Bridging DLs and SHACL Validation

A DL perspective

Magdalena Ortiz

TU Wien
Joint work with Anouk Oudshoorn, Mantas Šimkus, Shqiponja Ahmetaj

Table of contents

Introduction to Description Logics

Description Logics (DLs)

- Family of logics well-suited for describing structured knowledge through ***concepts*** (classes) and ***roles*** (relationships)
- Decidable fragments of first order logic (with a funny* syntax)

Description Logics (DLs)

- Family of logics well-suited for describing structured knowledge through ***concepts*** (classes) and ***roles*** (relationships)
- **Decidable fragments of first order logic** (with a funny* syntax)

$$
\begin{aligned}
\mathsf{AfricanCountry} &\sqsubseteq \mathsf{Country} \\
\mathsf{Country} &\sqsubseteq \exists\, \mathsf{hasCapital}.\mathsf{City} \\
\mathsf{City} &\sqsubseteq \exists\, \mathsf{locatedIn}.\mathsf{Country} \\
\mathsf{City} &\sqsubseteq \leq 1\, \mathsf{locatedIn}.\mathsf{Country} \\
\mathsf{AfricanCountry} &\equiv \mathsf{Country} \sqcap \exists \mathsf{locatedIn}.\{\mathsf{Africa}\} \\
\mathsf{hasCapital}^{-} &\sqsubseteq \mathsf{locatedIn}
\end{aligned}
$$

$$
\mathit{AfricanCountry}(\mathit{SouthAfrica}), \mathit{Country}(\mathit{Botswana}), \mathit{locatedIn}(\mathit{Botswana}, \mathit{Africa})
$$

Description Logics (DLs)

- Family of logics well-suited for describing structured knowledge through ***concepts*** (classes) and ***roles*** (relationships)
- **Decidable fragments of first order logic** (with a funny* syntax)

$$
\begin{aligned}
\text{AfricanCountry} &\sqsubseteq \text{Country} \\
\text{Country} &\sqsubseteq \exists\, \text{hasCapital}.\text{City} \\
\text{City} &\sqsubseteq \exists\, \text{locatedIn}.\text{Country} \\
\text{City} &\sqsubseteq \leq 1\, \text{locatedIn}.\text{Country} \\
\text{AfricanCountry} &\equiv \text{Country} \sqcap \exists \text{locatedIn}.\{\text{Africa}\} \\
\text{hasCapital}^{-} &\sqsubseteq \text{locatedIn}
\end{aligned}
$$

$$\mathit{AfricanCountry}(\mathit{SouthAfrica}), \mathit{Country}(\mathit{Botswana}), \mathit{locatedIn}(\mathit{Botswana}, \mathit{Africa})$$

- DL **knowledge base** = TBox (axioms) + **ABox** (data)

* Yes, it is an acquired taste.

DLs as FOL Fragments

- guarded quantification, no explicit variables, function-free fragments
- mostly contained in $\mathbf{FO^2}$ **or** $\mathcal{C}^2$

KBs are theories with two types of formulas:

- **ABox**: data, facts $B(c), B(d), r(c,d), r_2(d,e)\ldots$
- TBox: axioms, universal formulas

$$
\begin{array}{rr}
A \sqcap B \sqsubseteq C & \forall x\, A(x) \wedge B(x) \rightarrow C(x) \\
A \sqsubseteq \exists r.B & \forall x\, A(x) \rightarrow \exists y\, r(x,y) \wedge B(y) \\
C \sqsubseteq \forall r.\{a\} & \forall x,y\, C(x) \wedge r(x,y) \rightarrow y = a \\
r \sqsubseteq s & \forall x,y\, r(x,y) \rightarrow s(x,y)
\end{array}
$$

DLs: a toolbox of Computational Logics

DLs are a toolbox:

- different constructors combine into many DLs
- decidable fragments of standard FOL
- choice of the right logic
- taking into account the computational cost

			DL-Lite	$\mathcal{EL}$	$\mathcal{ALC}$	$\mathcal{SHOIQ}$
AfricanCountry	$\sqsubseteq$	Country	✓	✓	✓	✓
Country	$\sqsubseteq$	$\forall$hasCapital.City			✓	✓
Country	$\sqsubseteq$	$\neg$City	✓	✓	✓	✓
City $\sqcap$ $\exists$locIn.AfrCntr	$\sqsubseteq$	AfricanCity		✓	✓	✓
SouthAfrica	$\sqsubseteq$	$\exists$*locatedIn*.{Africa}				
hasCapital$^-$	$\sqsubseteq$	locatedIn	✓			✓
trans(locatedIn)						✓

DLs, OWA and the Real World

What are DLs good for?

What have we achieved in four decades?

- A rich family of logics to describe and reason about graph shaped data
- Good understanding of expressiveness vs. complexity trade-off
- Boundaries of decidability
- Fragments, safe combinations
- A whole arsenal of algorithmic techniques
- Efficient reasoners
- ...

The classic DL Setting

- Data as a labeled graph (ABox)
- Background knowledge in an ontology or TBox

The classic DL Setting

- Data as a labeled graph (ABox)
- Background knowledge in an ontology or TBox

Models as in classical predicate logic

- any structure that satisfies the theory
- for us: any extension of the ABox that satisfies the TBox

DL KB Examples

$$\mathit{VeggiePizza}(\mathit{margherita}) \qquad \mathit{hasTopping}(\mathit{margherita}, \mathit{mozzarella})$$

$$\mathit{VeggiePizza} \sqsubseteq \mathit{Pizza}$$

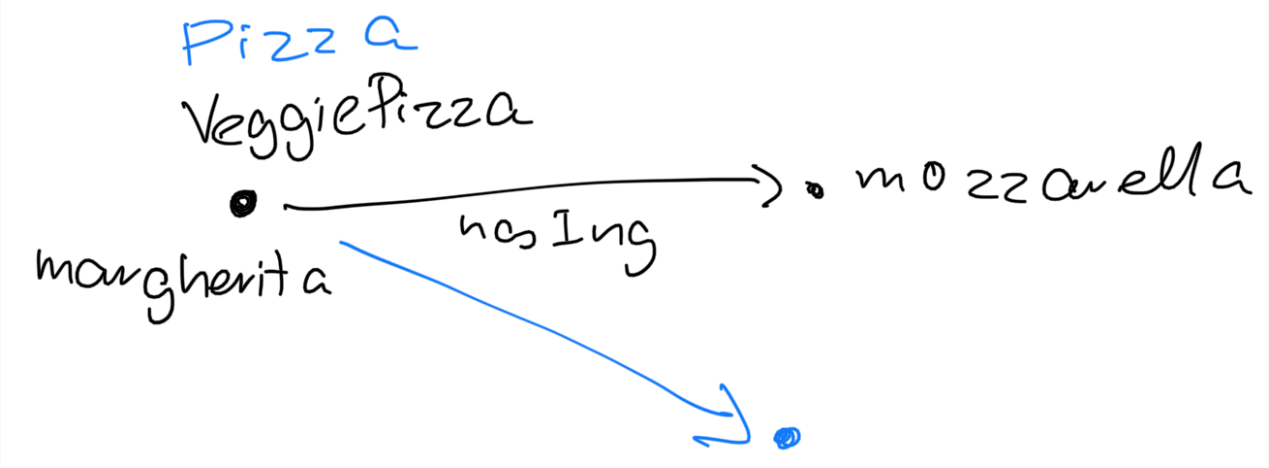
$$\mathit{Pizza} \sqsubseteq \geq_2 \mathit{hasTopping}.\top$$

DL KB Examples

$VeggiePizza(margherita)$ $hasTopping(margherita, mozzarella)$

$$VeggiePizza \sqsubseteq Pizza$$

$$Pizza \sqsubseteq \geq_2 hasTopping.\top$$



Models and DL Reasoning

Models : any extension of the ABox that satisfies the TBox

Models and DL Reasoning

Models : any extension of the ABox that satisfies the TBox

Typically, reason over all models, or similarly, model existence

- Logical entailment: subsumption, instance checking, query answering
- consistency, ...

In practice, (too) many models!

- costly reasoning
- some are unintended!
- sometimes misused or misunderstood

DL Reasoning Examples

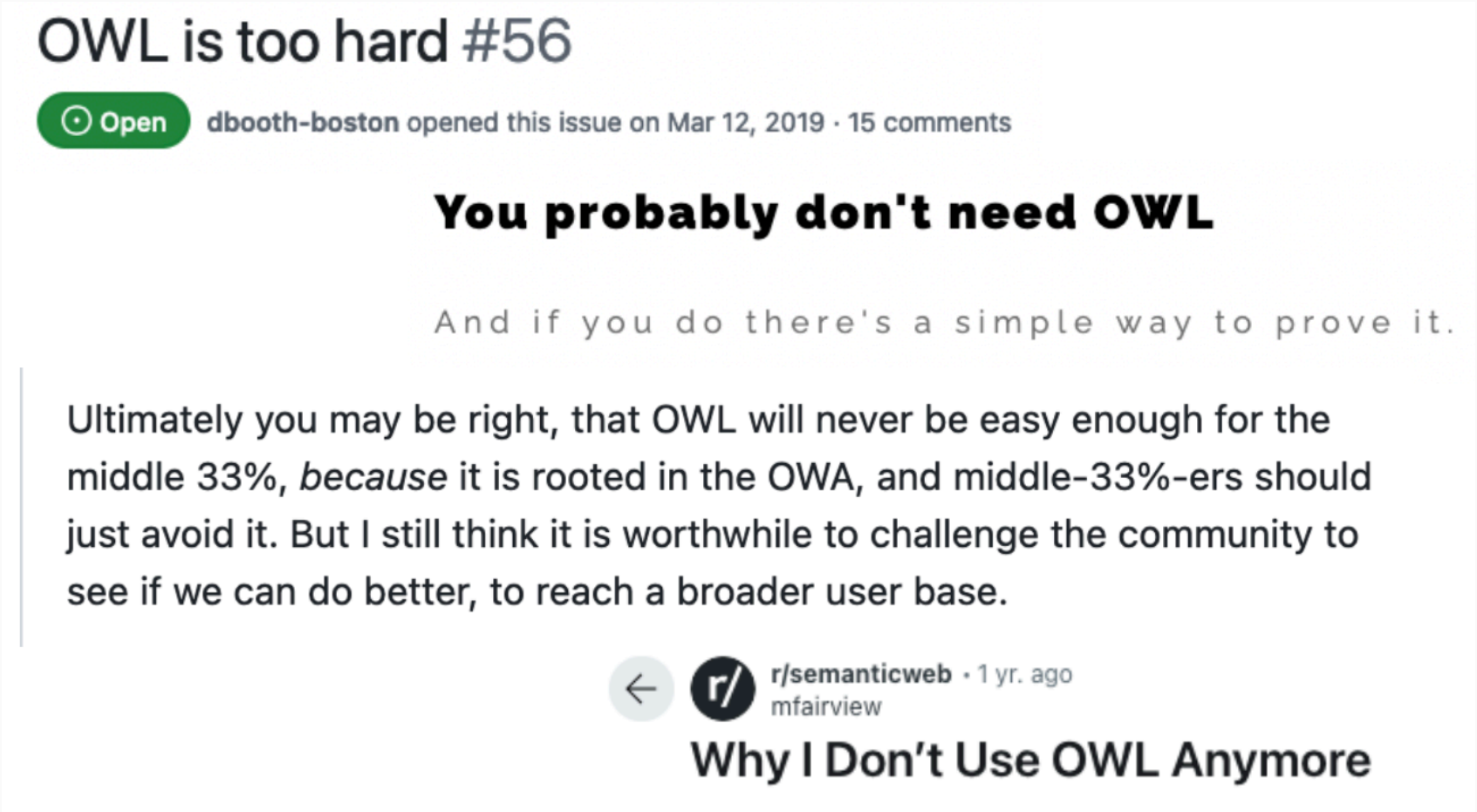
$$Pizza(margherita)$$

$$hasTopping(margherita, mozzarella) \qquad VeggieTopping(mozzarella)$$

$$hasTopping(margherita, tomato) \qquad VeggieTopping(tomato)$$

$$Pizza \sqcap \forall hasTopping.VeggieTopping \sqsubseteq VeggiePizza$$

Wrong expectations



OWL is too hard #56

Open dbooth-boston opened this issue on Mar 12, 2019 · 15 comments

You probably don't need OWL

And if you do there's a simple way to prove it.

> Ultimately you may be right, that OWL will never be easy enough for the middle 33%, *because* it is rooted in the OWA, and middle-33%-ers should just avoid it. But I still think it is worthwhile to challenge the community to see if we can do better, to reach a broader user base.

r/semanticweb · 1 yr. ago
mfairview

Why I Don't Use OWL Anymore

Wrong expectations

```
:Library rdfs:subClassOf [
      a owl:Restriction ;
      owl:allValuesFrom :Book ;
      owl:onProperty :holds
      ] ;
```

This is a ridiculously convoluted way to say that a :Library instance has a :holds property with zero or more :Book values.

Let me start by saying that rdfs:domain and rdfs:range are not constraints and don't mean what you imply. They are merely producing inferences. Having said this, many people have in the past used them to "mean" constraints simply because there was no other modeling language. For background, see

Attempts to use DLs as constraint languages to describe data!

Constraints for Graph Data

SHACL `https://www.w3.org/TR/shacl/`

- SHApe Constraint Language
- W3C standard since 2017
- For RDF data

Constraints for Graph Data

SHACL `https://www.w3.org/TR/shacl/`

- SHApe Constraint Language
- W3C standard since 2017
- For RDF data

DL syntax with constraint semantics

Introduction to SHACL

Basic SHACL Syntax

Vocabulary:

- set of property names P (aka roles)
- set of class names C (aka concepts)
- set of node names N (aka individuals)

We define shape expressions φ:

$$\varphi ::= A \mid \top \mid \{a\} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \geq_n E.\varphi \mid E = E$$

where $A \in \mathbf{C}$, $a \in \mathbf{I}$, $n \geq 0$, and E is a path expression

$$E ::= p \mid p^- \mid E \cup E \mid E \circ E \mid E^*$$

with $p \in \mathbf{P}$.

As usual, we can express

$$\varphi_1 \vee \varphi_2 \qquad \exists E.\varphi \qquad \forall E.\varphi \qquad \leq_{n-1} E.\varphi$$

Example Shape Expressions

$$\texttt{Pizza} \wedge \exists \texttt{hasTopping}.\{\texttt{mozzarella}\} \wedge \forall \texttt{hasTopping}.\texttt{Vegetarian}$$

$$\texttt{Pizza} \wedge \geq_2 \texttt{hasTopping}.(\{\texttt{aubergine}\} \vee \{\texttt{onion}\} \vee \{\texttt{artichoke}\})$$

SHACL Validation - basics

Input:

1. A data graph G
2. A shape expression φ
3. A set of target nodes $a_1, \ldots a_n$

SHACL Validation - basics

Input:

1. A data graph G
2. A shape expression φ
3. A set of target nodes $a_1, \ldots a_n$

The target nodes may be e.g., nodes in a class or complex query

SHACL Validation - basics

Input:

1. A **data graph** G
2. A **shape expression** φ
3. A set of **target nodes** $a_1, \ldots a_n$

The target nodes may be e.g., nodes in a class or complex query

Question: **Does G validate φ at each a_i?**

- View the **data graph as an interpretation**
  - Semantics of connectives in shape expressions as usual
- **Is a_i in the extension of φ?**, i.e., **is G a model of $a_i : \varphi$?**

SHACL Validation - basics

Input:

1. A **data graph** G
2. A **shape expression** φ
3. A set of **target nodes** $a_1, \ldots a_n$

The target nodes may be e.g., nodes in a class or complex query

Question: **Does G validate φ at each a_i?**

- View the **data graph as an interpretation**
  - Semantics of connectives in shape expressions as usual
- **Is a_i in the extension of φ?**, i.e., **is G a model of $a_i : \varphi$?**

Feasible in PTime

- **Model checking** is typically easy!

SHACL Validation - basics

Input:

1. A **data graph** G
2. A **shape expression** φ
3. A set of **target nodes** $a_1, \ldots a_n$

The target nodes may be e.g., nodes in a class or complex query

Question: **Does G validate φ at each a_i?**

- View the **data graph as an interpretation**
  - Semantics of connectives in shape expressions as usual
- **Is a_i in the extension of φ?**, i.e., **is G a model of $a_i : \varphi$?**

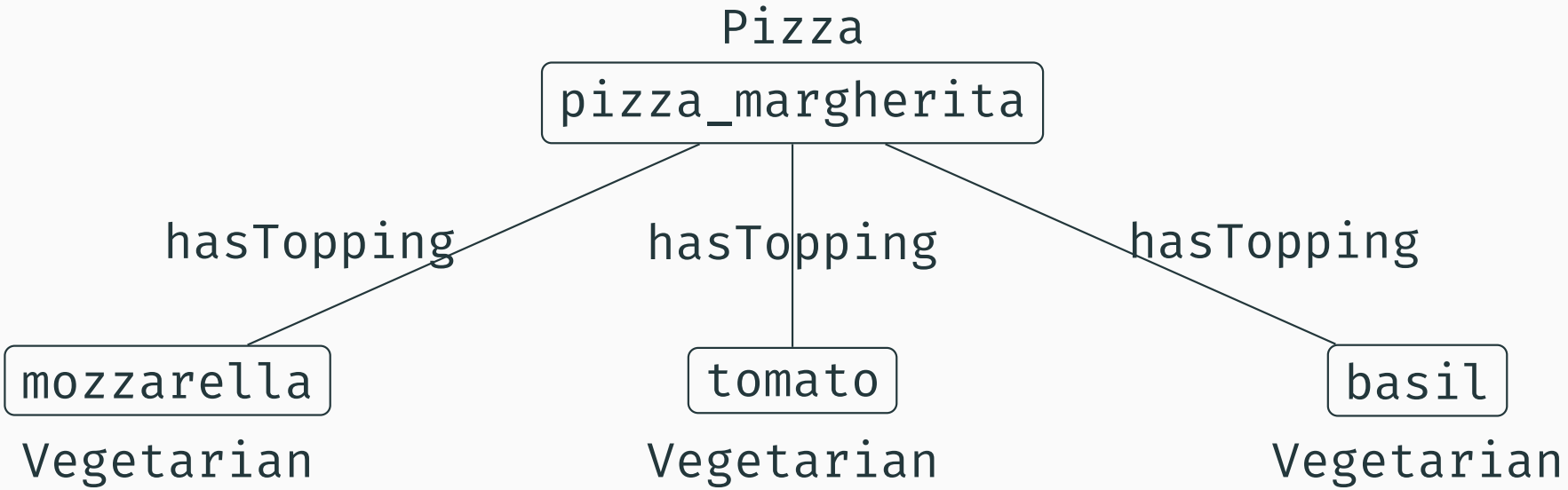
Feasible in PTime

- **Model checking** is typically easy!

Satisfiability and containment of SHACL constraints are undecidable!

Validation on an Example

Given the graph:



This shape expression validates for the target node `margherita`:

$$\texttt{Pizza} \wedge \forall \texttt{hasTopping}.\texttt{Vegetarian}$$

Shape names in SHACL

The SHACL vocabulary has more:

- set of property names P (aka roles)
- set of node names N (aka individuals)
- set of class names C (aka concepts)
- set of shape names S (similar to concepts)

Shape names in SHACL

The SHACL vocabulary has more:

- set of property names P (aka roles)
- set of node names N (aka individuals)
- set of class names C (aka concepts)
- set of shape names S (similar to concepts)

In SHACL constraints, we assign shape expressions to shape names

$$s \leftarrow \varphi$$

Shape names in SHACL

The SHACL vocabulary has more:

- set of property names P (aka roles)
- set of node names N (aka individuals)
- set of class names C (aka concepts)
- set of shape names S (similar to concepts)

In SHACL constraints, we assign shape expressions to shape names

$$s \leftarrow \varphi$$

Targets can then be given as sets of shape atoms

$$s_1(a_1), \ldots, s_n(a_n)$$

A shapes graph contains constraints $\mathcal{C}$ and targets $\mathcal{T}$

Recursion in SHACL

The SHACL specification allows **shape names** in shape expressions:

$$\varphi ::= s \mid A \mid \top \mid \{a\} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \geq_n E.\varphi \mid E = E$$

Recursion in SHACL

The SHACL specification allows **shape names** in shape expressions:

$$\varphi ::= s \mid A \mid \top \mid \{a\} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \geq_n E.\varphi \mid E = E$$

$$\begin{aligned} \mathit{Pizza} &\leftarrow \geq_2 \texttt{hasTopping}.\top, \\ \mathit{VeggiePizza} &\leftarrow \mathit{Pizza} \wedge \forall \texttt{hasTopping}.\mathit{VeggieTopping}, \\ \mathit{VeggieTopping} &\leftarrow \{\texttt{mozzarella}\} \vee \{\texttt{tomato}\} \vee \{\texttt{basil}\} \vee \{\texttt{artichoke}\} \end{aligned}$$

$$\mathit{Person} \leftarrow \exists \texttt{hasParent}.\mathit{Person}$$

Validation of Recursive SHACL

We assume each s occurs in only one constraint head:

$$s \leftarrow \varphi_1 \vee \cdots \vee \varphi_n$$

We now need a **shape assignment** $\mathcal{I}$ that assigns a set of nodes $s^{\mathcal{I}}$ to each shape name $\mathcal{I}$.

Validation of Recursive SHACL

We assume each s occurs in only one constraint head:

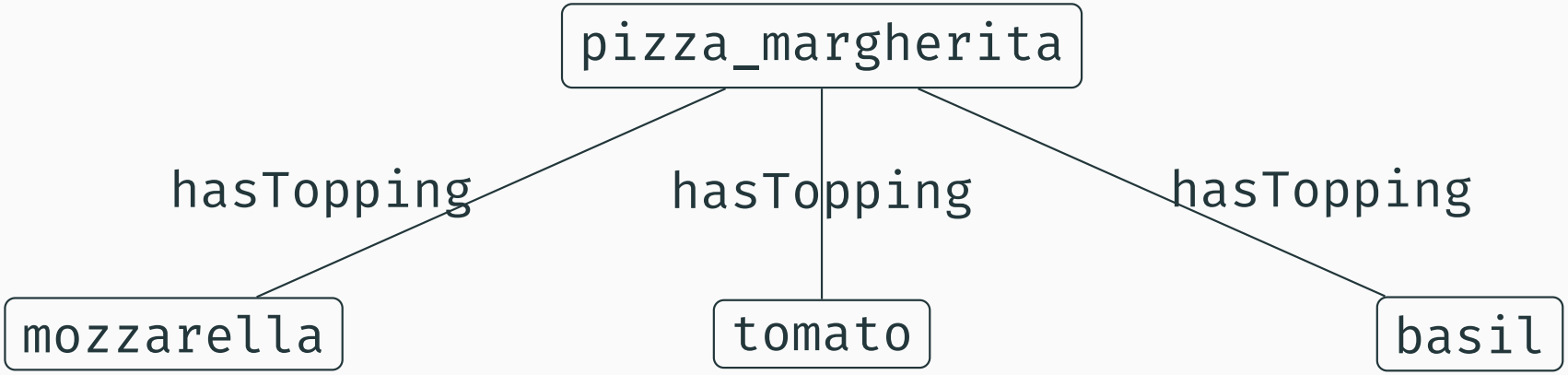
$$s \leftarrow \varphi_1 \vee \cdots \vee \varphi_n$$

We now need a **shape assignment** $\mathcal{I}$ that assigns a set of nodes $s^{\mathcal{I}}$ to each shape name $\mathcal{I}$.

Validation of $(\mathcal{C}, \mathcal{T})$ consists on finding an assignment $\mathcal{I}$ such that:

- $s^{\mathcal{I}} = \varphi^{\mathcal{I}}$ for each constraint $s \leftarrow \varphi$ in $\mathcal{C}$
- $a \in s^{\mathcal{I}}$ for every target $s(a)$ in $\mathcal{T}$

Back to our Example



$$\mathit{Pizza} \leftarrow \geq_2 \texttt{hasTopping}.\top,$$

$$\mathit{VeggiePizza} \leftarrow \mathit{Pizza} \wedge \forall \texttt{hasTopping}.\mathit{VeggieTopping},$$

$$\mathit{VeggieTopping} \leftarrow \{\texttt{mozzarella}\} \vee \{\texttt{tomato}\} \vee \{\texttt{basil}\} \vee \{\texttt{artichoke}\}$$

Semantics of recursive SHACL

Validation of $(\mathcal{C}, \mathcal{T})$ consists on finding an assignment $\mathcal{I}$ such that:

- $s^{\mathcal{I}} = \varphi^{\mathcal{I}}$ for each constraint $s \leftarrow \varphi$ in $\mathcal{C}$
- $a \in s^{\mathcal{I}}$ for every target $s(a)$ in $\mathcal{T}$

Semantics of recursive SHACL

Validation of $(\mathcal{C}, \mathcal{T})$ consists on finding an assignment $\mathcal{I}$ such that:

- $s^{\mathcal{I}} = \varphi^{\mathcal{I}}$ for each constraint $s \leftarrow \varphi$ in $\mathcal{C}$
- $a \in s^{\mathcal{I}}$ for every target $s(a)$ in $\mathcal{T}$

Satisfiability in DLs where:

- property names **P** and node names **N** are **closed**
- shape names **S** are **open**

Semantics of recursive SHACL

Validation of $(\mathcal{C}, \mathcal{T})$ consists on finding an assignment $\mathcal{I}$ such that:

- $s^{\mathcal{I}} = \varphi^{\mathcal{I}}$ for each constraint $s \leftarrow \varphi$ in $\mathcal{C}$
- $a \in s^{\mathcal{I}}$ for every target $s(a)$ in $\mathcal{T}$

Satisfiability in DLs where:

- property names **P** and node names **N** are **closed**
- shape names **S** are **open**

Constraints as concept definitions $s \equiv \varphi$, targets as concept assertions $s(a)$

Finding an assignment $\mathcal{I}$

- makes satisfiability coNP hard
- are all assignments equally good?

Validation in Recursive SHACL

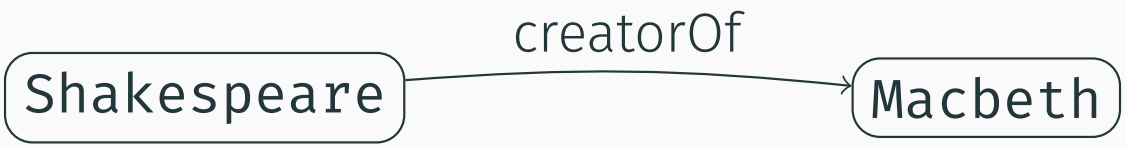
$$\mathit{Director} \leftarrow \exists \texttt{creatorOf}.\mathit{Movie}$$

$$\mathit{Movie} \leftarrow \exists \texttt{creatorOf}^{-}.\mathit{Director}$$

Validation in Recursive SHACL

$$\mathit{Director} \leftarrow \exists \texttt{creatorOf}.\mathit{Movie}$$

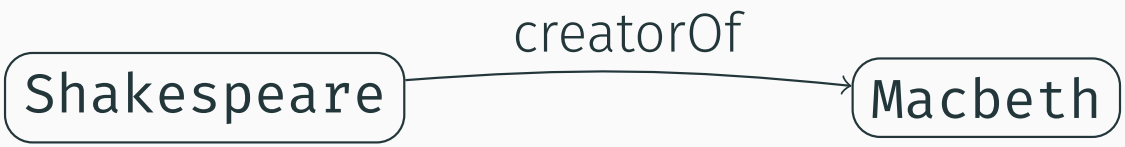
$$\mathit{Movie} \leftarrow \exists \texttt{creatorOf}^{-}.\mathit{Director}$$



Validation in Recursive SHACL

$$\mathit{Director} \leftarrow \exists \texttt{creatorOf}.\mathit{Movie}$$

$$\mathit{Movie} \leftarrow \exists \texttt{creatorOf}^{-}.\mathit{Director}$$



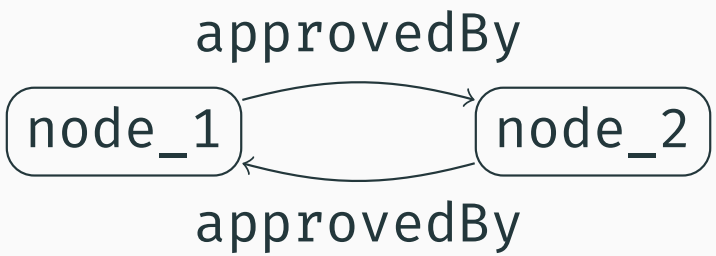
We can validate *Director*(Shakespeare) !?!

Validation in Recursive SHACL

$$\mathit{certifiedNode} \leftarrow \exists \texttt{granted}.\mathit{Certifate} \vee \exists \texttt{approvedBy}.\mathit{certifiedNode}$$

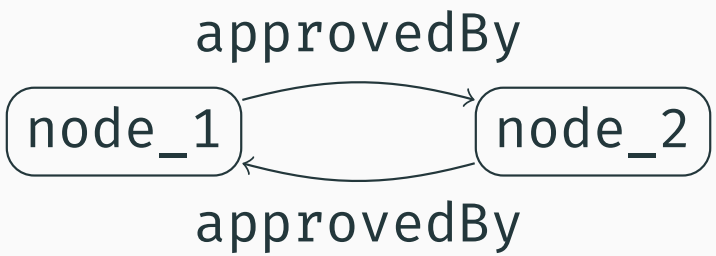
Validation in Recursive SHACL

$$\mathit{certifiedNode} \leftarrow \exists \texttt{granted}.\mathit{Certifate} \vee \exists \texttt{approvedBy}.\mathit{certifiedNode}$$



Validation in Recursive SHACL

$$\mathit{certifiedNode} \leftarrow \exists \texttt{granted}.\mathit{Certifate} \vee \exists \texttt{approvedBy}.\mathit{certifiedNode}$$



We can validate *certifiedNode*(node_1)

Refining the Semantics of Recursive SHACL

- Supported semantics: any shape assignment
- Other semantics choose **well-founded assignments**

Refining the Semantics of Recursive SHACL

- Supported semantics: any shape assignment
- Other semantics choose **well-founded assignments**

DLs in 1990s

descriptive vs. least fixpoint semantics for terminological cycles

Refining the Semantics of Recursive SHACL

- Supported semantics: any shape assignment
- Other semantics choose **well-founded assignments**

DLs in 1990s

descriptive vs. least fixpoint semantics for terminological cycles

LPs in 1990s

Logic Programs with negation

Refining the Semantics of Recursive SHACL

- Supported semantics: any shape assignment
- Other semantics choose **well-founded assignments**

DLs in 1990s

descriptive vs. least fixpoint semantics for terminological cycles

LPs in 1990s

Logic Programs with negation

> 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 Recommendation, §3.4.3

Giving Semantics to Recursive SHACL

- Stable model semantics as in ASP

$$married \leftarrow \neg single \qquad single \leftarrow \neg married$$

Two stable models: $\{married\}, \{single\}$

Giving Semantics to Recursive SHACL

- Stable model semantics as in ASP

$$married \leftarrow \neg single \qquad single \leftarrow \neg married$$

Two stable models: $\{married\}, \{single\}$

  - minimal models of the Gelfond-Lifschitz reduct
  - alternative level-wise definition for SHACL
  - NP-complete validation

Giving Semantics to Recursive SHACL

- Stable model semantics as in ASP

$$married \leftarrow \neg single \qquad single \leftarrow \neg married$$

  Two stable models: $\{married\}, \{single\}$

  - minimal models of the Gelfond-Lifschitz reduct
  - alternative level-wise definition for SHACL
  - NP-complete validation

- Well-founded semantics are a 3-valued

$$married \leftarrow \neg single \qquad single \leftarrow \neg married$$

  Both *married* and *single* undefined

Giving Semantics to Recursive SHACL

- Stable model semantics as in ASP

$$married \leftarrow \neg single \qquad single \leftarrow \neg married$$

  Two stable models: $\{married\}, \{single\}$

  - minimal models of the Gelfond-Lifschitz reduct
  - alternative level-wise definition for SHACL
  - **NP-complete** validation

- Well-founded semantics are a 3-valued

$$married \leftarrow \neg single \qquad single \leftarrow \neg married$$

  Both *married* and *single* undefined

  - approximation of stable models
  - **Polynomial** validation
  - inconsistency tolerant

Stratified Recursive SHACL

Stratified constraints: with only positive recursion cycles

Stratified Recursive SHACL

Stratified constraints: with only positive recursion cycles

$$alive \leftarrow \neg dead \quad certfied \leftarrow certified$$

Stratified Recursive SHACL

Stratified constraints: with only positive recursion cycles

$$alive \leftarrow \neg dead \quad certfied \leftarrow certified$$

stable model = well-founded model = Perfect model

Stratified Recursive SHACL

Stratified constraints: with only positive recursion cycles

$$alive \leftarrow \neg dead \quad certfied \leftarrow certified$$

stable model = well-founded model = Perfect model

- $\{alive\}$ is the only model, *certified* is false
- **P-complete** validation

Stratified Recursive SHACL

Stratified constraints: with only positive recursion cycles

$$alive \leftarrow \neg dead \quad certfied \leftarrow certified$$

stable model = well-founded model = Perfect model

- $\{alive\}$ is the only model, *certified* is false
- **P-complete** validation

In contrast, **supported validation**

- Two models: $\{alive\}$, $\{alive, certified\}$
- **NP-complete** validation

Validating SHACL with Ontologies

SHACL + OWL

- Open-world semantics: DLs
- Close-world semantics: SHACL

What if we want both?

- Validating constraints in the presence of ontologies

Our data contains:

$$\mathit{hasBird}(\mathit{linda}, \mathit{blu}) \qquad \mathit{hasDog}(\mathit{john}, \mathit{ace})$$

SHACL + OWL

- Open-world semantics: DLs
- Close-world semantics: SHACL

What if we want both?

- Validating constraints in the presence of ontologies

Our data contains:

$$\mathit{hasBird}(\mathit{linda}, \mathit{blu}) \qquad \mathit{hasDog}(\mathit{john}, \mathit{ace})$$

We want to validate *PetOwnerShape* for John and Linda

$$\mathit{PetOwnerShape} \leftarrow \exists \mathit{hasPet}$$

SHACL + OWL

- Open-world semantics: DLs
- Close-world semantics: SHACL

What if we want both?

- Validating constraints in the presence of ontologies

Our data contains:

$$hasBird(linda, blu) \qquad hasDog(john, ace)$$

We want to validate *PetOwnerShape* for John and Linda

$$PetOwnerShape \leftarrow \exists hasPet$$

But we also have a TBox:

$$hasBird \sqsubseteq hasPet \qquad hasDog \sqsubseteq hasPet$$

Certain Answers?

- SHACL has negation
- Certain answer semantics too weak!

Certain Answers?

- SHACL has **negation**
- **Certain answer semantics** too weak!

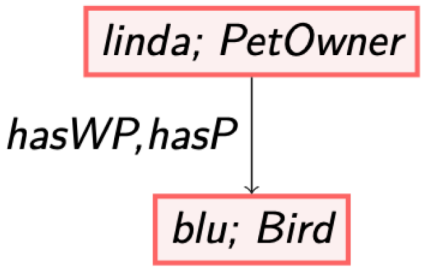
$$\mathit{hasPet}(\mathit{linda}, \mathit{blu}) \qquad \mathit{Bird}(\mathit{blu})$$

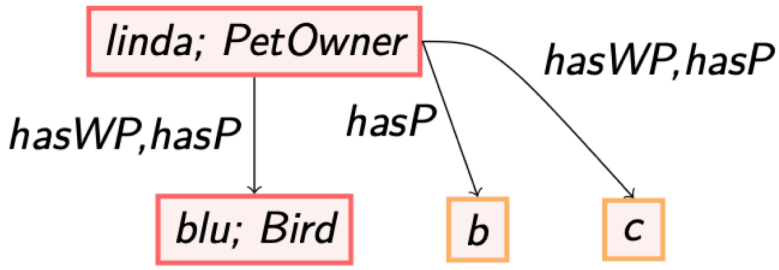
$$\mathit{otherPetOwner} \leftarrow \exists \mathit{hasPet}.\neg \mathit{Dog}$$

Universal models?

For **Horn DLs** we usually rely on the existence of a **universal model** that can be **homomorphically embedded** into **all modes**

- aka chase in databases





$$\mathcal{T} = \{PetOwner \sqsubseteq \exists hasPet, \quad hasWingedPet \sqsubseteq hasPet,$$
$$PetOwner \sqsubseteq \exists hasWingedPet\}$$

It matters which facts we add!

But SHACL expressions are **not preserved under homomorphism!**

Core universal model

Core universal model

Intuitively:

- universal
- does not contain itself as a substructure

Core universal model

Intuitively:

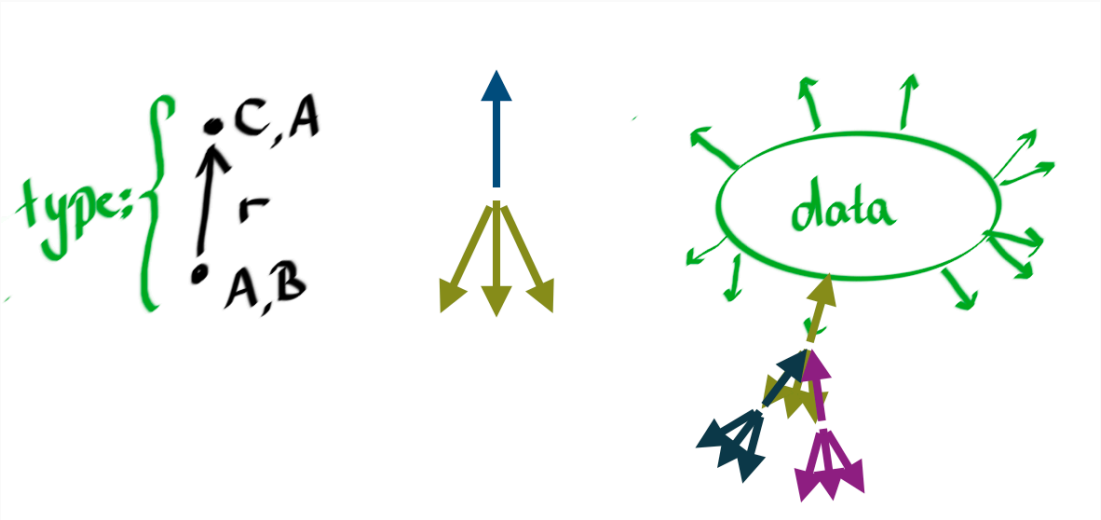
- universal
- does not contain itself as a substructure

Typical constructions check for ”coreness” after each chase step

Austere Model Construction

For Horn DLs like DL-Lite and $\mathcal{ELHI}$:

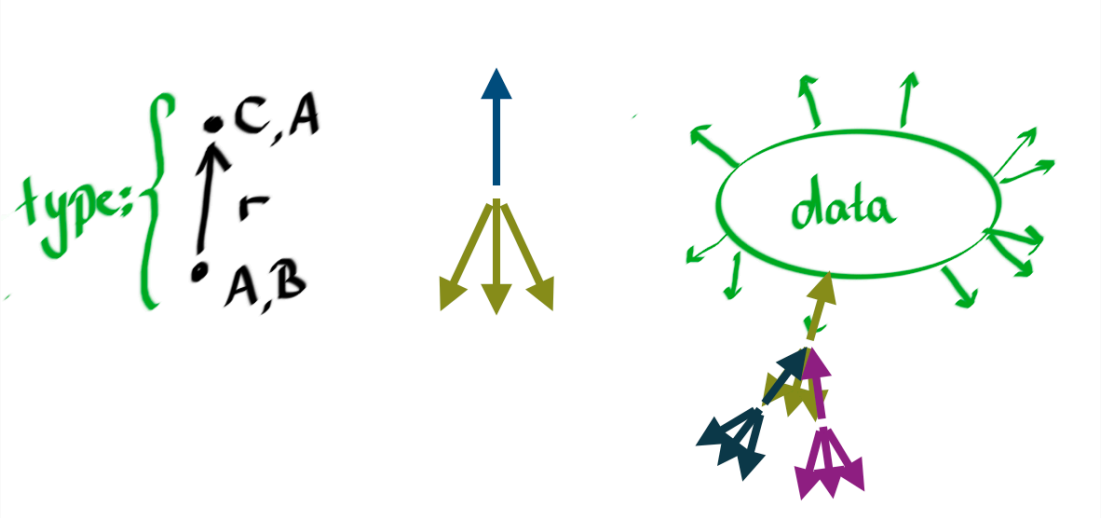
- compute a minimal local successor configuration for each 2-type
- build the model using these successors



Austere Model Construction

For Horn DLs like DL-Lite and $\mathcal{ELHI}$:

- compute a minimal local successor configuration for each 2-type
- build the model using these successors



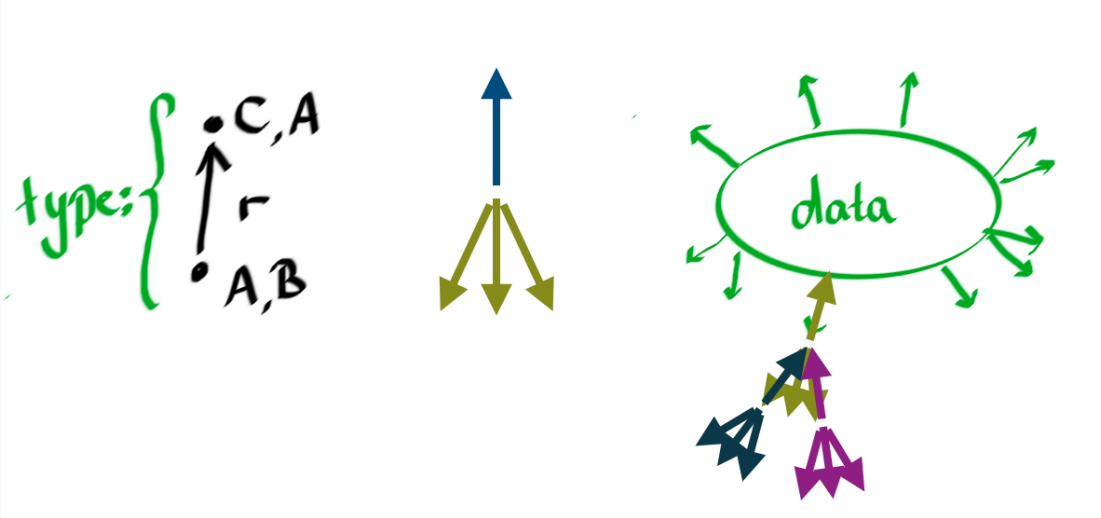
Theorem

The austere model is a (possibly infinite) universal core

Austere Model Construction

For Horn DLs like DL-Lite and $\mathcal{ELHI}$:

- compute a minimal local successor configuration for each 2-type
- build the model using these successors



Theorem

The austere model is a (possibly infinite) universal core

Avoid expensive minimality check after each chase step!

Validation via Rewriting

Given a TBox $\mathcal{T}$ and a set of constraints $\mathcal{C}$, obtain $\mathcal{C}$ such that, for every data graph G and any target M:

$$G \text{ validates } (C_{\mathcal{T}}, M)$$
$$\text{iff}$$
$$\text{the austere universal model of } (\mathcal{T}, G) \text{ validates } (C, M)$$

- Calculus to **propagate validated shapes** using the successor configuration
- After saturation, generate new constraints
- Works even with **stratified negation**

Theorem

Validation of stratified SHACL constraints in the presence of Horn-$\mathcal{ALCHI}$ ontologies is ExpTime complete in combined complexity and PTime complete in data complexity

Validation via Rewriting

Given a TBox $\mathcal{T}$ and a set of constraints $\mathcal{C}$, obtain $\mathcal{C}$ such that, for every data graph G and any target M:

$$G \text{ validates } (C_{\mathcal{T}}, M)$$
$$\text{iff}$$
$$\text{the austere universal model of } (\mathcal{T}, G) \text{ validates } (C, M)$$

- Calculus to **propagate validated shapes** using the successor configuration
- After saturation, generate new constraints
- Works even with **stratified negation**

Theorem

Validation of stratified SHACL constraints in the presence of Horn-$\mathcal{ALCHI}$ ontologies is ExpTime complete in combined complexity and PTime complete in data complexity

ExpTime hardness holds even for very simple ontologies!

Validation via Rewriting: an example

From

$$birdShape \leftarrow Bird \qquad birdOwnerShape \leftarrow \exists hasPet.birdShape$$

$$PetOwner \sqsubseteq \exists hasPet \quad hasWingedPet \sqsubseteq hasPet \quad \exists hasWingetPet^{-} \sqsubseteq Bird$$

Validation via Rewriting: an example

From

$$birdShape \leftarrow Bird \qquad birdOwnerShape \leftarrow \exists hasPet.birdShape$$

$$PetOwner \sqsubseteq \exists hasPet \quad hasWingedPet \sqsubseteq hasPet \quad \exists hasWingetPet^{-} \sqsubseteq Bird$$

the rewriting obtains new constraints like:

$$birdShape \leftarrow \exists hasWingedPet^{-} \qquad birdOwnerShape \leftarrow \exists hasWingedPet$$

What's next?

Many fun questions open:

- Full negation
- Make this work in practice
- Non-Horn DLs
- SHACL optimization
- validation under updates
- ...

Combining Open and Closed-world reasoning still a very exciting area!