SHACL Satisfiability and Containment

Paolo Pareti[1], George Konstantinidis[1], Fabio Mogavero[2], and Timothy J. Norman[1]

[1] University of Southampton, Southampton, United Kingdom
{pp1v17,g.konstantinidis,t.j.norman}@soton.ac.uk
[2] Università degli Studi di Napoli Federico II, Napoli, Italy
fabio.mogavero@unina.it

Abstract. The *Shapes Constraint Language (SHACL)* is a recent W3C recommendation language for validating RDF data. Specifically, SHACL documents are collections of constraints that enforce particular shapes on an RDF graph. Previous work on the topic has provided theoretical and practical results for the validation problem, but did not consider the standard decision problems of *satisfiability* and *containment*, which are crucial for verifying the feasibility of the constraints and important for design and optimization purposes. In this paper, we undertake a thorough study of different features of non-recursive SHACL by providing a translation to a new first-order language, called SCL, that precisely captures the semantics of SHACL w.r.t. satisfiability and containment. We study the interaction of SHACL features in this logic and provide the detailed map of decidability and complexity results of the aforementioned decision problems for different SHACL sublanguages. Notably, we prove that both problems are undecidable for the full language, but we present decidable combinations of interesting features.

1 Introduction

The Shapes Constraint Language (SHACL) has been recently introduced as a W3C recommendation language for the validation of RDF graphs, and it has already been adopted by mainstream tools and triplestores. A SHACL document is a collection of *shapes* which define particular constraints and specify which nodes in a graph should be validated against these constraints. The ability to validate data with respect to a set of constraints is of particular importance for RDF graphs, as they are schemaless by design. Validation can be used to detect problems in a dataset and it can provide data quality guarantees for the purpose of data exchange and interoperability.

Recent work has focused on defining precise semantics and implementations for validation of SHACL documents, in particular for the case of recursion [3,9]. In this paper, instead, we focus on the decision problems of satisfiability and containment for SHACL documents; problems which have not been previously investigated. Given a particular SHACL document, satisfiability is the problem of deciding whether there is an RDF graph which is validated by the document; we also investigate finite satisfiability, that is, whether there exists a valid graph of finite size. Containment studies whether a particular SHACL document is subsumed by a second one; that is, whether all graphs that are validated by the first are also validated by the second. We investigate whether

these decision problems can be decided not only at the level of documents, but also for individual shape constraints within documents.

Satisfiability and containment are standard decision problems that have important applications in optimization and design. When integrating two datasets subject to two different SHACL documents, for example, it is important to know whether the two SHACL documents are in conflict with each other, or if one of them is subsumed by the other. At the level of shapes, an unsatisfiable shape constraint might not necessarily cause the unsatisfiability of a whole SHACL document, but it is likely an indication of a design error. Being able to decide containment for individual shapes offers more design choices to the author of a SHACL document, and it is a venue for optimization.

In this paper we focus on the *core constraint components* of SHACL [17] and we do not consider recursion. Validation under recursion is left unspecified in SHACL, and while different semantics have been proposed [3,9], we already show that the language has undecidable satisfiability and containment, even without recursion. We present a translation of SHACL into `SCL`, a new fragment of first-order logic extended with counting quantifiers and the transitive closure operator. To the best of our knowledge such a translation has not been attempted before. Our approach translates a SHACL document/constraint to an `SCL` equisatisfiable sentence/formula, i.e., there is a valid RDF graph for the first if and only if there is a model for the second.

Distinct SHACL constructs translate to particular `SCL` features of different expressiveness. We identify eight such prominent features (such as counting quantifiers or transitive closure) that can be used on top of a base logic and study their interactions. On one hand, the full language is undecidable and, in fact, so are most fragments with just three or four features. On the other hand, our base language has decidable satisfiability and containment, and it is ExpTime-complete. We create a detailed map, in between these extremes, proving positive and negative results for many interesting combinations.

2 Background and Problem Definition

The core structure of the RDF data model is a graph whose nodes and edges are defined by a set of *triples*. A triple $\langle s, p, o\rangle$ identifies an edge with label p, called *predicate*, from a node s, called *subject*, to a node o, called *object*. The main type of entities that act as nodes and edges in RDF graphs are IRIs. We represent RDF graphs in Turtle syntax and by abbreviating IRIs using XML namespaces; the namespace `sh` refers to SHACL terms.

In an RDF graph, *literal* constants (representing datatype values) can only appear in the object position of a triple. *Generalized* RDF [10] describes graphs where literals can be in any position. Most of our results apply to both data models. We will use the generalized one to simplify our presentation and we will state clearly when a result does not apply to the standard, non-generalized one. We do not make use of variables in the predicate position in this paper and so we represent triples as binary relations in FOL. We will use the atom $R(s,o)$ as a shorthand for $\langle s, R, o\rangle$. We will use a minus sign to identify the *inverse* atom, namely $R^{-}(s,o) = R(o,s)$. We will use the binary relation name `isA` to represent class membership triples $\langle s, \texttt{rdf:type}, o\rangle$ as $\texttt{isA}(s,o)$.

SHACL defines constraints that can validate `RDF` graphs [17]. A SHACL document is a set of *shapes*. A shape, denoted $\texttt{s}{:}\langle t, d\rangle$, has three main components: (1) a set of

```
:studentShape a sh:NodeShape ;
   sh:targetClass :Student ;
   sh:not :disjFacultyShape .
:disjFacultyShape a sh:PropertyShape ;
     sh:path (:hasSupervisor :hasFaculty);
     sh:disjoint :hasFaculty .
```

```
:Alex a :Student ;
  :hasFaculty :CS ;
  :hasSupervisor :Jane .
:Jane :hasFaculty :CS .
```

Fig. 1: A SHACL document (left) and a graph that validates it (right).

constraints which are used in conjunction, and hence referred to as a single constraint d; (2) a set of *target declarations*, referred to as *target definition* t, which provides a set of RDF nodes that are validated against d; and (3) a *shape name* `s`. One can think of t and d as unary queries over the nodes of G. Given a node n in a graph G, and a shape $\mathtt{s}{:}\langle t, d\rangle$, we denote with $G \models t(n)$ the fact that node n that satisfies definition t, and $G \models d(n)$ denotes that a node n validates d in G. A graph G validates a shape $\mathtt{s}{:}\langle t, d\rangle$, formally $G \models \mathtt{s}{:}\langle t, d\rangle$, iff every node in the target t validates the constraints d, that is, iff for all $n \in G$, if $G \models t(n)$ then $G \models d(n)$. An empty target definition is never satisfied while an empty constraint definition is always satisfied. A graph G validates a set of shape definitions, i.e. a SHACL document, M, formally $G \models M$, iff G validates all the shapes in M. Constraints might refer to other shapes. When a shape is referenced by another shape it can be handed down a set of *focus nodes* to validate, in addition to those from its own target definition. A shape is *recursive* when it references itself (directly or through other shapes). As mentioned, we focus on non-recursive SHACL documents using the SHACL core constraint components. Without loss of generality, we assume that shape names in a SHACL document do not occur in other SHACL documents or graphs.

The example SHACL document in Figure 1 defines the constraint that, intuitively, all students must have at least one supervisor from the same faculty. The shape with name `:studentShape` has class `:Student` as a target, meaning that all members of this class must satisfy the constraint of the shape. The constraint definition of `:studentShape` requires the non-satisfaction of shape `:disjFacultyShape`, i.e., a node satisfies `:studentShape` if it does not satisfy `:disjFacultyShape`. The `:disjFacultyShape` shape states that an entity has no faculty in common with any of their supervisors (the `sh:path` term defines a property chain, i.e., a composition of roles `:hasSupervisor` and `:hasFaculty`). A graph that validates these shapes is provided in Figure 1. It can be made invalid by changing the faculty of `:Jane` in the last triple.

We now define the SHACL satisfiability and containment problems.

(i) **SHACL Satisfiability**: A SHACL document M is satisfiable iff there exists a graph G such that $G \models M$.
(ii) **Constraint Satisfiability**: A SHACL constraint d is satisfiable iff there exists a graph G and a node n such that $G \models d(n)$.
(iii) **SHACL Containment**: For all SHACL documents M_1, M_2, we say that M_1 is contained in M_2, denoted $M_1 \subseteq M_2$, iff for all graphs G, if $G \models M_1$ then $G \models M_2$.
(iv) **Constraint Containment**: For all SHACL constraints d_1 and d_2 we say that d_1 is contained in d_2, denoted by $d_1, \subseteq d_2$ iff for all graphs G and nodes n, if $G \models d_1(n)$ then $G \models d_2(n)$.

The satisfiability and containment problems for constraints can be reduced to SHACL satisfiability, as follows. A constraint d is satisfiable iff there exists a constant $\mathtt{c}$, either occurring in d or a fresh one, such that the SHACL document corresponding to shape $\mathtt{s}{:}\langle t_\mathtt{c}, d\rangle$ is satisfiable, where $t_\mathtt{c}$ is the target definition that targets node $\mathtt{c}$. Similarly, constraint d_1 is not contained in d_2 iff there exists a constant $\mathtt{c}$, occurring in d_1, d_2 or a fresh one, such that the SHACL document corresponding to shape $\mathtt{s}{:}\langle t_\mathtt{c}, d'\rangle$ is satisfiable; $d'(x)$ is true whenever $d_1(x)$ is true and $d_2(x)$ if false. Thus, satisfiability and containment of constraints in a given SHACL fragment are decidable whenever SHACL satisfiability of that fragment is decidable, and have the same complexity upper bound. However, undecidability of SHACL satisfiability in a fragment does not necessarily imply undecidability for the two constraint problems; we leave this as an open problem.

3 A First Order Language for SHACL Documents

In this section we present a translation of SHACL into an equisatisfiable fragment of FOL extended with counting quantifiers and the transitive closure operator, called `SCL`. As discussed before, for a shape $\mathtt{s}{:}\langle t, d\rangle$ in a SHACL document M, t and d can be seen as unary queries. Intuitively, given a suitable translation q from SHACL into FOL, M is satisfiable iff the sentence $\bigwedge_{\mathtt{s}:\langle t,d\rangle\in M} \forall x.\ q(t(x)) \rightarrow q(d(x))$ is satisfiable, i.e., a node in the target definition of a shape needs to satisfy its constraint, for every shape. We subsequently present an approach that constructs such a sentence. This is reminiscent of [8], where a SHACL document M is translated into a SPARQL query that is true on graphs which however violate M. Intuitively, this query corresponds to sentence $\bigvee_{\mathtt{s}:\langle t,d\rangle\in M} \exists x.q(t(x)) \wedge \neg q(d(x))$, i.e. the negation of the sentence above. Nevertheless, several assumptions made in [8], such that ordering two values is not more complex than checking their equivalence, do not hold for the purposes of satisfiability and containment. We will use τ to denote the translation function from a SHACL document M to an `SCL` sentence $\tau(M)$, which is polynomial in the size of M and computable in polynomial time. We refer to the appendix[3] for the complete translations of τ and its inverse τ^-.

Next, we present our grammar of `SCL` in Def. 1. For simplicity, we assume that target definitions contain at most one target declaration, and that shapes referenced by other shapes have an empty target definition. This does not affect generality, as any shape can be trivially split in multiple copies: one per target declaration and one without any. Letters in square brackets in Def. 1 are annotations naming `SCL` features and thus are not part of the grammar. The top-level symbol φ in `SCL` corresponds to a SHACL document. This could be empty ($\top$), a conjunction of documents, or the translation of an individual shape. A sentence that corresponds to a single shape could have five different forms in `SCL`, depending on the target definition of the translated shape. These are summarized in Table 1, where $\tau_d(x)$ is the `SCL` translation of the constraint of the shape. In SHACL only four types of target declarations are allowed: (1) a particular constant `c` (node target), (2) instances of class `c` (class target), or (3)/(4) subjects/objects of a triple with predicate `R` (subject-of/object-of target). Our translation function gives explicit names to referenced shapes using the `hasShape` relation. We refer to the last component of

[3] http://w3id.org/asset/ISWC2020

Table 1: Translation of shape $\mathtt{s}{:}\langle t,d\rangle$ in SCL with respect to its target definition t.

Target declaration in t	Translation $\tau(\mathtt{s}{:}\langle t,d\rangle)$
Node target (node c)	$\tau_d(\mathtt{c})$ (equivalent form of: $\forall x.\ x = \mathtt{c} \rightarrow \tau_d(x)$)
Class target (class c)	$\forall x.\mathtt{isA}(x,\mathtt{c}) \rightarrow \tau_d(x)$
Subjects-of target (relation R)	$\forall x,y.R(x,y) \rightarrow \tau_d(x)$
Objects-of target (relation R)	$\forall x,y.R^{-}(x,y) \rightarrow \tau_d(x)$
No target declaration	$\forall x.\ \mathtt{hasShape}(x,\mathtt{s}) \leftrightarrow \tau_d(x)$

the φ rule (i.e., $\forall x.\ \mathtt{hasShape}(x,\mathtt{s}) \leftrightarrow \psi(x)$) as a *referenced shape definition* and to its internal constant s as *referenced shape*.

The non terminal symbol $\psi(x)$ corresponds to the subgrammar of the SHACL constraints. Within this subgrammar, $\top$ identifies an empty constraint, $x = \mathtt{c}$ a constant equivalence constraint and F a monadic filter relation (e.g. $F^{\mathrm{IRI}}(x)$, true iff x is an IRI). By *filters* we refer to the SHACL constraints about ordering, node-type, datatype, language tag, regular expressions and string length. Filters are captured by $F(x)$ and the O component. The C component captures qualified value shape cardinality constraints. The E, D and O components capture the equality, disjointedness and order property pair components. The $\pi(x,y)$ subgrammar models SHACL property paths. Within this subgrammar S denotes sequence paths, A denotes alternate paths, Z denotes a zero-or-one path and T denotes a zero-or-more path.

Definition 1. The *SHACL* first-order language (SCL, for short) is the set of *sentences (φ)* and *one-variable formulas ($\psi(x)$)* built according to the following context-free grammar, where c and s are constants (from disjoint domains), F is a monadic-filter name, R is a binary-relation name, * indicates the transitive closure of the relation induced by $\pi(x,y)$, the superscript $\pm$ refers to a relation or its inverse, and $n \in \mathbb{N}$:

$$
\begin{aligned}
\varphi &:= \top \mid \psi(\mathtt{c}) \mid \forall x\,.\,\mathtt{isA}(x,\mathtt{c}) \rightarrow \psi(x) \mid \forall x,y\,.\,R^{\pm}(x,y) \rightarrow \psi(x) \mid \varphi \wedge \varphi;\ \mid \\
&\quad\ \forall x.\ \mathtt{hasShape}(x,\mathtt{s}) \leftrightarrow \psi(x)\,;\\
\psi(x) &:= \top \mid x = \mathtt{c} \mid F(x) \mid \mathtt{hasShape}(x,\mathtt{s}) \mid \neg\psi(x) \mid \psi(x) \wedge \psi(x) \mid \\
&\quad\ \exists y.\,\pi(x,y) \wedge \psi(y) \mid \neg\,\exists y.\,\pi(x,y) \wedge R(x,y)\,\texttt{[D]} \mid \forall y.\,\pi(x,y) \leftrightarrow R(x,y)\,\texttt{[E]} \mid \\
&\quad\ \forall y,z\,.\,\pi(x,y) \wedge R(x,z) \rightarrow \sigma(y,z)\,\texttt{[O]} \mid \exists^{\geq n} y\,.\,\pi(x,y) \wedge \psi(y)\,\texttt{[C]};\\
\pi(x,y) &:= R^{\pm}(x,y) \mid \exists z\,.\,\pi(x,z) \wedge \pi(z,y)\,\texttt{[S]} \mid x = y \vee \pi(x,y)\,\texttt{[Z]} \mid \pi(x,y) \vee \pi(x,y)\,\texttt{[A]} \mid \\
&\quad\ (\pi(x,y))^{\star}\,\texttt{[T]};\\
\sigma(x,y) &:= x <^{\pm} y \mid x \leq^{\pm} y.
\end{aligned}
$$

To enhance readability, we define the following syntactic shortcuts:

(i) $\psi_1(x) \vee \psi_2(x) \doteq \neg(\neg\psi_1(x) \wedge \neg\psi_2(x))$;
(ii) $\pi(x,\mathtt{c}) \doteq \exists y.\pi(x,y) \wedge y = \mathtt{c}$;
(iii) $\forall y\,.\,\pi(x,y) \rightarrow \psi(y) \doteq \neg\,\exists y\,.\,\pi(x,y) \wedge \neg\psi(y)$.

Our translation τ results in a subset of SCL sentences, called *well-formed*. An SCL sentence is well-formed if for every occurrence of a referenced shape s there is a

```
select ?x where {
    ?x rdf:type :Student .
    filter not exists {
        ?x :hasSupervisor ?z .
        ?z :hasFaculty ?y .
        ?x :hasFaculty ?y . } }
```

$$
\begin{aligned}
&(\forall x.\ \texttt{isA}(x, \texttt{:Student}) \rightarrow \\
&\quad \neg \texttt{hasShape}(x, \texttt{:disjFacultyShape})) \wedge \\
&(\forall x.\ \texttt{hasShape}(x, \texttt{:disjFacultyShape}) \leftrightarrow \\
&\quad \neg\, \exists y.\ (\ \exists z.\ R_{\texttt{:hasSupervisor}}(x,z) \wedge \\
&\qquad\qquad R_{\texttt{:hasFaculty}}(z,y) \wedge \\
&\qquad\qquad R_{\texttt{:hasFaculty}}(x,y)\ )\ )
\end{aligned}
$$

Fig. 2: Translation of the SHACL document from Fig. 1 into the SPARQL query that looks for violations (left) and into an SCL sentence (right).

Table 2: Relation between prominent SHACL components and SCL expressions.

Abbr.	Name	SHACL component	Corresponding expression
S	Sequence Paths	Sequence Paths	$\exists z\,.\,\pi(x,z) \wedge \pi(z,y)$
Z	Zero-or-one Paths	sh:zeroOrOnePath	$x = y \vee \pi(x,y)$
A	Alternative Paths	sh:alternativePath	$\pi(x,y) \vee \pi(x,y)$
T	Transitive Paths	sh:zeroOrMorePath sh:oneOrMorePath	$(\pi(x,y))^\star$
D	Property Pair Disjointness	sh:disjoint	$\neg\,\exists y.\pi(x,y) \wedge R(x,y)$
E	Property Pair Equality	sh:equals	$\forall y\,.\,\pi(x,y) \leftrightarrow R(x,y)$
O	Property Pair Order	sh:lessThanOrEquals	$x \leq^{\pm} y$ and $x <^{\pm} y$
C	Cardinality Constraints	sh:qualifiedValueShape sh:qualifiedMinCount sh:qualifiedMaxCount	$\exists^{\geq n} y\,.\,\pi(x,y) \wedge \psi(y)$ with $n \neq 1$

corresponding referenced shape definition sentence with the same s, and no referenced shape definitions are recursively defined. Fig. 2 shows the translation of the document from Fig. 1, into a SPARQL query, via [8], and a well-formed SCL sentence, via τ.

To distinguish different fragments of SCL, Table 2 lists a number of *prominent* SHACL components, that is, important for the purpose of satisfiability. The language defined without any of these constructs is our *base* language, denoted $\varnothing$. When using such an abbreviation of a prominent feature, we refer to the fragment of our logic that includes the base language together with that feature enabled. For example, SA identifies the fragment that only allows the base language, sequence paths and alternate paths.

The SHACL specification presents an unusual asymmetry in the fact that equality, disjointedness and order components forces one of their two path expressions to be an atomic relation. This can result in situations where the order constraints can be defined in just one direction, since only the less-than and less-than-or-equal property pair constraints are defined in SHACL. Our O fragment models a more natural order comparison that includes the $>$ and $\geq$ components. We instead denote with O' the fragment where the order relations in the $\sigma(x,y)$ subgrammar cannot be inverted.

Relying on the standard FOL semantics, we define the satisfiability and containment for SCL sentences, as well as the closely related finite-model property, in the natural way.

SCL Sentence Satisfiability An SCL sentence ϕ is satisfiable iff there exists a first-order structure Ω such that $\Omega \models \phi$.

SCL Sentence Containment For all SCL sentences ϕ_1, ϕ_2, we say that ϕ_1 is contained in ϕ_1, denoted $\phi_1 \subseteq \phi_2$, iff, for all first-order structures Ω, if $\Omega \models \phi_1$ then $\Omega \models \phi_2$.

SCL Finite-model Property An SCL sentence ϕ (resp. formula $\psi(x)$) enjoys the finite-model property iff whenever ϕ is satisfiable, it is so on a finite model.

In the following two subsections, we discuss SHACL-to-SCL satisfiability and containment. In this respect, we assume that filters are interpreted relations. In particular, we prove equisatisfiability of SHACL and SCL on models that we call *canonical*, that is, having the following properties: (1) the domain of the model is the set of RDF terms, (2) such a model contains built-in interpreted relations for filters, and (3) ordering relations $<^{\pm}$ and $\leq^{\pm}$ are the disjoint union of the total orders of the different comparison types allowed in SPARQL. In Sec. 3.3, we discuss an explicit axiomatization of the semantics of a particular set of filters in order to prove decidability of the satisfiability and containment problems for several SCL fragments in the face of these filters.

3.1 SHACL Satisfiability

A fine-grained analysis of the bidirectional translation between our grammar and SHACL, provided in the appendix, can lead to an inductive proof of equisatisfiability between the two languages. In particular, given a satisfiable SHACL document M which validates an RDF graph G, we can translate G and M into a canonical first-order structure I which models $\tau(M)$, thus proving the latter satisfiable, and vice versa. Intuitively, the structure I is composed of two substructures, Ω_G which corresponds to the translation of triples from G, and $\Omega_{G,M}$ which interprets the hasShape relation. These substructures, as explained below, have disjoint interpretations and we write $I = \Omega_G \cup \Omega_{G,M}$ to denote that I is the structure that considers the union of their domains and of their interpretations.

For any RDF predicate R in G, the structure Ω_G is a canonical structure that interprets the binary relation R as the set of all pairs $\langle s, o\rangle$ for which $\langle s, R, o\rangle$ is in G. The structure $\Omega_{G,M}$ interprets hasShape as the binary relation which, for all referenced shape definitions $\forall x.\ \texttt{hasShape}(x, \texttt{s}) \leftrightarrow \psi(x)$ in $\tau(M)$, it contains a pair $\langle c, s\rangle$ whenever Ω_G satisfies $\psi(c)$. We will call $\Omega_{G,M}$ the *shape definition model* of G and M. Since we do not address recursive shape definitions, this model always exists (corresponding to the *faithful total assignment* from [9]). Inversely, given a well-formed SCL sentence ϕ that is satisfiable and has a model I, by eliminating from I all references of hasShape and then transforming the elements of the relations to triples we get an RDF graph G that is valid w.r.t. the SHACL document $\tau^{-}(\phi)$.

Theorem 1. *For all SHACL documents M: (1) $\tau(M)$ is polynomially computable; (2) M is (finitely) satisfiable iff $\tau(M)$ is (finitely) satisfiable on a canonical model.*
For all well-formed SCL sentences ϕ: (1) $\tau^{-}(\phi)$ is polynomially computable; (2) ϕ is (finitely) satisfiable on canonical models iff $\tau^{-}(\phi)$ is (finitely) satisfiable.

3.2 SHACL Containment

Containment of two SHACL documents does not immediately correspond to the containment of their SCL translations. Given two SHACL documents M_1 and M_2 where

M_1 is contained in M_2, there might exist a first-order structure I that models $\tau(M_1)$ but not $\tau(M_2)$. Notice, in fact, that structure $I = \Omega_G \cup \Omega_{G,M_1}$ models M_1, but that Ω_{G,M_1} does not necessarily model the referenced shape definitions of $\tau(M_2)$. Let $\delta(\phi)$ be the definitions of referenced shapes in an SCL sentence ϕ. Note that for a graph G and a SHACL document M the shape definition model $\Omega_{G,M}$ models $\delta(\tau(M))$. The reduction of SHACL containment into SCL is, therefore, as follows. This result also applies for containment over finite structures.

Theorem 2. *For all SHACL documents M_1 and M_2: (1) $\delta(\tau(M_2))$ is polynomially computable; (2) $M_1 \subseteq M_2$ iff $\tau(M_1) \wedge \delta(\tau(M_2)) \subseteq \tau(M_2)$ on all canonical models.*

Proof. ($\Rightarrow$) Let $M_1 \subseteq M_2$. If M_1 is not satisfiable the theorem holds. If M_1 is satisfiable, let G be any graph that validates M_1, and thus M_2. It holds that $\Omega_G \cup \Omega_{G,M_1}$ models $\tau(M_1)$ per Sec. 3.1, and $\Omega_G \cup \Omega_{G,M_2}$ models $\tau(M_2)$. It is easy to see that if $\Omega_G \cup \Omega_{G,M_1}$ models $\tau(M_1)$ the union of another hasShape interpretation over a disjoint set of shape names, i.e., $\Omega_G \cup \Omega_{G,M_1} \cup \Omega_{G,M_2}$ also models $\tau(M_1)$. Similarly $\Omega_G \cup \Omega_{G,M_1} \cup \Omega_{G,M_2}$ models $\tau(M_2)$ as well.

($\Leftarrow$) If M_1 is not contained in M_2, then there is a graph G that models M_1 but not M_2. Thus, $\Omega_G \cup \Omega_{G,M_1}$ models $\tau(M_1)$ but $\Omega_G \cup \Omega_{G,M_2}$ does not model $\tau(M_2)$. So we have that $\Omega_G \cup \Omega_{G,M_1} \cup \Omega_{G,M_2}$ models $\tau(M_1) \cup \delta(\tau(M_2))$ but not $\tau(M_2)$. □

Since our grammar is not closed under negation we cannot trivially reduce (finite) SCL containment to (finite) SCL satisfiability. Nevertheless, all positive (decidability and complexity) results are obtained by exhibiting inclusion of some SCL fragment into a particular (extension of a) fragment of first-order logic already studied in the literature that is closed under negation. Thus we can always solve the (finite) SCL containment problem for sentences $\phi_1 \subseteq \phi_2$ by deciding (finite) unsatisfiability of a sentence $\phi_1 \wedge \neg\phi_2$. Dually, the unsatisfiability of an SCL sentence ϕ is equivalent to $\phi \subseteq \bot$. Hence, containment and unsatisfiability have the same complexity.

3.3 Filter Axiomatization

Decidability of SCL satisfiability depends on the decidability of filters. In this section we present a decidable axiomatization that allows us to treat some filters as simple relations instead of interpreted ones. In particular, we do not consider sh:pattern which supports complex regular expressions, and the sh:lessThanOrEquals or sh:lessThan that are binary relations (the O and O' components of our grammar). All other features defined as filters in Sec. 3 are represented by monadic relations $F(x)$ of the SCL grammar.

The actual problem imposed by filters w.r.t. deciding satisfiability and containment is that each combination of filters might be satisfied by a limited number of elements (zero, if the combination is unsatisfiable). For example, the number of elements of datatype boolean is two, the number of elements that are literals is infinite and the number of elements of datatype integer that are greater than 0 and lesser than 5 is four.

Let a *filter combination* $\mathbb{F}(x)$ denote a conjunction of atoms of the form $x = \mathtt{c}$, $x \neq \mathtt{c}$, $F(x)$ or $\neg F(x)$, where c is a constant and F is a filter predicate. Given a filter combination, it is possible to compute the number of elements that can satisfy it. Let γ be

the function from filter combinations to naturals returning this number. The computation of $\gamma(\mathbb{F}(x))$ for the monadic filters we consider is trivial as it boils down to determining: (1) the lexical space and compatibility of datatypes and node types (including those implied by language tag and order constraints); (2) the cardinality of intervals defined by order or string-length constraints; and (3) simple RDF-specific restrictions, e.g., the fact that each node has at most one datatype and language tag. Combinations of the previous three points are equally computable. Let $\mathbb{F}^{\varphi}$ be the set of filter combinations that can be constructed with the filters and constants occurring in a sentence φ. The filter axiomatization $\alpha(\varphi)$ of a sentence φ is the following conjunction (conjuncts where $\gamma(\mathbb{F}(x))$ is infinite are trivially simplified to $\top$).

$$\alpha(\varphi) = \bigwedge_{\mathbb{F}(x) \in \mathbb{F}^{\varphi}} \exists^{\leq \gamma(\mathbb{F}(x))}\, x.\ \mathbb{F}(x)$$

Theorem 3. *An* SCL *sentence ϕ is satisfiable on a canonical model iff $\phi \wedge \alpha(\phi)$ is satisfiable on an uninterpreted model. Containment $\phi_1 \subseteq \phi_2$ of two* SCL *sentences on all canonical models holds iff $\phi_1 \wedge \alpha(\phi_1 \wedge \phi_2) \subseteq \phi_2$ holds on all uninterpreted models.*

Proof sketch. We focus on satisfiability, since the proof for containment is similar. First notice that every canonical model I of φ is necessarily a model of $\phi \wedge \alpha(\phi)$. Indeed, by definition of the function γ, given a filter combination $\mathbb{F}(x)$, there cannot be more than $\gamma(\mathbb{F}(x))$ elements satisfying $\mathbb{F}(x)$, independently of the underlying canonical model. Thus, I satisfies $\alpha(\phi)$. Consider now a model I of $\phi \wedge \alpha(\phi)$ and let $I^{\star}$ be the structure obtained from I by replacing the interpretations of the monadic filter relations with their canonical ones. Obviously, for any filter combination $\mathbb{F}(x)$, there are exactly $\gamma(\mathbb{F}(x))$ elements in $I^{\star}$ satisfying $\mathbb{F}(x)$, since $I^{\star}$ is canonical. As a consequence, there exists a injection ι between the elements satisfying $\mathbb{F}(x)$ in I and those satisfying $\mathbb{F}(x)$ in $I^{\star}$. At this point, one can prove that $I^{\star}$ satisfies φ. Indeed, every time a value x, satisfying $\mathbb{F}(x)$ in I, is used to verify a subformula ψ of φ in I, one can use the value $\iota(x)$ to verify the same subformula ψ in $I^{\star}$. □

4 SCL Satisfiability

In this section we embark on a detailed analysis of the satisfiability problem for different fragments of SCL. Some of the proven and derived results are visualized in Figure 3. The decidability results are proved via embedding into known decidable (extensions of) fragments of first-order logic, while the undecidability ones are obtained through reductions from the domino problem. Since we are not considering filters explicitly, but via axiomatization, the only interpreted relations are the equality and the orderings.

For the sake of space and readability, the map depicted in the figure is not complete *w.r.t.* two aspects. First, it misses few fragments whose decidability can be immediately derived via inclusion into a more expressive decidable fragment, e.g., ZADEC or SZATD. Second, the rest of the missing cases have an open decidability problem. In particular, while there are several decidable fragments containing the T feature we do not know any decidable fragment with the O or O' feature. Notice that the undecidability results making use of the last two are only applicable to generalized RDF.

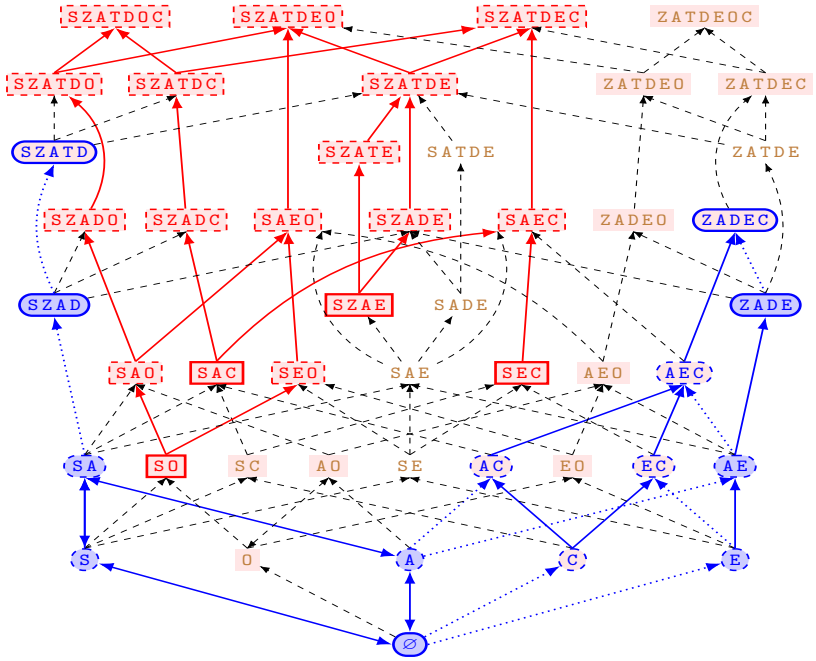

Fig. 3: Decidability and complexity map of SCL. Round (blue) and square (red) nodes denote decidable and undecidable fragments, respectively. Solid borders on nodes correspond to theorems in this paper, while dashed borders are implied results. Directed edges indicate inclusion of fragments, while bidirectional edges denote polynomial-time reducibility. Solid edges are preferred derivations to obtain tight results, while dotted ones leads to worst upper-bounds or model-theoretic properties. Finally, a light blue background indicates that the fragment enjoys the finite-model property, while those with a light red background do not satisfy this property.

As first result, we show that the base language $\varnothing$ is already powerful enough to express properties writable by combining the S, Z, and A features. In particular, the latter one does not augment the expressiveness when the D and O features are considered alone.

Theorem 4. *There are semantic-preserving and polynomial-time finite-model-invariant satisfiability-preserving translations between the following SCL fragments: 1.* $\varnothing \equiv S \equiv Z \equiv A \equiv SZ \equiv SA \equiv ZA \equiv SZA$*; 2.* $D \equiv AD$*; 3.* $O \equiv AO$*; 4.* $DO \equiv ADO$*.*

Proof. To show the equivalences between the fourteen SCL fragments mentioned in the thesis, we consider the following first-order formula equivalences that represent few distributive properties enjoyed by the S, Z, and A features *w.r.t.* some of the other language constructs. The verification of their correctness only requires the application of standard properties of Boolean connectives and first-order quantifiers.

– **[S].** The sequence combination of two path formulas π_1 and π_2 in the body of an existential quantification is removed by nesting two quantifications, one for each π_i:

$$\exists y\,.\,(\exists z\,.\,\pi_1(x,z) \wedge \pi_2(z,y)) \wedge \psi(y) \equiv \exists z\,.\,\pi_1(x,z) \wedge (\exists y\,.\,\pi_2(z,y) \wedge \psi(y)).$$

– **[Z].** The Z path construct can be removed from the body of an existential quantification on a free variable x by verifying whether the formula ψ in its scope is already satisfied by the value bound to x itself:

$$\exists y\,.\,(x = y \vee \pi(x,y)) \wedge \psi(y) \equiv \psi(x) \vee \exists y\,.\,\pi(x,y) \wedge \psi(y).$$

– **[A].** The removal of the A path construct from the body of an existential quantifier or of the D and O constructs can be done by exploiting the following equivalences:

$$\exists y\,.\,(\pi_1(x,y) \vee \pi_2(x,y)) \wedge \psi(y) \equiv (\exists y\,.\,\pi_1(x,y) \wedge \psi(y)) \vee (\exists y\,.\,\pi_2(x,y) \wedge \psi(y));$$
$$\neg\exists y.\,(\pi_1(x,y) \vee \pi_2(x,y)) \wedge R(x,y) \equiv (\neg\exists y.\,\pi_1(x,y) \wedge R(x,y)) \wedge (\neg\exists y.\,\pi_2(x,y) \wedge R(x,y));$$
$$\forall y,z\,.\,(\pi_1(x,y) \vee \pi_2(x,y)) \wedge R(x,z) \rightarrow \sigma(y,z) \equiv (\forall y,z\,.\,\pi_1(x,y) \wedge R(x,z) \rightarrow \sigma(y,z)) \wedge (\forall y,z\,.\,\pi_2(x,y) \wedge R(x,z) \rightarrow \sigma(y,z)).$$

At this point, the equivalences between the fragments naturally follow by iteratively applying the discussed equivalences.

The removal of the Z and A constructs from an existential quantification might lead, however, to an exponential blow-up in the size of the formula due to the duplication of the body ψ of the quantification. To obtain polynomial-time finite-model-invariant satisfiability-preserving translations, we first construct from the given sentence φ a finite-model-invariant equisatisfiable sentence $\varphi^\star$. The latter has a linear size in the original one and all the bodies of its quantifications are just plain relations. Then, we apply the above described semantic-preserving translations to $\varphi^\star$ that, in the worst case, only leads to a doubling in the size. The sentence $\varphi^\star$ is obtained by iteratively applying to φ the following two rewriting operations, until no complex formula appears in the scope of an existential quantification. Let $\psi'(x) = \exists y\,.\,\pi(x,y) \wedge \psi(y)$ be a subformula, where $\psi(y)$ does not contain quantifiers other than possibly those of the S, D, and O features. Then: (i) replace $\psi'(x)$ with $\exists y\,.\,\pi(x,y) \wedge \mathtt{hasShape}(y,\mathtt{s})$, where s is a fresh constant; (ii) conjoin the resulting sentence with $\forall x.\ \mathtt{hasShape}(x,\mathtt{s}) \leftrightarrow \psi(x)$. The two rewriting operations only lead to a constant increase of the size and are applied only a linear number of times. □

It turns out that the base language $\varnothing$ resembles the description logic $\mathcal{ALC}$ extended with universal roles, inverse roles, and nominals [4]. This resemblance is exploited as the key observation at the core of the following result.

Theorem 5. *All* **SCL** *subfragments of* **S Z A** *enjoy the finite-model property. Moreover, the satisfiability problem is ExpTime-complete.*

Proof. The finite-model property follows from the fact that Theorem 8 states the same property for the subsuming language S Z A D. As far as the satisfiability problem is concerned, thanks to Item 1 of Theorem 4, we can focus on the base language $\varnothing$. It can be observed that the description logic $\mathcal{ALC}$ extended with inverse roles and nominals [4] and the language $\varnothing$ deprived of the universal quantifications at the level of sentences are linearly interreducible. Indeed, every existential modality $\exists R.C$ (*resp.*, $\exists R^-.C$) precisely corresponds to the SCL construct $\exists y\,.\,R(x,y) \wedge \psi_C(y)$ (*resp.*, $\exists y\,.\,R^-(x,y) \wedge \psi_C(y)$), where $\psi_C(y)$ represents the concept C. Moreover, every nominal n corresponds to the

equality construct $x = \mathtt{c}_n$, where a natural bijection between nominals and constants is considered. Since the aforementioned description logic has an ExpTime-complete satisfiability problem [24,12], it holds that the same problem for all subfragments of S Z A is ExpTime-hard. Completeness follows by observing that the universal quantifications at the level of sentences can be encoded in the further extension of $\mathcal{ALC}$ with the universal roles [24], which has an ExpTime-complete satisfiability problem [23]. □

To derive properties of the Z A D E fragment, together with its sub-fragments (two of those – E and A E – are shown in Figure 3), we leverage on the syntactic embedding into the two-variable fragment of first-order logic.

Theorem 6. *The ZADE fragment of SCL enjoys the finite-model property. Moreover, the associated satisfiability problem is solvable in NExpTime.*

Proof. Via inspection of the SCL grammar one can notice that, by avoiding the S and O features of the language it is only possible to write formulas with at most two free variables [19]. For this reason, every Z A D E formula belongs to the two-variable fragment of first-order logic which is known to enjoy both the finite-model property and a NExpTime satisfiability problem [14]. □

The embedding used in the previous theorem can be generalized when the C feature is added to the picture. However, this additional expressive power does not come without a price since the complexity increases and the finite-model property is lost.

Theorem 7. *The C fragment of SCL does not enjoy the finite-model property and has a NExpTime-hard satisfiability problem. Nevertheless, the finite and unrestricted satisfiability problems for ZADEC are NExpTime-complete.*

Proof. As for the proof of Theorem 6, one can observe that every Z A D E C formula belongs to the two-variable fragment of first-order logic extended with counting quantifiers. Such a logic does not enjoy the finite-model property [15], since it syntactically contains a sentence that encodes the existence of an injective non-surjective function from the domain of the model to itself. The C fragment of SCL allows us to express the same property via the following sentence φ, thus implying the first part of the thesis:

$$\varphi \doteq \mathtt{isA}(0,\mathtt{c}) \wedge \neg\exists x\,.\,R^{-}(0,x) \wedge \forall x\,.\,\mathtt{isA}(x,\mathtt{c}) \rightarrow \psi(x);$$
$$\psi(x) \doteq \exists^{=1} y\,.\,(R(x,y) \wedge \mathtt{isA}(y,\mathtt{c})) \wedge \neg\exists^{\geq 2} y\,.\,R^{-}(x,y).$$

Intuitively, the first two conjuncts of φ force every model of the sentence to contain an element 0 that does not have any R-predecessor and that is related to c in the isA relation. In other words, 0 is not contained in the image of the relation R. The third conjunct of φ ensures that every element related to c *w.r.t.* isA has exactly one R-successor, also related to c in the same way, and at most one R-predecessor. Thus, a model of φ must contain an infinite chain of elements pairwise connected by the functional relation R.

By generalizing the proof of Theorem 5, one can notice that the C fragment of SCL semantically subsumes the description logic $\mathcal{ALC}$ extended with inverse roles, nominals, and cardinality restrictions [4]. Indeed, every qualified cardinality restriction $(\geq n\,R.C)$ (*resp.*, $(\leq n\,R.C)$) precisely corresponds to the SCL construct $\exists^{\geq n} y\,.\,R(x,y) \wedge$

$\psi_C(y)$ (*resp.*, $\neg\,\exists^{\geq n+1}\,y\,.\,R(x,y)\wedge\psi_C(y)$), where $\psi_C(y)$ represents the concept C. Thus, the hardness result for C follows by recalling that the specific $\mathcal{ALC}$ language has a NExpTime-hard satisfiability problem [25,18]. On the positive side, however, the extension of the two-variable fragment of first-order logic with counting quantifiers has decidable finite and unrestricted satisfiability problems. Specifically, both can be solved in NExpTime, even in the case of binary encoding of the cardinality constants [20,21]. Hence, the second part of the thesis follows as well. □

Thanks to the axiomatization of (the subset of) filters given in Sec. 3.3, it is immediate to see that the ZADEC fragment extended with these filters is decidable as well. Indeed, although the sentence $\alpha(\varphi)$ is not immediately expressible in SCL it belongs to the two-variable fragment of FOL extended with counting quantifiers. Notice however that, since $\alpha(\varphi)$ might be exponential in the size of φ, this approach only leads to a (potentially) coarse upper bound. An attempt to prove a tight complexity result might exploit the SMT-like approach described in [2] for the LTL part of Strategy Logic. Indeed, one could think to extend the decision procedure for the above FOL fragment in such a way that the filter axiomatization is implicitly considered during the check for satisfiability.

For the S Z A D fragment, we obtain model-theoretic and complexity results via an embedding into the unary-negation fragment of first-order logic. When the T feature is considered, the same embedding can be adapted to rewrite S Z A T D into the extension of the above first-order fragment with regular path expressions. Unfortunately, as for the addition of the C feature to Z A D E, we pay the price of losing the finite-model property. In this case, however, no increase of the complexity of the satisfiability problem occurs.

Theorem 8. *The SZAD fragment of SCL enjoys the finite-model property. The STD fragment does not enjoy the finite-model property. However, the finite and unrestricted satisfiability problems for SZATD are solvable in 2ExpTime.*

Proof. By inspecting the SCL grammar, one can notice that every formula that does not make use of the T, E, O, and C constructs can be translated into the standard first-order logic syntax, with conjunctions and disjunctions as unique binary Boolean connectives, where negation is only applied to formulas with at most one free variable. For this reason, every S Z A D formula semantically belongs to the unary-negation fragment of first-order logic, which is known to enjoy the finite-model property [6,7].

Similarly every S Z A T D formula belongs to the unary-negation fragment of first-order logic extended with regular path expressions [16]. Indeed, the grammar rule $\pi(x,y)$ of SCL, precisely resembles the way the regular path expressions are constructed in the considered logic, when one avoids the test construct. Unfortunately, as for the two-variable fragment with counting quantifiers, this logic also fails to satisfy the finite-model property since it is able to encode the existence of a non-terminating path without cycles. The S T D fragment of SCL allows us to express the same property, as described in the following. First of all, consider the S T path formula $\pi(x,y) \doteq \exists\, z\,.\,(R^{-}(x,z)\wedge (R^{-}(z,y))^{\star})$. Obviously, $\pi(x,y)$ holds between two elements x and y of a model if and only if there exists a non-trivial R-path (of arbitrary positive length) that, starting in y, leads to x. Now, by writing the S T D formula $\psi(x) \doteq \neg\,\exists\, y\,.\,(\pi(x,y)\wedge R(x,y))$, we express the fact that an element x does not belong to any R-cycle since, otherwise, there would be an R-successor y able to reach x itself. Thus, by ensuring that every element in

the model has an R-successor, but does not belong to any R-cycle, we can enforce the existence of an infinite R-path. The $\mathtt{STD}$ sentence φ expresses exactly this property:

$$\varphi \doteq \mathtt{isA}(0,\mathtt{c}) \wedge \forall x\,.\,\mathtt{isA}(x,\mathtt{c}) \rightarrow (\psi(x) \wedge \exists y\,.\,(R(x,y) \wedge \mathtt{isA}(y,\mathtt{c}))).$$

On the positive side, however, the extension of the unary-negation fragment of first-order logic with arbitrary transitive relations or, more generally, with regular path expressions has decidable finite and unrestricted satisfiability problems. Specifically, both can be solved in 2ExpTime [1,16,11]. □

At this point, it is interesting to observe that the $\mathtt{O}$ feature allows us to express a very weak form of counting restriction which is, however, powerful enough to lose the finite-model property. For the proof of the following we refer to our appendix.

Theorem 9. *$\mathtt{SCL}$ fragments $\mathtt{O}$ and $\mathtt{EO}$' do not satisfy the finite-model property.*

In the remaining part of this section, we show the undecidability of the satisfiability problem for five fragments of $\mathtt{SCL}$ through a semi-conservative reduction from the standard domino problem [26,5,22], whose solution is known to be Π^1_0-complete. A $\mathbb{N}\times\mathbb{N}$ tiling system $(T,\mathsf{H},\mathsf{V})$ is a structure built on a non-empty set T of domino types, a.k.a. tiles, and two horizontal and vertical matching relations $\mathsf{H},\mathsf{V}\subseteq T\times T$. The domino problem asks for a compatible tiling of the first quadrant $\mathbb{N}\times\mathbb{N}$ of the plane, *i.e.*, a solution mapping $\mathfrak{d}\colon \mathbb{N}\times\mathbb{N}\to T$ such that, for all $x,y\in\mathbb{N}$, both $(\mathfrak{d}(x,y),\mathfrak{d}(x+1,y))\in\mathsf{H}$ and $(\mathfrak{d}(x,y),\mathfrak{d}(x,y+1))\in\mathsf{V}$ hold true.

Theorem 10. *The satisfiability problems of the $\mathtt{SO}$, $\mathtt{SAC}$, $\mathtt{SEC}$, $\mathtt{SEO}$', and $\mathtt{SZAE}$ fragments of $\mathtt{SCL}$ are undecidable.*

Proof. The main idea behind the proof is to embed a tiling system into a model of a particular $\mathtt{SCL}$ sentence that is satisfiable if and only if the tiling system allows for an admissible tiling. The hardest part in the reduction consists in the definition of a satisfiable sentence all of whose models homomorphically contain the infinite grid of the tiling problem. In other words, this sentence should admit an infinite square grid graph as a minor of the model unwinding. Given that, the remaining part of the reduction can be completed in the base language $\varnothing$.

Independently of the fragment we are proving undecidable, consider the sentence

$$\varphi \doteq \textstyle\bigvee_{t\in T} \mathtt{isA}(0,t) \wedge \bigwedge_{t\in T} \forall x\,.\,\mathtt{isA}(x,t) \rightarrow (\psi^t_T(x) \wedge \psi_G(x)).$$

Intuitively, this first states the existence of the point 0, the origin of the grid, labeled by some tile and then ensures the fact that all points x, that are labeled by some tile t, need to satisfy the two formulas $\psi^t_T(x)$ and $\psi_G(x)$. The first formula is used to ensure the admissibility of the tiling, while the second one forces the model to embed a grid.

$$\psi^t_T(x) \doteq \bigwedge_{t'\in T}^{t'\neq t} \neg\mathtt{isA}(x,t') \wedge \left(\forall y\,.\,H(x,y) \rightarrow \bigvee_{(t,t')\in\mathsf{H}} \mathtt{isA}(y,t')\right) \wedge \left(\forall y\,.\,V(x,y) \rightarrow \bigvee_{(t,t')\in\mathsf{V}} \mathtt{isA}(y,t')\right)$$

The first conjunct of the $\varnothing$ formula $\psi_T^t(x)$ verifies that the point x is labeled by no other tile than t. The second part, instead, ensures that the points y on the right or above of x are labeled by some tile t' which is compatible with t, *w.r.t.* the constraints imposed by the horizontal H and vertical V relations, respectively.

At this point, we can focus on the formula $\psi_G(x)$ defined as follows:

$$\psi_G(x) \doteq (\exists y \,.\, H(x,y)) \wedge (\exists y \,.\, V(x,y)) \wedge \gamma(x).$$

The first two conjuncts guarantee the existence of an horizontal and vertical adjacent of the point x, while the subformula $\gamma(x)$, whose definition depends on the considered fragment of $\mathtt{SCL}$, needs to enforce the fact that x is the origin of a square. That is, that going horizontally and then vertically or, vice versa, vertically and then horizontally the same point is reached. In order to do this, we make use of the two $\mathtt{S}$ path formulas $\pi_{HV}(x,y) \doteq \exists z \,.\, (H(x,z) \wedge V(z,y))$ and $\pi_{VH}(x,y) \doteq \exists z \,.\, (V(x,z) \wedge H(z,y))$. In some cases, we also use the $\mathtt{S\,A}$ path formula $\pi_D(x,y) \doteq \pi_{HV}(x,y) \vee \pi_{VH}(x,y)$ combining the previous ones. We now proceed by a case analysis on the specific fragments.

- **[S O]** By assuming the existence of a non-empty relation D connecting a point with its opposite in the square, *i.e.*, the diagonal point, we can say that all points reachable through π_{HV} or π_{VH} are, actually, the same unique point:

$$\begin{aligned}\gamma(x) \doteq\ & \exists y \,.\, D(x,y) \\ & \wedge\ \forall y,z.\, \pi_{HV}(x,y) \wedge D(x,z) \rightarrow y \leq z\ \wedge\ \forall y,z.\, \pi_{HV}(x,y) \wedge D(x,z) \rightarrow y \geq z \\ & \wedge\ \forall y,z.\, \pi_{VH}(x,y) \wedge D(x,z) \rightarrow y \leq z\ \wedge\ \forall y,z.\, \pi_{VH}(x,y) \wedge D(x,z) \rightarrow y \geq z.\end{aligned}$$

  The $\mathtt{S\,O}$ formula $\gamma(x)$ ensures that relation D is non-empty and functional and that all points reachable via π_{HV} or π_{VH} are necessarily the one reachable through D.
- **[S A C]** By applying a counting quantifier to the formula π_D encoding the union of the points reachable through π_{HV} or π_{VH}, we can ensure the existence of a single diagonal point: $\gamma(x) \doteq \neg\, \exists^{\geq 2} y \,.\, \pi_D(x,y)$.
- **[S E C]** As for the $\mathtt{S\,O}$ fragment, here we use a diagonal relation D, which needs to contain all and only the points reachable via π_{HV} or π_{VH}. By means of the counting quantifier, we ensure its functionality:

$$\gamma(x) \doteq \neg\, \exists^{\geq 2} y.\, D(x,y) \wedge \forall y.\pi_{HV}(x,y) \leftrightarrow D(x,y) \wedge \forall y.\pi_{VH}(x,y) \leftrightarrow D(x,y).$$

- **[S E O']** This case is similar to the previous one, where the functionality of D is obtained by means of the $\mathtt{O}$ construct:

$$\begin{aligned}\gamma(x) \doteq\ & \forall y,z \,.\, D(x,y) \wedge D(x,z) \rightarrow y \leq z \\ & \wedge\ \forall y \,.\, \pi_{HV}(x,y) \leftrightarrow D(x,y)\ \wedge\ \forall y \,.\, \pi_{VH}(x,y) \leftrightarrow D(x,y).\end{aligned}$$

- **[S Z A E]** This proof is inspired by the one used for the undecidability of the guarded fragment extended with transitive closure [13]. This time, the functionality of the diagonal relation D is indirectly ensured by the conjunction of the four formulas $\gamma_1(x)$, $\gamma_2(x)$, $\gamma_3(x)$, and $\gamma_4(x)$ that exploit all the features of the fragment:

$$\gamma(x) \doteq \gamma_1(x) \wedge \gamma_2(x) \wedge \gamma_3(x) \wedge \gamma_4(x) \wedge \forall y \,.\, \pi_D(x,y) \leftrightarrow D(x,y), \text{ where}$$

$$\gamma_1(x) \doteq \forall y\,.\,\left(\bigvee_{i\in\{0,1\}} D_i(x,y)\right) \leftrightarrow D(x,y),$$

$$\gamma_2(x) \doteq \left(\bigvee_{i\in\{0,1\}} \neg\,\exists\, y.\, D_i(x,y)\right) \wedge \left(\bigwedge_{i\in\{0,1\}} \forall y.\, D_i(x,y) \rightarrow \exists\, z.\, D_{1-i}(y,z)\right),$$

$$\gamma_3(x) \doteq \bigwedge_{i\in\{0,1\}} \forall y\,.\,\left(x = y \vee D_i(x,y) \vee D_i^{-}(x,y)\right) \leftrightarrow E_i(x,y), \text{ and}$$

$$\gamma_4(x) \doteq \bigwedge_{i\in\{0,1\}} \forall y\,.\,(\exists\, z.(E_i(x,z) \wedge E_i(z,y))) \leftrightarrow E_i(x,y).$$

Intuitively, γ_1 asserts that D is the union of the two accessory relations D_0 and D_1, while γ_2 guarantees that a point can only have adjacents *w.r.t.* just one relation D_i and that these adjacents can only appear as first argument of the opposite relation D_{1-i}. In addition, γ_3 ensures that the additional relation E_i is the reflexive symmetric closure of D_i and γ_4 forces E_i to be transitive as well.
We can now prove that the relation D is functional. Suppose by contradiction that this is not case, *i.e.*, there exist values a, b, and c in the domain of the model of the sentence φ, with $b \neq c$ such that both $D(a,b)$ and $D(a,c)$ hold true. By the formula γ_1 and the first conjunct of γ_2, we have that $D_i(a,b)$ and $D_i(a,c)$ hold for exactly one index $i \in \{0,1\}$. Thanks to the full γ_2, we surely know that $a \neq b$, $a \neq c$, and neither $D_i(b,c)$ nor $D_i(c,b)$ can hold. Indeed, if $a = b$ then $D_i(a,a)$. This in turn implies $D_{1-i}(a,d)$ for some value d due to the second conjunct of γ_2. Hence, there would be pairs with the same first element in both relations, trivially violating the first conjunct of γ_2. Similarly, if $D_i(b,c)$ holds, then $D_{1-i}(c,d)$ needs to hold as well, for some value d, leading again to a contradiction. Now, by the formula γ_3, both $E_i(b,a)$ and $E_i(a,c)$ hold, but $E_i(b,c)$ does not. However, this clearly contradicts γ_4. As a consequence, D is necessarily functional.

Now, it is not hard to see that the above sentence φ (one for each fragment) is satisfiable if and only if the domino instance on which the reduction is based on is solvable. □

5 Conclusion

In this paper we define and study the decision problems of satisfiability and containment for SHACL documents and shape constraints. In order to do so we introduce translations between SHACL and `SCL`, a fragment of FOL extended with counting quantifiers and a transitive closure operator. Using these translations we lay out a map of SHACL fragments for which we are able to prove undecidability or decidability along with complexity results, for the satisfiability and containment problems. We also expose semantic properties and asymmetries within SHACL which might inform a future update of the specification. The satisfiability and containment problems are undecidable for the full SHACL specification. However, decidability can be achieved by restricting the usage of certain SHACL components, such as cardinality restrictions over property shapes or property paths. Nevertheless, the decidability of some fragments of SHACL remains an open question, worthy of further investigation.

References

1. A. Amarilli and M. Benedikt and P. Bourhis and M. Vanden Boom: Query Answering with Transitive and Linear-Ordered Data. In: IJCAI'16. pp. 893–899 (2016)
2. Acar, E., Benerecetti, M., Mogavero, F.: Satisfiability in Strategy Logic can be Easier than Model Checking. In: AAAI'19. pp. 2638–2645 (2019)
3. Andresel, M., Corman, J., Ortiz, M., Reutter, J.L., Savkovic, O., Simkus, M.: Stable Model Semantics for Recursive SHACL. In: Proceedings of The Web Conference 2020. p. 1570–1580. WWW '20 (2020)
4. Baader, F., Calvanese, D., McGuinness, D., Nardim, D., Patel-Scheider, P.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
5. Berger, R.: The Undecidability of the Domino Problem. MAMS **66**, 1–72 (1966)
6. ten Cate, B., Segoufin, L.: Unary Negation. In: STACS'11. pp. 344–355. LIPIcs 9, Leibniz-Zentrum fuer Informatik (2011)
7. ten Cate, B., Segoufin, L.: Unary Negation. LMCS **9**(3), 1–46 (2013)
8. Corman, J., Florenzano, F., Reutter, J.L., Savković, O.: Validating SHACL Constraints over a Sparql Endpoint. In: The Semantic Web – ISWC 2019. pp. 145–163 (2019)
9. Corman, J., Reutter, J.L., Savković, O.: Semantics and Validation of Recursive SHACL. In: The Semantic Web – ISWC 2018. pp. 318–336 (2018)
10. Cyganiak, R., Wood, D., Markus Lanthaler, G.: RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, W3C (2014), http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/
11. Danielski, D., Kieronski, E.: Finite Satisfiability of Unary Negation Fragment with Transitivity. In: MFCS'19. pp. 17:1–15. LIPIcs 138, Leibniz-Zentrum fuer Informatik (2019)
12. Donini, F., Massacci, F.: ExpTime Tableaux for $\mathcal{ALC}$. AI **124**(1), 87–138 (2000)
13. Grädel, E.: On The Restraining Power of Guards. JSL **64**(4), 1719–1742 (1999)
14. Grädel, E., Kolaitis, P., Vardi, M.: On the Decision Problem for Two-Variable First-Order Logic. BSL **3**(1), 53–69 (1997)
15. Grädel, E., Otto, M., Rosen, E.: Two-Variable Logic with Counting is Decidable. In: LICS'97. pp. 306–317. IEEECS (1997)
16. Jung, J., Lutz, C., Martel, M., Schneider, T.: Querying the Unary Negation Fragment with Regular Path Expressions. In: ICDT'18. pp. 15:1–18. OpenProceedings.org (2018)
17. Knublauch, H., Kontokostas, D.: Shapes constraint language (SHACL). W3C Recommendation, W3C (2017), https://www.w3.org/TR/shacl/
18. Lutz, C.: An Improved NExpTime-Hardness Result for Description Logic $\mathcal{ALC}$ Extended with Inverse Roles, Nominals, and Counting. Tech. Rep. 05-05, Dresden University of Technology, Dresden, Germany (2005)
19. Mortimer, M.: On Languages with Two Variables. MLQ **21**(1), 135–140 (1975)
20. Pratt-Hartmann, I.: Complexity of the Two-Variable Fragment with Counting Quantifiers. JLLI **14**(3), 369–395 (2005)
21. Pratt-Hartmann, I.: The Two-Variable Fragment with Counting Revisited. In: WOLLIC'10. pp. 42–54. LNCS 6188 (2010)
22. Robinson, R.: Undecidability and Nonperiodicity for Tilings of the Plane. IM **12**, 177–209 (1971)
23. Sattler, U., Vardi, M.: The Hybrid μ-Calculus. In: IJCAR'01. pp. 76–91. LNCS 2083 (2001)
24. Schild, K.: A Correspondence Theory for Terminological Logics: Preliminary Report. In: IJCAI'91. pp. 466–471 (1991)
25. Tobies, S.: The Complexity of Reasoning with Cardinality Restrictions and Nominals in Expressive Description Logics. JAIR **12**, 199–217 (2000)
26. Wang, H.: Proving Theorems by Pattern Recognition II. BSTJ **40**, 1–41 (1961)

A Translation from SHACL into SCL Grammar

We present our translation $\tau(M)$ from a SHACL document M (a set of SHACL shape definitions) into our SCL grammar. The translation into SCL grammar of a document M containing a set M^S of shapes can be defined as: $\bigwedge_{s \in M^S} \tau(s)$, where $\tau(s)$ is the translation of a single SHACL shape s. Notice that M contains an element for each shape name occurring in M. If M contains a shape name s that does not have a corresponding shape definition, M^S will include the empty shape definition $s{:}\langle\{\},\{\}\rangle$. Given a shape $s{:}\langle t, d\rangle$, its translation $\tau(s{:}\langle t,d\rangle)$ is defined in Table 1, where its constraint definition $\tau_d(x)$ equals $\tau(x,\mathtt{s})$. Note that we do not discuss implicit class-based targets, as they just represent a syntactic variant of class targets. In the reminder of this section we define how to compute $\tau(x,\mathtt{s})$.

As convention, we use c as an arbitrary constant and C as an arbitrary list of constants. We use s, $\mathtt{s}'$ and $\mathtt{s}''$ as shape names, and $\bar{S}$ as a list of shape names. Variables are defined as x, y and z. Arbitrary paths are identified with r.

The translation of the constraints of a shape $\tau(x,\mathtt{s})$ is defined in two cases as follows. The first case deals with the property shapes, which must have exactly one value for the sh:path property. The second case deals with node shapes, which cannot have any value for the sh:path property.

$$\tau(x,\mathtt{s}) = \top \wedge \begin{cases} \bigwedge_{\forall\langle \mathtt{s},y,z\rangle \in M} \tau_2(x,r,\langle \mathtt{s},y,z\rangle) & \text{if } \exists\, r.\langle \mathtt{s},\mathtt{sh{:}path},r\rangle) \in M \\ \bigwedge_{\forall\langle \mathtt{s},y,z\rangle \in M} \tau_1(x,\langle \mathtt{s},y,z\rangle) & \text{otherwise} \end{cases}$$

This translation is based on the following translations of node shapes triples, property shape triples and property paths.

A.1 Translation of Node Shape Triples

The translation of $\tau_1(x,\langle \mathtt{s},y,z\rangle)$ is split in the following cases, depending on the predicate of the triple. In case none of those cases are matched $\tau_1(x,\langle \mathtt{s},y,z\rangle) \doteq \top$. The latter ensures that any triple not directly described in the cases below does not alter the truth value of the conjunction in the definition of $\tau(x,\mathtt{s})$.

- $\tau_1(x,\langle \mathtt{s},\mathtt{sh{:}hasValue},\mathtt{c}\rangle) \doteq x = \mathtt{c}$.
- $\tau_1(x,\langle \mathtt{s},\mathtt{sh{:}in},C\rangle) \doteq \bigvee_{\mathtt{c}\in C} x = \mathtt{c}$.
- $\tau_1(x,\langle \mathtt{s},\mathtt{sh{:}class},\mathtt{c}\rangle) \doteq \exists y.\mathtt{isA}(x,y) \wedge y = \mathtt{c}$.
- $\tau_1(x,\langle \mathtt{s},\mathtt{sh{:}datatype},\mathtt{c}\rangle)) \doteq \mathrm{F}^{\mathrm{dt}=\mathtt{c}}(x)$.
- $\tau_1(x,\langle \mathtt{s},\mathtt{sh{:}nodeKind},\mathtt{c}\rangle) \doteq \mathrm{F}^{\mathrm{IRI}}(x)$ if $c =$sh:IRI; $\mathrm{F}^{\mathrm{literal}}(x)$ if $c =$sh:Literal; $\mathrm{F}^{\mathrm{blank}}(x)$ if $c =$sh:BlankNode. The translations for a c that equals sh:BlankNodeOrIRI, sh:BlankNodeOrLiteral or sh:IRIOrLiteral are trivially constructed by a conjunction of two of these three filters.
- $\tau_1(x,\langle \mathtt{s},\mathtt{sh{:}minExclusive},\mathtt{c}\rangle) \doteq x > \mathtt{c}$ if order is an interpreted relation, else $\mathrm{F}^{>\mathtt{c}}(x)$.
- $\tau_1(x,\langle \mathtt{s},\mathtt{sh{:}minInclusive},\mathtt{c}\rangle) \doteq x \geq \mathtt{c}$ if order is an interpreted relation, else $\mathrm{F}^{\geq\mathtt{c}}(x)$.

- $\tau_1(x, \langle \mathtt{s}, \mathtt{sh{:}maxExclusive}, \mathtt{c} \rangle) \doteq x < \mathtt{c}$ if order is an interpreted relation, else $\mathrm{F}^{<\mathtt{c}}(x)$.
- $\tau_1(x, \langle \mathtt{s}, \mathtt{sh{:}maxInclusive}, \mathtt{c} \rangle) \doteq x \leq \mathtt{c}$ if order is an interpreted relation, else $\mathrm{F}^{\leq \mathtt{c}}(x)$.
- $\tau_1(x, \langle \mathtt{s}, \mathtt{sh{:}maxLength}, \mathtt{c} \rangle) \doteq \mathrm{F}^{\text{maxLength}=\mathtt{c}}(x)$.
- $\tau_1(x, \langle \mathtt{s}, \mathtt{sh{:}minLength}, \mathtt{c} \rangle) \doteq \mathrm{F}^{\text{minLength}=\mathtt{c}}(x)$.
- $\tau_1(x, \langle \mathtt{s}, \mathtt{sh{:}pattern}, \mathtt{c} \rangle) \doteq \mathrm{F}^{\text{pattern}=\mathtt{c}}(x)$.
- $\tau_1(x, \langle \mathtt{s}, \mathtt{sh{:}languageIn}, C \rangle) \doteq \bigvee_{\mathtt{c} \in C} F^{\text{languageTag} = \mathtt{c}}(x)$.
- $\tau_1(x, \langle \mathtt{s}, \mathtt{sh{:}not}, \mathtt{s}' \rangle) \doteq \neg \mathtt{hasShape}(x, \mathtt{s}')$.
- $\tau_1(x, \langle \mathtt{s}, \mathtt{sh{:}and}, \bar{S} \rangle) \doteq \bigwedge_{\mathtt{s}' \in \bar{S}} \mathtt{hasShape}(x, \mathtt{s}')$.
- $\tau_1(x, \langle \mathtt{s}, \mathtt{sh{:}or}, \bar{S} \rangle) \doteq \bigvee_{\mathtt{s}' \in \bar{S}} \mathtt{hasShape}(x, \mathtt{s}')$.
- $\tau_1(x, \langle \mathtt{s}, \mathtt{sh{:}xone}, \bar{S} \rangle) \doteq \bigvee_{\mathtt{s}' \in \bar{S}} (\mathtt{hasShape}(x, \mathtt{s}') \wedge \bigwedge_{\mathtt{s}'' \in \bar{S} \setminus \{\mathtt{s}'\}} \neg \mathtt{hasShape}(x, \mathtt{s}''))$.
- $\tau_1(x, \langle \mathtt{s}, \mathtt{sh{:}node}, \mathtt{s}' \rangle) \doteq \mathtt{hasShape}(x, \mathtt{s}')$.
- $\tau_1(x, \langle \mathtt{s}, \mathtt{sh{:}property}, \mathtt{s}' \rangle) \doteq \mathtt{hasShape}(x, \mathtt{s}')$.

A.2 Translation of Property Shapes

The translation of $\tau_2(x, r, \langle \mathtt{s}, y, z \rangle)$ is split in the following cases, depending on the predicate of the triple. In case none of those cases are matched $\tau_2(x, r, \langle \mathtt{s}, y, z \rangle) \doteq \top$.

- $\tau_2(x, r, \langle \mathtt{s}, \mathtt{sh{:}hasValue}, \mathtt{c} \rangle) \doteq \exists y.r(x, y) \wedge \tau_1(y, \langle \mathtt{s}, \mathtt{sh{:}hasValue}, \mathtt{c} \rangle)$
- $\tau_2(x, r, \langle \mathtt{s}, p, \mathtt{c} \rangle) \doteq \forall y.\tau_3(x, r, y)) \rightarrow \tau_1(y, \langle \mathtt{s}, p, \mathtt{c} \rangle)$, if p equal to one of the following: `sh:class`, `sh:datatype`, `sh:nodeKind`, `sh:minExclusive`, `sh:minInclusive`, `sh:maxExclusive`, `sh:maxInclusive`, `sh:maxLength`, `sh:minLength`, `sh:pattern`, `sh:not`, `sh:and`, `sh:or`, `sh:xone`, `sh:node`, `sh:property`, `sh:in` .
- $\tau_2(x, r, \langle \mathtt{s}, \mathtt{sh{:}languageIn}, C \rangle) \doteq \forall y.\tau_3(x, r, y)) \rightarrow \tau_1(y, \langle \mathtt{s}, \mathtt{sh{:}languageIn}, C \rangle)$.
- $\tau_2(x, r, \langle \mathtt{s}, \mathtt{sh{:}uniqueLang}, \text{true} \rangle) \doteq \bigwedge_{\mathtt{c} \in L} \neg \exists^{\geq 2} y.r(x, y) \wedge F^{\text{lang}=c}(y)$ where $L = \{\mathtt{c} \mid \mathtt{c} \in C \wedge \exists \mathtt{s}'.\langle \mathtt{s}', \mathtt{sh{:}languageIn}, C \rangle \in M\}$. This translation is possible because `sh:languageIn` is the only constraint that can force language tags constraints on literals.
- $\tau_2(x, r, \langle \mathtt{s}, \mathtt{sh{:}minCount}, \mathtt{c} \rangle) \doteq \exists^{\geq \mathtt{c}} y.\tau_3(x, r, y))$.
- $\tau_2(x, r, \langle \mathtt{s}, \mathtt{sh{:}maxCount}, \mathtt{c} \rangle) \doteq \neg \exists^{\leq \mathtt{c}} y.\tau_3(x, r, y))$.
- $\tau_2(x, r, \langle \mathtt{s}, \mathtt{sh{:}equals}, \mathtt{c} \rangle) \doteq \forall y.\tau_3(x, r, y) \leftrightarrow \tau_3(x, \mathtt{c}, y)$.
- $\tau_2(x, r, \langle \mathtt{s}, \mathtt{sh{:}disjoint}, \mathtt{c} \rangle) \doteq \neg \exists y.\tau_3(x, r, y) \wedge \tau_3(x, \mathtt{c}, y)$.
- $\tau_2(x, r, \langle \mathtt{s}, \mathtt{sh{:}lessThan}, \mathtt{c} \rangle) \doteq \forall y, z\,.\,\tau_3(x, r, y) \wedge \tau_3(x, \mathtt{c}, z) \rightarrow y < z$.
- $\tau_2(x, r, \langle \mathtt{s}, \mathtt{sh{:}lessThanOrEquals}, \mathtt{c} \rangle) \doteq \forall y, z\,.\,\tau_3(x, r, y) \wedge \tau_3(x, \mathtt{c}, z) \rightarrow y \leq z$.
- $\tau_2(x, r, \langle \mathtt{s}, \mathtt{sh{:}qualifiedValueShape}, \mathtt{s}' \rangle) \doteq \alpha \wedge \beta$, where α and β are defined as follows. Let S' be the set of *sibling shapes* of $\mathtt{s}$ if M contains $\langle \mathtt{s}, \mathtt{sh{:}qualifiedValueShapesDisjoint}, \text{true} \rangle$, or the empty set otherwise. Let $\nu(x) = \mathtt{hasShape}(x, \mathtt{s}') \bigwedge_{\forall\, \mathtt{s}'' \in S'} \neg \mathtt{hasShape}(x, \mathtt{s}'')$. If M contains the triple

$\langle \mathtt{s}, \mathtt{sh{:}qualifiedMinCount}, \mathtt{c} \rangle$, then α is equal to $\exists^{\geq \mathtt{c}}\, y.\tau_3(x,r,y) \wedge \nu(x)$, otherwise α is equal to $\top$. If M contains the triple $\langle \mathtt{s}, \mathtt{sh{:}qualifiedMaxCount}, \mathtt{c} \rangle$, then β is equal to $\neg\, \exists^{\leq \mathtt{c}}\, y.\tau_3(x,r,y) \wedge \nu(x)$, otherwise β is equal to $\top$.

- $\tau_2(x, r, \langle \mathtt{s}, \mathtt{sh{:}close}, \text{true} \rangle) \doteq \bigwedge_{\forall R \in \Theta} \neg\, \exists\, y.R(x,y)$ if Θ is not empty, where Θ is defined as follows. Let Θ^{all} be the set of all relation names in M, namely $\Theta^{\text{all}} = \{R \mid \langle x, R, y \rangle \in M\}$. If this FOL translation is used to compare multiple SHACL documents, such in the case of deciding containment, then Θ^{all} must be extended to contain all the relation names in all these SHACL documents. Let Θ^{declared} be the set of all the binary property names $\Theta^{\text{declared}} = \{R \mid \{\langle \mathtt{s}, \mathtt{sh{:}property}, x \rangle \wedge \langle x, \mathtt{sh{:}path}, R \rangle)\} \subseteq M\}$. Let Θ^{ignored} be the set of all the binary property names declared as "ignored" properties, namely $\Theta^{\text{ignored}} = \{R \mid R \in \bar{R} \wedge \langle \mathtt{s}, \mathtt{sh{:}ignoredProperties}, \bar{R} \rangle \in M\}$, where $\bar{R}$ is a list of IRIs. The set Θ can now be defined as $\Theta = \Theta^{\text{all}} \setminus (\Theta^{\text{declared}} \cup \Theta^{\text{ignored}})$.

A.3 Translation of Property Paths

The translation $\tau_3(x,r,y))$ of any SHACL path r is given by the following cases. For simplicity, we will assume that all property paths have been translated into an equivalent form having only simple IRIs within the scope of the inverse operator. Using SPARQL syntax for brevity, where the inverse operator is identified by the hat symbol ^, the sequence path $\hat{}(r_1/r_2)$ can be simplified into $\hat{}r_2/\hat{r}_1$; an alternate path $\hat{}(r_1 \mid r_2)$ can be simplified into $\hat{}r_2 \mid \hat{}r_1$. We can simplify in a similar way zero-or-more, one-or-more and zero-or-one paths $\hat{}(r^{*/+/?})$ into $(\hat{}r)^{*/+/?}$.

- If r is an IRI R, then $\tau_3(x,r,y)) \doteq R(x,y)$
- If r is an inverse path, with $r =$ "`[ sh:inversePath` R `]`", then $\tau_3(x,r,y)) \doteq R^{-}(x,y)$
- If r is a conjunction of paths, with $r =$ "`(` r_1`,` r_2`, ...,` r_n `)`", then $\tau_3(x,r,y)) \doteq \exists\, z_1, z_2, ..., z_{n-1}.\tau_3(x,r_1,z_1)) \wedge \tau_3(z_1,r_2,z_2)) \wedge ... \wedge \tau_3(z_{n-1},r_2,y))$
- If r is a disjunction of paths, with $r =$ "`[ sh:alternativePath (` r_1`,` r_2`, ...,` r_n `) ]`", then $\tau_3(x,r,y)) \doteq \tau_3(x,r_1,y)) \vee \tau_3(x,r_2,y)) \vee ... \vee \tau_3(x,r_n,y))$
- If r is a zero-or-more path, with $r =$ "`[ sh:zeroOrMorePath` r_1`]`", then $\tau_3(x,r,y)) \doteq (\tau_3(x,r_1,y)))^*$
- If r is a one-or-more path, with $r =$ "`[ sh:oneOrMorePath` r_1`]`", then $\tau_3(x,r,y)) \doteq \exists\, z.\tau_3(x,r_1,z)) \wedge (\tau_3(z,r_1,y)))^*$
- If r is a zero-or-one path, with $r =$ "`[ sh:zeroOrOnePath` r_1`]`", then $\tau_3(x,r,y)) \doteq x = y \vee \tau_3(x,r_1,y))$

B Translation from SCL Grammar into SHACL

We present here the translation μ, inverse of τ, to translate a sentence in the SCL grammar into a SHACL document. We begin by defining the translation of the property path subgrammar $r(x,y)$ into SHACL property paths:

- $\mu(R) \doteq R$

- $\mu(R^-) \doteq$ [sh:inversePath R]
- $\mu(r^\star(x,y)) \doteq$
  [sh:zeroOrMorePath $\mu(r(x,y))$]
- $\mu(x = y \vee r(x,y)) \doteq$
  [sh:zeroOrOnePath $\mu(r(x,y))$]
- $\mu(r_1(x,y) \vee r_2(x,y)) \doteq$
  [sh:alternativePath ($\mu(r_1(x,y))$, $\mu(r_2(x,y))$])
- $\mu(r_1(x,y) \wedge r_2(x,y)) \doteq$
  ($\mu(r_1(x,y))$, $\mu(r_2(x,y))$)

The translation of the constraint subgrammar $\psi(x)$ is the following. we will use $\mu(\psi(x))$ to denote the SHACL translation of shape $\psi(x)$, and $\iota(\mu(\psi(x)))$ to denote its shape IRI. To improve legibility, we omit set brackets around sets of RDF triples, and we represent them in Turtle syntax. For example, a set of RDF triples such as "s a sh:NodeShape ; sh:hasValue c . " is to be interpreted as the set $\{\langle s, \texttt{rdf:type}, \texttt{sh:NodeShape}\rangle, \langle s, \texttt{sh:hasValue}, c\rangle\}$.

- $\mu(\top) \doteq$
  s a sh:NodeShape .
- $\mu(x = \texttt{c}) \doteq$
  s a sh:NodeShape ;
    sh:hasValue c .
- $\mu(F(x)) \doteq$
  s a sh:NodeShape ;
    f c .
  Predicate f is the filter function identified by F, namely one of the following: sh:datatype, sh:nodeKind, sh:minExclusive, sh:minInclusive, sh:maxExclusive, sh:maxInclusive, sh:maxLength, sh:minLength, sh:pattern, sh:languageIn.
- $\mu(\texttt{hasShape}(x, \texttt{s}')) \doteq$
  s a sh:NodeShape ;
    sh:node s′ .
  if s′ is a node shape, else:
  s a sh:NodeShape ;
    sh:property s′ .
- $\mu(\neg\psi(x)) \doteq$
  s a sh:NodeShape ;
    sh:not $\iota(\mu(\psi(x)))$.
- $\mu(\psi_1(x) \wedge \psi_2(x)) \doteq$
  s a sh:NodeShape ;
    sh:and ($\iota(\mu(\psi_1(x)))$, $\iota(\mu(\psi_2(x)))$) .
- $\mu(\exists^{\geq n} y.r(x,y) \wedge \psi(x)) \doteq$
  s a sh:NodeShape ;
    sh:property [
      sh:path $\mu(r(x,y))$;
      sh:qualifiedValueShape $\iota(\mu(\psi(x)))$;

```
      sh:qualifiedMinCount n ;
    ] .
```

– $\mu(\forall y.r(x,y) \leftrightarrow R(x,y)) \doteq$

```
s a sh:NodeShape ;
  sh:property [ ;
    sh:path μ(r(x,y)) ;
    sh:equals R ;
  ] .
```

– $\mu(\neg \exists y.r(x,y) \wedge R(x,y)) \doteq$

```
s a sh:NodeShape ;
  sh:property [ ;
    sh:path μ(r(x,y)) ;
    sh:disjoint R ;
  ] .
```

– $\mu(\forall y,z.r(x,y) \wedge R(x,z) \rightarrow y < z) \doteq$

```
s a sh:NodeShape ;
  sh:property [ ;
    sh:path μ(r(x,y)) ;
    sh:lessThan R ;
  ] .
```

– $\mu(\forall y,z.r(x,y) \wedge R(x,z) \rightarrow y \leq z) \doteq$

```
s a sh:NodeShape ;
  sh:property [ ;
    sh:path μ(r(x,y)) ;
    sh:lessThanOrEquals R ;
  ] .
```

We can now define the translation $\mu(\varphi)$ of a complete sentence of the φ-grammar into a SHACL document M (effectively a set of RDF triples) as follows.

– $\mu(\varphi_1 \wedge \varphi_2) \doteq \mu(\varphi_1) \cup \mu(\varphi_2))$
– $\mu(\psi_1(\mathtt{c})) \doteq \mu(\psi_1(x)) \cup$

```
s a sh:NodeShape ;
  sh:targetNode c;
  sh:node ι(μ(ψ1(x))) .
```

– $\mu(\forall x.\ \mathtt{isA}(x,\mathtt{c}) \rightarrow \psi_1(x)) \doteq \mu(\psi_1(x)) \cup$

```
s a sh:NodeShape ;
  sh:targetClass c;
  sh:node ι(μ(ψ1(x))) .
```

– $\mu(\forall x,y.\ R(x,y) \rightarrow \psi_1(x)) \doteq \mu(\psi_1(x)) \cup$

```
s a sh:NodeShape ;
  sh:targetSubjectsOf R ;
  sh:node ι(μ(ψ1(x))) .
```

– $\mu(\forall x,y.\ R^{-}(x,y) \rightarrow \psi_1(x)) \doteq \mu(\psi_1(x)) \cup$

```
s a sh:NodeShape ;
  sh:targetObjectsOf R ;
  sh:node ι(μ(ψ1(x))) .
```

– $\mu(\forall x.\, \texttt{hasShape}(x, \texttt{s}) \leftrightarrow \psi(x)) \doteq \mu(\psi_1(x)) \cup$
  s `a sh:NodeShape ;`
    `sh:node` $\iota(\mu(\psi_1(x)))$ `.`

C Additional Proof

Proof of Theorem 9. Similarly to the use of the `C` construct of `SCL`, a simple combination of few instances of the `O` feature allows to write the following sentence φ encoding the existence of an injective function that is not surjective. In more detail, a weaker version of the role of the counting quantifier is played here by the `O'` construct that enforces the functionality of the two relations F and G. In addition, by applying the `O` construct twice between the inverse of F and G, we are able to ensures that F^- is functional as well. Hence, the thesis easily follows.

$$
\begin{aligned}
\varphi &\doteq \texttt{isA}(0, \texttt{c}) \wedge \neg\, \exists x\,.\, F^-(0, x) \wedge \forall x\,.\, \texttt{isA}(x, \texttt{c}) \rightarrow \psi(x);\\
\psi(x) &\doteq \exists y\,.\, (F(x, y) \wedge \texttt{isA}(y, \texttt{c}))\\
&\quad \wedge\ \forall y, z\,.\, F(x, y) \wedge F(x, z) \rightarrow y \leq z \wedge \forall y, z\,.\, G(x, y) \wedge G(x, z) \rightarrow y \leq z\\
&\quad \wedge\ \forall y, z\,.\, F^-(x, y) \wedge G(x, z) \rightarrow y \leq z \wedge \forall y, z\,.\, F^-(x, y) \wedge G(x, z) \rightarrow z \leq y.
\end{aligned}
$$

To prove that `E O'` fragment does not enjoy the finite-model property too, it is enough to replace the last to applications of the `O` feature with the `E` formula $\forall y\,.\, F^-(x, y) \leftrightarrow G(x, y)$, which ensures the functionality of F^-, being G functional. □