Validation of SHACL Constraints over KGs with OWL 2 QL Ontologies via Rewriting

Ognjen "Oggy" Savković[1]
Evgeny Kharlamov[2,3]
Steffen Lamparter[4]

[1]Free University of Bozen-Bolzano, Italy
[2]University of Oslo, Oslo, Norway
[2]Bosch Centre for Artificial Intelligence, Renningen, Germany
[3]Siemens CT, Siemens AG, Munich, Germany

Knowledge Graphs (KGs)

Public Knowledge Graphs:

Knowledge Graphs (KGs)

Public Knowledge Graphs:

Commercial and Industrial Knowledge Graphs

Data Quality in KG

- Easy access to the data
- Enabler for analytics
- Flexible Schema

Data Quality in KG

- Easy access to the data
- Enabler for analytics
- Flexible Schema

Data Quality in KG

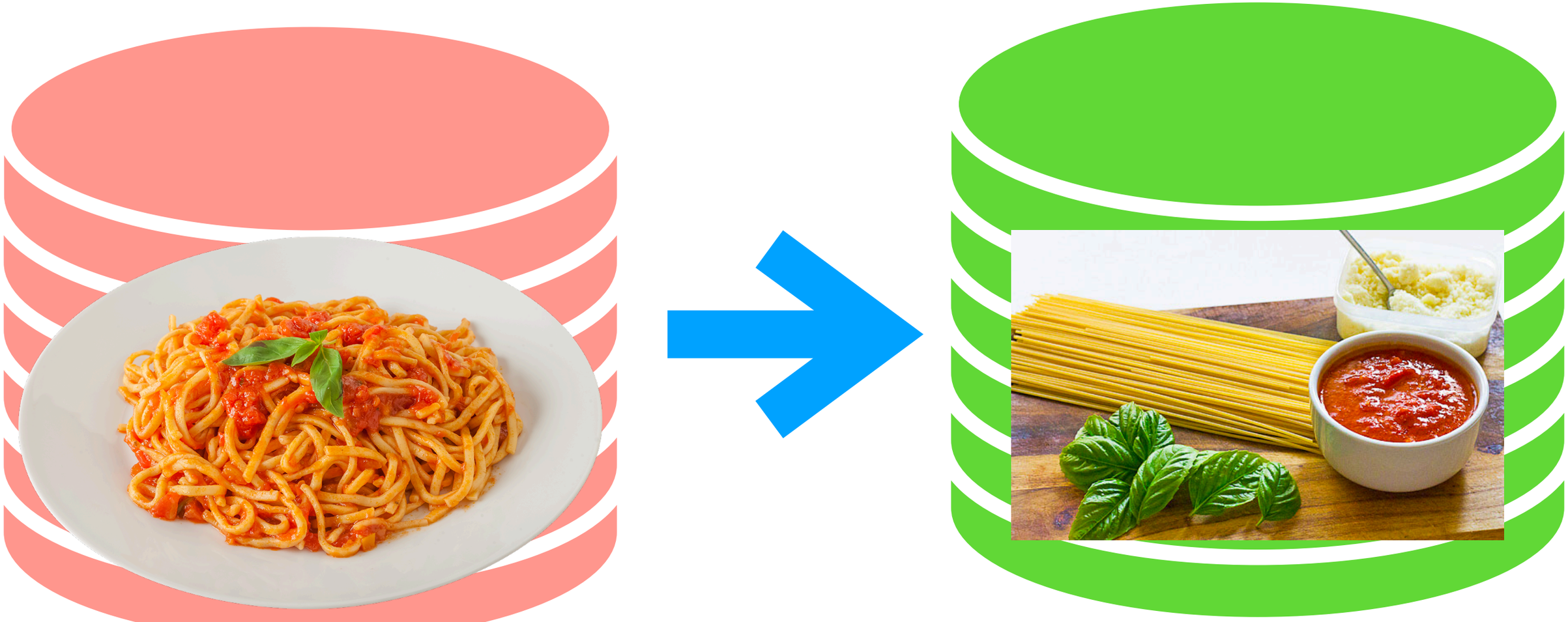
- Easy access to the data
- Enabler for analytics
- Flexible Schema

- Data Quality is very hard to maintain

Data Quality in KG

- Easy access to the data
- Enabler for analytics
- Flexible Schema

- Data Quality is very hard to maintain

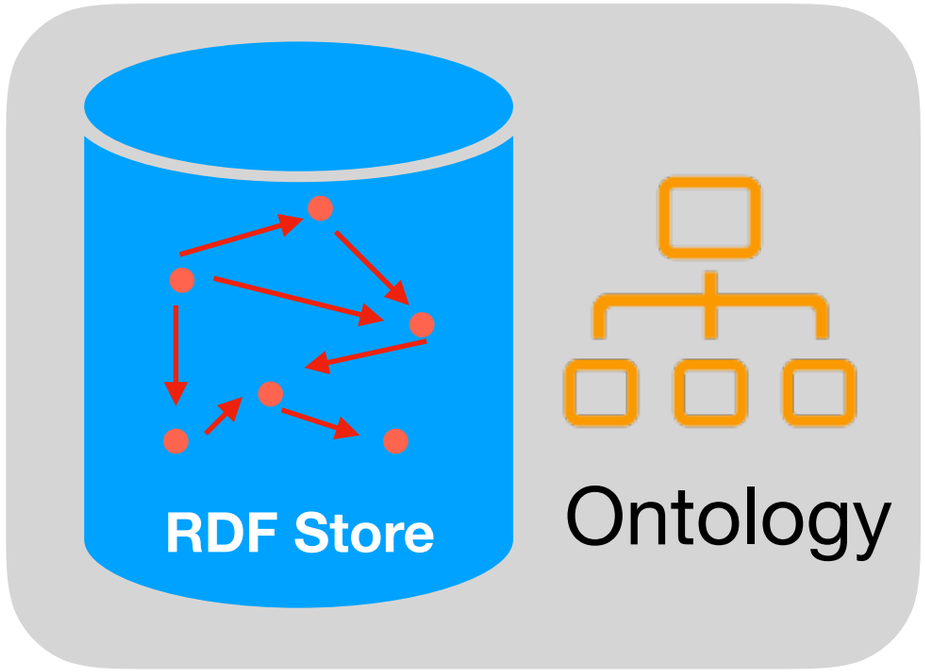


Our Setting

- **First Assumption:**
  our KGs have **OWL 2 QL ontology**
  (that may not be saturated)

- Example of an ontology:

$$O = \{\texttt{:Turbine} \sqsubseteq \texttt{:MechDevice}, \exists\texttt{:hasTuCat} \sqsubseteq \exists\texttt{:hasCat}\}$$



- **Second assumption:**
  we use **SHACL for Data Quality**

  - **W3C standard** and
    used in industry, e.g, TopBraid

Combing CWA & OWA

- Ontologies are under **OWA** (what is missing is **not false!)**
- Integrity Constraints are under **CWA** (including SHACL)
- **Third Assumption:** Take approach from the literature

  [1] *Adding Integrity Constraints to OWL,* Motik et. al.
  [2] *Integrity Constraints in OWL*, Tao et. al.

- **Semantics** of combing ontologies and constraints:
  “First apply ontology rules to “saturate” KG
  (compute minimal Herbrand model ~ canonical model)
  then check integrity constraints”

Challenges with *Saturation* and *Existing Approaches*

1. Saturated ontologies are **hard to maintain**
   (updating, deleting, etc)

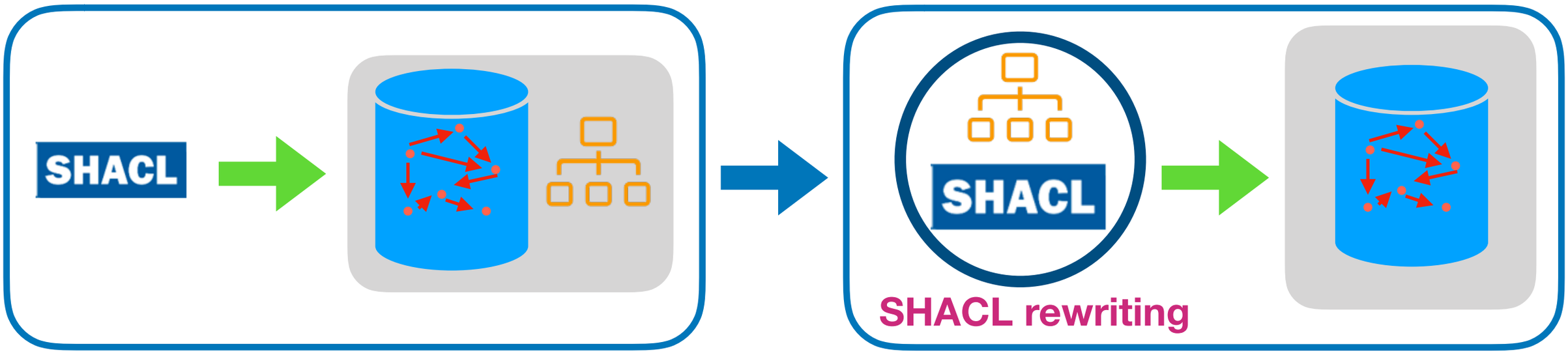
2. **Not** all data are **needed** when checking constraints

3. *OWL 2 QL saturation* means computing **canonical model**
   —> which may be **infinite** —> **not feasible** in practice

- Proposals like: [1] *Adding Integrity Constraints to OWL,* Motik et. al.
  - Reduces the problem to **(complex) logic programs**
  - Requires **different engine** to check to validate constraints

Our Approach

- Instead, we **combine** **constraints** and **ontology** into
  new set of constraints and **evaluate** over RDF



- Looks similar to **OBDA setting**,
  but **SHACL** and **conjunctive queries** are incomparable

- Since **OWL 2 QL** has **recursion** ->
  we need would need **recursive SHACL** constraints

Example of Rewriting

- Given a SHACL constraint with **a *target*** *and* ***shape def:***

$$\tau_s = \texttt{:MechDevice}(x) \qquad \phi_s = (\geq_1 \texttt{:hasCat}.\top)$$

- Then an DL-Lite ontology with two **assertions**:

$$O = \{\texttt{:Turbine} \sqsubseteq \texttt{:MechDevice}, \exists\texttt{:hasTuCat} \sqsubseteq \exists\texttt{:hasCat}\}$$

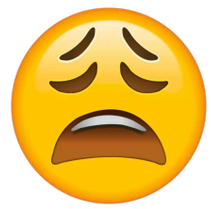
- **Rewritten** shape:

$$\tau'_s = \texttt{:MechDevice}(x) \vee \texttt{:Turbine}(x)$$
$$\phi'_s = (\geq_1 \texttt{:hasCat}.\top) \vee (\geq_1 \texttt{:hasTuCat}.\top)$$

The main contributions

This is a **foundational work**. We

- Studied conditions when **SHACL-rewritings exists**
- Shown that in most complex case
  it is not possible



- Found **two practical fragments** when SHACL-rewriting exists and established their **complexity bounds**

The rest of the talk

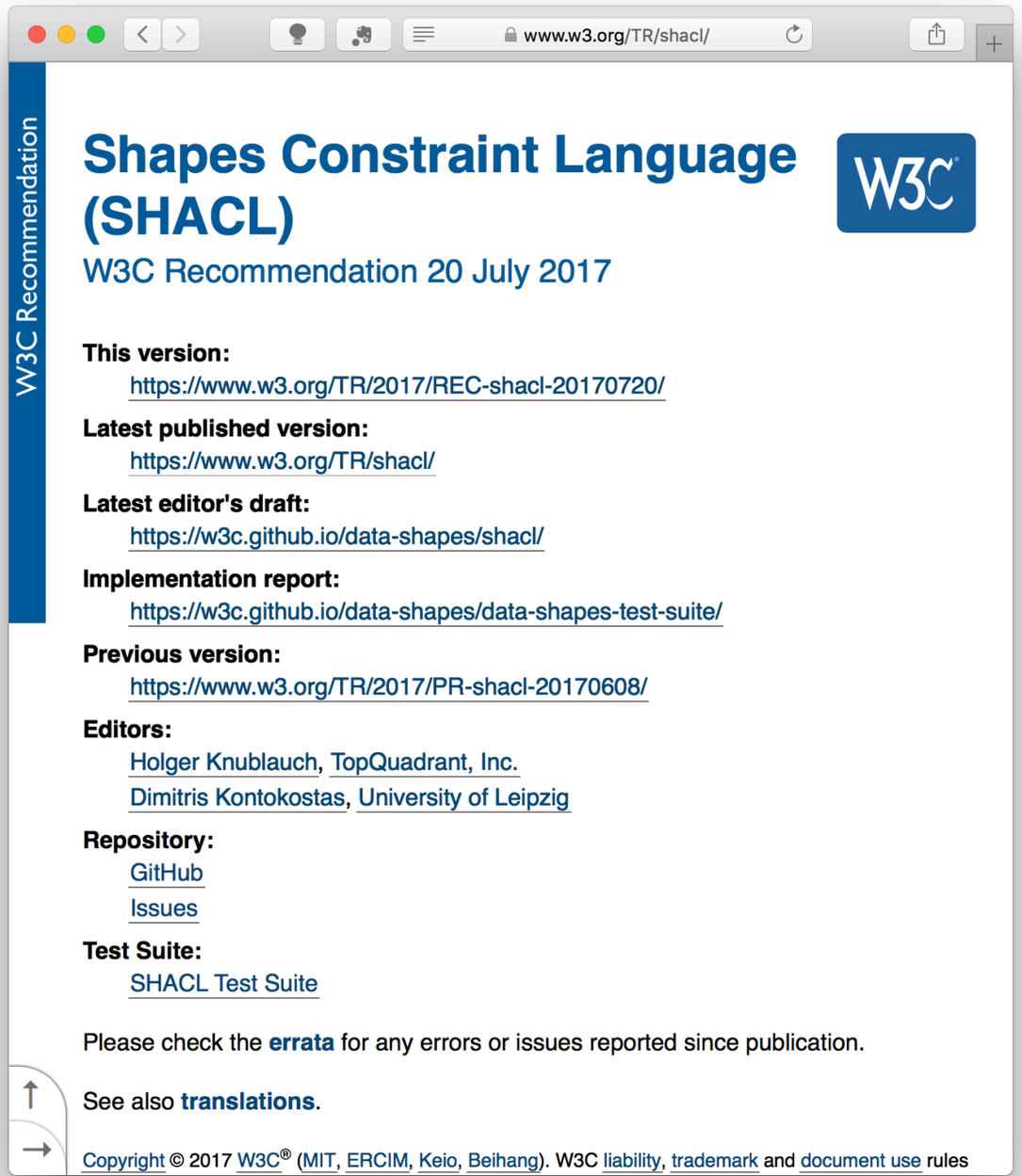
- **Preliminaries: SHACL (core) and OWL 2 QL**
- **Technical Results**
- **Conclude**

SHACL (core)

in 2 min

Shapes Constraint Language (SHACL)

- **Constraint language** for RDF
- **W3C** recommendation since **July 2017**



SHACL for local validations

```
:PlaneTicketShape
    a sh:NodeShape ;
    sh:targetClass :PlaneTicket ;

    sh:property [
        sh:maxCount 1 ;
        sh:path ex:passengerName;
        sh:datatype xsd:string .
    ];
    sh:property [
        sh:path :flightcode ;
        sh:maxCount 1 ;
        sh:datatype xsd:integer .
    ].
…
```

SHACL for local validations

Name of the constraint, aka **Shape**

```
:PlaneTicketShape
    a sh:NodeShape ;
    sh:targetClass :PlaneTicket ;

    sh:property [
        sh:maxCount 1 ;
        sh:path ex:passengerName;
        sh:datatype xsd:string .
    ];
    sh:property [
        sh:path :flightcode ;
        sh:maxCount 1 ;
        sh:datatype xsd:integer .
    ].
…
```

SHACL for local validations

```
:PlaneTicketShape
    a sh:NodeShape ;
    sh:targetClass :PlaneTicket ;
    sh:property [
        sh:maxCount 1 ;
        sh:path ex:passengerName;
        sh:datatype xsd:string .
    ];
    sh:property [
        sh:path :flightcode ;
        sh:maxCount 1 ;
        sh:datatype xsd:integer .
    ].
…
```

Identifies “target nodes”
to be validated against the shape

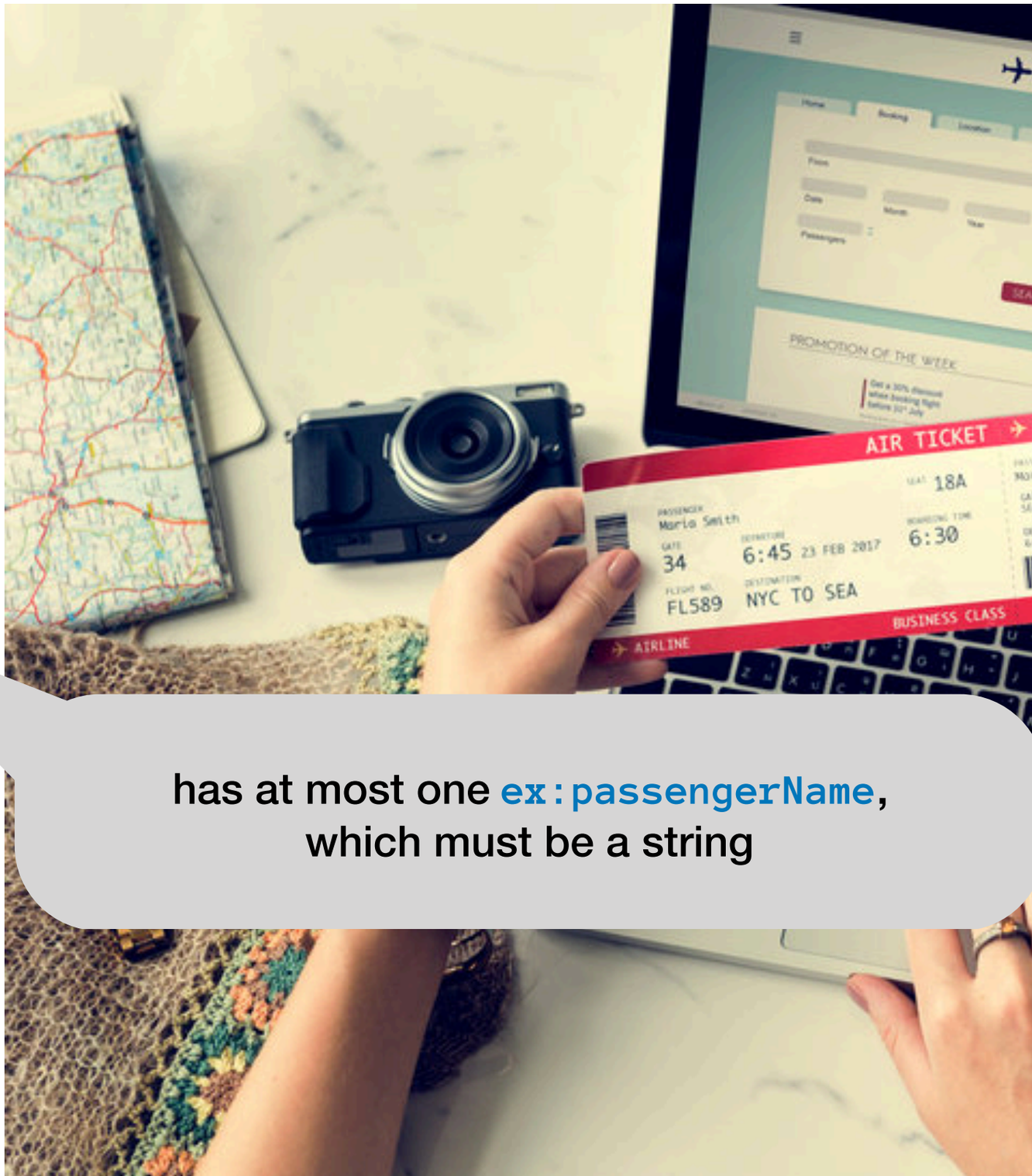
SHACL for local validations

```
:PlaneTicketShape
    a sh:NodeShape ;
    sh:targetClass :PlaneTicket ;

    sh:property [
        sh:maxCount 1 ;
        sh:path ex:passengerName;
        sh:datatype xsd:string .
    ];
    sh:property [
        sh:path :flightcode ;
        sh:maxCount 1 ;
        sh:datatype xsd:integer .
    ].
…
```

Constraints!

SHACL for local validations

```
:PlaneTicketShape
    a sh:NodeShape ;
    sh:targetClass :PlaneTicket ;

  sh:property [
      sh:maxCount 1 ;
      sh:path ex:passengerName ;
      sh:datatype xsd:string .
  ].
  sh:property [
      sh:path :flightcode ;
      sh:maxCount 1 ;
      sh:datatype xsd:integer .
  ].
…
```



has at most one `ex:passengerName`, which must be a string

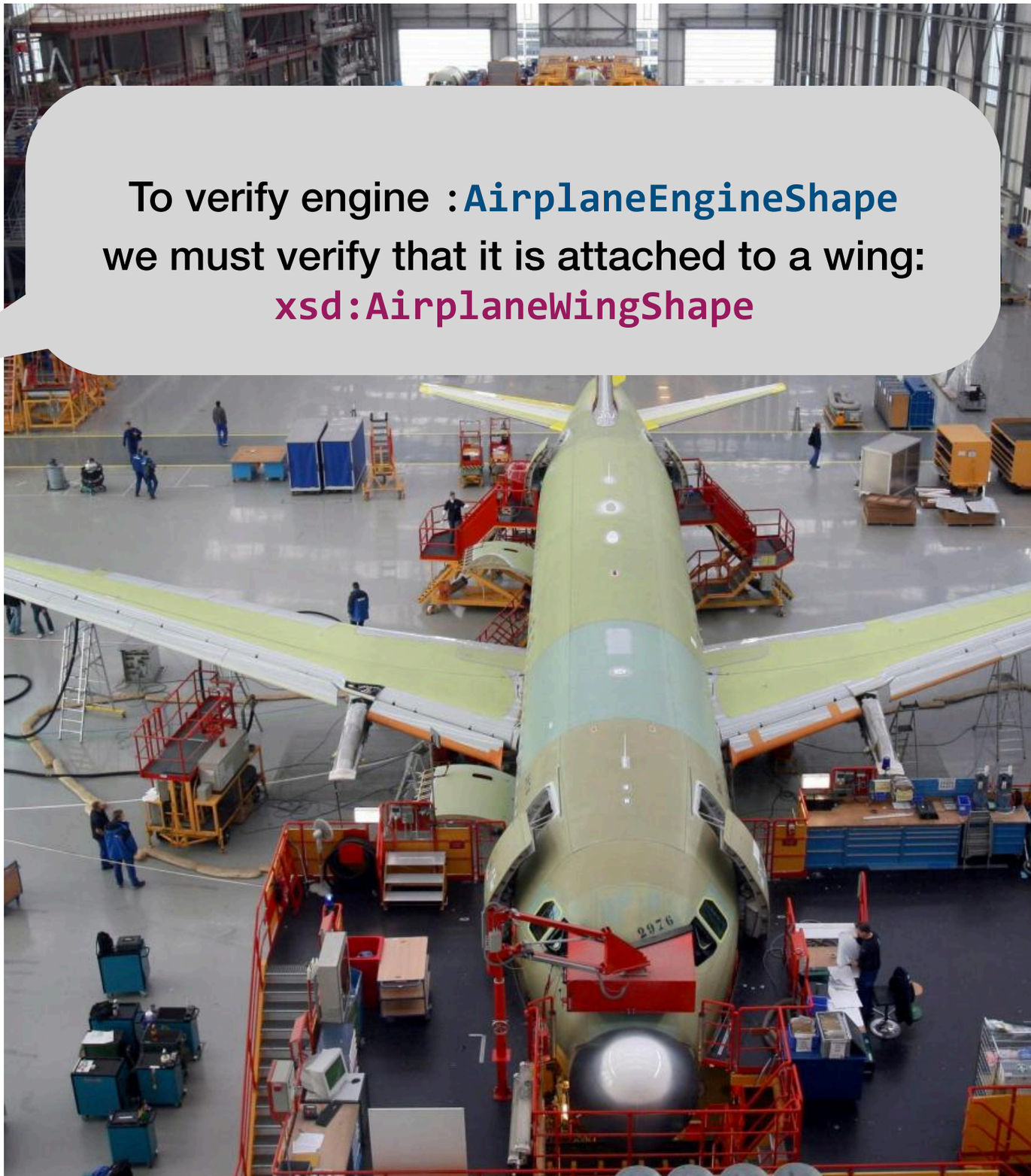
SHACL for propagated validations

```
:AirplaneEngineShape
  a sh:NodeShape ;
  sh:targetClass : AirplaneEngine ;
  sh:property [
     sh:maxCount 1 ;
     sh:path ex:attachedTo;
     sh:node xsd:AirplaneWingShape .
  ];
```

SHACL for propagated validations

```
:AirplaneEngineShape
  a sh:NodeShape ;
  sh:targetClass : AirplaneEngine ;
  sh:property [
     sh:maxCount 1 ;
     sh:path ex:attachedTo;
     sh:node xsd:AirplaneWingShape .
  ];
```

To verify engine :AirplaneEngineShape
we must verify that it is attached to a wing:
xsd:AirplaneWingShape

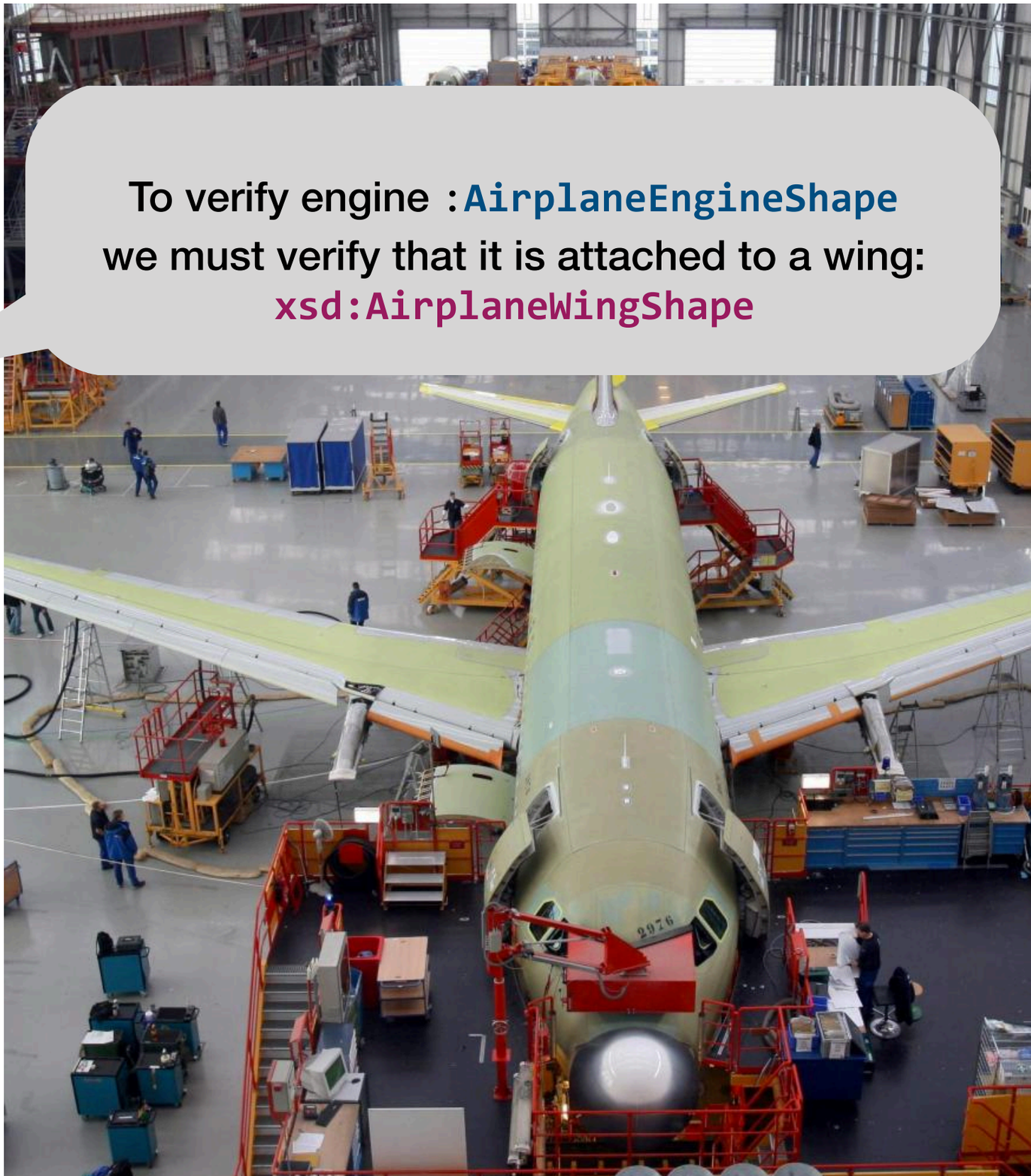


SHACL for propagated validations

```
:AirplaneEngineShape
  a sh:NodeShape ;
  sh:targetClass : AirplaneEngine ;
  sh:property [
     sh:maxCount 1 ;
     sh:path ex:attachedTo;
     sh:node xsd:AirplaneWingShape .
  ];
```

```
:AirplaneWingShape
    a sh:NodeShape ;
    sh:property [
      sh:minCount 1 ;
      sh:path ex:hasEngine;
      sh:node xsd:AirplaneEngineShape .
    ];
```

To verify engine :AirplaneEngineShape
we must verify that it is attached to a wing:
xsd:AirplaneWingShape



SHACL for propagated validations

```
:AirplaneEngineShape
   a sh:NodeShape ;
   sh:targetClass :Airplane ;
   sh:property [
      sh:maxCount 4 ;
      sh:path :hasEngine ;
      sh:node :AirplaneEngineWing ;
    ];
```

```
:AirplaneEngineWing
    a sh:NodeShape ;
    sh:property [
      sh:minCount 1 ;
      sh:path :isPartOf ;
      sh:node :AirplaneShape ;
    ];
```

FROM: https://www.w3.org/TR/shacl/

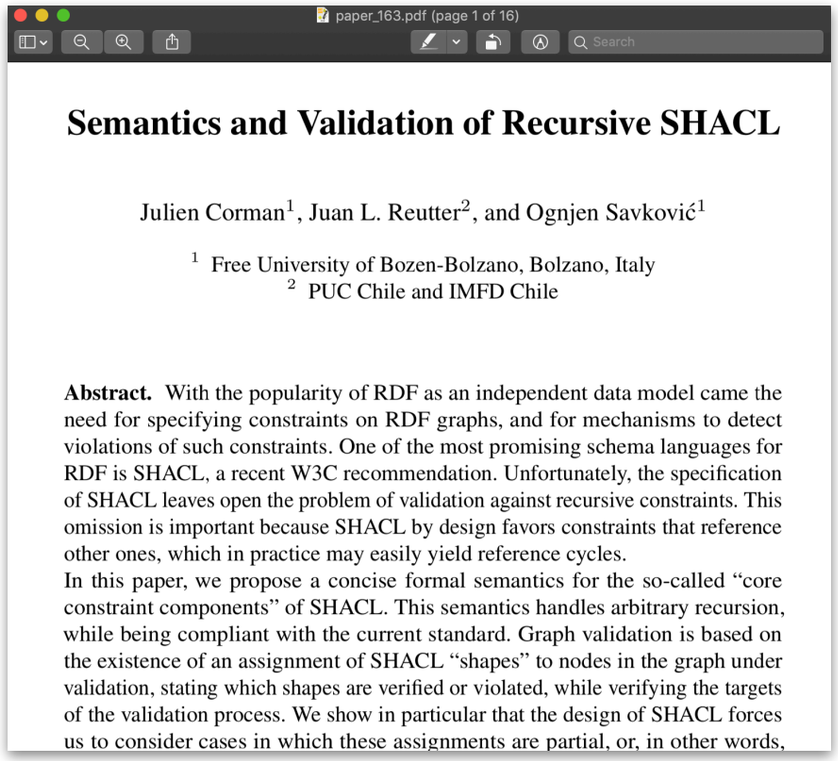
➡ **Not 100% formal semantics**
partly SPARQL, partly textual
mixing intention with implementation

➡ **Circular (recursive) constraints?**
validation explicitly undefined

```
:Airp
  a s
  sh:
  sh:

  ]
```

```
:Airp
  a
  s
    sh:
  ];
```

ISWC 2018
MONTEREY, CA OCTOBER 8-12


paper_163.pdf (page 1 of 16)

Search

Semantics and Validation of Recursive SHACL

Julien Corman[1], Juan L. Reutter[2], and Ognjen Savković[1]

[1] Free University of Bozen-Bolzano, Bolzano, Italy
[2] PUC Chile and IMFD Chile

Abstract. With the popularity of RDF as an independent data model came the need for specifying constraints on RDF graphs, and for mechanisms to detect violations of such constraints. One of the most promising schema languages for RDF is SHACL, a recent W3C recommendation. Unfortunately, the specification of SHACL leaves open the problem of validation against recursive constraints. This omission is important because SHACL by design favors constraints that reference other ones, which in practice may easily yield reference cycles.
In this paper, we propose a concise formal semantics for the so-called "core constraint components" of SHACL. This semantics handles arbitrary recursion, while being compliant with the current standard. Graph validation is based on the existence of an assignment of SHACL "shapes" to nodes in the graph under validation, stating which shapes are verified or violated, while verifying the targets of the validation process. We show in particular that the design of SHACL forces us to consider cases in which these assignments are partial, or, in other words,

"Semantics and Validation of Recursive SHACL"

- We defined **abstract syntax** of SHACL **core**
- Introduced semantic for **recursive SHACL** that extends **standard semantics**
- Studied general **validation algorithms** and **tractable** fragments

SHACL Constraints
Abstract Syntax

Constraints

$$\phi \quad ::= \quad \top \mid s' \mid c \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \geq_n R.\phi \mid \leq_n R.\phi \mid \mathsf{EQ}(r_1, r_2),$$

s' a shape name, c an IRI, r_1, r_2 are property paths, $n \in \mathbb{N}^+$

Example of **Targets** and **Shape Definitions**

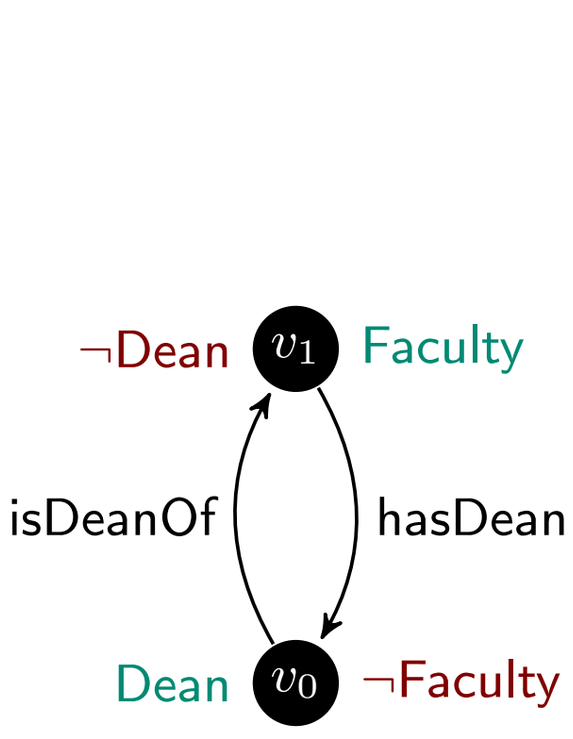
$$\tau_{s_2} = \exists y(\texttt{:hasTuCat}(x, y)), \qquad \phi_{s_2} = (\geq_1 \texttt{a.:Turbine}),$$

$$\tau_{s_1} = \exists y(\texttt{:deplAt}(x, y)), \qquad \phi_{s_1} = (\geq_1 \texttt{:hasCat.}\top)$$

Shapes Validation

“Labelling graph with shapes and check if the labelling is consistent”

Shapes Validation



Constraints

$\mathrm{def}(\mathsf{Dean}) \doteq \geq_1 \mathsf{isDeanOf}.\mathsf{Faculty}$

$\mathrm{def}(\mathsf{Faculty}) \doteq \geq_1 \mathsf{hasDean}.\mathsf{Dean}$

Target

$\mathsf{Dean}(v_0)$

Grounding constraint evaluation

Evaluate the constraints given a shape assignment

Assignment 3

Complies with target and constraints

"Labelling graph with shapes and check if the labelling is consistent"

OWL 2 QL in 1 slide

Modeling construct	*DL-Lite*	FOL formalization
ISA on classes	$A_1 \sqsubseteq A_2$	$\forall x(A_1(x) \rightarrow A_2(x))$
. . . and on relations	$R_1 \sqsubseteq R_2$	$\forall x, y(R_1(x, y) \rightarrow R_2(x, y))$
Disjointness of classes	$A_1 \sqsubseteq \neg A_2$	$\forall x(A_1(x) \rightarrow \neg A_2(x))$
. . . and of relations	$R_1 \sqsubseteq \neg R_2$	$\forall x, y(R_1(x, y) \rightarrow \neg R_2(x, y))$
Domain of relations	$\exists P \sqsubseteq A_1$	$\forall x(\exists y(P(x, y)) \rightarrow A_1(x))$
Range of relations	$\exists P^- \sqsubseteq A_2$	$\forall x(\exists y(P(y, x)) \rightarrow A_2(x))$
Mandatory participation *(min card = 1)*	$A_1 \sqsubseteq \exists P$	$\forall x(A_1(x) \rightarrow \exists y(P(x, y)))$
	$A_2 \sqsubseteq \exists P^-$	$\forall x(A_2(x) \rightarrow \exists y(P(y, x)))$
. . .	. . .	. . .

Technical Part

OWL 2 QL

+

full SHACL

(negative case)

Non-Existence of Rewriting

- Interaction between "**existential**" rules in OWL 2 QL, and

$$V \sqsubseteq \exists R.C$$

- **negation** or **negative cardinality**

$$\neg\phi \qquad \leq_n R.\phi$$

- allows to encode **(non) 3 coloring problem** to our problem
- complexity of the becomes is **DP-hard**
- Complexity of checking recursive SHACL is **NP-complete**

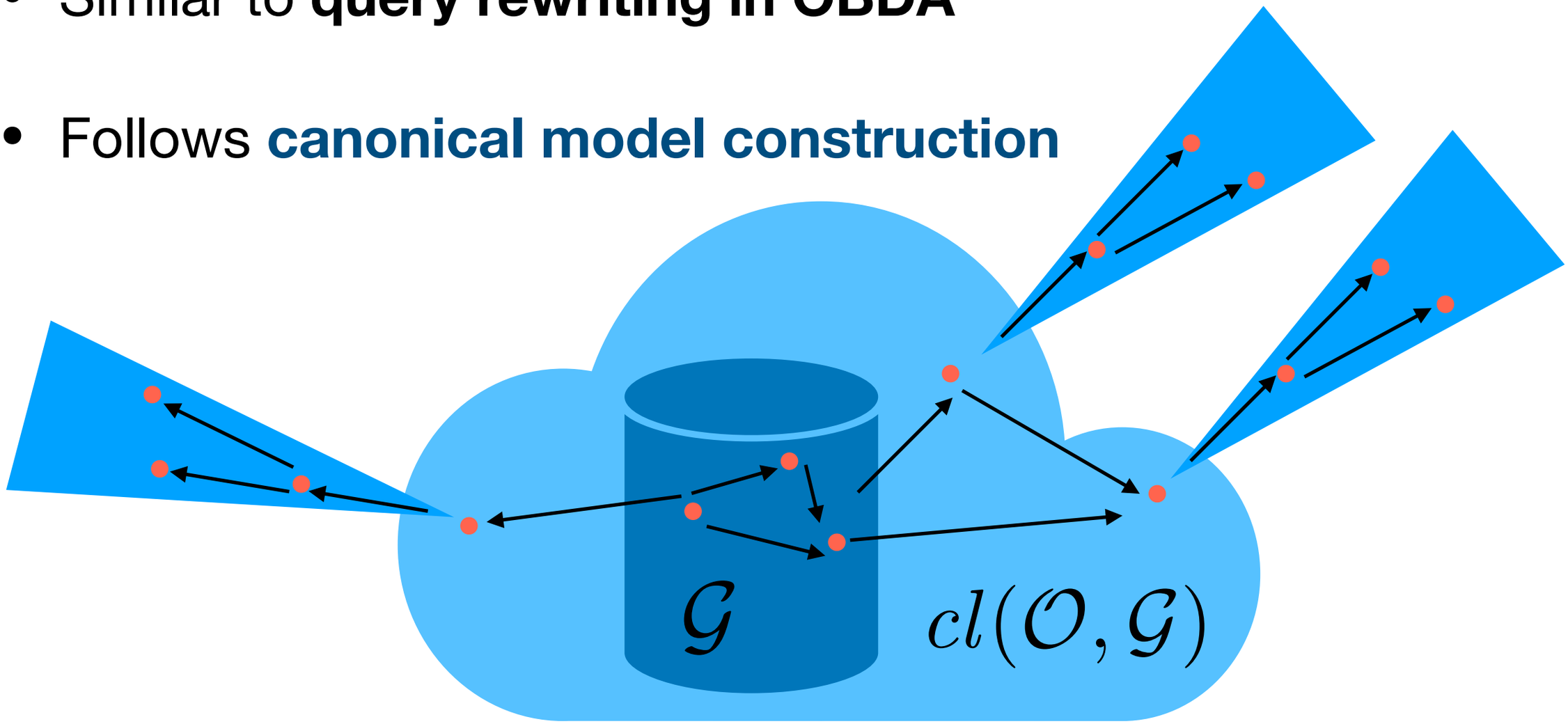
OWL 2 QL

+

positive SHACL

General Idea of Rewriting

- We do rewriting by “**injecting**” OWL statements into SHACL
- Similar to **query rewriting in OBDA**
- Follows **canonical model construction**



Rewriting Overview /1

1. Rewriting of **shape targets** via “**Perfect Reformulation**”

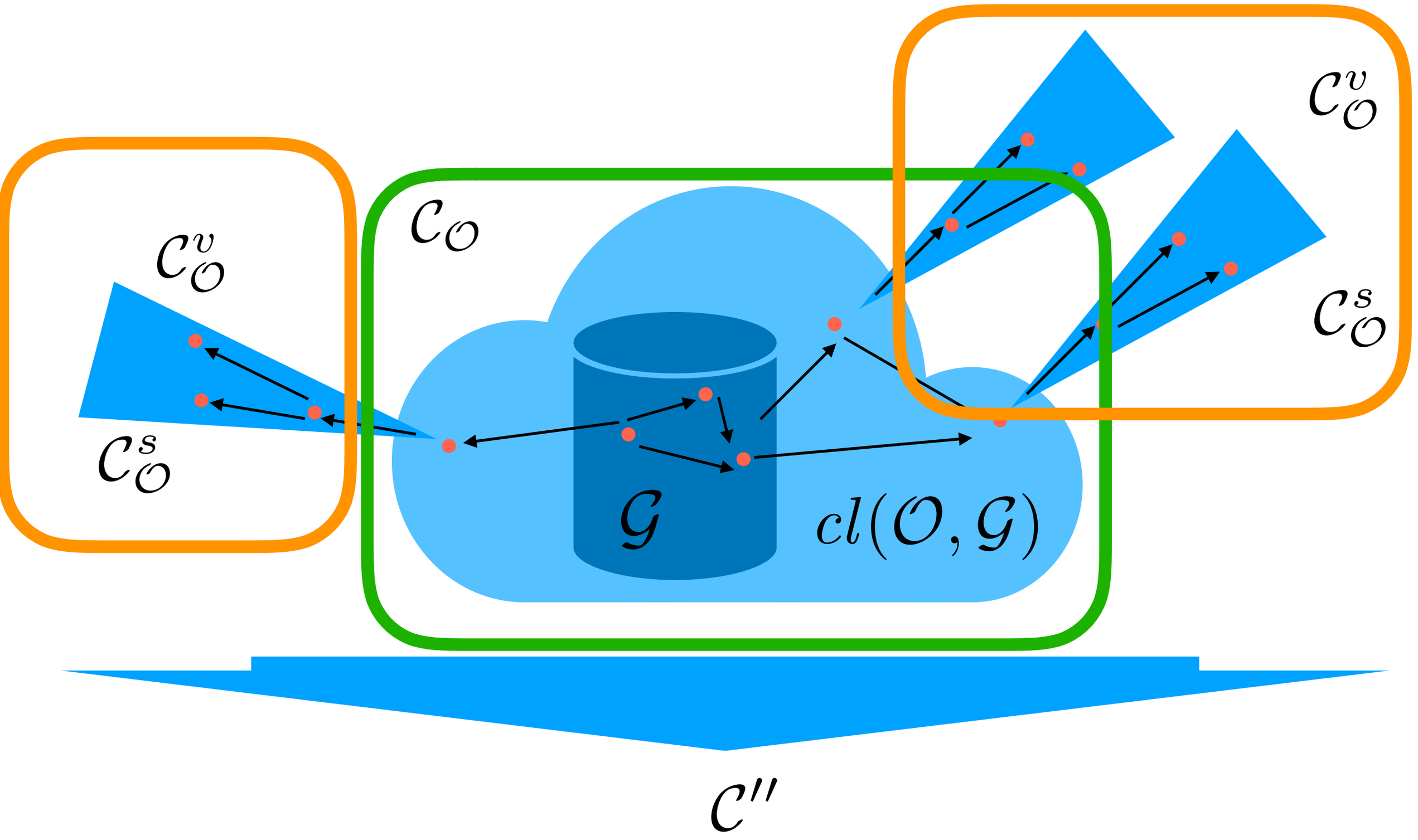
2. Rewriting consists of 3 kinds of auxiliary shapes:
    - “encode” concepts over **active domain** $\mathcal{C}_{\mathcal{O}}$
    - “encode” concepts over **virtual domain** $\mathcal{C}^{v}_{\mathcal{O}}$
    - “encode” property **successors** in ontology $\mathcal{C}^{s}_{\mathcal{O}}$
3. “rewritten” original shapes $\mathcal{C}''$

Rewriting Shape Targets

- Rewriting of shape targets via “Perfect Reformulation”

$$[\![\tau_s]\!]^{\langle \mathcal{O}, \mathcal{G} \rangle} = [\![\textsc{PerfRef}(\tau_s, \mathcal{O})]\!]^{\mathcal{G}}$$

Rewriting Overview /2



Concept into Shapes $\mathcal{C}_{\mathcal{O}}$

- **Base case** of our rewriting: encoding of ontology concept

A

$$\tau_{s_A} = A$$

$$\phi_{s_A} = (\geq_1 \mathsf{a}.A) \vee \bigvee_{\mathcal{O} \models C \sqsubseteq A} s_C ;$$

$\exists R$

$$\tau_{s_{\exists R}} = \exists R$$

$$\phi_{s_{\exists R}} = (\geq_1 R.\top) \vee \bigvee_{\mathcal{O} \models C \sqsubseteq \exists R} s_C \vee \bigvee_{\mathcal{O} \models R' \sqsubseteq R} s_{\exists R'}$$

(`:t852, a, :Turbine`) is in the $cl(\mathcal{O}, \mathcal{G})$ iff

`:t852` is valid in shape for turbine $s_{\texttt{:Turbine}}$

Rewriting and Virtual Shapes

- Each **shape** is rewritten into "**prime" version** or **virtual one**:

$\phi_s = s_1 \wedge s_2,$ → $\phi_{s'} = s'_1 \wedge s'_2 \quad \vee s^{virtual}$

$\phi_s = s_1 \vee s_2$ → $\phi_{s'} = s'_1 \vee s'_2 \quad \vee s^{virtual}$

$\phi_s = (\geq_k R.s_1)$ → $\phi'_s = (\geq_k R'.s'_1) \vee s^{virtual}$

$R' = R_1 | \ldots | R_i | \ldots | R_n$

$\mathcal{O} \models R_i \sqsubseteq R$

$\phi_s = \mathrm{EQ}(r_1, r_2)$ → $\phi_{s'} = \mathrm{EQ}(r'_1, r'_2)$

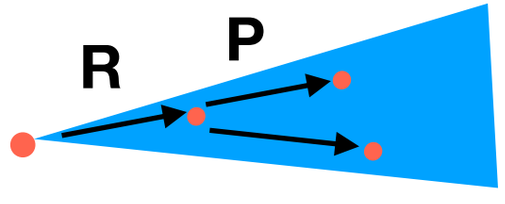
Virtual Shapes

- We introduce virtual shape both in the same way for connectors $\wedge, \vee$
- Except for $\phi_s = (\geq_k R.s_1)$

$$\phi_{s^{virtual}} = s^{virtual}_{\exists R} \wedge s^{virtual}_{1} \wedge s_{\exists R, s_1}$$

$\phi_{s_1} = s_2 \wedge \ s_3$ (and similarly for $\vee$) → $\phi_{s_{\exists R, s_1}} = s_{\exists R, s_2} \wedge s_{\exists R, s_3}$

$\phi_{s_1} = (\geq_k P.s_2)$ → $\phi_{\exists R, s_1} = s^{succ}_{R,P}$



- $\phi_{\exists R, s_1} = s^{succ}_{R,P}$ is **true** in **node** iff P follows R in "**fresh" part of the canonical model**

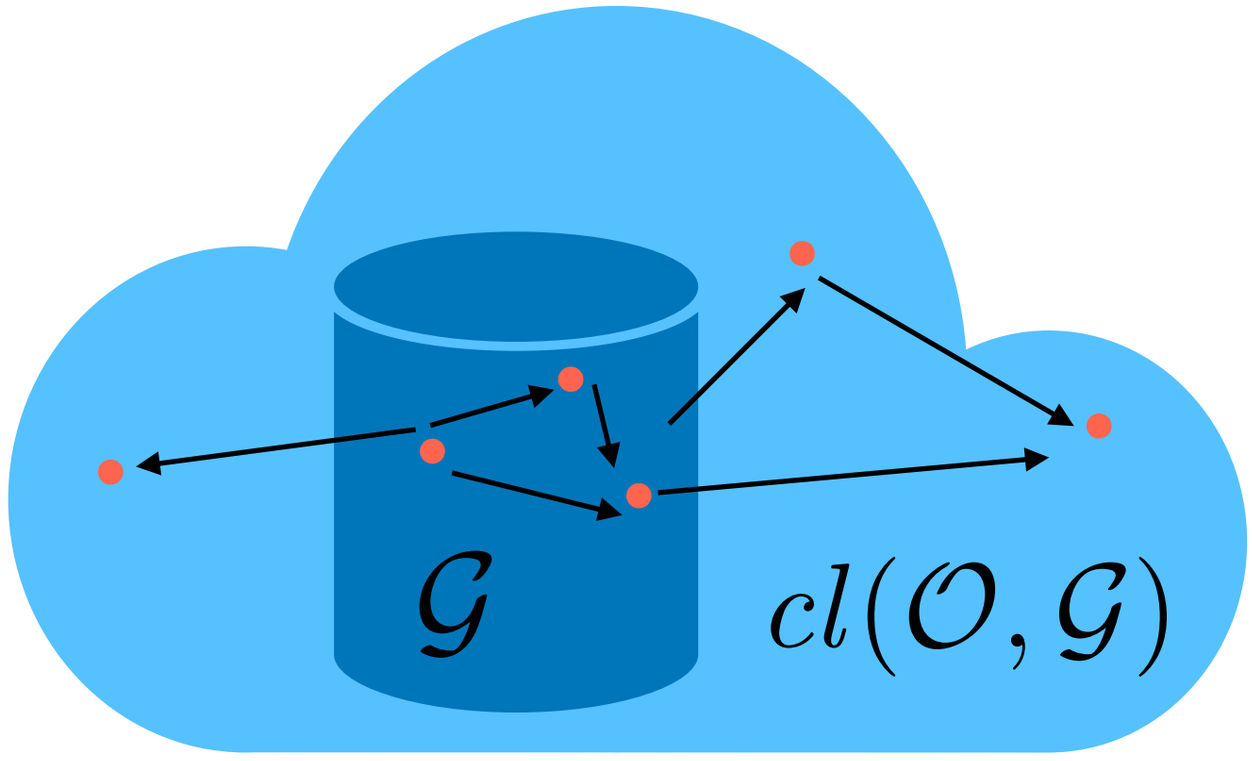
OWL 2 QL (without existential on the right-hand-side)

+

full SHACL

OWL 2 QL without Existentials

- **“Simpler"** case since canonical model **is finite**
- Corresponds to **RDFs** schemas, thus **useful** as well
- Rewriting is similar, except no complex “**virtual**” shapes



Summing Up

Complexity Table and Next Steps

Complexity Table and Next Steps

	Full SHACL	Positive SHACL
OWL 2 QL	DP-hard (not rewritable)	Ptime-complete

Complexity Table and Next Steps

	Full SHACL	Positive SHACL
OWL 2 QL	DP-hard (not rewritable)	Ptime-complete
RDFS	NP-complete	Ptime-complete

Complexity Table and Next Steps

	Full SHACL	Positive SHACL
OWL 2 QL	DP-hard (not rewritable)	Ptime-complete
RDFS	NP-complete	Ptime-complete

- Do SHACL check before KG materialization
- Richer ontologies (e.g, **OWL 2 EL**)

Complexity Table and Next Steps

	Full SHACL	Positive SHACL
OWL 2 QL	DP-hard (not rewritable)	Ptime-complete
RDFS	NP-complete	Ptime-complete

- Do SHACL check before KG materialization
- Richer ontologies (e.g, **OWL 2 EL**)

PostDoc and PhD Positions at KRDB Research Centre in Bolzano

This year we are starting **6 new projects** on **(virtual) KGs / OBDA**

- **Province Bolzano project** on **validation and learning SHACL** (1y2m)
- **EU Project INODE** on **data management** infrastructures (3 y)
- **Italian basic research project HOPE** on **open data publishing** (3 y)
- CHIST-ERA **EU Project PACMEL** on **OBDA for process mining** (2 y)
- **ESF Project IDEE** on **data integration** in the energy sector (3 y)
- **Industrial project** on **temporal OBDA** (9 months)

We are hiring 5-6 new postdocs on these projects
We are also opening 15 4-year PhD positions with bursaries

For general info, contact asap **diego.calvanese@unibz.it** or **ognjen.savkovic@unibz.it** for **SHACL related project**