-38288-8_44.

[27] Igor Miletic, Marko Vujasinovic, Nenad Ivezic, and Zoran Marjanovic. Enabling Semantic Mediation for Business Applications: XML-RDF, RDF-XML and XSD-RDFS transformations. In Ricardo J Gonçalves, Jörg P Müller, Kai Mertins, and Martin Zelm, editors, *Enterprise Interoperability II: New Challenges and Approaches*, pages 483–494, Madeira, 2007. Springer London. doi: 10.1007/978-1-84628-858-6_53.

[28] Adriaan Moors, Frank Piessens, and Martin Odersky. Parser combinators in Scala. *Department of Computer Science, KU Leuven*, 2008. https://limo.libis.be/primo-explore/fulldisplay?docid=LIRIAS1652814&context=L&vid=Lirias&search_scope=Lirias&tab=default_tab&lang=en_US.

[29] Falco Nogatz and Thom Frühwirth. *From XML Schema to JSON Schema - Comparison and Translation with Constraint Handling Rules*. Bachelor Thesis. Ulm: University of Ulm, Ulm, Germany, 2013.

[30] Martin J. O'Connor and Amar Das. Acquiring OWL ontologies from XML documents. In Mark A. Musen and Oscar Corcho, editors, *Proceedings of the sixth international conference on Knowledge capture*, pages 17–24, Banff, June 2011. ACM. doi: 10.1145/1999676.1999681.

[31] Giuseppe Della Penna, Antinisca Di Marco, Benedetto Intrigila, Igor Melatti, and Alfonso Pierantonio. Interoperability mapping from XML Schemas to ER diagrams. *Data & Knowledge Engineering*, 59(1):166–188, 2006. doi: 10.1016/j.datak.2005.08.002.

[32] Eric Prud'hommeaux, Iovka Boneva, Jose Emilio Labra Gayo, and Gregg Kellogg. Shape expressions language 2.0. http://shex.io/shex-semantics/index.html, 2017.

[33] Eric Prud'hommeaux, Jose Emilio Labra Gayo, and Harold Solbrig. Shape expressions: an RDF validation and transformation language. In Harald Sack, Agata Filipowska, Jens Lehmann, and Sebastian Hellmann, editors, *Proceedings of the 10th International Conference on Semantic Systems, SEMANTiCS 2014*, pages 32–40, Leipzig, September 2014. ACM. doi: 10.1145/2660517.2660523.

[34] Toni Rodrigues, Pedro Rosa, and Jorge Cardoso. Mapping XML Schema to Existing OWL ontologies. In Pedro Isaías, Miguel Baptista Nunes, and Inmaculada J. Martínez, editors, *International Conference WWW/Internet*, volume II, pages 72–77, Murcia, October 2006. IADIS.

[35] John Simpson and Susan Brown. From XML to RDF in the Orlando Project. In Kozaburo Hachimura, Toru Ishida, Naoko Tosa, Donghui Lin, and Akira Maeda, editors, *2013 International Conference on Culture and Computing*, pages 194–195, Kyoto, September 2013. IEEE. doi: 10.1109/CultureComputing.2013.61.

[36] C. Michael Sperberg-McQueen and Eric Miller. 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] Timo Sztyler, Jakob Huber, Jan Noessner, Jaimie Murdock, Colin Allen, and Mathias Niepert. LODE: Linking digital humanities content to the web of data. In George Buchanan, Martin Klein, Andreas Rauber, and Sally Jo Cunningham, editors, *IEEE/ACM Joint Conference on Digital Libraries*, pages 423–424, London, September 2014. IEEE and ACM. doi: 10.1109/JCDL.2014.6970206.

[38] Jiao Tao, Evren Sirin, Jie Bao, and Deborah L. McGuinness. Integrity constraints in OWL. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pages 1443–1448, Atlanta, July 2010. AAAI.

[39] Pham Thi Thu Thuy, Young-Koo Lee, Sungyoung Lee, and Byeong-Soo Jeong. Transforming valid XML documents into RDF via RDF schema. In Ajith Abraham and Sang Yong Han, editors, *Third International Conference on Next Generation Web Services Practices (NWeSP 2007).*, pages 35–40, Seoul, October 2007. IEEE. doi: 10.1109/NWESP.2007.23.

[40] Pham Thi Thu Thuy, Young-Koo Lee, Sungyoung Lee, and Byeong-Soo Jeong. Exploiting XML schema for interpreting XML documents as RDF. In Wil van der Aalst, Calton Pu, Elisa Bertino, Ephraim Feig, and Patrick C. K. Hung, editors, *2008 IEEE International Conference on Services Computing (SCC'08)*, volume 2, pages 555–558, Honolulu, 2008. IEEE. doi: 10.1109/SCC.2008.93.