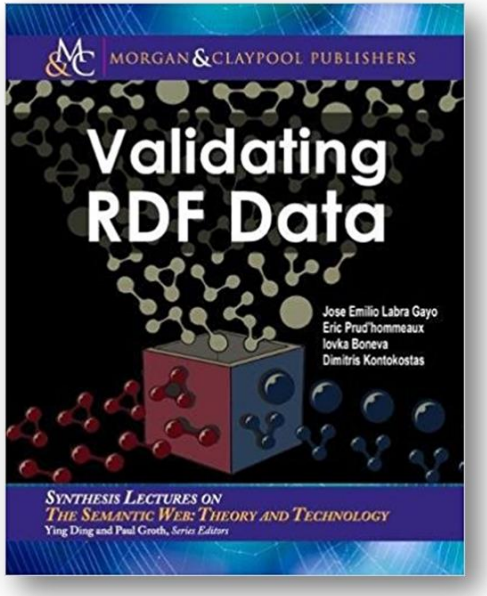
ShEx primer

Jose Emilio Labra Gayo
WESO Research group
University of Oviedo, Spain
https://labra.weso.es/

More information

Web page: http://shex.io
Semantics: http://shex.io/shex-semantics/
Primer: http://shex.io/shex-primer



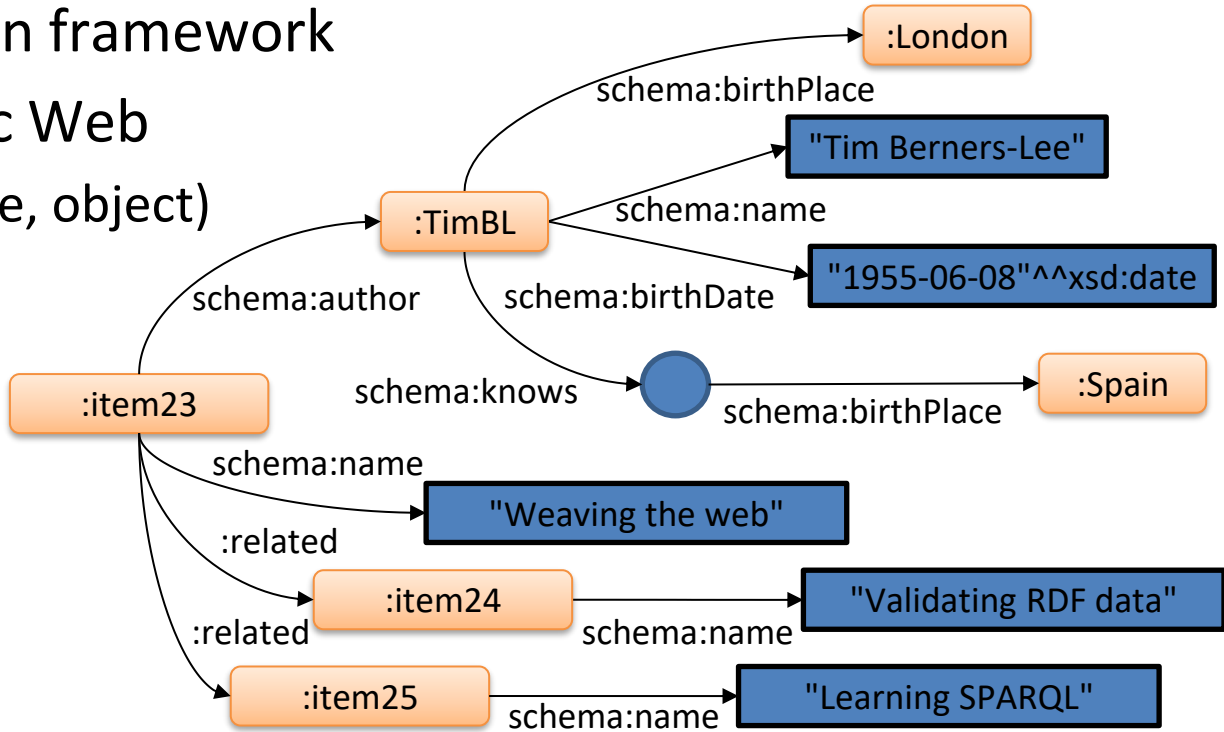
Jose E. Labra Gayo, Eric Prud'hommeaux, Iovka Boneva, Dimitris Kontokostas, *Validating RDF Data*, Synthesis Lectures on the Semantic Web, Vol. 7, No. 1, 1-328, DOI: 10.2200/S00786ED1V01Y201707WBE016, Morgan & Claypool (2018)
Online version: http://book.validatingrdf.com/

RDF graphs

RDF = resource description framework

Lingua franca of Semantic Web

- Triples: (subject, predicate, object)
- Predicates = URIs
  - Interoperability



Try it: https://rdfshape.weso.es/link/17089394117

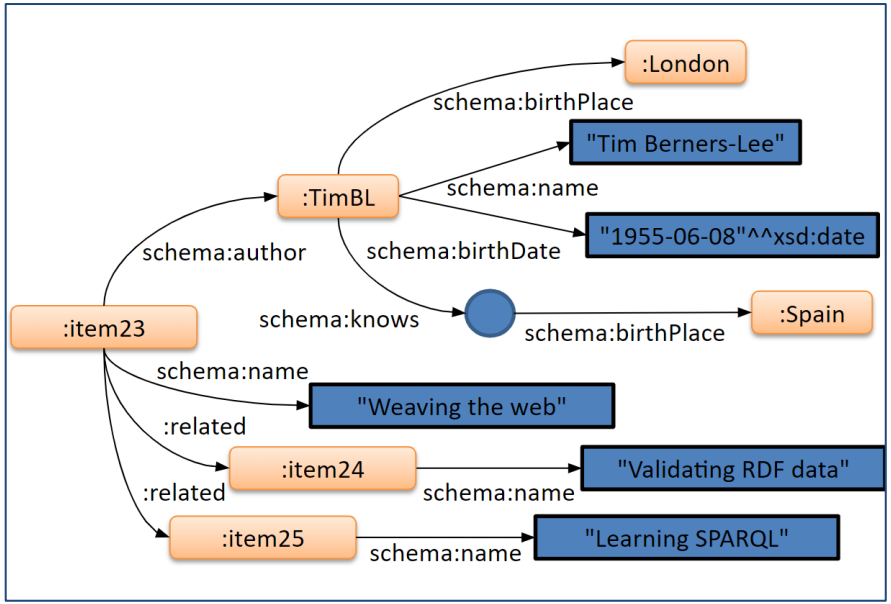
RDF syntaxes

One data model, several syntaxes: Turtle, N-Triples, JSON-LD, …

Turtle

```
prefix :       <http://example.org/>
prefix xsd:    <http://www.w3.org/2001/XMLSchema#>
prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix schema: <http://schema.org/>

:item23 schema:name      "Weaving the web"      ;
        schema:author     :timbl                ;
        :related          :item24, :item25      .
:timbl  schema:name       "Tim Berners-Lee"     ;
        schema:birthDate  "1955-06-08"^^xsd:date ;
        schema:birthPlace :london               ;
        schema:knows      _:1                   .
_:1     schema:birthPlace :Spain                .
:item24 schema:name      "Validating RDF data"  .
:item25 schema:name      "Learning SPARQL"      .
```



Try it: https://rdfshape.weso.es/link/17089394117

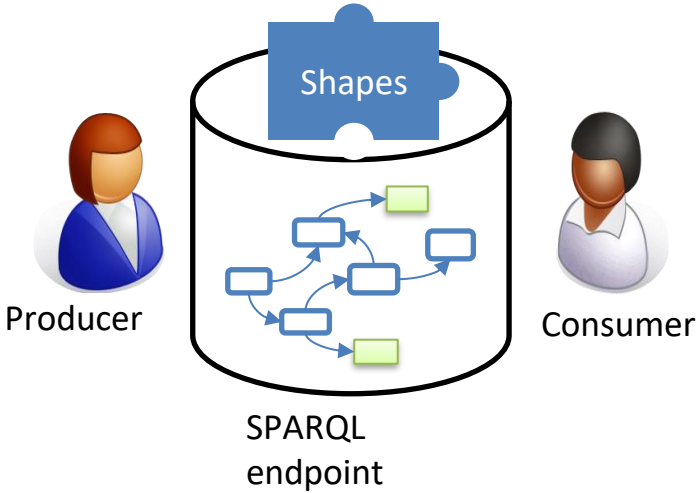
Why describe & validate RDF?

For producers

- Developers can understand the contents they are going to produce
- They can ensure they produce the expected structure
- Advertise and document the structure
- Generate interfaces

For consumers

- Understand the contents
- Verify the structure before processing it
- Query generation & optimization



Schemas for RDF?

RDF doesn’t impose a schema, but…

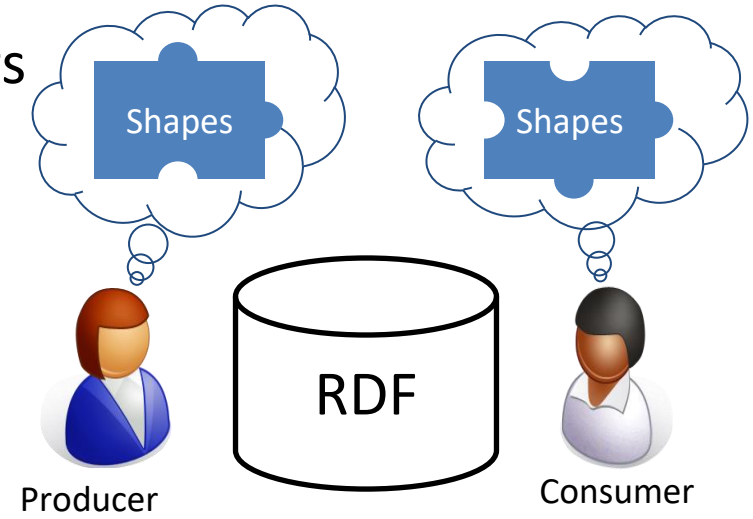
In practice, there are **implicit schemas**

Assumed by producers and consumers

Shapes make schemas explicit

Handle malformed/incomplete data

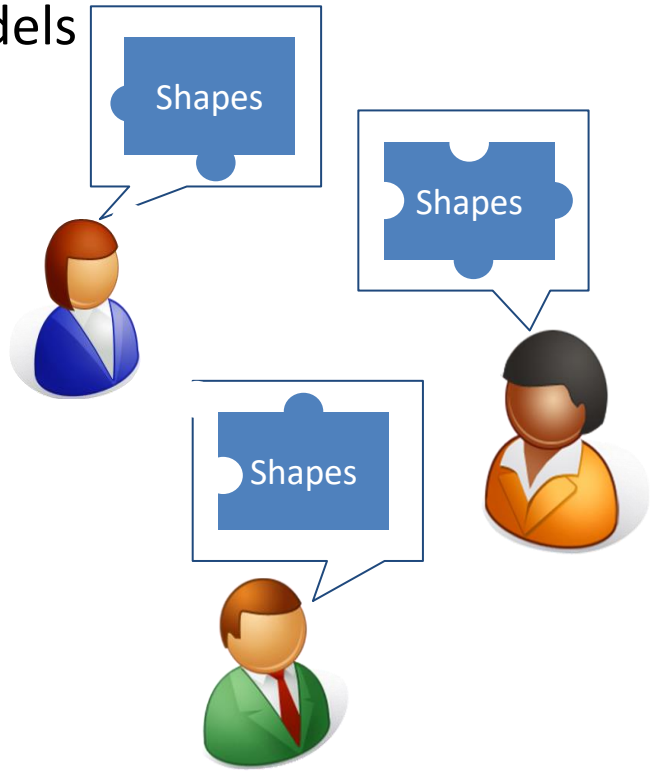
Avoid defensive programming



Focus discussions on what matters

Help domain experts define their own data models
- Understandable by domain experts
- …and machine processable

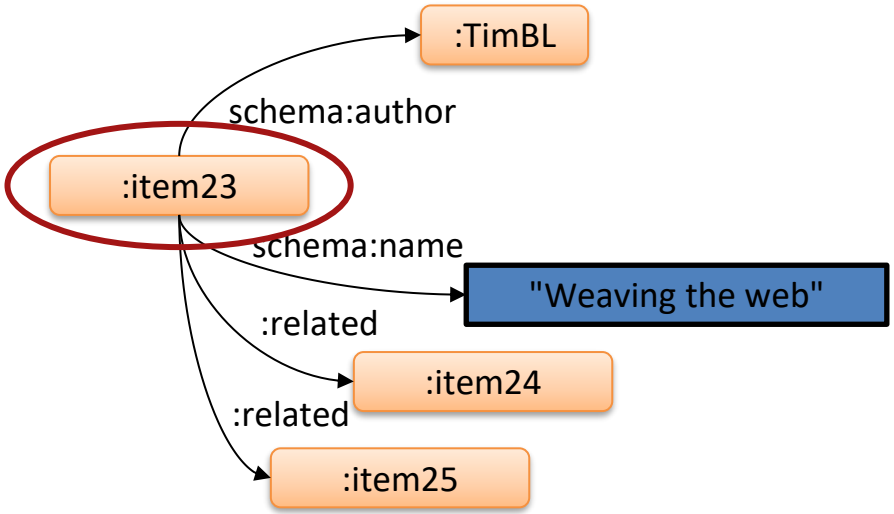
Initial motivation: clinical data models (FHIR)
- Distributed data model
  - Different location, authorities,...
- Extensible data models



What is a shape?

A shape describes

- The form of a node
- Incoming/outgoing arcs from a node
- Values associated with those arcs



RDF Node

```
:item23 schema:author :timbl              ;
        schema:name   "Weaving the web"   ;
        :related      :item24  , :item25 .
```

ShEx

```
:Book IRI and {
 schema:author @:Author    + ;
 schema:name   xsd:string    ;
 :related      @:Book      *
}
```

https://rdfshape.weso.es/link/17089392604

Evolution of ShEx

2013 RDF Validation Workshop

Conclusions of the workshop:

…need of a higher level, concise language for RDF Validation

ShEx initially proposed (v 1.0)

2014 W3C Data Shapes WG chartered

2017 ShEx 2.0 released as W3C Community group draft

2017 SHACL accepted as W3C recommendation

2019 ShEx 2.1 added support for imports

2019 ShEx adopted by Wikidata

2022 Added support for *extends*

Shape Expressions - ShEx

Goal: Concise and human-readable language

3 syntaxes:

- ShExC: Compact syntax, similar to Turtle or SPARQL
- ShExJ: JSON(-LD), for the spec
- ShExR: RDF, based on JSON-LD
  - **Note**: Round tripping is possible, convert from one to the other

Semantics inspired by regular expressions

ShEx libraries and demos

Implementations & libraries:

- shex.js: Javascript
- Jena-ShEx: Java
- SHaclEX: Scala (Jena/RDF4j)
- PyShEx: Python
- shex-java: Java
- Ruby-ShEx: Ruby
- RDF-Elixir: Elixir
- rudof: Rust 

Online demos & playgrounds

- ShEx-simple
- RDFShape
- Wikishape
- ShEx-Java
- ShExValidata

More info: http://shex.io

Simple example

Prefix declarations as in Turtle/SPARQL

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

:Book IRI AND {
 schema:name   xsd:string   ;
 :related      @:Book     *
}
```

Nodes conforming to `:Book` must

- Be `IRI`s and
- Have property `schema:name` with a value of type `xsd:string` (exactly one)
- Have property `:related` with values conforming to `:Book` (zero or more)

RDF Validation using ShEx

Schema

```
:Book IRI AND {
 :name    xsd:string   ;
 :related @:Book      *
}
```

RDF Data

```
:a   :name    "Title A" ;
     :related :b  .
:b   :related :a ;
     :name    "Title B".
:c   :name  "Title C1", "Title C2" .
:d   :name  234 .
:e   :namme  "Title E" .
:f   :name    "Title F" ;
     :related :a, _:1 .
_:1  :name  "Unknown title" .
```

Shape map

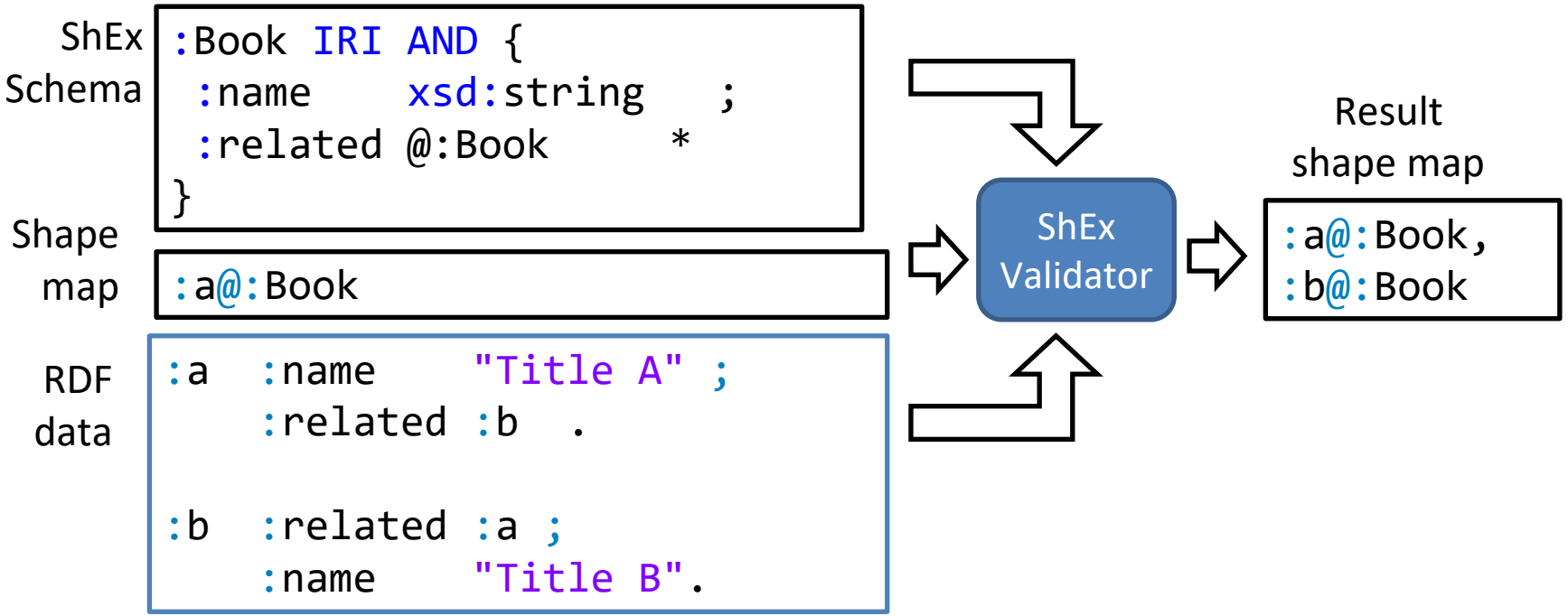
```
:a@:Book   ✓
:b@:Book,  ✓
:c@:Book,  ✗
:d@:Book,  ✗
:e@:Book,  ✗
:f@:Book   ✗
```

Try it (RDFShape)
Try it (Simple ShExDemo)

ShEx

Validation process

Input: RDF data, ShEx schema, Shape map
Output: Result shape map



Node constraints

Constraints over a node (without considering its neighbourhood)

```
:Book {
 :name           xsd:string
 :datePublished  xsd:date
 :numberOfPages  MinInclusive 1
 :author         @:Person
 :genre          [:Fiction :NonFiction ]
 :isbn           /isbn:[0-9X]{10}/
 :publisher      IRI
 :audio          .
 :maintainer     @:Person OR
                 @:Organization
}
:Person {}
:Organization {}
```

```
:item23
 :name           "Weaving the Web"                ;
 :datePublished  "2012-03-05"^^xsd:date           ;
 :numberOfPages  272                              ;
 :author         :timbl                           ;
 :genre          :NonFiction                      ;
 :isbn           "isbn:006251587X"                ;
 :publisher      <http://www.harpercollins.com/>  ;
 :audio          <http://audio.com/item23>        ;
 :maintainer     :alice                           .
```

Try it: (RDFShape)

Cardinalities

Inspired by regular expressions: +, ?, *, {m,n}

By default {1,1}

```
:Book {
 :name          xsd:string        ;
 :numberOfPages xsd:integer  ?    ;
 :author        @:Person     +    ;
 :publisher     IRI          ?    ;
 :maintainer    @:Person     {1,3} ;
 :related       @:Book       *
}
:Person {}
:Organization {}
```

```
:item23
  :name          "Weaving the Web"       ;
  :numberOfPages 272                     ;
  :author        :timbl, :markFischetti  ;
  :maintainer    :alice,:bob             .
```

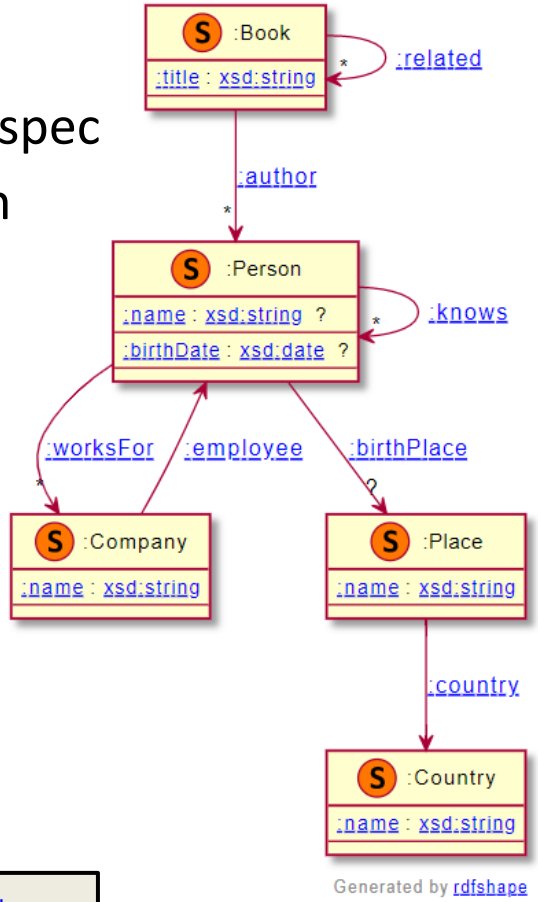
Try it: RDFShape

Recursive schemas

Support for recursive (cyclic) data models is part of the spec

Well formed semantics based on stratified negation

```
:Book    {  :title       xsd:string   ;
            :author      @:Person   * ;
            :related     @:Book     * }
:Person  {  :name        xsd:string ? ;
            :birthDate   xsd:date   ? ;
            :birthPlace  @:Place    ? ;
            :knows       @:Person   * ;
            :worksFor    @:Company  * }
:Place   {  :name        xsd:string   ;
            :country     @:Country    }
:Country {  :name        xsd:string   }
:Company {  :name        xsd:string   ;
            :employee    @:Person     }
```



Try it: RDFShape

Open/Closed content models

RDF semantics mostly presume open content models

Shape expressions are open by default

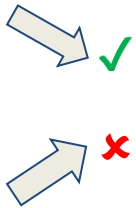
Enable extensibility

But…some use cases require closed content models

Added CLOSED keyword

```
:Book {
 :name     xsd:string   ;
}
```

```
:Book CLOSED {
 :name     xsd:string   ;
}
```



```
:a :name "Weaving the web" ;
   :isbn "006251587X"      .
```

Try it: RDFShape

Open/Closed properties

Property values are closed by default (closed properties)

```
:Book {
 :code /isbn:[0-9X]{10}/ ;
}
```

✓
```
:item23  :code "isbn:006251587X" .
```

✘
```
:item23  :code 23                .
```

Properties can be repeated

```
:Book {
  :code  /isbn:[0-9X]{10}/ ;
  :code  /isbn:[0-9]{13}/
}
```

```
:item23 :code "isbn:006251587X"    ,
        :code "isbn:9780062515872" .
```

EXTRA declares properties as open

```
:Book EXTRA :code {
 :code /isbn:[0-9X]{10}/ ;
}
```

```
:item23 :code "isbn:006251587X" ,
        :code 23                .
```

Try it: RDFShape

Triple expressions

“Unordered” regular expressions: Regular bag expressions

```
:Person {
 ( :name       xsd:string |
   :firstName xsd:string  + ;
   :lastName  xsd:string
 ) ;
 :birthDate   xsd:date    ?
}
```

```
(name |
  firstName + ;
  lastName
) ;
birthDate ?
```

```
(n | f + ; l ) ; b?
```

```
:alice :name      "Alice Cooper"          ;
       :birthDate "2010-02-23"^^xsd:date .

:bob   :firstName "Robert"                ;
       :lastName  "Smith"                 .

:carol :firstName "Carol"                 ;
       :lastName  "King"                  ;
       :firstName "Carole"                .
```

```
:alice → n
:bob   → f l
:carol → f l f
```

```
:dave  → n f
```

```
:dave :name      "Dave Navarro"           ;
      :firstName "Dave"                   .
```

Try it: RDFShape

Logical operators

Shape Expressions can be combined with AND, OR, NOT

```
:Book {
 :name   xsd:string ;
 :author @:Person OR @:Organization ;
}

:AudioBook @:Book AND {
 :name           MaxLength 20 ;
 :readBy         @:Person     ;
} AND NOT {
 :numberOfPages . +
}

:Person {}
:Organization {}
```

```
:item24 :name          "Weaving the Web" ;
        :author        :timbl            ;
        :readBy        :timbl            .
```

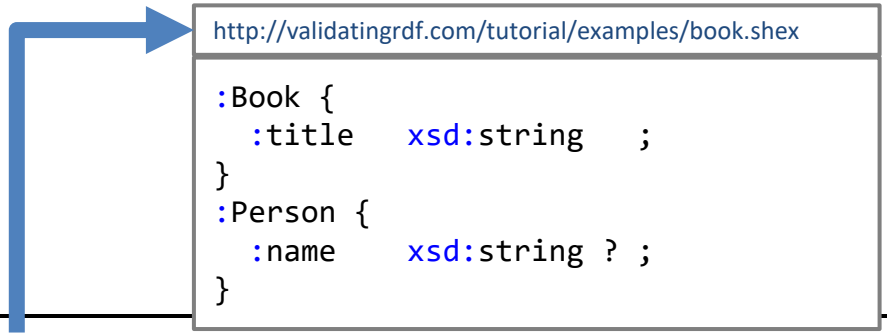
```
:item23 :name          "Weaving the Web" ;
        :author        :timbl            ;
        :numberOfPages 272               ;
        :readBy        :timbl            .
```

ShEx allows NOT combined with recursion (semantics based on stratified negation)

Try it: RDFShape

Importing schemas

`import` statement can be used to import schemas



http://validatingrdf.com/tutorial/examples/book.shex

```
:Book {
  :title    xsd:string   ;
}
:Person {
  :name     xsd:string ? ;
}
```

```
import <https://www.validatingrdf.com/examples/book.shex>

:AudioBook @:Book AND {
 :title          MaxLength 20 ;
 :readBy         @:Person     ;
}
```

```
:item24 :name          "Weaving the Web" ;
        :author        :timbl            ;
        :readBy        :timbl            .
```

Try it: RDFShape

Machine processable annotations

Annotations based on RDF

Lots of applications, e.g. generate forms from shapes

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

start = @:User

:User {
 schema:name      xsd:string                    // ui:label “Name"       ;
 schema:birthDate xsd:dateTime                  // ui:label "Birth date" ;
 schema:gender    [ schema:Male schema:Female ] // ui:label "Gender"
}
```

Try it:
Eric’s demo
https://rdfshape.weso.es/link/17095003321

More complex Shape Maps

Shape Maps define which nodes validate with which shapes

Examples:

`{FOCUS a :Person}@:User`	https://rdfshape.weso.es/link/17095033013
`{FOCUS schema:name _}@:Book`	https://rdfshape.weso.es/link/17095014386

`SPARQL """`
`PREFIX schema: <http://schema.org/>`
`SELECT ?book WHERE {`
`?book schema:name ?name;`
`schema:author ?author`
`}"""@:Book`
	https://rdfshape.weso.es/link/17095031848

Inheritance model for ShEx

`extends` allows to reuse existing shapes adding new content

Handles closed properties and shapes

```
:Book {
 :name    xsd:string ;
 :author  @:Person   ;
 :code    /isbn:[0-9]{13}/   ;
 :code    /isbn:[0-9X]{10}/
}

:LibraryBook extends @:Book {
 :code    /internal:[0-9]*/ ;
}
```

```
:item23 :name        "Weaving the Web"    ;
        :author      :timbl               ;
        :code        "isbn:006251587X"    ;
        :code        "isbn:9780062515872" ;
        :code        "internal:234"       .
```

Other features

- Multiple inheritance
- Abstract shapes

Try it: RDFShape

ShEx vs SPARQL

SPARQL pros:

- Expressive
- Ubiquitous

SPARQL cons:

- Expressive
- Idiomatic - many ways to encode the same constraint
- Non recursive

```
<User> {
  schema:name    xsd:string ;
  schema:gender [ schema:Female schema:Male]
}
```

```
ASK {{ SELECT ?Person {
          ?Person schema:name ?o .
    } GROUP BY ?Person HAVING (COUNT(*)=1)
  }
  { SELECT ?Person {
          ?Person schema:name ?o .
     FILTER ( isLiteral(?o) &&
              datatype(?o) = xsd:string )
     } GROUP BY ?Person HAVING (COUNT(*)=1)
  }
  { SELECT ?Person (COUNT(*) AS ?c1) {
          ?Person schema:gender ?o .
    } GROUP BY ?Person HAVING (COUNT(*)=1)}
    { SELECT ?Person (COUNT(*) AS ?c2) {
          ?S schema:gender ?o .
      FILTER ((?o = schema:Female ||
               ?o = schema:Male))
    } GROUP BY ?Person HAVING (COUNT(*)=1)}
    FILTER (?c1 = ?c2)
}
```

3 syntaxes: ShExC, ShExJ, ShExR

ShExC

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

:Book {
 schema:name   xsd:string   ;
 :related      @:Book     *
}
```

ShExR (RDF, Turtle)

```
:Book a sx:ShapeDecl ;
  sx:shapeExpr [ a sx:Shape ;
   sx:expression [ a sx:EachOf ;
    sx:expressions (
      [ a sx:TripleConstraint ;
        sx:predicate schema:name ;
        sx:valueExpr  [ a sx:NodeConstraint ;
                        sx:datatype xsd:string
                      ] ]
      [ a sx:TripleConstraint ;
        sx:predicate :related ;
        sx:valueExpr  [ a sx:NodeConstraint ;
        sx:valueExpr :Book ] ] ) ] ] .
```

It's posible to roudtrip from each one

ShExJ (JSON LD)

```
{ "type" : "Schema",
  "@context" : "http://www.w3.org/ns/shex.jsonld",
  "shapes" : [ {
      "type" : "Shape",
      "id" : "http://example.org/Book",
      "expression" : {
        "type" : "EachOf",
        "expressions" : [ {
            "type" : "TripleConstraint",
            "predicate" : "http://schema.org/name",
            "valueExpr" : {
              "type" : "NodeConstraint",
              "datatype" : "http://www.w3.org/2001/XMLSchema#string"
            }
          },
          { "predicate" : "http://example.org/related",
            "valueExpr" : "http://example.org/Book",
            "min" : 0,
            "max" : -1,
            "type" : "TripleConstraint"
          }
        ] } } ] }
```

ShEx from a more theoretical point of view

- Grammar divided in two main blocks
- Shape expressions (nodes)
- Triple expressions (neighbourhood)
  - Regular Bag Expressions
- Recursion and negation with stratified negation
  - Could it be changed to stable model semantics?

$\mathcal{S}$	::=	$l \mapsto se^*$	
se	::=	IRI | BNode | ...	Node constraints
	|	cond	A boolean condition on nodes
	|	se_1 AND se_2	Conjunction
	|	se_1 OR se_2	Disjunction
	|	NOT se	Negation
	|	$@l$	Shape label reference for $l \in \Lambda$
	|	{ te }	Triple expression te
te	::=	$te_1; te_2$	Each of te_1 and te_2
	|	$te_1 \mid te_2$	Some of te_1 or te_2
	|	$\_ \xrightarrow{p} @l$	Triple with predicate p that conforms to shape expression identified by l
	|	$te*$	Zero or more te

Conclusions

- More ShEx features
  - Stems, named expressions, nested shapes, semantic actions, …
- And ShEx tools
  - Inference ShEx from data (sheXer), editors, KG subsets, …
- ShEx and SHACL compared (see later)
  - Different underlying philosophy
    - ShEx more inspired on grammars than on constraints
  - Separation of concerns
    - Structure definition (ShEx) ≠ Ontology (OWL)
    - Structure definition (ShEx) ≠ Node/shape selection (ShapeMaps)