Satisfiability and Containment of Recursive SHACL

Paolo Pareti[a,*], George Konstantinidis[b], Fabio Mogavero[c]

[a] *University of Winchester, Sparkford Road, SO22 4NR, Winchester, United Kingdom*
[b] *University of Southampton, University Road, SO17 1BJ, Southampton, United Kingdom*
[c] *Universita degli Studi di Napoli Federico II, Corso Umberto I 40, 80138, Napoli, Italy*

Abstract

The *Shapes Constraint Language* (SHACL) is the recent W3C recommendation language for validating RDF data, by verifying certain *shapes* on graphs. Previous work has largely focused on the validation problem and the standard decision problems of *satisfiability* and *containment*, crucial for design and optimisation purposes, have only been investigated for simplified versions of SHACL. Moreover, the SHACL specification does not define the semantics of recursively-defined constraints, which led to several alternative *recursive* semantics being proposed in the literature. The interaction between these different semantics and important decision problems has not been investigated yet. In this article we provide a comprehensive study of the different features of SHACL, by providing a translation to a new first-order language, called SCL, that precisely captures the semantics of SHACL. We also present MSCL, a second-order extension of SCL, which allows us to define, in a single formal logic framework, the main recursive semantics of SHACL. Within this language we also provide an effective treatment of filter constraints which are often neglected in the related literature. Using this logic we provide a detailed map of (un)decidability and complexity results for the satisfiability and containment decision problems for different SHACL fragments. Notably, we prove that both problems are undecidable for the full language, but we present decidable combinations of interesting features, even in the face of recursion.

1. Introduction

Data validation is the process of ensuring data is clean, correct, and useful. The *Shapes Constraint Language* (SHACL, for short) [17] is a recent W3C recommendation language for validation of data in the form of RDF graphs [9] and is quickly becoming the established technology. Similar to ontology languages like OWL [30], SHACL can be seen as a language that strictly imposes a schema on graph data models, such as RDF, which are inherently schemaless. Unlike ontology languages, SHACL focuses more on the structural properties of a graph rather than the semantic ones, and it is not intended for inference. A SHACL *shape graph*, which we will call SHACL *document* in this

[*] Corresponding author.
Email addresses: `paolo.pareti@winchester.ac.uk` (Paolo Pareti), `g.konstantinidis@soton.ac.uk` (George Konstantinidis), `fabio.mogavero@unina.it` (Fabio Mogavero)

paper, validates an RDF graph by evaluating it against a set of constraints. In SHACL, constraints are modelled as a set of *shapes* which, intuitively, define the structure that certain entities in the graph must conform to.

Despite its ongoing widespread adoption many aspects of SHACL remain unexplored. On the one hand, several important theoretical properties of the language have not been studied. Among these are the decidability and complexity of different problems, including satisfiability and containment of SHACL documents. These problems are the main focus of this work. On the other hand, the W3C specification does not define the semantics of SHACL in its full generality, since it does not describe how to handle recursive constraints. Recent work [8] has suggested a theoretical modelling of the language in order to formally define a recursive semantics; the same work also studied the complexity of the validation problem. Alternative recursive semantics for SHACL have been further suggested in [2].

In this article, we extend [22] to capture SHACL semantics using mathematical logic. This is an important contribution on its own, as it offers a standard and well-established modelling of the language, where SHACL documents are translated into logical sentences that are interpreted in the usual way. This makes SHACL semantics easier to understand and study compared to existing approaches that rely on auxiliary *ad hoc* constructs and functions. In particular, [8] defines validation based on the existence of an assignment of SHACL shapes to data nodes. This assignment captures which shapes are satisfied/violated by which nodes, while at at same time the target nodes of the validation process are verified. As [8] argues, in the face of SHACL recursion one may consider *partial assignments*, where the truth value of a constraint at some nodes may be left unknown. In addition, [2] identifies two major ways, called *brave* and *cautious* validation, to verify the target nodes during the validation process. Deciding between partial or total assignments, and between brave or cautious validations gives rise to four different semantics for SHACL, each with its own definition of validation. Using our logical approach we are able to capture all four semantics in a clear and uniform way, providing for a better understanding of SHACL features and taking advantage of the rich field of computational logic.

Our contributions are the following:

- We prove that all four major semantics of SHACL coincide for non-recursive documents and that partial-assignment semantics reduces to the total-assignment one for all SHACL documents. (Section 3)

- We formalise non-recursive SHACL semantics by translating to a novel fragment of first-order logic extended with counting quantifiers and a transitive closure operator; we call this logic SCL for *Shapes Constraint Logic*. The provided translation from SHACL to SCL is actually an one-to-one correspondence between these languages and we have identified eight prominent SHACL features that translate to particular restrictions of SCL. In effect, SCL is the logical counterpart of SHACL. (Section 4)

- We extend SCL into a fragment of monadic second-order logic, called MSCL, that intuitively allows us to impose conditions over the space of all possible assignments and captures all four major recursive SHACL semantics. In particular, we reduce SHACL satisfiability and containment under all semantics to the MSCL satisfiability problem. We also demonstrate how our logical framework generalises previous languages designed to model SHACL. (Section 5)

- We pay particular attention to SHACL filters (*e.g.*, constraints on the value of particular elementary datatypes), which have not been previously addressed in the literature, and provide a corresponding axiomatisation in MSCL (Section 6).
- Finally, we turn our focus to SCL satisfiability, which corresponds to existential MSCL and can express several of SHACL decision problems. In particular, we study the finite/unrestricted satisfiability and containment problems for non-recursive documents and the finite/unrestricted satisfiability for recursive SHACL under brave semantics. We explore the interaction of the main language features we have identified and create a detailed map of decidability and complexity results for many interesting fragments. In general, satisfiability and containment for the the full logic are undecidable. However, the base language has an ExpTime-complete satisfiability and containment problem. (Section 7).

2. Preliminaries

With the term *graph* we implicitly refer to a set of *triples*, where each single triple $\langle \mathtt{s}, \mathtt{p}, \mathtt{o} \rangle$ identifies an edge with label $\mathtt{p}$, called *predicate*, from a node $\mathtt{s}$, called *subject*, to a node $\mathtt{o}$, called *object*. Graphs in this article are represented in *Turtle syntax* [5] using common XML namespaces, such as sh to refer to SHACL terms. Usually, in the *RDF data model* [9], subjects, predicates, and objects are defined over different but overlapping domains. For example, while *IRIs* can occupy any position in an RDF triple, *literals* (representing datatype values) can only appear in the object position. These differences are not central to the problem discussed in this article, and thus, for the sake of simplicity, we will assume that all elements of a triple are drawn from a single and infinite domain. This assumption actually corresponds to what is known in the literature as *generalised RDF* [9]. We model triples as binary relations in Fol, *i.e.*, we write the atom $R(s, o)$ as a shorthand for the tuple $\langle \mathtt{s}, \mathtt{R}, \mathtt{o} \rangle$, and call R a graph relation name. We use a minus sign to identify the *inverse* role, *i.e.*, we write $R^-(s, o)$ in place of $R(o, s)$. We also consider the distinguished binary relation name isA to represent class membership triples, that is, we write $\langle \mathtt{s}, \mathtt{rdf{:}type}, \mathtt{o} \rangle$ as $\mathtt{isA}(s, o)$.

3. Shapes Constraint Language: SHACL

In this section we describe the *Shapes Constraint Language* (SHACL), a W3C language to define formal constraints for the validation of RDF graphs [17]. Firstly, we introduce the main elements of its syntax, and explain the role they play in the validation process. We then discuss *assignments* [8], that is, mappings that allow us to capture which nodes in a graph satisfy or violate which constraints. Assignments have been used to formally define SHACL semantics and this can non-ambiguously happen for the non-recursive case. For recursive SHACL, the specification leaves the semantics of recursive constraints open for interpretation, and there have been more than one ways to extend the assignments-based semantics for this. We review and discuss the four major *extended* semantics that have been proposed in the literature to handle recursive constraints. Notably, in the absence of recursion, we show the collapse of all four extended semantics into the same one. We also show that two of these extended semantics can be considered a special case of the other two, by proving a reduction from *partial assignment* to *total assignment* semantics (defined later in this section). Having formalised SHACL semantics, we define the satisfiability and containment decision problems for SHACL documents.

3.1. SHACL *Syntax*

Data validation in SHACL requires two inputs: (1) an RDF graph G to be validated and (2) a SHACL document M that defines the conditions against which G must be validated. The SHACL specification defines the output of the data validation process as a *validation report*, detailing all the violations of the conditions set by M that were found in G. If the violation report contains no violations, a graph G is *valid w.r.t.* a SHACL document M. Determining whether a graph is valid *w.r.t.* a SHACL document is the decision problem called *validation.*

A SHACL *document* is a set of *shapes.* Shapes essentially restrict the structure that a valid graph should have, by defining a set of constraints that are evaluated against a set of nodes, known as the *target* nodes. Formally, a shape is a tuple $\langle \mathtt{s}, t, d\rangle$ defined by three components: (1) a shape name $\mathtt{s}$, which uniquely identifies the shape; (2) a *target definition* t which is a set of *target declarations*; each target declaration can be represented by a unary query and identifies the RDF nodes that must satisfy the constraints d; (3) a set of *constraints* which are used in conjunction, and hence hereafter referred to as the single constraint d. The SHACL specification defines several types of constraints, called *constraint components.* The `sh:datatype` component, for example, constraints an RDF term to be an RDF literal of a particular datatype. Without loss of generality, we assume that shape names in a SHACL document do not occur in other SHACL documents or graphs. As we formally define later, a graph is valid *w.r.t.* a document whenever all constraints of all shapes in the document are satisfied by the target nodes of the corresponding shapes.

It is worth noting that one type of SHACL target declaration might reference specific nodes to be validated that do not actually appear in the graph under consideration. Given a document M and a graph G, we denote by $\mathsf{nodes}(\mathsf{G}, \mathsf{M})$ the set of nodes in G together with those referenced by the node target declarations in M. In the absence of a document, we use $\mathsf{nodes}(\mathsf{G})$ to denote the nodes of a graph G. With $\mathsf{shapes}(\mathsf{M})$ we refer to all the shape names in a document M. When it is clear for the context, we might use a shape name $\mathtt{s}$ either to refer to the name itself or to the entire shape tuple.

Constraints can use the name of a shape as a short-hand to refer to the constraints of that shape. We call this a *shape reference.* Let S^d_0 be the set of all the shape names occurring in a constraint d of a shape $\langle \mathtt{s}, t, d\rangle$; these are the *directly-referenced* shapes of $\mathtt{s}$. Let S^d_{i+1} be the set of shapes in S^d_i union the directly-referenced shapes of the constraints of the shapes in S^d_i. A shape $\langle \mathtt{s}, t, d\rangle$ is *recursive* if $\mathtt{s} \in S^d_\infty$. A SHACL document M is said to be *recursive* if it contains a recursive shape, and *non-recursive* otherwise. For simplicity, all SHACL documents we consider in this work do not contain the `sh:xone` constraint over shape references, which models the logical operator of exclusive or. Any SHACL document, in fact, can be linearly transformed into an equivalent document that does not contain the `sh:xone` operator using a standard logical transformation. The intuition behind this transformation is that an `sh:xone` defined over shapes $\mathtt{s}_1$ to $\mathtt{s}_n$ is equivalent to an `sh:xone` between two shapes $\mathtt{s}_n$ and $\mathtt{s}_k$, where $\mathtt{s}_k$ is a fresh shape whose constraint is the `sh:xone` of shapes $\mathtt{s}_1$ to $\mathtt{s}_{n-1}$. Then, any exclusive or between two shapes can be linearly transformed into an equivalent expression that uses only conjunctions, disjunctions, and the negation operators.

3.2. Semantics of Non-Recursive SHACL

A target declaration t is a unary query over a graph G. We denote with $G \models t(n)$ that a node n is *in the target* of t *w.r.t.* a graph G. The target declaration t might be empty, in which case no node is in the target of t. To formally discuss about nodes satisfying the constraints of a shape we need to introduce the concept of *assignments* [8]. Intuitively, an assignment is used to keep track, for any RDF node, of all the shapes whose constraints the node satisfies and all of those that it does not.

Definition 1. *Given a graph G, and a* SHACL *document M, an* assignment *σ for G and M is a function mapping nodes in $\mathsf{nodes(G, M)}$, to subsets of shape literals in $\mathsf{shapes(M)} \cup \{\neg \mathsf{s} | \mathsf{s} \in \mathsf{shapes(M)}\}$, such that for all nodes n and shape names s, $\sigma(n)$ does not contain both s and $\neg\mathsf{s}$.*

Notice that given a document and a graph, an assignment does not have to associate all graph nodes to all document shapes or their negations. In fact, there might exist node-shape pairs (n, s) for which neither $\mathsf{s} \in \sigma(n)$ nor $\neg\mathsf{s} \in \sigma(n)$. This is the reason why sometimes assignments are called *partial assignments*, as opposed to *total assignments* which have to associate all nodes with all shape names or their negation.

Definition 2. *An assignment σ is* total w.r.t. *a graph G and a* SHACL *document M if, for all nodes n in $\mathsf{nodes(G, M)}$ and shapes $\langle \mathsf{s}, t, d\rangle$ in M, either $\mathsf{s} \in \sigma(n)$ or $\neg\mathsf{s} \in \sigma(n)$.*

For any graph G and SHACL document M, we denote with $A^{G,M}$ and $A_T^{G,M}$, respectively, the set of assignments, and the set of total assignments for G and M. Trivially, $A_T^{G,M} \subseteq A^{G,M}$ holds.

When trying to determine whether a node n of a graph G satisfies a constraint d of a shape, the outcome does not only depend on d, n, and G, but it might also depend, due to shape references, on whether other nodes satisfy the constraints of other shapes. This latter fact can be encoded in an assignment σ. The authors of [8], therefore, define the evaluation or *conformance* of a node n to a constraint d *w.r.t.* a graph G under an assignment σ as $[\![d]\!]^{n,G,\sigma}$. This expression can take one of the three truth values of Kleene's logic: True, False, or $\mathsf{Undefined}$. If $[\![d]\!]^{n,G,\sigma}$ is True (*resp.*, False) we say that node n *conforms* (*resp.*, does not conform) to constraint d *w.r.t.* G under σ. Intuitively, the evaluation of $[\![d]\!]^{n,G,\sigma}$ can be split into two parts (further details can be found in [8]): the first verifies conditions on G, such as the existence of certain triples. The second part examines other node-shape pairs that d itself is listing for conformance and, instead of triggering subsequent evaluation, checks whether their conformance is correctly encoded in σ. Since – in general and for arbitrary SHACL documents that might be recursive – σ is partial, it might be that $[\![d]\!]^{n,G,\sigma}$ is $\mathsf{Undefined}$. Even then, this might not affect the outcome of the graph validation process (see Section 3.3 for an example). Graph validation depends on the existence of an assignment such that even if it is $\mathsf{Undefined}$ for certain nodes, at least is consistent (as defined below) and is True for all target nodes on the constraints of the shapes that describe these nodes as targets. Such assignments are known as *faithful* assignments [8]. Note that, as we show in Lemma 2, for non-recursive documents there is a unique faithful assignment which is total and for which $\mathsf{Undefined}$ conformance never appears.

Definition 3. *For all graphs G and* SHACL *documents M, an assignment σ is* faithful w.r.t. *G and M, denoted by $(G, \sigma) \models M$, if the following two conditions hold true for any shape $\langle \mathsf{s}, t, d\rangle$ in $\mathsf{shapes(M)}$ and node n in $\mathsf{nodes(G, M)}$:*

```
: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 .
```

$\sigma(\texttt{:Alex}) =$
$\{\texttt{:studentShape},$
$\neg\texttt{:disjFacultyShape}\},$
$\sigma(\texttt{:Jane}) = \sigma(\texttt{:CS}) =$
$\sigma(\texttt{:Student}) =$
$\{\neg\texttt{:studentShape},$
$\texttt{:disjFacultyShape}\}.$

Figure 1: A SHACL document (left), a graph that validates it (centre), and a faithful assignment for this graph and document (right).

(1) $s \in \sigma(n)$ iff $[\![d]\!]^{n,G,\sigma}$ *is* True *and* $\neg s \in \sigma(n)$ iff $[\![d]\!]^{n,G,\sigma}$ *is* False*;*

(2) if $G \models t(n)$ *then* $s \in \sigma(n)$*.*

Intuitively, condition (1) ensures that the evaluation described by the assignment is indeed correct; while condition (2) ensures that the assignment agrees with the target definitions. The existence of a faithful assignment is a necessary and sufficient condition for validation of non-recursive SHACL documents [8].

Definition 4. *A graph G is valid* w.r.t. *a non-recursive* SHACL *document M if there exists an assignment σ such that $(G, \sigma) \models M$.*

An example SHACL document is shown in Figure 1. This example captures the requirement that 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 is valid with respects to these shapes is provided in Figure 1, along with a faithful assignment for this graph. The graph can be made invalid by changing the faculty of `:Jane` in the last triple to a different value.

As we will see later, the existence of a faithful assignment is also a necessary condition for all other semantics that allow recursion. For those cases, however, we will want to consider additional assignments where the first property of Definition 3 holds, but not necessarily the second, *i.e.*, assignments that agree with the constraint definitions, but not necessarily the target definitions of the shapes. In order to do this, we will remove the targets from a document and look for faithful assignments against the new document, since condition (2) of Definition 3 is trivially satisfied for SHACL documents where all target definitions are empty. Let $M^{\setminus t}$ denote the SHACL document obtained from substituting all target definitions in SHACL document M with the empty set. Then, the following lemma is immediate:

Lemma 1. *For all graphs G,* SHACL *documents M and assignments σ, condition (1) from Definition 3 holds for any shape s in* $\mathsf{shapes(M)}$ *and node n in* $\mathsf{nodes(G, M)}$ iff $(G, \sigma) \models M^{\setminus t}$.

For non-recursive SHACL documents, the next lemma states that for any graph, there exists a unique faithful total assignment for $M^{\setminus t}$ and, if there is a faithful assignment for M, then this must be it.

Lemma 2. *For all graphs G and non-recursive* SHACL *documents M, there exists a unique assignment ρ in $A_T^{G,M}$ such that $(G, \rho) \models M^{\setminus t}$, and for every assignment σ in $A^{G,M}$ such that $(G, \sigma) \models M$, then $\sigma = \rho$.*

Proof. If M is non recursive, then there exists a non empty subset M' of M that only contains shapes whose constraints do not use shape references. Intuitively, the constraints of the shapes in M' can be evaluated directly on any graph, independently of any assignment. Shape references are the only part of the evaluation of a constraint that depends on the assignment σ, and that could introduce the truth value Undefined under three-valued logic [8]. Thus, for all graphs G, assignments σ, nodes n and constraints c in of a shape in M', it holds that the evaluation of $[\![d]\!]^{n,G,\sigma}$ (1) does not depend on σ and (2) has a Boolean truth value. It is easy to see that properties (1) and (2) also hold for the document M'' which contains the shapes of M whose shape references (if any) only reference shapes in M'. This reasoning can be extended inductively to prove that properties (1) and (2) hold for all the shapes of M. Point (1) ensures that there cannot be more than one assignment such that $(G, \sigma) \models M^{\setminus t}$, while point (2) ensures that such an assignment is total. This assignment σ exists and it can be computed iteratively as follows. Let σ' be the assignment for M' such that for any shape $\langle \mathsf{s}, t, d \rangle$ in M' and node n, $\mathsf{s} \in \sigma'(n)$, if $[\![d]\!]^{n,G,\emptyset}$ is True, and $\neg \mathsf{s} \in \sigma'(n)$, otherwise. Then let σ'' be the assignment for M'' such that for any shape $\langle \mathsf{s}, t, d \rangle$ in M'' and node n, $\mathsf{s} \in \sigma''(n)$, if $[\![d]\!]^{n,G,\sigma'}$ is True, and $\neg \mathsf{s} \in \sigma''(n)$, otherwise. This process is repeated until the assignment σ, defined over all of the shapes of M, is computed. Notice that for all graphs G, SHACL documents M and assignments σ, fact $(G, \sigma) \models M$ implies $(G, \sigma) \models M^{\setminus t}$. Thus the existence of an assignment ρ' different than ρ such that $(G, \rho') \models M$, is in contradiction with the fact that there cannot be more than one assignment that if faithful for G and $M^{\setminus t}$. □

3.3. Semantics of Full SHACL

As mentioned, the semantics of recursive shape definitions in SHACL documents has been left undefined in the original W3C SHACL specification [17] and this gives rise to several possible interpretations. In this work, we consider previously introduced extended semantics of SHACL that define how to interpret recursive SHACL documents. These can be characterised by two dimensions, namely the choice between (1) *partial* and *total* assignments [8] and (2) between *brave* and *cautious* validation [2], which we will subsequently formally introduce. Together, these two dimensions result in the four extended semantics studied in this article, namely *brave-partial*, *brave-total*, *cautious-partial* and *cautious-total*. We do not consider the less obvious dimension of *stable-model* semantics [2], which relates to non-monotone reasoning in logic programs.

The first extended semantics that we consider coincides with Definition 4. That is, the existence of a faithful assignment can be directly used as a semantics for recursive documents as well. Nevertheless, in this case the assignment is not necessarily total, as is in the case of non-recursive documents, proven in Lemma 2. To stress this (as well as the "brave" nature of the semantics discussed later), we call this the brave-partial semantics.

Definition 5. *A graph G is valid* w.r.t. *a* SHACL *document M under* brave-partial semantics *if there exists an assignment $\sigma \in A^{G,M}$ such that $(G, \sigma) \models M$.*

The other three extended semantics are defined by adding further conditions to the one just introduced. To motivate those, first consider an example of a recursive document and of a non-total faithful assignment that evaluates the conformance of some nodes against some constraints to Undefined. This happens when recursion makes it impossible for a node n to either conform or not to conform to a shape $\mathtt{s}$ but, at the same time, validity does not depend on whether n conforms to shape $\mathtt{s}$ or not. Consider, for instance, the following SHACL document, containing the single shape $\langle \mathtt{s}^*, \emptyset, d^* \rangle$ defined as follows:

```
:InconsistentS a sh:NodeShape ;
  sh:not :InconsistentS .
```

This shape is defined as the negation of itself, that is, given a node n, a graph G and an assignment σ, fact $[\![d^*]\!]^{n,G,\sigma}$ is True *iff* $\neg\mathtt{s}^* \in \sigma(n)$, and False *iff* $\mathtt{s}^* \in \sigma(n)$. It is easy to see that any assignment that maps a node to either $\{\mathtt{s}^*\}$ or $\{\neg\mathtt{s}^*\}$ is not faithful, as it would violate condition (1) of Definition 3. However, an assignment that maps every node of a graph to the empty set would be faithful for that graph and document $\{\mathtt{s}^*\}$. Intuitively, this means that nodes in the graph cannot conform nor not conform to shape $\mathtt{s}^*$, but this should not be interpreted as a violation of any constraint, since this shape does not have any target node to validate. In effect, conformance for all nodes to the constraint of $\{\mathtt{s}^*\}$ is left as Undefined, but the existence of a faithful assignment makes any graph valid *w.r.t.* to $\{\mathtt{s}^*\}$.

In the W3C SHACL specification, where recursion semantics was left open to interpretation, nodes can either conform to, or not conform to a given shape, and the concept of an “undefined” level of conformance is arguably alien to the specification. It is natural, therefore, to consider restricting the evaluation of a constraint to the True and False values of Boolean logic. This is achieved by restricting assignments to be total.

Definition 6. *A graph G is valid* w.r.t. *a* SHACL *document M under* brave-total semantics *if there exists a total assignment $\sigma \in A_T^{G,M}$ such that $(G, \sigma) \models M$.*

Since total assignments are a more specific type of assignments, if a graph G is valid *w.r.t.* a SHACL document M under brave-total semantics, than it is also valid *w.r.t.* M under brave-partial semantics. The converse, instead, is only true for non-recursive SHACL documents. In fact, as we show later on, all extended semantics coincide, for non-recursive SHACL documents. Note also that there is no obvious preferable choice for the semantics of recursive documents. For example, while total assignments can be seen as a more natural way of interpreting the SHACL specification, they are not without issues of their own. Going back to our previous example, we can notice that there cannot exist a total faithful assignment for the SHACL document containing shape `:InconsistentS`, for any non-empty graph. This is a trivial consequence of the fact that no node can conform to, nor not conform to, shape `:InconsistentS`. In this example, however, brave-total semantics conflicts with the SHACL specification, since the latter implies that a SHACL document without target declarations in any of its shapes (such as the one in our example) should trivially validate any graph. If there are no target declarations, in fact, there are no target nodes on which to verify the conformance of certain shapes, and thus no violations of constraints should be detected.

Another dimension in the choices for extended semantics studied in literature [2] is the difference between brave and cautious validation of recursive documents. When a SHACL document M is recursive, there might exist multiple assignments satisfying property (1) of Definition 3, that is, multiple σ for which $(G, \sigma) \models M^{\setminus t}$. Intuitively, these can be seen as equally "correct" assignments with respect to the constraints of the shapes, and brave validation only checks whether at least one of them is compatible with the target definitions of the shapes. Cautious validation, instead, represents a stronger form of validation, where all such assignments must be compatible with the target definitions.

Definition 7. *A graph G is valid* w.r.t. *a* SHACL *document M under* cautious-partial *(*resp.*,* cautious-total*) semantics if it is (1) valid under* brave-partial *(*resp.*,* brave-total*) semantics and (2) for all assignments σ in $A^{G,M}$* (resp.*, $A_T^{G,M}$), it is true that if $(G, \sigma) \models M^{\setminus t}$ holds then $(G, \sigma) \models M$ holds as well.*

To exemplify this distinction, consider the following SHACL document M_1.

```
:VegDishShape a sh:PropertyShape ;
  sh:targetNode :DailySpecial ;
  sh:path :hasIngredient ;
  sh:minCount 1 ;
  sh:qualifiedMaxCount 0 ;
  sh:qualifiedValueShape [ sh: not :VegIngredientShape ] .

:VegIngredientShape a sh:PropertyShape ;
  sh:path [ sh:inversePath :hasIngredient ] ;
  sh:node :VegDishShape .
```

This document requires the daily special of a restaurant, node `:DailySpecial`, to be vegetarian, that is, to conform to shape `:VegDishShape`. This shape is recursively defined as follows. Something is a vegetarian dish if it contains an ingredient, and all of its ingredients are vegetarian, that is, entities conforming to the `:VegIngredientShape`. A vegetarian ingredient, in turn, is an ingredient of at least one vegetarian dish. Consider now a graph G_1 containing only the following triple.

```
:DailySpecial :hasIngredient :Chicken .
```

Due to the recursive definition of `:VegDishShape`, there exist two different assignments σ_1 and σ_2, which are both faithful for G_1 and $M_1^{\setminus t}$. In σ_1, no node in G_1 conforms to any shape, while σ_2 differs from σ_1 in that node `:DailySpecial` conforms to `:VegDishShape` and node `:Chicken` conforms to `:VegIngredientShape`. Essentially, either both the dish and the ingredient from graph G_1 are vegetarian, or neither is. Therefore, σ_2 is faithful for G_1 and M_1, while σ_1 is not. The question of whether the daily special is a vegetarian dish or not can be approached with different levels of "caution". Under brave validation, graph G_1 is valid *w.r.t.* M_1, since it is possible that the daily special is vegetarian. Cautious validation, instead, takes the more conservative approach, and under its definition G_1 is not valid *w.r.t.* by M_1, since it is also possible that the daily special is not vegetarian.

For each extended semantics, the definition of validity of a graph G *w.r.t.* a SHACL document M, denoted by $G \models M$, is summarised in the following list, and schematised in Table 1.

brave-partial there is an assignment that is faithful *w.r.t.* G and M;

brave-total there is an assignment that is total and faithful *w.r.t.* G and M;

	Brave	Cautious
Partial	$\exists\,\sigma.\ (G,\sigma) \models M$	$\exists\,\sigma.\ (G,\sigma) \models M$ and $\forall\,\sigma.$ if $(G,\sigma) \models M^{\setminus t}$, then $(G,\sigma) \models M$
Total	$\exists\,\rho.\ (G,\rho) \models M$	$\exists\,\rho.\ (G,\rho) \models M$ and $\forall\,\rho.$ if $(G,\rho) \models M^{\setminus t}$, then $(G,\rho) \models M$

Table 1: Definition of validity (from Definitions 5, 6 and 7) of a graph G under a SHACL document M ($G \models M$) *w.r.t.* the two dimensions of extended semantics considered in this article, where $\sigma \in A^{G,M}$ and $\rho \in A_T^{G,M}$.

cautious-partial there is an assignment that is faithful *w.r.t.* G and M, and every assignment that is faithful *w.r.t.* G and $M^{\setminus t}$ is also faithful *w.r.t.* G and M.

cautious-total there is an assignment that is total and faithful *w.r.t.* G and M, and every assignment that is total and faithful *w.r.t.* G and $M^{\setminus t}$ is also faithful *w.r.t.* G and M.

We now prove that, when considering only non-recursive SHACL documents, these four semantics are necessarily equivalent to each other, since the semantics of non-recursive SHACL documents is uniquely determined. The formalisation of this equivalence given in the next theorem is essentially a consequence of Lemma 2.

Theorem 1. *For any graph G, non-recursive* SHACL *document M, and extended semantics α and β, it holds that $G \models M$ under α* iff *$G \models M$ under β.*

Proof. Since $A_T^{G,M} \subseteq A^{G,M}$, for any graph G and SHACL document M, the definition of validity of *cautious-total* trivially subsumes the one of *brave-total* and *cautious-partial* which, in turn, subsumes the one of *brave-partial*. Notice that for all graphs G, SHACL documents M and assignments σ, if $\sigma \in A_T^{G,M^{\setminus t}}$, then $\sigma \in A_T^{G,M}$. From Lemma 2 we also know that a faithful assignment for M and G is necessarily total, and it is the same unique assignment that is faithful for $M^{\setminus t}$ and G. Thus, for non-recursive documents, the definition of validity of *brave-partial* subsumes the one of *cautious-total*, and consequently the four extended semantics are equivalent. $\square$

Given a notion of validity from Table 1, corresponding to one of the four extended semantics, we can define the following decision problems, which we study in the rest of the article.

- **SHACL Satisfiability**: A SHACL document M is satisfiable *iff* there exists a graph G such that $G \models M$.
- **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 $G \models M_2$.

Obviously, the more meaningful satisfiability problem is one on finite graphs.

- **SHACL Finite Model Property**: A SHACL document M enjoys the finite model property if whenever it is satisfiable it is so on a finite graph.

3.4. From Partial to Total Assignments

In this subsection we show that in order to study the theoretical properties of SHACL one can focus on total assignments semantics only, as partial assignment semantics can be seen as a special case of total. Thus, in the rest of this article we will focus on total assignments semantics without loss of generality. In particular, any SHACL document M can be linearly transformed into another document M^* such that a graph G is valid *w.r.t.* M under brave-partial, or cautious-partial, *iff* G is valid *w.r.t.* M^* under brave-total or cautious-total, respectively. Intuitively, this is achieved by splitting each shape s into two shapes s^+ and s^-, evaluated under total assignments semantics, such that the constraints of s^+ and s^- model the evaluation to True and False, respectively, of the constraints of s, and such that the evaluation to $\mathsf{Undefined}$ of the constraints of s correspond to the negation of the constraints of both s^+ and s^-.

In the following, we formalise the just discussed transformation by means of a function Γ over SHACL documents. With a slight abuse of notation, we use $\neg$ and $\wedge$ to denote, respectively, the negated form of a SHACL constraint, and the conjunction of two SHACL constraints. We also denote $\mathsf{s}(x)$ the constraint requiring node x to conform to shape s. We use s^+ and s^- to denote two unique fresh shape names, which are a function of s.

Definition 8. *Given a* SHACL *document M, document $\Gamma(M)$ contains shapes $\langle \mathsf{s}^+, t, \gamma(d)\rangle$ and $\langle \mathsf{s}^-, t, \gamma(\neg d)\rangle$ for every shape $\langle \mathsf{s}, t, d\rangle$ in M, such that, for every constraint d, the corresponding constraint $\gamma(d)$ is constructed by replacing, for every shape s, every occurrence of the negated atom "$\neg \mathsf{s}(x)$" in d with "$\neg \mathsf{s}^+(x) \wedge \mathsf{s}^-(x)$" and every occurrence of the non-negated atom "$\mathsf{s}(x)$" in d with "$\mathsf{s}^+(x) \wedge \neg \mathsf{s}^-(x)$".*

Definition 9. *Given an assignment σ, let σ^γ be the assignment such that for every node n the following holds: $\sigma^\gamma(n) = \{\mathsf{s}^+, \neg \mathsf{s}^- | \mathsf{s} \in \sigma(n)\} \cup \{\neg \mathsf{s}^+, \mathsf{s}^- | \neg \mathsf{s} \in \sigma(n)\} \cup \{\neg \mathsf{s}^+, \neg \mathsf{s}^- | \mathsf{s}, \neg \mathsf{s} \notin \sigma(n)\}$.*

We can observe that for any SHACL document M, graph G and assignment σ for M and G, assignment σ^γ is a total assignment for $\Gamma(M)$ and G. Also, it is easy to see that the complexity of the transformation $\Gamma(M)$ is linear in the size of the original document M.

Lemma 3. *Given a* SHACL *document M, a graph G, an assignment σ, and a node n, the following hold:*

- $[\![d]\!]^{n,G,\sigma}$ *is* True iff $[\![\gamma(d)]\!]^{n,G,\sigma^\gamma}$ *is* True*;*
- $[\![d]\!]^{n,G,\sigma}$ *is* False iff $[\![\gamma(\neg d)]\!]^{n,G,\sigma^\gamma}$ *is* True*;*
- $[\![d]\!]^{n,G,\sigma}$ *is* $\mathsf{Undefined}$ iff *both* $[\![\gamma(d)]\!]^{n,G,\sigma^\gamma}$ *and* $[\![\gamma(\neg d)]\!]^{n,G,\sigma^\gamma}$ *are* False*.*

Proof. Negation in SHACL is defined in the standard way, and therefore $[\![d]\!]^{n,G,\sigma}$ is True *iff* $[\![\neg d]\!]^{n,G,\sigma}$ is False. Since $[\![d]\!]^{n,G,\sigma}$ is False *iff* $[\![\neg d]\!]^{n,G,\sigma}$ is True, proof of the first statement of the lemma is also proof of the second. We can also notice that the third statement of the lemma necessarily follows from the first two. Thus the entire lemma can be proved by proving just the first statement. To prove the first item, we show the following two implications, separately:

($\Rightarrow$): if $[\![d]\!]^{n,G,\sigma}$ is True, then $[\![\gamma(d)]\!]^{n,G,\sigma^\gamma}$ is True;

($\Leftarrow$): if $[\![\gamma(d)]\!]^{n,G,\sigma^\gamma}$ is True, then $[\![d]\!]^{n,G,\sigma}$ is True.

In Kleene's 3-valued logic, the evaluation of a sentence into True or False implies that this evaluation does not depend on any of its sub-sentences that are evaluated to $\mathsf{Undefined}$ (*i.e.*, changing the truth value of one such sub-sentence would not affect the truth value of the whole sentence). Notice also that the only atoms that can be evaluated as $\mathsf{Undefined}$ are shape references $s(x)$ [8]. This means that if the 3-valued evaluation of a constraint d over a node, a graph and an assignment is True (*resp.*, False), then this evaluation would still be True (*resp.*, False), if every shape atom $s(x)$ that evaluates to $\mathsf{Undefined}$ evaluates to False instead.

($\Rightarrow$) If $[\![d]\!]^{n,G,\sigma}$ evaluates to True, then $[\![\gamma(d)]\!]^{n,G,\sigma^\gamma}$ must also evaluate to True, since in the transformation from d to $\gamma(d)$ (1) every constraint that is not a shape reference remains unchanged, and (2) every shape reference (in d) is transformed into a conjunction of shape references (in $\gamma(d)$) that still evaluates to the same truth value of the original expression, unless this truth value is $\mathsf{Undefined}$. However, by our previous observation, changing an $\mathsf{Undefined}$ truth value cannot affect the truth value of $[\![\gamma(d)]\!]^{n,G,\sigma^\gamma}$ since $[\![d]\!]^{n,G,\sigma}$ evaluates to True. Thus implication ($\Rightarrow$) holds.

($\Leftarrow$) Similarly, if $[\![\gamma(d)]\!]^{n,G,\sigma^\gamma}$ evaluates to True, then $[\![d]\!]^{n,G,\sigma}$ must also evaluate to True, since, in the inverse transformation from $\gamma(d)$ to d: (1) every constraint that is not a shape reference remains unchanged, and (2) every pair of shape references "$\mathsf{s}^+(x) \wedge \neg \mathsf{s}^-(x)$" or "$\neg \mathsf{s}^+(x) \wedge \mathsf{s}^-(x)$" is transformed into a single shape reference which either (a) evaluates to the same truth value, or (b) evaluates to the truth value of $\mathsf{Undefined}$ when the original constraint evaluates to False. Notice that in SHACL, the constraints of a shape are considered in conjunction, and negation only appears in front of shape references. Since $[\![\gamma(d)]\!]^{n,G,\sigma^\gamma}$ evaluates to True, a pair of shape references "$\mathsf{s}^+(x) \wedge \neg \mathsf{s}^-(x)$" or "$\neg \mathsf{s}^+(x) \wedge \mathsf{s}^-(x)$" that evaluates to False *w.r.t.* n, G and σ^γ can only appear in a disjunction in $\gamma(d)$ of which at least one disjunct evaluates to True *w.r.t.* n, G and σ^γ, since this disjunction cannot be within the scope of negation. Pairs of shape references "$\mathsf{s}^+(x) \wedge \neg \mathsf{s}^-(x)$" or "$\neg \mathsf{s}^+(x) \wedge \mathsf{s}^-(x)$" that evaluate to False *w.r.t.* n, G and σ^γ, therefore, do not affect the truth value of $[\![\gamma(d)]\!]^{n,G,\sigma^\gamma}$. Thus implication ($\Leftarrow$) holds as well. $\square$

From Definition 8 and Lemma 3 the main theorem of this subsection easily follows.

Theorem 2. *Given a* SHACL *document M and a graph G, it holds that G is valid* w.r.t. *M under brave-partial (*resp.*, cautious-partial) semantics* iff *G is valid* w.r.t. $\Gamma(M)$ *under brave-total (*resp.*, cautious-total) semantics.*

Thus, in the rest of the article we will only focus on total assignments and we shall use the term *brave semantics* to refer to brave-total and *cautious semantics* to refer to cautious-total.

4. Shapes Constraint Logic: SCL

In this section we provide a precise formalisation of SHACL semantics and related decision problems in a formal logical system. For the sake of simplicity of presentation,

Types of target declarations in t	SCL **target axiom**
Node target (node c)	$\Sigma_{\mathtt{s}}(c)$
Class target (class $\mathtt{c}$)	$\forall x.\, \mathtt{isA}(x, c) \to \Sigma_{\mathtt{s}}(x)$
Subjects-of target (relation R)	$\forall x, y.\, R(x, y) \to \Sigma_{\mathtt{s}}(x)$
Objects-of target (relation R)	$\forall x, y.\, R^{-}(x, y) \to \Sigma_{\mathtt{s}}(x)$

Table 2: Translation of a SHACL shape with name $\mathtt{s}$ and target declaration t, into an SCL target axiom.

we first focus on the brave semantics only, and then show how to adapt our system to model cautious semantics (recall that, as shown in Section 3.4, partial assignments semantics is, model-theoretically, a special case of total assignments semantics). The main component of this logical system is the SCL language, a novel fragment of first-order logic extended with counting quantifiers and the transitive closure operator, that precisely models SHACL documents. We will later show the equivalidity of SHACL and SCL, by demonstrating how, for any graph, the latter can be used to model total faithful assignments.

Our decision problems, instead, are modelled using MSCL, a fragment of monadic second-order logic defined on top of SCL, by extending the latter with second-order quantifications on monadic relations. Intuitively, MSCL allows us to define conditions over the space of all possible assignments, something that cannot be expressed in SCL. Nevertheless, as we will see later, several formulations of our decision problems are fully reducible to the first-order logic satisfiability problem.

4.1. A First-Order Logic for SHACL

In the presentation of our logical system and in the analysis of its decision problems, we consider arbitrary first-order relational models with equality as the only built-in relation. When we deal with the SHACL encoding, instead, we assume the first-order models to have the set of RDF terms as the domain of discourse, plus a set of interpreted relations for the SHACL filters.

Assignments are modelled by means of a set of monadic relation names Σ, called *shape relations*. In particular, each shape $\mathtt{s}$ is associated with a unique shape relation $\Sigma_{\mathtt{s}}$. If $\Sigma_{\mathtt{s}}$ is a shape relation associated with shape $\mathtt{s}$, then fact $\Sigma_{\mathtt{s}}(x)$ (resp. $\neg\Sigma_{\mathtt{s}}(x)$) describes an assignment σ such that $\mathtt{s} \in \sigma(x)$ (resp. $\neg\mathtt{s} \in \sigma(x)$). Since our logical system uses standard Boolean logic, for any element of the domain c and shape relation Σ, it holds that $\Sigma(c) \vee \neg\Sigma(c)$ holds, by the law of excluded middle. Thus any Boolean interpretation of shape relations defines a total assignment.

Sentences and formulae in the SCL language follow the grammar reported in Definition 10, whose main syntactic components are described later on. In the rest of the article, we will focus on this logic to study the decidability and complexity of our SHACL decision problems. In particular, we are going to reserve the symbols τ and τ^{-} to denote the translations from SHACL documents into SCL sentences and *vice versa* and refer the reader to the appendix for the full details about these translations. Bold capital letters in square brackets on the right of some of the grammar production rules are pure meta-annotations for naming SCL features and, obviously, not an integral part of the syntax.

Definition 10. *The* Shape Constraint Logic *(*SCL*, for short) is the set of first-order* sentences φ *built according to the following context-free grammar, where c is a constant from the domain of RDF terms, Σ is a shape-relation name,* F *is a filter-relation name, R is a binary-relation name, Kleene's star symbol* * *indicates the transitive closure of the binary relation induced by $\pi(x, y)$, the superscript $\pm$ stands for a relation or its inverse, and $n \in \mathbb{N}$:*

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

Intuitively, sentences obtained through grammar rule φ correspond to SHACL documents. These could be empty ($\top$), a conjunction of documents, a *target axiom* (production rules 3, 4, and 5 of rule φ) or a *constraint axiom* (production rule 6 of rule φ). Target axioms take one of three forms, based on the type of target declarations in the shapes of a SHACL document. There are four types of target declarations in SHACL, namely (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). The full correspondence of SHACL target declarations to SCL target axioms is summarised in Table 2. The correspondence of a target definition containing multiple target declarations, is simply the conjunction of the corresponding target axioms.

The non terminal symbol $\psi(x)$ corresponds to the subgrammar of the SHACL constraints components. Within this subgrammar, the true symbol $\top$ identifies an empty constraint, $x = c$ a constant equivalence constraint and F a monadic filter relation (*e.g.*, $\mathtt{F}^{\mathrm{IRI}}(x)$, 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 [17]. Filters are captured by the $\mathtt{F}(x)$ production rule 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, finally, T denotes a zero-or-more path.

As usual, to enhance readability, we define the following syntactic shortcuts:

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

Figure 2: Translation of the SHACL document from Figure 1 into an SCL sentence.

- $\psi_1(x) \vee \psi_2(x) := \neg(\neg\psi_1(x) \wedge \neg\psi_2(x))$;
- $\pi(x, c) := \exists y.\pi(x, y) \wedge y = c$;
- $\forall y\,.\,\pi(x, y) \rightarrow \psi(y) := \neg\exists y\,.\,\pi(x, y) \wedge \neg\psi(y)$.

The above mentioned translations τ and τ^- between SHACL and SCL are polynomial in the size of the input and computable in polynomial time. Intuitively, as we show later in Theorem 3, a SHACL document M validates a graph G *iff* a first-order structure representing the latter satisfies the SCL sentence $\tau(M)$. *Vice versa*, every SCL sentence φ is satisfied by a first-order structure representing graph G *iff* the SHACL document $\tau^-(\varphi)$ validates G.

Another important property of these translations is that they preserve the notion of SHACL *recursion*, that is, a SHACL document M is recursive *iff* the SHACL document $\tau^-(\tau(M))$ is recursive. We will call an SCL sentence ϕ *recursive* if $\tau^-(\phi)$ is recursive.

Given a SHACL document M, the SCL sentence $\tau(M)$ contains a shape relation $\Sigma_{\texttt{s}}$ for each shape $\texttt{s}$ in M. Sentence $\tau(M)$ can be be split into constraint axioms and target axioms. Intuitively, these are used to verify the first and second condition of Definition 3, respectively. The constraint axioms of $\tau(M)$ correspond to the sentence $\tau(M^{\setminus t})$, *i.e.*, to the translation of the document ignoring targets, while the target axioms of $\tau(M)$ correspond to taking targets into account, *i.e.*, to a sentence ϕ, where $\phi \wedge \tau(M^{\setminus t})$ is $\tau(M)$.

Note that our translation τ results in a particular structure of SCL sentences, that we will call *well-formed*, and thus we restrict the inverse translation τ^- and define it only on well-formed SCL sentences. An SCL sentence φ is well-formed if, for every shape relation Σ, sentence φ contains exactly one constraint axiom with relation Σ on the left-hand side of the implication. Intuitively, this condition ensures that every shape relation is "defined" by a corresponding constraint axiom. Figure 2 shows the translation of the document from Figure 1 into a well-formed SCL sentence.

Before defining the semantic correspondence between SHACL and SCL we introduce the translations of graphs and assignments into first-order structures.

Definition 11. *Given a graph G, the first-order structure G^τ contains a fact $R(s, o)$,* i.e., *$R(s, o)$ holds true in G^τ, if $\langle \texttt{s}, \texttt{R}, \texttt{o}\rangle \in G$.*

Definition 12. *Given a total assignment σ, the first-order structure σ^τ contains fact $\Sigma_{\mathtt{s}}(n)$,* i.e., *$\Sigma_{\mathtt{s}}(n)$ holds true in σ^τ, for every node n, if $\mathtt{s} \in \sigma(n)$.*

Definition 13. *Given a graph G and a total assignment σ, the first-order structure I* induced *by G and σ is the disjoint union of structures G^τ and σ^τ. Given a first-order structure I: (1) the graph G* induced *by I is the graph that contains triple $\langle \mathtt{s}, \mathtt{R}, \mathtt{o} \rangle$ if $I \models R(s, o)$ and (2) the assignment σ induced by I is the assignment such that, for all elements of the domain n and shape relations $\Sigma_{\mathtt{s}}$, fact $\mathtt{s} \in \sigma(n)$ is true if $I \models \Sigma_{\mathtt{s}}(n)$ and $\neg\mathtt{s} \in \sigma(n)$ is true if $I \not\models \Sigma_{\mathtt{s}}(n)$.*

The semantic correspondence between SHACL and SCL is captured by the following theorem.

Theorem 3. *For all graphs G, total assignments σ and* SHACL *documents M, it is true that $(G, \sigma) \models M$* iff *$I \models \tau(M)$, where I is the first-order structure induced by G and σ. For any first-order structure I and* SCL *sentence ϕ, it is true $I \models \phi$* iff *$(G, \sigma) \models \tau^-(\phi)$, where G and σ are, respectively, the graph and assignment induced by I.*

This theorem can be proved by a tedious but straightforward structural induction over the document syntax, with an operator-by-operator analysis of the translation we provide in the appendix.

Sentences in SCL have a direct correspondence to the sentences of the grammar presented in [22]. For each non-recursive SHACL document, the differences between the sentences obtained by translating this document are purely syntactic and the two sentences are equisatisfiable. In particular, the binary relation `hasShape` of [22] is now represented instead as a set of monadic relations. For recursive SHACL documents, the grammar of Definition 10 introduces a one-to-one correspondence between SHACL target declarations/constraints, and target/constraint axioms respectively.

The sub-grammar $\psi(x)$ in Definition 10 corresponds to the grammar of SHACL constraints from [8], with the addition of filters. The grammar from [8] omits filters by assuming that their evaluation is not more computationally complex than evaluating equality. This assumption is true for validation, the main decision problem addressed in [8], but it does not hold for satisfiability and containment, as we further discuss in Section 6.

To distinguish different fragments of SCL, Table 3 lists a number of *prominent* SHACL components. The language defined without any of these constructs is our *base* language, denoted $\varnothing$. When using 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, `S A` 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 (corresponding to `E`, `D`, and `O` in SCL) force one of their two path expressions to be an atomic relation. This can result in situations where 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, by using the inverse of $<$ and $\leq$. We instead denote by `O`' the fragment where the order relations in the $\varsigma(x, y)$ subgrammar cannot be inverted. In our formal analysis of Section 7 we will consider both `O` and `O`'.

Abbr.	**Name**	SHACL **component**	SCL **expression**
`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:lessThan` `sh:lessThanOrEquals`	$x <^{\pm} y$ and $x \leq^{\pm} y$
`C`	Cardinality constraints	`sh:qualifiedValueShape` `sh:qualifiedMinCount` `sh:qualifiedMaxCount`	$\exists^{\geq n} y.\, \pi(x,y) \wedge \psi(y)$ with $n \neq 1$
`S`	Sequence paths	SHACL list	$\exists z.\, \pi(x,z) \wedge \pi(z,y)$
`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`	$(\pi(x,y))^{\star}$

Table 3: Correspondence between prominent SHACL components and SCL expressions.

4.2. A Second-Order Logic for SHACL *Decision Problems*

In order to model SHACL decision problems, we introduce the *Monadic Shape Constraint Logic* (MSCL, for short) built on top of a *second-order interpretation* of SCL sentences. A second-order interpretation of an SCL sentence ϕ is the second-order formula obtained by interpreting shape relations as free monadic second order variables. Obviously, shape relations that are under the scope of the same quantifier describe the same assignment. While SCL can be used to describe the faithfulness of a single assignment, MSCL can express properties that must be true for all possible assignments. This is necessary to model all extended semantics. As usual, disjunction and implication symbols in MSCL sentences are just syntactic shortcuts.

Definition 14. *The* Monadic Shape Constraint Logic *(*MSCL*, for short) is the set of second-order sentences built according to the following context-free grammar* Φ*, where* φ *is an* SCL *sentence and* Σ *is the second-order variable corresponding to a shape relation.*

$$\Phi := \varphi \mid \neg\Phi \mid \Phi \wedge \Phi \mid \exists\Sigma.\, \Phi \mid \forall\Sigma.\, \Phi; \qquad\qquad \varphi := \text{SCL}.$$

The $\exists$SCL *(*resp.*,* $\forall$SCL*) fragment of* MSCL *is the set of sentences obtained by the above grammar deprived of the negation and universal (*resp.*, existential) quantifier rules.*

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

MSCL **Sentence Satisfiability** An MSCL sentence Φ is satisfiable if there exists a relational structure Ω such that $\Omega \models \Phi$.

MSCL **Finite-model Property** An MSCL sentence Φ enjoys the finite-model property if, whenever Φ is satisfiable, it is so on a relational structure.

In Section 5 we discuss the correspondence between the SHACL and MSCL decision problems. In this respect, we assume that filters are interpreted relations. In particular, we prove equivalence of SHACL and MSCL, for the purpose of validity, on models that we call *canonical*; that is, models having the following properties: (1) the domain of the model is the set of RDF terms, (2) constant symbols are interpreted as themselves (as in a standard Herbrand model [12]), (3) such a model contains built-in interpreted relations for filters, and (4) ordering relations $<$ and $\leq$ are the disjoint union of the total orders of the different comparison types allowed in SPARQL. To enforce the fact that different RDF terms are not equivalent to each other we adopt the unique name assumption for the constants of our language. For the purpose of our decision problems, it is sufficient to axiomatise the inequality of all the known constants.

Finally, we state a trivial result used later on to show how to solve some of the mentioned decision problem by looking at the "simpler" SCL satisfiability and validity decision problems.[1]

Proposition 1. *An* $\exists$SCL (resp., $\forall$SCL) *sentence* $\Phi \triangleq \exists \Sigma_1 \ldots \exists \Sigma_m . \varphi$ (resp., $\Phi \triangleq \forall \Sigma_1 \ldots \forall \Sigma_m . \varphi$) *is satisfiable* (resp., *valid*) iff *the subformula* φ *interpreted as an* SCL *sentence is satisfiable* (resp., *valid*).

5. From SHACL Decision Problems to MSCL Satisfiability

The rich expressiveness of the MSCL language, defined in the previous section, allows us to formally define several decision problems. We first use this language to define the main such problems studied in this article, namely SHACL validation, satisfiability and containment. We then show how MSCL can also capture a number of related decision problems that have been proposed in the literature.

5.1. Principal Decision Problems

In this section we describe the equivalidity of MSCL and SHACL, and provide a reduction of our decision problems into MSCL satisfiability. Notably, we also show how some of them can be further reduced into $\exists$SCL. As we will see later, this last reduction can be easily translated to a reduction into first-order logic, from which we derive several decidability results.

We again focus only on total assignment semantics which subsumes partial assignment semantics. Given a second-order formula ϕ, second-order interpretation of an SCL sentence, we denote with $\exists(\phi)$, respectively $\forall(\phi)$, the MSCL sentence obtained by existentially, respectively universally, quantifying all of the shape relations of ϕ. Recall that, by construction, the assignments induced by models of an MSCL sentence are total, and that the second-order variables under the scope of the same quantifier represent a single assignment.

The following corollaries, which rely on the standard notion of modelling of a sentence by a structure, easily follow from Theorem 3 and the definitions of validity from Table 1. The first two corollaries express the equisatisfiability of MSCL and SHACL. The last four corollaries express our formalisation of the SHACL satisfiability and containment decision problems in the case of brave validation and in the case of cautious validation. Recall also that G^{τ} denotes the first-order structure induced by a graph G, and $M^{\setminus t}$ denotes the SHACL document obtained by removing all target declarations from SHACL document M, which we use to test first condition of Def. 3 in isolation from the second.

[1]The term *valid* here refers to the notion of validity in mathematical logic and model theory, not to be confused with SHACL validation.

Corollary 1 (Brave-Total Validation)**.** *A graph G is valid* w.r.t. *a* SHACL *document M under brave-total semantics if $G^\tau \models \exists(\tau(M))$.*

Corollary 2 (Cautious-Total Validation)**.** *A graph G is valid* w.r.t. *a* SHACL *document M under cautious-total semantics if $G^\tau \models \exists(\tau(M)) \wedge \forall(\tau(M^{\setminus t}) \rightarrow \tau(M))$.*

Corollary 3 (Brave-Total Satisfiability)**.** *For any* SHACL *document M, document M is (finitely) satisfiable under brave-total semantics if $\exists(\tau(M))$ is (finitely) satisfiable.*

Corollary 4 (Cautious-Total Satisfiability)**.** *For any* SHACL *document M, document M is (finitely) satisfiable under cautious-total semantics if $\exists(\tau(M)) \wedge \forall(\tau(M^{\setminus t}) \rightarrow \tau(M))$ is (finitely) satisfiable.*

Corollary 5 (Brave-Total Containment)**.** *For any pair of* SHACL *documents M_1 and M_2, document M_1 is contained in M_2 under brave-total semantics iff $\exists(\tau(M_1)) \rightarrow \exists(\tau(M_2))$ is valid, that is, iff $\exists(\tau(M_1)) \wedge \neg\exists(\tau(M_2))$ is unsatisfiable.*

Corollary 6 (Cautious-Total Containment)**.** *For any pair of* SHACL *documents M_1 and M_2, document M_1 is contained in M_2 under cautious-total semantics if*

$\Big(\exists(\tau(M_1)) \wedge \forall(\tau(M_1^{\setminus t}) \rightarrow \tau(M_1))\Big) \rightarrow \Big(\exists(\tau(M_2)) \wedge \forall(\tau(M_2^{\setminus t}) \rightarrow \tau(M_2))\Big)$

is valid, that is, iff

$\Big(\exists(\tau(M_1)) \wedge \forall(\tau(M_1^{\setminus t}) \rightarrow \tau(M_1))\Big) \wedge \neg \Big(\exists(\tau(M_2)) \wedge \forall(\tau(M_2^{\setminus t}) \rightarrow \tau(M_2))\Big)$

is unsatisfiable.

We now provide a simplified definition of containment for non-recursive SHACL documents by exploiting the properties of Lemma 2, and the fact that all extended semantics are equivalent for non-recursive SHACL.

Lemma 4. *For any pair of non-recursive* SHACL *documents M_1 and M_2 document M_1 is contained in M_2 iff $\exists(\tau(M_1)) \wedge \exists(\tau(M_2^{\setminus t}) \wedge \neg\tau(M_2))$ is not satisfiable.*

Proof. For non-recursive SHACL documents all semantics are equivalent, thus containment of two non-recursvie SHACL documents can be expressed as containment under brave-total semantics (Corollary 5), namely the unsatisfiability of $\exists(\tau(M_1)) \wedge \forall(\neg\tau(M_2))$. Notice that for all assignment σ and graphs G, if $(G, \sigma) \not\models M^{\setminus t}$ then trivially $(G, \sigma) \not\models M$, thus we can rewrite containment as the unsatisfiability of the following sentence:

$\exists(\tau(M_1)) \wedge \forall(\neg\tau(M_2^{\setminus t}) \vee \neg\tau(M_2))$,

which is trivially equivalent to the following:

$\exists(\tau(M_1)) \wedge \forall(\tau(M_2^{\setminus t}) \rightarrow \neg\tau(M_2))$ is unsatisfiable.

From Lemma 2 we know that, for any graph G, there exists an assignment σ such that $(G, \sigma) \models M^{\setminus t}$. By Theorem 3, the structure G^τ induced by any G models $\exists(\tau(M_2^{\setminus t}))$, and thus $\exists(\tau(M_2^{\setminus t}))$ is true for any model. We can therefore rewrite the containment criterion as the unsatisfiability of the following sentence:

$\exists(\tau(M_1)) \wedge \exists(\tau(M_2^{\setminus t})) \wedge \forall(\tau(M_2^{\setminus t}) \rightarrow \neg\tau(M_2))$,

which is trivially equivalent to:

$\exists(\tau(M_1)) \wedge \exists(\tau(M_2^{\setminus t}) \wedge \neg\tau(M_2)) \wedge \forall(\tau(M_2^{\setminus t}) \rightarrow \neg\tau(M_2))$.

From Lemma 2 we also know that there is only one assignment σ such that $(G, \sigma) \models M^{\setminus t}$, thus the conjunct in the for all quantification can be removed. □

From the definitions above we can notice that several decision problems are reducible to the satisfiability of ∃SCL sentences, which, as defined in Proposition 1, can be further reduced to the satisfiability of SCL. In Section 7 we will study the properties of SCL to provide decidability and complexity results for our decision problems that can be reduced to ∃SCL satisfiability, namely the satisfiability and containment of non-recursive SHACL documents, and satisfiability of (recursive) SHACL documents under brave-total (and thus also brave-partial) semantics. The remaining decision problems, namely containment for recursive SHACL documents (under any extended semantics), and satisfiability for recursive SHACL documents under cautious validation, require the expressiveness of second-order logic, and are likely undecidable even for very restrictive fragments of SHACL.

5.2. Additional Decision Problems

Our logical framework allows us to express a number of additional decision problems that shift the focus on more fine-grained objects, such as shapes and constraints. While these additional decision problems are not the focus of this article, we discuss them the sake of completeness. To better model these additional problems, we will use t_n to denote a constraint definition that targets the single node n.

Given a SHACL document M, and two shapes $\mathtt{s}$ and $\mathtt{s}'$ in M, the decision problem of *shape containment* [19] determines whether $\mathtt{s}$ is *contained* in $\mathtt{s}'$. Intuitively, this means that whenever M is used for validation, nodes conforming to $\mathtt{s}$ necessarily conform to $\mathtt{s}'$. The definition of shape containment, adapted to the notation of our article, is the following.

Definition 15. *Given a* SHACL *document M, and two shapes $\langle \mathtt{s}, t, d\rangle$ and $\langle \mathtt{s}', t', d'\rangle$ in M, $\mathtt{s}$ is* shape contained *in $\mathtt{s}'$ under brave-partial (resp. brave-total) semantics if, for all graphs G, nodes n in $\mathsf{nodes}(\mathsf{G}, \emptyset)$ and assignments σ in $A^{G,M}$ (resp. $A_T^{G,M}$) such that $(G, \sigma) \models M$, if $\mathtt{s} \in \sigma(n)$ then $\mathtt{s}' \in \sigma(n)$.*

While the original definition only considered brave-total semantics, our formulation is more general, as it also includes brave-partial. It is important to notice that, if a SHACL document is unsatisfiable, any pair of shapes within that document trivially contain each other. In other words, the containment of a shape into another is not necessarily caused by any particular property of those shapes.

We should also note that the fragment studied in [19] for which shape containment is decidable is the SHACL fragment corresponding to the SCL sub-fragment of $\mathtt{C}$ (the base language plus counting quantifiers) where filters are not allowed. This is in agreement with our decidability results, that we present in Sec. 7, where we demonstrate decidability of of the similar SHACL satisfiability problem for even more general fragments of $\mathtt{C}$.

The shape containment problem can be expressed as the existence of a node n such that document $M \cup \{\langle \mathtt{s}^*, t_n, d^*\rangle\}$ is unsatisfiable under brave-partial (resp. brave-total) semantics, where $\mathtt{s}^*$ is a fresh shape name, t_n is a target declaration that targets only node n, and d^* is the constraint obtained by conjuncting d' and the negation of d.

Theorem 4. *Given a* SHACL *document M, and two shapes $\langle \mathtt{s}, t, d\rangle$ and $\langle \mathtt{s}', t', d'\rangle$ in M, $\mathtt{s}$ is not* shape contained *in $\mathtt{s}'$ under brave-partial (resp. brave-total) semantics iff there exist a node n such that document $M \cup \{\langle \mathtt{s}^*, t_n, d^*\rangle\}$ is satisfiable under brave-partial (resp. brave-total) semantics, where $\mathtt{s}^*$ is a fresh shape name, t_n is a target declaration that* *targets only node n, and d^* is the constraint obtained by conjuncting d and the negation of d'.*

Proof. Given a node n let $M' = M \cup \{\langle \mathsf{s}^*, t_n, d^* \rangle\}$.

$\Rightarrow$) If M' is satisfiable, let G be a graph that is valid *w.r.t.* it. If $n \in \mathsf{nodes(G)}$ it is easy to see that the following properties are true for graph G: (1) it is valid *w.r.t.* M (since M is a subset of M'), (2) there exists an assignment σ that is faithful (resp. faithful and total) for M and G, and such that $\mathsf{s} \in \sigma(n)$ and $\neg\mathsf{s}' \in \sigma(n)$ (since n satisfies constraints d, but not d'). One such assignment σ can be obtained by taking an assignment σ', faithful for G and M', and by removing elements s^* and $\neg\mathsf{s}^*$ from all the sets in the codomain of the σ' function. Thus, shape s is not contained in s' w.r.t. M. Instead, if $n \notin \mathsf{nodes(G)}$, then there exists another graph G' such that G' is valid *w.r.t.* M' and $n \in \mathsf{nodes(G')}$. One such graph G' is $G \cup \{<n^*, r^*, n>\}$, where n^* and r^* are, respectively, a fresh constant and a fresh relation name. This is because the shapes of a SHACL document can only target nodes mentioned in the document, or those that are reachable by the relations mentioned in the document. Moreover, the evaluation of any SHACL constraints on a node is unaffected by that node being the object of a triple with an unknown predicate. Since G' satisfies the same properties as G, we can apply the same reasoning as above (as for case $n \in \mathsf{nodes(G)}$) to prove that shape s is not contained in s' w.r.t. M.

$\Leftarrow$) If shape s is not contained in s' w.r.t. M then there exists a graph G, an assignment σ faithful (resp. faithful and total) for G and M, and a node n such that $\mathsf{s} \in \sigma(n)$ and $\neg\mathsf{s}' \in \sigma(n)$. Therefore, $[\![d^*]\!]^{n,G,\sigma}$ must be true. Let σ^* be the extension of the σ assignment that accounts for the s^* shape, namely $\sigma^*(j) = \sigma(j) \cup \{\mathsf{s}^* | [\![d^*]\!]^{j,G,\sigma} = \top\} \cup \{\neg\mathsf{s}^* | \neg[\![d^*]\!]^{j,G,\sigma} = \top\}$, for any node j in $\mathsf{nodes(G, M)}$. It is easy to see that assignment σ^* is faithful (resp. faithful and total) for M' and G, and thus M' is satisfiable. ☐

The above mentioned theorem introduces the following auxiliary decision problem.

Definition 16. *Given a* SHACL *document M, a shape name s not in M and a constraint d that only references shapes in $M \cup \{\mathsf{s}\}$,* template satisfiability *under brave-partial (resp. brave-total) semantics is the problem of deciding whether there exists a node n such that document $M \cup \{\langle \mathsf{s}, t_n, d \rangle\}$ is satisfiable under brave-partial (resp. brave-total) semantics.*

Two additional decision problems, *constraint satisfiability* and *constraint containment*, are defined in [22] to study the properties of non-recursive SHACL constraints. Intuitively, a constraint d is satisfiable if there exists a node that conforms to d, and a constraint d is contained in d' if every node that conforms to d also conforms to d'. We provide here a generalisation of these problems by introducing a SHACL document as an additional input. The primary purpose of this additional document is to study constraints under recursion, that is, constraints that reference recursive shapes. However, it can also be used to study constraint satisfiability and containment subject to a particular document being valid. When this document is empty the following decision problems correspond to the ones defined in [22], namely constraint satisfiability and containment without recursion.

Definition 17. *Given a* SHACL *constraint d and a* SHACL *document M, such that d does not reference shapes not included in M, constraint d is* satisfiable *under extended semantics α if there exists a node n such that* SHACL *document $M \cup \{\langle \mathsf{s}, t_n, d \rangle\}$ is satisfiable under α, where s is a fresh shape name.*

Definition 18. *Given two* SHACL *constraints d and d' and a* SHACL *documents M such that d and d' do not reference shapes not included in M, constraint d is* contained *in d' under extended semantics α if for all nodes n, document $M \cup \{\langle \mathtt{s}, t_n, d\rangle\}$ is contained in $M \cup \{\langle \mathtt{s}', t_n, d'\rangle\}$ under α, where $\mathtt{s}$ and $\mathtt{s}'$ are fresh shape names.*

The problem of constraint satisfiability under brave-partial and brave-total semantics are, by definition, sub-problems of SHACL template satisfiability for the respective semantics. Constraint containment for non-recursive SHACL documents is also a sub-problem of SHACL template satisfiability. This is a consequence of the fact that containment of two non-recursive SHACL documents can be decided by deciding the satisfiability of an ∃SCL sentence (Lemma 4). As we will prove later in Section 6, the problem of template satisfiability can be expressed as ∃SCL sentence satisfiability. Therefore, our positive results that will be presented in Section 7 also provide decidability and upper bound complexity results for the decision problems expressible as template satisfiability, namely (1) shape containment, (2) constraint satisfiability under brave-partial and brave-total semantics and (3) constraint containment for non-recursive SHACL documents.

6. From Interpreted To Uninterpreted Models via Filter Axiomatisation

In this section we discuss explicit axiomatizations of the semantics of a set of filters. The main goal of these axiomatisations is to account for filter semantics without requiring filters to be interpreted relations. For any MSCL sentence Φ we construct axiomatisations α such that Φ is satisfiable on a canonical model if and only if $\Phi \wedge \alpha$ is satisfiable on an *uninterpreted* models, that is, models whose domain is the set of RDF terms, but where filters and ordering relations are simple relations instead of interpreted ones. This reduction to standard FOL allows us to prove decidability of the satisfiability and containment problems for several SCL fragments in the face of filters.

We first present a simplified but expensive formulation of this axiomatisation, that is exponential on size of the original sentence. We then provide an alternative axiomatisation, polynomial on size of the original sentence, that however requires counting quantifiers to express certain filters. We exclude from our axiomatisation the `sh:lessThanOrEquals` or `sh:lessThan` constraints (the `O` and `O'` components of our grammar) that are binary relations, and which do not belong to any decidable fragment we have so far identified, as shown in the next section. For the `sh:pattern` constraint, which tests whether the string representation of a node follows a certain regular expression, we only consider standard regular expressions (i.e. regular expressions that can be converted into a finite state machine). All features defined as filters in Sec. 5, with the exception of `O` and `O'` components, are represented by monadic relations $F(x)$ of the SCL grammar. While equality remains an interpreted relation, for which we do not provide an axiomatisation, we will also consider equality to a constant `c` as a monadic filter relation (which we call equality-to-a-constant) whose interpretation is the singleton set containing `c`.

6.1. Naïve Axiomatisation

The semantics of each monadic filter relation is a predetermined interpretation over the domain. For example, the interpretation of filter relation $\mathtt{F}^{\mathrm{IRI}}$ is the set of all IRIs, since $\mathtt{F}^{\mathrm{IRI}}(x)$ is true *iff* x is an IRI. Notice also that filters are the only components of MSCL whose interpretation is predetermined. Thus, we can axiomatise the semantics of filters w.r.t. deciding satisfiability by capturing which conjunctions of filters are unsatisfiable, and which conjunctions of filters are satisfiable only by a finite set of elements. For example, the number of elements of the Boolean datatype is two, the number of elements that are literals is infinite, and there are four elements of integer datatype that are both greater than 0 and lesser than 5. 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 $\mathtt{c}$ is a constant and F is a filter predicate. Given a filter combination, it is possible to compute the set of elements of the domain that can satisfy it. Let γ be the function from filter combinations to subsets of the domain that returns this set. The computation of $\gamma(\mathbb{F}(x))$ for the monadic filters we consider is tedious but trivial as it boils down to determining: (1) the lexical space of datatypes; (2) the cardinality of intervals defined by order or string-length constraints; (3) the number of elements accepted by a regular expression; (4) well-known RDF-specific restrictions, e.g., the fact that each RDF term has exactly one node type, and at most one datatype and one language tag. Combinations of the previous four points are similarly computable. Let $\mathbb{F}^{\Phi}$ be the set of filter combinations that can be constructed with the filters predicates and constants occurring in an MSCL sentence Φ. The *naïve* filter axiomatization $\alpha(\Phi)$ of a sentence Φ is the following conjunction, where Σ_f is a fresh shape name.

$$\alpha(\phi) = \bigwedge_{\mathbb{F}(x)\in\mathbb{F}^{\phi},|\gamma(\mathbb{F}(x))\neq\infty|} (\forall x.\ \Sigma_f(x) \leftrightarrow \mathbb{F}(x)) \\ \wedge \left(\forall x.\ \Sigma_f(x) \leftrightarrow \begin{cases} \bot, & |\gamma(\mathbb{F}(x))| = 0 \\ \bigvee_{\mathtt{c}\in\gamma(\mathbb{F}(x))} x = \mathtt{c}, & \text{otherwise} \end{cases}\right)$$

To better illustrate this axiomatisation, consider the following MSCL sentence ϕ^*.

$$\phi^* = \Sigma(\mathtt{q}) \wedge \forall x.\ \big(\Sigma(x) \leftrightarrow \exists^4 y.\ R(x,y) \wedge \mathbb{F}^{>0}(y) \wedge \mathbb{F}^{\leq 5}(y) \wedge F^{\mathrm{dt}=xsd:int}(y) \\ \wedge\, y \neq 2 \wedge y \neq 3\big)$$

Intuitively, this sentence is satisfiable if a constant $\mathtt{q}$ can be in the R relation with four different integers that (a) are greater than 0, (b) that are less than or equal than 5, and (c), that are not equal to 2 or 3. Since there are only three integers that satisfy the conditions (a), (b) and (c) simultaneously, this sentence is not satisfiable on a canonical model. This sentence contains the filters $\mathbb{F}^{>0}(x)$, $\mathbb{F}^{\leq 5}(x)$ and $F^{\mathrm{dt}=xsd:int}(x)$, that denote, respectively, the fact that x is greater than the number 0, the fact that x is less or equal than the number 5, and the fact that x belongs to the XSD integer datatype[2]. The set of known constants of ϕ^* is $\{2, 3, \mathtt{q}\}$. We will assume that $\mathtt{q}$ is an IRI and that all other known constants are literals of the XSD integer datatype.

The naïve filter axiomatisation $\alpha(\phi^*)$ contains, among others, the following conjuncts, where Σ_f is a fresh shape name.

$$\big(\forall x.\ \Sigma_f(x) \leftrightarrow \mathbb{F}^{>0}(x) \wedge \mathbb{F}^{\leq 5}(x) \wedge F^{\mathrm{dt}=xsd:int}(x) \wedge x \neq 2 \wedge x \neq 3\big) \\ \wedge (\forall x.\ \Sigma_f(x) \leftrightarrow x = 1 \vee x = 4 \vee x = 5)$$

[2]The `https://www.w3.org/TR/xmlschema11-2/#integer` datatype is supported by SPARQL 1.1, and thus it has a predetermined lexical space.

This axiomatisation states that only three constants satisfy the main filter combination of ϕ^*, and thus $\phi^* \wedge \alpha(\phi^*)$ is unsatisfiable on an uninterpreted model.

Theorem 5. *Given an* MSCL *sentence ϕ and its naïve filter axiomatisation $\alpha(\phi)$, 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* MSCL *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. We focus on satisfiability, since the proof for containment is similar. Let $\mathtt{c}$ be any element of the domain and $\mathbb{F}(x)$ be any filter combination that can be constructed with the constants and filter relations in ϕ. Since the semantics of filter relations has a universal interpretation, $\mathbb{F}(\mathtt{c})$ is either true on all canonical models, or false on all canonical models. Notice that, by construction of our axiomatisation, the truth value of $\mathbb{F}(\mathtt{c})$ on all canonical models corresponds to the truth value of $\mathbb{F}(\mathtt{c})$ on all uninterpreted models of $\alpha(\phi)$. Let I' be an uninterpreted model of $\phi \wedge \alpha(\phi)$, we can construct I, canonical model of ϕ, by (1) changing all the uninterpreted filter relations in I' for their corresponding interpreted ones in I and (2) dropping from I' the interpretation of all the shape relations that occur in $\alpha(\phi)$. Let I be a canonical model of ϕ, we can construct I', uninterpreted model of $\phi \wedge \alpha(\phi)$, by (1) changing all the interpreted filter relations in I for their corresponding uninterpreted ones in I' and (2) by adding the following interpretation of each shape relation $\Sigma_f(x)$ occurring in $\alpha(\phi)$ to I: let $\mathbb{F}(x)$ be the filter combination such that $\forall x.\ \Sigma_f(x) \leftrightarrow \mathbb{F}(x)$ is one of the conjuncts of $\alpha(\phi)$ (notice that one such conjunct exists for any shape relation), relation Σ_f contains all the elements of the domain which satisfy the filter combination $\mathbb{F}(x)$ on canonical models. $\square$

6.2. Bounded Axiomatisation

The main exponential factor in the axiomatisations above is the set of all possible filter combinations. However, we can limit an axiomatisation to filter combinations having a number of atoms smaller or equal to a constant number, thus making our axiomatisation polynomial w.r.t. an MSCL sentence Φ. Intuitively, this can be achieved because $\mathbb{F}^{\Phi}$ contains several redundant filter combinations. To illustrate this point, consider *datatype filters* atoms $F^{\mathrm{dt}=\mathtt{c}}(x)$, derived from the `sh:datatype` constraint component, that are true if x is a literal with datatype $\mathtt{c}$.[3] Let Φ be an MSCL sentence and $\mathbb{F}(x)$ be a filter combination $F^{\mathrm{dt}=\mathtt{c}}(x) \wedge F^{\mathrm{dt}=\mathtt{c}'}(x)$ of $\mathbb{F}^{\Phi}$, where $\mathtt{c} \neq \mathtt{c}'$. Since no RDF term can have two different datatypes, the truth value of $\mathbb{F}(x)$ is always false (i.e. $|\gamma(\mathbb{F}(x))| = 0$). Trivially, any filter combination in $\mathbb{F}^{\Phi}$ whose conjuncts are a proper superset of $\mathbb{F}(x)$ is also false, and thus its axiomatisation is not necessary.

In order to limit the size of the filter combinations to a constant number, we reason about each filter type to determine the maximum number of conjuncts of that type to consider in any filter combination. We call this number the *maximum non-redundant capacity* (MNRC) of that filter type. Any filter combination that contains more conjuncts of that type than its MNRC, is necessarely *redundant.*

Definition 19. *A filter combination $\mathbb{F}(x)$ is redundant if there exists a filter combination $\mathbb{F}'(x)$ such that $\gamma(\mathbb{F}(x)) = \gamma(\mathbb{F}'(x))$ and $\mathbb{F}'(x)$ is a proper subset of $\mathbb{F}(x)$.*

[3]According to the SPARQL standard literals with different datatype or language tags are different RDF terms (e.g. literal “10” of datatype integer is not equal to literal “10” of datatype float).

We will now define the MNRC for all the monadic SHACL filter types. In the following proofs we will assume that all conjuncts of a filter combination are syntactically different from each other as any filter combination that contains multiple copies of the same conjunct is trivially redundant. The MNRC of datatype filters is two.

Lemma 5. *Any filter combination $\mathbb{F}(x)$ that contains more than two datatype filter conjuncts is redundant.*

Proof. Since no RDF term can have two datatypes, if $\mathbb{F}(x)$ contains two positive datatype filter conjuncts, then $\mathbb{F}(x)$ is unsatisfiable. Thus $\mathbb{F}(x)$ cannot contain more than two positive datatype filter conjuncts without being redundant. Since RDF literals do not need to be annotated with a datatype, any negation $\neg F^{\mathrm{dt}=\mathtt{c}}(x)$ of a datatype filter does not affect the truth value of a filter combination, unless the datatype filter also contains conjunct $F^{\mathrm{dt}=\mathtt{c}}(x)$, in which case the filter combination is trivially unsatisfiable. Thus, if $\mathbb{F}(x)$ is not redundant, either it does not contain negated datatype filters, or it contains the two filters $F^{\mathrm{dt}=\mathtt{c}}(x)$ and $\neg F^{\mathrm{dt}=\mathtt{c}}(x)$ for a constant `c`. In this last case, the occurrence of any further datatype filter in $\mathbb{F}(x)$ would make the filter combination redundant. □

We represent *language tag* filters, derived from the `sh:languageIn` and `sh:uniqueLang`, with the $F^{\mathrm{languageTag}\ =\ \mathtt{c}}(x)$ filter relation, which is true if x is string literal with language tag `c`. Since not all string literals have a language tag, but no string literal has more than one such tag, this type of filter behaves analogously to the datatype filter. The proof of the following lemma, which states that the MNRC of language tag filters is two, can be derived from the one above.

Lemma 6. *Any filter combination $\mathbb{F}(x)$ that contains more than two language tag filter conjuncts is redundant.*

The *order comparison* filters, which are expressible in SHACL with the `sh:minExclusive`, `sh:maxExclusive`, `sh:minInclusive` and `sh:maxInclusive` constraint components, denote the $x > c$, $x < c$, $x \geq c$ and $x \leq c$ operators, respectively. Order comparison filters have an MNRC of two.

Lemma 7. *Any filter combination $\mathbb{F}(x)$ that contains more than two order comparison filter conjuncts is redundant.*

Proof. If two order comparison filters in $\mathbb{F}(x)$ are defined over incompatible comparison types (e.g. strings and dates) then $\mathbb{F}(x)$ is unsatisfiable, and all the other comparison filters in $\mathbb{F}(x)$ are redundant. In a set of filters, we define as the *most restrictive* the one with the smallest number of elements satisfying it, or any such filter if there is more than one. If all the comparison filters in $\mathbb{F}(x)$ are defined over the same comparison type, let α be the most restrictive conjunct in $\mathbb{F}(x)$ of type $x > c$, $\neg x < c$, $x \geq c$ and $\neg x \leq c$ (or $\top$ if none such conjunct exists), and ω be the most restrictive conjunct in $\mathbb{F}(x)$ of type $\neg x > c$, $x < c$, $\neg x \geq c$ and $x \leq c$. Trivially, $\mathbb{F}(x)$ is semantically equivalent to $\mathbb{F}'(x)$, which is constructed by removing from $\mathbb{F}(x)$ all comparison filters that are not α or ω. □

String length comparison filters are expressed in SHACL with the constraint components `sh:minLength` and `sh:maxLength`, and they behave analogously to the order comparison filters. The proof of the following lemma, which states that the MNRC of string length comparison filters is two, can be derived from the one above.

Lemma 8. *Any filter combination $\mathbb{F}(x)$ that contains more than two string length comparison filter conjuncts is redundant.*

Node kind filters can be represented by three filter relations $\mathrm{F}^{\,\mathrm{IRI}}(x)$, $\mathrm{F}^{\,\mathrm{literal}}(x)$ and $\mathrm{F}^{\,\mathrm{blank}}(x)$ that are true if x is, respectively, an IRI, a literal or a blank node. Node kind filters have an MNRC of three.

Lemma 9. *Any filter combination $\mathbb{F}(x)$ that contains more than three node kind filter conjuncts is redundant.*

Proof. This lemma can be proven in the same manner as Lemma 5, with the exception that, since all RDF terms belong to exactly one of the tree node kinds, filter combination $\neg\mathrm{F}^{\,\mathrm{IRI}}(x) \wedge \neg\mathrm{F}^{\,\mathrm{literal}}(x) \wedge \neg\mathrm{F}^{\,\mathrm{blank}}(x)$ is unsatisfiable and it is not redundant. $\square$

We can establish an MNRC of 1 for the equality-to-a-constant operator (expressed in SHACL with the `sh:hasValue` and `sh:in` constraints), by noticing that any variable x, by the law of excluded middle, is either interpreted as one of the known constants, or as none of them. In SCL we can express with $\Sigma_\nu(x)$ the fact that x is none of the known constants C, where ν is a unique shape name defined as $\Sigma_\nu(x) \leftrightarrow \bigwedge_{c\in C} \neg x = c$. Intuitively, we consider all possible interactions of the equality operator with filter combinations by considering whether an element x is one of the known constants, or whether it conforms to shape $\Sigma_\nu(x)$. In order to use this new shape ν in our axiomatisation, we redefine a *filter combination* $\mathbb{F}(x)$ as a conjunction of atoms of the form $x = \mathtt{c}$, $x\neg = \mathtt{c}$, $\Sigma_\nu(x)$, $F(x)$ and $\neg F(x)$.

Lemma 10. *Any filter combination $\mathbb{F}(x)$ that contains more than one equality-to-a-constant conjuncts is redundant.*

Proof. Any filter combination $\mathbb{F}(x)$ that contains more than one equality-to-a-constant operator, of which at least one is in positive form, is redundant. In fact, a filter combination is made redundant by: (a) any two positive equality-to-a-constant operators $x = \mathtt{c} \wedge x = \mathtt{c}'$, with $\mathtt{c} \neq \mathtt{c}'$ (recall that we are using the unique name assumption), which is unsatisfiable by the standard interpretation of the equality operator, and (b) any pair of a positive and a negative equality-to-a-constant operators $x = \mathtt{c} \wedge x \neq \mathtt{c}'$ because (b.1) if $\mathtt{c}$ and $\mathtt{c}'$ are the same constant, then the pair of conjuncts is unsatisfiable by the standard interpretation of the equality operator and (b.2) if $\mathtt{c}$ is not the same constant as $\mathtt{c}'$ then conjunct $\neg x = \mathtt{c}'$ is redundant.

Moreover, any filter combination $\mathbb{F}(x)$ that contains equality-to-a-constant operators, but all negated, is also redundant. Let D be the domain of discourse, C be the set of known constants in the sentence Φ from which the filter combinations have been created, and C^- the set of constants that are in the negated equality-to-a-constant operators of $\mathbb{F}(x)$. The equality-to-a-constant operators in $\mathbb{F}(x)$ restricts the domain to elements $D \setminus C^-$. Let $\mathbb{F}^*(x)$ be the subset of $\mathbb{F}(x)$ without equality-to-a-constant conjuncts. We can rewrite $\mathbb{F}(x)$ into an equivalent set of filter combinations $\bar{\mathbb{F}}$ that contain at most one equality-to-a-constant operator by noticing that we can rewrite $D \setminus C^-$ as $(D \setminus C) \cup (C \setminus C^-)$, and that the left-hand side of this last union of sets corresponds to the elements in the interpretation of $\Sigma_\nu(x)$, while the right-hand side is a finite set of known constants. The set of filter combinations $\bar{\mathbb{F}}$ that makes $\mathbb{F}(x)$ redundant is defined as follows: $\bar{\mathbb{F}} = \{\mathbb{F}^*(x) \wedge \Sigma_\nu(x)\} \cup \{\mathbb{F}^*(x) \wedge x = \mathtt{c} | \mathtt{c} \in C \setminus C^-\}$. Since every element of the domain either belongs to $\Sigma_\nu(x)$ or it is one of the known constants, the restrictions imposed by $\mathbb{F}(x)$ and by the set $\bar{\mathbb{F}}$ are equivalent.

$\square$

The only filter constraint that does not have a maximum non-redundant capacity is `sh:pattern`, since any number of regular expressions can be combined together to generate novel and non-redundant regular expressions.

We define the set of *bounded filter combinations* $\mathbb{F}'^{\phi}$ of an MSCL sentence ϕ the set of all conjunctions such that (1) the conjuncts are atoms of the form $x = \mathtt{c}$, $\Sigma_\nu(x)$, $F(x)$ or $\neg F(x)$, where $\mathtt{c}$ is a constant occurring in ϕ and F is a filter predicate occurring in ϕ; (2) the number of conjuncts of each filter type, and of equality, does not exceed its maximum non-redundant capacity.

Notice that in the previous axiomatisation the size of each conjunct depends on the size of the finite sets computed by the γ function. While certain filter constraints, such as `sh:nodeKind`, are either satisfiable by an infinite number of elements, or are unsatisfiable, other constraints can be satisfied by an arbitrarily large number of elements. We can reduce the size of each conjunct to a logarithmic factor (with a binary numeric representation) by using counting quantifiers. This allows us to express the maximum number of elements that can satisfy a filter combination without explicitly enumerating them.

Given an MSCL sentence ϕ and the set C of all known constants in ϕ, the *bounded* axiomatisation $\bar{\alpha}(\phi)$ of ϕ is defined as follows.

$$\bar{\alpha}(\phi) = \left(\Sigma_\nu(x) \leftrightarrow \bigwedge_{c \in C} \neg x = c\right) \wedge \bigwedge_{\mathbb{F}(x) \in \mathbb{F}'^{\phi}, |\gamma(\mathbb{F}(x)) \neq \infty|} \exists^{\leq \gamma(\mathbb{F}(x))} x.\ \mathbb{F}(x)$$

By lemmas 5 to 10, if ϕ does not contain any filter of the `sh:pattern` type, the bounded axiomatisation only includes filter combinations of up to 12 conjuncts. Thus, the size of the bounded axiomatisation is polynomial w.r.t. ϕ.

To better explain this second axiomatisation, let us consider again the example of the MSCL sentence ϕ^* defined before. The bounded axiomatisation $\bar{\alpha}(\phi^*)$ of ϕ^* contains, among others, the following conjuncts:

$$\begin{aligned}
&(\Sigma_\nu(x) \leftrightarrow x \neq 2 \wedge x \neq 3 \wedge x \neq \mathtt{q}) \\
&\wedge \left(\exists^{\leq 5} x.\ \mathbb{F}^{>0}(x) \wedge \mathbb{F}^{\leq 5}(x) \wedge F^{\mathrm{dt}=xsd:int}(x)\right) \\
&\wedge \left(\exists^{\leq 3} x.\ \mathbb{F}^{>0}(x) \wedge \mathbb{F}^{\leq 5}(x) \wedge F^{\mathrm{dt}=xsd:int}(x) \wedge \Sigma_\nu(x)\right) \\
&\wedge \left(\exists^{\leq 0} x.\ \mathbb{F}^{>0}(x) \wedge \mathbb{F}^{\leq 5}(x) \wedge F^{\mathrm{dt}=xsd:int}(x) \wedge x = \mathtt{q}\right)
\end{aligned}$$

Of the four elements required by the existentially bounded sub-formula of ϕ^* to satisfy a filter combination, only three can belong to $\Sigma_\nu(x)$ (by the third line of the axiomatisation). The remaining one must satisfy both $(x = 2 \vee x = 3 \vee x = \mathtt{q})$ and $x \neq 2 \wedge x \neq 3$, and thus cannot be a constant other than $\mathtt{q}$. However, $\mathtt{q}$ is not compatible with the filter combination (by the last line of the axiomatisation). Therefore, $\phi^* \wedge \bar{\alpha}(\phi^*)$ is unsatisfiable on an uninterpreted model.

It should be noted that the bounded axiomatisation does not follow the MSCL grammar, while the naïve filter axiomatisation does, albeit not resulting in well-formed sentences. The differences between our axiomatisations and well-formed MSCL sentences, however, do not affect our decidability and complexity results presented in the following section since (a), the positive results are applicable to fragments of first-order logic that are general enough to express our axiomatisations and (b), the negative results are applicable to SHACL sentences without filters, which therefore do not require an axiomatisation. For the purposes of the decidability and complexity analysis presented in the following section, the naïve filter axiomatisation is compatible with all of the language fragments, while the bounded filter axiomatisation is compatible with the fragments that include counting quantifiers.

Theorem 6. *Given an* MSCL *sentence ϕ and its bounded filter axiomatisation $\bar{\alpha}(\phi)$, sentence ϕ is satisfiable on a canonical model iff $\phi \wedge \bar{\alpha}(\phi)$ is satisfiable on an uninterpreted model. Containment $\phi_1 \subseteq \phi_2$ of two* MSCL *sentences on all canonical models holds iff $\phi_1 \wedge \bar{\alpha}(\phi_1 \wedge \phi_2) \subseteq \phi_2$ holds on all uninterpreted models.*

Proof. 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 $I^\star$. □

6.3. *From Template Satisfiability to* MSCL *Satisfiability*

As anticipated in the previous section, the problem of template satisfiability (Def. 16) can be reduced into an ∃SCL satisfiability problem. In particular, achieving this reduction in the face of filters requires the additional machinery of the bounded filter axiomatisation. The correspondence between SHACL template satisfiability and ∃SCL sentence satisfiability is given by the following theorem. The intuition behind this theorem is that, in an uninterpreted model, unknown constant symbols are interchangable. Therefore, on an uninterpreted model, considering template satisfiability for one unknown constant symbol amounts to considering this problem for all possible constants. Let $Constant(\phi)$ denote the set of constants in ϕ.

Theorem 7. *The answer to the template satisfiability problem for M, s and d under brave-total semantics is* True *iff there exists a constant symbol $f \in Constant(\phi) \cup \{\mathsf{c}\}$, with c a fresh constant symbol, such that $\phi \wedge \bar{\alpha}(\phi) \wedge \Sigma_{\mathsf{s}}(f)$ is satisfiable on an uninterpreted model, where $\phi = \tau(M \cup \{\langle \mathsf{s}, \emptyset, d\rangle\})$.*

Proof. Recall that, by Theorem 6, there exists a canonical model I' such that $I \models \phi \wedge \Sigma_{\mathsf{s}}(n)$ iff there exists an uninterpreted model J such that $J \models \phi \wedge \Sigma_{\mathsf{s}}(n) \wedge \bar{\alpha}(\phi \wedge \Sigma_{\mathsf{s}}(n))$.

(⇒) Assume that the answer to the template satisfiability problem for M, s and d under brave-total semantics is true. Per Def. 16 this means that there exists an RDF graph G and a node n such that G is valid w.r.t. $M \cup \{\langle \mathtt{s}, t_n, d\rangle\}$. From the translation of target declarations in Table 2 it follows that $\tau(M \cup \{\langle \mathtt{s}, t_n, d\rangle\})$ can be written as $\phi \wedge \Sigma_{\mathtt{s}}(n)$, where $\phi = \tau(M \cup \{\langle \mathtt{s}, \emptyset, d\rangle\})$. Moreover, by Theorem 3, there exists a canonical structure I such that $I \models \tau(M \cup \{\langle \mathtt{s}, t_n, d\rangle\})$, which means that $I \models \phi \wedge \Sigma_{\mathtt{s}}(n)$, thanks to our previous observation. Consider the following cases: (1) $n \in \mathit{Constant}(\phi)$ and (2) $n \notin \mathit{Constant}(\phi)$

In the first case, let f be n. Then there exists an uninterpreted model J such that $J \models \phi \wedge \Sigma_{\mathtt{s}}(f) \wedge \bar{\alpha}(\phi \wedge \Sigma_{\mathtt{s}}(f))$. Notice also that the bounded filter axiomatisation of an MSCL sentence ρ depends only on the set of filter relations and the set of constants in ρ. Therefore, if $n \in \mathit{Constant}(\phi)$ then $\bar{\alpha}(\phi \wedge \Sigma_{\mathtt{s}}(f)) = \bar{\alpha}(\phi)$. Thus the thesis follows.

In the second case there exists an uninterpreted model J and a constant n such that $J \models \phi \wedge \Sigma_{\mathtt{s}}(n) \wedge \bar{\alpha}(\phi \wedge \Sigma_{\mathtt{s}}(n))$. Notice that $\bar{\alpha}(\phi \wedge \Sigma_{\mathtt{s}}(n))$ implies $\bar{\alpha}(\phi)$, since sentence $\phi \wedge \Sigma_{\mathtt{s}}(n)$ contains the same filter relations as ϕ, and all the constants of ϕ plus one additional constant. The additional constant in $\phi \wedge \Sigma_{\mathtt{s}}(n)$ only results in a stronger axiomatisation that considers more cases. Thus $J \models \phi \wedge \bar{\alpha}(\phi)$ and $\Sigma_{\mathtt{s}}$ is not empty in J. Let J^* be the extension of the uninterpreted model J where constant symbol f is mapped to n, then $J^* \models \phi \wedge \bar{\alpha}(\phi) \wedge \Sigma_{\mathtt{s}}(f)$ as required by the theorem statement.

($\Leftarrow$) Assume that there exists an uninterpreted model J such that $J \models \phi \wedge \bar{\alpha}(\phi) \wedge \Sigma_{\mathtt{s}}(f)$. We distinguish two cases similar to the cases discussed before: (1) $f \in \mathit{Constant}(\phi)$ and (2) $f \notin \mathit{Constant}(\phi)$.

In the first case, the thesis can be proven by following the reverse proof of the first case of the previous directionality. More specifically, $\bar{\alpha}(\phi) = \bar{\alpha}(\phi \wedge \Sigma_{\mathtt{s}}(f))$ and thus $J \models \phi \wedge \bar{\alpha}(\phi \wedge \Sigma_{\mathtt{s}}(f)) \wedge \Sigma_{\mathtt{s}}(f)$. By Theorem 6 there exists a canonical model I such that $I \models \phi \wedge \Sigma_{\mathtt{s}}(f)$.

In case (2), we prove that $J \models \phi \wedge \bar{\alpha}(\phi) \wedge \Sigma_{\mathtt{s}}(f)$ implies the existence of a value v in the domain of constants such that the uninterpreted model $J[f \mapsto v]$ (obtained by mapping constant symbol f to v in J) models $\phi \wedge \Sigma_{\mathtt{s}}(f) \wedge \bar{\alpha}(\phi \wedge \Sigma_{\mathtt{s}}(f))$. If no such value v exists, then it must follow that there exist a non-empty filter combination $\mathbb{F}$, without equality operators, such that $J \models \mathbb{F}(f)$, but such that $\bar{\alpha}(\phi \wedge \Sigma_{\mathtt{s}}(f)) \rightarrow \forall x.\neg\mathbb{F}(x)$. Since $\mathbb{F}$ does not contain equality operators, and since $\phi \wedge \Sigma_{\mathtt{s}}(f)$ and ϕ contain the same shape relations, it follows that $\bar{\alpha}(\phi) \rightarrow \forall x.\neg\mathbb{F}(x)$, which is in contradiction to the premises. Intuitively, this is due to the fact that the interpretation of filters is universal, so if a filter combination $\mathbb{F}$ is unsatisfiable, it is unsatisfiable in all axiomatisations whose filter relations can express $\mathbb{F}$. Having proven the existence of uninterpreted model $J[f \mapsto v]$, such that $J[f \mapsto v] \models \phi \wedge \Sigma_{\mathtt{s}}(f) \wedge \bar{\alpha}(\phi \wedge \Sigma_{\mathtt{s}}(f))$ the existence of a canonical model I such that $I \models \phi \wedge \Sigma_{\mathtt{s}}(v)$ easily follows, and thus the thesis is proven. $\square$

By this theorem, the positive decidability results that we will present in Sect. 7 are also applicable to SHACL template satisfiability, and the complexity of the corresponding decision procedures can be considered an upper bound for the complexity of SHACL template satisfiability in the same fragment, when it is at least polynomial. This, in turn, allows us to extend our positive results to many of the additional decision problems discussed in Section 5.2.

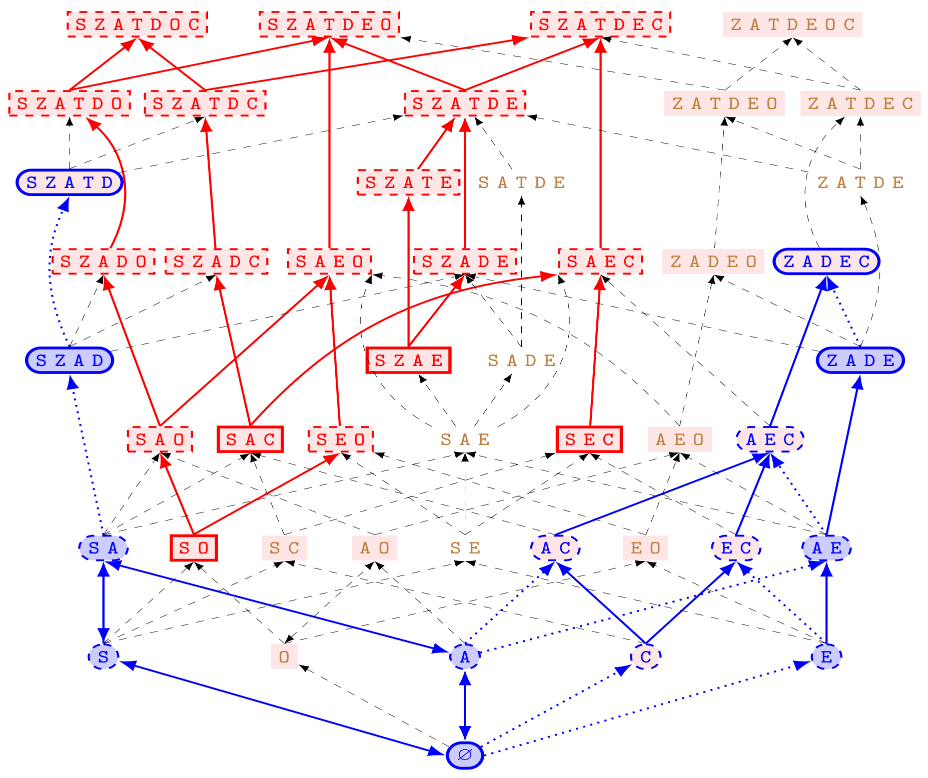

Figure 3: Decidability and complexity map of SCL fragments. Round (blue) and square (red) nodes denote decidable and undecidable fragments, respectively. Solid borders on nodes correspond to theorems in this paper, while dashed ones are implied results. Directed edges indicate inclusion of fragments, while bidirectional ones denote polynomial-time reducibility. Solid edges are preferred derivations to obtain complexity-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. Nothing is known for the remaining fragments reported in the figure.

7. SCL **Satisfiability**

We finally embark on a detailed analysis of the satisfiability problem for different fragments of SCL. Some of the proven and derived results are visualised in Figure 3. The decidability results are proved via embedding in known decidable (extensions of) fragments of first-order logic, while the undecidability ones are obtained through reductions from the classic domino problem [31]. Since we are not considering filters explicitly, but through axiomatisation, the only interpreted relations are equality and orderings between elements.

For the sake of clarity 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.*, `Z A D E C` or `S Z A T D`. 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'` features. Notice that the undecidability results exploiting the last two features are only applicable in the case of generalised RDF.

7.1. Decidability Results

As a preliminary 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 last one does not increase in expressive power when the `D` and `O` features are also taken in consideration.

Theorem 8. *There are (a) semantic-preserving and (b) polynomial-time finite-model-invariant satisfiability-preserving translations among the following* SCL *fragments: 1.* $\varnothing \equiv \mathtt{S} \equiv \mathtt{Z} \equiv \mathtt{A} \equiv \mathtt{SZ} \equiv \mathtt{SA} \equiv \mathtt{ZA} \equiv \mathtt{SZA}$*; 2.* $\mathtt{D} \equiv \mathtt{AD}$*; 3.* $\mathtt{O} \equiv \mathtt{AO}$*; 4.* $\mathtt{DO} \equiv \mathtt{ADO}$.

Proof. To show the equivalences among the fourteen SCL fragments mentioned in the statement, 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 formulae π_1 and π_2 in the body of an existential quantification is removed by nesting two plain 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:

$$\begin{aligned}
\exists y. (\pi_1(x, y) \vee \pi_2(x, y)) \wedge \psi(y) &\equiv (\exists y. \pi_1(x, y) \wedge \psi(y)) \vee \\
&\vee (\exists y. \pi_2(x, y) \wedge \psi(y)); \\
\neg\exists y. (\pi_1(x, y) \vee \pi_2(x, y)) \wedge R(x, y) &\equiv (\neg\exists y. \pi_1(x, y) \wedge R(x, y)) \wedge \\
&\wedge (\neg\exists y. \pi_2(x, y) \wedge R(x, y)); \\
\forall y, z. (\pi_1(x, y) \vee \pi_2(x, y)) \wedge R(x, z) \rightarrow \sigma(y, z) &\equiv (\forall y, z. \pi_1(x, y) \wedge R(x, z) \rightarrow \sigma(y, z)) \wedge \\
&\wedge (\forall y, z. \pi_2(x, y) \wedge R(x, z) \rightarrow \sigma(y, z))
\end{aligned}$$

At this point, the equivalences between the fragments naturally follow by an iterative application of the reported equivalences used as rewriting rules. This clearly concludes the proof of Item a.

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. Therefore, to prove Item b, *i.e.*, 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 size linear 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 of 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 $\exists y.\, \pi(x, y) \wedge \Sigma(y)$, where Σ is a fresh monadic relation; (ii) conjoin the resulting sentence with $\forall x.\, \Sigma(x) \leftrightarrow \psi(x)$. The two rewriting operations in isolation 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 [3]. This resemblance is effectively exploited as a key observation at the core of the following result.

Theorem 9. *All* SCL *subfragments of* `SZA` *enjoy the finite-model property and an* ExpTime-complete *satisfiability problem.*

Proof. The finite-model property follows from the fact that the subsuming `S Z A D` fragment enjoys the same property, as shown later on in Theorem 12.

As far as the satisfiability problem is concerned, thanks to Item 1 of Theorem 8, we can focus on the base fragment $\varnothing$.

On the one hand, on the hardness side, one can be observe that the description logic $\mathcal{ALC}$ extended with inverse roles and nominals ($\mathcal{ALCOI}$) [3] and the fragment $\varnothing$ deprived of the universal quantifications at the level of sentences (*i.e.*, the $\varnothing$ subfragment generated by grammar rule $\varphi := \top \mid \varphi \wedge \varphi \mid \Sigma(c)$) are linearly interreducible. Indeed, every existential modality $\exists R.C$ (*resp.*, $\exists R^{-}.C$) can be translated back-and-forth to the SCL construct $\exists y.\, R(x, y) \wedge \psi_C(y)$ (*resp.*, $\exists y.\, R^{-}(x, y) \wedge \psi_C(y)$), where ψ_C represents the recursive translation of the concept C. Moreover, every nominal n corresponds to the equality construct $x = c_n$, where a natural bijection between nominals and constant symbols is considered. At this point, since the aforementioned description logic has an ExpTime-complete satisfiability problem [28, 11], it holds that the same problem for all subfragments of `S Z A` is ExpTime-hard.

On the other hand, 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 role U [28, 18, 26], which has an ExpTime-complete satisfiability problem [27]. Indeed, the universal sentences of the form (a) $\forall x.\, \mathtt{isA}(x, c) \rightarrow \Sigma(x)$, (b) $\forall x, y.\, R^{\pm}(x, y) \rightarrow \Sigma(x)$, (c) and $\forall x.\, \Sigma(x) \leftrightarrow \psi(x)$ can be translated, respectively, as follows: (a) $n_c \wedge \forall \mathtt{isA}^{-}.\Sigma$, where n_c is the nominal for the constant c; (b) $\forall U.\forall R^{\mp}.\Sigma$; (c) $\forall U.(\Sigma \leftrightarrow C_{\psi})$, where C_{ψ} is the concept obtained by translating the $\varnothing$-formula ψ into $\mathcal{ALCOI}$. □

To derive properties of the `Z A D E` fragment, together with its sub-fragments (two of those – `E` and `A E` – are included in Figure 3), we leverage on the syntactic embedding in the *two-variable fragment* of first-order logic [21].

Theorem 10. *The* `ZADE` *fragment of* SCL *enjoys the finite-model property and a* NExpTime *satisfiability problem.*

Proof. Via a syntactic inspection of the SCL grammar one can observe that, by avoiding the `S` and `O` features of the language, it is only possible to write formulae with at most two free variables. For this reason, every `Z A D E`-formula belongs to the two-variable fragment of first-order logic [21] which is known to enjoy both the exponentially-bounded finite-model property and a NExpTime-complete satisfiability problem [14]. □

The embedding in the two-variable fragment used in the previous theorem can be generalised when the `C` feature is added to the picture. However, the gained additional expressive power does not come without a price, since the finite-model property is not preserved.

Theorem 11. *The non-recursive* `C` *fragment of* SCL *does not enjoy the finite-model property (on both sentences and formulae) and has a* NExpTime-hard *satisfiability problem. Nevertheless, the finite and unrestricted satisfiability problems for the* `ZADEC` *fragment are* NExpTime-Complete.

Proof. As for the proof of Theorem 10, 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 non-recursive `C` fragment of SCL allows us to express a similar property via the following sentence φ, thus proving the first part of the statement:

$$\varphi \triangleq \mathtt{isA}(0,c) \wedge \Sigma(0) \wedge \forall x.\, \Sigma(x) \leftrightarrow \psi_1(x) \wedge \forall x.\, \mathtt{isA}(x,c) \rightarrow \psi_2(x);$$
$$\psi_1(x) \triangleq \neg\exists y.\, R^-(x,y);$$
$$\psi_2(x) \triangleq \exists^{=1} y.\, (R(x,y) \wedge \mathtt{isA}(y,c)) \wedge \neg\exists^{\geq 2} y.\, R^-(x,y).$$

Intuitively, the first three conjuncts of φ force every model of the sentence to contain a distinguished element 0 that (i) does not have any R-predecessor and (ii) is related to an arbitrary but fixed constant c *w.r.t.* `isA`. In other words, 0 is contained in the domain of the relation `isA`, but is not contained in the image of the relation R. Then, the final 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.

It is interesting to observe that the ability to model an infinity axiom is already present at the level of constraints, as witnessed by the following `C`-formula, where the constant 0 is replaced by the existentially quantified variable x, where $\psi_1(x)$ and $\psi_2(x)$ are the previously introduced formulae with one free variable:

$$\widetilde{\psi}(z) \triangleq (z = c) \wedge \exists x.\, (\mathtt{isA}^-(z,x) \wedge \psi_1(x)) \wedge \forall x.\, \mathtt{isA}^-(z,x) \rightarrow \psi_2(x).$$

By generalising the proof of Theorem 9, one can notice that the `C` fragment of SCL semantically subsumes the description logic $\mathcal{ALC}$ extended with inverse roles, nominals, and cardinality restrictions ($\mathcal{ALCOIQ}$) [3]. 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 ψ_C represents the recursive translation of the concept C. Thus, the hardness result for `C` follows by recalling that the specific $\mathcal{ALC}$ language has a NExpTime-hard satisfiability problem [29, 20].

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 [23, 24]. Hence, the second part of the statement follows as well. $\square$

For the `S Z A D` fragment, we obtain model-theoretic and complexity results via an embedding in the *unary-negation fragment* of first-order logic [6]. When the `T` feature is considered, the same embedding can be adapted to rewrite `S Z A T D` into the extension of the mentioned first-order fragment with regular path expressions [16]. Unfortunately, as for the addition of the `C` feature to `Z A D E`, we need to pay the price of losing the finite-model property.

Theorem 12. *The* `SZAD` *fragment of* SCL *enjoys the finite-model property, while the non-recursive* `STD` *fragment does not (on both sentences and formulae). Nevertheless, the finite and unrestricted satisfiability problems for the* `SZATD` *fragment are solvable in 2*ExpTime*.*

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 formulae 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].

Mutatis mutandis, 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 non-recursive `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 $\pi(x, y) \triangleq \exists z. (R^-(x, z) \wedge (R^-(z, y))^\star)$. Obviously, $\pi(x, y)$ holds between two elements x and y of a model *iff* 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 $\psi(x) \triangleq \neg\exists y. (\pi(x, y) \wedge R(x, y))$, 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 non-recursive `S T D` sentence φ expresses exactly this property, where c is an arbitrary but fixed constant:

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

The same can be stated via the following non-recursive `S T D`-formula:

$$\widetilde{\psi}(z) \triangleq (z = c) \wedge \mathtt{isA}(0, z) \wedge \forall x. \mathtt{isA}^{-1}(z, x) \rightarrow (\psi(x) \wedge \exists y. (R(x, y) \wedge \mathtt{isA}(y, c))).$$

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, 10]. $\square$

At this point, it is interesting to observe that the O feature allows us to express a very weak form of counting restriction which is, however, powerful enough to describe an infinity axiom.

Theorem 13. *The non-recursive* O *and* EO′ *fragments of* SCL *do not enjoy the finite-model property (on both sentences and formulae).*

Proof. Similarly to the use of the C construct of SCL, a simple combination of just few instances of the O feature allows us to write the following sentence φ encoding the existence of an injective function that is not surjective. Indeed, 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 R and S. Then, by applying both O' and O to the inverse of R and S, we ensure that S is equal to R^{-}, which in its turn implies that the latter is functional as well. Hence, the statement of the theorem immediately follows.

$$\begin{aligned}
\varphi &\triangleq \mathtt{isA}(0,c) \wedge \Sigma(0) \wedge \forall x.\, \Sigma(x) \leftrightarrow \psi_1(x) \wedge \forall x.\, \mathtt{isA}(x,c) \rightarrow \psi_2(x);\\
\psi_1(x) &\triangleq \neg\exists y.\, R^{-}(x,y);\\
\psi_2(x) &\triangleq \exists y.\, (R(x,y) \wedge \mathtt{isA}(y,c))\\
&\quad \wedge\ \forall y,z.\, R(x,y) \wedge R(x,z) \rightarrow y \leq z \wedge \forall y,z.\, S(x,y) \wedge S(x,z) \rightarrow y \leq z\\
&\quad \wedge\ \forall y,z.\, R^{-}(x,y) \wedge S(x,z) \rightarrow y \leq z \wedge \forall y,z.\, R^{-}(x,y) \wedge S(x,z) \rightarrow y \geq z.
\end{aligned}$$

To show that the E O' fragment does not enjoy the finite-model property too, it is enough to replace the last two applications of the O' and O features with the E-formula $\forall y.\, R^{-}(x,y) \leftrightarrow S(x,y)$, which clearly ensures the functionality of R^{-}, being S functional.

Notice that also in this case we can express the above property at the level of formulae with one free variable, where $\psi_1(x)$ and $\psi_2(x)$ are defined as above:

$$\widetilde{\psi}(z) \triangleq (z = c) \wedge \exists x.\, (\mathtt{isA}^{-}(z,x) \wedge \psi_1(x)) \wedge \forall x.\, \mathtt{isA}^{-}(z,x) \rightarrow \psi_2(x). \quad \square$$

7.2. Undecidability Results

In the remaining part of this section, we show the undecidability of the satisfiability problem for several fragments of SCL through a semi-conservative reduction from the standard *domino problem* [31, 4, 25], whose solution is known to be Π^1_0-complete. A $\mathbb{N}\times\mathbb{N}$ tiling system $\langle \mathrm{T}, H, V\rangle$ is a structure built on a non-empty set T of domino types, *a.k.a.* tiles, and two horizontal and vertical matching relations $H, V \subseteq \mathrm{T}\times\mathrm{T}$. The domino problem asks for a compatible tiling of the first quadrant $\mathbb{N}\times\mathbb{N}$ of the discrete plane, *i.e.*, a solution mapping $\eth\colon \mathbb{N}\times\mathbb{N} \rightarrow \mathrm{T}$ such that, for all $x, y \in \mathbb{N}$, both $(\eth(x,y), \eth(x+1,y)) \in H$ and $(\eth(x,y), \eth(x,y+1)) \in V$ hold true.

Theorem 14. *The sentence satisfiability problems of the non-recursive* SO*,* SAC*,* SEC*,* SEO′*, and* SZAE *fragments of* SCL *are undecidable.*

Proof. The main idea behind the proof is to embed a tiling system into a model of a particular SCL sentence φ that is satisfiable *iff* 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 carried out in the base language $\varnothing$.

Independently of the fragment we choose to prove undecidable, consider the following definition:

$$\varphi \triangleq \Big(\bigvee_{t \in \mathrm{T}} \mathtt{isA}(0, t) \Big) \wedge \Big(\bigwedge_{t \in \mathrm{T}} \forall x.\, \mathtt{isA}(x, t) \rightarrow (\psi_T^t(x) \wedge \psi_G(x)) \Big).$$

Intuitively, the first conjunct ensures the existence of the point 0, *i.e.*, the origin of the grid, labelled by some arbitrary tile in the set T. Notice that T is lifted to a set of constants in SCL. The second conjunct, then, states that all points x, labelled by some tile t, need to satisfy the properties expressed by the two monadic formulae $\psi_T^t(x)$ and $\psi_G(x)$. The first one, called *tiling formula*, is used to ensure the admissibility of the tiling, while the second one, called *grid formula*, forces all models of φ to necessarily embed a grid.

$$\begin{aligned}\psi_T^t(x) \triangleq {} & \bigwedge_{t' \in \mathrm{T}}^{t' \neq t} \neg \mathtt{isA}(x, t') \\ & \wedge \left(\forall y.\, \mathtt{H}(x, y) \rightarrow \bigvee_{(t,t') \in H} \mathtt{isA}(y, t') \right) \wedge \left(\forall y.\, \mathtt{V}(x, y) \rightarrow \bigvee_{(t,t') \in V} \mathtt{isA}(y, t') \right).\end{aligned}$$

The first conjunct of the tiling formula $\psi_T^t(x)$ verifies that the point associated with the argument x is labelled by no other tile than t itself. The second part, instead, ensures that the points y on the right or above of x are labelled by some tile t' which is compatible with t, *w.r.t.* the constraints imposed by the horizontal H and vertical V matching relations, respectively. Notice here that the relation symbols H and V are the syntactic counterpart of H and V, respectively.

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

$$\psi_G(x) \triangleq (\exists y.\, \mathtt{H}(x, y)) \wedge (\exists y.\, \mathtt{V}(x, y)) \wedge \gamma(x).$$

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 SCL, needs to enforce the fact that x is the origin of a square. In other words, this means that, going horizontally and then vertically or, *vice versa*, vertically and then horizontally, the same point is reached. To do this, we make use of the two S-path-formulae $\pi_{\mathtt{HV}}(x, y) \triangleq \exists z.\, (\mathtt{H}(x, z) \wedge \mathtt{V}(z, y))$ and $\pi_{\mathtt{VH}}(x, y) \triangleq \exists z.\, (\mathtt{V}(x, z) \wedge \mathtt{H}(z, y))$. In some cases, we also consider the S A-path-formula $\pi_{\mathtt{D}}(x, y) \triangleq \pi_{\mathtt{HV}}(x, y) \vee \pi_{\mathtt{VH}}(x, y)$ combining the previous ones, which implicitly define a diagonal relation. We now proceed by a case analysis on the specific fragment.

- **[S0]** 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 express the fact that all points reachable through $\pi_{\mathtt{HV}}$ or $\pi_{\mathtt{VH}}$ are, actually, the same unique point:

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

The S O-formula $\gamma(x)$ ensures that the relation D is both non-empty and functional and that all points reachable via $\pi_{\mathtt{HV}}$ or $\pi_{\mathtt{VH}}$ are necessarily the single one reachable through D.

- **[SAC]** By applying a counting quantifier to the formula $\pi_{\mathtt{D}}$, which encodes the union of the points reachable through $\pi_{\mathtt{HV}}$ or $\pi_{\mathtt{VH}}$, we can ensure the existence of a single diagonal point:
$$\gamma(x) \triangleq \neg\exists^{\geq 2} y.\, \pi_{\mathtt{D}}(x,y).$$

- **[SEC]** As for the S O fragment, here we use a diagonal relation D, which needs to contain all and only the points reachable via $\pi_{\mathtt{HV}}$ or $\pi_{\mathtt{VH}}$. By means of the counting quantifier, we enforce its functionality:
$$\begin{aligned}\gamma(x) \triangleq\ & \neg\exists^{\geq 2} y.\, \mathtt{D}(x,y) \\ & \wedge \forall y.\, \pi_{\mathtt{HV}}(x,y) \leftrightarrow \mathtt{D}(x,y) \\ & \wedge \forall y.\, \pi_{\mathtt{VH}}(x,y) \leftrightarrow \mathtt{D}(x,y).\end{aligned}$$

- **[SEO′]** This case is similar to the previous one, where the functionality of D is obtained by means of the O' construct:
$$\begin{aligned}\gamma(x) \triangleq\ & \forall y,z.\, \mathtt{D}(x,y) \wedge \mathtt{D}(x,z) \rightarrow y \leq z \\ & \wedge \forall y.\, \pi_{\mathtt{HV}}(x,y) \leftrightarrow \mathtt{D}(x,y) \\ & \wedge \forall y.\, \pi_{\mathtt{VH}}(x,y) \leftrightarrow \mathtt{D}(x,y).\end{aligned}$$

- **[SZAE]** The proof for this final case is inspired by the one proposed for the undecidability of the guarded fragment extended with transitive closure of binary relations [13]. This time, the functionality of the diagonal relation D is indirectly ensured by the conjunction of the four formulae $\gamma_1(x)$, $\gamma_2(x)$, $\gamma_3(x)$, and $\gamma_4(x)$ that exploit all the features of the fragment:
$$\begin{aligned}\gamma(x) \triangleq\ & \gamma_1(x) \wedge \gamma_2(x) \wedge \gamma_3(x) \wedge \gamma_4(x) \\ & \wedge \forall y.\, \pi_{\mathtt{D}}(x,y) \leftrightarrow \mathtt{D}(x,y),\end{aligned}$$

where

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

$$\gamma_2(x) \triangleq \left(\bigvee_{i\in\{0,1\}} \neg\exists y.\, \mathtt{D}_i(x,y) \right) \wedge \left(\bigwedge_{i\in\{0,1\}} \forall y.\, \mathtt{D}_i(x,y) \rightarrow \exists z.\, \mathtt{D}_{1-i}(y,z) \right),$$

$$\gamma_3(x) \triangleq \bigwedge_{i\in\{0,1\}} \forall y.\, \big(x = y \vee \mathtt{D}_i(x,y) \vee \mathtt{D}_i^{-}(x,y)\big) \leftrightarrow \mathtt{E}_i(x,y), \text{ and}$$

$$\gamma_4(x) \triangleq \bigwedge_{i\in\{0,1\}} \forall y.(\exists z.\, (\mathtt{E}_i(x,z) \wedge \mathtt{E}_i(z,y))) \leftrightarrow \mathtt{E}_i(x,y).$$

Intuitively, γ_1 asserts that $\mathtt{D}$ is the union of the two accessory relations $\mathtt{D}_0$ and $\mathtt{D}_1$, while γ_2 guarantees that a point can only have adjacents *w.r.t.* just one relation $\mathtt{D}_i$ and that these adjacents can only appear as first argument of the opposite relation $\mathtt{D}_{1-i}$. In addition, γ_3 ensures that the additional relation $\mathtt{E}_i$ is the reflexive symmetric closure of $\mathtt{D}_i$ and γ_4 forces $\mathtt{E}_i$ to be transitive too.

We can now prove that the relation $\mathtt{D}$ 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 $\mathtt{D}(a,b)$ and $\mathtt{D}(a,c)$ hold true. By the formula γ_1 and the first conjunct of γ_2, we have that $\mathtt{D}_i(a,b)$ and $\mathtt{D}_i(a,c)$ 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 $\mathtt{D}_i(b,c)$ nor $\mathtt{D}_i(c,b)$ can hold. Indeed, if $a = b$ then $\mathtt{D}_i(a,a)$. This in turn implies $\mathtt{D}_{1-i}(a,d)$ 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 $\mathtt{D}_i(b,c)$ holds, then $\mathtt{D}_{1-i}(c,d)$ needs to hold as well, for some value d, leading again to a contradiction. Now, by the formula γ_3, both $\mathtt{E}_i(b,a)$ and $\mathtt{E}_i(a,c)$ hold, but $\mathtt{E}_i(b,c)$ does not. However, this clearly contradicts γ_4. As a consequence, $\mathtt{D}$ is necessarily functional.

Now, it is not hard to see that the above sentence φ (one for each fragment) is satisfiable *iff* the domino instance on which the reduction is based on is solvable. Indeed, on the one hand, every compatible tiling $\eth\colon \mathbb{N} \times \mathbb{N} \rightarrow \mathrm{T}$ of a tiling system $\langle \mathrm{T}, H, V \rangle$ induces a grid model that trivially satisfies φ. On the other hand, a model of φ necessarily embed a grid whose points are labelled by tiles satisfying the horizontal and vertical relations. ☐

Theorem 15. *The formula satisfiability problems of the non-recursive* STO*,* SATC*,* STEC*,* STEO′*, and* SZATE *fragments of* SCL *are undecidable.*

Proof. The proof of this theorem builds on top of the one of the previous result, by showing that, with the addition of the transitive closure operator, we can encode the solution of a domino problem as the existence of a constant satisfying the following SCL formula $\psi(x)$, where the relation symbols $\mathtt{H}$ and $\mathtt{V}$ and the tiling and grid formulae ψ_T^t

and ψ_G are defined as in Theorem 14:

$$\psi(x) \triangleq \left(\bigvee_{t \in \mathrm{T}} \mathtt{isA}(x, t)\right)$$
$$\wedge\, \forall y. \left((\exists z. (\mathtt{H}(x, z))^* \wedge (\mathtt{V}(z, y))^*) \to \left(\bigwedge_{t \in \mathrm{T}} \mathtt{isA}(y, t) \to (\psi_T^t(y) \wedge \psi_G(y))\right)\right).$$

Intuitively, the formula $\psi(x)$ is satisfied by a constant c if this element is labelled by a tile in T and every other element y, reachable from c via an arbitrary numbers of horizontal steps followed by another arbitrary number of vertical steps, satisfies both the tiling and grid formulae. Obviously, $\psi(x)$ is satisfied at the root of a grid model induced by a compatible tiling $\eth\colon \mathbb{N} \times \mathbb{N} \to \mathrm{T}$ of a tiling system $\langle \mathrm{T}, H, V \rangle$. Indeed, every node in the grid is reachable from the root by following a *first-horizontal then-vertical path.* Moreover, its labelling is coherent with what is prescribed by the two matching relations H and V, so, $\psi_T^t(y)$ necessarily holds at every node of the grid. *Vice versa*, every structure satisfying $\psi(c)$ induces a compatible tiling, as the set of elements reachable from c form a grid, due to the formula ψ_G, and are suitably labelled thanks to the formula ψ_T^t. $\square$

8. Conclusion

In this article we have studied the satisfiability and containment problems for SHACL documents and shape constraints. In order to do so, we examined several recursive semantics proposed in the literature and proved that they all coincide for non-recursive documents. As well, we proved that one can focus only on total assignments semantics since partial assignments semantics reduces to it. We then provided a complete translation between: (1) non-recursive SHACL and SCL, a new fragment of first-order logic extended with counting quantifiers and transitive closure, (2) recursive SHACL and MSCL, an extension of SCL into a monadic second-order logic, where shape names become monadic second-order variables. These translations into mathematical logic are effective since, firstly, they offer a standard framework to model the language, contrary to previous *ad hoc* modellings, and, secondly, they allow us to study several formal properties: from capturing the semantics of filters (that have not been addressed in literature before), to laying out a detailed map of SHACL fragments for which we are able to prove (un)decidability along with complexity results, for our decision problems. We also expose semantic properties and asymmetries within SHACL which might inform a future update of the W3C language specification. Although the satisfiability and containment problems are both undecidable for the full SHACL, decidability can be achieved by restricting the usage of certain SHACL components, such as cardinality restrictions over shape or path properties. Nevertheless, the status of some weak fragments of SHACL, such as `O`, `S C`, and `S E`, remains an open question for further investigation.

Acknowledgements

G. Konstantinidis has been partially supported by an Alan Turing Fellowship, a Turing Enhancement Project and a UoS Web Science Institute Stimulus Fund. F. Mogavero has been partially supported by the GNCS 2020 project "Ragionamento Strategico e Sintesi Automatica di Sistemi Multi-Agente".

References

[1] Amarilli, A., Benedikt, M., Bourhis, P., Boom, M.V., 2016. Query Answering with Transitive and Linear-Ordered Data., in: International Joint Conference on Artificial Intelligence'16, International Joint Conference on Artificial Intelligence' & AAAI Press. pp. 893–899.

[2] Andresel, M., Corman, J., Ortiz, M., Reutter, J., Savković, O., Simkus, M., 2020. Stable Model Semantics for Recursive SHACL., in: WWW'20, pp. 1570–1580.

[3] Baader, F., Calvanese, D., McGuinness, D., Nardim, D., Patel-Scheider, P., 2003. The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press.

[4] Berger, R., 1966. The Undecidability of the Domino Problem. Memoirs of the American Mathematical Society 66, 1–72.

[5] Carothers, G., Prud'hommeaux, E., 2014. RDF 1.1 Turtle. W3C Recommendation. W3C. https://www.w3.org/TR/2014/REC-turtle-20140225/.

[6] ten Cate, B., Segoufin, L., 2011. Unary Negation., in: Symposium on Theoretical Aspects of Computer Science'11, Leibniz-Zentrum fuer Informatik. pp. 344–355.

[7] ten Cate, B., Segoufin, L., 2013. Unary Negation. Logical Methods in Computer Science 9, 1–46.

[8] Corman, J., Reutter, J., Savković, O., 2018. Semantics and Validation of Recursive SHACL., in: ISWC'18, pp. 318–336.

[9] Cyganiak, R., Wood, D., Lanthaler, M., 2014. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation. W3C. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[10] Danielski, D., Kieronski, E., 2019. Finite Satisfiability of Unary Negation Fragment with Transitivity., in: Mathematical Foundations of Computer Science'19, Leibniz-Zentrum fuer Informatik. pp. 17:1–15.

[11] Donini, F., Massacci, F., 2000. ExpTime Tableaux for $\mathcal{ALC}$. Artificial Intelligence 124, 87–138.

[12] Ebbinghaus, H.D., Flum, J., 1995. Finite Model Theory. Perspectives in Mathematical Logic., Springer.

[13] Grädel, E., 1999. On The Restraining Power of Guards. Journal of Symbolic Logic 64, 1719–1742.

[14] Grädel, E., Kolaitis, P., Vardi, M., 1997a. On the Decision Problem for Two-Variable First-Order Logic. Bulletin of Symbolic Logic 3, 53–69.

[15] Grädel, E., Otto, M., Rosen, E., 1997b. Two-Variable Logic with Counting is Decidable., in: Logic in Computer Science'97, IEEE Computer Society. pp. 306–317.

[16] Jung, J., Lutz, C., Martel, M., Schneider, T., 2018. Querying the Unary Negation Fragment with Regular Path Expressions., in: International Conference on Database Theory'18, OpenProceedings.org. pp. 15:1–18.

[17] Knublauch, H., Kontokostas, D., 2017. Shapes constraint language (SHACL). W3C Recommendation. W3C. https://www.w3.org/TR/shacl/.

[18] Krötzsch, M., Simančík, F., Horrocks, I., 2012. A Description Logic Primer. Technical Report. arXiv.

[19] Leinberger, M., Seifer, P., Rienstra, T., Lämmel, R., Staab, S., 2020. Deciding SHACL Shape Containment through Description Logics Reasoning, in: The Semantic Web – ISWC 2020, Springer International Publishing. pp. 366–383. (*this volume*).

[20] Lutz, C., 2005. An Improved NExpTime-Hardness Result for Description Logic $\mathcal{ALC}$ Extended with Inverse Roles, Nominals, and Counting. Technical Report 05-05. Dresden University of Technology, Dresden, Germany.

[21] Mortimer, M., 1975. On Languages with Two Variables. Mathematical Logic Quarterly 21, 135–140.

[22] Pareti, P., Konstantinidis, G., Mogavero, F., Norman, T.J., 2020. Shacl satisfiability and containment, in: The Semantic Web – ISWC 2020, Springer International Publishing, Cham. pp. 474–493.

[23] Pratt-Hartmann, I., 2005. Complexity of the Two-Variable Fragment with Counting Quantifiers. Journal of Logic, Language, and Information' 14, 369–395.

[24] Pratt-Hartmann, I., 2010. The Two-Variable Fragment with Counting Revisited., in: Workshop on Logic, Language, Information and Computation'10, Springer. pp. 42–54.

[25] Robinson, R., 1971. Undecidability and Nonperiodicity for Tilings of the Plane. Inventiones Mathematicae 12, 177–209.

[26] Rudolph, S., Krötzsch, M., Hitzler, P., 2008. Cheap Boolean Role Constructors for Description Logics., in: European Conference on Logics in Artificial Intelligence'08, Springer. pp. 362–374.

[27] Sattler, U., Vardi, M., 2001. The Hybrid μ-Calculus., in: International Joint Conference on Automated Reasoning'01, Springer. pp. 76–91.

[28] Schild, K., 1991. A Correspondence Theory for Terminological Logics: Preliminary Report., in: International Joint Conference on Artificial Intelligence'91, Morgan Kaufmann. pp. 466–471.

[29] Tobies, S., 2000. The Complexity of Reasoning with Cardinality Restrictions and Nominals in Expressive Description Logics. Journal of Artificial Intelligence Research 12, 199–217.
[30] W3C OWL Working Group, 2012. OWL 2 Web Ontology Language Document Overview (Second Edition). W3C Recommendation. W3C. Https://www.w3.org/TR/2012/REC-owl2-overview-20121211/.
[31] Wang, H., 1961. Proving Theorems by Pattern Recognition II. Bell System Technical Journal 40, 1–41.

Appendix A. Translation from SHACL into SCL

In this section we present our translation $\tau(M)$ from a SHACL document M (a set of SHACL shape definitions) into sentences of our SCL grammar. For the sake of completeness, we define our translation τ, for any SHACL document. However, it should be noted that within SHACL, the same constraint can sometimes be expressed with syntactically different, but semantically equivalent expressions. This syntactic detail is not relevant to our analysis of SHACL, which is focused on semantics. Nevertheless, our formulation of Theorem 3 requires us to work with a "standardised" syntactic representation of SHACL documents. Thus, we restrict ourselves to *standardised* SHACL documents, that we will define next. In essence, a standardised SHACL document restricts the usage of these syntactic variations without affecting generality. Given any SHACL document, it is always possible to transform it into a standardised one in linear size and time.

Definition 20. *A* standardised SHACL *document is a SHACL document that has the following properties:*

1. *all shape names are identified by IRIs (instead of blank nodes);*
2. *it does not contain the following terms:* `sh:qualifiedValueShapesDisjoint`*,* `sh:in`*,* `sh:class`*,* `sh:minCount`*,* `sh:maxCount`*,* `sh:qualifiedMaxCount`*,* `sh:and`*,* `sh:or`*, and* `sh:xone`*;*
3. *it does not contain triples with a subject that is a property shape, and a predicate that is one of the following:* `sh:hasValue`*,* `sh:datatype`*,* `sh:nodeKind`*,* `sh:pattern`*,* `sh:node`*,* `sh:property`*,* `sh:minExclusive`*,* `sh:minInclusive`*,* `sh:maxExclusive`*,* `sh:maxInclusive`*,* `sh:maxLength`*,* `sh:minLength` *and* `sh:not`*;*
4. *triples with* `sh:languageIn` *as the property contain a list with a single element as the object;*

The translation into SCL grammar of a document M is $\bigwedge_{\mathtt{s}\in M}\tau(\mathtt{s})$, where $\tau(\mathtt{s})$ is the translation of a single SHACL shape $\mathtt{s}$ in M. Given a shape $\langle \mathtt{s}, t, d\rangle$, its translation $\tau(\langle \mathtt{s}, t, d\rangle)$ the following, where $\tau_{t,\mathtt{s}}$ and $\tau_{d,\mathtt{s}}$ are, respectively, the target and constraint axioms of the shape.

$$\tau(\langle \mathtt{s}, t, d\rangle) = \tau_{t,\mathtt{s}} \wedge \tau_{d,\mathtt{s}}$$

The translation of axiom $\tau_{t,\mathtt{s}}$ is defined in Table 2 if t is not empty, or else it is $\top$. We do not discuss implicit class-based targets, as they just represent a syntactic variant of class targets. The translation $\tau_{d,\mathtt{s}}$ is the following, where $\tau(x, \mathtt{s}, d)$ is the unary formula that models the constraints d of shape $\mathtt{s}$.

$$\tau_{d,\mathtt{s}} = \Sigma_{\mathtt{s}}(x) \leftrightarrow \tau(x, \mathtt{s}, d)$$

In the reminder of this section we define how to compute $\tau(x, \mathtt{s}, d)$. The constraints of d of a shape $\mathtt{s}$ in a SHACL document M, is the set of triples that (1) have $\mathtt{s}$ as the subject in the RDF graph representing M, or (2) define property paths or lists of elements. As convention, we use $\mathtt{c}$ as an arbitrary constant and C as an arbitrary list of constants. We use $\mathtt{s}$, $\mathtt{s}'$ 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 $\tau(x, \mathtt{s}, d)$ 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. Recall that we use notation $<s, p, o>$ to represent an RDF triple with subject S, predicate p and object o.

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

Next, we define the translations τ_1 of node shapes triples, τ_2 of property shape triples and τ_3 of property paths.

Appendix A.1. Translation of Node Shape Triples

The translation of $\tau_1(x, <\mathtt{s}, y, z>)$ is split in the following cases, depending on the predicate of the triple. In case none of those cases are matched $\tau_1(x, <\mathtt{s}, y, z>) \doteq \top$. 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 $\tau(x, \mathtt{s})$.

- $\tau_1(x, <\mathtt{s}, \mathtt{sh{:}hasValue}, \mathtt{c}>) \doteq x = \mathtt{c}$.
- $\tau_1(x, <\mathtt{s}, \mathtt{sh{:}in}, C>) \doteq \bigvee_{\mathtt{c} \in C} x = \mathtt{c}$.
- $\tau_1(x, <\mathtt{s}, \mathtt{sh{:}class}, \mathtt{c}>) \doteq \exists y. \mathtt{isA}(x, y) \wedge y = \mathtt{c}$.
- $\tau_1(x, <\mathtt{s}, \mathtt{sh{:}datatype}, \mathtt{c}>)) \doteq \mathrm{F}^{\,\mathrm{dt}=\mathtt{c}}(x)$.
- $\tau_1(x, <\mathtt{s}, \mathtt{sh{:}nodeKind}, \mathtt{c}>) \doteq \mathrm{F}^{\,\mathrm{IRI}}(x)$ 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.
- $\tau_1(x, <\mathtt{s}, \mathtt{sh{:}minExclusive}, \mathtt{c}>) \doteq x > \mathtt{c}$ if order is an interpreted relation, else $\mathrm{F}^{\,>\mathtt{c}}(x)$.
- $\tau_1(x, <\mathtt{s}, \mathtt{sh{:}minInclusive}, \mathtt{c}>) \doteq x \geq \mathtt{c}$ if order is an interpreted relation, else $\mathrm{F}^{\,\geq\mathtt{c}}(x)$.
- $\tau_1(x, <\mathtt{s}, \mathtt{sh{:}maxExclusive}, \mathtt{c}>) \doteq x < \mathtt{c}$ if order is an interpreted relation, else $\mathrm{F}^{\,<\mathtt{c}}(x)$.
- $\tau_1(x, <\mathtt{s}, \mathtt{sh{:}maxInclusive}, \mathtt{c}>) \doteq x \leq \mathtt{c}$ if order is an interpreted relation, else $\mathrm{F}^{\,\leq\mathtt{c}}(x)$.
- $\tau_1(x, <\mathtt{s}, \mathtt{sh{:}maxLength}, \mathtt{c}>) \doteq \mathrm{F}^{\,\mathrm{maxLength}=\mathtt{c}}(x)$.
- $\tau_1(x, <\mathtt{s}, \mathtt{sh{:}minLength}, \mathtt{c}>) \doteq \mathrm{F}^{\,\mathrm{minLength}=\mathtt{c}}(x)$.

- $\tau_1(x, \texttt{<s}, \texttt{sh:pattern}, \texttt{c>}) \doteq \mathrm{F}^{\,\text{pattern}=\texttt{c}}(x)$.
- $\tau_1(x, \texttt{<s}, \texttt{sh:languageIn}, C\texttt{>}) \doteq \bigvee_{\texttt{c} \in C} F^{\text{languageTag = }\texttt{c}}(x)$.
- $\tau_1(x, \texttt{<s}, \texttt{sh:not}, \texttt{s}'\texttt{>}) \doteq \neg\Sigma_{\texttt{s}'}(x)$.
- $\tau_1(x, \texttt{<s}, \texttt{sh:and}, \bar{S}\texttt{>}) \doteq \bigwedge_{\texttt{s}' \in \bar{S}} \Sigma_{\texttt{s}'}(x)$.
- $\tau_1(x, \texttt{<s}, \texttt{sh:or}, \bar{S}\texttt{>}) \doteq \bigvee_{\texttt{s}' \in \bar{S}} \Sigma_{\texttt{s}'}(x)$.
- $\tau_1(x, \texttt{<s}, \texttt{sh:node}, \texttt{s}'\texttt{>}) \doteq \Sigma_{\texttt{s}'}(x)$.
- $\tau_1(x, \texttt{<s}, \texttt{sh:property}, \texttt{s}'\texttt{>}) \doteq \Sigma_{\texttt{s}'}(x)$.

Appendix A.2. Translation of Property Shapes

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

- $\tau_2(x, r, \texttt{<s}, \texttt{sh:hasValue}, \texttt{c>}) \doteq \exists y.r(x, y) \wedge \tau_1(y, \texttt{<s}, \texttt{sh:hasValue}, \texttt{c>})$
- $\tau_2(x, r, \texttt{<s}, p, \texttt{c>}) \doteq \forall y.\tau_3(x, r, y)) \to \tau_1(y, \texttt{<s}, p, \texttt{c>})$, 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` .
- $\tau_2(x, r, \texttt{<s}, \texttt{sh:languageIn}, C\texttt{>}) \doteq$
  $\forall y.\tau_3(x, r, y)) \to \tau_1(y, \texttt{<s}, \texttt{sh:languageIn}, C\texttt{>})$.
- $\tau_2(x, r, \texttt{<s}, \texttt{sh:uniqueLang}, \text{true}\texttt{>}) \doteq \bigwedge_{\texttt{c} \in L} \neg\exists^{\geq 2} y.r(x, y) \wedge F^{\text{lang}=c}(y)$
  where $L = \{\texttt{c} | \texttt{c} \in C \wedge \exists \texttt{s}^{'}.\texttt{<s}^{'}, \texttt{sh:languageIn}, C\texttt{>} \in M\} \cup \{c^{\text{uniqueL}}\}$ and c^{uniqueL} is a fresh unique constant. This translation is possible because `sh:languageIn` is the only constraint that can force language tags constraints on literals.
- $\tau_2(x, r, \texttt{<s}, \texttt{sh:minCount}, \texttt{c>}) \doteq \exists^{\geq \texttt{c}} y.\tau_3(x, r, y))$.
- $\tau_2(x, r, \texttt{<s}, \texttt{sh:maxCount}, \texttt{c>}) \doteq \neg\exists^{\leq \texttt{c}} y.\tau_3(x, r, y))$.
- $\tau_2(x, r, \texttt{<s}, \texttt{sh:equals}, \texttt{c>}) \doteq \forall y.\tau_3(x, r, y) \leftrightarrow \tau_3(x, \texttt{c}, y)$.
- $\tau_2(x, r, \texttt{<s}, \texttt{sh:disjoint}, \texttt{c>}) \doteq \neg\exists y.\tau_3(x, r, y) \wedge \tau_3(x, \texttt{c}, y)$.
- $\tau_2(x, r, \texttt{<s}, \texttt{sh:lessThan}, \texttt{c>}) \doteq \forall y, z.\tau_3(x, r, y) \wedge \tau_3(x, \texttt{c}, z) \to y < z$.
- $\tau_2(x, r, \texttt{<s}, \texttt{sh:lessThanOrEquals}, \texttt{c>}) \doteq \forall y, z \,.\, \tau_3(x, r, y) \wedge \tau_3(x, \texttt{c}, z) \to y \leq z$.
- $\tau_2(x, r, \texttt{<s}, \texttt{sh:qualifiedValueShape}, \texttt{s}'\texttt{>}) \doteq \alpha \wedge \beta$, where α and β are defined as follows. Let $S^{'}$ be the set of *sibling shapes* of `s` if M contains <`s`, `sh:qualifiedValueShapesDisjoint`, true>, or the empty set otherwise.
  Let $\nu(x) = \Sigma_{\texttt{s}'}(x) \bigwedge_{\forall \texttt{s}'' \in S'} \neg\Sigma_{\texttt{s}''}(x)$. If triple $\texttt{<s}, \texttt{sh:qualifiedMinCount}, \texttt{c>}$ is contained in M, then α is equal to $\exists^{\geq \texttt{c}} y.\tau_3(x, r, y) \wedge \nu(x)$, otherwise α is equal to $\top$. If M contains the triple $\texttt{<s}, \texttt{sh:qualifiedMaxCount}, \texttt{c>}$, then β is equal to $\neg\exists^{\leq \texttt{c}} y.\tau_3(x, r, y) \wedge \nu(x)$, otherwise β is equal to $\top$.

- $\tau_2(x, r, <\texttt{s}, \texttt{sh:close}, \text{true}>) \doteq \bigwedge_{\forall R \in \Theta} \neg\exists y.R(x, y)$ if Θ is not empty, or else $\top$, where Θ is defined as follows. Let Θ^{all} be the set of all relation names in M, namely $\Theta^{\text{all}} = \{R | <x, R, y> \in M\}$. 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 $\Theta^{\text{declared}} = \{R | \{<\texttt{s}, \texttt{sh:property}, x> \wedge <x, \texttt{sh:path}, R)>\} \subseteq M\}$. Let $R^{\texttt{closed}}$ be a unique fresh relation name, Θ^{ignored} be the set of all the binary property names declared as "ignored" properties, namely $\Theta^{\text{ignored}} = \{R | R \in \bar{R} \wedge <\texttt{s}, \texttt{sh:ignoredProperties}, \bar{R}> \in M\}$, where $\bar{R}$ is a list of IRIs. The set Θ can now be defined as $\Theta = (\Theta^{\text{all}} \cup \{R^{\texttt{closed}}\}) \setminus (\Theta^{\text{declared}} \cup \Theta^{\text{ignored}})$.

Appendix 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 ˆ, 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 $\tau_3(x, r, y)) \doteq x = y \vee \tau_3(x, r_1, y))$

Appendix B. Translation from SCL into SHACL

In this section we present the translation τ^{-}, inverse of τ, from sentences in the SCL grammar into SHACL documents. We begin by defining the translation of the property path subgrammar $r(x, y)$ into SHACL property paths:

- $\tau^{-}(R) \doteq R$

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

The translation of the constraint subgrammar $\psi(x)$ is the following. we will use $\tau^-(\psi(x))$ to denote the SHACL translation of shape $\psi(x)$, and $\iota(\tau^-(\psi(x)))$ to denote the IRI corresponding to its shape name. 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 $\{\langle s, \texttt{rdf:type}, \texttt{sh:NodeShape}\rangle, \langle s, \texttt{sh:hasValue}, c\rangle\}$. When alternative translations are possible, the one listed first takes precedence. In other words, a translation in the following list is applied to a formula only if no translation listed before it are applicable.

- $\tau^-(\top) \doteq$
  s `a sh:NodeShape .`
- $\tau^-(x = \texttt{c}) \doteq$
  s `a sh:NodeShape ;`
  `sh:hasValue c .`
- $\tau^-\left(\bigwedge_{\texttt{c}\in L} \neg\exists^{\geq 2} y.r(x,y) \wedge F^{\text{lang}=c}(y)\right)$, where $c^{\texttt{uniqueL}} \in L$
  s `a sh:PropertyShape ;`
  `sh:path r ;`
  `sh:uniqueLang true .`
- $\tau^-(F(x)) \doteq$
  s `a sh:NodeShape ;`
  f C `.`
  Predicate f and the RDF term C 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`. Depending on the type of the filter, C could be a literal, an IRI, or an RDF list with a single element.
- $\tau^-(\Sigma_{\texttt{s}'}(x)) \doteq$
  s `a sh:NodeShape ;`
  `sh:node s′ .`
  if $\texttt{s}'$ is a node shape, else:
  s `a sh:NodeShape ;`
  `sh:property s′ .`
- $\tau^-\left(\bigwedge_{\forall R\in\Theta} \neg\exists y.R(x,y)\right)$ where Θ is a set of property relation names that includes $R^{\texttt{closed}}$, and $\Theta^{\texttt{list}}$ is the RDF list representation of all the property relation names

in the SCL formula that are not in Θ
s `a sh:PropertyShape ;`
  `sh:close true ;`
  `sh:ignored` $\Theta^{\texttt{list}}$`.`

- $\tau^-(\neg\psi(x)) \doteq$
  s `a sh:NodeShape ;`
  `sh:not` $\iota(\tau^-(\psi(x)))$ `.`

- $\tau^-(\psi_1(x) \wedge \psi_2(x)) \doteq$
  s `a sh:NodeShape ;`
  `sh:and (`$\iota(\tau^-(\psi_1(x)))$`,` $\iota(\tau^-(\psi_2(x)))$`)` `.`

- $\tau^-(\exists^{\geq n} y.r(x,y) \wedge \psi(y)) \doteq$
  s `a sh:PropertyShape ;`
  `sh:path` $\tau^-(r(x,y))$ `;`
  `sh:qualifiedValueShape` $\iota(\tau^-(\psi(y)))$ `;`
  `sh:qualifiedMinCount` n `.`

- $\tau^-(\forall y.r(x,y) \leftrightarrow R(x,y)) \doteq$
  s `a sh:PropertyShape ;`
  `sh:path` $\tau^-(r(x,y))$ `;`
  `sh:equals` R `.`

- $\tau^-(\neg\exists y.r(x,y) \wedge R(x,y)) \doteq$
  s `a sh:PropertyShape ;`
  `sh:path` $\tau^-(r(x,y))$ `;`
  `sh:disjoint` R `.`

- $\tau^-(\forall y,z.r(x,y) \wedge R(x,z) \rightarrow y < z) \doteq$
  s `a sh:PropertyShape ;`
  `sh:path` $\tau^-(r(x,y))$ `;`
  `sh:lessThan` R `.`

- $\tau^-(\forall y,z.r(x,y) \wedge R(x,z) \rightarrow y \leq z) \doteq$
  s `a sh:PropertyShape ;`
  `sh:path` $\tau^-(r(x,y))$ `;`
  `sh:lessThanOrEquals` R `.`

We now define the translation $\tau^-(\varphi)$ of a complete sentence of the φ-grammar into a SHACL document M as follows.

- $\tau^-(\varphi_1 \wedge \varphi_2) \doteq \tau^-(\varphi_1) \cup \tau^-(\varphi_2))$
- $\tau^-(\Sigma_{\mathtt{s}}(\mathtt{c})) \doteq s$ `sh:targetNode c.`
- $\tau^-(\forall x.\ \mathtt{isA}(x,\mathtt{c}) \rightarrow \Sigma_{\mathtt{s}}(x)) \doteq s$ `sh:targetClass c.`
- $\tau^-(\forall x,y.\ R(x,y) \rightarrow \Sigma_{\mathtt{s}}(x)) \doteq s$ `sh:targetSubjectsOf` R `.`
- $\tau^-(\forall x,y.\ R^-(x,y) \rightarrow \Sigma_{\mathtt{s}}(x)) \doteq s$ `sh:targetObjectsOf` R `.`
- $\tau^-\big(\Sigma_{\mathtt{s}}(x) \leftrightarrow \psi(x)\big) \doteq \tau^-(\psi_1(x))$ such that $\mathtt{s} = \iota(\tau^-(\psi_1(x)))$.