KRDB Research Centre Technical Report:

Semantics and Validation of Recursive SHACL
[Technical Report]

Julien Corman¹, Juan L. Reutter², Ognjen Savković¹

Affiliation
1: Free University of Bozen-Bolzano, Bolzano, Italy
2: PUC Chile and Center for Semantic Web Research, Santiago, Chile,

Corresponding author
Julien Corman: julienlouismichel.corman@unibz.it
Juan L. Reutter: jreutter@ing.puc.cl
Ognjen Savković: ognjen.savkovic@unibz.it

Keywords
SHACL, RDF Graph Validation, Graph constraints,
RDF, Recursive constraints

Number
KRDB18-01

Date
April 7, 2018

URL
http://www.inf.unibz.it/krdb/
Abstract

With the popularity of RDF as an independent data model came the need for specifying constraints on RDF graphs, and for mechanisms to detect violations of such constraints. One of the most promising schema languages for RDF is SHACL, a recent W3C recommendation. Unfortunately, the specification of SHACL leaves open the problem of validation against recursive constraints. This omission is important because SHACL by design favors constraints that reference other ones, which in practice may easily yield reference cycles.

In this paper, we propose a concise formal semantics for the so-called “core constraint components” of SHACL. This semantics handles arbitrary recursion, while being compliant with the current standard. Graph validation is based on the existence of an assignment of SHACL “shapes” to nodes in the graph under validation, stating which shapes are verified or violated, while verifying the targets of the validation process. We show in particular that the nature of SHACL forces us to consider cases in which these assignments are partial, or, in other words, where the truth value of a constraints at some nodes of a graph may be left unknown.

Dealing with recursion comes at a price, as validating an RDF graph against SHACL constraints is NP-hard in the size of the graph, and this lower bound still holds for a fragment of SHACL using stratified negation. Therefore we also propose a tractable approximation to the validation problem.

Acknowledgments

This work has been partially funded by the FUB projects QUEST and OBATS.
Semantics and Validation of Recursive SHACL

Julien Corman\(^1\), Juan L. Reutter\(^2\), and Ognjen Savković\(^1\)

\(^1\) Free University of Bozen-Bolzano, Bolzano, Italy
\(^2\) PUC Chile and Center for Semantic Web Research, Santiago, Chile

Abstract. With the popularity of RDF as an independent data model came the need for specifying constraints on RDF graphs, and for mechanisms to detect violations of such constraints. One of the most promising schema languages for RDF is SHACL, a recent W3C recommendation. Unfortunately, the specification of SHACL leaves open the problem of validation against recursive constraints. This omission is important because SHACL by design favors constraints that reference other ones, which in practice may easily yield reference cycles.

In this paper, we propose a concise formal semantics for the so-called “core constraint components” of SHACL. This semantics handles arbitrary recursion, while being compliant with the current standard. Graph validation is based on the existence of an assignment of SHACL “shapes” to nodes in the graph under validation, stating which shapes are verified or violated, while verifying the targets of the validation process. We show in particular that the nature of SHACL forces us to consider cases in which these assignments are partial, or, in other words, where the truth value of a constraints at some nodes of a graph may be left unknown.

Dealing with recursion comes at a price, as validating an RDF graph against SHACL constraints is NP-hard in the size of the graph, and this lower bound still holds for a fragment of SHACL using stratified negation. Therefore we also propose a tractable approximation to the validation problem.

1 Introduction

The success of RDF was largely due the fact that it can be easily published and queried without bounding to a specific schema [3]. But RDF over time has turned into more than a simple data exchange format [1], and a key challenge for current RDF-based applications is checking quality (correctness and completeness) of a dataset. Nowadays, several systems already provide facilities for RDF validation [11], including commercial products.\(^3\) This created a need for standardizing a declarative language for RDF constraints, and for formal mechanisms to detect and describe violations of such constraints.

One of the most promising efforts in this direction is SHACL, or Shapes Constraint Language,\(^6\) which has become a W3C recommendation in 2017. The idea of SHACL is to group constraints in so-called “shapes” to be verified by certain nodes of the graph under validation, and such that shapes may reference each other. Consequently, a set of

\(^3\) https://www.w3.org/2001/sw/wiki/ShEx#Implementations
\(^4\) https://www.topquadrant.com/technology/shacl/
\(^5\) https://www.stardog.com/docs/
\(^6\) https://www.w3.org/TR/shacl/
shapes can also be seen as a way of describing a set of RDF graphs; precisely those that can be validated against it.

Figure 1 presents two SHACL shapes. The leftmost, named :NIAddressShape, is meant to define valid addresses in Northern Italy, whereas the right one, named :PolentoneShape, defines northern Italians, stereotypically referred to as Polentoni.\footnote{This example is borrowed from Peter Patel-Schneider: https://research.nuance.com/wp-content/uploads/2017/03/shacl.pdf}

A node $v$ satisfying the first shape must verify two constraints: the first one states that there can be at most one successor of $v$ via property :telephone. The second one states that there must be exactly one successor ($sh:\minCount 1$ and $sh:\maxCount 1$) of $v$ via property :locatedIn, with value :NorthernItaly.

Validating an RDF graph against a set of shapes is based on the notion of “target nodes”, which mandates for each shape which nodes have to conform to it. For instance, :PolentoneShape contains the triple :PolentoneShape sh:targetClass :Polentone, stating that its targets are all instances of :Polentone in the graph under validation. But nodes may also have to conform to additional shapes, due to shape references. For instance, in Figure 1, the shape to the right contains one (non-recursive) shape reference, to :NIAddressShape, stating that every node $v$ conforming to :PolentoneShape must have exactly one :address, which must conform to :NIAddressShape, and one recursive reference, stating that each successor of $v$ via :knows must conform to :PolentoneShape.

By recursion, we will always refer to such reference cycles, possibly n-ary (where shape $s_1$ references $s_2$, $s_2$ references $s_3$, …, $s_n$ references $s_1$). Unfortunately, the semantics of graph validation with reference cycles is left explicitly undefined in the SHACL specification: …The validation with recursive shapes is not defined in SHACL and is left to SHACL processor implementations. For example, SHACL processors may support recursion scenarios or produce a failure when they detect recursion. The specification nonetheless expresses the expectation that validation of recursive shapes end up being defined in future work. Indeed, nested shapes are a core feature of SHACL, and their use is encouraged in examples throughout the specification. Furthermore, in a Semantic Web context, where shapes are expected to be exchanged or reused, reference cycles

\begin{verbatim}
:NIAddressShape
  a sh:NodeShape ;
  sh:property [ sh:path :telephone ; sh:maxCount 1 ] ;
  sh:property [ sh:path :locatedIn ; sh:minCount 1 ; sh:maxCount 1 ;
                 sh:hasValue :NorthernItaly ] .

:PolentoneShape
  a sh:NodeShape ;
  sh:targetClass :Polentone ;
  sh:property [ sh:path :address ; ] ;
  sh:property [ sh:path :knows ;
                 sh:node :NIAddressShape
                 sh:maxCount 1 ;
                 sh:property [
                   NorthernItaly sh:node :PolentoneShape ] ] .
\end{verbatim}

\textbf{Fig. 1.} Two SHACL shapes, about Polentoni and addresses in Northern Italy.
Semantics and Validation of Recursive SHACL

may naturally appear, intentional or not. Finally, recursion may be viewed as one of the distinctive features of SHACL: without recursion, one ends up with a constraint language whose expressive power is essentially the same as SPARQL.

Another current limitation of the SHACL specification is the lack of a unified and concise formal semantics for the so-called “core constraint components” of the language. Instead, the specification provides a combination of SPARQL queries and textual definitions to characterize these operators. This may be sufficient for reading or writing SHACL constraints, but a more abstract underlying formalization is still missing, in order for instance to devise efficient constraint validation algorithms, identify computational bottlenecks, or to compare SHACL’s expressivity with other languages.

Contributions. In this article, we propose a formal semantics for the core constraint components of SHACL, which is robust enough to handle arbitrary recursion, while being compliant with the current standard in the non-recursive case. It turns out that defining such a semantics is far from trivial, due essentially to the combination of three features of the language: recursion, arbitrary negation, and the target-based validation mechanism introduced above. One of the main difficulties is to define validation of shapes with so-called non-stratified negation, where negation is used arbitrarily in reference cycles, in a satisfactory way.

To do this, we base our semantic on the existence of a partial assignment of shapes to nodes that verifies both constraints and targets, i.e. intuitively a validation of nodes against shapes which may leave undetermined whether a given node verifies a shape or violates it. We show that this semantics has desirable formal properties, such as equivalence with classical validation in the presence of stratified constraints.

Recursion, however, comes at a cost, as we show that the problem of validating a graph is worst-case intractable in the size of the graph. Perhaps more surprisingly, we show that this property already holds for stratified constraints, and for a limited fragment of the language, without counting or path expressions. This observation leads us to propose a sound approximation, polynomial in the size of the graph, and whose worst-case execution time can be parameterized.

Organization. Section 2 discusses the problem of recursive SHACL constraints validation, with concrete examples. Then Section 3 defines a robust semantics for SHACL, together with a concise abstract syntax, and investigates its formal properties. Section 4 studies computational complexity of the graph validation problem under this semantics, and Section 5 proposes a sound approximation algorithm, in order to regain tractability (in the size of the graph under validation). Finally, Section 6 reviews alternative languages and formal semantics for graph constraints validation, with an emphasis on RDF.

An extended abstract of this paper has been accepted at the AMW workshop [8]. In addition, an appendix with detailed proofs and a translation from SHACL into our abstract syntax and conversely can be found at [7].

2 Validating SHACL Constraints with Targets

This section provides a brief overview of the constraint validation mechanism described in the SHACL specification, and discusses its extension to the case of recursive constraints. We focus here on the problem of deciding whether a graph is valid against a set of
Fig. 2. A SHACL shapes for semi-Polentone, and a graph $G$ to be validated against this shape, together with the shapes of Figure 1

shapes. Therefore we purposely ignore the notion of “validation report” defined in the specification, and encourage the interested reader to consult the specification directly.

Validating a graph $G$ against a set $S$ of shapes may be viewed as a two steps process. The first step consists in iterating over all shapes $s \in S$, and retrieve their respective targets in $G$. SHACL provides a dedicated language to describe the intended targets of a shape (e.g. the $\text{sh:targetClass}$ property in Figure 1), which is orthogonal to the language used to define constraints. Furthermore, this language has a very limited expressivity, allowing all targets of $s$ in $G$ to be retrieved in time linear in the size of $G$, before constraint validation.

The second step consists in iterating over each target node $v$ of each shape $s$, and check whether $v$ validates $s$. This check can be represented as a call to a recursive function $\text{validates}(s, G, v)$. Some of the constraints for $s$ may be validated by looking locally at the graph, i.e. at the IRI of $v$ and its outgoing paths. But $\text{validates}(s, G, v)$ may also trigger a recursive call $\text{validates}(s', G, v')$, where $s'$ is a shape referenced by $s$, and $v'$ is a successor of $v$ in $G$. It should be noted that $v'$ does not need to be a target of $s'$. In turn, $\text{validates}(s', G, v')$ may trigger another recursive call, etc.

Another important feature of SHACL is the possibility to declare negated constraints. For instance, shape $\text{SemiPolentoneShape}$ in Figure 2 uses $\text{sh:not}$ to describe someone who knows at least one person who is not a Polentone (but still lives in Northern Italy). In this case, $\text{validates}(\text{SemiPolentoneShape}, G, v)$ will succeed only if some successor of $v$ via property $\text{:knows}$ violates the constraints for $\text{PolentoneShape}$.

2.1 Recursive Constraints with Stratified Negation

Figures 1 and 2, considered together, illustrate a simple case of recursive constraint validation (i.e. constraints with reference cycles). The RDF triple $\text{:SemiPolentoneShape a sh:NodeShape ; sh:targetNode :Enrico ; sh:property [ sh:path :address ; sh:maxCount 1 ; sh:node :NIAddressShape ] ; sh:not [ sh:path :knows ; sh:node :PolentoneShape ] .}$

Figures 1 and 2, considered together, illustrate a simple case of recursive constraint validation (i.e. constraints with reference cycles). The RDF triple $\text{:SemiPolentoneShape a sh:NodeShape ; sh:targetNode :Enrico ; sh:property [ sh:path :address ; sh:maxCount 1 ; sh:node :NIAddressShape ] ; sh:not [ sh:path :knows ; sh:node :PolentoneShape ] .}$

To check if the graph $G$ of Figure 2 validates $\text{:SemiPolentoneShape}$, the validation process described in the specification would call $\text{validates}(\text{SemiPolentoneShape}, G, \text{:Enrico})$, triggering an infinite sequence of recursive calls to $\text{validates}(\text{PolentoneShape}, G, \text{:Davide})$. Intuitively, the problems is that $\text{validates}$ does not keep track of what has been validated or violated so far.
A classical solution to ground constraint evaluation in such cases is to define it w.r.t. an assignment of (positive and negated) shape labels to nodes. In this example, Enrico can be assigned :SemiPolentoneShape, and :Davide can be assigned the negation of :PolentoneShape. This assignment complies with constraints, and verifies the target, allowing us to validate the graph. Alternatively, it is possible to comply with all constraints by assigning :PolentoneShape to :Davide, and the negation of :SemiPolentoneShape to :Enrico. But this latter assignment would not allow us to validate the graph, because it does not verify the target.

A significant number of formal languages dealing with recursion (such as recursive Datalog[9], fixed-point logics, etc.) have semantics based on a similar intuition. Even within RDF, this notion of assignment is used in [6] to define recursive constraint validation for ShEx, a constraint language very similar to SHACL. However, the semantics proposed in [6] also has some limitations. As we explain in Section 5, the graph of Figure 2 would be rejected, taking only one possible (maximal) assignment into consideration, where :Davide is assigned :PolentoneShape, and therefore :Enrico cannot verify :SemiPolentoneShape. Another limitation of the semantics defined in [6] is that it only covers so-called stratified constraints, i.e. constraints such that reference cycles have no reference in the scope of a negation (see Definition 8).

2.2 Non-stratified Constraints

Extending assignment-based validation to the non-stratified case raises an interesting question, namely whether such an assignment should be total, i.e. assign each shape or its negation to each node of the graph. We illustrate this with validating the graph $G$ of Figure 2 against the constraints of Figure 3.

The node :Davide is the only target of :HappyPersonShape, which is valid if :Davide has an address, or knows a naive polentone. Because :Davide has an address, we could validate the graph with a simple call to `validates(HappyPersonShape, G, :Davide)`. But if we use total assignments, we need to decide whether :NaivePolentoneShape or its negation should be assigned to :Davide, and this cannot be done in a consistent manner: if :NaivePolentoneShape is assigned, then :Davide does not verify the corresponding constraint, but if the negation of :NaivePolentoneShape is assigned, then :Davide does not violate the constraint. Therefore a semantics based on total assignments would reject the graph.

It should be emphasized that this example is not a limit case: the same problem would appear for any (satisfiable) set of shapes containing a reference cycle (of any size),

```shacl
:HappyPersonShape a sh: NodeShape ;
  sh: targetNode : Davide ;
  sh: or ( 
    [ sh: path : address sh: minCount 1 ] 
  ) .

:NaivePolentoneShape a sh: NodeShape ;
```

**Fig. 3.** Two SHACL shapes which illustrates the need for partial assignments
and such that an odd number of references in this cycle are in the scope of a negation. Therefore, if one wants to define a robust semantics based on assignments for recursive SHACL, it should be based on partial assignments, leaving the possibility to assign neither a shape nor its negation to some nodes.

3 Semantics for SHACL

This section provides a semantics for recursive SHACL. Constraint validation is based on partial assignment, and (i) complies with the current semantics of SHACL for non-recursive constraints, (ii) supports arbitrary recursion and negation, and (iii) can handle simultaneous validation of multiple targets.

A target is validated iff there exists a valid assignment (called here faithful) verifying it. This is an essential difference from query answering, or cautious reasoning in Datalog, interested in certain answers, holding for all valid assignments. For instance, Figure 2: Davide may be assigned :PolentoneShape in some assignments verifying the constraints, but not in all of them.

Section 3.1 defines an abstract syntax and semantics for SHACL constraints, used in Section 3.2 to define graph validation. Section 3.3 investigates some properties of graph validation under this semantics, and Section 3.4 compares partial and total assignments for the stratified fragment.

Notation. As with SHACL, to define our logic, we borrow the notion of Property Paths from SPARQL, a feature that allows one to reason about pairs of nodes connected by a path satisfying certain regular constraints on the graph (for the syntax and semantics we defer to the SPARQL standard [13]). Following [15], we denote property paths as \( r, r_1, r_2, \ldots \). If \( r \) is a property path and \( G \) is a graph, then \( r(G) \) is the evaluation of \( r \), and consists of all pairs \((v, v')\) in \( G \) such that there is a path from \( v \) to \( v' \) satisfying \( r \).

We also use \( |X| \) to denote the size of structure \( X \).

3.1 Abstract Syntax and Semantics for SHACL constraints

As usual, we find it better to work with a logical abstraction of the concrete SHACL language. The basic idea of our abstraction is to use a fragment of first order logic to simulate node shapes, and then unravel property shapes as modal formulas over nodes.

Syntax. Constraints are defined by the following syntax:

\[
\phi ::= \top \mid s \mid I \mid \phi_1 \land \phi_2 \mid \neg \phi \mid \geq n.r\phi \mid r_1 = r_2 \tag{1}
\]

where \( s \) is a shape name, \( I \) is an IRI, \( r \) is a property path, and \( n \in \mathbb{N}^+ \). Using the above grammar, we can also express standard logic operators such as \( \lor \), and variations of modal operators such as \( \leq n.r\phi \) for \( \neg(\geq (n + 1).r\phi) \), \( = n.r\phi \) for \( \geq n.r\phi \land \leq n.r\phi \), and \( \bot \) for \( \neg \top \).

Let \( \mathcal{L} \) be the language defined by this grammar. A full operator-by-operator translation from SHACL core constraint components to \( \mathcal{L} \) and conversely is provided in the online appendix [7] of this article. For non-recursive shape constraints, this is a correct translation, in the sense that a set of constraints in one language and its translation in the other language validate exactly the same graphs, given the same targets. Unfortunately, in the absence of formal semantics for SHACL, this claim cannot be formally proven, but is
based on our understanding of the specification. We cannot claim that this also holds for
recursive shapes though, because SHACL validation in this case is not defined.

**Example 1.** We illustrate our language with the example from Figure 3. To express
SHACL cardinality constraints (e.g. `sh:maxCount`), we use $1.r\phi$, which says that a
node must have at most 1 $r$-successor satisfying $\phi$, or $1.r\phi$ for exactly one. Then :

$$s_{\text{niaddr}} \doteq (\leq 1.\text{:telephone }\top) \land (\leq 1.\text{:locatedIn :NorthernItaly})$$

where here $\top$ simply stands for a formula that is true in every node. In the same way, we
can define the formula $s_{\text{pol}}$ that expresses :PolentoneShape. Both $s_{\text{niaddr}}$ and $s_{\text{pol}}$
appear in the definition of $s_{\text{pol}}$. This mimics the SHACL syntax, where both shapes
were mentioned:

$$s_{\text{pol}} \doteq (\leq 0.\text{:knows }\neg s_{\text{pol}}) \land (\leq 1.\text{:address }s_{\text{niaddr}})$$

**Semantics.** Because shape names may appear in constraint formulas, we define the
inductive evaluation of a formula in terms of a node, a graph, and an assignment that
mandates which shapes are true or false in each node.

**Definition 1 ((Partial) assignment).** Let $S$ be a set of shape names, and $G$ a graph.
An assignment $\sigma$ for $G$ and $S$ is a mapping assigning a subset of $S \cup \{-s \mid s \in S\}$ to
each node in $G$, such that $s$ and $-s$ cannot be both in $\sigma(v)$

An assignment may be partial, meaning that for some node $v$ and $s \in S$, $\sigma(v)$ may
contain neither $s$ nor $-s$. The evaluation $[\phi]^{v,G,\sigma}$ of formula $\phi$ at node $v$ in graph $G$
given $\sigma$ is defined inductively in Table 1. In order to model partial assignments, we use a
three-valued logic, which, in addition to the usual 1 and 0 for true and false, uses 0.5 to
represent an unknown truth value. But if assignments are required to completely specify
the nodes, then this third value is not needed:

**Definition 2 (Total assignment).** A total assignment $\sigma$ for $G$ and $S$ is an assignment
such that for each $s \in S$, either $s \in \sigma(v)$ or $-s \in \sigma(v)$

**Observation 1** Let $\sigma$ be a total assignment for $G$ and $S$, and $\phi$ a formula using shape
names in $S$. Then for each node $v$ of $G$, either $[\phi]^{v,G,\sigma} = 0$ or $[\phi]^{v,G,\sigma} = 1$

The inductive definition of $[\phi]^{v,G,\sigma}$ is standard, aside maybe for the operator $\geq n.r$
Intuitively, $\geq n.r \phi$ evaluates to true iff at least $n$ $r$-successors of $v$ validate $\phi$, whereas $\geq n.r \phi$ evaluates to false iff the number of $r$-successors of $v$ which do or could validate $\phi$ is strictly inferior to $n$. This allows the semantics to comply with SHACL cardinality
constraints in the non-recursive case.

**From SHACL to $\mathcal{L}$ constraints.** We model a SHACL shape as a triple $(s, \phi_s, \text{target}_s)$,
where $s$ is a shape name, $\phi_s$ is a constraint in $\mathcal{L}$, and $\text{target}_s$ is a (possibly empty) monadic
query to retrieve the target nodes for $s$. A valid set $S$ of shapes is such that for each $(s, \phi_s, \text{target}_s) \in S$, if $\phi_s$ mentions shape name $s'$, then the shape $(s', \phi'_s, \text{target}'_s) \in S$.
In what follows, we assume that a set of shapes is always valid. Abusing notation, we
will also constantly refer to shapes by their name only.
3.2 Validation

We finally have all components in place to define graph validation. Intuitively, a graph $G$ is valid against a set $S$ of shapes if one can find an assignment $\sigma$ for $G$ and $S$, such that $\sigma$ complies with the shape targets, while verifying all constraints in $S$. We call such an assignment faithful. A natural question is whether all targets should be required to be verified simultaneously, i.e. by the same assignment, or one after the other, by different assignments, which may be mutually incompatible. We choose to focus on the former, seemingly harder problem. As shown in Proposition 3, this distinction is also of little interest from a computational perspective. A faithful assignment is defined as follows:

**Definition 3 (Faithful Assignment).** A assignment $\sigma$ for $G$ and $S$ is faithful iff target$_s(G) \subseteq \sigma(v)$ for each $(s, \phi_s, \text{target}_s) \in S$, and, for each node $v$ in $G$:
- if $s \in \sigma(v)$, then $[\phi_s]^{v,G,\sigma} = 1$
- if $s \not\in \sigma(v)$, then $[\phi_s]^{v,G,\sigma} = 0$

**Definition 4 (Validation).** A graph $G$ validates against a set of shapes $S$ iff there is a faithful assignment $\sigma$ for $G$ and $S$

As already mentioned, our (online) appendix provides a full translation from SHACL to sets of shapes which preserves validation, as long as the SHACL shapes are non-recursive (i.e. contain no reference cycle). But our notion of validation is more robust, in that it is also well-defined for recursive shapes. In the following section, we study the complexity of the validation problem. But for now, we provide some insight on the properties of this semantics.

3.3 Properties of Validation

Before studying validation properties, we need to introduce additional technical notation.

First, we use $\Sigma^{G,S}$ to designate the set of all assignments for $G$ and $S$. Then we define the "immediate evaluation" operator $T^{G,S}$ for $G$ and $S$, or simply $T$ when obvious.
from the context. It takes an assignment $\sigma$, and returns the assignment $T(\sigma)$ obtained by evaluating each $\phi_s$ at each node of $G$.

**Definition 5 (Immediate evaluation operator $T$).**

$T : \Sigma^{G,S} \rightarrow \Sigma^{G,S}$ is the function defined by

$s \in (T(\sigma))(v) \iff [\phi_s]^v_{G,\sigma} = 1$, and $\neg s \in (T(\sigma))(v) \iff s \in [\phi_s]^v_{G,\sigma} = 0$

Finally, we define the preorder $\preceq$ over $\Sigma^{G,S}$ by:

**Definition 6 (Preorder $\preceq$).**

$\sigma_1 \preceq \sigma_2$ iff $\sigma_1(v) \subseteq \sigma_2(v)$ for each node $v$ in $G$

Equivalently, $\preceq$ can be viewed as a set inclusion between assignments viewed as sets of positive and negative atoms.

**Validation without target always succeeds.** A particularity of the SHACL specification is that a graph $G$ must validate against a set $S$ of shapes if no shape in $s$ has target in $G$. This also holds in the recursive case for our semantics.

**Proposition 1.** If $S$ has no target in $G$, then $T^{G,S}$ admits a unique minimal fixed-point $\sigma$ w.r.t $\preceq$, and $\sigma$ is a faithful assignment for $G$ and $S$.

Interestingly, this property may not hold for total assignments. For instance, there is no total faithful assignment for the graph of Figure 2 and the two shapes of Figure 3 without target query.

**A stricter notion of faithfulness.** From Definition 3, a faithful assignment is only required to assign $s$ to a node $v$ if $\phi_s$ is verified by $v$, and $\neg s$ to $v$ if $\phi_s$ is violated by $v$. But it is also free to assign none of these two, regardless of whether $v$ validates or violates $\phi_s$. This may seem counterintuitive, which leads to a stricter notion of faithfulness:

**Definition 7 (Strictly-faithful assignment).** An assignment $\sigma$ for $S$ under $G$ is strictly-faithful if it verifies all targets for $G$ and $S$, and in addition:

- if $s \in \sigma(v)$, then $[\phi_s]^v_{G,\sigma} = 1$
- if $\neg s \in \sigma(v)$, then $[\phi_s]^v_{G,\sigma} = 0$
- otherwise, $[\phi_s]^v_{G,\sigma} = 0.5$

We also say that a graph $G$ strictly validates against a set of shapes $S$ if there is a strictly faithful assignment for $G$ and $S$.

For instance, in Figure 2, a strictly faithful assignment would have to assign $\neg :$SemiPolentoneShape to :addr1, because :addr1 violates the constraint for :SemiPolentoneShape. But a faithful assignment could assign neither :SemiPolentoneShape nor its negation to :addr1.

The operator $T$ provides a more concise way to define these two notions of faithfulness. Both faithful and strictly faithful assignments must validate the targets for $G$ and $S$. But in addition, a faithful assignment $\sigma$ must be such that $\sigma \preceq T(\sigma)$, whereas a strictly faithful assignment $\sigma'$ must be such that $\sigma' = T(\sigma')$.

Somehow surprisingly, these two notions of validation coincide. To prove this, we first need a useful property, the monotonicity of $T$ w.r.t $\preceq$:

**Lemma 1 (monotonicity of $T$).** For any $G$, $S$ and $\sigma_1, \sigma_2 \in \Sigma^{G,S}$:

if $\sigma_1 \preceq \sigma_2$, then $T(\sigma_1) \preceq T(\sigma_2)$
We can now state the equivalence:

**Proposition 2.** For any $G$ and $S$, $G$ validates $S$ iff $G$ strictly validates $S$.

*Proof (Sketch).* The right direction is trivial, because a strictly faithful assignment is a faithful assignment. In the other direction, let $\sigma_0$ be a faithful assignment for $G$ and $S$. Define $\Sigma' \subseteq \Sigma^{G,S}$ as all extensions of $\sigma_0$. Then for any $\sigma' \in \Sigma'$, $\sigma_0 \preceq \sigma'$. So from Lemma 1, $T(\sigma_0) \preceq T(\sigma')$. And because $\sigma_0$ is faithful, $\sigma_0 \preceq T(\sigma)$. Therefore $\sigma_0 \preceq T(\sigma')$, i.e. $T(\sigma') \in \Sigma'$.

Now consider the (meet) semi-lattice $(\Sigma', \preceq)$ rooted in $\sigma$. We just showed that for each $\sigma' \in \Sigma'$, $T(\sigma') \in \Sigma'$. In addition, from Lemma 1, $T$ is monotone over $(\Sigma', \preceq)$. So from a (weaker version of) the Knaster-Tarski Theorem, $T$ admit a fixed-point $\sigma_2$ over $\Sigma'$. And because $\sigma_0 \preceq \sigma_2$, $\sigma_2$ complies with all targets for $G$ and $S$. Therefore $\sigma_2$ is a strictly faithful assignment. □

**All we need is one target.** As explained at the beginning of this section, we chose to focus on validating all targets simultaneously, because it is a seemingly harder problem. But the following observation shows that from a computational perspective, validating multiple targets instead of one has no impact in the worst-case:

**Proposition 3.** For every graph $G$ and every set $S$ of shapes, one can construct in linear time (in both $|G|$ and $|S|$) a graph $G'$ and set $S'$ of shapes, such that $G$ validates against $S$ iff $G'$ validates against $S'$, and $S'$ has a single target in $G'$.

*Proof (Sketch).* Let $s_1, ..., s_n$ be the shapes in $S$, with respective targets $v_1^1, ..., v_1^m$, ..., $v_n^1, ..., v_n^m$. Extend $G$ with a fresh node $v_0$, and an edge $(v_0, e_i, v_i^j)$ for each $e_i^j$, where $e_i^j$ is a fresh edge label. Then delete all target expressions in $S$, and extend $S$ with a fresh shape $s_0$, with unique target $v_0$, and constraint $\phi_{s_0} \equiv \geq 1. e_1^m \land ... \land \geq 1. e_n^m$. □

### 3.4 Validation and Stratified Negation

So far we have promoted the idea that the need for partial assignments was due to constraints involving circular references and negation. We now make this intuition more precise, showing that we can indeed focus solely on total assignments, as long as the input shapes do not mix circular references and negation.

To formalize this idea, we borrow from Datalog[9] the notion of *stratified* negation (assuming w.l.o.g that constraints do not contain two consecutive negation symbols).

**Definition 8 (stratification).** A set $S$ of shape definitions is stratified if there is function $\text{str} : S \rightarrow \mathbb{N}$ such that:

- If $s_1$ appears in $\phi_{s_2}$, then $\text{str}(s_1) \leq \text{str}(s_2)$
- If $s_1$ appears in $\phi_{s_2}$ in the scope of a negation then $\text{str}(s_1) < \text{str}(s_2)$.

It must be emphasized that the language $\mathcal{L}$ does not include $\forall$ or $\leq$.r. If these operators were included, then one would need to redefine the second condition accordingly, considering for example that $\leq n.r$ is a form of negation.

The following result confirms that a semantics based on total assignment is sufficient for sets of shapes with stratified negation.
Proposition 4. Let $S$ be a set of shapes with stratified negation and $G$ a graph. Then there exists a faithful assignment for $G$ and $S$ iff there exists a total faithful assignment for $G$ and $S$.

Proof (Sketch). For the right direction, the proof is trivial. For the left direction, to simplify notation, we consider assignments as sets of positive and negative atoms. Let $\sigma$ be a faithful assignment for $G$ and $S$, and let $S_1, \ldots, S_n$ be the strata of $S$, from lowest to highest. The proof constructs an extension $\sigma'$ of $\sigma$, stratum by stratum, initialized with the empty set. For each stratum $S_i$ (starting from $S_0$), $\sigma'$ is extended in three steps. First, $\sigma'$ is extended with $\sigma$ reduced to atoms with shape names in $S_i$. Then $T$ is applied to $\sigma'$ recursively, until a fixed-point is reached. Finally, $\sigma'$ is extended with each $s(v)$ such that $v$ is a node in $G$, $s \in S_i$ and $\neg s(v) \notin \sigma'$. It can be shown by induction on $i$ that this extension of $\sigma'$ always exists, and complies with all constraints for shapes in $S_0, \ldots, S_i$. So when $i$ reaches $n$, the last extension of $\sigma'$ is a total faithful assignment for $G$ and $S$.

This result is important for computational reasons. It implies that validation based on partial assignments is not easier (in the stratified case) than validation based on total assignments, which is somehow surprising.

4 Complexity

We now study the computational complexity of the validation problem.

<table>
<thead>
<tr>
<th>VALIDATION:</th>
<th>Input: Set $S$ of shapes, graph $G$</th>
<th>Decide: $G$ validates against $S$</th>
</tr>
</thead>
</table>

From Proposition 3, this problem is equivalent to deciding validation with a single target node. Table 2 summarizes our results. As is customary, since the size of $G$ is likely to be orders of magnitude larger than the size of $S$, we also study the problems $\text{VALIDATION}(S)$ and $\text{VALIDATION}(G)$, for a fixed set $S$ of shapes or fixed graph $G$, which correspond to the data complexity and constraint complexity of validation.

<table>
<thead>
<tr>
<th>Fragment</th>
<th>Data</th>
<th>Constraint</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>(recursive) $\mathcal{L}^+$</td>
<td>PTIME-c</td>
<td>PTIME-c</td>
<td>PTIME-c</td>
</tr>
<tr>
<td>stratified $\mathcal{L}_{0, \neg, \wedge}$</td>
<td>NP-c</td>
<td>NP-c</td>
<td>NP-c</td>
</tr>
<tr>
<td>$\mathcal{L}$ (= SHACL)</td>
<td>NP-c</td>
<td>NP-c</td>
<td>NP-c</td>
</tr>
</tbody>
</table>

Table 2. Computational complexity of $\text{VALIDATION}$. -c stands for complete.

We start by showing a general NP upper bound for the complete validation problem, based on guessing a witnessing faithful assignment. Then we show that this NP upper bound is tight even for a fixed set of shapes using stratified negation and basic operators. We also show that this bound is tight for a fixed graph. Lastly, we show that disallowing recursive negation immediately guarantees the validation problem to be in polynomial time.

Let us start with the upper bound. As we mentioned, to prove membership in NP we describe an algorithm that first guesses a faithful assignment $\sigma$ respecting the targets,
and then checks that this assignment is indeed faithful (that the targets are respected is immediate to check). To check that an assignment is faithful, we need to compute the value of $[\phi]^{v,G,\sigma}_s$ for each positive or negative shape that is assigned to a node in $G$. In our case we can show that checking the truth value of a given formula is in polynomial time, with an algorithm similar in spirit to algorithms for other modal logics with counting (see e.g. [14]). In particular, property paths are not a problem, because we can pre-compute the answers to all of them, as in e.g. [15]. Summing up, we have:

**Proposition 5 (Combined – Upper Bound).**\textsc{Validation} is in $NP$.

As for the lower bound, let $L_{\bot,-,\land}$ be the fragment without property path, counting or path equality, i.e. defined with the grammar: $\phi ::= \top | I | s | \phi_1 \land \phi_2 | \neg \phi \geq 1.p \phi$, where $p$ is an IRI. Then validation is already intractable in data complexity for stratified $L_{\bot,-,\land}$. This may come as a surprise, considering that data complexity for reasoning in stratified Datalog is in PTIME [9]. We show hardness by a reduction from the satisfiability problem of a propositional circuit; there is a fixed set $S$ of shapes such that every propositional circuit can be transformed into a graph, and this graph validates against $S$ if and only if the circuit is satisfiable.

**Proposition 6 (Data – Lower Bound).** There is a stratified fixed set $S$ of shapes in $L_{\bot,-,\land}$ such that \textsc{Validation}(S) is NP-hard.

Naturally, this result implies that the complexity of \textsc{Validation} (i.e. combined complexity) is NP-hard. We can also show that the problem is NP-hard in constraint complexity for the same fragment:

**Proposition 7 (Constraint – Lower Bound).** There is a fixed graph $G$ such that \textsc{Validation}(G) is NP-hard, even if $S$ is restricted to stratified sets of shapes in $L_{\bot,-,\land}$.

As a positive result, validation is in PTIME as long as negation (in propositional form or as a modality such as $\leq n.r$) is disallowed on recursive calls. Formally, let $L^+$ be the fragment of $L$ in which every mention of a shape $s$ is under the scope of an even number of negations. Note that this extends the cases when the current SHACL specification is defined, because $L^+$ allows arbitrary formulas without recursion. Hardness for PTIME can be shown by a log-space reduction from the problem of evaluating a monotone boolean circuit.

**Proposition 8 (Combined – Matching).** \textsc{Validation} is PTIME-c for (recursive) $L^+$.

5 Approximation

The above intractability result for data complexity (Proposition 7), and even for a stratified set of shapes, is an important limitation. In order to alleviate this problem, we present in this section an approximation algorithm to decide whether a graph $G$ validates a set $S$ of shapes, with an integer parameter $\rho$. If $\rho$ is set, then the algorithm is sound, and it runs in time polynomial in $|G|$. If $\rho$ is unbound, then the algorithm is sound and complete, but may run in time exponential in $|G|$. The approximation is sound in that the algorithm returns \textsf{Valid} (resp. \textsf{Invalid}) only if $G$ validates (resp. does not validate) against $S$.

For readability, from Proposition 3, we focus on validation with a single target node $v_0$ for shape $s_0$. Algorithm 1 describes the procedure, composed of two steps. The first
Step 1: minimal fixed-point. As a reminder from Section 3.2, we use \( \Sigma^{G,S} \) to denote the set of all (possibly partial) assignments for \( G \) and \( S \). The first step of the algorithm consists in computing the minimal fixed-point \( \sigma_{\text{minFix}} \) of the operator \( T \) (see Definition 3.3) over \( \langle \Sigma^{G,S}, \preceq \rangle \). Because \( \langle \Sigma^{G,S}, \preceq \rangle \) is a semi-lattice and \( T \) is monotone over \( \langle \Sigma^{G,S}, \preceq \rangle \) (Lemma 1), this minimal fixed-point must exist and be unique. Furthermore, it can be computed in time polynomial in \( |G| \), initializing \( \sigma_{\text{minFix}} \) with the empty set, and then applying \( T \) to \( \sigma_{\text{minFix}} \) recursively, until a fixed-point is reached. This is performed by procedure \textsc{ComputeMinFix}. If \( s_0 \in \sigma_{\text{minFix}}(v_0) \), then the graph is valid. Line 2. Furthermore, any strictly faithful assignment of for \( G \) and \( S \) must be a fixed-point of \( T \) (see Section 3.2), and therefore it must extend \( \sigma_{\text{minFix}} \). So from Proposition 2, If \( \lnot s_0 \in \sigma_{\text{minFix}}(v_0) \), then the graph is invalid, Line 3.

Step 2: breadth-first search. The next step consists in searching for a faithful assignment, in a breadth-first fashion, starting from the target node \( v_0 \). We may abuse notation use set notation \((\in, \cup, \text{ etc.})\) to describe the stack. Similarly, for readability, we represent assignments interchangeably as functions or as sets of positive and negative atoms.

Each element of the stack (i.e. each “branch” of this exploration) is a tuple \( \langle \sigma, \sigma^P, A, n \rangle \), where:

- \( \sigma \) is the current assignment being constructed, initialized with \( \sigma_{\text{minFix}} \cup \{s_0(v_0)\} \)
- \( \sigma^P \preceq \sigma \) keeps track of shapes freshly assigned to a node during the previous expansion of \( \sigma \). For any element of the stack, if \( \sigma^P \) is empty, then no constraint needs to be propagated in this branch, i.e. \( \sigma \) is a faithful assignment, and so the graph is validated, line 7.
- \( A \) is a set of atoms of the form \( s(v) \), such that \( s(v) \notin \sigma \) and \( \lnot s(v) \notin \sigma \), and this condition is required to hold.
- \( n \) is the current depth of the exploration, incremented each time \( \sigma \) is extended. When \( n \) reaches \( \rho \), the size of the stack cannot be extended anymore, which triggers a call to \textsc{Reduce}, line 11, to merge some of the current branches.

Line 8, function \textsc{Extend} computes each minimal extensions \( \sigma' \) of \( \sigma \) such that:

- If \( \sigma^P(v) \) contains \( s \) then \( \lnot [\sigma^P](v)_{\sigma'} = 1 \).
- If \( \sigma^P(v) \) contains \( \lnot s \) then \( [\sigma^P](v)_{\sigma'} = 0 \), and
- if \( A \) contains \( s(v) \) then \( \sigma(v) \) does not contain neither \( s \) nor \( \lnot s \)

Because of the complexity of our logic, we can show that each call to \textsc{Extend} can be executed in time \( O(|G||S|) \).

Finally, if the depth \( n \) of the exploration reaches \( \rho \), line 11, then procedure \textsc{Reduce} prevents the number of elements in the stack to increase. Line 18, function \textsc{GetClosestPair} retrieves the two closest assignments \( \sigma_1 \) and \( \sigma_2 \) (in terms of edit distance) in the Stack. Then function \textsc{GetConflicts} line 20 retrieves the (possibly empty) set \( A \) of atoms which \( \sigma_1(v) \) and \( \sigma_2(v) \) disagree on, i.e. \( s(v) \in A \) if both \( s \) and \( \lnot s \) are in \( \sigma_1(v) \cup \sigma_2(v) \), and the procedure \textsc{Replace} sets each \( \sigma_i \) to \( \sigma_i \setminus \{s(v), \lnot s(v)\} \). After this step, either \( \sigma_1 \preceq \sigma_2 \) or \( \sigma_2 \preceq \sigma_1 \) must hold, and Line 23 only the largest one is retained and pushed in the stack.
Algorithm 1 APPROXIMATION

Require: \( G', S, s_0, v_0, \sigma_{\text{minFix}}, \rho \)
1: \( \sigma_{\text{minFix}} \leftarrow \text{COMPUTE}_{\text{MINFix}}(G', S) \)
2: if \( s_0 \in \sigma_{\text{minFix}}(v_0) \) then return Valid
3: if \( \neg s_0 \in \sigma_{\text{minFix}}(v_0) \) then return Invalid
4: Stack \( \leftarrow \{ \sigma_{\text{minFix}} \cup \{ s_0(v_0) \}, \{ s_0(v_0) \}, \{ \text{atoms}(G', S) \}, 0 \} \)
5: while \( \text{NONEMPTY}(\text{Stack}) \) do
6: \( \langle \sigma, \sigma^P, A, n \rangle \leftarrow \text{POP}(\text{Stack}) \)
7: if \( \sigma^P = \emptyset \) then return Valid
8: for all \( \sigma' \in \text{EXTEND}(\sigma, \sigma^P, A) \) do
9: \( \text{PUSH}(T, \langle \sigma', \sigma' \setminus \sigma, A, n + 1 \rangle) \)
10: end for
11: if \( n \geq \rho \) then \( \text{Stack} \leftarrow \text{REDUCE}(\text{Stack}, |T|) \)
12: end while
13: return Unknown
14: procedure \( \text{REDUCE}(\text{Stack}, m) \)
15: \( i = 0 \)
16: while \( i \leq m \) do
17: \( (\langle \sigma_1, \sigma_1^P, A_1, n_1 \rangle, \langle \sigma_2, \sigma_2^P, A_2, n_2 \rangle) \leftarrow \text{GETCLOSESTPAIR}(\text{Stack}) \)
18: \( \text{Stack} \leftarrow \text{Stack} \setminus \{ \langle \sigma_1, \sigma_1^P, A_1, n_1 \rangle, \langle \sigma_2, \sigma_2^P, A_2, n_2 \rangle \} \)
19: \( A \leftarrow \text{GETCONFLICTS}(\sigma_1, \sigma_2) \)
20: \( \sigma_1 \leftarrow \text{REPLACE}(\sigma_1, A) \)
21: \( \sigma_2 \leftarrow \text{REPLACE}(\sigma_2, A) \)
22: \( \sigma = \max\{\sigma_1, \sigma_2\} \)
23: \( \text{PUSH}(\text{Stack}, \langle \sigma, \sigma_1^P \cup \sigma_2^P, A \cup A_1 \cup A_2, \max\{n_1, n_2\} \rangle) \)
24: \( i \leftarrow i + 1 \)
25: end while
26: end procedure

The number of possible assignments is of order \( 2^{|G|} \), since every shape can be assigned an arbitrary subset of the nodes of \( G \). However, if the parameter \( \rho \) is fixed, then the reduced stack makes sure that the execution time is bounded by \( O((|G|^{|S|})^\rho)) \).

6 Related Work

Several schema languages have been proposed or implemented for RDF before SHACL, and some of them are closely associated to the genesis of SHACL. But first, it should be mentioned that RDF Schema (RDFS), contrary to what its name may suggest, is not a schema language in the classical sense, in that it is primarily used to infer implicit facts.

SPIN\(^8\) allows the user to express constraints as SPARQL queries (natively, or using templates) and to declare targets for these constraints, similar to SHACL targets. SPIN became a W3C member submission in 2011, before being explicitly superseded by SHACL in 2017. Being based on SPARQL, it supports negation, but not full recursion.

ShEx has been actively developed since 2012 [5], as a dedicated constraint language for RDF, strongly inspired by XML schema languages. The first version of ShEx did

\(^8\)http://spinrdf.org/
support recursion, but no negation. A formal semantics was provided in [19], based on regular bag expressions. Recently, ShEx 2.0\(^9\) incorporated negation, and a formal semantics was provided in [6], together with a abstract language called Shape Schemas. As highlighted in [4], ShEx and SHACL have lot in common, and the semantics provided in [6] can be directly adapted to SHACL. This proposal is also similar to the one made in this article, in that validation is based on a typing or shape assignment verifying target and constraints. Unfortunately, the semantics proposed in [6] only handles stratified negation. Moreover, the (unique) typing used to define validation favors the validation of shapes in the lowest stratum, thus rejecting graphs which may intuitively appear as valid, as illustrated by Figure 2.

Another line of work is inspired by the Web Ontology Language (OWL), which is based on Description Logics (DLs) [2]. Like RDFS, OWL was not designed as a schema language, but adopts the open-world assumption, not well-suited to express constraints. Still, proposals have been made to reason with DLs understood as constraints: by introducing auto-epistemic operators [10], partitioning DL formulas into regular and constraint axioms [16, 20], or reasoning with closed predicates [18]. This last approach was actually proposed as a semantic grounding for SHACL [17], reducing constraint validation to classical satisfiability with closed binary predicates. But as illustrated with Example 3, this semantics does not behave well in the presence of targets and non-stratified constraints.

Recursion over negation has been traditionally studied in logical programming [9], where stable model semantics (SMS) is one of the most prominent paradigms [12]. Ground fact entailment under SMS is NP-hard in data complexity, but tractable in the stratified case, in contrast to our semantics. SMS being based on minimal models, it cannot be directly applied in our case. A possible way to relate the two semantics, at least for the stratified case, is to reason on shape “complements” under SMS. Still, our preliminary investigations show that such task is not trivial.

7 Conclusion

The article proposes an abstract syntax and formal semantics for SHACL core constraint components. This semantics is robust enough to handle constraints with arbitrary recursion, which can be expressed in SHACL, but whose validation is left explicitly open in the specification. One of our contributions is to highlight semantic issues related to non-stratified SHACL targets. To address such cases, we adopt a notion of partial assignment of (positive and negated) shapes to nodes, and define a semantics with desirable properties, such as monotonicity of forward-chaining, or equivalence with total assignments in the stratified case. We then show that the validation problem is NP-complete for any fragment with at least conjunction, negation and existential quantification, in the size of either graph or constraints, regardless of stratification or of the encoding of cardinality constraints. Therefore we propose a sound approximation algorithm, parameterized by an integer \(\rho\), which guarantees termination in time polynomial in the size of the graph.

As a continuation, we plan to investigate further problems, such as (finite) satisfiability of a set of shapes, or SPARQL query containment in the presence of SHACL constraints.

\(^9\)http://shex.io/shex-semantics/
We also expect this formalization to be abstract enough to be extended to other constraint languages for graphs, such as ShEx, in order to handle arbitrary recursion.

Bibliography

Semantics and Validation of Recursive SHACL: Appendix

Julien Corman\textsuperscript{1}, Juan L. Reutter\textsuperscript{2}, and Ognjen Savkovi\v{c}\textsuperscript{1}

\textsuperscript{1} Free University of Bozen-Bolzano, Bolzano, Italy
\textsuperscript{2} PUC Chile and Center for Semantic Web Research, Santiago, Chile

Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Translation from SHACL core constraint components to $L$ and back</td>
<td>2</td>
</tr>
<tr>
<td>1.1</td>
<td>Scope of the SHACL-to-$L$ translation</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Translation from SHACL to $L$</td>
<td>2</td>
</tr>
<tr>
<td>1.3</td>
<td>Translation from $L$ to SHACL</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Proofs</td>
<td>6</td>
</tr>
<tr>
<td>2.1</td>
<td>Preliminaries</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Semantic Properties: proofs</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>Complexity Proofs</td>
<td>14</td>
</tr>
</tbody>
</table>
The above article was submitted to the ISWC 2018 conference (more exactly, a slightly different version of it). This appendix extends the article with full proofs, as well as a translation from SHACL to our abstract syntax (and conversely). In order to provide a higher-level introduction, the notation adopted in the article differs slightly from the one used in the proofs. The numbering of theorems also differ. Section 2.1 explains the additional notation, and provides a mapping between numbering schemes.

1 Translation from SHACL core constraint components to $\mathcal{L}$ and back

1.1 Scope of the SHACL-to-$\mathcal{L}$ translation

Let $D$ be a set of SHACL shape definitions, and let $d_s \in D$ be the definition of shape $s$. W.l.o.g., we assume that $D$ does not contain anonymous shapes, i.e. each (node or property) shape definition $d_s \in D$ is a set of RDF triples with subject $s$, where $s$ is an IRI.

The SHACL specification allows some RDF triples in $d_s$ to be omitted (e.g. triples with $\text{sh:property}$ in certain circumstances), as syntactic sugar. We assume that this is not the case in $d_s$.

Within $d_s$, we focus on triples expressing constraints, ignoring for instance target declaration, shape name declaration, severity, validation report, etc. Within these, we focus on triples expressed with SHACL core constraint components, Section 4 of the SHACL specification.

In addition, as already explained, the language $\mathcal{L}$ abstracts away from constraints on IRIs (datatype checks, regular expressions, value comparison, etc.), which can be verified in linear time (this is the reason why $\mathcal{L}$ contains only a single terminal symbol $I$ (for IRI) other than $\top$). Therefore we assume that a shape definition $d_s$ does not contain triples with property: $\text{sh:datatype}$, $\text{sh:nodeKind}$, $\text{sh:minExclusive}$, $\text{sh:minInclusive}$, $\text{sh:maxExclusive}$, $\text{sh:maxInclusive}$, $\text{sh:minLength}$, $\text{sh:maxLength}$, $\text{sh:pattern}$, $\text{sh:languageIn}$, $\text{sh:uniqueLanguage}$ or $\text{sh:in}$.

Similarly, because we assume (with a slight abuse only) that ordering two values is not harder than checking whether they differ, we abstract away from property pair value comparisons, ignoring $\text{sh:lessThan}$ and $\text{sh:lessThanOrEquals}$, and choose to translate only $\text{sh:equals}$ and $\text{sh:disjoint}$.

1.2 Translation from SHACL to $\mathcal{L}$

Given a shape definition $d_s$, we denote with $\text{cons}(d_s)$ the set of all remaining triples, after discarding the ones just mentioned. We divide our translation $t$ from SHACL to $\mathcal{L}$ into:

- IRI constraints
- boolean combinations of shape definitions
- node shape definitions
- property shape definitions

Abusing notation, if $(s, p, o)$ is an RDF triple, we use $t(s, p, o)$ instead of $t((s, p, o))$. 


1.2.1 IRI constraints
The only remaining constraints on IRI are triples of the form \((s, \text{sh} : \text{hasValue}, I)\). We define:
\[
t(s, \text{sh} : \text{hasValue}, I) \doteq I
\]

1.2.2 Boolean combinations of shape definitions
\(t\) is defined as expected for boolean operators (with \(s, s', s_i\) shapes and \(l\) a SHACL list of shapes).
- \(t(s, \text{sh} : \text{not}, s') = \neg s'\)
- \(t(s, \text{sh} : \text{and}, l) = \bigwedge_{s' \in l} s'\)
- \(t(s, \text{sh} : \text{or}, l) = \bigvee_{s' \in l} s'\)
- \(t(s, \text{sh} : \text{not}, s', \text{sh} : \text{or}, l, \text{sh} : \text{and}, m) = \bigvee_{1 \leq i \leq n} (s_i \land \bigwedge_{1 \leq j \leq m, i \neq j} \neg s_j)\)

1.2.3 Node shape definitions
Let \(d_s\) be a node shape definition. Then \(\text{cons}(d_s)\) can be partitioned into 3 subsets \(\text{pp}(d_s)\), \(\text{closed}(d_s)\) and \(\text{loc}(d_s)\).

1.2.3.1 \(\text{pp}(d_s)\).
\(\text{pp}(d_s)\) is composed of all triple with property \(\text{sh} : \text{property}\).

We translate it as:
\[
t(\text{pp}(d_s)) \doteq \bigwedge_{(s, \text{sh} : \text{property}, s') \in \text{pp}(d_s)} t(d_s)
\]

1.2.3.2 \(\text{closed}(d_s)\).
\(\text{closed}(d_s)\) contains all triple with property \(\text{sh} : \text{closed}\) or \(\text{sh} : \text{ignoredProperties}\)
- If \(\text{closed}(d_s)\) contains \((s, \text{sh} : \text{closed}, \text{false})\), then \(t(\text{closed}(d_s)) = \top\).
- If \(\text{closed}(d_s)\) contains \((s, \text{sh} : \text{closed}, \text{true})\), then we define \(\text{ignored}(d_s)\) as follows:
  - If \(\text{closed}(d_s)\) contains \((s, \text{sh} : \text{ignoreProperties}, l)\), then \(\text{ignored}(d_s) = \{p \mid p \in l\}\).
  - Otherwise, \(\text{ignored}(d_s) = \emptyset\).
Then if \(r\) is a property path, let \(\text{signat}(r)\) be all properties appearing in \(r\).
And let \(R = \{r \mid (s, \text{sh} : \text{property}, s') \in \text{pp}(d_s)\text{ and } (s', \text{sh} : \text{path}, r) \in d_s\}\).
We define the regular path expression \(e(s)\), which is the complement (\(\neg\)) of the disjunction (\(\lor\)) of all properties appearing in \(R\) or in \(\text{closed}(d_s)\).
More formally, let \(E_s = (\bigcup_{r \in R} \text{signat}(r)) \cup \text{closed}(d_s)\).
Then \(e_s = \text{false}(e_1, .., e_n)\), where \(E_s = \{e_1, .., e_n\}\).
We can now define \(t(\text{closed}(d_s)) \doteq 0 . e_s \top\)

1.2.3.3 \(\text{loc}(d_s)\).
\(\text{loc}(d_s)\) contains all triples with property \(\text{sh} : \text{hasValue}, \text{sh} : \text{not}, \text{sh} : \text{or}, \text{sh} : \text{and}\) or \(\text{sh} : \text{xone}\).
If \(\text{loc}(d_s) = \emptyset\), then \(t(\text{loc}(d_s)) \doteq \top\)
Otherwise, \(t(\text{loc}(d_s)) \doteq \bigwedge_{m \in \text{loc}(d_s)} t(m)\)

1.2.3.4 Translation of \(d_s\).
We are now ready to define the translation \(t(d_s)\) of \(\text{cons}(d_s)\) into \(\mathcal{L}\) as:
\[
t(d_s) \doteq t(\text{pp}(d_s)) \land t(\text{closed}(d_s)) \land t(\text{loc}(d_s))
\]
1.2.4 Property shape definition

Let $d_s$ be a property shape definition, with $r_s$ the value for $sh:\text{path}$ in $d_s$.

1.2.4.1 $\text{loc}(d_s)$. Within $\text{cons}(d_s)$, we first isolate the set $\text{loc}(d_s)$ of triples with property
$sh:\text{hasValue, sh:not, sh:and, sh:one or sh:or}$. If $\text{loc}(d_s) = \emptyset$, then $t(\text{loc}(d_s)) = 1$.
Otherwise, $t(\text{loc}(d_s)) = \bigwedge_{m \in \text{loc}(d_s)} t(m)$.

1.2.4.2 $\text{nsRef}(d_s)$. Let $\text{nsRef}(s)$ be the set of triples with property $sh:\text{node}$ in $d_s$.
Then $t(\text{nsRef}(d_s)) = \bigwedge_{(s,sh,\text{node,}s') \in \text{nsRef}(d_s)} s'$.

1.2.4.3 $\text{nsRefQ}(d_s)$. Similarly, let $\text{nsRefQ}(d_s)$ be the set of triples with property
$sh:\text{qualifiedValueShape}$ in $d_s$.
Then $t(\text{nsRefQ}(d_s)) = \bigwedge_{(s,sh,\text{qualifiedValueShape,}s') \in \text{nsRefQ}(d_s)} s'$.

1.2.4.4 $\text{eq}(d_s)$. Let $\text{eq}(d_s)$ be the set of triples with property $sh:\text{equals}$. Then
we define $t(\text{eq}(d_s)) = \bigwedge_{(s,sh,\text{equals,}r) \in \text{eq}(d_s)} r_s = r$.

1.2.4.5 $\text{disj}(d_s)$. Let $\text{disj}(d_s)$ be the set of triples with property $sh:\text{disjoint}$. Then
we define $t(\text{disj}(d_s)) = \bigwedge_{(s,sh,\text{disjoint,}r) \in \text{disj}(d_s)} (\leq 0,r_s \land \leq 0,\neg s^2_r)$,
where $s^1_r$ and $s^2_r$ are fresh shape names, whose definitions in $L$ are $\phi_{s^1_r} = \neg s^2_r$ and
$\phi_{s^2_r} = \neg s^1_r$.

1.2.4.6 $\text{quant}(d_s)$. For quantification, let $\text{quant}(d_s)$ be the set of triples in $d_s$ with
property path $sh:\text{minCount or sh:maxCount}$. For each triple $(s,p,n) \in \text{quant}(d_s)$, we define:
$t(s,p,n) = \geq n.r_s$ if $p$ is $sh:\text{minCount}$, and
$t(s,p,n) = \leq n.r_s$ if $p$ is $sh:\text{maxCount}$.

1.2.4.7 $\text{quantQ}(d_s)$. Similarly, let $\text{quantQ}(d_s)$ be the set of triples in $d_s$ with
property path $sh:\text{qualifiedMinCount or sh:qualifiedMaxCount}$. For each triple $(s,p,n) \in \text{quantQ}(d_s)$, we define:
$t(s,p,n) = \geq n.r_s$ if $p$ is $sh:\text{qualifiedMinCount}$, and
$t(s,p,n) = \leq n.r_s$ if $p$ is $sh:\text{qualifiedMaxCount}$.

1.2.4.8 $\text{sib}(s)$. Finally, let $\text{sib}(s)$ be the set of sibling shapes of $s$ if $d_s$ contains the
triple $(s,sh,:\text{qualifiedValueShapesDisjoint,}true)$, and $\text{sib}(s) = \emptyset$ otherwise.

1.2.4.9 Translation of $d_s$. We are now ready to define the translation $t(d_s)$ of $\text{cons}(d_s)$ into $L$. For readability, we group the translation of constraints without qualified value, defined as:

$$nqual(d_s) = t(\text{loc}(d_s)) \land t(\text{nsRef}(d_s))$$
Semantics and Validation of Recursive SHACL

Then \( t(d_s) \) is defined as:

\[
    t(d_s) = t(eq(d_s)) \land t(disj(d_s)) \land \ldots
\]

If \( \phi = \geq n.r \phi' \), then \( h(\phi) \) is defined as:

\[
    s_\phi
    \text{a sh:NodeShape ;}
    \text{sh:property [}
        \text{sh:path (r)} ;
        \text{sh:qualifiedValueShape ;}
    \text{]}
\]

We also use shape definitions whose size is linear in simple, and sufficient for the purpose of this article, as it produces a set \( D \) of SHACL shape definitions for has target \( h \) contains one SHACL shape definition is partitioned into \( \{ \phi \} \) each.

Let \( S = \{ (s_1, \phi_{s_1}, \text{target}_{s_1}), \ldots, (s_n, \phi_{s_n}, \text{target}_{s_n}) \} \) be a stratified set of shapes, where each \( \phi_{s_i} \) is a formula in \( \mathcal{L} \).

We provide a translation of \( S \) as an equivalent set \( D \) of SHACL shape definitions. \( D \) is partitioned into \( \{ u(\phi_{s_1}), \ldots, u(\phi_{s_n}) \} \), where each \( u(\phi_{s_i}) \) translates \( \phi_{s_i} \).

Let \( \text{subf}(\phi_{s_i}) \) be the set of all subformulas of \( \phi_{s_i} \) (including \( \phi_{s_i} \) itself). Then \( u(\phi_{s_i}) \) contains one SHACL shape definition \( h(\phi) \) for each \( \phi \in \text{subf}(\phi_{s_i}) \). Furthermore, \( h(\phi_{s_i}) \) has target \( s_i \) as target definition, whereas all other \( h(\phi) \) in \( u(\phi_{s_i}) \) have no target definition.

It should be emphasized that this translation is far from optimal, i.e. a more concise set of SHACL shape definitions for \( S \) could be produced in general. But it is conceptually simple, and sufficient for the purpose of this article, as it produces a set \( D \) of SHACL shape definitions whose size is linear in \( |S| \).

As a reminder, \( \mathcal{L} \) is defined by the following grammar:

\[
    \phi ::= \top \mid s \mid I \mid \phi_1 \land \phi_2 \mid \neg \phi \mid \geq n.r \phi \mid r_1 = r_2
\]

If \( d \) is a SHACL shape definition for node shape \( s \) we will use \( \text{sh}(d) \) to denote \( s \). We also use \( [s_1, s_2] \) to denote a SHACL list containing 2 SHACL shapes (i.e. IRIs) \( s_1 \) and \( s_2 \).

We are now ready to define \( h(\phi) \), by induction on \( \phi \):

- If \( \phi = \top \), then \( h(\phi) \) is a shape definition with no constraint, i.e.:
  \[
  s_\phi \text{a sh:NodeShape }.
  \]
- If \( \phi = s \), then \( h(\phi) \) is defined as:
  \[
  s_\phi \text{a sh:NodeShape ; sh:hasValue } I .
  \]

This workaround is due to the fact that the SHACL syntax does not provide an operator to immediately reference a node shape within a node shape definition.

- If \( \phi = I \), then \( h(\phi) \) is defined as:
  \[
  s_\phi \text{a sh:NodeShape ; sh:hasValue } I .
  \]
- If \( \phi = \phi_1 \land \phi_2 \), then \( h(\phi) \) is defined as:
  \[
  s_\phi \text{a sh:NodeShape ; sh:and } (\{ \text{sh}(h(\phi_1)), \text{sh}(h(\phi_2)) \}).
  \]
- If \( \phi = \neg \phi' \), then \( h(\phi) \) is defined as:
  \[
  s_\phi \text{a sh:NodeShape ; sh:not } \text{sh}(h(\phi_1)) .
  \]
- If \( \phi = \geq n.r \phi' \), then \( h(\phi) \) is defined as:
  \[
  s_\phi \text{a sh:NodeShape ; sh:property [}
      \text{sh:path (r)} ;
      \text{sh:qualifiedValueShape ;}
  \text{]}
  \]
A qualified value shape is used here to correctly capture quantification, namely the fact that there may be $r$-successors not satisfying $\phi'$, as long as the number satisfying $\phi'$ is at least $n$. This is why we cannot use $\text{sh:minCount}$ only.

- If $\phi$ is $r_1 = r_2$, then $h(\phi)$ is defined as:

```
  a sh:NodeShape ;
  sh:property [ sh:path (r1) ; sh:equals (r2) ] .
```

Note that we allow full SPARQL property paths, but the specification only allows a subset of them, called SHACL paths. For our complexity results this is not an issue, as all lower bounds are shown without using property paths. Nevertheless, we can use $\text{sh:closed}$ to show that the logic has the same expressive power, albeit with a much more involved translation.

2 Proofs

2.1 Preliminaries

2.1.1 Numbering

The results presented in the above article and in this document follow different numbering schemes. The following is a mapping from propositions in the submitted version to propositions in the current document:

- Lemma 1 $\mapsto$ Lemma 2
- Proposition 1 $\mapsto$ Proposition 3
- Proposition 2 $\mapsto$ Proposition 1
- Proposition 4 $\mapsto$ Proposition 2
- Proposition 5 $\mapsto$ Proposition 6
- Proposition 6 $\mapsto$ Proposition 4
- Proposition 7 $\mapsto$ Proposition 5

2.1.2 Notation

In order to provide a higher-level introduction, the notation adopted in the submitted version slightly differs from the one used in the following proofs.

The main differences are the following:

- In what follows, the input of the validation problem is a triple $\langle G, S, s_0(v_0) \rangle$, with $G$ a graph, $S = \{ s_0 \mapsto \phi_{s_0}, \ldots, s_n \mapsto \phi_{s_n} \}$ is a function mapping shape names to formulas in $\mathcal{L}$, and $s_0(v_0)$ is the unique target atom, meaning that shape $s_0$ has vertex $v_0$ as unique “target node”. From Proposition 3 (in the submitted version), this is w.l.o.g.

As a shortcut, we may refer to $S$ as a “set of shapes”. If there is no ambiguity, we may also use “(let/there exists/for all) $\phi_s \in S$” as a shortcut for “(let/there exists/for all) $\phi_s$ such that $S(s) = \phi_s$ for some $s \in \text{dom}(S)$”.
– $G$ is represented as a pair $(V_G,E_G)$, with $V_G$ its vertices, and $E_G \subseteq V_G \times P \times V_G$ its edges, where $P$ is a set of edge labels (IRIS).

– Assignments are primarily represented as functions from atoms to truth values (defined in Section 2.1.4 below)

– As syntactic sugar, we use $\diamond \phi$ for $\geq 1.r \phi$, and $\square \phi$ for $\leq 0.r \neg \phi$.

### 2.1.3 Function, lattice, fixed point

– If $f$ is a function, then $\text{dom}(f)$ designates its domain, and $\text{range}(f)$ its range.

– If $A \subseteq \text{dom}(f)$, then $f|_A$ designates $f$ restricted to $A$.

– If $f$ is a function, and $A \subseteq \text{dom}(f)$, then $\text{fix}(f, A)$ designates all fixed points of $f$ over $A$.

– If $P = \langle U, \preceq \rangle$ is a partially ordered set and $u \in U$, then $\text{ext}_P(u)$ designates $\{u' \in U \mid u \preceq u'\}$.

**Definition 1.** Let $P = \langle U, \preceq \rangle$ be a partially ordered set, and $f$ a function from $U$ to $U$. $u$ is the least fixed point of $f$ over $P$ if $u \in \text{fix}(f, U)$, and $u \preceq u'$ for all $u' \in \text{fix}(f, U)$.

**Definition 2.** Let $P = \langle U, \preceq \rangle$ be a partially ordered set. A function $f : U \rightarrow U$ is monotone over $P$ if for all $u \in U$, $u \preceq f(u)$.

We will use a weaker version of the Knaster-Tarski theorem:

**Theorem 1.** If $P = \langle U, \preceq \rangle$ is a meet semi-lattice and $f : U \rightarrow U$ a monotone function over $P$, then $f$ has a (unique) least fixed point over $P$.

### 2.1.4 Shape assignments

If $G$ is a graph and $S$ a set of shape constraint definitions, then $\text{atoms}(G, S) = \{s(v) \mid \phi \in S \text{ and } v \in V_G\}$.

**Definition 3.** Given a graph $G$ and a set $S$ of shape constraint definitions, a (3-valued) shape assignment $\sigma$ for $G$ and $S$ is a total function from $\text{atoms}(G, S)$ to $\{0, 0.5, 1\}$.

**Definition 4.** Given a graph $G$ and a set $S$ of shape constraint definitions, a 2-valued shape assignment $\sigma$ for $G$ and $S$ is a total function from $\text{atoms}(G, S)$ to $\{0, 1\}$.

For readability, for shape assignment $\sigma$, shape name $s$ and vertex $v$, we write $\sigma s(v)$ instead of $\sigma(s(v))$.

Alternatively, for the sake of brevity, an assignment $\sigma$ may be represented as a set of positive and negative atoms, i.e. $s(v) \in \sigma$ iff $\sigma s(v) = 1$, and i.e. $\neg s(v) \in \sigma$ iff $\sigma s(v) = 0$.

If $S$ is a set of shape constraint definitions, then $\text{signat}(S)$ designates all predicates and constants appearing in some constraint formula of $S$.

**Definition 5** (Target compliant assignment). $\sigma$ is a target compliant assignment for $(G, S, s_0(v_0))$ iff:

– $\sigma$ is an assignment for $G$ and $S$, and

– $\sigma s_0(v_0) = 1$
Definition 6 (Constraint satisfying assignment). 
\(\sigma\) is a constraint satisfying assignment for a graph \(G\) and set \(S\) of shapes iff:
- \(\sigma\) is an assignment for \(G\) and \(S\), and
- for each \(s(v) \in \text{atoms}(G, S)\), \(\sigma s(v) = 0\) implies \(\llbracket \phi_s \rrbracket^{v,G,\sigma} = 0\), and \(\sigma s(v) = 1\) implies \(\llbracket \phi_s \rrbracket^{v,G,\sigma} = 1\).

Definition 7. (Fixed-point assignment) 
\(\sigma\) is a fixed-point assignment for a graph \(G\) and set \(S\) of shapes iff:
- \(\sigma\) is an assignment for \(G\) and \(S\), and
- for each \(s(v) \in \text{atoms}(G, S)\), \(\sigma s(v) = \llbracket \phi_s \rrbracket^{v,G,\sigma}\)

Definition 8 (Faithful assignment). 
\(\sigma\) is a faithful assignment for \(\langle G, S, s_0(v_0) \rangle\) iff it is both target compliant for \(\langle G, S, s_0(v_0) \rangle\) and constraint satisfying for \(G\) and \(S\).

Definition 9 (Strictly faithful assignment). 
\(\sigma\) is a strictly faithful assignment for \(\langle G, S, s_0(v_0) \rangle\) iff it is both target compliant for \(\langle G, S, s_0(v_0) \rangle\) and a fixed-point assignment for \(G\) and \(S\).

2.1.5 Sets of assignments

Given a graph \(G\), set \(S\) of shapes and \(n \in \{2, 3\}\):
- \(\Sigma^n_{G,S}\) designates all \(n\)-valued assignments for \(G\) and \(S\)
- \(\Sigma^n_{G,S,s_0}(v_0)\) designates all \(n\)-valued target compliant assignments for \(G, S\) and \(s_0(v_0)\)
- \(\Sigma^n_{G,S,cs}(v_0)\) designates all \(n\)-valued constraint satisfying assignments for \(G\) and \(S\)
- \(\Sigma^n_{G,S,fp}(v_0)\) designates all \(n\)-valued fixed-point assignments for \(G\) and \(S\)
- \(\Sigma^n_{G,S,fa}(v_0)\) designates all \(n\)-valued faithful assignments for \(G, S\) and \(s_0(v_0)\)
- \(\Sigma^n_{G,S,strict}(v_0)\) designates all \(n\)-valued strictly faithful assignments for \(G, S\) and \(s_0(v_0)\)

Similarly, \(\Sigma^0_S\) designates all \(n\)-valued assignments for \(S\) and any \(G\).

2.1.6 Assignment ordering

Given a graph \(G\) and set \(S\) of shapes, \(\preceq\) denotes set inclusion between 3-valued assignments viewed as sets of (possibly negated) atoms.

In other words, for \(\sigma_1, \sigma_2 \in \Sigma^3_{G,S}\), \(\sigma_1 \preceq \sigma_2\) iff for any \(s(v) \in \text{atoms}(G, S)\):
- \(\sigma_1 s(v) = 0\) implies \(\sigma_2 s(v) = 0\), and
- \(\sigma_1 s(v) = 1\) implies \(\sigma_2 s(v) = 1\).

Note that \(L = \langle \Sigma^3_{G,S}, \preceq \rangle\) is a (meet) semi-lattice over \(\Sigma^3_{G,S}\); the greatest lower bound of two elements of \(\Sigma^3_{G,S}\) (viewed as sets) is their intersection, but they may not have a least upper bound (e.g. for \(\sigma_1 = \{s(v)\}\) and \(\sigma_2 = \{\neg s(v)\}\)).

Furthermore, for any \(\sigma \in \Sigma^3_{G,S}\), \((\text{ext}_L(\sigma), \preceq)\) is also a (meet) semi-lattice over \(\text{ext}_L(\sigma)\).

Definition 10. If \(\sigma \in \Sigma^3_{G,S}\), then \(\text{fil}(\sigma) \in \Sigma^2_{G,S}\) is the assignment defined by \(\text{fil}(\sigma)s(v) = 1\) if \(\sigma s(v) = 0.5\), and \(\text{fil}(\sigma)s(v) = \sigma s(v)\) otherwise.
2.1.7 Stratification
In what follows, all shape formulas are assumed to be in NNF.

**Definition 11.** A set $S$ of shapes constraint definitions is stratified if there is function $\text{str} : S \rightarrow \mathbb{N}$ such that:
- If $s_1$ appears in $\phi_{s_2}$, then $\text{str}(s_1) \leq \text{str}(s_2)$
- If $s_1$ appears in $\phi_{s_2}$ in the scope of a negation, then $\text{str}(s_1) < \text{str}(s_2)$

If $S$ is a stratified set of shapes constraint definitions with strata $S_1, \ldots, S_n$ (from lowest to highest), we use $S_{\leq j}$ to designate $\bigcup_{i=1}^{j} S_i$.

2.1.8 Immediate evaluation

**Definition 12.** Given a graph $G$ and set $S$ of shapes, the function $T^{G,S} : \Sigma_{G,S}^n \rightarrow \Sigma_{G,S}^n$ is defined by:
- For readability, we write $T^{G,S} \sigma$ instead of $T^{G,S}(\sigma)$.
- Abusing notation, if $\Sigma' \subseteq \Sigma_{G,S}^n$, we use $\{T^{G,S} \sigma | \sigma \in \Sigma'\}$.

2.1.9 Constraint language and fragments

2.1.9.1 $\mathcal{L}$ As a reminder, the language $\mathcal{L}$ for shape constraints is defined by the following grammar.

\[ \phi ::= \top | s | I | \phi_1 \land \phi_2 | \neg \phi | \geq n.r \phi | r_1 = r_2 \]  

(1)

where $s$ is a shape name, $I$ is an IRI, $r$ is a property path, and $n \in \mathbb{N}^+$.

The following additional operators are used as syntactic sugar:
- $\phi_1 \lor \phi_2$ for $\neg (\neg \phi_1 \land \neg \phi_2)$
- $\leq n.r \phi$ for $\neg (\geq (n + 1).r \phi)$
- $= n.r \phi$ for $\geq n.r \phi \land \leq n.r \phi$,
- $\downarrow r \phi$ for $\geq 1.r \phi$
- $\square r \phi$ for $\leq 0.r \neg \phi$
- $\bot$ for $\neg \top$

2.1.9.2 $\mathcal{L}_{\neg,\land,\lor}$ The constraint language $\mathcal{L}_{\neg,\land,\lor}$ is used for NP-hardness. It is the fragment of $\mathcal{L}$ without property path, counting or path equality, i.e. defined with the grammar:

\[ \phi ::= \top | s | I | \phi_1 \land \phi_2 | \neg \phi | \geq 1.p \phi \]  

(2)

where $s$ is a shape name, and $I$ and $p$ are IRIs.

2.1.9.3 $\mathcal{L}_{\lor,\land,\neg}$ The constraint language $\mathcal{L}_{\lor,\land,\neg}$ is used for P-hardness. It is the fragment of $\mathcal{L}$ without property path, path equality, or negation, but with disjunction as a primitive operator. i.e. defined with the grammar:

\[ \phi ::= \top | s | I | \phi_1 \land \phi_2 | \phi_1 \lor \phi_2 | \geq n.p \phi \]  

(3)

where $s$ is a shape name, $I$ and $p$ are IRIs, and $n \in \mathbb{N}^+$. 
2.2 Semantic Properties: proofs

**Lemma 1.** Let \( G \) be a graph, \( S \) a set of shape constraint definitions, and let \( \sigma_1, \sigma_2 \in \Sigma_{G,S}^3 \), with \( \sigma_1 \preceq \sigma_2 \). For any \( \phi \) over \( \text{signat}(S) \) and \( v \in V_G \), if \( \llbracket \phi \rrbracket^{v,G,\sigma_1} \neq 0.5 \), then \( \llbracket \phi \rrbracket^{v,G,\sigma_1} = \llbracket \phi \rrbracket^{v,G,\sigma_2} \).

**Proof.** By induction on \( \phi \).

\[
\phi = \top : \\
\llbracket \top \rrbracket^{v,G,\sigma_1} = \llbracket \top \rrbracket^{v,G,\sigma_2} = 1
\]

\[
\phi = I : \\
\text{If } \llbracket I \rrbracket^{v,G,\sigma_1} = 1, \text{ then } v = I, \text{ therefore } \llbracket I \rrbracket^{v,G,\sigma_2} = 1. \\
\text{If } \llbracket I \rrbracket^{v,G,\sigma_1} = 0, \text{ then } v \neq I, \text{ therefore } \llbracket I \rrbracket^{v,G,\sigma_2} = 0.
\]

\[
\phi = s : \\
\text{Let } \sigma_1 s(v) = 0. \text{ Because } \sigma_1 \preceq \sigma_2, \sigma_2 s(v) = \llbracket s \rrbracket^{v,G,\sigma_2} = 0. \\
\text{Similarly for the case } \sigma_1 s(v) = 1.
\]

\[
\phi = \phi_1 \land \phi_2 : \\
\text{If } \llbracket \phi_1 \rrbracket^{v,G,\sigma_1} = 0, \text{ then } \llbracket \phi_1 \rrbracket^{v,G,\sigma_1} = 0 \text{ or } \llbracket \phi_2 \rrbracket^{v,G,\sigma_1} = 0. \\
\text{So by IH, } \llbracket \phi_1 \rrbracket^{v,G,\sigma_2} = 0 \text{ or } \llbracket \phi_2 \rrbracket^{v,G,\sigma_2} = 0. \\
\text{Therefore } \llbracket \phi \rrbracket^{v,G,\sigma_1} = 0.
\]

\[
\phi = \phi_1 \lor \phi_2 : \\
\text{If } \llbracket \phi_1 \rrbracket^{v,G,\sigma_1} = 0, \text{ then } \llbracket \phi_1 \rrbracket^{v,G,\sigma_1} = 0 \text{ or } \llbracket \phi_2 \rrbracket^{v,G,\sigma_1} = 0. \\
\text{So by IH, } \llbracket \phi_2 \rrbracket^{v,G,\sigma_2} = 0. \\
\text{Therefore } \llbracket \phi \rrbracket^{v,G,\sigma_1} = 0.
\]

\[
\phi = \neg \phi' : \\
\text{If } \llbracket \phi \rrbracket^{v,G,\sigma_1} = 0, \text{ then } \llbracket \phi' \rrbracket^{v,G,\sigma_1} = 1. \\
\text{So by IH, } \llbracket \phi' \rrbracket^{v,G,\sigma_2} = 1. \\
\text{Therefore } \llbracket \phi \rrbracket^{v,G,\sigma_1} = 0.
\]

\[
\phi \geq n.r \phi' : \\
\text{If } \llbracket \phi \rrbracket^{v,G,\sigma_1} = 1, \text{ then } |\{v' \mid G \models r(v,v') \text{ and } \llbracket \phi \rrbracket^{v',G,\sigma_1} = 1\}| \geq n.
\]

By IH, if \( \llbracket \phi \rrbracket^{v',G,\sigma_1} = 1 \), then \( \llbracket \phi \rrbracket^{v',G,\sigma_2} = 1 \).

\[
|\{v' \mid G \models r(v,v') \text{ and } \llbracket \phi \rrbracket^{v',G,\sigma_1} = 1\}| \geq n.
\]

By IH, if \( \llbracket \phi \rrbracket^{v',G,\sigma_1} = 1 \), then \( \llbracket \phi \rrbracket^{v',G,\sigma_2} = 1 \).

\[
|\{v' \mid G \models r(v,v') \text{ and } \llbracket \phi \rrbracket^{v',G,\sigma_1} = 1\}| \geq n.
\]

And by IH, if \( \llbracket \phi \rrbracket^{v',G,\sigma_1} = 0 \), then \( \llbracket \phi \rrbracket^{v',G,\sigma_2} = 0 \).

\[
|\{v' \mid G \models r(v,v') \text{ and } \llbracket \phi \rrbracket^{v',G,\sigma_1} = 0\}| \leq |\{v' \mid G \models r(v,v') \text{ and } \llbracket \phi \rrbracket^{v',G,\sigma_2} = 0\}|.
\]

It follows that \( |\{v' \mid G \models r(v,v')\}| - |\{v' \mid G \models r(v,v') \text{ and } \llbracket \phi \rrbracket^{v',G,\sigma_1} = 0\}| \leq |\{v' \mid G \models r(v,v')\}| - |\{v' \mid G \models r(v,v') \text{ and } \llbracket \phi \rrbracket^{v',G,\sigma_1} = 0\}| < n.
\]

Therefore \( \llbracket \phi \rrbracket^{v,G,\sigma_2} = 0 \).

**Lemma 2.** For any graph \( G \) and set \( S \) of shapes, \( T^{G,S} \) is monotone over \( \langle \Sigma_{G,S}^3, \preceq \rangle \).

**Proof.** For each \( s \in \text{dom}(S) \), \( \phi_s \) will designate \( S(s) \).

Let \( \sigma_1, \sigma_2 \in \Sigma_{G,S}^3 \), with \( \sigma_1 \preceq \sigma_2 \).
Take any $s(v) \in \text{atoms}(G,S)$. From Lemma 1, if $[\phi_s]^{v,G,\sigma_1} = 0$, then $[\phi_s]^{v,G,\sigma_2} = 0$.
Similarly, if $[\phi_s]^{v,G,\sigma_1} = 1$, then $[\phi_s]^{v,G,\sigma_2} = 1$.
Then from Definition 12, $T^{G,S}\sigma_1 s(v) = [\phi_s]^{v,G,\sigma_1}$.
Therefore $T^{G,S}\sigma_1 \preceq T^{G,S}\sigma_2$. \qed

**Lemma 3.** Let $G$ be a graph, $S$ a set of shapes, and $L = (\Sigma^3_{G,S}, \preceq)$.

For any $\sigma \in \Sigma^3_{G,S}$, $\sigma \preceq T^{G,S}\sigma$ iff $T^{G,S}\text{ext}_L(\sigma) \preceq \text{ext}_L(\sigma)$.

**Proof.**

$\Rightarrow$

Let $\sigma \in \Sigma^3_{G,S}$, and $\sigma \preceq T^{G,S}\sigma$. We need to show that $T^{G,S}\sigma' \in \text{ext}_L(\sigma)$ for any $\sigma' \in \text{ext}_L(\sigma)$.

Because $\sigma' \in \text{ext}_L(\sigma)$, $\sigma \preceq \sigma'$.

So from Lemma 2, $T^{G,S}\sigma \preceq T^{G,S}\sigma'$.

Then as $\sigma \preceq T^{G,S}\sigma$, from the transitivity of $\preceq$, $\sigma \preceq T^{G,S}\sigma'$.

Therefore $T^{G,S}\sigma' \in \text{ext}_L(\sigma)$.

$\Leftarrow$

Let $T^{G,S}\text{ext}_L(\sigma) \preceq \text{ext}_L(\sigma)$.

Then for each $\sigma' \in \text{ext}_L(\sigma)$, $T^{G,S}\sigma' \in \text{ext}_L(\sigma)$.

In particular, because $\sigma \in \text{ext}_L(\sigma)$, $T^{G,S}\sigma \in \text{ext}_L(\sigma)$.

So from the definition of $\text{ext}_L(\sigma)$, $\sigma \preceq T^{G,S}\sigma$. \qed

**Proposition 1.** For any graph $G$, set $S$ of shapes and $s_0(v_0) \in \text{atoms}(G,S)$, if there is a $\sigma \in \Sigma^3_{G,S}$ such that $\sigma s_0(v_0) = 1$, then there is a $\sigma' \in \Sigma^3_{G,S}$ such that $\sigma'^s_0(v_0) = 1$.

**Proof.** For each $s \in \text{dom}(S)$, $\phi_s$ will designate $S(s)$.

Let $L = (\Sigma^3_{G,S}, \preceq)$, and let $\sigma \in \Sigma^3_{G,S}$, such that $\sigma s_0(v_0) = 1$.

Take any $s(v) \in \text{atoms}(G,S)$.

- If $\sigma s(v) = 0$, because $\sigma \in \Sigma^3_{G,S}$, $[\phi_s]^{v,G,\sigma} = T^{G,S}\sigma s(v) = 0$.
- Similarly, if $\sigma s(v) = 1$, then $[\phi_s]^{v,G,\sigma} = T^{G,S}\sigma s(v) = 1$.

Therefore $\sigma \preceq T^{G,S}\sigma$.

So from Lemma 3, $T^{G,S}\text{ext}_L(\sigma) \preceq \text{ext}_L(\sigma)$, i.e. $T^{G,S}|_{\text{ext}_L(\sigma)}$ is a function from $\text{ext}_L(\sigma)$ to $\text{ext}_L(\sigma)$.

In addition, because $\text{ext}_L(\sigma) \subseteq \Sigma^3_{G,S}$, from Lemma 2, $T^{G,S}$ is monotone over $(\text{ext}_L(\sigma), \preceq)$.

Therefore from Theorem 1, $T^{G,S}$ has a minimal fixpoint $\sigma'$ over $(\text{ext}_L(\sigma), \preceq)$.

So $\sigma' \in \Sigma^3_{G,S}$. And because $\sigma' \in \text{ext}_L(\sigma)$, $\sigma'^s_0(v_0) = 1$. \qed

**Lemma 4.** Let $S$ be a stratified set of shapes, with strata $S_1, ..., S_n$ (from lowest to highest), and let $S_0 = \emptyset$.

For $1 \leq i \leq n$, if $\sigma \in \Sigma^3_{G,S_{i-1}}$, and $\sigma|_{S_{i-1}} \in \Sigma^2_{G,S_{i-1}}$, then $\text{fill}(\sigma)|_{S_{i-1}} \in \Sigma^2_{G,S_{i-1}}$.

**Proof.** For each $s \in \text{dom}(S)$, $\phi_s$ will designate $S(s)$.

Let $\sigma \in \Sigma^3_{G,S_{i-1}}$, with $\sigma|_{S_{i-1}} \in \Sigma^2_{G,S_{i-1}}$, let $\sigma' = \text{fill}(\sigma)|_{S_{i-1}}$, and let $s(v) \in S_{i-1}$.

We need to show that $\sigma' s(v) = [\phi_s]^{v,G,\sigma'}$. We first consider the case $\sigma s(v) \neq 0.5$, and then $\sigma s(v) = 0.5$. 


\(\sigma s(v) \neq 0.5:\)

Because \(\sigma \in \Sigma^{3,\text{fix}}_{G,S,S}, \sigma s(v) = [\phi_s]^{v,G,\sigma} \).

Then because \(\sigma \preceq \sigma'\) and \(\sigma s(v) \neq 0.5\), from Lemma 1, \([\phi_s]^{v,G,\sigma} = [\phi_s]^{v,G,\sigma'}\).

Finally, since \(\sigma \preceq \sigma'\) and \(\sigma s(v) \neq 0.5\), \(\sigma' s(v) = \sigma s(v)\).

This yields \(\sigma' s(v) = \sigma s(v) = [\phi_s]^{v,G,\sigma} = [\phi_s]^{v,G,\sigma'}\).

\(\sigma s(v) = 0.5:\)

Because \(\sigma' = \text{fil}(\sigma|_{S \leq 3}), \sigma' s(v) = 1\). So we need to show that \([\phi_s]^{v,G,\sigma'} = 1\).

First, because \(\sigma s(v) = 0.5\) and \(\sigma \in \Sigma^{3,\text{fix}}_{G,S,S}, [\phi_s]^{v,G,\sigma} = 0.5\) must hold.

Then we show below that for any \(w \in V_G\) and subformula \(\phi\) of \(\phi_s\), if \([\phi]^{w,G,\sigma} = 0\), then \([\phi]^{w,G,\sigma'} = 0\), and \([\phi]^{w,G,\sigma'} = 1\) otherwise.

In particular, because \([\phi_s]^{v,G,\sigma} \neq 0\), this implies \([\phi_s]^{v,G,\sigma'} = 1\).

By induction on \(\phi\):

\(\phi = \top:\)

\([\top]^{w,G,\sigma} = [\top]^{w,G,\sigma'} = 1\)

\(\phi = I:\)

If \(w = I\), then \([I]^{w,G,\sigma} = [I]^{w,G,\sigma'} = 1\).

If \(w \neq I\), then \([I]^{w,G,\sigma} = [I]^{w,G,\sigma'} = 0\).

\(\phi = s:\)

If \([s]^{w,G,\sigma} = 0\), then \(\sigma s(w) = 0\), and because \(\sigma \preceq \sigma'\), \([s]^{w,G,\sigma'} = \sigma' s(w) = 0\).

Similarly, if \([s]^{w,G,\sigma} = 1\), then \([s]^{w,G,\sigma'} = \sigma' s(w) = 1\).

If \([s]^{w,G,\sigma} = 0.5\), then \(\sigma s(w) = 0.5\), and because \(\sigma' = \text{fil}(\sigma|_{S \leq 3}), \sigma' s(w) = 1\).

Therefore \([s]^{w,G,\sigma'} = 1\).

\(\phi = \phi_1 \land \phi_2:\)

If \([\phi_1]^{w,G,\sigma} = 0\), then \([\phi_j]^{w,G,\sigma} = 0\) must hold for some \(j \in \{1, 2\}\).

So by IH. \([\phi_j]^{w,G,\sigma'} = 0\).

Therefore \([\phi]^{w,G,\sigma'} = 0\).

If \([\phi_1]^{w,G,\sigma} \geq 0.5\), then \([\phi_j]^{w,G,\sigma} \geq 0.5\) must hold for all \(j \in \{1, 2\}\).

So by IH. \([\phi_j]^{w,G,\sigma'} = 1\).

Therefore \([\phi]^{w,G,\sigma'} = 1\).

\(\phi = \neg \phi':\)

Because \(S\) is stratified and \(\phi'\) is a subformula of \(\phi_s\), for any shape name \(s'\) appearing in \(\phi'\), \(s'\) must be defined in \(S_{\leq i-1}\).

Then because \(\sigma|_{S_{\leq i-1}} \in \Sigma^{3,\text{fix}}_{G,S_{\leq i-1}}, \) for any \(v \in V_G, \sigma s'(v) \neq 0.5\). So by induction on the structure of \(\phi', [\phi']^{w,G,\sigma} \neq 0.5\), which implies \([\phi]^{w,G,\sigma} \neq 0.5\).

Therefore the only two possible cases are \([\phi]^{w,G,\sigma} = 0\) and \([\phi]^{w,G,\sigma} = 1\).

If \([\phi]^{w,G,\sigma} = 0\), then \([\phi']^{w,G,\sigma} = 1\) must hold.

So by IH. \([\phi']^{w,G,\sigma'} = 1\).

Therefore \([\phi']^{w,G,\sigma'} = 0\).

If \([\phi]^{w,G,\sigma} = 1\), then \([\phi']^{w,G,\sigma} = 0\) must hold.

So by IH. \([\phi']^{w,G,\sigma'} = 0\).

Therefore \([\phi']^{w,G,\sigma'} = 1\).
$\phi \equiv \forall n. r \phi'$

Let $Y = \{ y \mid G \models r(w, y) \}$, $Y_0^0 = \{ y \in Y \mid [\phi']^G, \sigma = 0 \}$, $Y_0^{0.5} = \{ y \in Y \mid [\phi']^G, \sigma = 0.5 \}$, and $Y_1^1 = \{ y \in Y \mid [\phi']^G, \sigma = 1 \}$.

Similarly, define $Y_0^0, Y_0^{0.5}$ and $Y_1^1$.

If $[\phi']^G, \sigma = 0$, then $|Y| - |Y_0^0| < n$.

By IH, for each $y \in Y_0^0$, $[\phi']^G, \sigma' = 0$.

So $|Y| - |Y_0^0| < n$.

Therefore $[\phi']^G, \sigma' = 0$.

If $[\phi']^G, \sigma \geq 0.5$, then $|Y| - (|Y_0^{0.5}| + |Y_1^1|) \geq n$.

By IH, for each $y \in Y_0^{0.5} \cup Y_1^1$, $[\phi']^G, \sigma' = 1$.

So $|Y_0^{0.5}| + |Y_1^1| = |Y_1^1|$.

This yields $|Y| - |Y_1^1| \geq n$.

Therefore $[\phi']^G, \sigma' = 1$.  

\[ \square \]

**Proposition 2.** For any graph $G$, stratified set $S$ of shapes and $s_0(v_0) \in \text{atoms}(G, S)$, if there is a $\sigma \in \Sigma^3_{G, S}$ such that $\sigma s_0(v_0) = 1$, then there is a $\sigma' \in \Sigma^2_{G, S}$ such that $\sigma' s_0(v_0) = 1$.

**Proof.** Let $S$ be a stratified set of shapes, with strata $S_1, \ldots, S_n$, let $G$ be a graph, let $L = (\Sigma^3_{G, S}, \preceq)$, let $s_0(v_0) \in \text{atoms}(G, S)$, and let $\sigma \in \Sigma^3_{G, S}$ such that $\sigma s_0(v_0) = 1$.

For each $s \in \text{dom}(S)$, $\phi_s$ will designate $S(s)$.

\[ \sigma'|_{S_{i+1}} \text{ is defined by induction on } 1 \leq i \leq n, \text{ as follows:} \]

- $\sigma'|_{S_{i+1}} = \text{fil}(\sigma|_{S_i})$.
- Let $\tau_i = \sigma'|_{S_{i-1}} \cup \sigma|_{S_i}$, and let $\theta_i$ be the minimal fixpoint of $T^{G, S_{i+1}}$ over $(\text{ext}_L(\tau_i), \preceq)$ (we show below that $\theta_i$ must exist). Then $\sigma'|_{S_{i+1}} = \text{fil}(\theta_i)$.

We now show that $\sigma'|_{S_{i+1}} \in \Sigma^2_{G, S_{i+1}}$ for $1 \leq i \leq n$. It follows that $\sigma' = \sigma'|_{S_n} \in \Sigma^2_{G, S_n}$.

First, observe that for any $1 \leq i \leq n$, because $\theta_i$ is a fixpoint of $T^{G, S_{i+1}}$ over $(\text{ext}_L(\theta_i), \preceq)$, and because $\text{ext}_L(\theta_i) \subseteq \Sigma^3_{G, S_i}$, $\theta_i$ is also a fixpoint of $T^{G, S_{i+1}}$ over $(\Sigma^3_{G, S_i}, \preceq)$, i.e. $\theta_i \in \Sigma^3_{G, S_{i+1}}$.

So for the base case $i = 1$, because $\sigma'|_{S_{i+1}} = \text{fil}(\theta_i)$, from Lemma 4, $\sigma'|_{S_{i+1}} \in \Sigma^2_{G, S_{i+1}}$, must hold.

For the inductive case, by IH, $\sigma'|_{S_{i+1}} \in \Sigma^2_{G, S_{i+1}}$. In addition, $\tau_i|_{S_{i+1}} = \sigma'|_{S_{i+1}}$. Finally, because $\theta_i \in \text{ext}_L(\tau_i)$, $\tau_i \preceq \theta_i$ must hold. Therefore $\theta_i|_{S_{i+1}} = \tau_i|_{S_{i+1}} = \sigma'|_{S_{i+1}} \in \Sigma^2_{G, S_{i+1}}$. So from Lemma 4, $\sigma'|_{S_{i+1}} \in \Sigma^2_{G, S_{i+1}}$.

To complete the proof, we show that $\theta_i$ must exist (in the inductive case $i > 1$), i.e. that $T^{G, S_{i+1}}$ must admit a fixpoint over $(\text{ext}_L(\tau_i), \preceq)$.

From Lemma 3, it is sufficient to show that $\tau_i \preceq T^{G, S_{i+1}}(\tau_i)$, i.e. that for any $s(v) \in \text{atoms}(G, S_{i+1})$, if $s(v) \neq 0$, then $\tau_i s(v) = [\phi_v]^G, \tau_i$.

We will first consider the case where $s$ is defined in $S_{i+1}$, and then the case where $s$ is defined in $S_i$.

$s$ is defined in $S_{i+1}$

From the definition of $\tau_i$, $\tau_i|_{S_{i+1}} = \sigma'|_{S_{i+1}}$.  

For each $s \in \text{dom}(S)$, $\phi_s$ will designate $S(s)$.
Then because $S$ is stratified, $\left[\phi_{s}\right]_{v,G,\tau} = \left[\phi_{s}\right]_{v,G,\tau}^{G,S_{\leq i-1}}$.

Finally, because $\sigma'|_{S_{\leq i-1}} \in \Sigma_{G,S_{\leq i}}^{2,\text{fs}}$, $\sigma'|_{S_{\leq i-1}} s(v) = \left[\phi_{s}\right]_{v,G,\sigma'}^{G,S_{\leq i-1}}$.

This yields $\tau_{i}s(v) = \left[\phi_{s}\right]_{v,G,\tau_{i}}^{G,S_{\leq i}}$.

$s$ is defined in $S_{i}$. From the definition of $\sigma'$, $\sigma|_{S_{\leq i-1}} \preceq \sigma'|_{S_{\leq i-1}}$.

And from the definition of $\tau_{i}$, $\sigma'|_{S_{\leq i-1}} = \tau_{i}|_{S_{\leq i-1}}$.

Therefore $\sigma|_{S_{\leq i-1}} \preceq \tau_{i}|_{S_{\leq i-1}}$.

In addition, from the definition of $\tau_{i}$ still, $\sigma|_{S_{i}} = \tau_{i}|_{S_{i}}$.

This yields $\sigma|_{S_{\leq i}} \preceq \tau_{i}$.

Then because $\sigma \in \Sigma_{G,S_{\leq i}}^{3,\text{fs}}$, and $S$ is stratified, $\sigma|_{S_{\leq i}} \in \Sigma_{G,S_{\leq i}}^{3,\text{fs}}$.

So $\sigma|_{S_{\leq i}} s(v) = \left[\phi_{s}\right]_{v,G,\sigma}|_{S_{\leq i}}$.

Now let $\tau_{i}s(v) \neq 0.5$.

Because $\sigma|_{S_{i}} = \tau_{i}|_{S_{i}}$, $\sigma|_{S_{i}} s(v) \neq 0.5$.

And because $\sigma|_{S_{\leq i}} s(v) = \left[\phi_{s}\right]_{v,G,\sigma}|_{S_{\leq i}}^\Sigma_{G,S_{\leq i}} \neq 0.5$.

Then as $\sigma|_{S_{\leq i}} \preceq \tau_{i}$, from Lemma 1, $\left[\phi_{s}\right]_{v,G,\sigma}|_{S_{\leq i}} = \left[\phi_{s}\right]_{v,G,\tau}$.

This yields $\tau_{i}s(v) = \left[\phi_{s}\right]_{v,G,\tau_{i}}^{G,S_{\leq i}}$.

Proposition 3. Let $G$ be a graph, and $S$ a set of shape constraint definitions. Then $T_{G,S}^{G,S}$ admits a unique minimal fixed-point $\sigma$ over $\langle \Sigma_{G,S_{\leq i}}^{3}, \preceq \rangle$, and $\sigma$ is a constraint-compliant assignment for $G$ and $S$.

Proof. Because $\langle \Sigma_{G,S_{\leq i}}^{3}, \preceq \rangle$ is a meet semi-lattice and $\text{dom}(T_{G,S}^{G,S}) = \Sigma_{G,S_{\leq i}}^{3}$ and $\text{range}(T_{G,S}^{G,S}) \subseteq \Sigma_{G,S_{\leq i}}^{3}$ from Lemma 2 and Theorem 1, $T_{G,S}^{G,S}$ admits a unique minimal fixed point over $\langle \Sigma_{G,S_{\leq i}}^{3}, \preceq \rangle$.

Then from Definitions 6 and 7, a fixed-point assignment for $G$ and $S$ is also a constraint compliant assignment for $G$ and $S$.

\[Q.E.D.\]

2.3 Complexity Proofs

Proposition 4 (Data Complexity). For stratified $\mathcal{L}_{\phi,\neg,\wedge}$, VALIDATION is NP-hard in data complexity.

Proof. Reduction from CIRCUIT-SAT. Let $\psi$ be a boolean formula with variables $\{x_1, \ldots, x_n\}$. We assume w.l.o.g. that $\psi$ contains only AND and NOT as boolean operators.

We build an instance $(G_\psi, S, s_0(v_0))$ of VALIDATION such that $\psi$ is satisfiable iff $(G_\psi, S, s_0(v_0))$ is valid. The graph $G_\psi$ is similar to the propositional DAG for $\psi$, whereas $S$ is independent from $\psi$.

Let $\psi$ be the set of all subformulas of $\psi$. Each $\psi' \in \psi$ is injectively mapped to a vertex $v_{\psi'}$. Then $G_\psi$ is the graph defined by $V_{G_\psi} = \{v_0\} \cup \{v_{\psi'} \mid \psi' \in \psi\}$ and $E_{G_\psi} = \{(v_0, \text{eval}, v_{\psi'})\} \cup H_\psi$, where $H_\psi$ is defined by induction on $\psi$, as follows:

- if $\psi = x_i$, then $H_\psi = \{(v_{\psi}, \text{self}, v_{\psi})\}$
- if $\psi = \neg \psi'$, then $H_\psi = H_{\psi'} \cup \{(v_0, u, v_{\psi}), (v_0, \text{self}, v_{\psi}), (v_{\psi}, \text{not}, v_{\psi'})\}$
- if $\psi = \psi_1 \land \psi_2$, then $H_\psi = H_{\psi_1} \cup H_{\psi_2} \cup \{(v_0, u, v_{\psi}), (v_0, \text{self}, v_{\psi}), (v_{\psi_1}, \text{and}, v_{\psi_2}), (v_{\psi_2}, \text{and}, v_{\psi_1})\}$

$S = \{s_0 \mapsto \phi_{s_0}, s_{TV} \mapsto \phi_{s_{TV}}, s_T \mapsto \phi_{s_T}\}$, with the following definitions:

So $\tau_{i}s(v) = \sigma'|_{S_{\leq i-1}} s(v)$, and $\left[\phi_{s}\right]_{v,G,\tau_{i-1}} = \left[\phi_{s}\right]_{v,G,\sigma'|_{S_{\leq i-1}}}$.
– \( \phi_{s^T} = \varnothing_{\text{self}^S} \)
– \( \phi_{sTV} = (\varnothing_{\text{self}^S} \land \square_{\text{and}}^S \land \square_{\text{not}}^S) \land (\varnothing_{\text{self}^S} \land \varnothing_{\text{not}^S} \land (\square_{\text{not}}^S \land (\square_{\text{and}}^S) )) \)
– \( \phi_{s_0} = \varnothing_{\text{eval}^S} \land \square_{\text{not}}^S \)

S is stratified, as shown by the function \( \{s^T \mapsto 1, s_{TV} \mapsto 2, s_0 \mapsto 3 \} \).

We show that \( \psi \) is satisfiable iff there is a \( \sigma \in \Sigma^{2, \text{fix}}_{G^e, S} \) with \( \sigma_{s_0}(v_0) = 1 \). Then because \( S \) is stratified, from Proposition 2, it follows that there must be a \( \sigma' \in \Sigma^{3, \text{fix}}_{G^e, S} \) such that \( \sigma_{s_0}(v_0) = 1 \), i.e. \( \sigma' \) is a (strictly) faithful assignment for \((G, S, s_0(v_0))\).

If \( M \) is a graph and \( v \in V_M \) \( \text{suc}_e^M(v) \) will designate the \( e \)-successors of \( v \) in \( M \), i.e. \( \{v' \in V_M \mid (v, e, v') \in M\} \).

Consider the subset \( S' = \{\phi_{sTV}, \phi_{s^T}\} \) of \( S \), and the subgraph \( G^\beta_\psi \) of \( G_\psi \), defined by \( V_{G^\beta_\psi} = V_{G_\psi} \setminus \{v_0\} \) and \( E_{G^\beta_\psi} \) is the set of triples in \( E_{G_\psi} \) with edge label in \{self, and, not\}. Note that \( V_{G^\beta_\psi} = \{v_0' \mid \psi' \in \Psi \} \).

Let \( X_\psi \) be the set of boolean variables appearing in \( \psi \), and let \( B(X_\psi) \) be the set of boolean valuations over \( X_\psi \), i.e. all (total) functions from \( X_\psi \) to \{0, 1\}. If \( \beta \in B(X_\psi) \), the evaluation of formula \( \psi \) given \( \beta \) will be denoted with \([\psi]^{\beta}\).

Now let \( t \) be the function from \( B(X_\psi) \) to \( \Sigma^{2, \text{fix}}_{G^\beta_\psi, S'} \), defined by \( t(\beta)_{s^T}(v_0) = [\psi]^{\beta} \), and \( t(\beta)_{sTV}(v_0) = 1 \). And let \( t(B(X_\psi)) = \{t(\beta) \mid \beta \in B(X_\psi)\} \). Finally, let \( \Sigma_{TV} = \{\sigma \in \Sigma^{2, \text{fix}}_{G^\beta_\psi, S'} \mid \sigma_{sTV}(v_0) = 1\} \). We will show below that \( t(B(X_\psi)) = \Sigma_{TV} \).

For now, assuming that this claim holds, we show that \( \psi \) is satisfiable iff there is a \( \sigma \in \Sigma^{2, \text{fix}}_{G^e, S} \) such \( \sigma_{s_0}(v_0) = 1 \).

\[ \Rightarrow \] Let \( \psi \) be satisfiable.

Then there is a \( \beta \in B(X_\psi) \) such that \([\psi]^{\beta} = 1\). Define \( \sigma = t(\beta) \cup \{\neg s_0(v) \mid v \in G^\beta_\psi \} \cup \{s_0(v_0), \neg s_{TV}(s_0), \neg s_{T}(s_0)\} \).

Then \( \sigma_{s^T}(v_0) = 1 \).

Therefore because \( \text{suc}^{G_\psi}_{\text{eval}}(v_0) = \{v_0\} \), \([\varnothing_{\text{eval}^S}]^{v_0, G^\beta_\psi, \sigma} = 1\).

Similarly, for all \( v_0' \in V_{G^\beta_\psi} \), \( \sigma_{sTV}(v_0') = 1 \), and \( \text{suc}^{G_\psi}_{\text{eval}}(v_0) = \{V_{G^\beta_\psi}\} \), therefore \([\square_{TV}^\top]^{v_0, G^\beta_\psi, \sigma} = 1\).

So from the definition of \( \phi_{sTV} \), \([\phi_{sTV}]^{v_0, G^\beta_\psi, \sigma} = \sigma_{sTV}(v_0) = 1\).

In addition, because \( \text{suc}^{G_\psi}_{\text{self}}(v_0) = \emptyset \), from the definition of \( \phi_{sself} \), \([\phi_{sself}]^{v_0, G^\beta_\psi, \sigma} = \sigma_{sself}(v_0) = 0\).

Now take any \( v \in V_{G^\beta_\psi} \).

Because \( t(\beta) \preceq \sigma \), \( \sigma_{s^T}(v) = t(\beta)_{s^T}(v) \).

Then because \( \beta \in B(X_\psi) \) and \( t(B(X_\psi)) = \Sigma_{TV} \), \( t(\beta) \in \Sigma^{2, \text{fix}}_{G^\beta_\psi, S'} \).

Therefore \( t(\beta)_{s^T}(v) = [\phi_{s^T}]^{v, G^\beta_\psi, t(\beta)} \).

Then from the definition of \( G^\beta_\psi \), \( \text{suc}^{G_\psi}_{\text{eval}}(v) = \text{suc}^{G_\psi}_{\text{self}}(v) \).

Therefore, from the definition of \( \phi_{s^T} \), \([\phi_{s^T}]^{v, G^\beta_\psi, t(\beta)} = [\phi_{s^T}]^{v, G^\beta_\psi, \sigma} = [\phi_{s^T}]^{v, G^\beta_\psi, \sigma} \) must hold.

This yields \( \sigma_{s^T}(v) = t(\beta)_{s^T}(v) = [\phi_{s^T}]^{v, G^\beta_\psi, t(\beta)} = [\phi_{s^T}]^{v, G^\beta_\psi, \sigma} \), therefore \( \sigma_{s^T}(v) = [\phi_{s^T}]^{v, G^\beta_\psi, \sigma} \).

A similar argument can be used to show \( \sigma_{sTV}(v) = [\phi_{sTV}]^{v, G^\beta_\psi, \sigma} \).
Finally, because \( \text{succ}_{G^\psi}^v(v) = \emptyset \), from the definition of \( \phi_{s_0} \),
\[
\llbracket \phi_{s_0} \rrbracket^{v, G^\psi, \sigma} = \sigma s_0(v_0) = 0.
\]

So for any \( v' \in \{ v_0 \} \cup V_{G^\psi}^R = V_{G^\psi} \), and for any \( s \in S \),
\[
\llbracket \phi_s \rrbracket^{v', G^\psi, \sigma} = \sigma s(v') \quad \text{holds}.
\]
Therefore \( \sigma \in \Sigma_{G^\psi, S}^{\text{fix}} \).

\(-\Leftrightarrow\). Let \( \sigma \in \Sigma_{G^\psi, S}^{\text{fix}} \) such that \( \sigma s_0(v_0) = 1 \).

Take any \( v \in V_{G^\psi}^R \).

Because \( s \in S' \), \( s^\tau(v) \in \text{atoms}(G^R_{\psi}, S') \).
And because \( E_{G^\psi} \subseteq E_{G^R_{\psi}} \) and \( s' \subseteq S \), \( \text{atoms}(G^R_{\psi}, S') \subseteq \text{atoms}(G^\psi, S) \).
Therefore \( \sigma|_{\text{atoms}(G^R_{\psi}, S')} s^\tau(v) = \sigma s^\tau(v) \).

Then because \( \sigma \in \Sigma_{G^\psi, S}^{\text{fix}} \), \( \sigma s^\tau(v) = \llbracket \phi_{s} \rrbracket^{v, G^\psi, \sigma} \).

Finally, because \( \text{succ}_{\text{self}}^G(v) = \text{succ}_{\text{self}}^G(v) \), from the definitions of \( \phi_{s^\tau} \),
\[
\llbracket \phi_{s^\tau} \rrbracket^{v, G^\psi, \sigma} = \llbracket \phi_{s^\tau} \rrbracket^{v, G^R_{\psi}, \sigma|_{\text{atoms}(G^R_{\psi}, S')}} = \llbracket \phi_{s^\tau} \rrbracket^{v, G^R_{\psi}, \sigma|_{\text{atoms}(G^R_{\psi}, S')}}.
\]
This yields \( \sigma|_{\text{atoms}(G^R_{\psi}, S')} s^\tau(v) = \sigma s^\tau(v) = \llbracket \phi_{s} \rrbracket^{v, G^R_{\psi}, \sigma|_{\text{atoms}(G^R_{\psi}, S')}} \).

Therefore \( \sigma|_{\text{atoms}(G^R_{\psi}, S')} s^\tau(v) = \llbracket \phi_{s} \rrbracket^{v, G^R_{\psi}, \sigma|_{\text{atoms}(G^R_{\psi}, S')}} \).

A similar argument can be used to show \( \sigma|_{\text{atoms}(G^R_{\psi}, S')} s^\tau(v) = \llbracket \phi_{s} \rrbracket^{v, G^R_{\psi}, \sigma|_{\text{atoms}(G^R_{\psi}, S')}} \).
So as \( S' = \{ s^\tau, s^\tau \} \), \( \sigma|_{\text{atoms}(G^R_{\psi}, S')} \in \Sigma_{G^R_{\psi}, S'}^{\text{fix}} \) holds.

Now because \( \sigma s_0(v_0) = 1 \) and \( \text{succ}_{\text{self}}^G(v) \), from the definition of \( \phi_{s_0} \),
\( \sigma|_{\text{atoms}(G^R_{\psi}, S')} s^\tau(v) = 1 \) must hold for each \( v \in V_{G^\psi} \).
Therefore \( \sigma|_{\text{atoms}(G^R_{\psi}, S')} \in \Sigma_{G^R_{\psi}, S'}^{\text{fix}} \).

Finally, because \( \sigma s_0(v_0) = 1 \) and \( \text{succ}_{\text{self}}^G(v) \), from the definition of \( \phi_{s_0} \),
\( \sigma s^\tau(v_0) = \sigma|_{\text{atoms}(G^R_{\psi}, S')} s^\tau(v_0) = 1 \) must hold.

Then as \( \sigma|_{\text{atoms}(G^R_{\psi}, S')} \in \Sigma_{G^R_{\psi}, S'}^{\text{fix}} \), because \( t(B(X_\psi)) = \Sigma_{G^R_{\psi}, S'}^{\text{fix}} \), from the definition of \( t \),
there must be a \( \beta \in B(X_\psi) \) such that \( \llbracket \beta \rrbracket^\psi = 1 \).
Therefore \( \psi \) is satisfiable.

To complete the proof, we need to show that \( t(B(X_\psi)) = \Sigma_{G^R_{\psi}, S'}^{\text{fix}} \):

\(-\Rightarrow\).

Let \( \sigma \in t(B(X_\psi)) \).

From the definition of \( f \), for all \( v \in V_{G^\psi} \), \( \sigma s^\tau(v) = 1 \) holds.
So from the definition of \( \Sigma_{G^R_{\psi}, S'}^{\text{fix}} \), we only need to show that \( \sigma \in \Sigma_{G^R_{\psi}, S'}^{\text{fix}} \)
for \( S' = \{ s^\tau, s^\tau \} \), i.e. that \( \sigma s(v) = \llbracket \phi_{s} \rrbracket^{v, G^R_{\psi}, \sigma} \) for each \( s(v) \in \text{atoms}(G^R_{\psi}, S') \).

If \( \sigma s^\tau(v) = 0 \), then because \( \text{succ}_{\text{self}}^G(v) \), \( \llbracket \phi_{\text{self}} \rrbracket^{v, G^R_{\psi}, \sigma} = 0 \).
So from the definition of \( \phi_{s^\tau} \), \( \sigma s^\tau(v) = \llbracket \phi_{s^\tau} \rrbracket^{v, G^R_{\psi}, \sigma} = 0 \).
Semantics and Validation of Recursive SHACL

Similarly, if $\sigma_s(\psi) = 1$, then $\llbracket \phi_s \rrbracket^{v_G^R_s, \sigma} = 1$.

So we only need to show that $\sigma_{TV}(\psi) = \llbracket \phi_{TV} \rrbracket^{v_G^R_s, \sigma}$.
And because $\sigma \in t(B(X_\psi))$, from the definition of $t$, $\sigma_{TV}(v) = 1$.
Therefore it is sufficient to show that $\llbracket \phi_{TV} \rrbracket^{v_G^R_s, \sigma} = 1$.

By induction on $\psi$:

- $\psi = x$
  Then $G^R_\psi = \{\psi, \text{self}, \psi\}$, with $V_{G_\psi} = \{\psi\}$.

  Let $\sigma_s((v_\psi) = 0$.
  And let $\phi_1$ be the right disjunct of $\phi_{TV}$, i.e. $\phi_1 = \hat{\psi} \land \neg s_T$.
  Because $\sigma_s(\psi) = 0$, $\llbracket \psi \rrbracket^{v_G^R_s, \sigma} = 0$.
  So $\llbracket \phi_1 \rrbracket^{v_G^R_s, \sigma} = 1$.
  Therefore $\llbracket \phi_{TV} \rrbracket^{v_G^R_s, \sigma} = 1$.

Now let $\sigma_s((v_\psi) = 0$.
And let $\phi_2$ be the left disjunct of $\phi_{TV}$, i.e. $\phi_2 = \hat{\psi} \land s_T$.
Because $\sigma_s(\psi) = 0$, $\llbracket \psi \rrbracket^{v_G^R_s, \sigma} = 0$.
In addition, because $\text{suc}_{\text{and}} = 0$,
$\llbracket \psi \rrbracket^{v_G^R_s, \sigma} = 1$.
So $\llbracket \phi_2 \rrbracket^{v_G^R_s, \sigma} = 1$.
Therefore $\llbracket \phi_{TV} \rrbracket^{v_G^R_s, \sigma} = 1$.

- $\psi = \text{NOT} \psi$

  Let $\sigma_s((v_\psi) = 0$.
  Because $\sigma \in t(B(X_\psi))$, $\sigma = t(\beta)$ for some $\beta \in B(X_\psi)$.
  And because $\sigma_s((v_\psi) = 0$, from the definition of $t$, $\llbracket \psi \rrbracket^{\beta} = 0$.
  Then as $\psi = \text{NOT} \psi$, $\llbracket \psi \rrbracket^{\beta} = 1$ must hold,
  and from the definition of $t$, $\sigma_s((v_\psi) = 1$.
  Finally, because $\text{suc}_{\text{and}}((v_\psi) = 0$ and $\text{suc}_{\text{not}}((v_\psi) = \{\psi^{'\prime}\}$, from the definition of $\phi_{TV}$,
$\llbracket \phi_{TV} \rrbracket^{v_G^R_s, \sigma} = 1$.

A similar argument can be used for the case $\sigma_s((v_\psi) = 1$.

- $\psi = \psi_1 \text{AND} \psi_2$

  Let $\sigma_s((v_\psi) = 0$.
  Because $\sigma \in t(B(X_\psi))$, $\sigma = t(\beta)$ for some $\beta \in B(X_\psi)$.
And because $\sigma s(v_2) = 0$, from the definition of $t$, $\|\psi\|^\beta = 0$.
Then as $\psi = \psi_1$ AND $\psi_2$, either $[\psi_1]^\beta = 0$ or $[\psi_2]^\beta = 0$ must hold,
and from the definition of $t$, if $[\psi_1]^\beta = 0$, then $\sigma s(v_1) = 0$.
Finally, because $\text{suc}^{G^R}_{\psi}(v_\psi) = \{v_{\psi_1}, v_{\psi_2}\}$, from the definition of $\phi_{TV}$,
$[\phi_{TV}]_{v_\psi} G^R_{\psi}, \sigma = 1$.

Now let $\sigma s(v_\psi) = 1$.
Because $\sigma \in \ell(B(X_\psi))$, $\sigma = \ell(\beta)$ for some $\beta \in B(X_\psi)$.
And because $\sigma s(v_\psi) = 1$, from the definition of $t$, $[\psi]^\beta = 1$.
Then as $\psi = \psi_1$ AND $\psi_2$, both $[\psi_1]^\beta = 1$ and $[\psi_2]^\beta = 1$ must hold,
and from the definition of $t$, $\sigma s(v_\psi) = 1$.
Finally, because $\text{suc}^{G^R}_{\psi}(v_\psi) = \{v_{\psi_1}, v_{\psi_2}\}$, from the definition of $\phi_{TV}$,
$[\phi_{TV}]_{v_\psi} G^R_{\psi}, \sigma = 1$.

--≤.
Let $\sigma \in \Sigma_{TV}$.
We need show that there is a $\beta \in B(X_\psi)$ such that $\sigma = \ell(\beta)$.
From the definition of $\Sigma_{TV}$, we already know that $s_{TV}(v_{\psi'}) = 1$ for all $v_{\psi'} \in V_{G^R}$.
So from the definition of $t$, it is sufficient to show that there is a $\beta \in B(X_\psi)$ such that
$\sigma s(v_{\psi'}) = [\psi']^\beta$ for each $v_{\psi'} \in V_{G^R}$.
By induction on $\psi$:

- $\psi \equiv x$.
  If $\sigma s(v_x) = 0$, set $\beta = \{x \mapsto 0\}$.
  If $\sigma s(v_x) = 1$, set $\beta = \{x \mapsto 1\}$.

- $\psi = \text{NOT} \psi'$.

  Let $\sigma' = \sigma|_{\text{atoms}(G^R_{\psi'}, S')}$.

  From the definition of $G^R_{\psi'}$, for any $v \in V_{G^R_{\psi}}$ and $e \in E_{G^R_{\psi}}$,
  $\text{suc}^{G^R_{\psi}}(v) = \text{suc}^{G^R_{\psi'}}(v)$.
  So for any $\phi$ and $v \in G^R_{\psi'}$, $[\phi]^{v, G^R_{\psi'}, \sigma} = [\phi]^{v, G^R_{\psi'}, \sigma'}$.
  In addition, $\sigma' \preceq \sigma$, and because $\sigma \in \Sigma_{TV}$, $\sigma \in \Sigma_{G^R_{\psi'}, S'}$.
  Therefore $\sigma' \in \Sigma_{G^R_{\psi'}, S'}$.
  So by IH, $\sigma' = \ell(\beta)$ for some $\beta \in B(X_{\psi'})$.
  Therefore for each $v_{\psi_m} \in G^R_{\psi'}$, $\sigma' s(v_{\psi_m}) = [\psi_m]^\beta$.
  So as $\sigma' \preceq \sigma$, for each $v_{\psi_m} \in G^R_{\psi'}$, $\sigma s(v_{\psi_m}) = [\psi_m]^\beta$ holds.
  Therefore as $V_{G^R_{\psi}} \setminus V_{G^R_{\psi'}} = \{v_\psi\}$, we only need to show that $\sigma s(v_\psi) = [\psi]^\beta$.

If $[\psi]^\beta = 0$, then $[\psi']^\beta = 1$.
So as $\sigma' \preceq \sigma$ and $\sigma' = \ell(\beta)$, $\sigma s(v_{\psi'}) = 1$ must hold.
So from the definition of $G^R_{\psi}$, $[\text{not}s]^{\psi, G^R_{\psi}, \sigma} = 0$,
and $[\text{not}s]^{\psi, G^R_{\psi}, \sigma} = 1$. 
In addition, because $\sigma \in \SigmaTV$, $[\phiTV]_{v^G,R,\sigma}^{\SigmaTV} = 1$.
So from the definition of $\phiTV$, $[\text{self} \leftarrow sT]_{v^G,R,\sigma}^{\SigmaTV} = 1$ must hold.
And because $\text{suc}_e(G^R_{\psi}) = \{v_0\}$, $[sT]_{v^G,R,\sigma}^{\SigmaTV} = 0$ must hold.
Finally, because $\sigma \in \SigmaTV$, $[sT]_{v^G,R,\sigma}^{\SigmaTV} = \sigma sT(v_{\psi})$.
Therefore $\sigma sT(v_{\psi}) = 0$.
A similar argument can be used for the case $[\psi^\beta] = 1$.

- $\psi = \psi_1 \text{AND } \psi_2$

Let $\sigma_1 = \sigma|_{\text{atoms}(G^R_{\psi_1}, S)}$, and $\sigma_2 = \sigma|_{\text{atoms}(G^R_{\psi_2}, S)}$.

From the definition of $G^R_{\psi}$, for $i \in \{1, 2\}$ and for any $v \in G^R_{\psi_i}$ and $e \in E_{G^R_{\psi_i}}$,
$$\text{suc}_e^G(v) = \text{suc}_e^{\psi_i}(v).$$

So for any $\phi$ and $v \in G^R_{\psi}$, $[\phi]_{v^G,R,\sigma}^{\SigmaTV} = [\phi]_{v^G_{\psi_i},\sigma_i}$.

In addition, $\sigma \preceq \sigma$, and because $\sigma \in \SigmaTV$, $\sigma \in \Sigma^{\text{self}_{\psi_i}, S}$.

Therefore $\sigma_i \in \Sigma^{\text{self}_{\psi_i}, S_i}$.

So by IH, $\sigma_1 = t(\beta_1)$ for some $\beta_1 \in B(X_{\psi_1})$, and $\sigma_2 = t(\beta_2)$ for some $\beta_2 \in B(X_{\psi_2})$.

And from the definition of $t$, for all $v_x \in G^R_{\psi_i}$ with $x$ a propositional variable,
$$\sigma_1 sT(v_x) = \beta_1(x).$$

In addition, because $\sigma_1 \cup \sigma_2 \preceq \sigma$, if $v_x \in V_{G^R_{\psi_1}} \cap V_{G^R_{\psi_2}}$, then
$$\sigma_1 sT(v_x) = \sigma_2 sT(v_x) = \sigma sT(v_x).$$

Therefore if $x \in \text{dom}(\beta_1) \cap \text{dom}(\beta_2)$, then $\beta_1(x) = \beta_2(x)$, i.e. $\beta_1(x)$ and $\beta_2(x)$ are compatible.

Now define $\beta = \beta_1 \cup \beta_2$.

Because $\sigma_1 = t(\beta_i)$ for $i \in \{1, 2\}$, for each $v_{\psi_i} \in G^R_{\psi_i}$,
$$\sigma_1 sT(v_{\psi_i}) = [v_{\psi_i}]^{\beta_1} = [v_{\psi_i}]^{\beta_2}.$$

So as $\sigma_1 \cup \sigma_2 \preceq \sigma$, for each $v_{\psi_i} \in G^R_{\psi_1} \cup G^R_{\psi_2}$, $\sigma sT(v_{\psi_i}) = [v_{\psi_i}]^{\beta}$ holds.

Therefore as $V_{G^R_{\psi}} \setminus V_{G^R_{\psi_1} \cup G^R_{\psi_2}} = \{v_{\psi_i}\}$, we only need to show that
$$\sigma sT(v_{\psi_i}) = [v_{\psi_i}]^{\beta}.$$

If $[\psi]^{\beta} = 0$, then $[v_{\psi_i}]^{\beta} = 0$ for some $i \in \{1, 2\}$.
So as $\sigma \preceq \sigma$ and $\sigma = t(\beta_i)$ for some $\beta_i \in B(X_{\psi_i})$, $\sigma sT(v_{\psi_i}) = 0$ must hold.

So from the definition of $G^R_{\psi}$, $[\Box_{\text{and}} sT]_{v^G,R,\sigma}^{\SigmaTV} = 0$, and $[\Box_{\text{and}} sT]_{v^G,R,\sigma}^{\SigmaTV} = 1$.

In addition, because $\sigma \in \SigmaTV$, $[\phiTV]_{v^G,R,\sigma}^{\SigmaTV} = 1$.

So from the definition of $\phiTV$, $[\text{self} \leftarrow sT]_{v^G,R,\sigma}^{\SigmaTV} = 1$ must hold.

And because $\text{suc}_e(G^R_{\psi}) = \{v_0\}$, $[sT]_{v^G,R,\sigma}^{\SigmaTV} = 0$ must hold.

Finally, because $\sigma \in \SigmaTV$, $[sT]_{v^G,R,\sigma}^{\SigmaTV} = \sigma sT(v_{\psi_i})$.

Therefore $\sigma sT(v_{\psi_i}) = 0$.
A similar argument can be used for the case $[\psi]^{\beta} = 1$. 
Proposition 5 (Constraint – Lower Bound). For stratified \( \mathcal{L}_{\wedge,\neg,\top} \), VALIDATION is NP-hard in constraint complexity.

Proof. Reduction from CIRCUIT-SAT (with OR, AND and NOT).

Let \( \psi \) be a boolean formula with variables \( \{x_1, \ldots, x_n\} \). We build an instance \( \langle G, S, s_0(v_0) \rangle \) of VALIDATION as follows.

The input graph \( G \) is defined by \( V_G = \{v_0, v_1\} \) and \( E_G = \{(v_0, p, v_1), (v_1, p, v_1)\} \).

For each \( x_i \), we use two shapes \( s^+_i \) and \( s^-_i \), defined by \( \phi^+_i = \wedge p \neg s^-_i \), and \( \phi^-_i = \wedge p s^-_i \).

Then we use an additional shape \( s_0 \), with unique target \( v_0 \), and defined by \( \phi_0 = \wedge \top \wedge \neg \top \wedge \neg \top \wedge \neg \top \wedge \neg \top \wedge \neg \top \). Let \( s’ \) be a boolean formula with variables \( \{x_1, \ldots, x_n\} \). Then from Theorem 2, for a given pair \( \langle G, S, s_0(v_0) \rangle \), there must be a stratified, from Proposition 2, there must be a matching above:

- if \( x_i \) is a non-negated variable, then \( tr(x_i) = s^+_i \)
- if \( x_i \) is a negated variable, then \( tr(x_i) = s^-_i \)
- \( tr(\psi_1 \wedge \psi_2) = tr(\psi_1) \wedge tr(\psi_2) \)
- \( tr(\psi_1 \wedge \psi_2) = \neg (tr(\psi_1) \wedge \neg tr(\psi_2)) \)

Now let \( S = \{s_0 \mapsto \phi_0\} \cup \bigcup_{1 \leq i \leq j} \{s^+_i \mapsto \phi^+_i, s^-_i \mapsto \phi^-_i\} \). Then by induction on \( NNF(\psi) \), it can be easily checked that \( NNF(\psi) \) is satisfiable if there is a \( \sigma \in \Sigma_{G,S}^{2,fix} \) such that \( \sigma s_0(v_0) = 1 \).

And because \( S \) is stratified, from Proposition 2, there must be a \( \sigma’ \in \Sigma_{G,S}^{3,fix} \) such that \( \sigma s_0(v_0) = 1 \), i.e. \( \sigma’ \) is a (strictly) faithful assignment for \( \langle G, S, s_0(v_0) \rangle \).

\( \square \)

Theorem 2 (Kostylev et al., 2015, [1]).

Let \( r \) a SPARQL property path expression, \( G \) a graph, and \( v_1, v_2 \) two nodes in \( G \). Then it can be decided in time polynomial in \(|G|\) whether there is a path from \( v_1 \) to \( v_2 \) in \( G \) matching \( r \).

Lemma 5. Let \( \langle G, S, s_0(v_0) \rangle \) be an instance of VALIDATION. Then it can be transformed in polynomial time into an instance \( \langle G’, S’, s_0(v_0) \rangle \) such that \( \langle G, S, s_0(v_0) \rangle \) is valid iff \( \langle G’, S’, s_0(v_0) \rangle \) is valid, and \( S’ \) contains no property path expression.

Proof. If \( r \) is a property path expression and \( G \) a graph, let \( [r]^G \subseteq (V_G)^2 \) designate all pairs \( (v_1, v_2) \) of nodes in \( G \) such that there is a path in \( G \) from \( v_1 \) to \( v_2 \) matching \( r \)

Then from Theorem 2, for a given pair \( (v_1, v_2) \in (V_G)^2 \), it can be decided in time polynomial in \(|G|\) whether \( (v_1, v_2) \in [r]^G \) (see [1]).

Let \( \langle G, S, s_0(v_0) \rangle \) be an instance of VALIDATION, let \( R \) be the set of property path expressions appearing in \( S \), and for each \( r \in R \), let \( p_r \) be a fresh edge label.

\( G’ \) is defined as \( V_{G’} = V_G \), and \( E_{G’} = E_G \cup \{(v_1, p_r, v_2) \mid r \in R \text{ and } (v_1, v_2) \in [r]^G\} \).

So \( G’ \) can be computed by deciding for each \( r \in R \) and \( (v_1, v_2) \in (V_G)^2 \) whether the \( (v_1, v_2) \in [r]^G \).

From the above observation, each decision is polynomial in \(|G|\).

And because \(|V_G|^2 = O(|G|^2)\) and \(|R| = O(|S|)\), the number of such decisions is
Now let $f_R$ be the function which takes an $\mathcal{L}$ formula $\phi$, and returns $\phi$ where each occurrence of $r$ is replaced by $p_r$, for each $r \in R$.

And let $S'$ be the function defined by $\text{dom}(S') = \text{dom}(S)$, and for each $s \in \text{dom}(S)$, $S'(s) = f_R(S(s))$.

Then for each $(v_1, v_2) \in (V_G)^2 = (V_{G'})^2$, $[\phi]^G = [\phi]^G_{G'}$.

It follows that for any $v \in V_G = V_{G'}$, and for constraint formula $\phi$, immediately by induction on $\phi$, $[\phi(v)]^{G,S} = [\phi(v)]^{G',S'}$.

In particular, for any shape $s$ defined in $S$ (and therefore also in $S'$), $[\phi_s(v)]^{G,S} = [\phi_s(v)]^{G',S'}$.

Therefore $\langle G, S, s_0(v_0) \rangle$ is valid iff $\langle G', S', s_0(v_0) \rangle$. \hfill \Box

**Lemma 6.** Let $\langle G, S, s_0(v_0) \rangle$ be an instance of VALIDATION, such that no constraints in range($S$) contains a property path expression.

Then there is a function $S_n$ such that $\langle G, S, s_0(v_0) \rangle$ is valid iff $\langle G, S_n, s_0(v_0) \rangle$ is valid, each constraint in range($S_n$) contains at most one operator and no property path expression, and $S_n$ can be computed in time polynomial in $|S|$.

**Proof.** Intuitively, $S$ can be transformed into $S_n$ by introducing a fresh shape name for each subformula of each constraint in range($S$).

More formally, if $\phi \in \mathcal{L}$, let $\text{opN}(\phi)$ be the number of operators (either $\lnot$, $\land$ or $\lor$, or $\leq q,r$) present in $\phi$.

We define the function $\text{norm}(\cdot)$, which takes a function $S = \{ s_0 \mapsto \phi_{s_0}^S, \ldots, s_m \mapsto \phi_{s_m}^S \}$ from shape names to formulas in $\mathcal{L}$, and returns another:

- If $\text{opN}(\phi) \leq 1$ for each $\phi \in \text{range}(S)$, then $\text{norm}(S) = S$.
- Otherwise, let $s_i$ be some $s_i \in \text{dom}(S)$ such that $\text{opN}(\phi_{s_i}^S) > 1$. And let $N_S$ be a set of shape names disjoint from $\text{dom}(S)$.

Then $\text{norm}(S)$ is defined as $\text{norm}(S) = S|_{\text{dom}(S) \setminus \{s_i\}} \cup \text{fold}(s_i, \phi_{s_i}^S, N_S)$, where $\text{fold}(s_i, \phi_{s_i}^S, N_S)$ is defined as follows:

- If $\phi_{s_i}^S = \phi_1 \land \phi_2$, then $\text{fold}(s_i, \phi_{s_i}^S, N_S) = \{ s_i \mapsto s'_1 \land s'_2 \}$, with $s'_1, s'_2 \in N_S$.
- If $\phi_{s_i}^S = \neg \phi$, then $\text{fold}(s_i, \phi_{s_i}^S, N_S) = \{ s_i \mapsto \neg s' \}$ with $s' \in N_S$.
- If $\phi_{s_i}^S = \geq n.r \phi$, then $\text{fold}(s_i, \phi_{s_i}^S, N_S) = \{ s_i \mapsto \geq n.r s' \}$ with $s' \in N_S$.

For any $s \in \text{dom}(S)$, let $\phi_{s_{\text{norm}(S)}}$ designate $\text{(norm}(S))(s)$.

Then immediately from the definition of $[\cdot]$, for any graph $G$ and $v \in V_G$.

$[\phi_{s}(v)]^{G,S} = [\phi_{s_{\text{norm}(S)}}(v)]^{G,\text{norm}(S)}$.

In addition, if no constraint in range($S$) contains a property path expression, then no constraint in range($\text{norm}(S)$) contains a property path expression.

Now let $S = S_0, S_1, \ldots$ be the sequence defined by $S_0 = S$, and $S_{k+1} = \text{norm}(S_k)$.

And let $s_k \in \text{dom}(S_k)$ be the shape name selected when computing $\text{norm}(S_k)$, i.e. such that $S_{k+1} = S_k|_{\text{dom}(s_k) \setminus \{s_k\}} \cup \text{fold}(s_k, \phi_{s_k}^{S_k}, N_{S_k})$.

From the above definition, $\text{opN}(\phi_{s_k}^{S_{k+1}}) = 1$, and for each fresh $s' \in \text{dom}(S_{k+1}) \setminus \text{dom}(S_k)$, $\text{opN}(\phi_{s_k}^{S_{k+1}}) = \text{opN}(\phi_{s_k}^{S_k}) - 1$.

It follows that $S$ must reach a fixed point $S_n$ of $\text{norm}(\cdot)$, with $\text{opN}(S_n(s)) \leq 1$ for each
Each application of $\text{norm}()$ can introduce at most 2 fresh shape names ($s'_1$ and $s'_2$), therefore $n$ is bounded by $2 \times \sum_{s_i \in \text{dom}(S)} \text{opN}(S(s_i)) = O(|S|)$.

It follows also that the size $|S_n|$ of $S_n$ is bounded by $|S| + 2 \times \sum_{s_i \in \text{dom}(S)} \text{opN}(S(s_i)) = O(|S|)$.

And for $0 \leq k \leq n$, because $|S_k| \leq |S_{k+1}|$, $|S_k| \leq |S_n| = O(|S|)$ must hold. Finally, for each $0 \leq k < n$, from the definition of $\text{norm}()$, $\text{norm}(S_{k+1})$ can be computed in time linear in $|S_k| = O(|S|)$.

Therefore $S_n$ can be computed in $O(|S|) \times O(|S|) = O(|S|^2)$.

Now let $(G, S, s_0(v_0))$ be an instance of VALIDATION, such that no constraints in $\text{range}(S)$ contains a property path expression. From the above observations, $S_n$ can be computed in time polynomial in $|S|$, and each constraint in $\text{range}(S_n)$ has at most one operator. By induction on $n$, no constraint in $\text{range}(S_n)$ contains a property path expression.

In addition, for any $v \in V_G$ and $s \in \text{dom}(S)$, by induction on $n$ still, $\llbracket \phi^S(v) \rrbracket^{G,S} = \llbracket \phi^S_{s_0}(v) \rrbracket^{G,S}$.

In particular $\llbracket \phi^S_{s_0}(v) \rrbracket^{G,S} = \llbracket \phi^S_{s_0}(v) \rrbracket^{G,S}$. It follows that $(G, S, s_0(v_0))$ is valid iff $(G, S_n, s_0(v_0))$ is valid.

\begin{lemma}
Let $(G, S, s_0(v_0))$ be a valid instance of VALIDATION such that each constraint in $S$ contains at most one operator and no property path expression, and let $\sigma \in \Sigma_{G,S,s_0(v_0)}^\str$. Then it can be verified in time polynomial in $|\sigma| + |G| + |S|$ that $\sigma \in \Sigma_{G,S,s_0(v_0)}^\str$ even with a binary encoding of cardinality constraints.
\end{lemma}

\begin{proof}
Let $(G, S, s_0(v_0))$ be a valid instance of VALIDATION such that each constraint in $S$ contains at most one operator and no property path expression. And let $\sigma \in \Sigma_{G,S,s_0(v_0)}^\str$.

For each $s \in \text{dom}(S)$, $\phi_s$ will designate $S(s)$.

In order to verify that $\sigma \in \Sigma_{G,S,s_0(v_0)}^\str$, it is sufficient to verify that $\sigma s(v) = \llbracket \phi_s(v) \rrbracket^{G,S}$ for each $s(v) \in \text{atoms}(G, S)$.

We show below that $\sigma s(v) = \llbracket \phi_s(v) \rrbracket^{G,S}$ can be verified in time polynomial in $|\sigma| + |G| + |S|$. Because $|\text{atoms}(G, S)| = |G| \times |S|$ and $|\sigma| < |S|$, it follows that $\sigma \in \Sigma_{G,S,s_0(v_0)}^\str$ can be verified in time in time polynomial in $|\sigma| + |G| + |S|$.

The procedure to verify $\sigma s(v) = \llbracket \phi_s(v) \rrbracket^{G,S}$ is the following:

- If $\phi_s \equiv \top$, then verify whether $\sigma s(v) = 1$, in $O(|\phi_s| + |\sigma|)$
- If $\phi_s \equiv \neg \top$, then verify whether $\sigma s(v) = 0$, in $O(|\phi_s| + |\sigma|)$
- If $\phi_s \equiv \phi'_s$, then verify whether $\sigma s(v) = \sigma s'(v)$, in $O(|\phi_s| + |\sigma|)$
- If $\phi_s \equiv \neg \phi'_s$, then verify whether $\sigma s(v) = \sigma s'(v)$, in $O(|\phi_s| + |\sigma|)$
- If $\phi_s \equiv \phi_1 \land \phi_2$, then verify whether $\sigma s(v) = \min\{\sigma s_1(v), \sigma s_2(v)\}$, in $O(|\phi_s| + |\sigma|)$
- If $\phi_s \equiv (p_1 = p_2)$, then check whether there is a $v_2 \in V_G$ such that either $(v, p_1, v_2) \in E_G$ and $(v, p_2, v_2) \not\in E_G$, or $(v, p_2, v_2) \in E_G$ and $(v, p_1, v_2) \not\in E_G$. If this is
the case, then verify whether \( \sigma_s(v) = 0 \), otherwise verify whether \( \sigma_s(v) = 1 \). The whole procedure is in \( O(\left| \phi_s \right| + |G|^2 + |\sigma|) \).

- If \( \phi_s = \neg n.p \), then let \( V' = \{ v' \in V_G \mid (v, p, v') \in E_G \} \). \( V' \) can be computed in \( O(|G|) \).

Then decide if \( |V'| < n \), in \( O(|G| + |\phi_s|) \).

If this is the case, verify whether \( \sigma_s(v) = 0 \), in \( O(|\phi_s| + |\sigma|) \).

Otherwise, count the number \( q \) of \( v' \in X \) such that \( \sigma_s(v') = 1 \), in \( O(|G| \times |\sigma|) \).

If \( q \geq n \), then verify whether \( \sigma_s(v) = 1 \), in \( O(|\phi_s| + |\sigma|) \).

Otherwise, verify whether \( \sigma_s(v) = 0 \), in \( O(|\phi_s| + |\sigma|) \).

- If \( \phi_s = n.p \), the procedure is identical to the previous one, but without the need to check \( \sigma_s(v') = 1 \).

\[ \square \]

**Proposition 6 (Combined – Upper bound).** **VALIDATION** is in \( NP \) for \( L \)

**Proof.** Let \( (G_0, S_0, s_0(v_0)) \) be an instance of VALIDATION

From Lemma 5, it can be transformed in polynomial time into an instance \( (G_1, S_1, s_0(v_0)) \) such that \( (G_0, S_0, s_0(v_0)) \) is valid if \( (G_1, S_1, s_0(v_0)) \) is valid, and constraints in \( S_1 \) contain no property path expression.

Then from Lemma 6, \( (G_1, S_1, s_0(v_0)) \) can be transformed in polynomial time into an instance \( (G, S, s_0(v_0)) \) such that \( (G_1, S_1, s_0(v_0)) \) is valid if \( (G, S, s_0(v_0)) \) is valid, and each constraint in \( S \) contains at most one operator and no property path expression.

In order to verify that \( (G_0, S_0, s_0(v_0)) \) is valid, it is sufficient to verify that \( (G, S, s_0(v_0)) \) is valid.

Let us assume that \( (G, S, s_0(v_0)) \) is valid.

Then \( \mathcal{X}_{G,S,s_0(v_0)} \neq \emptyset \).

Let \( \sigma \in \mathcal{X}_{G,S,s_0(v_0)} \).

Then \( \sigma \) is a function from \( \text{atoms}(G, S) \) to \( \{0, 0.5, 1\} \).

And because \( |\text{atoms}(G, S)| = |V_G| \times |S| \leq |G| \times |S| \), \( \sigma \) can be encoded as a string whose size is polynomial in the encoding of \( (G, S, s_0(v_0)) \).

Now let us assume an oracle which, given a valid instance \( (G, S, s_0(v_0)) \) of VALIDATION, returns some \( \sigma \in \mathcal{X}_{G,S,s_0(v_0)} \).

From Lemma 7, it can be verified in time polynomial in \( |\sigma| + |G| + |S| \) that \( \sigma \in \mathcal{X}_{G,S,s_0(v_0)} \), even with a binary encoding of cardinality constraints.

It follows that VALIDATION is in \( NP \). \( \square \)

**Proposition 7 (Combined – Lower bound).** VALIDATION is PTIME-hard for \( L_{n,\land,\lor} \)

**Proof.** Reduction from the problem of evaluating a monotone boolean circuit. The input for this problem is a set \( C = \{ C_1, \ldots, C_n \} \) of gates, partitioned as input gates, AND gates, OR gates, and where \( C_n \) is denoted the output gate. The problem is to decide whether the circuit evaluates to 1, or in other words, whether the values of the input gates can be propagated in a valid way so that \( C_n \) is assigned value 1.

Let \( I_1, \ldots, I_n, I_{true} \) be a set of different URIs. We construct a graph \( G \) in the following way. First, for each gate \( C_i \), we create a triple \( (C_i, \text{name}, I_i) \). Then for each AND gate
Ci, let B1, . . . , Bℓ be the inputs of Ci. We create a triple (Ci, and, Bj) for 1 ≤ j ≤ ℓ. Similarly, for each OR gate Ci, if B1, . . . , Bℓ are the inputs of Ci, we create a triple (Ci, or, Bj) for 1 ≤ j ≤ ℓ. Finally, for each input gate Ci whose value is 1, we create a triple (Ci, value, Itrue).

Our reduction uses one shape si for each gate Ci, whose intention is to verify, if Ci is assigned the value 1, that this value is a valid propagation of its inputs.

Shape si is defined depending on the gate Ci:

- If Ci is an AND gate with k inputs, then φsi = (≥ k : and (s1 ∨ · · · ∨ sn)) ∧ ≥ 1.name Ii
- If Ci is an OR gate, then φsi = (≥ 1 : or (s1 ∨ · · · ∨ sn)) ∧ ≥ 1.name Ii
- If Ci is an input gate, then φsi = (≥ 1 : name Ii) ∧ ≥ 1.value Ione

Let S = {s1⇒φs1, . . . , sn⇒φsn}. Then it is straightforward to check that the circuit evaluates to 1 iff there is a faithful assignment for ⟨G, S, si(Ci)⟩. And clearly, this reduction can be constructed in LOGSPACE, assuming any reasonable encoding of the monotone circuit value problem.

Bibliography