

|--|

Matrix Lana Wachowski



name	director
Matrix	Lana Wachowski
Matrix	Lilly Wachowski











```
{
    "name": "Crazy, Stupid Love"
    "director: "Glenn Ficarra"
}
```

```
{
    "name": "Crazy, Stupid Love"
    "director: "Glenn Ficarra"
},
{
    "name": "Matrix"
    "director": ["Lana W.", "Lilly W." ]
}
```

As databases grow, we need a way to understand what they have and how to query them

As databases grow, we need a way to understand what they have and how to query them

wiki_500.json

a mental spin news at the at sumpring a to a struct a ("id":"Q369","type":"iten","aliases":{"ang":[{"language":"ang","value":"Wicicwide"},{"language":"ang","value":"\u01f7ikic\u01bfide"},{"language":"ang "type":"item","aliases":{"fr":[{"language":"fr","value":"chk chk chk"}],"en-gb":[{"language":"en-gb","value":"chk chk chk"},{"language":"e "property","aliases":{"vi":[{"language":"vi","value":"x\u01b0\u1edbng ng\u00f4n vi\u00ean"}],"it":[{"language":"it", "type":"property","aliases":{"it":[{"language":"it","value":"vettore"}],"en":[{"language":"en","value":"carrier rocket"},{"language":"en type":"item","alimen":{"be-tmrmsk":{{"imngumge":"be-tmrmsk","value":"\u0406\u0437\u06124\u0413\u0413\u0413\u0413 "property","aliases":{"it":[{"language":"it","value":"CDS"}],"en":[{"language":"en","value":"municipality code (Netherlands)"}],"ro ':"item","aliases":{"en":[{"language":"en","value":"rue Sainte-Catherine"}],"nb":[{"language":"nb","value":"rue Sainte-Catherine"}]}, ':"item","aliases":{"nb":[{"language":"nb","value":"m\udde5leteknikk"},{"language":"nb","value":"m\udde5leteknikk vitenskap "ites","aliases";("nds-nl";[{"language";"nds-nl","value";"Babrein")],"it";[("language";"it","value";"Regno del Babrain")],"ta";[("la ':"item","aliases":("rup":[{"language":"rup","value":"\$\ubbe?rbit"}],"nan":[{"language":"nan","value":"Serbia"},("language": "property", "aliases": ("es": [{"language":"es", "value":"lengua"}, {"language":"es", "value":"idiona"}, ("language":"es", "value":"idiona 'property","aliases":{"fr":[{"language":"fr","value":"NLA"}],"ii":[{"language":"ii","value":"National Library of Australia"}],"es":[:"iten","aliases":{"pl":[{"language":"pl","value":"egrobiologia"},{"language":"pl","value":"kosmobiologia"},{"language":"pl","val "item","descriptions":{"en":{"language":"en","value":"two plagues attached to the Pioneer 18 and Pioneer 11 spacecraft, in case 'item","aliases":{"be-tarask":[{"language":"be-tarask","value":"\u0418\u0448\u0449\u0413\u0413\u0447\u0445\u0443d\u0438"],"sgs":[{"l "property", "aliases":{"be-tarask":[{"language":"be-tarask", "value":"\u0444\u043e\u043d\u043d\u043d\u0432\u0432\u0432\u0432 "type":"iten","aliases":{{"language":"sgs","value":"Peru"}},"nan":[{"language":"nan","value":"Peru"}},"nb":[{"language":"nb";"value" "type":"property","aliases":{"en":{{"language":"en","value":"timezone"},{"language":"en","value":"tz"},{"language":"en","value":"time zone" "id":"0424","type":"iten","aliases":{"en":[{"language":"en","value":"Kingdom of Cambodia"},{"language":"en","value":"kh"}},"de":[{"language":"de","valu "id":"P428", "type":"property", "aliases":{"nl":[{"language":"nl", "value":"afkorting botanist"}, {"language":"nl", "value":"botanist-afkorting"}, {"language "id":"0479","type":"iten". "descriptions":{"pl":{"language":"pl","value":"village in Poland"},"fr":{"language":"fr","value":"village de Pologne"},"de":{ "id":"D431","type":"item","aliases":{"en":[{"language":"en","value":"zo\u0016logy"}],"de":[{"language":"de","value":"Tierkunde"}],"hu":[{"lang ":"item","descriptions":{"pl":{"language":"pl","value":"gmina wiejska w Polsce, w wojew/ubbf/drtwie lubelskim, w powiecie pu "property","aliases":{"en":[{"language":"en","value":"artist MDTD"},{"language":"en","value":"artist id"},{"language":"en "descriptions":{"pl":{"language":"pl","value":"miasto w Polsce"},"fr":{"language":"fr","value":"ville de Pologne" usicBrainr ID \u8434\u8435\u844f \u843a\ "property", "aliases":{"ru":[{"language":"ru", "value "iten","aliases":{"fr":[{"language":"fr","value":"Je Berlinois")]."fi":[{"language":"fi"."va "property", "aliases":{"ru":{{"language":"ru", "value":"\u843f\u8440\u843e\ "iten","aliases":{"en":[{"language":"en","value":"Encyclop\u00e9die, ou ':"property","aliases":{"is":[{"language":"is","value":"\ul@aa\ul@ea\ul@ba\ul@ca\ul@eb\ul@cb\ul@cd\ul@cl "iten","alianes":{"de":[{"language":"de","value":"friede"}]},"descriptions":{"en":{"language":"en","value":"state "property", "aliases": {"nl": [{"language":"nl", "value": "DO-nu mmer")],"scn":[{"language":"scn", "type":"property","aliases":{"fa":[{"language":"fa","value":"\u0645\u067a\u0672\u0677\u0672\u0644"}],"cs":[{"language":"cs", "type":"item","aliases":{"ru":[{"language":"ru","value":"\u8428\u843e\u8436\u8436\u8436\u8436\u8436\u843f\u8446 "type":"item","aliases":{"cs":[{"language":"cs","value":"OBpedie"}]},"descriptions":{"en":{"language":"en","value":"online database project "property","aliases":{"it":[{"language":"it","value":"promulgato__da"},{"language":"it","value":"emanato_da"}],"ru":[{"language":"ru ":"iten","aliases":{"hu":[{"language":"hu","value":"Eduardo Nicanor Frei Montalva"}]},"descriptions":{"es":{"language":"es","value":"/ "type":"property","aliases":{"vi":[{"language":"vi","value":"bi\uleclu \u0111\uled] gu\u0129 \u0111\uleala"}]},"descriptions":{"en":{"lan "type":"iten","aliases":{"ru"::{{"language":"ru","value":"Beakman's World"}}},"descriptions":{"es":{"language":"es","value":"Programma de tel "type":"iten","aliases":{"en":[{"language":"en","value":"Torino"},{"language":"en","value":"Turin, Italy"}],"be":[{"language":"be","value" "type":"item","aliases":{"de":[{"language":"de","value":"Peranal"},{"language":"de","value":"Poperze"},{"language":"de", "property","aliases":{"fr":[{"language":"fr","value":"CDDN"}],"it":[{"language":"it","value":"CDDN"}],"ja":[{"language":"it","value":"it

As databases grow, we need a way to understand what they have and how to query them



So, it appears we do need schemas

- for JSON data
- for graphs
- even for tabular or text data!

This talk:

SHACL (Shapes Constraint Language) ShEx (Shape Expressions)

JSON Schema

Schemas for graphs and other forms of semi-structured data

Why these? Why now?

SHACL: W3C recommendation (Mid 2017) ShEx Group still working (last update 4'2019) JSON Schema: Working Group in IETF

Also working group for schemas in property graphs



Schemas for graphs and other forms of semi-structured data

Juan L. Reutter PUC Chile

Felipe Pezoa, Domagoj Vrgoč, Martín Ugarte, Fernando Suarez, Pierre Bouhris, Ognjen Savkovic, Julien Corman, Fernando Florenzano, Iovka Boneva, Sławek Staworko

RDF Graphs



In this talk: Set V of nodes Edges over V x V, labelled with a string

JSON

```
{
    "name": "Crazy, Stupid Love"
    "director: "Glenn Ficarra"
},
{
    "name": "Matrix"
    "director": [Lana W., Lilly W.]
}
```

Data is always about resources, and linking resources with other resources.

Schema imposes conditions on some of them.

Shape-based Schemas - general form



 $\mathcal{L}_{\text{const}}$

language to express shapes

language to express constraints

Shape-based Schemas - general form



 \mathcal{L}_{const}

language to express shapes

language to express constraints

 $T(\mathbf{X})$ Answers of this query must be of a shape

 $\boldsymbol{\varphi}(\boldsymbol{x})$

Nodes of the shape must satisfy this query

Shape-based Schemas - general form



 \mathcal{L}_{const}

language to express shapes

language to express constraints

 $T(\mathbf{X})$ Answers of this query must be of a shape

 $\boldsymbol{\varphi}(\mathbf{x})$

Nodes of the shape must satisfy this query

 $T(\mathbf{x}) \rightarrow \boldsymbol{\varphi}(\mathbf{x})$

```
JSON Schema
                                    "name": "Aconcagua",
                                    "elevation": 6960,
                                    "country": "Argentina",
                                    "first ascender": {
                                     "name": "Matthias",
                                     "surname": "Zurbriggen"
                                  }
   "type": "object",
   "properties": {
     "name": {"type": "string"},
     "elevation": {"type": "integer"},
     "country": {"type": "string"},
      "first_ascender": {
```

JSON Schema

 $\mathcal{L}_{\text{type}}$

root shape must conform root JSON Schema



There must be a name (string), there must be a country (string),...

If there is a first ascender, then

JSON Schema "name": "Aconcagua", "elevation": 6960, "country": "Argentina", "first ascender": { "name": "Matthias", "surname": "Zurbriggen" } "type": "object", "properties": { "name": {"type": "string"}, "elevation": {"type": "integer"}, "country": {"type": "string"}, "first_ascender":

JSON Schema

"type": "object",

"name": {"type": "string"},

"surname": {"type": "string"}

"properties": {

"definitions": {

"person": {

}

"name": "Aconcagua",
"elevation": 6960,
"country": "Argentina",
"first_ascender": {
 "name": "Matthias",
 "surname": "Zurbriggen"
}

```
"type": "object",
"properties": {
    "name": {"type": "string"},
    "elevation": {"type": "integer"}
    "country": {"type": "string"},
    "first_ascender": {
```

"\$ref": "#/definitions/person"

JSON Schema

 $\mathcal{L}_{\text{type}}$

root shape must conform root JSON Schema



There must be a name (string), there must be a country (string),...

If there is a first ascender, then it satisfies shape person

Real JSON schemas use a lot of shapes



Shape-based Schemas - general form \mathcal{L}_{type} \mathcal{L}_{const}

language to express shapes

language to express constraints

Set of shapes (person, address, mountain, etc...)

Answers of this query must be of shape S

 $\varphi_{S}(x)$

 $T_{\rm S}(\mathbf{X})$

Nodes of shape S must satisfy this query. Query can use shape names!

SHACL

```
:movieShape
    a sh:NodeShape;
    sh:targetClass :movie;
    sh:property [
        sh:path :starring;
        sh:node :personShape
    ];
    sh:path :director;
        sh:minCount 1;
        sh:node :personShape
    ];
```

```
:personShape
    a sh:NodeShape;
    sh:property [
    sh:path :spouse;
    sh:node :personShape
    ] .
```

SHACL

```
:movieShape
    a sh:NodeShape;
    sh:targetClass :movie;
    sh:property [
        sh:path :starring;
        sh:node :personShape
    ];
    sh:path :director;
        sh:minCount 1;
        sh:node :personShape
    ];
```

```
:personShape
   a sh:NodeShape;
   sh:property [
   sh:path :spouse;
   sh:node :personShape
  ].
```

All nodes of type :movie must conform to :movieShape





these nodes must conform to :movieShape

SHACL

```
:personShape
   a sh:NodeShape;
   sh:property [
   sh:path :spouse;
   sh:node :personShape
  ].
```

Neighbours of nodes assigned :movieShape, connected by :starring, must satisfy :personShape


these nodes must conform to :personShape

```
:movieShape
    a sh:NodeShape;
    sh:targetClass :movie;
    sh:property [
        sh:path :starring;
        sh:node :personShape
    ];
    sh:path :director;
        sh:minCount 1;
        sh:node :personShape
    ];
```

```
:personShape
   a sh:NodeShape;
   sh:property [
   sh:path :spouse;
   sh:node :personShape
  ].
```

Neighbours of nodes assigned :movieShape, connected by :director, must satisfy :personShape, we need at least 1

this node must conform to :personShape



violation: every movie needs at least one director

```
:movieShape
    a sh:NodeShape;
    sh:targetClass :movie;
    sh:property [
        sh:path :starring;
        sh:node :personShape
    ];
    sh:path :director;
        sh:minCount 1;
        sh:node :personShape
    ];
```

```
:personShape
    a sh:NodeShape;
    sh:property [
    sh:path :spouse;
    sh:node :personShape
    ] .
```

Neighbours of nodes assigned :personShape, connected by :spouse, must satisfy :personShape



these nodes must conform to :personShape

Shape-based Schemas - general form \mathcal{L}_{type} \mathcal{L}_{const}

language to express shapes

language to express constraints

Set of shapes (person, address, mountain, etc...)

Answers of this query must be of shape S

 $\varphi_{S}(x)$

 $T_{\rm S}(\mathbf{X})$

Nodes of shape S must satisfy this query. Query can use shape names!

Individual nodes Answers of query {?x rdf:type U}



 $\mathcal{L}_{\mathsf{type}}$

- Is a string, is a number, ...
- # of neighbours connected by a path
- what my neighbours satisfy (these can be other shapes)



Individual nodes Answers of query {?x rdf:type U}



 $\mathcal{L}_{\mathsf{type}}$

- Is a string, is a number, ...
- # of neighbours connected by a path
- what my neighbours satisfy (these can be other shapes)
 - ~ FO with 2 variables + counting + paths (modal logic with counting)



 \mathcal{L}_{type}

Individual nodes Answers of query {?x rdf:type U}



- Is a string, is a number, ...
- # of neighbours connected by a path
- what my neighbours satisfy (these can be other shapes)
 - ~ FO with 2 variables + counting + paths (modal logic with counting)

semantics is not trivial!



ShEx

\mathcal{L}_{type} allows any pattern of the form {?x p U} or {U p ?x}

\mathcal{L}_{const} very similar to SHACL (will return to this this)

Why do we study Shape-based Schemas?

All these languages are specifications or established drafts

Need for formal specification.

Understand best way of defining things

How to solve tasks: validation, satisfiability, ...

JSON Schema was quite messy when we started (2015)

	V1	V2	V3	V4	V5
T1:	Ν	Y	Y	Ν	Y
T2:	Y	Ν	Y	Ν	Y
T3:	Ν	Y	N	Ν	Ν
T4:			N		_

N invalid	
_	
– unsupported	

Each test T1-T4: validating a document against a schema V1-V5: first 5 validators in google search (circa 10/2015)

SHACL official W3C recommendation

The validation with recursive shapes is not defined in SHACL and is left to SHACL processor implementations. For example, SHACL processors may support recursion scenarios or produce a failure when they detect recursion.

SHACL official W3C recommendation

The validation with recursive shapes is not defined in SHACL and is left to SHACL processor implementations. For example, SHACL processors may support recursion scenarios or produce a failure when they detect recursion.

ShEx report late 2018

This is an editor's draft of the Shape Expressions specification. ShEx 2.x differs significantly from the W3C ShEx Submission. The July 2017 publication included a definition of validation which implied infinite recursion. This version explicitly includes recursion checks. No tests changed as a result of this and no implementations or applications are known to have been affected.

Why do we study Shape-based Schemas?

All these languages are specifications or established drafts

Need for formal specification.

Understand best way of defining things

How to solve tasks: validation, satisfiability, ...

Why do we study Shape-based Schemas?

All these languages are specifications or established drafts

Need for formal specification.

Understand best way of defining things

Graphs

How to solve tasks: validation, satisfiability, ...

SHACL/ShEx

Best way of defining things

- syntax
- semantics

Tasks: validation, satisfiability, ...

$\mathcal{L}_{type} + \mathcal{L}_{const} + Semantics$

\mathcal{L}_{type} Way of selecting nodes that must be of shape S

- must select particular node
- Specs use very simple queries {?x p U} or {U p ?x}

 \mathcal{L}_{type} Way of selecting nodes that must be of shape S

- must select particular node
- Specs use very simple queries {?x p U} or {U p ?x}

Any unary query would do

$$\text{if } \mathcal{L}_{type} \subseteq \mathcal{L}_{const}$$

then most likely this does not affect the expressive power

Defining Shape-based Schemas: SHACL

- unary tests (is a string, is this node, etc)
- shape tests (node is assigned a shape)

Defining Shape-based Schemas: SHACL

 \mathcal{L}_{const} What nodes of shape S must satisfy

- unary tests (is a string, is this node, etc)
- shape tests (node is assigned a shape)
- counting neighbours:

 $\geq_n p. \varphi$ $\leq_n p. \varphi$

min/max # of p-neighbours satisfying $oldsymbol{arphi}$

Defining Shape-based Schemas: SHACL

 \mathcal{L}_{const} What nodes of shape S must satisfy

- unary tests (is a string, is this node, etc)
- shape tests (node is assigned a shape)
- counting neighbours:

 $\geq_n p. \varphi$ $\leq_n p. \varphi$

min/max # of p-neighbours satisfying $oldsymbol{arphi}$

- comparing paths:

 $EQ(p_1, p_2)$ set of p_1 -neighbours = set of p_2 -neighbours

Paths are defined using RPQs/property paths

```
:movieShape
a sh:NodeShape;
sh:targetClass :movie;
sh:property [
sh:path :starring;
sh:node :personShape
];
sh:property [
sh:path :director;
sh:minCount 1;
sh:minCount 1;
sh:node :personShape
];
```

\leq_0 :starring.(\neg :personShape)

:movieShape

```
a sh:NodeShape ;
sh:targetClass :movie ;
sh:property [
    sh:path :starring ;
    sh:node :personShape
] ;
sh:path :director ;
    sh:path :director ;
    sh:minCount 1 ;
    sh:node :personShape
] ;
```

\geq_1 :director.(:personShape)

:movieShape

```
a sh:NodeShape;
sh:targetClass :movie;
sh:property [
    sh:path :starring;
    sh:node :personShape
];
sh:property [
    sh:path :director;
    sh:minCount 1;
    sh:node :personShape
];
```

$T_{:movieShape} = \{?x \ rdf: type : movie\}$

```
:movieShape
    a sh:NodeShape;
    sh:targetClass :movie;
    sh:property [
        sh:path :starring;
        sh:node :personShape
    ];
    sh:path :director;
        sh:minCount 1;
        sh:node :personShape
    ];
```

 $T_{:movieShape} = \{ ?x \text{ rdf:type :movie} \}$ $\varphi_{:movieShape} = \leq_0 : starring.(\neg:personShape) \land \geq_1 : director.(:personShape)$

- unary tests (is a string, is this node, etc)
- shape tests (node is assigned a shape)

- unary tests (is a string, is this node, etc)
- shape tests (node is assigned a shape)



- unary tests (is a string, is this node, etc)
- shape tests (node is assigned a shape)



 \mathcal{L}_{const} What nodes of shape S must satisfy

- unary tests (is a string, is this node, etc)
- shape tests (node is assigned a shape)
- regular bag expressions over p @S interpreted over bag of neighbours

$exp = p @s | \varepsilon | exp|exp | exp; exp | exp[m, n]$

So which one is better?

Word is still open for consideration.

Both formalisms are incomparable:

(a @S; a @S)[0, *]



So which one is better?

Word is still open for consideration.

Both formalisms are incomparable:

EQ(ab, ac)



So which one is better?

Word is still open for consideration.

Both formalisms are incomparable.

Complexity issues (data complexity):

- checking if a SHACL constraint holds in a node is tractable
- checking if a ShEx constraint holds is not
So which one is better?

Word is still open for consideration.

Both formalisms are incomparable

Complexity issues (data complexity):

- checking if a SHACL constraint holds in a node is tractable
- checking if a ShEx constraint holds is not

(usually one restricts to ShEx where the * is not nested)

So which one is better?

Word is still open for consideration.

Both formalisms are incomparable

Complexity issues.

Expressive power / ease to write

SHACL/ShEx

Best way of defining things

- syntax
- semantics

Tasks: validation, satisfiability, ...

Semantics

these nodes must conform to :personShape



these nodes must conform to :MovieShape

Semantics: iteratively assign shapes when needed?

Semantics: iteratively assign shapes when needed?



"Spouses of persons are persons"

```
:personShape
a sh:NodeShape;
sh:property [
sh:path :spouse;
sh:node :personShape
] .
```

Semantics: guess a good assignment?

Semantics: guess a good assignment?





"I have a blue neighbour"



Semantics: guess a good assignment?





"I have a blue neighbour"



Semantics: guess a good assignment?





"I have a blue neighbour"



Semantics: guess a partial good assignment

Semantics: guess a partial good assignment





"I have a blue neighbour"





graph



shapes



 T_{s_1},\ldots,T_{s_n}

some nodes are

assigned shapes

nodes in a shape must satisfy these

Can I assign shapes and satisfy all constraints?



graph

 $s_1, ..., s_n$

shapes

 $\boldsymbol{\varphi}_{\mathtt{S}_1},\ldots,\boldsymbol{\varphi}_{\mathtt{S}_n}$

 T_{s_1},\ldots,T_{s_n}

some nodes are

assigned shapes

nodes in a shape must satisfy these

Can I assign shapes and satisfy all constraints?

- Assignment respects T_{s_1}, \ldots, T_{s_n}
- Assignment agrees with $\varphi_{s_1}, \ldots, \varphi_{s_n}$
- Every node is assigned a shape, its negation, or nothing



green

node 2 must be green



"I have a blue neighbour"











node **2** must be green "I have a blue neighbour" "My neighbours are not blue"



Deciding if a graph validates a schema is NP-complete

- Consider only complete assignments



- Consider only complete assignments



- Consider only complete assignments



complete assignment => partial assignment

- Consider only complete assignments

- Restrict schemas using stratified negation

- Consider only complete assignments

- Restrict schemas using stratified negation



"I have a blue neighbour"



- Consider only complete assignments

- Restrict schemas using stratified negation



"I have a blue neighbour"



- still NP-hard
- only need complete assignments

- Consider only complete assignments

- Restrict schemas using stratified negation

+

- Only care about assignments that can be built iteratively

- easy to compute

- misses assignments: may not validate reasonable graphs

Graph validating a schema: everything is pretty when non-recursive

- All notions of assignment are equivalent

- Problem in PTIME if checking constraints is in PTIME

- Can even be transformed into SPARQL queries

Graph validating a schema: everything is pretty when non-recursive

- All notions of assignment are equivalent

- Problem in PTIME if checking constraints is in PTIME

- Can even be transformed into SPARQL queries

:personShape
a sh:NodeShape;
sh:property [
sh:path :spouse;
sh:node :personShape
] .

SHACL/ShEx

Best way of defining things

Tasks: validation, satisfiability, ...

Validation



graph



Is this valid?



- Compute SPARQL query







For classes of schemas we know validation is in PTIME

This runs in PTIME

Approach introduces design considerations






language to express shapes

language to express constraints

fast queries!

fast queries!







language to express shapes

language to express constraints

fast queries!

fast queries! queries without many answers





language to express shapes

language to express constraints

fast queries!

fast queries! queries without many answers

true in SHACL ShEx (no nesting of *)

Understand what is fast / what is not



Understand what is fast / what is not

Explanations



Understand what is fast / what is not

Explanations

Property Graphs? Text? CSV?

Understand what is fast / what is not

More theory (satisfiability, data exchange, ...)

Explanations

Property Graphs? Text? CSV?

Understand what is fast / what is not

More theory (satisfiability, data exchange, ...)

Explanations

Schema design

Property Graphs? Text? CSV?

Understand what is fast / what is not

More theory (satisfiability, data exchange, ...)

Explanations

Query Optimisation

Schema design

Property Graphs? Text? CSV?

Learn schemas



Schemas for graphs and other forms of semi-structured data

Juan L. Reutter PUC Chile

S. Staworko, I. Boneva, J. E. Labra Gayo, S. Hym, E. G. Prud'hommeaux, and H. Solbrig. Complexity and Expressiveness of ShEx for RDF. In ICDT, 2015

I. Boneva, J. E. L. Gayo, and E. G. Prud'hommeaux. Semantics and Validation of Shapes Schemas for RDF. In ISWC, 2017.

J. Corman, J. L. Reutter, and O. Savkovic. A tractable notion of stratification for SHACL. In ISWC, 2018.

Felipe Pezoa, Juan L. Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of JSON schema. In WWW, 2016.