Comparative expressiveness of ShEx and SHACL (Early working draft)

Iovka Boneva

Comparative expressiveness of ShEx and SHACL (Early working draft)

Iovka Boneva

March 14, 2016

Contributions

- We propose a simple formal language for graph shapes that subsumes both ShEx and SHACL. The semantics of the language is based on the semantics of Datalog, and also equivalently defined in terms of Monadic Second Order Logic with Presburger constraints.
- We propose a **formal semantics of SHACL** as a translation to this language. Thanks to this translation, we show that SHACL can be extended with well-defined stratified recursion.
- We show how ShEx can be translated to this language.
- We explore the necessary restrictions on ShEx so that it can be translated to SHACL, and also the possible modifications of SHACL so that it can capture a bigger fragment of ShEx.

1 Preliminaries

Glossary

n, k	natural constants
u	variable over naturals
p, q	property
x, y, z, X, Y, Z	MSO variables
t	term in Presburger logic
e	element of a graph

RDF graph We define a logical structure that corresponds to RDF graphs. Let $Dom = Iri \cup Blank \cup Lit$ be a three-sorted set, where Iri, $Blank$ and Lit are three countably infinite sets and are mutually disjoint. Consider the purely relational[1] signature $\sigma_{\text{RDF}} = (\{Triple\} \cup \{F \mid F \in \mathcal{F}\})$, where $Triple$ is a ternary relation, and $\mathcal{F}$ is a family of unary relations.

An *RDF graph*, or simply a *graph* $G = (Triple^G \cup \bigcup_{F \in \mathcal{F}} F^G)$ is a finite σ_{RDF}-structure over Dom. We denote by $dom(G)$ the active domain of G. The triples in $Triple$ define the graph structure and are such that the first element of a triple cannot be in Lit, and the second element of a triple can only be

[1] All symbols in the signature are first order relation symbols.

element of *Iri*. The unary relations in $\mathcal{F}$ are used for describing properties of the elements of the nodes of the graph, such as "a node is a literal of type integer and its value is smaller than 5", "an *Iri* starts with `http://foaf.org/`", etc. We suppose that for every such relation F, $\mathcal{F}$ also contains its complement relation, denoted $\bar{F}$. That is, in every graph G and for every relation $F \in \mathcal{F}$, we have $\bar{F}^G = dom(G) \setminus F^G$. The family $\mathcal{F}$ is potentially infinite, however in every graph we are going to use only a finite subset of such relation symbols.

Given a graph G and a node n in $dom(G)$, the *neighbourhood of n in G* is the set of facts $neigh_G(n) = \{\, Triple^G(n, p, n') \mid Triple^G(n, p, n') \in G\}$.

Typed RDF graph Assume a finite set $\mathcal{S}$ of shape names, and let $\sigma_{\mathrm{TRDF}} = \sigma_{\mathrm{RDF}} \cup \{S\}_{S \in \mathcal{S}}$ be the purely relational signature obtained by adding to σ_{RDF} the unary relational symbols from $\mathcal{S}$. A *typed RDF graph*, or simply a *typed graph* is a finite σ_{TRDF}-structure. Given a graph G, a *typing of G* is as an extension of G by an interpretation of the relations in $\mathcal{S}$. That is, if $G = (Triple^G \cup \bigcup_{F \in \mathcal{F}} F^G)$, then $G_t = (Triple^G \cup \bigcup_{F \in \mathcal{F}} F^G \cup \bigcup_{S \in \mathcal{S}} S^G)$ for S^G being a unary relation on $dom(G)$, for all $S \in \mathcal{S}$.

Notation: for a typed graph G_t as above, $e \in dom(G)$ and for a unary relation symbol $F \in \mathcal{F}$, we sometimes write $e \in F^G$ as synonym of $F^G(e)$, and similarly $e \in S^G$ for a shape name S.

How a typed RDF graph relates to ShEx and SHACL ? Both ShEx and SHACL aim at answering the following problem: Given

- a set of shape constraints, say $\{S_1, \ldots, S_n\}$ that possibly refer to each other,
- a RDF graph G,
- a set of node-shape associations $\{(e_1, S'_1), \ldots (e_m, S'_m)\}$ where the $e_1, /dots, e_m$ are nodes of G and the S'_j are among $\{S_1, \ldots, S_n\}$.

answer the question (yes or no answer) whether e_j satisfies the constraint S'_j for all $j \in 1..m$.

In the above question, the word "satisfies" is not precisely stated. By defining semantics for ShEx and SHACL, we means giving a precise mathematical meaning of this word in this context. We propose the following formalization.

> e_j satisfies S'_j for all $j \in 1..m$ means that there exists a typed graph G_t that is a typing of G, and such that G_t is a model for the shape constraints $\{S_1, \ldots, S_n\}$ and such that $S'_j(e_j) \in G_t$ for all $j \in 1..m$.

We give a precise mathematical meaning of G_t *is a model for the shape constraints* $\{S_1, \ldots, S_n\}$. TODO: refer to the sections and explain how this claim here relates to the sequel.

2 The shapes-constraint language

We define a simple language for graph shapes.

2.1 Presburger logic

The language uses numerical constraints on the neighbourhood of nodes in a graph. These constraint will be expressed using sentences of Presburger logic that we introduce here.

Let $\mathcal{U}$ be a countable set of variables over natural numbers. A quantifier-free Presburger formulas α is defined by the following syntax:

$$\begin{array}{rcll} \alpha & ::= & t \leq t' \mid \mathit{Div}_k(t) \mid \alpha \wedge \alpha \mid \neg\alpha & k \in \mathbb{N} \\ t & ::= & n \mid u \mid t + t & (u \in \mathcal{U}, n \in \mathbb{N}) \end{array}$$

where $\leq$ is the natural ordering over $\mathbb{N}$ and where for every natural number k, the unary predicate $\mathit{Div}_k(n)$ holds if n is divisible by k. Given a Presburger formula α and a valuation μ that maps the free variables in α, we say that μ satisfies α, written $\mu \models \alpha$, if the structure of natural numbers with addition and Div_k ($k \in \mathbb{N}$) is a model of α under the valuation μ.

For example, consider the formula $\alpha = (u_1 + 2 \leq 4) \wedge \mathit{Div}_3(u_2 + 1)$. Then the valuation $\mu = (u_1 \mapsto 1, u_2 \mapsto 26)$ satisfies α.

Remark that $\alpha_1 \vee \alpha_2$ can be defined in the usual way using conjunction and negation.

2.2 Syntax of the shapes-constraint language

Assume the σ_{TRDF} signature with $\mathcal{S}$ a finite set of *shape names*, and $\mathcal{F}$ is a family of unary predicates. A *triple constraint* is defined by the following syntax:

$$T := P :: F \mid P :: @S \mid P :: @\bar{S} \mid P :: \_$$

where P is a finite or co-finite subset of *Iri*, $F \in \mathcal{F}$ and $S \in \mathcal{S}$. Intuitively, a triple constraint is a filter on the triples in a graph. For all four kinds of triple constraints, a triple satisfies a constraint $P :: X$ if its predicate is in the set P (whatever X is). Moreover, $P :: F$ is satisfied by all triples whose object belongs to F. $P :: @S$ is satisfied by all triples whose object satisfies the shape S. $P :: @\bar{S}$ is satisfied by all triples whose object does not satisfy the shape S. Finally, $P :: \_$ does not impose a constraint on the object of the triples. We give in the sequel formal definition of what it means for a triple to satisfy or not satisfy S.

Remark that when P is a singleton set say $P = \{p\}$, we write in triple constraints e.g. $p :: F$ instead of $\{p\} :: F$.

Given a finite set $\mathcal{TC}$ of triple constraints, consider the set of natural variables $\mathcal{U} = \{\#T \mid T \in \mathcal{TC}\}$. A *shape definition* is of the form

$$S \leftarrow \alpha$$

where $S \in \mathcal{S}$ and α is a Presburger formula with free variables from $\mathcal{U}$. A *shapes-constraint* is a sequence[2] of shape definitions such that every shape name that appears in the right hand side of some shape definition is defined exactly once (i.e. appears exactly once in the left-hand side of a shape definition).

[2]TODO: needed ? We use a sequence instead of a set in order to be able to identify the definitions by their rank in the sequence. The ordering in the sequence is not otherwise important.

Example 1. Suppose that name, firstName, lastName, friend are elements of *Iri*. Let $\mathcal{S} = \{\text{Person}\}$ contains a unique shape name. Consider the following triple constraints T_n, T_{fn}, T_{ln}, T_{fr}, T_o. Note that when a set of *Iri* is a singleton, we omit writing the curly braces $\{\}$. The F_{str} relation contains all literals that are strings, and the F_{notstr} relation contains all nodes that are not literal strings (literals, IRI or blank nodes), i.e. F_{notstr} is the complement of F_{str} in every graph.

- $T_n = \text{name} :: F_{\text{str}}$ (the predicate is name and the object is a string);
- $T_{fn} = \text{ firstName} :: F_{\text{str}}$;
- $T_{ln} = \text{ lastName} :: F_{\text{str}}$;
- $T_{fr} = \text{ friend} :: @\text{Person}$ (the predicate is friend and the object satisfies the shape Person) ;
- $T_o = P :: \_$ where $P = \{\text{name, firstName, lastName, friend}\}$;
- $T_{cl} = P_{cl} :: \_$ where $P_{cl} = \mathit{Iri} \setminus P$ (i.e. predicate is not one of name, firstName, lastName.

The following shape Person specifies that a person has either one name, or one firstName and one lastName, and at least one friend, and has no other properties than those specified (i.e. it is a closed shape).

$$\begin{aligned}\text{Person} \leftarrow \quad & ((\#T_n = 1 \wedge \#T_{fn} = 0 \wedge \#T_{ln} = 0) \vee (\#T_n = 0 \wedge \#T_{fn} = 1 \wedge \#T_{ln} = 1)) \wedge \\ & \#T_{fr} \geq 1 \wedge \\ & \#T_o = \#T_n + \#T_{fn} + \#T_{ln} + \#T_{fr} \wedge \\ & \#T_{cl} = 0\end{aligned}$$

The first line expresses the constraint either one name, or one first name and one last name. The second line expresses that a person has at least one friend. The third line expresses that the predicates name, firstName, lastName and friend should not appear differently than with the above stated constraints. Finally, the last line expresses the fact that no other predicates than those mentioned can appear. □

Definition 1 (Dependency graph)**.** Given a shapes-constraint $H = S_1 \leftarrow \alpha_1, \ldots, S_n \leftarrow \alpha_n$, its *dependency graph* Dep_H is a graph whose set of nodes is $\{S_1, \ldots, S_n\}$, and that has two types of edge relations E^+ and E^- (for positive and negative). There is an edge $E^+(S_i, S_j)$ iff there exists a triple constraint of the form $P :: @S_j$ in α_i, and there is an edge $E^-(S_i, S_j)$ iff there exists a triple constraint of the from $P :: @\bar{S}_j$ in α_i, for P a set of properties. If $E^+(S_i, S_j)$, then we say that S_i depends positively on S_j, and if $E^-(S_i, S_j)$, then we say that S_i depends negatively on S_j. □

Definition 2 (Stratified shapes-constraint)**.** A shapes-constraint $H = S_1 \leftarrow \alpha_1, \ldots, S_n \leftarrow \alpha_n$ is called *stratified* if there exist a natural number k and a partitioning $\mathit{stratum} : \{S_1, \ldots, S_n\} \rightarrow 1..k$ of the shapes that appear in H s.t. for all shape names S_i, S_j:

- if $E^+(S_i, S_j)$, then $\mathit{stratum}(S_i) \geq \mathit{stratum}(S_j)$;

- if $E^-(S_i, S_j)$, then $\mathit{stratum}(S_i) > \mathit{stratum}(S_j)$.

□

From now on, we consider only stratified shapes-constraints. Remark that a shape constraint w/o recursion (i.e. without loops in the dependency graph) is stratified, and also a shapes-constraint w/o negative dependencies.

2.3 Semantics of the shapes-constraint language

The semantics of shapes-constraints captures how one can associate shape names with the nodes of a graph so that every node satisfies the shape names associated with it. This is captured by the notion of typing of a graph.

We present both an operational and a model-theoretical semantics of shapes-constraints. For the former, a shapes-constraint is seen as a logical program. For the latter, it is seen as a sentence in Monadic Second-Order logic (MSO).

2.3.1 Model

We start by defining a notation. Let a typed graph $G = (\mathit{Triple}^G \cup \bigcup_{F\in\mathcal{F}} F^G \cup \bigcup_{S\in\mathcal{S}} S^G)$ and $P \subseteq \mathit{Iri}$ a finite or co-finite set of IRI. Given a set $D \subseteq \mathit{dom}(G)$, for all element $e \in \mathit{dom}(G)$ and all $i \in 1..n$, we denote $\mathit{Triple}^G_{|e,P,D}$ the set of triples $\{\mathit{Triple}^G(e, p, e') \mid p \in P \text{ and } e \in D\}$. That is, $\mathit{Triple}^G_{|e,P,D}$ is the set of neighbourhood triples of e whose predicate is in P and whose object is in D. This will be used with D being either S^G or F^G, for some $S \in \mathcal{S}$ and $F \in \mathcal{F}$.

Definition 3 (Model)**.** Let a shapes-constraint $H = S_1 \leftarrow \alpha_1, \ldots, S_n \leftarrow \alpha_n$, and a typed graph $G = (\mathit{Triple}^G \cup \bigcup_{F\in\mathcal{F}} F^G \cup \bigcup_{S\in\mathcal{S}} S^G)$. For all $i \in 1..n$, let U_i be the set of free variables of α_i. Recall that every variable in U_i is of the form $\#T$ for some triple constraint T. For all $i \in 1..n$ and all e s.t. $e \in S_i^G$, we define the valuation $\mu^G_{e,i}$ of the variables in U_i defined depending on the kind of triple constraint as:

- if $T = P :: F$, then $\mu^G_{e,i}(\#T) = |\mathit{Triple}^G_{|e,P,F^G}|$;
- if $T = P :: @S_j$, then $\mu^G_{e,i}(\#T) = |\mathit{Triple}^G_{|e,P,S_j^G}|$ (for $j \in 1..n$);
- if $T = P :: @\bar{S}_j$, then $\mu^G_{e,i}(\#T) = |\mathit{Triple}^G_{|e,P,D}|$ where $D = \mathit{dom}(G) \setminus S_j^G$ (for $j \in 1..n$);
- if $T = P :: \_$ then $\mu^G_{e,i}(\#T) = |\mathit{Triple}^G_{|e,P,\mathit{dom}(G)}|$.

We say that G *is a model of* H, written $G \models H$, if for all $i \in 1..n$ and for all $e \in \mathit{dom}(G)$, it is the case that if $\mu^G_{e,i} \models \alpha_i$, then $S_i(e)$ holds in G. □

Remark 1 (Properties of models of shapes-constraints)**.** The following properties can be witnessed by simple examples.

- There exist shapes-constraints that do not admit a model. This is due to the Presburger constraints that can be unsatisfiable. Moreover, the empty typing might not be a model because of the "ground" triple constraints (i.e. involving the F relations) might not be satisfied.
- For the same graph, there can exist several different typed graphs that are all models of H.

2.3.2 Operational semantics

The operational semantics of shapes-constraints is based on the semantics of stratified Datalog programs.

For the sequel of this section we consider $\mathcal{S} = \{S_1, \ldots, S_n\}$, and we assume fixed a graph G and a shapes-constraint $H = S_1 \leftarrow \alpha_1, \ldots, S_n \leftarrow \alpha_n$.

For a fixed graph G, we define a partial ordering relation $\prec$ on the typings of G. Let G_1 and G_2 be two typings of G, then

$$G_1 \prec G_2 \text{ if } S^{G_1} \subseteq S^{G_2} \text{ for all } S \in \mathcal{S}$$

Note that a typing G_1 of G can be defined by giving simply the interpretation of the $\mathcal{S}$ relations, thus we write $G_1 = G \cup S_1^{G_1} \cup \ldots \cup S_n^{G_1}$.

Let *stratum* be a stratification of H with m strata. For all $k \in 1..m$, we define the operator R_H^k on typed graphs associated with the k^{th} stratum. Intuitively, this operator enriches the types in the k^{th} stratum by new types that can be deduced assuming some initial typing is correct. For a typed graph $G_1 = G \cup S_1^{G_1} \cup \ldots \cup S_n^{G_1}$:

$$\begin{aligned}
&R_H^k(G_1) = G_2, \text{ with} \\
&S_i^{G_2} = S_i^{G_1} \cup \left\{ e \mid \mu_{e,i}^{G_1} \models \alpha_i \right\} \text{ for all } i \text{ s.t. } \mathit{stratum}(i) = k \\
&S_i^{G_2} = S_i^{G_1} \text{ for all } i \text{ s.t. } \mathit{stratum}(i) \neq k
\end{aligned}$$

and where for all $e \in \mathit{dom}(G)$ and all $i \in 1..n$, the valuation $\mu_{e,i}^{G_1}$ is as in Definition 3. Remark that the R_H^k operators leaves unchanged all $S_i^{G_1}$ such that $\mathit{stratum}(i) \neq k$. Note now that for all typed graph G_1 and all $k \in 1..m$, we have $G_1 \prec R_H^k(G_1)$. Therefore, the R_H^k operators are monotonic (for all $k \in 1..m$), and because the number of possible typings of G is finite, we deduce that the R_H^k operators when applied iteratively starting for some typed graph will converge to a fixed point. For a typed graph G_1 and a stratum $k \in 1..m$, we denote $\mathit{fp}(R_H^k(G_1))$ the fixed point obtained by iterating R_H^k starting from G_1.

Definition 4 (Operational solution)**.** Now consider the sequence of typed graphs given by:

$$\begin{aligned}
&G_0 = G \cup \emptyset \cup \ldots \cup \emptyset \qquad (n \text{ times } \emptyset) \\
&G_k = \mathit{fp}(R_H^k(G_{k-1}))
\end{aligned}$$

The graph G_k is called *operational solution* of G, H with the stratification *stratum*.

The following Lemma 1 and Theorem 1 are conjectured here, and should be provable by using similar arguments as for stratified Datalog. The differences w.r.t. Datalog is that the rules of the form $S_i \leftarrow \alpha_i$ are not expressible as constraints in first-order logic, because of the counting constraints. This however should not invalidate the arguments used for showing the minimality of the solution being computed.

Lemma 1. *The operational solution is independent on the stratification stratum.*

Theorem 1 (Operational solution is a minimal model)**.** *For all graph G and shapes-constraint $H = S_1 \leftarrow \alpha_1, \ldots, S_n \leftarrow \alpha_n$, if G_t is the operational solution of G, H then $G_k \models H$. Moreover, for all G_1 s.t. $G_1 \models H$, it holds that $G_t \sqsubseteq G_1$.*

2.3.3 Model theory semantics

The model theory semantics defines what is a desired typing independently on the way how it can be computed. With every shapes-constraint we associate a sentence (closed formula) of Presburger Mondadic Second-Order Logic (PMSO), and we define a model as a typed graph that satisfies that sentence. We also conjecture that the minimal model is equivalent to the operational solution from Definition 4. The model theory semantics is explained in Section 7.

3 Translating SHACL to shapes-constraint

We propose how to map SHACL constraints to the shapes-constraint language. We start by a discussion aiming to explain the contributions of this section, and how they relate to the current SHACL specification.

3.1 Foreword

The current SHACL specification proposes an RDF vocabulary which aim is to be able to define constraints on graphs. This vocabulary contains particular terms that specify kinds of constraints, e.g. *property* constraint, *or* constraint (disjunction), *and* constraint (conjunction), qualified value constraint, etc. The specification also gives what are the allowed properties of each kind of constraint, and explains how it should be understood. The explanation is given both in textual form and as a SPARQL query. The current specification does not make it explicit what is to be considered as a meaningful shape constraint (i.e. there is no "SHACL for SHACL"). One can write graphs using the SHACL vocabulary for which it is not clear whether they define a meaningful constraint.

Therefore, we do not pretend to give here a formal semantics of *every* graph using the SHACL vocabulary. We rather describe *our understanding* of the kind of constraints that SHACL aims to be able to define. This is done by seeing a SHACL constraints graph in some abstract form, and then by showing how it can be translated to a shapes-constraints (as defined in the previous section). In particular,

> We claim that if the SHACL language is syntactically restricted so that it only allows to write the constraints from the abstraction proposed here, then our formal semantics can be used as formal semantics for SHACL. We believe that our formal semantics coincides with the intended semantics for SHACL. Additionally, the semantics that we propose is well defined for recursion with stratified negation, so it allows to enrich SHACL with recursion with stratified negation.

It should also be pointed out that the semantics that we propose, even though operational, does not necessarily allow to evaluate SHACL in a top-down fashion, which we believe is the preferable one. Without recursion, the current evaluation mechanism of SHACL does work, the difficulty of evaluation might come from recursion with stratified negation. We discuss evaluation in Section 6, in particular we believe that the evaluation mechanism already proposed for ShEx can be adapted and used for SHACL top-down evaluation.

In order to deal with the informal way of defining the semantics of SHACL, we use here informal definitions and informal conjectures, instead of definitions,

conjectures and theorems, hoping that the readers will understand those the same way as the author meant them.

3.2 The translation

We give what we mean by a SHACL constraint.

Informal definition 1 (SHACL constraint)**.** A *SHACL constraint* Q is an RDF graph using the SHACL vocabulary, and that conforms to the restrictions stated in the current SHACL specification regarding the allowed predicates for nodes of some type (e.g. allowed predicates for a node y s.t. (x sh:Shape y) is in Q).

Consider a fixed RDF graph G, and a fixed SHACL constraint Q. Let $S_{x_1}, \dots S_{x_n}$ be all the nodes in Q that have rdf:type sh:Shape. We assume that the rff:type sh:Shape is never omitted for such nodes, in particular when they are objects of sh:valueShape or sh:qualifiedValueShape[3] We are going to explain how to translate the constraints in Q into a shapes-constraint $H = S_{x_1} \leftarrow \alpha_1, \dots, S_{x_n} \leftarrow \alpha_n$ s.t. for all constraint S_{x_i} ($i \in 1..n$) and all node e in G, e satisfies S_{x_i} in SHACL iff $S_{x_i}^{G_t}(e)$ holds, where G_t is the operational solution of G, H.

Some of the aspects of SHACL constraints are going to be encoded in the $\mathcal{F}$-relations of the graph. Other will be handled as shape names. The counting constraints are handled by the Presburger logic. Finally, the aspects that allow to compare values among them are not handled by the shapes-constraint language. Figure 1 classifies every property kind of a sh:property constraint in one of the above four categories, called in the sequel $\mathcal{F}$-constraints, $\mathcal{S}$-constraints, and counting constraints as defined on that figure. Note the currently non supported aspects can be added to the logic while keeping all the good properties of models as studied in Section 2. We prefer here to avoid the unnecessary complications that it would require, and that are not in the focus of this study.

$\mathcal{F}$-constraints	$\mathcal{S}$-constraints	counting constraints	not supported
sh:class	sh:valueShape	sh:xxxCount	sh:equals
sh:classIn	sh:qualifiedValueShape	sh:qualifiedXxxCount	sh:lessThan
sh:datatype			sh:lessaThanOrEquals
sh:datatypeIn			sh:notEquals
sh:directType			sh:uniqueLang
sh:in			sh:hasValue
sh:xxxLength			
sh:xxxXclusive			
sh:nodeKind			
sh:pattern			

Figure 1: How the different aspects of a SHACL constraint are handled in a shapes-constraint.

Let us explain what we mean by aspects handled in the $\mathcal{F}$-relations. We are now interested only in the $\mathcal{F}$-constraints, i.e. those in the left-most column

[3]The SHACL specification says that rdf:type sh:Shape can be omitted.

on Figure 1. Remark that all of them restrict *all* the values of the focus node reachable by a given property.

Assumption on SHACL 1. The current SHACL specification gives the meaning of each type of constraint independently on the others, whereas a sh:property constraint is allowed to use all the constraints from Figure 1 (provided none is repeated). Because the SHACL specification does not specify the meaning of combinations of different kinds of properties, we **assume** that combinations are understood as conjunction, i.e. all the properties must hold. □

Informal definition 2 (Treatment of the $\mathcal{F}$-constraints)**.** For all triple (q sh:property x) that occurs in Q, let (x p_1 y_1), ..., (x p_k y_k) be all the triples with subject x and such that their objects y_j appear in the left-most column of Figure 1. We assume that $\mathcal{F}$ contains a relation $F_{y1,...,yk}$ and $F^G_{y1,...,yk}$ contains exactly those nodes of G that satisfy all the constraints given by the y_j. □

For example, if (x sh:datatype ex:Animal) (x sh:minInclusive 3.14) appear in Q, and there are no other (x p y) in Q with p in the left-most column of Figure 1, then we assume that $\mathcal{F}$ contains the relation $F_{\text{datatype ex:Animal, value} \geq 3.14}$ and its interpretation in G contains exactly the animals that are greater or equal to pi.

We are now ready to explain the translation, by giving for every S_i ($i \in 1..n$) its specification α_i. We distinguish between the different types of constraints.

Property constraints (sh:property) Let (x sh:property y) defining a property constraint (note that in this case, (x rdf:type sh:Shape) in Q). Let p be the predicate of y, i.e. (y sh:predicate p) in Q, and let F_y be the relation as defined in Informal definition 2. Then α_x is obtained as a conjunction of several elements, some of which are omitted depending on the constraints on y:

$$\begin{aligned}
\alpha_x = & \#(p :: _) \geq min \wedge && (1)\\
& \#(p :: _) \leq max \wedge && (2)\\
& \#(p :: F_y) = \#(p :: _) \wedge && (3)\\
& \#(p :: S_z) = \#(p :: _) \wedge && (4)\\
& \#(p :: S_{qz}) \geq qmin \wedge && (5)\\
& \#(p :: S_{qz}) \leq qmax && (6)
\end{aligned}$$

where

- min is such that (y sh:minCount min), and line (1) is omitted when there is no sh:minCount specified;
- max is such that (y sh:maxCount max), and line (2) is omitted when there is no sh:maxCount specified;
- F_y is the relation as defined in Informal definition 2, and line 3 is omitted if there is no (y q min) with q being a $\mathcal{F} - constraint$;
- S_z is such that (y sh:valueShape z), and line (4) is omitted if there is no sh:valueShape specified;
- S_{qz} is such that (y sh:qualifiedValueShape qz), and line (5) is omitted when there is no sh:qualifiedMinCount specified, and similarly line (6) is omitted when there is no sh:qualifiedMaxCount specified.

And, Or and Not constraints Let (x rdf:type sh:Shape) and (x sh:constraint y) and (y sh:not z) in Q, is this case (z rdf:type sh:Shape). Then we define

$$\alpha_x = \neg\alpha_z$$

Let (x rdf:type sh:Shape) and (x sh:constraint y) and (y sh:and $(z_1, \ldots, z_k)$) in Q, where $(z_1, \ldots, z_k)$ is a list of nodes that are rdf:type sh:Shape. Then we define

$$\alpha_x = \bigwedge_{i \in 1..k} \alpha_{z_i}$$

Finally, let (x rdf:type sh:Shape) and (x sh:constraint y) and (y sh:or $(z_1, \ldots, z_k)$) in Q, where $(z_1, \ldots, z_k)$ is a list of nodes that are rdf:type sh:Shape. Then we define

$$\alpha_x = \bigvee_{i \in 1..k} \alpha_{z_i}$$

Correctness of the translation

Informal conjecture 1 (SHACL translation is correct)**.** *The translation described in this section is correct w.r.t. the SHACL specification. That is, for all graph Q that specifies SHACL constraints without recursion and that does not use any of the properties from the right-most column on Figure 1, for all RDF graph G enriched with an interpretation of the $\mathcal{F}$-relations, and for all node e is G, it is the case that e satisfies a constraint x (where x is a rdf:type sh:Shape node in Q) iff $S_x^{G_t}(e)$ holds in G_t, the operational solution of G, H, where H is the translation of SHACL into shapes-constraint described in this section.*

Note that in the current SHACL specification, recursion is not allowed, so Q from Informal conjunction 1 is *non-recursive*. Note also that, as far as we are aware of, the current SHACL specification does not state what it means for a constraint Q to be recursive. We define the notion of recursion in the following section, and we propose semantics for SHACL with recursion.

3.3 Recursion in SHACL

For Q a graph defining a SHACL constraint and two sh:Shape nodes x and y, we say that x *depends on* y if there is a non empty path from x to y using only the predicates sh:Property, sh:valueShape, sh:qualifiedValueShape, sh:and, sh:or, sh:not, sh:constraint. Such a path is called a dependency path. [TODO: check whether this is a precise definition of dependence]. We say that Q is recursive if there exists a node x s.t. (x rdf:type sh:Shape) and x depends on x. We say that Q is with stratified negation if for all dependency path from x to x (and for all x), the dependency path does not contain a sh:not edge. It is well known that this is an equivalent definition for stratification, and that if the dependency graph satisfies such constraint, then an actual stratification does exist. From now on we assume only stratified SHACL constraints.

We now argue, in a quite informal way, how to extend SHACL with recursion. Let Q be SHACL constraint possibly with recursion and negation. We can translate Q into a shapes-constraint H as explained in Section 3.2. Then we conjecture that the resulting shapes-constraint is stratified. We also conjecture that it defines the desired notion of recursion for SHACL. In particular, as

far as we know, one of required properties for recursion in SHACL is that if during validation, in order to check whether node e satisfies shape S we need to check this same thing again, then SHACL considers that there is violation. This property is a least-fixed point property, and is intuitively verified by the operational solution of a shapes-definition because the latter is defined as a least fixed point.

4 Translating shapes-definition to SHACL

We conjecture that there is a simple syntactic restriction of shapes-definition that allows to translate them to SHACL.

Conjecture 1. *Let a shapes definition $H = S_1 \leftarrow \alpha_1, \ldots, S_n \leftarrow \alpha_n$ such that:*

- *no Div_k relation is used in none of the α_i, and*
- *no $+$ is used in none of the α_i, and*
- *no triple constraint in H uses $\bar{S}$ for some shape name S, and*
- *all triple constraint $P :: F$ in H for $F \in \mathcal{F}$ is such that the relation F is definable using the $\mathcal{F}$-constraints from Figure 1.*

Then there is a SHACL constraint Q that is equivalent to H, that is, such that for all graph G and all node e in G, if G_t is the operational solution of G, H, then $S^{G_t}(e)$ holds iff e satisfies Q.

The translation is quite obvious. Regarding the proof of the conjecture, it should be immediate if Informal conjecture 1 holds.

5 Translating ShEx to shapes-constraint

We adopt here a simplified syntax of ShEx along the lines of [3], but with negation on shape names. We also adopt some syntactic restrictions on ShEx with respect to the previous definitions in [3] and [1].

Let $\mathcal{S}$ be a finite set of shape names, and consider the set $\mathcal{T}$ of triple constraints as defined in Section 2.2. A *regular bag expression* E over $\mathcal{T}$ satisfies the following grammar:

$$E ::= \epsilon \mid T \mid E_1 | \ldots | E_k \mid E_1, \ldots, E_k \mid E^I$$

where ϵ is interpreted as the empty bag, $T \in \mathcal{T}$ is a singleton bag containing only T, the '|' in $E_1 | \ldots | E_k$ is the choice operator of regular expressions extended to its k-ary version, we require that $k \geq 2$, the ',' in $E_1, \ldots, E_k$ is unordered concatenation extended to a k-ary version for $k \geq 2$, and $I = [min, max]$ is a (possibly unbounded) interval of natural numbers.

A *ShEx schema* is a set of rules of the form

$$S \leftarrow (\text{EXTRA}P)?(\text{CLOSED})?E$$

for all $S \in \mathcal{S}$, and where all E are regular bag expressions over $\mathcal{T}$, P is a set of *Iri* called extra properties, and CLOSED and EXTRA are reserved keywords. Moreover, we require some syntactic restrictions on a ShEx schema:

- the interval constructs cannot be nested, and cannot appear above some unordered conjunction. That is, i.e. in all E^I that appears in some shape expression (possibly as sub-expression), the regular bag expression E does not contain the interval construct, neither the , construct;

- without loss of generality, we suppose that the | and , operators are flattened, that is $E_1|(E_2|E_3)$ is transformed into $E_1|E_2|E_3$.

Apart from the syntactic restrictions here above, the difference of this definition of ShEx with respect to the one in [1] is that here, we do not allow conjunctions in triple constraints. That is, here we do not allow triple constraints of the form $p :: (F \wedge S \wedge \neg S')$. Such constraints can be added to the shapes-constraint language, but we prefer to avoid unnecessary complexity that this would bring.

We now define the semantics of ShEX in terms of shapes-constraints. The semantics defined here differs from the one in [1] in that it treats the , (unordered concatenation) operator differently. This difference has two consequences. First, the (theoretical) complexity of ShEx validation drops down from NP-complete to polynomial. Second, the semantics of ShEx gets closer to SHACL, hopefully allowing a more direct translation from one to another. These aspects will be discussed in more detail. We will explain the differences in detail later on.

5.1 Semantics of ShEx as shapes-constraint

Note: this section is not well formalized in its current version, because of time constraints.

It is well known[4] that with every regular bag expression E over a finite alphabet $\mathcal{T}$, one can associate a Presburger formula β_E which variables are $\{\#T \mid T \in \mathcal{T}\}$ and such that a bag w over $\mathcal{T}$ belongs to the language of E iff the valuation $\mu = [\#T_1 \mapsto w(T_1), \ldots, \#T_n \mapsto w(T_n)]$ satisfies β_E. Recall that a bag w over is a mapping from $\mathcal{T}$ to $\mathbb{N}$. Moreover, this translation is effective.

Example 2. Let $\mathcal{T} = \{T_a, T'_a, T_b, T_c\}$. Let $E = (T_a|T_b), (T'_a|T_c)$. Then one possible corresponding Presburger formula for E is $\#T_a+\#T_b = 1 \wedge \#T'_a+\#T_c = 1$. Another, equivalent formula is $((\#T_a = 0 \wedge \#T_b = 1) \vee (\#T_a = 1 \wedge \#T_b = 0)) \wedge ((\#T'_a = 0 \wedge \#T_c = 1) \vee (\#T'_a = 0 \wedge \#T_c = 1))$. Note already that the latter is translatable to SHACL, whether the former is not.

We now define the semantics of ShEx. For all $S \leftarrow \alpha$ with $\alpha = \ldots E$ (where the $\ldots$ represent the possible EXTRA and CLOSED modifiers that we will treat later on), we define the Presburger formula β_E as follows. If E has n occurrences of triple constraints, then consider $\mathcal{T} = \{T_1, \ldots, T_n\}$ and a one-to-one mapping from the triple constraints in E to the elements of $\mathcal{T}$. Note that the T_i are pairwise distinct. Then, replace every triple constraint in E by the corresponding T_i. Thus, we rendered E a single occurrence expression. This will result in a difference of the semantics of ShEx w.r.t. [3] and [1].

Now, we construct β_E, the Presburger formula that corresponds to the single-occurrence version of E. The fact that E is considered single-occurrence, to-

[4] by Parikh theorem, and by the properties of Presburger arithmetic, in particular that every Presburger formula is equivalent to a quantification-free formula using the modula Div_k operators

gether with the syntactic constraints imposed here above, simplifies the construction and β_E, and also simplifies the formula itself. In particular, the formula does not use the Div_k relations.

Because of time constraints, we only give a brief overview of the missing results.

We give here an algorithm for the construction:

[TODO]

Then we enrich the resulting Presburger formulas so that they handle the EXTRA and CLOSED constraints.

[TODO]

Then we show how we "undo" the simple-occurrence requirement, and what are the consequences for the ShEx semantics w.r.t. the previous version. One consequence is that the validation problem becomes polynomial. The previous and the current semantics coincide on graphs with so called "excluded middle", that is, graphs in which the neighbors of a focus node never satisfy two different triple constraints that participate in the definition of the same shape.

About the translation from ShEx to SHACL The Presburger formulas that result from the translation from ShEx to shapes-definition use '+', therefore are not directly translatable in SHACL. For now, it is not clear whether equivalent formulas w/o + can be obtained, this is not excluded.

Restrictions on the usage of EXTRA and CLOSED are also needed.

Naturally, if SHACL allows for some kind of '+' in the Presburger formulas, then it would be able to capture the whole fragment of ShEx presented above.

If we further restrict ShEx and allow arbitrary intervals only on triple constraints, then ShEx becomes translatable to SHACL (which is not so surprising). Maybe we can also allow unbounded intervals (Kleene *) on arbitrary expressions (while still keeping the above stated restrictions).

About the recursion mechanism in ShEx The syntactic restriction of ShEx from [1] corresponds to a limited form of stratification, in which positive loops are allowed only in the highest stratum (whereas for general stratification, positive loops can occur in every stratum).

A more fundamental difference is that ShEx does not consider the lest solution, but a solution that is minimal on all strata except for the highest one. The user requirement behind that is that for the following graph and constraint, we would like to say that $< e1 >$ satisfies S.

```
Data
<e1> :p <e2> .
<e2> :p <e1> .

Shape
S <- #T = 1
with
T = :p :: @S
```

An ad-hoc proof of the well-foundedness of ShEx semantics is already given in [1]. This proof can be made more elegant by exploiting the above mentioned link with stratification, least model and any model.

6 Efficient evaluation of shapes-constraint

[TODO]

7 Presburger Monadic Second-Order Logic for Graphs (PMSOG)

We consider Monadic Second-Order Logic that allows to define Presburger constraints on the neighbourhood of nodes. It is inspired by PMSO for trees from [2]. Consider a countable set of *Dom* variables ranging over $x, y, z, \ldots$, and a countable set of *Dom*-set variables ranging over $X, Y, Z, \ldots$ (i.e. a variable X will be interpreted as a subset of *Dom*). A PMSOG formula ϕ is defined by the following syntax.

$$\begin{aligned} \phi ::= \quad & \mathit{Triple}(x,y,z) \mid F(x) \mid x \in X \mid x/\alpha \mid \qquad (\text{for } F \in \mathcal{F}) \\ & \phi \wedge \phi \mid \neg\phi \mid \exists x.\phi \mid \exists X.\phi \end{aligned}$$

where α is a Presburger formula with free variables of the form $\#X$ for X a set variable. The interpretation of x/α is that that the neighbourhood of the node x satisfies the Presburger formula α, where every variable $\#X$ is interpreted as the number of neighbours of the node x that are in the set X.

[TODO: to be continued]

We show how the shapes-constraint language is expressible in PMSOG, thus providing an alternative, model-theoretic semantics of that language.

References

[1] Iovka Boneva, Jose E. Labla Gayo, Eric Prud'hommeaux, and Slawek Staworko. Shape expressions schemas. `http://arxiv.org/abs/1510.05555`, 2015.

[2] Helmut Seidl, Thomas Schwentick, and Anca Muscholl. Counting in trees. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*., pages 575–612, 2008.

[3] Slawek Staworko, Iovka Boneva, Jose E. Labra Gayo, Samuel Hym, Eric Prudhommeaux, and Harold Solbrig. Complexity and Expressiveness of ShEx for RDF.