VOILA! 2022

Shacled Turtle

Schema-Based Auto-Completion

Julian Bruyat[1], Pierre-Antoine Champin[12], Lionel Médini[1], Frédérique Laforest[1]

[1] Université de Lyon - INSA Lyon / Université Lyon 1 / LIRIS Laboratory - TWEAK Team

[2] INRIA - WIMMICS Team / W3C

https://github.com/BruJu/shacled-turtle - NPM shacled-turtle

Who writes RDF documents today?

- RDF is mainly generated:
  - From other data (R2RML, JSON-LD ...)
  - From software (Protégé, ...)
  - From user inputs in forms

- But writing or editing small RDF documents is still useful:
  - Fine-tuning ontologies
  - SHACL shapes
  - Mappings
  - Template graphs in SPARQL queries
  - Teaching the basis of RDF

```
 9  ex:Alice rdf:type ex:Person .
10  ex:ACME rdf:type ex:Organization .
11  ex:Bob rdf:type ex:Person .
12
13  ex:ACME ex:f
14
```

➢ How can one help humans to write RDF documents?
Use the content of schemas to provide autocompletion

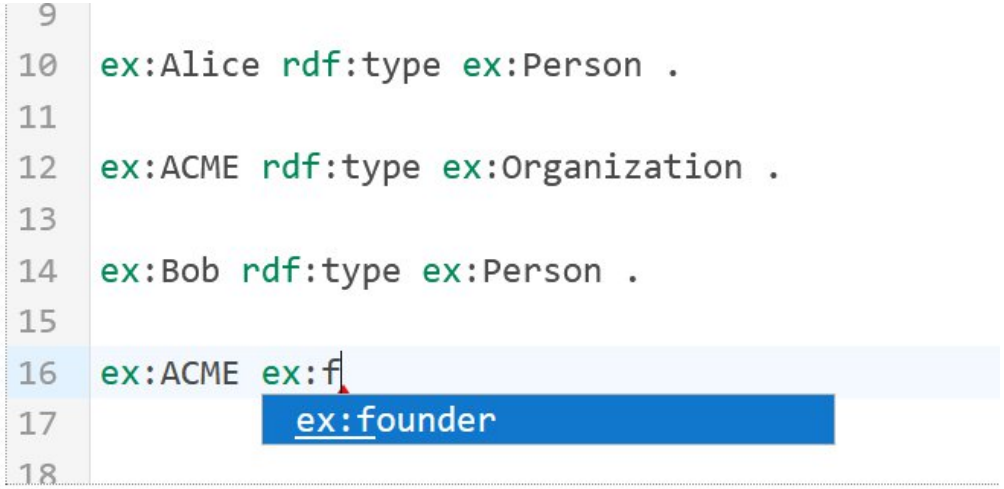
Proposal: Shacled Turtle

- Implemented as a plugin for Code Mirror 6, a web browser code editor
- Try it online: https://bruy.at/demo/shacled-turtle/demo.html

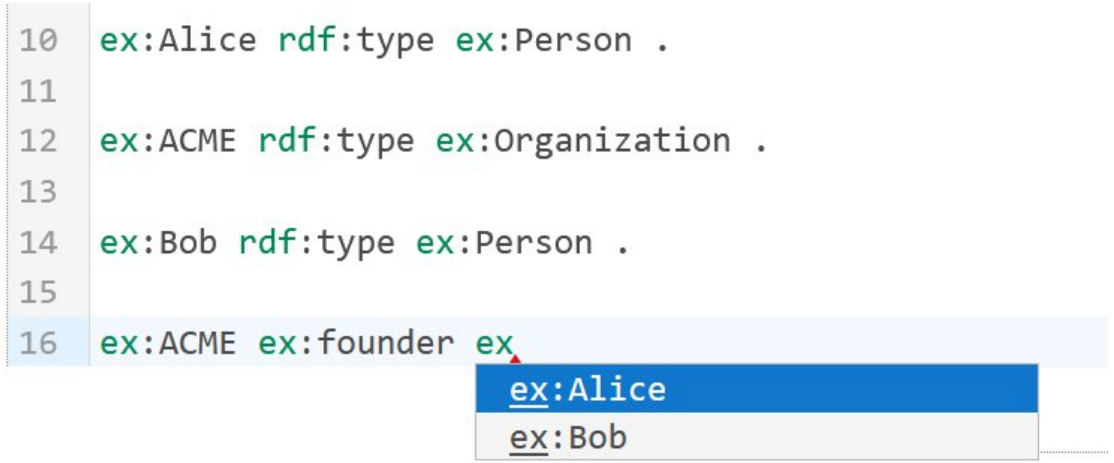
```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix ex: <http://www.example.org/> .

...
ex:birthDate rdfs:domain ex:Person .
ex:founder rdfs:domain ex:Organization .
ex:founder rdfs:range ex:Person .

...
```



ex:ACME is an organization: show relevant properties for organizations



The range of ex:founder is ex:Person: show all persons

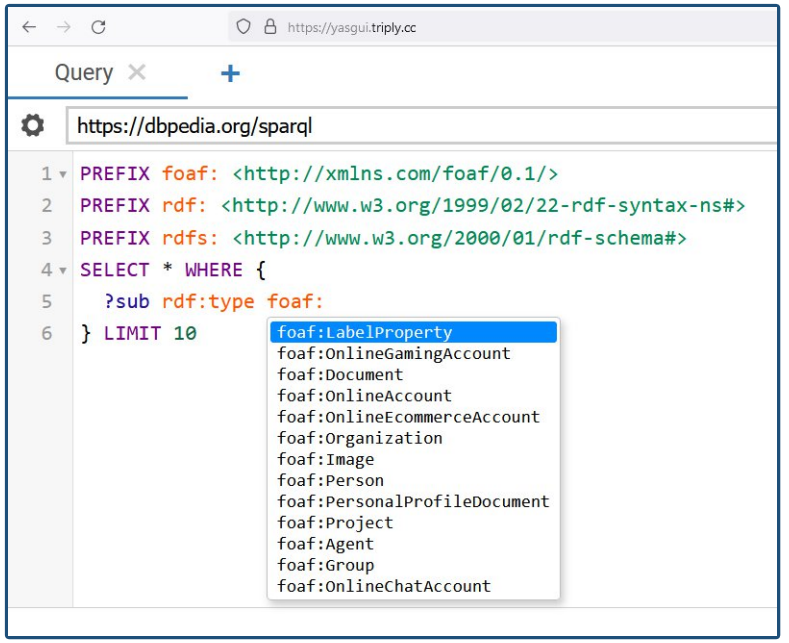
Structure of the talk

- **Writing RDF graphs**
- Shacled Turtle
- Experiments and results

Making writing RDF easier today with autocompletion

- Strategies only viable for SPARQL graph patterns
  - Subqueries to populate the autocompletion base [1]
  - Preprocessing on the stored data [2]
- Strategies applicable to RDF documents
  - YASGUI uses ontologies to populate the autocompletion list

Can we do better?



YASGUI shows all classes or properties that exist in the ontology

[1] Gombos, G., & Kiss, A. (2014, June). SPARQL query writing with recommendations based on datasets. In International Conference on Human Interface and the Management of Information (pp. 310-319). Springer, Cham.
[2] de la Parra, G., & Hogan, A. (2021). Fast Approximate Autocompletion for SPARQL Query Builders. In *VOILA@ ISWC* (pp. 41-55).

RDFS (RDF Schema)

- W3C vocabulary for data modelling and inference
- https://www.w3.org/TR/rdf-schema/

```
@prefix ex: <http://www.example.org/> .

ex:ACME ex:founder ex:Alice .
```

An RDF graph

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://www.example.org/> .

# Organization membership in RDFS
ex:founder rdfs:domain ex:Organization . # [1]
ex:founder rdfs:range ex:Person .        # [2]
```

An ontology about persons and organizations in RDFS

- As ex:ACME is the subject of a triple whose predicate is ex:founder, from [1], we can entail that ex:ACME rdf:type ex:Organization.

```
@prefix ex: <http://www.example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ex:ACME  rdf:type ex:Organization . # From [1] and rdfs2
ex:Alice rdf:type ex:Person .      # From [2] and rdfs3
```

Some of the inferrable triples

SHACL (Shapes Contraint Language)

- W3C standard for RDF graph validation
- https://www.w3.org/TR/shacl/

```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix ex: <http://www.example.org/> .

# A person must have at least one name
ex:PersonShape a sh:NodeShape ;
  sh:targetClass ex:Person ;
  sh:property [
    sh:path ex:name ;
    sh:nodeKind sh:Literal ;
    sh:minCount 1 ;
    sh:name "Name" ;
    sh:description "The name of a person"
  ] .
```

A SHACL shape graph

```
@prefix ex: <http://www.example.org/> .


ex:ACME a ex:Organization .
```

✔ ACME is not affected by PersonShape

```
ex:Alice a ex:Person .
ex:Alice ex:name "Alice" .
```

✔ Alice has the type Person and has a name

```
ex:Bob a ex:Person .
```

✗ Bob has the type Person but has no name

A data graph that is supposed to comply with the shape graph

Structure of the talk

- Writing RDF graphs
- **Shacled Turtle**
- Experiments and results

Schema to Rules Converter - RDFS

Inference rules *Suggestion rules*

```
ex:founder rdfs:domain ex:Organization .
```

```
?u ex:founder ?v                    none
-------------------------------------------
?u rdf:type ex:Organization
```

```
?u ... ...          ?u rdf:type ex:Organization
-------------------------------------------------
suggest(ex:founder)
```

```
?u ... ...          No info on ?u
-------------------------------------------------
suggest(ex:founder)
```

```
ex:founder rdfs:range ex:Person .
```

```
?u ex:founder ?v                    none
-------------------------------------------
?v rdf:type ex:Person
```

```
?u ex:founder ...                   none
-------------------------------------------
suggestAll(rdf:type, ex:Person)
```

Schema to Rules Converter - SHACL

```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix ex: <http://www.example.org/> .

# A person must have at least one name
ex:PersonShape a sh:NodeShape ;
  sh:targetClass ex:Person ;
  sh:property [
    sh:path ex:name ;
    sh:nodeKind sh:Literal ;
    sh:minCount 1 ;
    sh:name "Name" ;
    sh:description "The name of a person"
  ] .
```

Inference rules *Suggestion rules*

Shape target

none `?u rdf:type ex:Person`

`?u :pathsOf ex:PersonShape`

`ex:name` property

`?u ... ...` `?u :pathsOf ex:PersonShape`

`suggest(ex:name)`

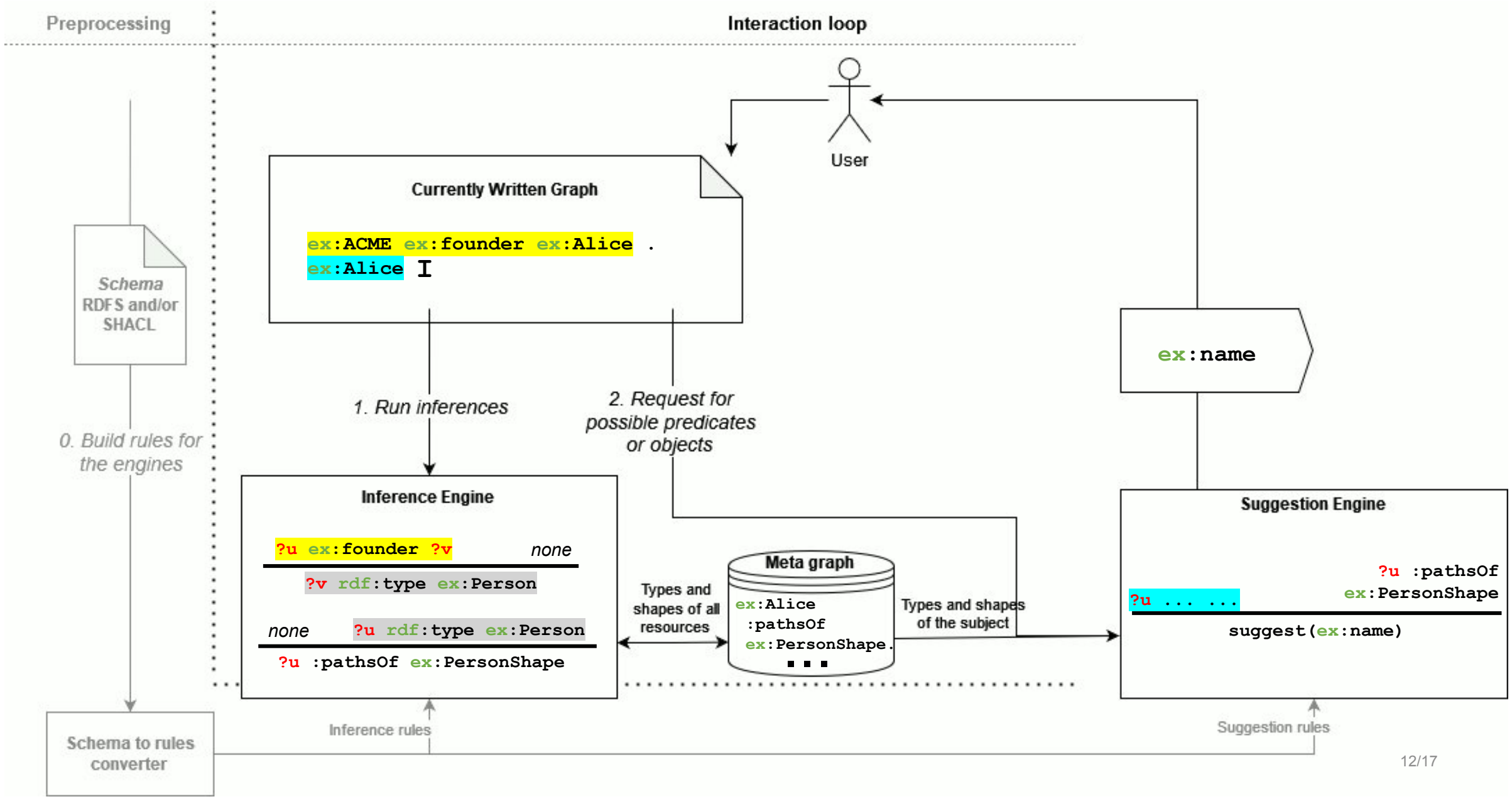
Schema to Rules Converter - Handling composite SHACL paths

```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix ex: <http://www.example.org/> .

# A person must have at least one name
ex:PersonShape a sh:NodeShape ;
  sh:targetClass ex:Person ;
  sh:property [
    sh:path ex:name ;
    sh:nodeKind sh:Literal ;
    sh:minCount 1 ;
    sh:name "Name" ;
    sh:description "The name of a person"
  ] ;
  sh:property [
    sh:path (
      ex:knows
      ex:likes
      sh:alternatePath ( ex:hates ex:likes )
    ) ;
    sh:minCount 1
  ]
  .
```

- **`ex:name`** is a predicate path
- But SHACL paths can be composite
- In real life, for SHACL paths:
  - Transform the composite path into a Finite State Automaton
  - Then transform the Finite State Automaton into rules

Shacled Turtle architecture



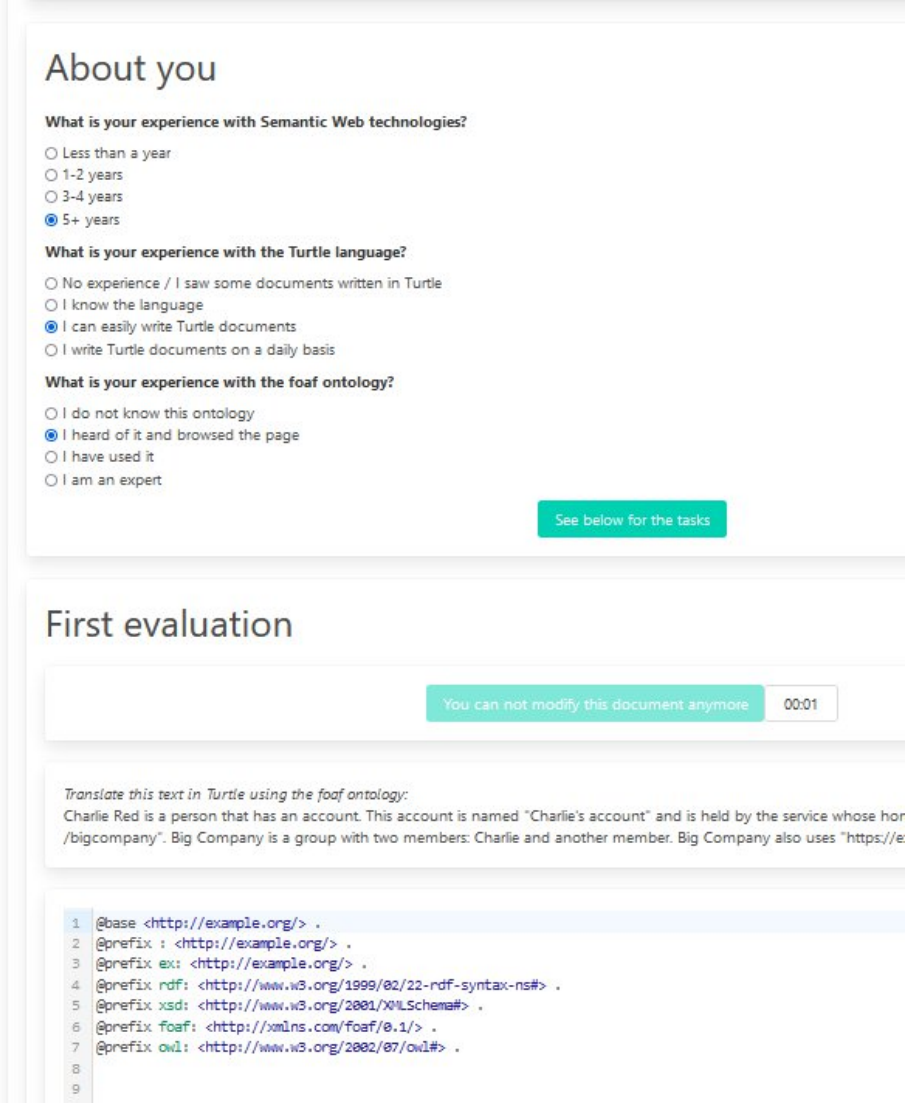
Structure of the talk

- Writing RDF graphs
- Shacled Turtle
- **Experiments and results**

Experimental setup

- Assumption: Reducing the number of suggestions will make users happier.
- Experiment
  - 34 volunteers translate two short texts
  - One text must be translated by using:
    - A naive approach (à la YASGUI)
    - With Shacled Turtle
  - Two tested ontologies:
    - schema.org
    - foaf (Friend of a friend)
  - Final questionnaire with closed- and open-ended questions

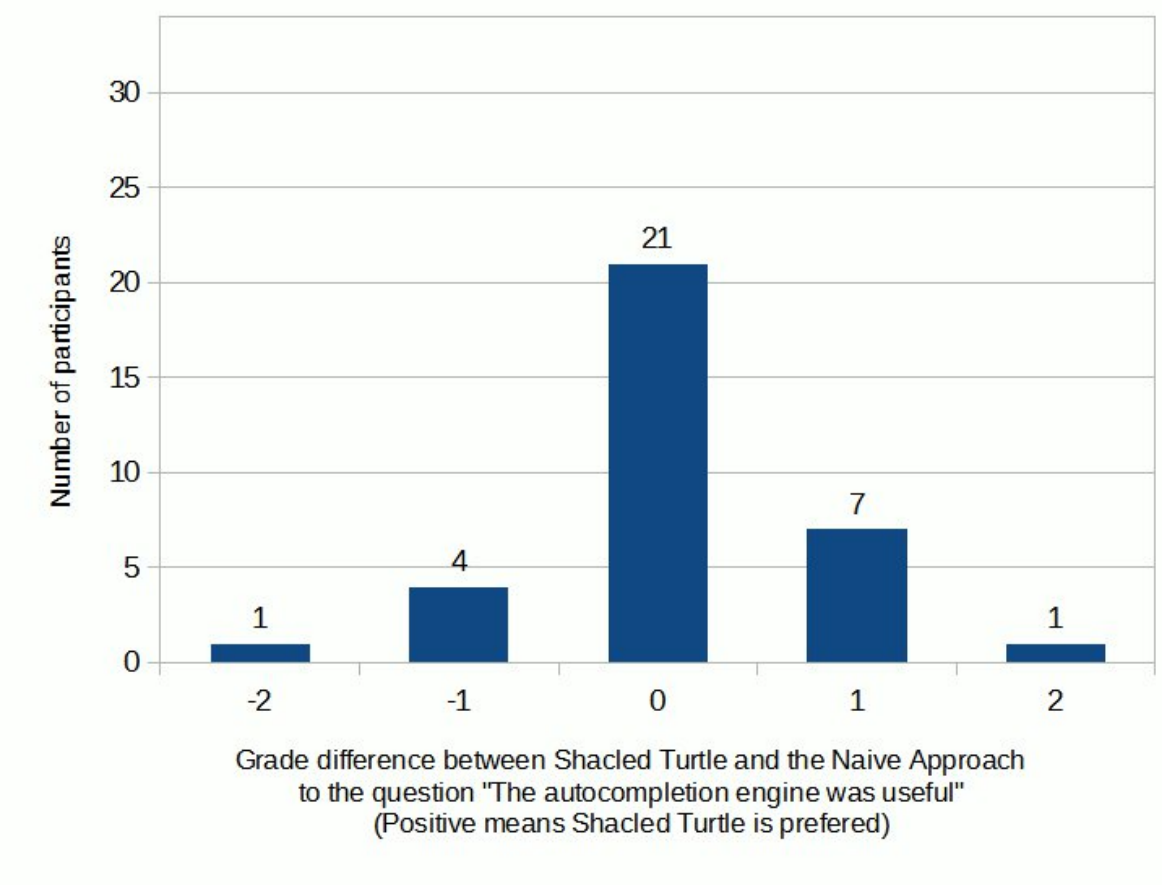
- Code, live demo and evaluation results:
  - https://github.com/BruJu/shacled-turtle-evaluation



A screenshot of the experiment platform

Results - Closed-ended questions

- More than 50% of the volunteers show no preference for one tool or the other
- Volunteers who have a preference for one tool or the other do not show a strong preference



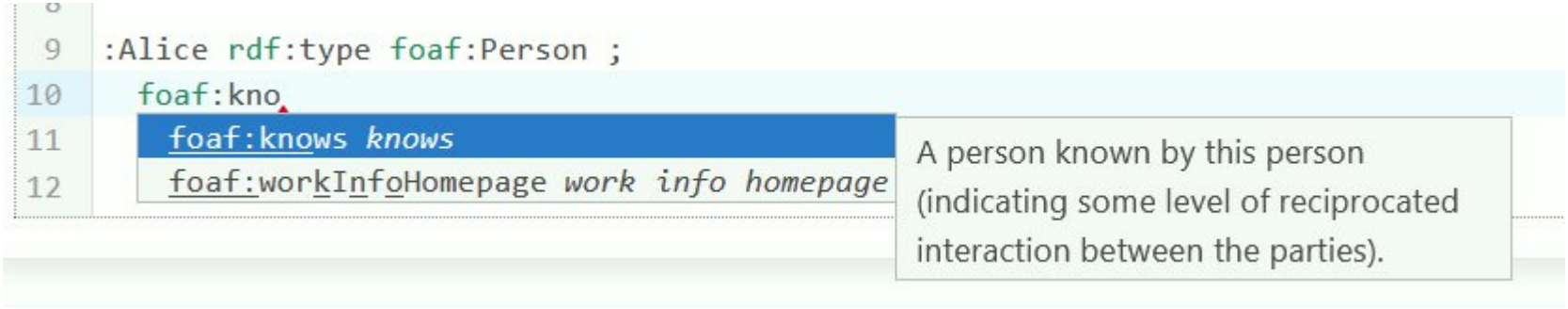
Results - Open-ended questions

In this experiment:

- Users mostly relied on filtering by name
  - 5 users did not notice the difference

- 2 volunteers appreciated that data are likely to be valid
- 3 volunteers expected more suggested terms

- 7 volunteers highlighted the importance of documentation



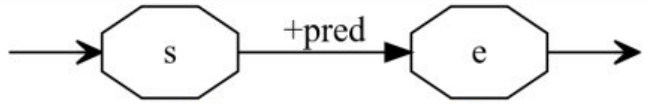
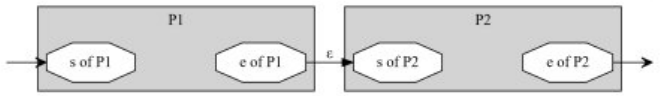
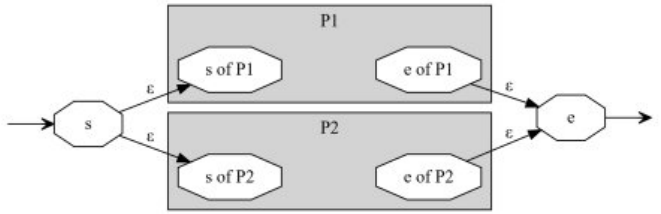
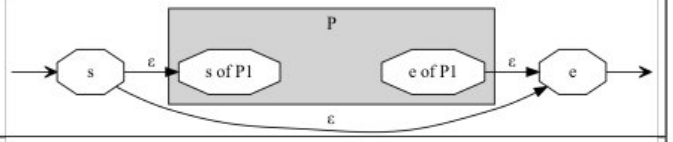
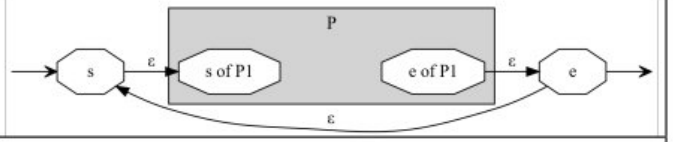
Conclusion

- Shacled Turtle: an approach to provide context-based suggestions

- Experiments results
  - Not yet exactly what users expected

- Future work
  - Use schema analysis to enrich the description of the terms
  - Rather than hiding the other terms, promote the relevant one
  - Explain why some terms are promoted

Rules generation - SHACL paths to automata

Kind and SHACL Syntax	*Regex equivalent*	*Built automaton*
Predicate `pred`	p	
Inverse `[ sh:inversePath P ]`	None	Take automaton P Inverse all transitions Transform all + into - Transform all - into +
Sequence `( P1 P2 )`	P1 P2	
Alternate `[ sh:alternatePath ( P1 P2 ) ]`	(P1 | P2)	
Zero or one `[ sh:zeroOrOnePath P ]`	P?	
One or more `[ sh:oneOrMorePath P ]`	P+	
Zero or more `[ sh:zeroOrMorePath P ]`	P*	Equivalent to (P+)?

Rules generation - Automata to rules

– m is a function that maps all states to a fresh node

Transition	*Inference rules*	*Suggestion rules*
$(S, +P, E)$	$\frac{(?u, P, ?v) \quad (?u, :pathsOf, m(S))}{(?v, :pathsOf, m(E))}$	$\frac{(?u, \ldots, \ldots) \quad (?u, :pathsOf, m(S))}{suggest(P)}$ $\frac{(?u, P, \ldots) \quad (?u, :pathsOf, m(S))}{suggestAll(:pathsOf, m(E))}$
$(S, -P, E)$	$\frac{(?u, P, ?v) \quad (?v, :pathsOf, m(S))}{(?u, :pathsOf, m(E))}$	

$$\frac{\textit{none} \qquad \texttt{?u :pathsOf ShapeOnWhichThePropertyIsOn .}}{\texttt{?u :pathsOf m(InitialStateOfTheAutomaton) .}}$$

$$\frac{\textit{none} \qquad \texttt{?u :pathsOf m(FinalStateOfTheAutomaton) .}}{\texttt{?u :pathsOf m(TargetNodeOfTheProperty) .}}$$

Hacking Schemas for Autocompletion?

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix ex: <http://www.example.org/> .

# Organization membership in RDFS
ex:founder rdfs:domain ex:Organization .


ex:founder rdfs:range ex:Person .


# A person must have at least one name
ex:PersonShape a sh:NodeShape ;
  sh:targetClass ex:Person ;
  sh:property [
    sh:path ex:name ;
    sh:nodeKind sh:Literal ;
    sh:minCount 1 ;
    sh:name "Name" ;
    sh:description "The name of a person"
  ] .
```

Running example schema

- A schema describes the relationships between classes and properties.

- ex:founder enables us to type the subject
- ex:founder is a predicate related to organizations
- After ex:founder, the user may want to use one of the existing persons.

- If we write an RDF graph that must comply with this schema, if a resource is a person, we will probably want to use the ex:name predicate.
- On the opposite, for organizations, ex:name may not be relevant.